Red Hat OpenStack Platform 16.1

Advanced Overcloud Customization

Methods for configuring advanced features using Red Hat OpenStack Platform director

Last Updated: 2023-11-17

Red Hat OpenStack Platform 16.1 Advanced Overcloud Customization

Methods for configuring advanced features using Red Hat OpenStack Platform director

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Configure certain advanced features for a Red Hat OpenStack Platform (RHOSP) enterprise environment with Red Hat OpenStack Platform director. This includes features such as network isolation, storage configuration, SSL communication, and general configuration methods.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. INTRODUCTION TO OVERCLOUD CONFIGURATION

Red Hat OpenStack Platform (RHOSP) director provides a set of tools that you can use 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 a basic overcloud. However, a production-level overcloud might require additional configuration:

- 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.
- Red Hat Content Delivery Network node registration, or registration with your internal Red Hat Satellite 5 or 6 server.
- Various system-level options.
- Various OpenStack service options.

NOTE

The examples in this guide are optional steps to configure the overcloud. These steps are necessary only if you want to provide the overcloud with additional functionality. Use the steps that apply to the requirements 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 Red Hat OpenStack Platform director.

2.1. HEAT TEMPLATES

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

A heat template has three main sections:

parameters

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

resources

Use the **resources** section to define the resources, such as compute instances, networks, and storage volumes, that you can create when you deploy a stack using this template. Red Hat OpenStack Platform (RHOSP) contains a set of core resources that span across all components. These are the specific objects to create and configure as part of a stack. RHOSP contains a set of core resources that span across all components. These are defined in the **resources** section of a template.

outputs

Use the **outputs** section to declare the output parameters that your cloud users can access after the stack is created. Your cloud users can use these parameters to request details about the stack, such as the IP addresses of deployed instances, or URLs of web applications deployed as part of the stack.

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 that the cloud user specifies. 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 that you define with your template. You can view the stack hierarchy with the following command:

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

2.2. ENVIRONMENT FILES

An environment file is a special type of template that you can use to customize your heat templates. You can include environment files in the deployment command, in addition to the core heat templates. An environment file contains three main sections:

resource_registry

This section defines custom resource names, linked to other heat templates. This provides a method to create custom resources that do not exist within the core resource collection.

parameters

These are common settings that you apply to the parameters of the top-level template. For example, if you have a template that deploys nested stacks, such as resource registry mappings, the parameters apply only to the top-level template and not to templates for the nested resources.

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.

IMPORTANT

Use **parameter_defaults** instead of **parameters** when you create custom environment files for your overcloud, so that your parameters apply to all stack templates for the overcloud.

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
```

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** file 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.

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

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

NOTE

For RHOSP to use the heat template file as a custom template resource, the file extension must be either .yaml or .template.

2.3. CORE OVERCLOUD HEAT TEMPLATES

Director contains a core heat template collection and environment file collection for the overcloud. This collection is stored in **/usr/share/openstack-tripleo-heat-templates**.

The main files and directories in this template collection are:

overcloud.j2.yaml

This is the main template file that director uses 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 that director uses to create the overcloud environment. It provides a set of configurations for Puppet modules stored on the overcloud image. After director writes the overcloud image to each node, heat starts the Puppet configuration for each node by 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

This file contains the definitions of the roles in an overcloud and maps services to each role.

network_data.yaml

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

plan-environment.yaml

This file contains the definitions of 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

This file contains a mapping of environment files for an overcloud plan.

deployment

This directory contains heat templates. The **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.

environments

This directory contains additional heat environment files that you can use for your overcloud creation. These environment files enable extra functions for your resulting Red Hat OpenStack Platform (RHOSP) environment. For example, the directory contains an environment file to enable Cinder NetApp backend storage (**cinder-netapp-config.yaml**).

network

This directory contains a set of heat templates that you can use to create isolated networks and ports.

puppet

This directory contains templates that control Puppet configuration. The **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

This directory contains legacy heat templates for all service configuration. The templates in the **deployment** directory replace most of the templates in the **puppet/services** directory.

extraconfig

This directory contains templates that you can use to enable extra functionality.

firstboot

This directory contains example **first_boot** scripts that director uses when initially creating the nodes.

2.4. PLAN ENVIRONMENT METADATA

You can define metadata for your overcloud plan in a plan environment metadata file. Director applies metadata during the overcloud creation, and when importing and exporting your overcloud plan.

Use plan environment files to define workflows which director can execute with the OpenStack Workflow (Mistral) service. 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) that you want to use to store the plan files.

template

The core parent template that you want 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 that you want to use. Specify the name and relative locations of each environment file with the **path** sub-parameter.

parameter_defaults

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

workflow_parameters

Use this parameter 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 snippet 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/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 with the **-p** option:

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

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

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

2.5. INCLUDING ENVIRONMENT FILES IN OVERCLOUD CREATION

Include environment files in the deployment command with the **-e** option. You can include as many environment files as necessary. However, the order of the environment files is important as the parameters and resources that you define in subsequent environment files take precedence. For example, you have two environment files that contain a common resource type **OS::TripleO::NodeExtraConfigPost**, and a common parameter **TimeZone**:

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'
```

You include both environment files in the deployment command:

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

The **openstack overcloud deploy** command runs through the following process:

1. Loads the default configuration from the core heat template collection.
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 defined in **environment-file-2.yaml**.
- **TimeZone** parameter is set to **Hongkong**, as defined in **environment-file-2.yaml**.
- **RabbitFDLimit** parameter is set to **65536**, as defined in **environment-file-1.yaml**. **environment-file-2.yaml** does not change this value.

You can use this mechanism to define custom configuration for your overcloud without values from multiple environment files conflicting.

2.6. USING CUSTOMIZED CORE HEAT TEMPLATES

When creating the overcloud, 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 the following Git workflows to manage your custom template collection:

Procedure

- Create an initial Git repository that contains the heat template collection:
  a. Copy the template collection to the **/home/stack/templates** directory:

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

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

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

c. Configure your Git user name and email address:

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

Replace **<USER_NAME>** with the user name that you want to use. Replace **<EMAIL_ADDRESS>** with your email address.

d. Stage all templates for the initial commit:

```
$ git add *
```

e. Create an initial commit:

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

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

- 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:

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

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

b. Edit the files in the custom branch.

c. Stage the changes in git:

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

d. 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.

- When you update the undercloud, the **openstack-tripleo-heat-templates** package might also receive updates. When this occurs, you must also update your custom template collection:

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

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

b. 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
```

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

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

d. Add all templates for the initial commit:

```
$ git add *
```

e. Create a commit for the package update:

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

f. Merge the branch into master. If you use a Git management system (such as GitLab), use the management workflow. If you use 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.

- Update the **my-customization** branch,:

  a. Change to the **my-customizations** branch:

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

  b. Rebase the branch off **master**:

  ```
  $ git rebase master
  ```

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

- Resolve any conflicts that occur during the rebase:

  a. Check which files contain the conflicts:

  ```
  $ git status
  ```

  b. Resolve the conflicts of the template files identified.

  c. Add the resolved files:

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

d. Continue the rebase:

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

- Deploy the custom template collection:

a. Ensure that you have switched to the **my-customization** branch:

```
git checkout my-customizations
```

b. 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

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.

2.7. JINJA2 RENDERING

The core heat templates in **/usr/share/openstack-tripleo-heat-templates** contain a number of files that have the **j2.yaml** file extension. These files contain Jinja2 template syntax and director renders these files to their static heat template equivalents that have the **.yaml** extension. For example, the main **overcloud.j2.yaml** file renders into **overcloud.yaml**. Director uses the resulting **overcloud.yaml** file.

The Jinja2-enabled heat templates use Jinja2 syntax to create parameters and resources for iterative values. For example, the **overcloud.j2.yaml** file contains the following snippet:

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

When director renders the Jinja2 syntax, director iterates over the roles defined in the **roles_data.yaml** file and populates the **{{role.name}}Count** parameter with the name of the role. The default **roles_data.yaml** file contains five roles and results in the following parameters from our example:

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

A example rendered version of the parameter looks like this:

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

Director renders Jinja2-enabled templates and environment files only from within the directory of your core heat templates. The following use cases demonstrate the correct method to render the Jinja2 templates.

Use case 1: Default core templates

Template directory: /usr/share/openstack-tripleo-heat-templates/

Environment file: /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.j2.yaml

Director uses the default core template location (**--templates**) and renders the **network-isolation.j2.yaml** file into **network-isolation.yaml**. When you run the **openstack overcloud deploy** command, use the **-e** option to include the name of the rendered **network-isolation.yaml** file.

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

Use case 2: Custom core templates

Template directory: /home/stack/tripleo-heat-templates

Environment file: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml

Director uses a custom core template location (**--templates /home/stack/tripleo-heat-templates**) and director renders the **network-isolation.j2.yaml** file within the custom core templates into **network-isolation.yaml**. When you run the **openstack overcloud deploy** command, use the **-e** option to include the name of the rendered **network-isolation.yaml** file.

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

Use case 3: Incorrect usage

Template directory: /usr/share/openstack-tripleo-heat-templates/

Environment file: /home/stack/tripleo-heat-templates/environments/network-isolation.j2.yaml

Director uses a custom core template location (**--templates /home/stack/tripleo-heat-templates**). However, the chosen **network-isolation.j2.yaml** is not located within the custom core templates, so it will not render into **network-isolation.yaml**. This causes the deployment to fail.

Processing Jinja2 syntax into static templates

Use the **process-templates.py** script to render the Jinja2 syntax of the **openstack-tripleo-heat-templates** into a set of static templates. To render a copy of the **openstack-tripleo-heat-templates** collection with the **process-templates.py** script, change to the **openstack-tripleo-heat-templates** directory:

```
$ cd /usr/share/openstack-tripleo-heat-templates
```

Run the **process-templates.py** script, which is located in the **tools** directory, along with the **-o** option to define a custom directory to save the static copy:

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

This converts all Jinja2 templates to their rendered YAML versions and saves the results to **~/openstack-tripleo-heat-templates-rendered**.

CHAPTER 3. HEAT PARAMETERS

Each heat template in the director template collection contains a **parameters** section. This section contains definitions for 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
- **deployment/*.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.

3.1. EXAMPLE 1: CONFIGURING THE TIME ZONE

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.

To obtain lists of timezones run the **timedatectl list-timezones** command. The following example command retrieves the timezones for Asia:

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

After you identify your timezone, set the *TimeZone* parameter in an environment file. The following example environment file sets the value of *TimeZone* to *Asia/Tokyo*:

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

3.2. EXAMPLE 2: CONFIGURING RABBITMQ FILE DESCRIPTOR LIMIT

For certain configurations, you might need to increase the file descriptor limit for the RabbitMQ server. Use the **deployment/rabbitmq/rabbitmq-container-puppet.yaml** heat template to set a new limit in the **RabbitFDLimit** parameter. Add the following entry to an environment file:

```
parameter_defaults:
  RabbitFDLimit: 65536
```

3.3. EXAMPLE 3: ENABLING AND DISABLING PARAMETERS

You might need to initially set a parameter 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, include the following entry in an environment file:

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

To disable this parameter from a future deployment, it is not sufficient to remove the parameter. Instead, you must set the parameter to an empty value:

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

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

3.4. EXAMPLE 4: ROLE-BASED PARAMETERS

Use the **[ROLE]Parameters** parameters, replacing **[ROLE]** with a composable role, to set parameters for a specific role.

For example, director configures **sshd** on both Controller and Compute nodes. To set a different **sshd** parameters for Controller and Compute nodes, create an environment file that contains both the **ControllerParameters** and **ComputeParameters** parameter and set the **sshd** parameters for each specific role:

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

3.5. IDENTIFYING PARAMETERS THAT YOU WANT TO MODIFY

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

1. Identify the option that you want 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, no hieradata exists for this option. If possible, you can set the option manually after the overcloud completes deployment.

3. Check the core heat template collection for the Puppet variable in the form of hieradata. The templates in **deployment/*** usually correspond to the Puppet modules of the same services. For example, the **deployment/keystone/keystone-container-puppet.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 that you can 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.

Procedure

1. To change the notification format for OpenStack Identity (keystone), use the workflow and complete the following steps:
   a. Identify the OpenStack parameter that you want to configure (**notification_format**).
   b. Search the **keystone** Puppet module for the **notification_format** setting:

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

   In this case, the **keystone** module manages this option using the **keystone::notification_format** variable.

   c. Search the **keystone** service template for this variable:

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

   The output shows that director uses 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**

You set the **KeystoneNotificationFormat** in an overcloud environment file, which then sets the **notification_format** option in the **keystone.conf** file during the overcloud configuration.

CHAPTER 4. CONFIGURATION HOOKS

Use configuration hooks to inject your own custom configuration functions into the overcloud deployment process. You can create hooks to inject custom configuration before and after the main overcloud services configuration, and hooks for modifying and including Puppet-based configuration.

4.1. FIRST BOOT: CUSTOMIZING FIRST BOOT CONFIGURATION

Director uses **cloud-init** to perform configuration on all nodes after the initial creation of the overcloud. You can use the **NodeUserData** resource types to call **cloud-init**.

OS::TripleO::NodeUserData

cloud-init configuration to apply to all nodes.

OS::TripleO::Controller::NodeUserData

cloud-init configuration to apply to Controller nodes.

OS::TripleO::Compute::NodeUserData

cloud-init configuration to apply to Compute nodes.

OS::TripleO::CephStorage::NodeUserData

cloud-init configuration to apply to Ceph Storage nodes.

OS::TripleO::ObjectStorage::NodeUserData

cloud-init configuration to apply to Object Storage nodes.

OS::TripleO::BlockStorage::NodeUserData

cloud-init configuration to apply to Block Storage nodes.

OS::TripleO::[ROLE]::NodeUserData

cloud-init configuration to apply to custom nodes. Replace **[ROLE]** with the composable role name.

In this example, update the nameserver with a custom IP address on all nodes:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to append the **resolv.conf** file on each node 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}
```

2. Create an environment file **~/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
```

3. To add the first boot configuration to your overcloud, add the environment file to the stack, along with your other environment files:

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

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

IMPORTANT

You can only register the **NodeUserData** resources to one heat template per resource. Subsequent usage overrides the heat template to use.

4.2. PRE-CONFIGURATION: CUSTOMIZING SPECIFIC OVERCLOUD ROLES

The overcloud uses Puppet for the core configuration of OpenStack components. Director provides a set of hooks that you can use to perform custom configuration for specific node roles after the first boot completes and before the core configuration begins. These hooks include:

IMPORTANT

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

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, append the **resolv.conf** file on all nodes of a particular role with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to write a variable nameserver to the **resolv.conf** file of a node:

```
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 parameters:

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 references the **CustomExtraConfigPre** resource so that heat knows which 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, you want to 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 to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/templates/pre_config.yaml** that registers your heat template to the role-based resource type. For example, to apply the configuration 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
```

3. Add the environment file to the stack, along with your other environment files:

```
$ 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 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. Director provides a hook that you can use 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, append the **resolv.conf** file on each node with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to append the **resolv.conf** file of each node 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 parameters:

CustomExtraConfigPre

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

CustomExtraDeploymentPre

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

- The **config** parameter references the **CustomExtraConfigPre** resource so that heat knows which 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, you 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 to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/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
```

3. Add the environment file to the stack, along with your other environment files:

```
$ 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 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 heat template collection requires dedicated use of these hooks, which means that you should not use them for custom use. Instead, use the **OS::TripleO::NodeExtraConfigPost** hook outlined here.

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

OS::TripleO::NodeExtraConfigPost

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

In this example, append the **resolv.conf** file on each node with a variable nameserver:

Procedure

1. Create a basic heat template **~/templates/nameserver.yaml** that runs a script to append the **resolv.conf** file of each node 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
  EndpointMap:
    default: {}
    type: json

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 parameters:

CustomExtraConfig

This defines a software configuration. In this example, you 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 references the **CustomExtraConfig** resource so that heat knows which 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, you want 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 to ensure that the resource reapplies on subsequent overcloud updates.

2. Create an environment file **~/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
```

3. Add the environment file to the stack, along with your other environment files:

```
$ 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 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 that you can use to pass extra configuration to certain node types. These parameters save the configuration as hieradata for the Puppet configuration on the node:

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.

Procedure

1. 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, use the following entries in the **ComputeExtraConfig** parameter:

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

2. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

IMPORTANT

You can define each parameter only 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:

Procedure

1. Identify the system UUID from the introspection data for a node:

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

This command returns a system UUID. For example:

```
"f5055c6c-477f-47fb-afe5-95c6928c407f"
```

2. Create an environment file to define node-specific hieradata and register the **per_node.yaml** template to a pre-configuration hook. Include the system UUID of the node that you want to configure in the **NodeDataLookup** parameter:

```
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" ]}}'
```

3. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

The **per_node.yaml** template generates a set of hieradata files on nodes that correspond to each system UUID and contains the hieradata that you define. If a UUID is not defined, the resulting hieradata file is empty. In this example, the **per_node.yaml** template runs on all Compute nodes as defined by the **OS::TripleO::ComputeExtraConfigPre** hook, but only the Compute node with system UUID **f5055c6c-477f-47fb-afe5-95c6928c407f** receives hieradata.

You can use this mechanism to tailor each node according to specific requirements.

For more information about **NodeDataLookup**, see Altering the disk layout in Ceph Storage nodes in the *Deploying an overcloud with containerized Red Hat Ceph* guide.

4.7. PUPPET: APPLYING CUSTOM MANIFESTS

In certain circumstances, you might want to install and configure some additional components on your overcloud nodes. You can achieve this with a custom Puppet manifest that applies to nodes after the main configuration completes. As a basic example, you might want to install **motd** on each node

Procedure

1. Create a heat template **~/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
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    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 example includes the **/home/stack/templates/motd.pp** within the template and passes it to nodes for configuration. The **motd.pp** file contains the Puppet classes necessary to install and configure **motd**.

2. Create an environment file **~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
```

3. Include this environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment.

```
$ 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. ANSIBLE-BASED OVERCLOUD REGISTRATION

Director uses Ansible-based methods to register overcloud nodes to the Red Hat Customer Portal or to a Red Hat Satellite Server.

If you used the **rhel-registration** method from previous Red Hat OpenStack Platform versions, you must disable it and switch to the Ansible-based method. For more information, see Switching to the rhsm composable service and RHEL-Registration to rhsm mappings .

In addition to the director-based registration method, you can also manually register after deployment. For more information, see Section 5.9, “Running Ansible-based registration manually”

5.1. RED HAT SUBSCRIPTION MANAGER (RHSM) COMPOSABLE SERVICE

You can use the **rhsm** composable service to register overcloud nodes through Ansible. Each role in the default **roles_data** file contains a **OS::TripleO::Services::Rhsm** resource, which is disabled by default. To enable the service, register the resource to the **rhsm** composable service file:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

The **rhsm** composable service accepts a **RhsmVars** parameter, which you can use to define multiple sub-parameters relevant to your registration:

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 8.2
```

You can also use the **RhsmVars** parameter in combination with role-specific parameters, for example, **ControllerParameters**, to provide flexibility when enabling specific repositories for different nodes types.

5.2. RHSMVARS SUB-PARAMETERS

Use the following sub-parameters as part of the **RhsmVars** parameter when you configure the **rhsm** composable service. For more information about the Ansible parameters that are available, see the role documentation.

rhsm	Description
rhsm_method	Choose the registration method. Either **portal**, **satellite**, or **disable**.

rhsm	Description
rhsm_org_id	The organization that you want to use for registration. To locate this ID, run **sudo subscription-manager orgs** from the undercloud node. Enter your Red Hat credentials at the prompt, and use the resulting **Key** value. For more information on your organization ID, see Understanding the Red Hat Subscription Management Organization ID.
rhsm_pool_ids	The subscription pool ID that you want to use. Use this parameter if you do not want to auto-attach subscriptions. To locate this ID, run **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"** from the undercloud node, and use the resulting**Pool ID** value. Use a list format to pass multiple IDs to this parameter.
rhsm_activation_key	The activation key that you want to use for registration.
rhsm_autosubscribe	Use this parameter to attach compatible subscriptions to this system automatically. Set the value to **true** to enable this feature.
rhsm_baseurl	The base URL for obtaining content. The default URL is the Red Hat Content Delivery Network. If you use a Satellite server, change this value to the base URL of your Satellite server content repositories.
rhsm_server_hostname	The hostname of the subscription management service for registration. The default is the Red Hat Subscription Management hostname. If you use a Satellite server, change this value to your Satellite server hostname.
rhsm_repos	A list of repositories that you want to enable.
rhsm_username	The username for registration. If possible, use activation keys for registration.
rhsm_password	The password for registration. If possible, use activation keys for registration.
rhsm_release	Red Hat Enterprise Linux release for pinning the repositories. This is set to 8.2 for Red Hat OpenStack Platform
rhsm_rhsm_proxy_hostname	The hostname for the HTTP proxy. For example: **proxy.example.com**.
rhsm_rhsm_proxy_port	The port for HTTP proxy communication. For example: **8080**.
rhsm_rhsm_proxy_user	The username to access the HTTP proxy.
rhsm_rhsm_proxy_password	The password to access the HTTP proxy.

IMPORTANT

You can use **rhsm_activation_key** and **rhsm_repos** together only if **rhsm_method** is set to **portal**. If **rhsm_method** is set to *satellite*, you can only use either **rhsm_activation_key** or **rhsm_repos**.

5.3. REGISTERING THE OVERCLOUD WITH THE RHSM COMPOSABLE SERVICE

Create an environment file that enables and configures the **rhsm** composable service. Director uses this environment file to register and subscribe your nodes.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-8-for-x86_64-baseos-tus-rpms
      - rhel-8-for-x86_64-appstream-tus-rpms
      - rhel-8-for-x86_64-highavailability-tus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: 8.2
```

- The **resource_registry** section associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

- The **RhsmVars** variable passes parameters to Ansible for configuring your Red Hat registration.

3. Save the environment file.

5.4. APPLYING THE RHSM COMPOSABLE SERVICE TO DIFFERENT ROLES

You can apply the **rhsm** composable service on a per-role basis. For example, you can apply different sets of configurations to Controller nodes, Compute nodes, and Ceph Storage nodes.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-tus-rpms
        - rhel-8-for-x86_64-appstream-tus-rpms
        - rhel-8-for-x86_64-highavailability-tus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - advanced-virt-for-rhel-8-x86_64-eus-rpms
        - openstack-16.1-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-tus-rpms
        - rhel-8-for-x86_64-appstream-tus-rpms
        - rhel-8-for-x86_64-highavailability-tus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - advanced-virt-for-rhel-8-x86_64-eus-rpms
        - openstack-16.1-for-rhel-8-x86_64-rpms
        - fast-datapath-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 8.2
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-8-for-x86_64-baseos-tus-rpms
        - rhel-8-for-x86_64-appstream-tus-rpms
        - rhel-8-for-x86_64-highavailability-tus-rpms
        - ansible-2.9-for-rhel-8-x86_64-rpms
        - openstack-16.1-deployment-tools-for-rhel-8-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
      rhsm_method: "portal"
      rhsm_release: 8.2
```

The **resource_registry** associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

The **ControllerParameters**, **ComputeParameters**, and **CephStorageParameters** parameters each use a separate **RhsmVars** parameter to pass subscription details to their respective roles.

NOTE

Set the **RhsmVars** parameter within the **CephStorageParameters** parameter to use a Red Hat Ceph Storage subscription and repositories specific to Ceph Storage. Ensure the **rhsm_repos** parameter contains the standard Red Hat Enterprise Linux repositories instead of the Extended Update Support (EUS) repositories that Controller and Compute nodes require.

3. Save the environment file.

5.5. REGISTERING THE OVERCLOUD TO RED HAT SATELLITE SERVER

Create an environment file that enables and configures the **rhsm** composable service to register nodes to Red Hat Satellite instead of the Red Hat Customer Portal.

Procedure

1. Create an environment file named **templates/rhsm.yml** to store the configuration.

2. Include your configuration in the environment file. For example:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 8.2
```

   The **resource_registry** associates the **rhsm** composable service with the **OS::TripleO::Services::Rhsm** resource, which is available on each role.

   The **RhsmVars** variable passes parameters to Ansible for configuring your Red Hat registration.

3. Save the environment file.

5.6. SWITCHING TO THE RHSM COMPOSABLE SERVICE

The previous **rhel-registration** method runs a bash script to handle the overcloud registration. The scripts and environment files for this method are located in the core heat template collection at **/usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/**.

Complete the following steps to switch from the **rhel-registration** method to the **rhsm** composable service.

Procedure

1. Exclude the **rhel-registration** environment files from future deployments operations. In most cases, exclude the following files:

- **rhel-registration/environment-rhel-registration.yaml**
- **rhel-registration/rhel-registration-resource-registry.yaml**

2. If you use a custom **roles_data** file, ensure that each role in your **roles_data** file contains the **OS::TripleO::Services::Rhsm** composable service. For example:

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  ...
  ServicesDefault:
    ...
    - OS::TripleO::Services::Rhsm
    ...
```

3. Add the environment file for **rhsm** composable service parameters to future deployment operations.

This method replaces the **rhel-registration** parameters with the **rhsm** service parameters and changes the heat resource that enables the service from:

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

To:

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

You can also include the **/usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml** environment file with your deployment to enable the service.

5.7. RHEL-REGISTRATION TO RHSM MAPPINGS

To help transition your details from the **rhel-registration** method to the **rhsm** method, use the following table to map your parameters and values.

rhel-registration	rhsm / RhsmVars
rhel_reg_method	**rhsm_method**
rhel_reg_org	**rhsm_org_id**
rhel_reg_pool_id	**rhsm_pool_ids**
rhel_reg_activation_key	**rhsm_activation_key**

rhel-registration	rhsm / RhsmVars
rhel_reg_auto_attach	**rhsm_autosubscribe**
rhel_reg_sat_url	**rhsm_satellite_url**
rhel_reg_repos	**rhsm_repos**
rhel_reg_user	**rhsm_username**
rhel_reg_password	**rhsm_password**
rhel_reg_release	**rhsm_release**
rhel_reg_http_proxy_host	**rhsm_rhsm_proxy_hostname**
rhel_reg_http_proxy_port	**rhsm_rhsm_proxy_port**
rhel_reg_http_proxy_username	**rhsm_rhsm_proxy_user**
rhel_reg_http_proxy_password	**rhsm_rhsm_proxy_password**

5.8. DEPLOYING THE OVERCLOUD WITH THE RHSM COMPOSABLE SERVICE

Deploy the overcloud with the **rhsm** composable service so that Ansible controls the registration process for your overcloud nodes.

Procedure

1. Include **rhsm.yml** environment file with the **openstack overcloud deploy** command:

   ```
   openstack overcloud deploy \
       <other cli args> \
       -e ~/templates/rhsm.yaml
   ```

   This enables the Ansible configuration of the overcloud and the Ansible-based registration.

2. Wait until the overcloud deployment completes.

3. Check the subscription details on your overcloud nodes. For example, log in to a Controller node and run the following commands:

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

5.9. RUNNING ANSIBLE-BASED REGISTRATION MANUALLY

You can perform manual Ansible-based registration on a deployed overcloud with the dynamic inventory script on the director node. Use this script to define node roles as host groups and then run a playbook against them with **ansible-playbook**. Use the following example playbook to register Controller nodes manually.

Procedure

1. Create a playbook that uses the **redhat_subscription** modules to register your nodes. For example, the following playbook applies to Controller nodes:

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - openstack-beta-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.2
        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.
     - Disable any auto-enabled repositories.
     - Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

2. After you deploy 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 performs the following actions:

   - 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 **hosts** parameter of the playbook, which in this case is the Controller group.

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, you can also create custom roles that contain specific sets of services.

You can use this flexibility 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 ROLE ARCHITECTURE

The following architectures are available when you use custom roles and composable services:

Default architecture

Uses the default **roles_data** files. All controller services are contained within one Controller role.

Supported standalone roles

Use the predefined files in **/usr/share/openstack-tripleo-heat-templates/roles** to generate a custom **roles_data** file`. For more information, see Section 6.4, "Supported custom roles".

Custom composable services

Create your own roles and use them to generate a custom **roles_data** file. Note that only a limited number of composable service combinations have been tested and verified and Red Hat cannot support all composable service combinations.

6.2. EXAMINING THE ROLES_DATA FILE

The **roles_data** file contains a YAML-formatted list of the roles that director deploys onto nodes. Each role contains definitions of all of the services that comprise the role. Use the following example snippet to understand the **roles_data** syntax:

```
- name: Controller
  description: |
    Controller role that has all the controller 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 contains definitions of the following role types:

- **Controller**

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

The **openstack overcloud deploy** command includes the default **roles_data.yaml** file during deployment. However, you can use the **-r** argument to override this file with a custom **roles_data** file:

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

6.3. CREATING A ROLES_DATA FILE

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

Procedure

1. List the default role templates:

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

2. View the role definition in YAML format with the **openstack overcloud roles show** command:

   ```
   $ openstack overcloud roles show Compute
   ```

3. Generate a custom **roles_data** file. Use the **openstack overcloud roles generate** command to join multiple predefined roles into a single file. For example, run the following command to generate a **roles_data.yaml** file that contains the **Controller**, **Compute**, and **Networker** roles:

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

   Use the **-o** option to define the name out of the output file.

   This command 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 that the networking services scale from the **Controller** role to the **Networker** role and the overcloud balances 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 that you require. This allows you to generate a **roles_data** file from your own custom roles.

4. Copy the directory from the core heat template collection to the home directory of the **stack** user:

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

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

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

   This command 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** in combination with the standalone **Networker**, **Messaging**, and **Database** roles.

6.4. SUPPORTED CUSTOM ROLES

The following table contains information about the available custom roles. You can find custom role templates in the **/usr/share/openstack-tripleo-heat-templates/roles** directory.

Role	Description	File
BlockStorage	OpenStack Block Storage (cinder) node.	**BlockStorage.yaml**
CephAll	Full standalone Ceph Storage node. Includes OSD, MON, Object Gateway (RGW), Object Operations (MDS), Manager (MGR), and RBD Mirroring.	**CephAll.yaml**
CephFile	Standalone scale-out Ceph Storage file role. Includes OSD and Object Operations (MDS).	**CephFile.yaml**
CephObject	Standalone scale-out Ceph Storage object role. Includes OSD and Object Gateway (RGW).	**CephObject.yaml**
CephStorage	Ceph Storage OSD node role.	**CephStorage.yaml**
ComputeAlt	Alternate Compute node role.	**ComputeAlt.yaml**
ComputeDVR	DVR enabled Compute node role.	**ComputeDVR.yaml**
ComputeHCI	Compute node with hyper-converged infrastructure. Includes Compute and Ceph OSD services.	**ComputeHCI.yaml**

Role	Description	File
ComputeInstanceHA	Compute Instance HA node role. Use in conjunction with the **environments/compute-instanceha.yaml`** environment file.	**ComputeInstanceHA.yaml**
ComputeLiquidio	Compute node with Cavium Liquidio Smart NIC.	**ComputeLiquidio.yaml**
ComputeOvsDpdkRT	Compute OVS DPDK RealTime role.	**ComputeOvsDpdkRT.yaml**
ComputeOvsDpdk	Compute OVS DPDK role.	**ComputeOvsDpdk.yaml**
ComputePPC64LE	Compute role for ppc64le servers.	**ComputePPC64LE.yaml**
ComputeRealTime	Compute role optimized for real-time behaviour. When using this role, it is mandatory that an **overcloud-realtime-compute** image is available and the role specific parameters **IsolCpusList**, **NovaComputeCpuDedicatedSet** and **NovaComputeCpuSharedSet** are set according to the hardware of the real-time compute nodes.	**ComputeRealTime.yaml**
ComputeSriovRT	Compute SR-IOV RealTime role.	**ComputeSriovRT.yaml**
ComputeSriov	Compute SR-IOV role.	**ComputeSriov.yaml**
Compute	Standard Compute node role.	**Compute.yaml**
ControllerAllNovaStandalone	Controller role that does not contain the database, messaging, networking, and OpenStack Compute (nova) control components. Use in combination with the **Database**, **Messaging**, **Networker**, and **Novacontrol** roles.	**ControllerAllNovaStandalone.yaml**
ControllerNoCeph	Controller role with core Controller services loaded but no Ceph Storage (MON) components. This role handles database, messaging, and network functions but not any Ceph Storage functions.	**ControllerNoCeph.yaml**
ControllerNovaStandalone	Controller role that does not contain the OpenStack Compute (nova) control component. Use in combination with the **Novacontrol** role.	**ControllerNovaStandalone.yaml**

Role	Description	File
ControllerOpenstack	Controller role that does not contain the database, messaging, and networking components. Use in combination with the **Database**, **Messaging**, and **Networker** roles.	**ControllerOpenstack.yaml**
ControllerStorageNfs	Controller role with all core services loaded and uses Ceph NFS. This roles handles database, messaging, and network functions.	**ControllerStorageNfs.yaml**
Controller	Controller role with all core services loaded. This roles handles database, messaging, and network functions.	**Controller.yaml**
ControllerSriov (ML2/OVN)	Same as the normal Controller role but with the OVN Metadata agent deployed.	**ControllerSriov.yaml**
Database	Standalone database role. Database managed as a Galera cluster using Pacemaker.	**Database.yaml**
HciCephAll	Compute node with hyper-converged infrastructure and all Ceph Storage services. Includes OSD, MON, Object Gateway (RGW), Object Operations (MDS), Manager (MGR), and RBD Mirroring.	**HciCephAll.yaml**
HciCephFile	Compute node with hyper-converged infrastructure and Ceph Storage file services. Includes OSD and Object Operations (MDS).	**HciCephFile.yaml**
HciCephMon	Compute node with hyper-converged infrastructure and Ceph Storage block services. Includes OSD, MON, and Manager.	**HciCephMon.yaml**
HciCephObject	Compute node with hyper-converged infrastructure and Ceph Storage object services. Includes OSD and Object Gateway (RGW).	**HciCephObject.yaml**
IronicConductor	Ironic Conductor node role.	**IronicConductor.yaml**
Messaging	Standalone messaging role. RabbitMQ managed with Pacemaker.	**Messaging.yaml**
Networker	Standalone networking role. Runs OpenStack networking (neutron) agents on their own. If your deployment uses the ML2/OVN mechanism driver, see additional steps in Deploying a Custom Role with ML2/OVN.	**Networker.yaml**

Role	Description	File
NetworkerSriov	Same as the normal Networker role but with the OVN Metadata agent deployed. See additional steps in Deploying a Custom Role with ML2/OVN	**NetworkerSriov.yaml**
Novacontrol	Standalone **nova-control** role to run OpenStack Compute (nova) control agents on their own.	**Novacontrol.yaml**
ObjectStorage	Swift Object Storage node role.	**ObjectStorage.yaml**
Telemetry	Telemetry role with all the metrics and alarming services.	**Telemetry.yaml**

6.5. EXAMINING ROLE PARAMETERS

Each role contains 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**.

description

(Optional) A plain text description for the role.

tags

(Optional) A YAML list of tags that 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 that you define becomes the primary role. Ensure that this role is the Controller role.

networks

A YAML list or dictionary of networks that you want to configure on the role. If you use a YAML list, list each composable network:

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

If you use a dictionary, map each network to a specific **subnet** in your composable networks.

```
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
    StorageMgmt:
      subnet: storage_mgmt_subnet
    Tenant:
      subnet: tenant_subnet
```

Default networks include **External**, **InternalApi**, **Storage**, **StorageMgmt**, **Tenant**, and **Management**.

CountDefault

(Optional) Defines the default number of nodes that you want 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 director. Use this parameter when you deploy an overcloud with pre-provisioned nodes. For more information, see Configuring a Basic Overcloud with Pre-Provisioned Nodes in the *Director Installation and Usage* guide.

update_serial

(Optional) Defines how many nodes to update simultaneously during the OpenStack update options. In the default **roles_data.yaml** file:

- The default is **1** for Controller, Object Storage, and Ceph Storage nodes.
- The default is **25** for Compute and Block Storage nodes.

If you omit this parameter from a custom role, the default is **1**.

ServicesDefault

(Optional) Defines the default list of services to include on the node. For more information, see Section 6.8, "Examining composable service architecture" .

You can use these parameters to create new roles and also define which services to include in your roles.

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.

For example, the following snippet contains the resource definition for each role in the **overcloud.j2.yaml** heat template:

```
  {{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 an **OS::Heat::ResourceGroup** resource. This in turn uses each **name** parameter from the **roles_data** file to name each respective **OS::Heat::ResourceGroup** resource.

6.6. CREATING A NEW ROLE

You can use the composable service architecture to create new roles according to the requirements of your deployment. For example, you might want to create a new **Horizon** role to host only the OpenStack Dashboard (**horizon**).

NOTE

Role names must start with a letter, end with a letter or digit, and contain only letters, digits, and hyphens. Underscores must never be used in role names.

Procedure

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

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

2. Create a new file called **~/roles/Horizon.yaml** and create a new **Horizon** role that contains base and core OpenStack Dashboard services:

```
- 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
```

Set the **CountDefault** to **1** so that a default overcloud always includes the **Horizon** node.

3. Optional: If you want to scale the services in an existing overcloud, retain the existing services on the **Controller** role. If you want to create 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
    ...
```

4. Generate the new **roles_data-horizon.yaml** file using the **~/roles** directory as the source:

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

5. 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:

   a. Create a **horizon** flavor:

   ```
   (undercloud)$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4 horizon
   ```

NOTE

These properties are not used for scheduling instances, however, the Compute scheduler does use the disk size to determine the root partition size.

b. Tag each bare metal node that you want to designate for the Dashboard service (horizon) with a custom resource class:

```
(undercloud)$ openstack baremetal node set --resource-class baremetal.HORIZON
<NODE>
```

Replace **<NODE>** with the ID of the bare metal node.

c. Associate the **horizon** flavor with the custom resource class:

```
(undercloud)$ openstack flavor set --property
resources:CUSTOM_BAREMETAL_HORIZON=1 horizon
```

To determine the name of a custom resource class that corresponds to a resource class of a bare metal node, convert the resource class to uppercase, replace punctuation with an underscore, and prefix the value with **CUSTOM_**.

NOTE

A flavor can request only one instance of a bare metal resource class.

d. Set the following flavor properties to prevent the Compute scheduler from using the bare metal flavor properties for scheduling instances:

```
(undercloud)$ openstack flavor set --property resources:VCPU=0 --property
resources:MEMORY_MB=0 --property resources:DISK_GB=0 horizon
```

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

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

7. Include the new **roles_data-horizon.yaml** file and environment file in the **openstack overcloud deploy** command, along with any other environment files relevant to your deployment:

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

This configuration creates a three-node overcloud that consists of one Controller node, one Compute node, and one Networker node. To view the list of nodes in your overcloud, run the following command:

```
$ openstack server list
```

6.7. GUIDELINES AND LIMITATIONS

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

For services not managed by Pacemaker:

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

For services managed by Pacemaker:

- You can assign Pacemaker-managed services to standalone custom roles.
- Pacemaker has a 16 node limit. If you assign the Pacemaker service (**OS::TripleO::Services::Pacemaker**) to 16 nodes, 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 role that contains **OS::TripleO::Services::Pacemaker** or **OS::TripleO::Services::PacemakerRemote** services.

General limitations:

- You cannot change custom roles and composable services during a major version upgrade.
- 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.

6.8. EXAMINING COMPOSABLE SERVICE ARCHITECTURE

The core heat template collection contains two sets of composable service templates:

- **deployment** contains the templates for key OpenStack services.
- **puppet/services** contains legacy templates for configuring composable services. In some cases, the composable services use templates from this directory for compatibility. In most cases, the composable services use the templates in the **deployment** directory.

Each template contains a description that identifies its purpose. For example, the **deployment/time/ntp-baremetal-puppet.yaml** service template contains the following description:

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

These service templates are registered as resources specific to a Red Hat OpenStack Platform deployment. This means that 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.

Some resources use the base composable service templates directly:

```
resource_registry:
  ...
  OS::TripleO::Services::Ntp: deployment/time/ntp-baremetal-puppet.yaml
  ...
```

However, core services require containers and use the containerized service templates. For example, the **keystone** containerized service uses the following resource:

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: deployment/keystone/keystone-container-puppet.yaml
  ...
```

These containerized templates usually reference other templates to include dependencies. For example, the **deployment/keystone/keystone-container-puppet.yaml** template stores the output of the base template in the **ContainersCommon** resource:

```
resources:
  ContainersCommon:
    type: ../containers-common.yaml
```

The containerized template can then incorporate functions and data from the **containers-common.yaml** template.

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.

NOTE

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.

6.9. 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 want to remove OpenStack Orchestration (heat) from the Controller nodes.

Procedure

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

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

2. 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
   ```

3. Generate the new **roles_data** file:

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

4. Include this new **roles_data** file when you run the **openstack overcloud deploy** command:

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

   This command 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
```

6.10. 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 entry:

Procedure

1. Add an entry in an environment file that links the **CinderBackup** service to the heat template that contains the **cinder-backup** configuration:

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

   This entry overrides the default null operation resource and enables the service.

2. Include this environment file when you run the **openstack overcloud deploy** command:

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

6.11. CREATING A GENERIC NODE WITH NO SERVICES

You can create generic Red Hat Enterprise Linux 8.2 nodes without any OpenStack services configured. This is useful when you need to host software outside of the core Red Hat OpenStack Platform (RHOSP) environment. For example, RHOSP provides integration with monitoring tools such as Kibana and Sensu. For more information, see the *Monitoring Tools Configuration Guide* . While Red Hat does not provide support for the monitoring tools themselves, director can create a generic Red Hat Enterprise Linux 8.2 node to host these tools.

NOTE

The generic node still uses the base **overcloud-full** image rather than a base Red Hat Enterprise Linux 8 image. This means the node has some Red Hat OpenStack Platform software installed but not enabled or configured.

Procedure

1. Create a generic role in your custom **roles_data.yaml** file that does not contain a **ServicesDefault** list:

```
- name: Generic
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

   Ensure that you retain the existing **Controller** and **Compute** roles.

2. Create an environment file **generic-node-params.yaml** to specify how many generic Red Hat Enterprise Linux 8 nodes you require and the flavor when selecting nodes to provision:

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

3. Include both the roles file and the environment file when you run the **openstack overcloud deploy** command:

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

   This configuration deploys a three-node environment with one Controller node, one Compute node, and one generic Red Hat Enterprise Linux 8 node.

CHAPTER 7. CONTAINERIZED SERVICES

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

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/deployment/**.

You must enable the **OS::TripleO::Services::Podman** service in the role for all nodes that use containerized services. When you create a **roles_data.yaml** file for your custom roles configuration, include the **OS::TripleO::Services::Podman** service along with the base composable services. For example, the **IronicConductor** role uses the following role definition:

```
- name: IronicConductor
  description: |
    Ironic Conductor node role
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-ironic-%index%'
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Fluentd
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicPxe
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
```

7.2. CONTAINERIZED SERVICE PARAMETERS

Each containerized service template contains an **outputs** section that defines a data set passed to the OpenStack Orchestration (heat) service. In addition to the standard composable service parameters (see Section 6.5, "Examining role parameters"), the template contains 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, 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 volume that stores the configuration.
- **puppet_tags** - Tags to pass to Puppet during configuration. OpenStack uses these tags to restrict the Puppet run to the configuration resource of a particular service. For example, the OpenStack Identity (keystone) containerized service uses the **keystone_config** tag to ensure that all require 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 configuration container for the service. All tasks are grouped into the following steps to help director perform a staged deployment:

- 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. PREPARING CONTAINER IMAGES

The overcloud installation requires an environment file to determine where to obtain container images and how to store them. Generate and customize this environment file that you can use to prepare your container images.

NOTE

If you need to configure specific container image versions for your overcloud, you must pin the images to a specific version. For more information, see Pinning container images for the overcloud.

Procedure

1. Log in to your undercloud host as the **stack** user.

2. Generate the default container image preparation file:

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

   This command includes the following additional options:

   - **--local-push-destination** sets the registry on the undercloud as the location for container images. This means that director pulls the necessary images from the Red Hat Container Catalog and pushes them to the registry on the undercloud. Director uses this registry as the container image source. To pull directly from the Red Hat Container Catalog, omit this option.

   - **--output-env-file** is an environment file name. The contents of this file include the parameters for preparing your container images. In this case, the name of the file is **containers-prepare-parameter.yaml**.

   NOTE

   You can use the same **containers-prepare-parameter.yaml** file to define a container image source for both the undercloud and the overcloud.

3. Modify the **containers-prepare-parameter.yaml** to suit your requirements.

7.4. CONTAINER IMAGE PREPARATION PARAMETERS

The default file for preparing your containers (**containers-prepare-parameter.yaml**) contains the **ContainerImagePrepare** heat parameter. This parameter defines a list of strategies for preparing a set of images:

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

Each strategy accepts a set of sub-parameters that defines which images to use and what to do with the images. The following table contains information about the sub-parameters that you can use with each **ContainerImagePrepare** strategy:

Parameter	Description
excludes	List of regular expressions to exclude image names from a strategy.

Parameter	Description
includes	List of regular expressions to include in a strategy. At least one image name must match an existing image. All **excludes** are ignored if **includes** is specified.
modify_append_tag	String to append to the tag for the destination image. For example, if you pull an image with the tag 16.1.3-5.161 and set the **modify_append_tag** to **-hotfix**, the director tags the final image as 16.1.3-5.161-hotfix.
modify_only_with_labels	A dictionary of image labels that filter the images that you want to modify. If an image matches the labels defined, the director includes the image in the modification process.
modify_role	String of ansible role names to run during upload but before pushing the image to the destination registry.
modify_vars	Dictionary of variables to pass to **modify_role**.
push_destination	Defines the namespace of the registry that you want to push images to during the upload process. • If set to **true**, the **push_destination** is set to the undercloud registry namespace using the hostname, which is the recommended method. • If set to **false**, the push to a local registry does not occur and nodes pull images directly from the source. • If set to a custom value, director pushes images to an external local registry. If you set this parameter to **false** in production environments while pulling images directly from Red Hat Container Catalog, all overcloud nodes will simultaneously pull the images from the Red Hat Container Catalog over your external connection, which can cause bandwidth issues. Only use **false** to pull directly from a Red Hat Satellite Server hosting the container images. If the **push_destination** parameter is set to **false** or is not defined and the remote registry requires authentication, set the **ContainerImageRegistryLogin** parameter to **true** and include the credentials with the **ContainerImageRegistryCredentials** parameter.

Parameter	Description
pull_source	The source registry from where to pull the original container images.
set	A dictionary of **key: value** definitions that define where to obtain the initial images.
tag_from_label	Use the value of specified container image metadata labels to create a tag for every image and pull that tagged image. For example, if you set **tag_from_label: {version}-{release}**, director uses the **version** and **release** labels to construct a new tag. For one container, **version** might be set to 16.1.3 and **release** might be set to **5.161**, which results in the tag 16.1.3-5.161. Director uses this parameter only if you have not defined **tag** in the **set** dictionary.

IMPORTANT

When you push images to the undercloud, use **push_destination: true** instead of **push_destination: UNDERCLOUD_IP:PORT**. The **push_destination: true** method provides a level of consistency across both IPv4 and IPv6 addresses.

The **set** parameter accepts a set of **key: value** definitions:

Key	Description
ceph_image	The name of the Ceph Storage container image.
ceph_namespace	The namespace of the Ceph Storage container image.
ceph_tag	The tag of the Ceph Storage container image.
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	The name, namespace, and tag of the Ceph Storage Alert Manager container image.
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	The name, namespace, and tag of the Ceph Storage Grafana container image.

Key	Description
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	The name, namespace, and tag of the Ceph Storage Node Exporter container image.
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	The name, namespace, and tag of the Ceph Storage Prometheus container image.
name_prefix	A prefix for each OpenStack service image.
name_suffix	A suffix for each OpenStack service image.
namespace	The namespace for each OpenStack service image.
neutron_driver	The driver to use to determine which OpenStack Networking (neutron) container to use. Use a null value to set to the standard **neutron-server** container. Set to **ovn** to use OVN-based containers.
tag	Sets a specific tag for all images from the source. If not defined, director uses the Red Hat OpenStack Platform version number as the default value. This parameter takes precedence over the **tag_from_label** value.

NOTE

The container images use multi-stream tags based on the Red Hat OpenStack Platform version. This means that there is no longer a **latest** tag.

7.5. GUIDELINES FOR CONTAINER IMAGE TAGGING

The Red Hat Container Registry uses a specific version format to tag all Red Hat OpenStack Platform container images. This format follows the label metadata for each container, which is **version-release**.

version

Corresponds to a major and minor version of Red Hat OpenStack Platform. These versions act as streams that contain one or more releases.

release

Corresponds to a release of a specific container image version within a version stream.

For example, if the latest version of Red Hat OpenStack Platform is 16.1.3 and the release for the container image is **5.161**, then the resulting tag for the container image is 16.1.3-5.161.

The Red Hat Container Registry also uses a set of major and minor **version** tags that link to the latest release for that container image version. For example, both 16.1 and 16.1.3 link to the latest **release** in the

16.1.3 container stream. If a new minor release of 16.1 occurs, the 16.1 tag links to the latest **release** for the new minor release stream while the 16.1.3 tag continues to link to the latest **release** within the 16.1.3 stream.

The **ContainerImagePrepare** parameter contains two sub-parameters that you can use to determine which container image to download. These sub-parameters are the **tag** parameter within the **set** dictionary, and the **tag_from_label** parameter. Use the following guidelines to determine whether to use **tag** or **tag_from_label**.

- The default value for **tag** is the major version for your OpenStack Platform version. For this version it is 16.1. This always corresponds to the latest minor version and release.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        tag: 16.1
        ...
  ```

- To change to a specific minor version for OpenStack Platform container images, set the tag to a minor version. For example, to change to 16.1.2, set **tag** to 16.1.2.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        tag: 16.1.2
        ...
  ```

- When you set **tag**, director always downloads the latest container image **release** for the version set in **tag** during installation and updates.

- If you do not set **tag**, director uses the value of **tag_from_label** in conjunction with the latest major version.

  ```
  parameter_defaults:
    ContainerImagePrepare:
    - set:
        ...
        # tag: 16.1
        ...
      tag_from_label: '{version}-{release}'
  ```

- The **tag_from_label** parameter generates the tag from the label metadata of the latest container image release it inspects from the Red Hat Container Registry. For example, the labels for a certain container might use the following **version** and **release** metadata:

  ```
    "Labels": {
      "release": "5.161",
      "version": "16.1.3",
      ...
    }
  ```

- The default value for **tag_from_label** is **{version}-{release}**, which corresponds to the version and release metadata labels for each container image. For example, if a container image has

16.1.3 set for **version** and 5.161 set for **release**, the resulting tag for the container image is 16.1.3-5.161.

- The **tag** parameter always takes precedence over the **tag_from_label** parameter. To use **tag_from_label**, omit the **tag** parameter from your container preparation configuration.
- A key difference between **tag** and **tag_from_label** is that director uses **tag** to pull an image only based on major or minor version tags, which the Red Hat Container Registry links to the latest image release within a version stream, while director uses **tag_from_label** to perform a metadata inspection of each container image so that director generates a tag and pulls the corresponding image.

7.6. OBTAINING CONTAINER IMAGES FROM PRIVATE REGISTRIES

The **registry.redhat.io** registry requires authentication to access and pull images. To authenticate with **registry.redhat.io** and other private registries, include the **ContainerImageRegistryCredentials** and **ContainerImageRegistryLogin** parameters in your **containers-prepare-parameter.yaml** file.

ContainerImageRegistryCredentials

Some container image registries require authentication to access images. In this situation, use the **ContainerImageRegistryCredentials** parameter in your **containers-prepare-parameter.yaml** environment file. The **ContainerImageRegistryCredentials** parameter uses a set of keys based on the private registry URL. Each private registry URL uses its own key and value pair to define the username (key) and password (value). This provides a method to specify credentials for multiple private registries.

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

In the example, replace **my_username** and **my_password** with your authentication credentials. Instead of using your individual user credentials, Red Hat recommends creating a registry service account and using those credentials to access **registry.redhat.io** content.

To specify authentication details for multiple registries, set multiple key-pair values for each registry in **ContainerImageRegistryCredentials**:

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

IMPORTANT

The default **ContainerImagePrepare** parameter pulls container images from **registry.redhat.io**, which requires authentication.

For more information, see Red Hat Container Registry Authentication .

ContainerImageRegistryLogin

The **ContainerImageRegistryLogin** parameter is used to control whether an overcloud node system needs to log in to the remote registry to fetch the container images. This situation occurs when you want the overcloud nodes to pull images directly, rather than use the undercloud to host images.

You must set **ContainerImageRegistryLogin** to **true** if **push_destination** is set to false or not used for a given strategy.

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

However, if the overcloud nodes do not have network connectivity to the registry hosts defined in **ContainerImageRegistryCredentials** and you set **ContainerImageRegistryLogin** to **true**, the deployment might fail when trying to perform a login. If the overcloud nodes do not have network connectivity to the registry hosts defined in the **ContainerImageRegistryCredentials**, set **push_destination** to **true** and **ContainerImageRegistryLogin** to **false** so that the overcloud nodes pull images from the undercloud.

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

7.7. LAYERING IMAGE PREPARATION ENTRIES

The value of the **ContainerImagePrepare** parameter is a YAML list. This means that you can specify multiple entries. The following example demonstrates two entries where director uses the latest version of all images except for the **nova-api** image, which uses the version tagged with **16.2-44**:

```
ContainerImagePrepare:
- tag_from_label: "{version}-{release}"
  push_destination: true
  excludes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp-rhel8
    name_prefix: openstack-
    name_suffix: ''
- push_destination: true
  includes:
  - nova-api
  set:
    namespace: registry.redhat.io/rhosp-rhel8
    tag: 16.2-44
```

The **includes** and **excludes** parameters use regular expressions to control image filtering for each entry. The images that match the **includes** strategy take precedence over **excludes** matches. The image name must the **includes** or **excludes** regular expression value to be considered a match.

7.8. MODIFYING IMAGES DURING PREPARATION

It is possible to modify images during image preparation, and then immediately deploy the overcloud with modified images.

NOTE

Red Hat OpenStack Platform (RHOSP) director supports modifying images during preparation for RHOSP containers, not for Ceph containers.

Scenarios for modifying images include:

- As part of a continuous integration pipeline where images are modified with the changes being tested before deployment.
- As part of a development workflow where local changes must be deployed for testing and development.
- When changes must be deployed but are not available through an image build pipeline. For example, adding proprietary add-ons or emergency fixes.

To modify an image during preparation, invoke an Ansible role on each image that you want to modify. The role takes a source image, makes the requested changes, and tags the result. The prepare command can push the image to the destination registry and set the heat parameters to refer to the modified image.

The Ansible role **tripleo-modify-image** conforms with the required role interface and provides the behaviour necessary for the modify use cases. Control the modification with the modify-specific keys in the **ContainerImagePrepare** parameter:

- **modify_role** specifies the Ansible role to invoke for each image to modify.
- **modify_append_tag** appends a string to the end of the source image tag. This makes it obvious that the resulting image has been modified. Use this parameter to skip modification if the **push_destination** registry already contains the modified image. Change **modify_append_tag** whenever you modify the image.
- **modify_vars** is a dictionary of Ansible variables to pass to the role.

To select a use case that the **tripleo-modify-image** role handles, set the **tasks_from** variable to the required file in that role.

While developing and testing the **ContainerImagePrepare** entries that modify images, run the image prepare command without any additional options to confirm that the image is modified as you expect:

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

IMPORTANT

To use the **openstack tripleo container image prepare** command, your undercloud must contain a running **image-serve** registry. As a result, you cannot run this command before a new undercloud installation because the **image-serve** registry will not be installed. You can run this command after a successful undercloud installation.

7.9. UPDATING EXISTING PACKAGES ON CONTAINER IMAGES

NOTE

Red Hat OpenStack Platform (RHOSP) director supports updating existing packages on container images for RHOSP containers, not for Ceph containers.

Procedure

- The following example **ContainerImagePrepare** entry updates in all packages on the container images by using the dnf repository configuration of the undercloud host:

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

7.10. INSTALLING ADDITIONAL RPM FILES TO CONTAINER IMAGES

You can install a directory of RPM files in your container images. This is useful for installing hotfixes, local package builds, or any package that is not available through a package repository.

NOTE

Red Hat OpenStack Platform (RHOSP) director supports installing additional RPM files to container images for RHOSP containers, not for Ceph containers.

NOTE

When you modify container images in existing deployments, you must then perform a minor update to apply the changes to your overcloud. For more information, see Keeping Red Hat OpenStack Platform Updated.

Procedure

- The following example **ContainerImagePrepare** entry installs some hotfix packages on only the **nova-compute** image:

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

7.11. MODIFYING CONTAINER IMAGES WITH A CUSTOM DOCKERFILE

You can specify a directory that contains a Dockerfile to make the required changes. When you invoke the **tripleo-modify-image** role, the role generates a **Dockerfile.modified** file that changes the **FROM** directive and adds extra **LABEL** directives.

NOTE

Red Hat OpenStack Platform (RHOSP) director supports modifying container images with a custom Dockerfile for RHOSP containers, not for Ceph containers.

Procedure

1. The following example runs the custom Dockerfile on the **nova-compute** image:

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

2. The following example shows the **/home/stack/nova-custom/Dockerfile** file. After you run any **USER** root directives, you must switch back to the original image default user:

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

USER "root"

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

USER "nova"
```

7.12. DEPLOYING A VENDOR PLUGIN

To use some third-party hardware as a Block Storage back end, you must deploy a vendor plugin. The following example demonstrates how to deploy a vendor plugin to use Dell EMC hardware as a Block Storage back end.

For more information about supported back end appliances and drivers, see Third-Party Storage Providers in the *Storage Guide*.

Procedure

1. Create a new container images file for your overcloud:

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

2. Edit the containers-prepare-parameter-dellemc.yaml file.

3. Add an **exclude** parameter to the strategy for the main Red Hat OpenStack Platform container images. Use this parameter to exclude the container image that the vendor container image will replace. In the example, the container image is the **cinder-volume** image:

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      excludes:
      - cinder-volume
      set:
        namespace: registry.redhat.io/rhosp-rhel8
        name_prefix: openstack-
        name_suffix: ''
        tag: 16.1
        ...
      tag_from_label: "{version}-{release}"
```

4. Add a new strategy to the **ContainerImagePrepare** parameter that includes the replacement container image for the vendor plugin:

```
parameter_defaults:
  ContainerImagePrepare:
    ...
    - push_destination: true
      includes:
        - cinder-volume
      set:
        namespace: registry.connect.redhat.com/dellemc
        name_prefix: openstack-
        name_suffix: -dellemc-rhosp16
        tag: 16.1-2
        ...
```

5. Add the authentication details for the registry.connect.redhat.com registry to the **ContainerImageRegistryCredentials** parameter:

```
parameter_defaults:
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      [service account username]: [service account password]
    registry.connect.redhat.com:
      [service account username]: [service account password]
```

6. Save the **containers-prepare-parameter-dellemc.yaml** file.

7. Include the **containers-prepare-parameter-dellemc.yaml** file with any deployment commands, such as as **openstack overcloud deploy**:

```
$ openstack overcloud deploy --templates
    ...
    -e containers-prepare-parameter-dellemc.yaml
    ...
```

When director deploys the overcloud, the overcloud uses the vendor container image instead of the standard container image.

IMPORTANT

The **containers-prepare-parameter-dellemc.yaml** file replaces the standard **containers-prepare-parameter.yaml** file in your overcloud deployment. Do not include the standard **containers-prepare-parameter.yaml** file in your overcloud deployment. Retain the standard **containers-prepare-parameter.yaml** file for your undercloud installation and updates.

CHAPTER 8. BASIC NETWORK ISOLATION

Configure the overcloud to use isolated networks so that you can host specific types of network traffic in isolation. Red Hat OpenStack Platform (RHOSP) includes a set of environment files that you can use to configure this network isolation. You might also require additional environment files to further customize your networking parameters:

- An environment file that you can use to enable network isolation (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**).

> NOTE
>
> Before you deploy RHOSP with director, the files **network-isolation.yaml** and **network-environment.yaml** are only in Jinja2 format and have a **.j2.yaml** extension. Director renders these files to **.yaml** versions during deployment.

- An environment file that you can use to configure network defaults (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**).
- A **network_data** file that you can use to define network settings such as IP ranges, subnets, and virtual IPs. This example shows you how to create a copy of the default and edit it to suit your own network.
- Templates that you can use to define your NIC layout for each node. The overcloud core template collection contains a set of defaults for different use cases.
- An environment file that you can use to enable NICs. This example uses a default file located in the **environments** directory.

8.1. NETWORK ISOLATION

The overcloud assigns services to the provisioning network by default. However, director can divide overcloud network traffic into isolated networks. To use isolated networks, the overcloud contains an environment file that enables this feature. The **environments/network-isolation.j2.yaml** file in the core heat templates is a Jinja2 file that defines all ports and VIPs for each network in your composable network file. When rendered, it results in a **network-isolation.yaml** file in the same location with the full resource registry:

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

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

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

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

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

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.

Other functions of overcloud networking, such as Chapter 9, *Custom composable networks* and Chapter 10, *Custom network interface templates* rely on the **network-isolation.yaml** environment file. Therefore you must include the the rendered environment file in your deployment commands:

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

8.2. MODIFYING ISOLATED NETWORK CONFIGURATION

Copy the default **network_data.yaml** file and modify the copy to configure the default isolated networks.

Procedure

1. Copy the default **network_data.yaml** file:

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

2. Edit the local copy of the **network_data.yaml** file and modify the parameters to suit your networking requirements. For example, the Internal API network contains the following default network details:

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

Edit the following values for each network:

- **vlan** defines the VLAN ID that you want to use for this network.
- **ip_subnet** and **ip_allocation_pools** set the default subnet and IP range for the network.
- **gateway** sets the gateway for the network. Use this value to define the default route for the External network, or for other networks if necessary.

Include the custom **network_data.yaml** file with your deployment using the **-n** option. Without the **-n** option, the deployment command uses the default network details.

8.3. NETWORK INTERFACE TEMPLATES

The overcloud network configuration requires a set of the network interface templates. These templates are standard heat templates in YAML format. Each role requires a NIC template so that director can configure each node within that role correctly.

All NIC templates contain the same sections as standard heat templates:

heat_template_version

The syntax version to use.

description

A string description of the template.

parameters

Network parameters to include in the template.

resources

Takes parameters defined in **parameters** and applies them to a network configuration script.

outputs

Renders the final script used for configuration.

The default NIC templates in **/usr/share/openstack-tripleo-heat-templates/network/config** use Jinja2 syntax to render the template. For example, the following snippet from the **single-nic-vlans** configuration renders a set of VLANs for each network:

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

For default Compute nodes, this renders only the network information for the Storage, Internal API, and Tenant networks:

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

Chapter 10, *Custom network interface templates* explores how to render the default Jinja2-based templates to standard YAML versions, which you can use as a basis for customization.

8.4. DEFAULT NETWORK INTERFACE TEMPLATES

Director contains templates in /**usr**/**share**/**openstack-tripleo-heat-templates**/**network**/**config**/ to suit most common network scenarios. The following table outlines each NIC template set and the respective environment file that you must use to enable the templates.

NOTE

Each environment file for enabling NIC templates uses the suffix **.j2.yaml**. This is the unrendered Jinja2 version. Ensure that you include the rendered file name, which uses the **.yaml** suffix, in your deployment.

NIC directory	Description	Environment file
single-nic-vlans	Single NIC (**nic1**) with control plane and VLANs attached to default Open vSwitch bridge.	**environments/net-single-nic-with-vlans.j2.yaml**
single-nic-linux-bridge-vlans	Single NIC (**nic1**) with control plane and VLANs attached to default Linux bridge.	**environments/net-single-nic-linux-bridge-with-vlans**
bond-with-vlans	Control plane attached to **nic1**. Default Open vSwitch bridge with bonded NIC configuration (**nic2** and **nic3**) and VLANs attached.	**environments/net-bond-with-vlans.yaml**

NIC directory	Description	Environment file
multiple-nics	Control plane attached to **nic1**. Assigns each sequential NIC to each network defined in the **network_data.yaml** file. By default, this is Storage to **nic2**, Storage Management to **nic3**, Internal API to **nic4**, Tenant to **nic5** on the **br-tenant** bridge, and External to **nic6** on the default Open vSwitch bridge.	**environments/net-multiple-nics.yaml**

NOTE

Environment files exist for deploying the overcloud without an external network, for example, **net-bond-with-vlans-no-external.yaml**, and for IPv6 deployments, for example, **net-bond-with-vlans-v6.yaml**. These are provided for backwards compatibility and do not function with composable networks.

Each default NIC template set contains a **role.role.j2.yaml** template. This file uses Jinja2 to render additional files for each composable role. For example, if your overcloud uses Compute, Controller, and Ceph Storage roles, the deployment renders new templates based on **role.role.j2.yaml**, such as the following templates:

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

8.5. ENABLING BASIC NETWORK ISOLATION

Director includes templates that you can use to enable basic network isolation. These files are located in the **/usr/share/openstack-tripleo-heat-templates/environments** directoy. For example, you can use the templates to deploy an overcloud on a single NIC with VLANs with basic network isolation. In this scenario, use the **net-single-nic-with-vlans** template.

Procedure

1. When you run the **openstack overcloud deploy** command, ensure that you include the following rendered environment files:
   - The custom **network_data.yaml** file.
   - The rendered file name of the default network isolation file.
   - The rendered file name of the default network environment file.
   - The rendered file name of the default network interface configuration file.
   - Any additional environment files relevant to your configuration.

For example:

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

CHAPTER 9. CUSTOM COMPOSABLE NETWORKS

You can create custom composable networks if you want to host specific network traffic on different networks. To configure the overcloud with an additional composable network, you must configure the following files and templates:

- The environment file to enable network isolation (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**).
- The environment file to configure network defaults (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**).
- A custom **network_data** file to create additional networks outside of the defaults.
- A custom **roles_data** file to assign custom networks to roles.
- Templates to define your NIC layout for each node. The overcloud core template collection contains a set of defaults for different use cases.
- An environment file to enable NICs. This example uses a default file that is located in the **environments** directory.
- Any additional environment files to customize your networking parameters. This example uses an environment file to customize OpenStack service mappings to composable networks.

NOTE

Some of the files in the previous list are Jinja2 format files and have a **.j2.yaml** extension. Director renders these files to **.yaml** versions during deployment.

9.1. COMPOSABLE NETWORKS

The overcloud uses the following pre-defined set of network segments by default:

- Control Plane
- Internal API
- Storage
- Storage Management
- Tenant
- External
- Management (optional)

You can use composable networks to add networks for various services. For example, if you have a network that is dedicated to NFS traffic, you can present it to multiple roles.

Director supports the creation of custom networks during the deployment and update phases. You can use these additional networks for ironic bare metal nodes, system management, or to create separate networks for different roles. You can also use them 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 you want to deploy. Include this file with your deployment command using the **-n** option. Without this option, the deployment uses the default **/usr/share/openstack-tripleo-heat-templates/network_data.yaml** file.

9.2. ADDING A COMPOSABLE NETWORK

Use composable networks to add networks for various services. For example, if you have a network that is dedicated to storage backup traffic, you can present the network to multiple roles.

Procedure

1. Copy the default **network_data.yaml** file:

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

2. Edit the local copy of the **network_data.yaml** file and add a section for your new network:

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 21
  vip: true
  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'
```

You can use the following parameters in your **network_data.yaml** file:

name

Sets the human readable name of the network. This parameter is the only mandatory parameter. You can also use **name_lower** to normalize names for readability. For example, change **InternalApi** to **internal_api**.

name_lower

Sets the lowercase version of the name, which director maps to respective networks assigned to roles in the **roles_data.yaml** file.

vlan

Sets the VLAN that you want to use for this network.

vip: true

Creates a virtual IP address (VIP) on the new network. This IP is used as the target IP for services listed in the service-to-network mapping parameter (**ServiceNetMap**). Note that VIPs are used only by roles that use Pacemaker. The overcloud load-balancing service redirects traffic from these IPs to their respective service endpoint.

ip_subnet

Sets the default IPv4 subnet in CIDR format.

allocation_pools

Sets the IP range for the IPv4 subnet

gateway_ip

Sets the gateway for the network.

routes

Adds additional routes to the network. Uses a JSON list that contains each additional route. Each list item contains a dictionary value mapping. Use the following example syntax:

```
routes: [{'destination':'10.0.0.0/16', 'nexthop':'10.0.2.254'}]
```

subnets

Creates additional routed subnets that fall within this network. This parameter accepts a **dict** value that contains the lowercase name of the routed subnet as the key and the **vlan**, **ip_subnet**, **allocation_pools**, and **gateway_ip** parameters as the value mapped to the subnet. The following example demonstrates this layout:

```
- name: StorageBackup
  name_lower: storage_backup
  vlan: 200
  vip: true
  ip_subnet: '172.21.0.0/24'
  allocation_pools: [{'start': '171.21.0.4', 'end': '172.21.0.250'}]
  gateway_ip: '172.21.0.1'
  subnets:
    storage_backup_leaf1:
      vlan: 201
      ip_subnet: '172.21.1.0/24'
      allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
      gateway_ip: '172.19.1.254'
```

This mapping is common in spine leaf deployments. For more information, see the *Spine Leaf Networking* guide.

Include the custom **network_data.yaml** file in your deployment command using the **-n** option. Without the **-n** option, the deployment command uses the default set of networks.

9.3. INCLUDING A COMPOSABLE NETWORK IN A ROLE

You can assign composable networks to the overcloud roles defined in your environment. For example, you might include a custom **StorageBackup** network with your Ceph Storage nodes.

Procedure

1. If you do not already have a custom **roles_data.yaml** file, copy the default to your home directory:

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

2. Edit the custom **roles_data.yaml** file.

3. Include the network name in the **networks** list for the role that you want to add the network to. For example, to add the **StorageBackup** network to the Ceph Storage role, use the following example snippet:

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

```
- Storage
- StorageMgmt
- StorageBackup
```

4. After you add custom networks to their respective roles, save the file.

When you run the **openstack overcloud deploy** command, include the custom **roles_data.yaml** file using the **-r** option. Without the **-r** option, the deployment command uses the default set of roles with their respective assigned networks.

9.4. ASSIGNING OPENSTACK SERVICES TO COMPOSABLE NETWORKS

Each OpenStack service is assigned to a default network type in the resource registry. These services are 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 can differ as defined in the network environment file. You can reassign OpenStack services to different network types by defining a new network map in an environment file, for example, **/home/stack/templates/service-reassignments.yaml**. The **ServiceNetMap** parameter determines the network types that you want to use for each service.

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

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

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

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

Service mappings apply to networks that use **vip: true** in the **network_data.yaml** file for nodes that use Pacemaker. The overcloud load balancer redirects traffic from the VIPs to the specific service endpoints.

NOTE

You can find a full list of default services in the **ServiceNetMapDefaults** parameter in the **/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** file.

9.5. ENABLING CUSTOM COMPOSABLE NETWORKS

Enable custom composable networks using one of the default NIC templates. In this example, use the Single NIC with VLANs template (**net-single-nic-with-vlans**).

Procedure

1. When you run the **openstack overcloud deploy** command, ensure that you include the following files:
   - The custom **network_data.yaml** file.
   - The custom **roles_data.yaml** file with network-to-role assignments.
   - The rendered file name of the default network isolation.
   - The rendered file name of the default network environment file.
   - The rendered file name of the default network interface configuration.
   - Any additional environment files related to your network, such as the service reassignments.

For example:

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

This example command deploys the composable networks, including your additional custom networks, across nodes in your overcloud.

IMPORTANT

Remember that you must render the templates again if you are introducing a new custom network, such as a management network. Simply adding the network name to the **roles_data.yaml** file is not sufficient.

9.6. RENAMING THE DEFAULT NETWORKS

You can use the **network_data.yaml** file to modify the user-visible names of the default networks:

- InternalApi
- External
- Storage
- StorageMgmt
- Tenant

To change these names, do not modify the **name** field. Instead, change the **name_lower** field to the new name for the network and update the ServiceNetMap with the new name.

Procedure

1. In your **network_data.yaml** file, enter new names in the **name_lower** parameter for each network that you want to rename:

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

2. Include the default value of the **name_lower** parameter in the **service_net_map_replace** parameter:

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

CHAPTER 10. CUSTOM NETWORK INTERFACE TEMPLATES

After you configure Chapter 8, *Basic network isolation*, you can create a set of custom network interface templates to suit the nodes in your environment. For example, you can include the following files:

- The environment file to enable network isolation (**/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml**).
- The environment file to configure network defaults (**/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml**).
- Templates to define your NIC layout for each node. The overcloud core template collection contains a set of defaults for different use cases. To create a custom NIC template, render a default Jinja2 template as the basis for your custom templates.
- A custom environment file to enable NICs. This example uses a custom environment file (**/home/stack/templates/custom-network-configuration.yaml**) that references your custom interface templates.
- Any additional environment files to customize your networking parameters.
- If you customize your networks, a custom **network_data.yaml** file.
- If you create additional or custom composable networks, a custom **network_data.yaml** file and a custom **roles_data.yaml** file.

NOTE

Some of the files in the previous list are Jinja2 format files and have a **.j2.yaml** extension. Director renders these files to **.yaml** versions during deployment.

10.1. CUSTOM NETWORK ARCHITECTURE

The default NIC templates might not suit a specific network configuration. For example, you might want to create your own custom NIC template that suits a specific network layout. You might want to separate the control services and data services on to separate NICs. In this situation, you can map the service to NIC assignments in the following way:

- NIC1 (Provisioning)
  - Provisioning / 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

10.2. RENDERING DEFAULT NETWORK INTERFACE TEMPLATES FOR CUSTOMIZATION

To simplify the configuration of custom interface templates, render the Jinja2 syntax of a default NIC template and use the rendered templates as the basis for your custom configuration.

Procedure

1. Render a copy of the **openstack-tripleo-heat-templates** collection with the **process-templates.py** script:

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

   This converts all Jinja2 templates to their rendered YAML versions and saves the results to **~/openstack-tripleo-heat-templates-rendered**.

   If you use a custom network file or custom roles file, you can include these files using the **-n** and **-r** options respectively:

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

2. Copy the multiple NIC example:

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

3. Edit the template set in **custom-nics** to suit your own network configuration.

10.3. NETWORK INTERFACE ARCHITECTURE

The custom NIC templates that you render in Section 10.2, "Rendering default network interface templates for customization" contain the **parameters** and **resources** 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 use a network environment file to modify the values for some parameters.

Resources

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

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

This snippet runs a script (**run-os-net-config.sh**) that creates a configuration file for **os-net-config** to use to configure 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.

IMPORTANT

If you create custom NIC templates, you must set the **run-os-net-config.sh** script location to an absolute path for each NIC template. The script is located at **/usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh** on the undercloud.

10.4. NETWORK INTERFACE REFERENCE

Network interface configuration contains the following parameters:

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
```

Table 10.1. 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 list of IP addresses assigned to the interface.

Option	Default	Description
routes		A list of routes assigned to the interface. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the interface.
ethtool_opts		Set this option to **"rx-flow-hash udp4 sdfn"** to improve throughput when you use VXLAN on certain NICs.

vlan

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

For example:

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

Table 10.2. vlan options

Option	Default	Description
vlan_id		The VLAN ID.

Option	Default	Description
device		The parent device to attach the VLAN. Use this parameter when the VLAN is not a member of an OVS bridge. 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 list of IP addresses assigned to the VLAN.
routes		A list of routes assigned to the VLAN. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the VLAN as the primary interface.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the VLAN.

ovs_bond

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

For example:

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

Table 10.3. 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 list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. For more information, see routes.
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 that you want 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 set as the OVS_EXTRA parameter in the network configuration file of the bond.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.

Option	Default	Description
dns_servers	None	List of DNS servers that you want to use for the bond.

ovs_bridge

Defines a bridge in Open vSwitch, which connects multiple **interface**, **ovs_bond**, and **vlan** objects together.

The network interface type, **ovs_bridge**, takes a parameter **name**.

NOTE

If you have multiple bridges, you must use distinct bridge names other than accepting the default name of **bridge_name**. If you do not use distinct names, then during the converge phase, two network bonds are placed on the same bridge.

If you are defining an OVS bridge for the external tripleo network, then retain the values **bridge_name** and **interface_name** as your deployment framework automatically replaces these values with an external bridge name and an external interface name, respectively.

For example:

```
- 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 Networking service (neutron) server to obtain 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 is lost whenever you upgrade OVS, or the OVS bridge is restarted by the admin user or process. This causes some downtime. If downtime is not acceptable in these circumstances, then you must place the Control group networks on a separate interface or bond rather than on an OVS bridge:

- You can achieve a minimal setting when you put the Internal API network on a VLAN on the provisioning interface and the OVS bridge on a second interface.
- To implement bonding, you need at least two bonds (four network interfaces). Place the control group 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.

Table 10.4. 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 list of IP addresses assigned to the bridge.
routes		A list of routes assigned to the bridge. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
members		A sequence of interface, VLAN, and bond objects that you want 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 network configuration file of the bridge.

Option	Default	Description
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bridge.

linux_bond

Defines a Linux bond that joins two or more **interfaces** together. This helps with redundancy and increases bandwidth. Ensure that you include the kernel-based bonding options in the **bonding_options** parameter.

For example:

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

Note that **nic2** uses **primary: true** to ensure that the bond uses the MAC address for **nic2**.

Table 10.5. 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.

Option	Default	Description
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. See routes.
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 that you want to use in the bond.
bonding_options		A set of options when creating the bond.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.
dns_servers	None	List of DNS servers that you want to use for the bond.

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.

For example:

```
- 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}
```

Table 10.6. 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 list of IP addresses assigned to the bridge.
routes		A list of routes assigned to the bridge. For more information, see routes.
mtu	1500	The maximum transmission unit (MTU) of the connection.
members		A sequence of interface, VLAN, and bond objects that you want to use in the bridge.
defroute	True	Use a default route provided by the DHCP service. Only applies when you enable **use_dhcp** or **use_dhcpv6**.
persist_mapping	False	Write the device alias configuration instead of the system names.
dhclient_args	None	Arguments that you want to pass to the DHCP client.

Option	Default	Description
dns_servers	None	List of DNS servers that you want to use for the bridge.

routes

Defines a list of routes to apply to a network interface, VLAN, bridge, or bond.

For example:

```
    - type: interface
      name: nic2
      ...
      routes:
        - ip_netmask: 10.1.2.0/24
          gateway_ip: 10.1.2.1
```

Option	Default	Description
ip_netmask	None	IP and netmask of the destination network.
default	False	Sets this route to a default route. Equivalent to setting **ip_netmask: 0.0.0.0/0**.
next_hop	None	The IP address of the router used to reach the destination network.

10.5. EXAMPLE NETWORK INTERFACE LAYOUT

The following snippet for an example Controller node NIC template demonstrates how to configure the custom network scenario to keep the control group separate from the OVS bridge:

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
              - type: interface
                name: nic1
                mtu:
                 get_param: ControlPlaneMtu
  use_dhcp: false
  addresses:
  - ip_netmask:
      list_join:
      - /
      - - get_param: ControlPlaneIp
        - get_param: ControlPlaneSubnetCidr
  routes:
    list_concat_unique:
      - get_param: ControlPlaneStaticRoutes
- type: ovs_bridge
  name: bridge_name
  dns_servers:
    get_param: DnsServers
  domain:
    get_param: DnsSearchDomains
  members:
  - type: ovs_bond
    name: bond1
    mtu:
      get_attr: [MinViableMtu, value]
    ovs_options:
      get_param: BondInterfaceOvsOptions
    members:
      - type: interface
        name: nic2
        mtu:
          get_attr: [MinViableMtu, value]
        primary: true
      - type: interface
        name: nic3
        mtu:
          get_attr: [MinViableMtu, value]
  - type: vlan
    mtu:
      get_param: StorageMtu
    vlan_id:
      get_param: StorageNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: StorageIpSubnet
    routes:
      list_concat_unique:
        - get_param: StorageInterfaceRoutes
  - type: vlan
    mtu:
      get_param: StorageMgmtMtu
    vlan_id:
      get_param: StorageMgmtNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: StorageMgmtIpSubnet
    routes:
      list_concat_unique:
        - get_param: StorageMgmtInterfaceRoutes
  - type: vlan
```

```
            mtu:
             get_param: InternalApiMtu
            vlan_id:
              get_param: InternalApiNetworkVlanID
            addresses:
            - ip_netmask:
                get_param: InternalApiIpSubnet
            routes:
              list_concat_unique:
                - get_param: InternalApiInterfaceRoutes
           - type: vlan
             mtu:
               get_param: TenantMtu
             vlan_id:
               get_param: TenantNetworkVlanID
             addresses:
             - ip_netmask:
                 get_param: TenantIpSubnet
            routes:
              list_concat_unique:
                - get_param: TenantInterfaceRoutes
           - type: vlan
             mtu:
               get_param: ExternalMtu
             vlan_id:
               get_param: ExternalNetworkVlanID
             addresses:
             - ip_netmask:
                 get_param: ExternalIpSubnet
            routes:
              list_concat_unique:
                - get_param: ExternalInterfaceRoutes
                - - default: true
                    next_hop:
                      get_param: ExternalInterfaceDefaultRoute
```

This template uses three network interfaces and assigns a number of tagged VLAN devices to the numbered interfaces, **nic1** to **nic3**. On **nic2** and **nic3** this template creates the OVS bridge that hosts the Storage, Tenant, and External networks. As a result, it creates the following layout:

- NIC1 (Provisioning)
  - Provisioning / Control Plane
- NIC2 and NIC3 (Management)
  - Internal API
  - Storage
  - Storage Management
  - Tenant Network (VXLAN tunneling)
  - Tenant VLANs / Provider VLANs
  - External (Public API)

- External VLANs (Floating IP/SNAT)

10.6. NETWORK INTERFACE TEMPLATE CONSIDERATIONS FOR CUSTOM NETWORKS

When you use composable networks, the **process-templates.py** script renders the static templates to include networks and roles that you define in your **network_data.yaml** and **roles_data.yaml** files. Ensure that your rendered NIC templates contain the following items:

- A static file for each role, including custom composable networks.
- The correct network definitions in the static file for each role.

Each static file requires all of the parameter definitions for any custom networks, even if the network is not used on the role. Ensure that the rendered templates contain these parameters. For example, if you add a **StorageBackup** network only to the Ceph nodes, you must also include this definition in the **parameters** section in the NIC configuration templates for all roles:

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

You can also include the **parameters** definitions for VLAN IDs and/or gateway IP, if necessary:

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

The **IpSubnet** parameter for the custom network appears in the parameter definitions for each role. However, since the Ceph role might be the only role that uses the **StorageBackup** network, only the NIC configuration template for the Ceph role uses 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
```

10.7. CUSTOM NETWORK ENVIRONMENT FILE

The custom network environment file (in this case, **/home/stack/templates/custom-network-configuration.yaml**) is a heat environment file that describes the overcloud network environment and points to the custom network interface configuration templates. 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 **resource_registry** section contains references to the custom network interface templates for each node role. Each resource registered uses the following format:

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

[ROLE] is the role name and **[FILE]** is the respective network interface template for that particular role. For example:

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

The **parameter_defaults** section contains a list of parameters that define the network options for each network type.

10.8. NETWORK ENVIRONMENT PARAMETERS

The following table is a list of parameters that you can use in the **parameter_defaults** section of a network environment file to override the default parameter values in your NIC templates.

Parameter	Description	Type
ControlPlaneDefaultRoute	The IP address of the router on the Control Plane, which is used as a default route for roles other than the Controller nodes. Set this value to the undercloud IP if you use IP masquerade instead of a router.	string
ControlPlaneSubnetCidr	The CIDR netmask of the IP network used on the Control Plane. If the Control Plane network uses 192.168.24.0/24, the CIDR is **24**.	string (though is always a number)
***NetCidr**	The full network and CIDR netmask for a particular network. The default is automatically set to the network **ip_subnet** setting in the **network_data.yaml** file. For example, **InternalApiNetCidr: 172.16.0.0/24**.	string

Parameter	Description	Type
***AllocationPools**	The IP allocation range for a particular network. The default is automatically set to the network **allocation_pools** setting in the **network_data.yaml** file. For example, **InternalApiAllocationPools: [{'start': '172.16.0.10', 'end': '172.16.0.200'}]**.	hash
***NetworkVlanID**	The VLAN ID for a node on a particular network. The default is set automatically to the network **vlan** setting in the **network_data.yaml** file. For example, **InternalApiNetworkVlanID: 201**.	number
***InterfaceDefaultRoute**	The router address for a particular network, which you can use as a default route for roles or for routes to other networks. The default is automatically set to the network **gateway_ip** setting in the **network_data.yaml** file. For example, **InternalApiInterfaceDefaultRoute: 172.16.0.1**.	string
DnsServers	A list of DNS servers added to resolv.conf. Usually allows a maximum of 2 servers.	comma delimited list
EC2MetadataIp	The IP address of the metadata server used to provision overcloud nodes. Set this value to the IP address of the undercloud on the Control Plane.	string
BondInterfaceOvsOptions	The options for bonding interfaces. For example, **BondInterfaceOvsOptions: "bond_mode=balance-slb"**.	string

Parameter	Description	Type
NeutronExternalNetworkBridge	Legacy value for the name of the external bridge that you want to use for OpenStack Networking (neutron). This value is empty by default, which means that you can define multiple physical bridges in the **NeutronBridgeMappings**. In normal circumstances, do not override this value.	string
NeutronFlatNetworks	Defines the flat networks that you want to configure in neutron plugins. The default value is **datacentre** to permit external network creation. For example, **NeutronFlatNetworks: "datacentre"**.	string
NeutronBridgeMappings	The logical to physical bridge mappings that you want to use. The default value maps the external bridge on hosts (**br-ex**) to a physical name (**datacentre**). Refer to the logical name when you create OpenStack Networking (neutron) provider networks or floating IP networks. For example **NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"**.	string
NeutronPublicInterface	Defines the interface that you want to bridge onto **br-ex** for network nodes when you do not use network isolation. Usually not used except in small deployments with only two networks. For example: **NeutronPublicInterface: "eth0"**.	string

Parameter	Description	Type
NeutronNetworkType	The tenant network type for OpenStack Networking (neutron). To specify multiple values, use a comma separated list. The first type that you specify is used until all available networks are exhausted, then the next type is used. For example, **NeutronNetworkType: "vxlan"**. Note that vxlan is not supported by the ML2/OVN mechanism driver, which is the default ML2 mechanism driver.	string
NeutronTunnelTypes	The tunnel types for the neutron tenant network. To specify multiple values, use a comma separated string. For example, **NeutronTunnelTypes: 'gre,vxlan'**. Note that vxlan is not supported by the ML2/OVN mechanism driver, which is the default ML2 mechanism driver.	string / comma separated list
NeutronTunnelIdRanges	Ranges of GRE tunnel IDs that you want to make available for tenant network allocation. For example, **NeutronTunnelIdRanges "1:1000"**.	string
NeutronVniRanges	Ranges of VXLAN VNI IDs that you want to make available for tenant network allocation. For example, **NeutronVniRanges: "1:1000"**.	string
NeutronEnableTunnelling	Defines whether to enable or completely disable all tunnelled networks. Leave this enabled unless you are sure that you do not want to create tunelled networks in future. The default value is **true**.	Boolean

Parameter	Description	Type
NeutronNetworkVLANRanges	The ML2 and Open vSwitch VLAN mapping range that you want to support. Defaults to permitting any VLAN on the **datacentre** physical network. To specify multiple values, use a comma separated list. For example, **NeutronNetworkVLANRanges: "datacentre:1:1000,tenant:100:299,tenant:310:399"**.	string
NeutronMechanismDrivers	The mechanism drivers for the neutron tenant network. The default value is **ovn**. To specify multiple values, use a comma-separated string. For example, **NeutronMechanismDrivers: 'openvswitch,l2population'**.	string / comma separated list

10.9. EXAMPLE CUSTOM NETWORK ENVIRONMENT FILE

The following snippet is an example of an environment file that you can use to enable your NIC templates and set custom parameters.

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

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

10.10. ENABLING NETWORK ISOLATION WITH CUSTOM NICS

To deploy the overcloud with network isolation and custom NIC templates, include all of the relevant networking environment files in the overcloud deployment command.

Procedure

1. When you run the **openstack overcloud deploy** command, include the following files:
   - The custom **network_data.yaml** file.
   - The rendered file name of the default network isolation.
   - The rendered file name of the default network environment file.
   - The custom environment network configuration that includes resource references to your custom NIC templates.
   - Any additional environment files relevant to your configuration.

For example:

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

- Include the **network-isolation.yaml** file first, then the **network-environment.yaml** file. The subsequent **custom-network-configuration.yaml** overrides the **OS::TripleO::[ROLE]::Net::SoftwareConfig** resources from the previous two files.
- If you use composable networks, include the **network_data.yaml** and **roles_data.yaml** files with this command.

CHAPTER 11. ADDITIONAL NETWORK CONFIGURATION

This chapter follows on from the concepts and procedures outlined in *Chapter 10, Custom network interface templates* and provides some additional information to help configure parts of your overcloud network.

11.1. CONFIGURING CUSTOM INTERFACES

Individual interfaces might require modification. The following example shows the modifications that are necessary to use a second NIC to connect to an infrastructure network with DHCP addresses, and to use a third and fourth NIC 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 you use 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 the NICs of both hosts 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 includes only live interfaces, for example, if the interfaces have a cable attached to the switch. If you have some hosts with four interfaces and some with six interfaces, use **nic1** to **nic4** and attach only four cables on each host.

You can hardcode physical interfaces to specific aliases so that you can pre-determine which physical NIC maps as **nic1** or **nic2** and so on. You can also map a MAC address to a specified alias.

NOTE

Normally, **os-net-config** registers only the interfaces that are already connected in an **UP** state. However, if you hardcode interfaces with a custom mapping file, the interface is registered even if it is in a **DOWN** state.

Interfaces are mapped to aliases with an environment file. In this example, each node has predefined entries for **nic1** and **nic2**.

NOTE

If you want to use the **NetConfigDataLookup** configuration, you must also include the **os-net-config-mappings.yaml** file in the **NodeUserData** resource registry.

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

The resulting configuration is applied by **os-net-config**. On each node, you can see the applied configuration in the **interface_mapping** section of the **/etc/os-net-config/mapping.yaml** file.

NOTE

The **NetConfigDataLookup** parameter is not applied during a deployment onto pre-provisioned nodes. If you want to use custom interface mappings with pre-provisioned nodes, you must create the **/etc/os-net-config/mapping.yaml** file on each node before the deployment. Use the following example interface mapping in the **/etc/os-net-config/mapping.yaml** file:

```
interface_mapping:
  nic1: em1
  nic2: em2
```

11.2. CONFIGURING ROUTES AND DEFAULT ROUTES

You can set the default route of a host in one of two ways. If the interface uses 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 uses only the gateway 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: false** for interfaces other than the interface that uses the default route.

For example, you might want a DHCP interface (**nic3**) to be the default route. Use the following YAML snippet 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 applies only 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
```

11.3. CONFIGURING POLICY-BASED ROUTING

On Controller nodes, to configure unlimited access from different networks, configure policy-based routing. Policy-based routing uses route tables where, on a host with multiple interfaces, you can send traffic through a particular interface depending on the source address. You can route packets that come from different sources to different networks, even if the destinations are the same.

For example, you can configure a route to send traffic to the Internal API network, based on the source address of the packet, even when the default route is for the External network. You can also define specific route rules for each interface.

Red Hat OpenStack Platform uses the **os-net-config** tool to configure network properties for your overcloud nodes. The **os-net-config** tool manages the following network routing on Controller nodes:

- Routing tables in the **/etc/iproute2/rt_tables** file
- IPv4 rules in the **/etc/sysconfig/network-scripts/rule-{ifname}** file
- IPv6 rules in the **/etc/sysconfig/network-scripts/rule6-{ifname}** file
- Routing table specific routes in the **/etc/sysconfig/network-scripts/route-{ifname}**

Prerequisites

- You have installed the undercloud successfully. For more information, see Installing director in the *Director Installation and Usage* guide.
- You have rendered the default **.j2** network interface templates from the **openstack-tripleo-heat-templates** directory. For more information, see Section 10.2, “Rendering default network interface templates for customization”.

Procedure

1. Create **route_table** and **interface** entries in a custom NIC template from the **~/templates/custom-nics** directory, define a route for the interface, and define rules that are relevant to your deployment:

```
$network_config:
  network_config:

  - type: route_table
    name: custom
    table_id: 200

  - type: interface
    name: em1
    use_dhcp: false
    addresses:
      - ip_netmask: 192.0.2.1/24

    routes:
      - ip_netmask: 10.1.3.0/24
        next_hop: 192.0.2.5
        route_options: "metric 10"
        table: 200
    rules:
      - rule: "iif em1 table 200"
        comment: "Route incoming traffic to em1 with table 200"
      - rule: "from 192.0.2.0/24 table 200"
        comment: "Route all traffic from 192.0.2.0/24 with table 200"
      - rule: "add blackhole from 172.19.40.0/24 table 200"
      - rule: "add unreachable iif em1 from 192.168.1.0/24"
```

2. Set the **run-os-net-config.sh** script location to an absolute path in each custom NIC template that you create. The script is located in the **/usr/share/openstack-tripleo-heat-templates/network/scripts/** directory on the undercloud:

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

3. Include your custom NIC configuration and network environment files in the deployment command, along with any other environment files relevant to your deployment:

```
$ openstack overcloud deploy --templates \
-e ~/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

Verification

- Enter the following commands on a Controller node to verify that the routing configuration is functioning correctly:

```
$ cat /etc/iproute2/rt_tables
$ ip route
$ ip rule
```

11.4. 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 because 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. Ensure that you include the MTU value on the bond or interface.

The Storage, Storage Management, Internal API, and Tenant networks all benefit from jumbo frames.

WARNING

Routers typically cannot forward jumbo frames across Layer 3 boundaries. To avoid connectivity issues, do not change the default MTU for the Provisioning interface, External interface, and any floating IP interfaces.

```
- type: ovs_bond
  name: bond1
  mtu:
    get_param: [MaxViableMtu, value]
  ovs_options:
    get_param: BondInterfaceOvsOptions
  members:
    - type: interface
      name: nic2
      mtu: 9000
      primary: true
    - type: interface
      name: nic3
      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:
    list_concat_unique
      - get_param: ExternalInterfaceRoutes
      - - default: true
          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
```

11.5. CONFIGURING ML2/OVN NORTHBOUND PATH MTU DISCOVERY FOR JUMBO FRAME FRAGMENTATION

If a VM on your internal network sends jumbo frames to an external network, and the maximum transmission unit (MTU) of the internal network exceeds the MTU of the external network, a northbound frame can easily exceed the capacity of the external network.

ML2/OVS automatically handles this oversized packet issue, and ML2/OVN handles it automatically for TCP packets.

But to ensure proper handling of oversized northbound UDP packets in a deployment that uses the ML2/OVN mechanism driver, you need to perform additional configuration steps.

These steps configure ML2/OVN routers to return ICMP "fragmentation needed" packets to the sending VM, where the sending application can break the payload into smaller packets.

NOTE

In east/west traffic, a RHOSP ML2/OVN deployment does not support fragmentation of packets that are larger than the smallest MTU on the east/west path. For example:

- VM1 is on Network1 with an MTU of 1300.
- VM2 is on Network2 with an MTU of 1200.
- A ping in either direction between VM1 and VM2 with a size of 1171 or less succeeds. A ping with a size greater than 1171 results in 100 percent packet loss. With no identified customer requirements for this type of fragmentation, Red Hat has no plans to add support.

Prerequisites

- RHEL 8.2.0.4 or later with kernel-4.18.0-193.20.1.el8_2 or later.

Procedure

1. Check the kernel version.

   ```
   ovs-appctl -t ovs-vswitchd dpif/show-dp-features br-int
   ```

2. If the output includes **Check pkt length action: No**, or if there is no **Check pkt length action** string in the output, upgrade to RHEL 8.2.0.4 or later, or do not send jumbo frames to an external network that has a smaller MTU.

3. If the output includes **Check pkt length action: Yes**, set the following value in the [ovn] section of ml2_conf.ini.

   ```
   ovn_emit_need_to_frag = True
   ```

11.6. CONFIGURING 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 is 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:
      list_concat_unique:
        - get_param: ExternalInterfaceRoutes
        - - default: true
            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 you move the address or 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 is attached to all roles, so if the Storage network is on the default VLAN, all roles require modifications.

11.7. INCREASING THE MAXIMUM NUMBER OF CONNECTIONS THAT NETFILTER TRACKS

The Red Hat OpenStack Platform (RHOSP) Networking service (neutron) uses netfilter connection tracking to build stateful firewalls and to provide network address translation (NAT) on virtual networks. There are some situations that can cause the kernel space to reach the maximum connection limit and result in errors such as **nf_conntrack: table full, dropping packet.** You can increase the limit for connection tracking (conntrack) and avoid these types of errors. You can increase the conntrack limit for one or more roles, or across all the nodes, in your RHOSP deployment.

Prerequisites

- A successful RHOSP undercloud installation.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Create a custom YAML environment file.

Example

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

4. Your environment file must contain the keywords **parameter_defaults** and **ExtraSysctlSettings**. Enter a new value for the maximum number of connections that netfilter can track in the variable, **net.nf_conntrack_max**.

Example

In this example, you can set the conntrack limit across all hosts in your RHOSP deployment:

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

Use the **<role>Parameter** parameter to set the conntrack limit for a specific role:

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

- Replace **<role>** with the name of the role.
  For example, use **ControllerParameters** to set the conntrack limit for the Controller role, or **ComputeParameters** to set the conntrack limit for the Compute role.
- Replace **<simultaneous_connections>** with the quantity of simultaneous connections that you want to allow.

  Example

  In this example, you can set the conntrack limit for only the Controller role in your RHOSP deployment:

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

  NOTE

  The default value for **net.nf_conntrack_max** is **500000** connections. The maximum value is: **4294967295**.

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   IMPORTANT

   The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   Example

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

Additional resources

- Environment files
- Including environment files in overcloud creation

CHAPTER 12. NETWORK INTERFACE BONDING

You can use various bonding options in your custom network configuration.

12.1. NETWORK INTERFACE BONDING FOR OVERCLOUD NODES

You can bundle multiple physical NICs together to form a single logical channel known as a bond. You can configure bonds to provide redundancy for high availability systems or increased throughput.

Red Hat OpenStack Platform supports Open vSwitch (OVS) kernel bonds, OVS-DPDK bonds, and Linux kernel bonds.

Table 12.1. Supported interface bonding types

Bond type	Type value	Allowed bridge types	Allowed members
OVS kernel bonds	**ovs_bond**	**ovs_bridge**	**interface**
OVS-DPDK bonds	**ovs_dpdk_bond**	**ovs_user_bridge**	**ovs_dpdk_port**
Linux kernel bonds	**linux_bond**	**ovs_bridge** or **linux_bridge**	**interface**

IMPORTANT

Do not combine **ovs_bridge** and **ovs_user_bridge** on the same node.

12.2. CREATING OPEN VSWITCH (OVS) BONDS

You create OVS bonds in your network interface templates. For example, you can create a bond as part of an OVS user space bridge:

```
...
          params:
            $network_config:
              network_config:
              - type: ovs_user_bridge
                name: br-ex
                use_dhcp: false
                members:
                - type: ovs_dpdk_bond
                  name: dpdkbond0
                  mtu: 2140
                  ovs_options: {get_param: BondInterfaceOvsOptions}
                  rx_queue:
                    get_param: NumDpdkInterfaceRxQueues
                  members:
                  - type: ovs_dpdk_port
                    name: dpdk0
                    mtu: 2140
                    members:
                    - type: interface
                  name: p1p1
                - type: ovs_dpdk_port
                  name: dpdk1
                  mtu: 2140
                  members:
                  - type: interface
                    name: p1p2
```

In this example, you create the bond from two DPDK ports.

The **ovs_options** parameter contains the bonding options. You can configure a bonding options in a network environment file with the **BondInterfaceOvsOptions** parameter:

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

12.3. OPEN VSWITCH (OVS) BONDING OPTIONS

You can set various Open vSwitch (OVS) bonding options with the **ovs_options** heat parameter in your NIC template files.

bond_mode=balance-slb

Source load balancing (slb) balances flows based on source MAC address and output VLAN, with periodic rebalancing as traffic patterns change. When you configure a bond with the **balance-slb** bonding option, there is no configuration required on the remote switch. The Networking service (neutron) assigns each source MAC and VLAN pair to a link and transmits all packets from that MAC and VLAN through that link. A simple hashing algorithm based on source MAC address and VLAN number is used, with periodic rebalancing as traffic patterns change. The **balance-slb** mode is similar to mode 2 bonds used by the Linux bonding driver. You can use this mode to provide load balancing even when the switch is not configured to use LACP.

bond_mode=active-backup

When you configure a bond using **active-backup** bond mode, the Networking service keeps one NIC in standby. 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 switch 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

Set active-backup as the bond mode if LACP fails.

other_config:lacp-time=[fast | slow]

Set the LACP heartbeat to one 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 (milliseconds).

bond_updelay=1000

Set the interval (milliseconds) that a link must be up to be activated to prevent flapping.

other_config:bond-rebalance-interval=10000

Set the interval (milliseconds) that flows are rebalancing between bond members. Set this value to zero to disable flow rebalancing between bond members.

12.4. USING LINK AGGREGATION CONTROL PROTOCOL (LACP) WITH OPEN VSWITCH (OVS) BONDING MODES

You can use bonds with the optional Link Aggregation Control Protocol (LACP). LACP is a negotiation protocol that creates a dynamic bond for load balancing and fault tolerance.

Use the following table to understand support compatibility for OVS kernel and OVS-DPDK bonded interfaces in conjunction with LACP options.

IMPORTANT

The OVS/OVS-DPDK **balance-tcp** mode is available as a technology preview only.

IMPORTANT

On control and storage networks, Red Hat recommends that you use Linux bonds with VLAN and LACP, because OVS bonds carry the potential for control plane disruption that can occur when OVS or the neutron agent is restarted for updates, hot fixes, and other events. The Linux bond/LACP/VLAN configuration provides NIC management without the OVS disruption potential.

Table 12.2. LACP options for OVS kernel and OVS-DPDK bond modes

Objective	OVS bond mode	Compatible LACP options	Notes
High availability (active-passive)	**active-backup**	**active**, **passive**, or **off**	
Increased throughput (active-active)	**balance-slb**	**active**, **passive**, or **off**	<ul><li>Performance is affected by extra parsing per packet.</li><li>There is a potential for vhost-user lock contention.</li></ul>

	balance-tcp	**active** or **passive**	<ul><li>**Tech preview only**. Not recommended for use in production.</li><li>Recirculation needed for L4 hashing has a performance impact.</li><li>As with balance-slb, performance is affected by extra parsing per packet and there is a potential for vhost-user lock contention.</li><li>LACP must be enabled.</li></ul>

12.5. CREATING LINUX BONDS

You create linux bonds in your network interface templates. For example, you can create a linux bond that bond two interfaces:

```
...
          params:
            $network_config:
              network_config:
              - 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"
```

The **bonding_options** parameter sets the specific bonding options for the Linux bond.

mode

Sets the bonding mode, which in the example is **802.3ad** or LACP mode. For more information about Linux bonding modes, see "Upstream Switch Configuration Depending on the Bonding Modes" in the Red Hat Enterprise Linux 8 Configuring and Managing Networking guide.

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 minimum configuration helps to mitigate port flapping outages.

miimon

The interval in milliseconds that is used for monitoring the port state using the MIIMON functionality of the driver.

Use the following additional examples as guides to configure your own Linux bonds:

- Linux bond set to **active-backup** mode with one VLAN:

```
....
          params:
            $network_config:
              network_config:
              - type: linux_bond
                name: bond_api
                bonding_options: "mode=active-backup"
                use_dhcp: false
                dns_servers:
                  get_param: DnsServers
                members:
                - type: interface
                  name: nic3
                  primary: true
                - type: interface
                  name: nic4

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

- Linux bond on OVS bridge. Bond set to **802.3ad** LACP mode with one VLAN:

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

CHAPTER 13. CONTROLLING NODE PLACEMENT

By default, director selects nodes for each role randomly, usually according to the profile tag of the node. However, you can also define specific node placement. This is useful in the following scenarios:

- Assign specific node IDs, for example, **controller-0**, **controller-1**
- Assign custom host names
- 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. Ensure that any statically defined IP addresses fall outside the allocation pools.

13.1. ASSIGNING SPECIFIC NODE IDS

You can assign node IDs to specific nodes, for example, **controller-0**, **controller-1**, **compute-0**, and **compute-1**.

Procedure

1. Assign the ID as a per-node capability that the Compute scheduler matches on deployment:

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

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

2. Create a heat environment file (for example, **scheduler_hints_env.yaml**) that uses scheduler hints to match the capabilities for each node:

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

   Use the following parameters to configure scheduler hints for other role types:

   - **ControllerSchedulerHints** for Controller nodes.
   - **ComputeSchedulerHints** 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.

3. Include the **scheduler_hints_env.yaml** environment file in the **overcloud deploy command**.

NOTE

Node placement takes priority over profile matching. To avoid scheduling failures, use the default **baremetal** flavor for deployment and not the flavors that are designed for profile matching (**compute**, **control**):. Set the respective flavor parameters to baremetal in an environment file:

```
parameter_defaults:
  OvercloudControllerFlavor: baremetal
  OvercloudComputeFlavor: baremetal
```

13.2. ASSIGNING CUSTOM HOST NAMES

In combination with the node ID configuration in Section 13.1, “Assigning specific node IDs” , director can also assign a specific custom host name to each node. This is useful when you need to define where a system is located (for example, **rack2-row12**), match an inventory identifier, or other situations where a custom hostname is desirable.

IMPORTANT

Do not rename a node after it has been deployed. Renaming a node after deployment creates issues with instance management.

Procedure

- Use the **HostnameMap** parameter in an environment file, such as the **scheduler_hints_env.yaml** file from Section 13.1, “Assigning specific node IDs” :

```
parameter_defaults:
  ControllerSchedulerHints:
    'capabilities:node': 'controller-%index%'
  ComputeSchedulerHints:
    'capabilities:node': 'compute-%index%'
  HostnameMap:
    overcloud-controller-0: overcloud-controller-prod-123-0
    overcloud-controller-1: overcloud-controller-prod-456-0
    overcloud-controller-2: overcloud-controller-prod-789-0
    overcloud-novacompute-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 with **HostnameFormat** parameters (for example, **overcloud-controller-0**) and the second value is the desired custom hostname for that node (**overcloud-controller-prod-123-0**).

Use this method in combination with the node ID placement to ensure that each node has a custom hostname.

13.3. ASSIGNING PREDICTABLE IPS

For further control over the resulting environment, director can assign overcloud nodes with specific IP addresses on each network.

Procedure

1. Create an environment file to define the predictive IP addressing:

   ```
   $ touch ~/templates/predictive_ips.yaml
   ```

2. Create a **parameter_defaults** section in the **~/templates/predictive_ips.yaml** file and use the following syntax to define predictive IP addressing for each node on each network:

   ```
   parameter_defaults:
     <role_name>IPs:
       <network>:
       - <IP_address>
       <network>:
       - <IP_address>
   ```

   Each node role has a unique parameter. Replace **<role_name>IPs** with the relevant parameter:

   - **ControllerIPs** for Controller nodes.
   - **ComputeIPs** 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. 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, use the following example syntax if you want to deploy three Ceph Storage nodes in your overcloud with predictive IP addresses:

     ```
     parameter_defaults:
       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.

To configure predictable IP addresses on the control plane, copy the **/usr/share/openstack-tripleo-heat-templates/environments/ips-from-pool-ctlplane.yaml** file to the **templates** directory of the **stack** user:

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

Configure the new **ips-from-pool-ctlplane.yaml** file with the following parameter example. You can combine the control plane IP address declarations with the IP address declarations for other networks and use only one file to declare the IP addresses for all networks on all roles. You can also use predictable IP addresses for spine/leaf. Each node must have IP addresses from the correct subnet.

```
parameter_defaults:
  ControllerIPs:
    ctlplane:
    - 192.168.24.10
    - 192.168.24.11
    - 192.168.24.12
    internal_api:
    - 172.16.1.20
    - 172.16.1.21
    - 172.16.1.22
    external:
    - 10.0.0.40
    - 10.0.0.57
    - 10.0.0.104
  ComputeLeaf1IPs:
    ctlplane:
    - 192.168.25.100
    - 192.168.25.101
    internal_api:
    - 172.16.2.100
    - 172.16.2.101
  ComputeLeaf2IPs:
    ctlplane:
    - 192.168.26.100
    - 192.168.26.101
    internal_api:
    - 172.16.3.100
    - 172.16.3.101
```

Ensure that the IP addresses that you choose fall outside the allocation pools for each network that you define in your network environment file. For example, ensure that the **internal_api** assignments fall outside of the **InternalApiAllocationPools** range to avoid conflicts with any IPs chosen automatically. Also ensure that the IP assignments do not conflict with the VIP configuration, either for standard predictable VIP placement (see Section 13.4, “Assigning predictable Virtual IPs”) or external load balancing (see Section 21.4, “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.

3. To apply this configuration during a deployment, include the **predictive_ips.yaml** environment file with the **openstack overcloud deploy** command.

IMPORTANT

If you use network isolation, include the **predictive_ips.yaml** file after the **network-isolation.yaml** file:

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

13.4. ASSIGNING PREDICTABLE VIRTUAL IPS

In addition to defining predictable IP addresses for each node, you can also define predictable Virtual IPs (VIPs) for clustered services.

Procedure

- Edit the 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'}]
  OVNDBsVirtualFixedIPs: [{'ip_address':'172.16.0.7'}]
```

  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.

NOTE

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

CHAPTER 14. ENABLING SSL/TLS ON OVERCLOUD PUBLIC ENDPOINTS

By default, the overcloud uses unencrypted endpoints for the overcloud services. To enable SSL/TLS in your overcloud, Red Hat recommends that you use a certificate authority (CA) solution.

When you use a certificate authority (CA) solution, you have production ready solutions such as a certificate renewals, certificate revocation lists (CRLs), and industry accepted cryptography. For information on using Red Hat Identity Manager (IdM) as a CA, see Implementing TLS-e with Ansible .

You can use the following manual process to enable SSL/TLS for Public API endpoints only, the Internal and Admin APIs remain unencrypted. You must also manually update SSL/TLS certificates if you do not use a CA. For more information, see Manually updating SSL/TLS certificates .

Prerequisites

- Network isolation to define the endpoints for the Public API.
- The **openssl-perl** package is installed.
- You have an SSL/TLS certificate. For more information see Configuring custom SSL/TLS certificates.

14.1. INITIALIZING THE SIGNING HOST

The signing host is the host that generates and signs new certificates 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.

Procedure

1. The **/etc/pki/CA/index.txt** file contains records of all signed certificates. Ensure that the filesystem path and **index.txt** file are present:

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. 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 the file does not exist, create a new file with a new starting value:

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

14.2. CREATING A CERTIFICATE AUTHORITY

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

Procedure

1. 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
```

2. The **openssl req** command requests certain details about your authority. Enter these details at the prompt. These commands create a certificate authority file called **ca.crt.pem**.

3. Set the certificate location as the value for the **PublicTLSCAFile** parameter in the **enable-tls.yaml** file. When you set the certificate location as the value for the **PublicTLSCAFile** parameter, you ensure that the CA certificate path is added to the **clouds.yaml** authentication file.

```
parameter_defaults:
    PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

14.3. 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 to your Red Hat OpenStack Platform environment.

Procedure

1. Copy the certificate authority to the client system:

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

2. After you copy the certificate authority file to each client, run the following command on each client to add the certificate to the certificate authority trust bundle:

```
$ sudo update-ca-trust extract
```

14.4. CREATING AN SSL/TLS KEY

Enabling SSL/TLS on an OpenStack environment requires an SSL/TLS key to generate your certificates.

Procedure

1. Run the following command to generate the SSL/TLS key (**server.key.pem**):

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

14.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST

Complete the following steps to create a certificate signing request.

Procedure

1. Copy the default OpenSSL configuration file:

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

2. Edit the new **openssl.cnf** file and configure the SSL parameters that you want to use for director. 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 = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

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

- If you are using an IP address to access director over SSL/TLS, use the **undercloud_public_host** parameter in the **undercloud.conf** file.
- If you are using a fully qualified domain name to access director over SSL/TLS, use the domain name.

Edit the **alt_names** section to include the following entries:

- **IP** - A list of IP addresses that clients use to access director over SSL.
- **DNS** - A list of domain names that clients use to access director over SSL. Also include the Public API IP address as a DNS entry at the end of the **alt_names** section.

NOTE

For more information about **openssl.cnf**, run the **man openssl.cnf** command.

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

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

Ensure that you include your OpenStack SSL/TLS key with the **-key** option.

This command generates a **server.csr.pem** file, which is the certificate signing request. Use this file to create your OpenStack SSL/TLS certificate.

14.6. CREATING THE SSL/TLS CERTIFICATE

To generate the SSL/TLS certificate for your OpenStack environment, the following files must be present:

openssl.cnf
: The customized configuration file that specifies the v3 extensions.

server.csr.pem
: The certificate signing request to generate and sign the certificate with a certificate authority.

ca.crt.pem
: The certificate authority, which signs the certificate.

ca.key.pem
: The certificate authority private key.

Procedure

1. Run the following command to create 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 following options:

   -config
   : Use a custom configuration file, which is the **openssl.cnf** file with v3 extensions.

   -extensions v3_req
   : Enabled v3 extensions.

   -days
   : Defines how long in days until the certificate expires.

   -in'
   : The certificate signing request.

   -out
   : The resulting signed certificate.

   -cert
   : The certificate authority file.

   -keyfile
   : The certificate authority private key.

This command creates a new certificate named **server.crt.pem**. Use this certificate in conjunction with your OpenStack SSL/TLS key

14.7. ENABLING SSL/TLS

To enable SSL/TLS in your overcloud, you must create an environment file that contains parameters for your SSL/TLS certiciates and private key.

Procedure

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

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

2. 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:

```
parameter_defaults:
  SSLCertificate: |
    -----BEGIN CERTIFICATE-----
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGS
    ...
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQ
    -----END CERTIFICATE-----
```

IMPORTANT

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

SSLIntermediateCertificate

If you have an intermediate certificate, copy the contents of the intermediate certificate into the **SSLIntermediateCertificate** parameter:

```
parameter_defaults:
  SSLIntermediateCertificate: |
    -----BEGIN CERTIFICATE-----
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvpBCwUAMFgxCzAJB
    ...
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQE
    -----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:

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO
```

```
...
ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4X
-----END RSA PRIVATE KEY-----
```

IMPORTANT

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

14.8. 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.

Procedure

1. Copy the **inject-trust-anchor-hiera.yaml** environment file from the heat template collection:

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

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

CAMap

Lists each certificate authority content (CA) to inject into the overcloud. The overcloud requires the CA files used to sign the certificates for both the undercloud and the overcloud. Copy the contents of the root certificate authority file (**ca.crt.pem**) into an entry. For example, your **CAMap** parameter might look like the following:

```
parameter_defaults:
  CAMap:
    ...
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDlTCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCS
        BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBw
        UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBA
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDBzCCAe+gAwIBAgIJAIc75A7FD++DMA0GCS
        BAMMD3d3dy5leGFtcGxlLmNvbTAeFw0xOTAxMz
        Um54yGCARyp3LpkxvyfMXX1DokpS1uKi7s6CkF
        -----END CERTIFICATE-----
```

IMPORTANT

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

You can also inject additional CAs with the **CAMap** parameter.

14.9. CONFIGURING DNS ENDPOINTS

If you use a DNS hostname to access the overcloud through SSL/TLS, copy the **/usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** file into the **/home/stack/templates** directory.

NOTE

It is not possible to redeploy with a TLS-everywhere architecture if this environment file is not included in the initial deployment.

Configure the host and domain names for all fields, adding parameters for custom networks if needed:

CloudDomain
: the DNS domain for hosts.

CloudName
: The DNS hostname of the overcloud endpoints.

CloudNameCtlplane
: The DNS name of the provisioning network endpoint.

CloudNameInternal
: The DNS name of the Internal API endpoint.

CloudNameStorage
: The DNS name of the storage endpoint.

CloudNameStorageManagement
: The DNS name of the storage management endpoint.

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

Procedure

- Add a list of DNS servers to use under parameter defaults, in either a new or existing environment file:

```
parameter_defaults:
  DnsServers: ["10.0.0.254"]
  ....
```

14.10. ADDING ENVIRONMENT FILES DURING OVERCLOUD CREATION

Use the **-e** option with the deployment command **openstack overcloud deploy** to include environment files in the deployment process. 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 (**custom-domain.yaml**)

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

Procedure

- Use the following deployment command snippet as an example of how to include your SSL/TLS environment files:

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

14.11. MANUALLY UPDATING SSL/TLS CERTIFICATES

Complete the following steps if you are using your own SSL/TLS certificates that are not auto-generated from the TLS everywhere (TLS-e) process.

Procedure

1. Edit your heat templates with the following content:
   - 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-hiera.yaml** file and update the **CAMap** parameter.
2. Rerun the deployment command:

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-
dns.yaml
```

3. On the director, run the following command for each Controller:

```
ssh heat-admin@<controller> sudo podman \
restart $(podman ps --format="{{.Names}}" | grep -w -E 'haproxy(-bundle-.*-[0-9]+)?')
```

CHAPTER 15. 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, director integrates with a Red Hat Identity Management (IdM) server to act as a certificate authority and manage the overcloud certificates.

To check the status of TLS support across the OpenStack components, refer to the TLS Enablement status matrix.

15.1. IDENTITY MANAGEMENT (IDM) SERVER RECOMMENDATIONS FOR OPENSTACK

Red Hat provides the following information to help you integrate your IdM server and OpenStack environment.

For information on preparing Red Hat Enterprise Linux for an IdM installation, see Installing Identity Management.

Run the **ipa-server-install** command to install and configure IdM. You can use command parameters to skip interactive prompts. Use the following recommendations so that your IdM server can integrate with your Red Hat OpenStack Platform environment:

Table 15.1. Parameter recommendations

Option	Recommendation
--admin-password	Note the value you provide. You will need this password when configuring Red Hat OpenStack Platform to work with IdM.
--ip-address	Note the value you provide. The undercloud and overcloud nodes require network access to this ip address.
--setup-dns	Use this option to install an integrated DNS service on the IdM server. The undercloud and overcloud nodes use the IdM server for domain name resolution.
--auto-forwarders	Use this option to use the addresses in **/etc/resolv.conf** as DNS forwarders.
--auto-reverse	Use this option to resolve reverse records and zones for the IdM server IP addresses. If neither reverse records or zones are resolvable, IdM creates the reverse zones. This simplifies the IdM deployment.
--ntp-server, **--ntp-pool**	You can use both or either of these options to configure your NTP source. Both the IdM server and your OpenStack environment must have correct and synchronized time.

You must open the firewall ports required by IdM to enable communication with Red Hat OpenStack Platform nodes. For more information, see Opening the ports required by IdM .

Additional resources

- Configuring and Managing Identity Management
- Red Hat Identity Management Documentation

15.2. IMPLEMENTING TLS-E WITH ANSIBLE

You can use the new **tripleo-ipa** method to enable SSL/TLS on overcloud endpoints, called TLS everywhere (TLS-e). Due to the number of certificates required, Red Hat OpenStack Platform integrates with Red Hat Identity management (IdM). When you use **tripleo-ipa** to configure TLS-e, IdM is the certificate authority.

Prerequisites

Ensure that all configuration steps for the undercloud, such as the creation of the stack user, are complete. For more details, see Director Installation and Usage for more details

Procedure

Use the following procedure to implement TLS-e on a new installation of Red Hat OpenStack Platform, or an existing deployment that you want to configure with TLS-e. You must use this method if you deploy Red Hat OpenStack Platform with TLS-e on pre-provisioned nodes.

NOTE

If you are implementing TLS-e for an existing environment, you are required to run commands such as **openstack undercloud install**, and **openstack overcloud deploy**. These procedures are idempotent and only adjust your existing deployment configuration to match updated templates and configuration files.

1. Configure the **/etc/resolv.conf** file:
   Set the appropriate search domains and the nameserver on the undercloud in **/etc/resolv.conf**. For example, if the deployment domain is **example.com**, and the domain of the FreeIPA server is **bigcorp.com**, then add the following lines to /etc/resolv.conf:

   ```
   search example.com bigcorp.com
   nameserver $IDM_SERVER_IP_ADDR
   ```

2. Install required software:

   ```
   sudo dnf install -y python3-ipalib python3-ipaclient krb5-devel
   ```

3. Export environmental variables with values specific to your environment.:

   ```
   export IPA_DOMAIN=bigcorp.com
   export IPA_REALM=BIGCORP.COM
   export IPA_ADMIN_USER=$IPA_USER
   export IPA_ADMIN_PASSWORD=$IPA_PASSWORD
   export IPA_SERVER_HOSTNAME=ipa.bigcorp.com
   ```

```
export UNDERCLOUD_FQDN=undercloud.example.com
export USER=stack
export CLOUD_DOMAIN=example.com
```

> **NOTE**
>
> The IdM user credentials must be an administrative user that can add new hosts and services.

4. Run the **undercloud-ipa-install.yaml** ansible playbook on the undercloud:

```
ansible-playbook \
--ssh-extra-args "-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" \
/usr/share/ansible/tripleo-playbooks/undercloud-ipa-install.yaml
```

5. Add the following parameters to undercloud.conf

```
undercloud_nameservers = $IDM_SERVER_IP_ADDR
overcloud_domain_name = example.com
```

6. Deploy the undercloud:

```
openstack undercloud install
```

Verification

Verify that the undercloud was enrolled correctly by completing the following steps:

1. List the hosts in IdM:

```
$ kinit admin
$ ipa host-find
```

2. Confirm that **/etc/novajoin/krb5.keytab** exists on the undercloud.

```
ls /etc/novajoin/krb5.keytab
```

> **NOTE**
>
> The **novajoin** directory name is for legacy naming purposes only.

Configuring TLS-e on the overcloud

When you deploy the overcloud with TLS everywhere (TLS-e), IP addresses from the Undercloud and Overcloud will automatically be registered with IdM.

1. Before deploying the overcloud, create a YAML file **tls-parameters.yaml** with contents similar to the following. The values you select will be specific for your environment:

```
parameter_defaults:
   DnsSearchDomains: ["example.com"]
   DnsServers: ["192.168.1.13"]
   CloudDomain: example.com
```

```
    CloudName: overcloud.example.com
    CloudNameInternal: overcloud.internalapi.example.com
    CloudNameStorage: overcloud.storage.example.com
    CloudNameStorageManagement: overcloud.storagemgmt.example.com
    CloudNameCtlplane: overcloud.ctlplane.example.com
    IdMServer: freeipa-0.redhat.local
    IdMDomain: redhat.local
    IdMInstallClientPackages: False

resource_registry:
      OS::TripleO::Services::IpaClient: /usr/share/openstack-tripleo-heat-
templates/deployment/ipa/ipaservices-baremetal-ansible.yaml
```

- The shown value of the **OS::TripleO::Services::IpaClient** parameter overrides the default setting in the **enable-internal-tls.yaml** file. You must ensure the **tls-parameters.yaml** file follows **enable-internal-tls.yaml** in the **openstack overcloud deploy** command.

2. Deploy the overcloud. You will need to include the tls-parameters.yaml in the deployment command:

```
DEFAULT_TEMPLATES=/usr/share/openstack-tripleo-heat-templates/
CUSTOM_TEMPLATES=/home/stack/templates

openstack overcloud deploy \
-e ${DEFAULT_TEMPLATES}/environments/ssl/tls-everywhere-endpoints-dns.yaml \
-e ${DEFAULT_TEMPLATES}/environments/services/haproxy-public-tls-certmonger.yaml \
-e ${DEFAULT_TEMPLATES}/environments/ssl/enable-internal-tls.yaml \
-e ${CUSTOM_TEMPLATES}/tls-parameters.yaml \
...
```

3. Confirm each endpoint is using HTTPS by querying keystone for a list of endpoints:

```
openstack endpoint list
```

15.3. ENROLLING NODES IN RED HAT IDENTITY MANAGER (IDM) WITH NOVAJOIN

Novajoin is the default tool that you use to enroll your nodes with Red Hat Identity Manager (IdM) as part of the deployment process. Red Hat recommends the new ansible-based **tripleo-ipa** solution over the default **novajoin** solution to configure your undercloud and overcloud with TLS-e. For more information see Implementing TLS-e with Ansible.

You must perform the enrollment process before you proceed with the rest of the IdM integration. The enrollment process includes the following steps:

1. Adding the undercloud node to the certificate authority (CA)
2. Adding the undercloud node to IdM
3. Optional: Setting the IdM server as the DNS server for the overcloud
4. Preparing the environment files and deploying the overcloud
5. Testing the overcloud enrollment in IdM and in RHOSP

6. Optional: Adding DNS entries for novajoin in IdM

NOTE

IdM enrollment with novajoin is currently only available for the undercloud and overcloud nodes. Novajoin integration for overcloud instances is expected to be supported in a later release.

15.4. ADDING THE UNDERCLOUD NODE TO THE CERTIFICATE AUTHORITY

Before you deploy the overcloud, add the undercloud to the certificate authority (CA) by installing the **python3-novajoin** package on the undercloud node and running the **novajoin-ipa-setup** script.

Procedure

1. On the undercloud node, install the **python3-novajoin** package:

```
$ sudo dnf install python3-novajoin
```

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

```
$ sudo /usr/libexec/novajoin-ipa-setup \
    --principal admin \
    --password <IdM admin password> \
    --server <IdM server hostname> \
    --realm <realm> \
    --domain <overcloud cloud domain> \
    --hostname <undercloud hostname> \
    --precreate
```

Use the resulting One-Time Password (OTP) to enroll the undercloud.

15.5. ADDING THE UNDERCLOUD NODE TO RED HAT IDENTITY MANAGER (IDM)

After you add the undercloud node to the certificate authority (CA), register the undercloud with IdM and configure novajoin. Configure the following settings in the **[DEFAULT]** section of the **undercloud.conf** file.

Procedure

1. Enable the **novajoin** service:

```
[DEFAULT]
enable_novajoin = true
```

2. Set a One-Time Password (OTP) so that you can register the undercloud node with IdM:

```
ipa_otp = <otp>
```

3. Set the overcloud's domain name to be served by neutron's DHCP server:

   ```
   overcloud_domain_name = <domain>
   ```

4. Set the 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, review the novajoin connection timeout values. In the **undercloud.conf** file, 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. Optional: If you want the local openSSL certificate authority to generate the SSL certificates for the public endpoints in director, set the **generate_service_certificate** parameter to **true**:

   ```
   generate_service_certificate = true
   ```

8. Save the **undercloud.conf** file.

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

   ```
   $ openstack undercloud install
   ```

Verification

Verify that the undercloud was enrolled correctly by completing the following steps:

1. List the hosts in IdM:

   ```
   $ kinit admin
   $ ipa host-find
   ```

2. Confirm that **/etc/novajoin/krb5.keytab** exists on the undercloud.

   ```
   ls /etc/novajoin/krb5.keytab
   ```

15.6. SETTING RED HAT IDENTITY MANAGER (IDM) AS THE DNS SERVER FOR THE OVERCLOUD

To enable automatic detection of your IdM environment and easier enrollment, set IdM as your DNS server. This procedure is optional but recommended.

Procedure

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.

15.7. PREPARING ENVIRONMENT FILES AND DEPLOYING THE OVERCLOUD WITH NOVAJOIN ENROLLMENT

To deploy the overcloud with IdM integration, you create and edit environment files to configure the overcloud to use the custom domain parameters **CloudDomain** and **CloudName** based on the domains that you define in the overcloud. You then deploy the overcloud with all the environment files and any additional environment files that you need for the deployment.

Procedure

1. 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:

```
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. Choose the implementation of TLS appropriate for your environment:

   - Use the **enable-tls.yaml** environment file to protect external endpoints with your custom certificate:

     a. Copy **/usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml** to **/home/stack/templates**.

b. Modify the **/home/stack/enable-tls.yaml** environment file to include your custom certificate and key.

c. Include the following environment files in your deployment to protect internal and external endpoints:

   - enable-internal-tls.yaml
   - tls-every-endpoints-dns.yaml
   - custom-domain.yaml
   - enable-tls.yaml

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

- Use the **haproxy-public-tls-certmonger.yaml** environment file to protect external endpoints with an IdM issued certificate. For this implementation, you must create DNS entries for the VIP endpoints used by novajoin:

a. You must create DNS entries for the VIP endpoints used by novajoin. Identify the overcloud networks located in your custom **network-environment.yaml file in `/home/stack/templates**:

```
parameter_defaults:
    ControlPlaneDefaultRoute: 192.168.24.1
    ExternalAllocationPools:
    -   end: 10.0.0.149
        start: 10.0.0.101
    InternalApiAllocationPools:
    -   end: 172.17.1.149
        start: 172.17.1.10
    StorageAllocationPools:
    -   end: 172.17.3.149
        start: 172.17.3.10
    StorageMgmtAllocationPools:
    -   end: 172.17.4.149
        start: 172.17.4.10
```

b. Create a list of virtual IP addresses for each overcloud network in a heat template, for example, **/home/stack/public_vip.yaml**.

```
parameter_defaults:
    ControlFixedIPs: [{'ip_address':'192.168.24.101'}]
    PublicVirtualFixedIPs: [{'ip_address':'10.0.0.101'}]
    InternalApiVirtualFixedIPs: [{'ip_address':'172.17.1.101'}]
    StorageVirtualFixedIPs: [{'ip_address':'172.17.3.101'}]
    StorageMgmtVirtualFixedIPs: [{'ip_address':'172.17.4.101'}]
    RedisVirtualFixedIPs: [{'ip_address':'172.17.1.102'}]
```

c. Add DNS entries to the IdM for each of the VIPs, and zones as needed:

```
ipa dnsrecord-add lab.local overcloud --a-rec 10.0.0.101
ipa dnszone-add ctlplane.lab.local
ipa dnsrecord-add ctlplane.lab.local overcloud --a-rec 192.168.24.101
ipa dnszone-add internalapi.lab.local
ipa dnsrecord-add internalapi.lab.local overcloud --a-rec 172.17.1.101
ipa dnszone-add storage.lab.local
ipa dnsrecord-add storage.lab.local overcloud --a-rec 172.17.3.101
ipa dnszone-add storagemgmt.lab.local
ipa dnsrecord-add storagemgmt.lab.local overcloud --a-rec 172.17.4.101
```

d. Include the following environment files in your deployment to protect internal and external endpoints:

- enable-internal-tls.yaml
- tls-everywhere-endpoints-dns.yaml
- haproxy-public-tls-certmonger.yaml
- custom-domain.yaml
- public_vip.yaml

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

NOTE

You cannot use novajoin to implement TLS everywhere (TLS-e) on a pre-existing deployment.

Additional resources

- Implementing TLS-e with Ansible

CHAPTER 16. CONFIGURING THE IMAGE IMPORT METHOD AND SHARED STAGING AREA

The default settings for the OpenStack Image service (glance) are determined by the heat templates that you use when you install Red Hat OpenStack Platform. The Image service heat template is **deployment/glance/glance-api-container-puppet.yaml**.

You can import images with the following methods:

web-download

Use the **web-download** method to import an image from a URL.

glance-direct

Use the **glance-direct** method to import an image from a local volume.

16.1. CREATING AND DEPLOYING THE GLANCE-SETTINGS.YAML FILE

Use a custom environment file to configure the import parameters. These parameters override the default values that are present in the core heat template collection. The example environment content contains parameters for the interoperable image import.

```
parameter_defaults:
  # Configure NFS backend
  GlanceBackend: file
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.168.122.1:/export/glance

  # Enable glance-direct import method
  GlanceEnabledImportMethods: glance-direct,web-download

  # Configure NFS staging area (required for glance-direct import method)
  GlanceStagingNfsShare: 192.168.122.1:/export/glance-staging
```

The **GlanceBackend**, **GlanceNfsEnabled**, and **GlanceNfsShare** parameters are defined in the Storage Configuration section in the *Advanced Overcloud Customization Guide*.

Use two new parameters for interoperable image import to define the import method and a shared NFS staging area.

GlanceEnabledImportMethods

Defines the available import methods, web-download (default) and glance-direct. This parameter is necessary only if you want to enable additional methods besides web-download.

GlanceStagingNfsShare

Configures the NFS staging area that the glance-direct import method uses. This space can be shared among nodes in a high-availability cluster configuration. If you want to use this parameter, you must also set the **GlanceNfsEnabled** parameter to **true**.

Procedure

1. Create a new file, for example, **glance-settings.yaml**. Use the syntax from the example to populate this file.

2. Include the **glance-settings.yaml** file in the **openstack overcloud deploy** command, as well as any other environment files that are relevant to your deployment:

   ```
   $ openstack overcloud deploy --templates -e glance-settings.yaml
   ```

For more information about using environment files, see the Including Environment Files in Overcloud Creation section in the *Advanced Overcloud Customization Guide*.

16.2. CONTROLLING IMAGE WEB-IMPORT SOURCES

You can limit the sources of web-import image downloads by adding URI blocklists and allowlists to the optional **glance-image-import.conf** file.

You can allow or block image source URIs at three levels:

- scheme (allowed_schemes, disallowed_schemes)
- host (allowed_hosts, disallowed_hosts)
- port (allowed_ports, disallowed_ports)

If you specify both allowlist and blocklist at any level, the allowlist is honored and the blocklist is ignored.

The Image service (glance) applies the following decision logic to validate image source URIs:

1. The scheme is checked.
   a. Missing scheme: reject
   b. If there is an allowlist, and the scheme is not present in the allowlist: reject. Otherwise, skip C and continue on to 2.
   c. If there is a blocklist, and the scheme is present in the blocklist: reject.
2. The host name is checked.
   a. Missing host name: reject
   b. If there is an allowlist, and the host name is not present in the allowlist: reject. Otherwise, skip C and continue on to 3.
   c. If there is a blocklist, and the host name is present in the blocklist: reject.
3. If there is a port in the URI, the port is checked.
   a. If there is a allowlist, and the port is not present in the allowlist: reject. Otherwise, skip B and continue on to 4.
   b. If there is a blocklist, and the port is present in the blocklist: reject.
4. The URI is accepted as valid.

If you allow a scheme, either by adding it to an allowlist or by not adding it to a blocklist, any URI that uses the default port for that scheme by not including a port in the URI is allowed. If it does include a port in the URI, the URI is validated according to the default decision logic.

16.3. IMAGE IMPORT EXAMPLE

For example, the default port for FTP is 21. Because *ftp* is an allowlisted scheme, this URL is allowed: ftp://example.org/some/resource But because 21 is not in the port allowlist, this URL to the same resource is rejected: ftp://example.org:21/some/resource

```
allowed_schemes = [http,https,ftp]
disallowed_schemes = []
allowed_hosts = []
disallowed_hosts = []
allowed_ports = [80,443]
disallowed_ports = []
```

16.4. DEFAULT IMAGE IMPORT BLOCKLIST AND ALLOWLIST SETTINGS

The **glance-image-import.conf** file is an optional file that contains the following default options:

- allowed_schemes - [*http*, *https*]
- disallowed_schemes - empty list
- allowed_hosts - empty list
- disallowed_hosts - empty list
- allowed_ports - [80, 443]
- disallowed_ports - empty list

If you use the defaults, end users can access URIs by using only the **http** or **https** scheme. The only ports that users can specify are **80** and **443**. Users do not have to specify a port, but if they do, it must be either **80** or **443**.

You can find the **glance-image-import.conf** file in the **etc/** subdirectory of the Image service source code tree. Ensure that you are looking in the correct branch for your release of Red Hat OpenStack Platform.

16.5. INJECTING METADATA ON IMAGE IMPORT TO CONTROL WHERE VMS LAUNCH

End users can upload images to the Image service and use these images to launch VMs. These user-provided (non-admin) images must be launched on a specific set of compute nodes. The assignment of an instance to a compute node is controlled by image metadata properties.

The Image Property Injection plugin injects metadata properties to images during import. Specify the properties by editing the [image_import_opts] and [inject_metadata_properties] sections of the **glance-image-import.conf** file.

To enable the Image Property Injection plugin, add the following line to the **[image_import_opts]** section:

```
[image_import_opts]
image_import_plugins = [inject_image_metadata]
```

To limit the metadata injection to images provided by a certain set of users, set the **ignore_user_roles** parameter. For example, use the following configuration to inject one value for **property1** and two values for **property2** into images downloaded by any non-admin user.

```
[DEFAULT]
[image_conversion]
[image_import_opts]
image_import_plugins = [inject_image_metadata]
[import_filtering_opts]
[inject_metadata_properties]
ignore_user_roles = admin
inject = PROPERTY1:value,PROPERTY2:value;another value
```

The parameter **ignore_user_roles** is a comma-separated list of the Identity service (keystone) roles that the plugin ignores. This means that if the user that makes the image import call has any of these roles, the plugin does not inject any properties into the image.

The parameter **inject** is a comma-separated list of properties and values that are injected into the image record for the imported image. Each property and value must be quoted and separated by a colon **(':')**.

You can find the **glance-image-import.conf** file in the **etc/** subdirectory of the Image service source code tree. Ensure that you are looking in the correct branch for your release of Red Hat OpenStack Platform.

CHAPTER 17. STORAGE CONFIGURATION

This chapter outlines several methods that you can use to configure storage options for your overcloud.

IMPORTANT

The overcloud uses local ephemeral storage and Logical Volume Manager (LVM) storage for the default storage options. Local ephemeral storage is supported in production environments but LVM storage is not supported.

17.1. CONFIGURING NFS STORAGE

You can configure the overcloud to use shared NFS storage.

17.1.1. Supported configurations and limitations

Supported NFS storage

- Red Hat recommends that you use a certified storage back end and driver. Red Hat does not recommend that you use NFS storage that comes from the generic NFS back end, because its capabilities are limited compared to a certified storage back end and driver. For example, the generic NFS back end does not support features such as volume encryption and volume multi-attach. For information about supported drivers, see the Red Hat Ecosystem Catalog .

- For Block Storage (cinder) and Compute (nova) services, you must use NFS version 4.0 or later. Red Hat OpenStack Platform (RHOSP) does not support earlier versions of NFS.

Unsupported NFS configuration

- RHOSP does not support the NetApp feature NAS secure, because it interferes with normal volume operations. Director disables the feature by default. Therefore, do not edit the following heat parameters that control whether an NFS back end or a NetApp NFS Block Storage back end supports NAS secure:
  - **CinderNetappNasSecureFileOperations**
  - **CinderNetappNasSecureFilePermissions**
  - **CinderNasSecureFileOperations**
  - **CinderNasSecureFilePermissions**

Limitations when using NFS shares

- Instances that have a swap disk cannot be resized or rebuilt when the back end is an NFS share.

17.1.2. Configuring NFS storage

You can configure the overcloud to use shared NFS storage.

Procedure

1. Create an environment file to configure your NFS storage, for example, **nfs_storage.yaml**.

2. Add the following parameters to your new environment file to configure NFS storage:

```
parameter_defaults:
  CinderEnableIscsiBackend: false
  CinderEnableNfsBackend: true
  GlanceBackend: file
  CinderNfsServers: 192.0.2.230:/cinder
  GlanceNfsEnabled: true
  GlanceNfsShare: 192.0.2.230:/glance
```

NOTE

Do not configure the **CinderNfsMountOptions** and **GlanceNfsOptions** parameters, as their default values enable NFS mount options that are suitable for most Red Hat OpenStack Platform (RHOSP) environments. You can see the value of the **GlanceNfsOptions** parameter in the **environments/storage/glance-nfs.yaml** file. If you experience issues when you configure multiple services to share the same NFS server, contact Red Hat Support.

3. Add your NFS storage environment file to the stack with your other environment files and deploy the overcloud:

```
(undercloud)$ openstack overcloud deploy --templates \
 -e [your environment files] \
 -e /home/stack/templates/nfs_storage.yaml
```

17.1.3. Configuring an external NFS share for conversion

When the Block Storage service (cinder) performs image format conversion on the overcloud Controller nodes, and the space is limited, conversion of large Image service (glance) images can cause the node root disk space to be completely used. You can use an external NFS share for the conversion to prevent the space on the node from being completely filled.

There are two director heat parameters that control the external NFS share configuration:

- **CinderImageConversionNfsShare**
- **CinderImageConversionNfsOptions**

Procedure

1. Log in to the undercloud as the **stack** user and source the **stackrc** credentials file.

```
$ source ~/stackrc
```

2. In a new or existing storage-related environment file, add information about the external NFS share.

```
parameter_defaults:
  CinderImageConversionNfsShare: 192.168.10.1:/convert
```

NOTE

The default value of the **CinderImageConversionNfsOptions** parameter, that controls the NFS mount options, is sufficient for most environments.

3. Include the environment file that contains your new configuration in the openstack overcloud deploy command with any other environment files that are relevant to your environment.

```
$ openstack overcloud deploy \
--templates \
...
-e <existing_overcloud_environment_files> \
-e <new_environment_file> \
...
```

- Replace **<existing_overcloud_environment_files>** with the list of environment files that are part of your existing deployment.

- Replace **<new_environment_file>** with the new or edited environment file that contains your NFS share configuration.

17.2. CONFIGURING CEPH STORAGE

Use one of the following methods to integrate Red Hat Ceph Storage into a Red Hat OpenStack Platform overcloud.

Creating an overcloud with its own Ceph Storage cluster

You can create a Ceph Storage Cluster during the creation on the overcloud. Director creates a set of Ceph Storage nodes that use the Ceph OSD to store data. Director also installs the Ceph Monitor service on the overcloud Controller nodes. This means that 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 .

Integrating an existing Ceph Storage cluster into an overcloud

If you have an existing Ceph Storage Cluster, you can integrate this cluster into a Red Hat OpenStack Platform overcloud during deployment. This means that you can 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.

17.3. USING AN EXTERNAL OBJECT STORAGE CLUSTER

You can reuse an external OpenStack Object Storage (swift) cluster by disabling the default Object Storage service deployment on your Controller nodes. This disables both the proxy and storage services for Object Storage and configures haproxy and OpenStack Identify (keystone) to use the given external Object Storage endpoint.

NOTE

You must manage user accounts on the external Object Storage (swift) cluster manually.

Prerequisites

- 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.

Procedure

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 with the following external Object Storage service environment files, as well as any other environment files that are relevant to your deployment:

   ```
   openstack overcloud deploy --templates \
   -e [your environment files] \
   -e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml \
   -e swift-external-params.yaml
   ```

17.4. CONFIGURING CEPH OBJECT STORE TO USE EXTERNAL CEPH OBJECT GATEWAY

Red Hat OpenStack Platform (RHOSP) director supports configuring an external Ceph Object Gateway (RGW) as an Object Store service. To authenticate with the external RGW service, you must configure RGW to verify users and their roles in the Identity service (keystone).

For more information about how to configure an external Ceph Object Gateway, see Configuring the Ceph Object Gateway to use Keystone authentication in the *Using Keystone with the Ceph Object Gateway Guide*.

Procedure

1. Add the following **parameter_defaults** to a custom environment file, for example, **swift-external-params.yaml**, and adjust the values to suit your deployment:

   ```
   parameter_defaults:
      ExternalSwiftPublicUrl: 'http://<Public RGW endpoint or
   loadbalancer>:8080/swift/v1/AUTH_%(project_id)s'
      ExternalSwiftInternalUrl: 'http://<Internal RGW endpoint>:8080/swift/v1/AUTH_%
   (project_id)s'
      ExternalSwiftAdminUrl: 'http://<Admin RGW endpoint>:8080/swift/v1/AUTH_%(project_id)s'
      ExternalSwiftUserTenant: 'service'
      SwiftPassword: 'choose_a_random_password'
   ```

NOTE

The example code snippet contains parameter values that might differ from values that you use in your environment:

- The default port where the remote RGW instance listens is **8080**. The port might be different depending on how the external RGW is configured.
- The **swift** user created in the overcloud uses the password defined by the **SwiftPassword** parameter. You must configure the external RGW instance to use the same password to authenticate with the Identity service by using the **rgw_keystone_admin_password**.

2. Add the following code to the Ceph config file to configure RGW to use the Identity service. Replace the variable values to suit your environment:

```
    rgw_keystone_api_version = 3
    rgw_keystone_url = http://<public Keystone endpoint>:5000/
    rgw_keystone_accepted_roles = member, Member, admin
    rgw_keystone_accepted_admin_roles = ResellerAdmin, swiftoperator
    rgw_keystone_admin_domain = default
    rgw_keystone_admin_project = service
    rgw_keystone_admin_user = swift
    rgw_keystone_admin_password =
<password_as_defined_in_the_environment_parameters>
    rgw_keystone_implicit_tenants = true
    rgw_keystone_revocation_interval = 0
    rgw_s3_auth_use_keystone = true
    rgw_swift_versioning_enabled = true
    rgw_swift_account_in_url = true
```

NOTE

Director creates the following roles and users in the Identity service by default:

- rgw_keystone_accepted_admin_roles: ResellerAdmin, swiftoperator
- rgw_keystone_admin_domain: default
- rgw_keystone_admin_project: service
- rgw_keystone_admin_user: swift

3. Deploy the overcloud with the additional environment files with any other environment files that are relevant to your deployment:

```
openstack overcloud deploy --templates \
-e <your_environment_files>
-e /usr/share/openstack-tripleo-heat-templates/environments/swift-external.yaml
-e swift-external-params.yaml
```

Verification

1. Log in to the undercloud as the **stack** user.

2. Source the **overcloudrc** file:

```
$ source ~/stackrc
```

3. Verify that the endpoints exist in the Identity service (keystone):

```
$ openstack endpoint list --service object-store

+---------+-----------+-------+-------+---------+-----------+---------------+
| ID | Region    | Service Name | Service Type | Enabled | Interface | URL |
+---------+-----------+-------+-------+---------+-----------+---------------+
| 233b7ea32aaf40c1ad782c696128aa0e | regionOne | swift | object-store | True    | admin     |
http://192.168.24.3:8080/v1/AUTH_%(project_id)s |
| 4ccde35ac76444d7bb82c5816a97abd8 | regionOne | swift | object-store | True    | public    |
https://192.168.24.2:13808/v1/AUTH_%(project_id)s |
| b4ff283f445348639864f560aa2b2b41 | regionOne | swift | object-store | True    | internal  |
http://192.168.24.3:8080/v1/AUTH_%(project_id)s |
+---------+-----------+-------+-------+---------+-----------+---------------+
```

4. Create a test container:

```
$ openstack container create <testcontainer>
+----------------+---------------+------------------------------------+
| account | container | x-trans-id |
+----------------+---------------+------------------------------------+
| AUTH_2852da3cf2fc490081114c434d1fc157 | testcontainer | tx6f5253e710a2449b8ef7e-
005f2d29e8 |
+----------------+---------------+------------------------------------+
```

5. Create a configuration file to confirm that you can upload data to the container:

```
$ openstack object create testcontainer undercloud.conf
+-----------------+---------------+----------------------------------+
| object          | container     | etag                             |
+-----------------+---------------+----------------------------------+
| undercloud.conf | testcontainer | 09fcffe126cac1dbac7b89b8fd7a3e4b |
+-----------------+---------------+----------------------------------+
```

6. Delete the test container:

```
$ openstack container delete -r <testcontainer>
```

17.5. CONFIGURING CINDER BACK END FOR THE IMAGE SERVICE

Use the **GlanceBackend** parameter to set the back end that the Image service uses to store images.

IMPORTANT

The default maximum number of volumes you can create for a project is 10.

Procedure

1. To configure **cinder** as the Image service back end, add the following line to an environment file:

```
parameter_defaults:
  GlanceBackend: cinder
```

2. If the **cinder** back end is enabled, the following parameters and values are set by default:

```
cinder_store_auth_address = http://172.17.1.19:5000/v3
cinder_store_project_name = service
cinder_store_user_name = glance
cinder_store_password = ****secret****
```

3. To use a custom user name, or any custom value for the **cinder_store_** parameters, add the **ExtraConfig** parameter to **parameter_defaults** and include your custom values:

```
ExtraConfig:
    glance::config::api_config:
      glance_store/cinder_store_auth_address:
        value: "%{hiera('glance::api::authtoken::auth_url')}/v3"
      glance_store/cinder_store_user_name:
        value: <user-name>
      glance_store/cinder_store_password:
        value: "%{hiera('glance::api::authtoken::password')}"
      glance_store/cinder_store_project_name:
        value: "%{hiera('glance::api::authtoken::project_name')}"
```

17.6. CONFIGURING THE MAXIMUM NUMBER OF STORAGE DEVICES TO ATTACH TO ONE INSTANCE

By default, you can attach an unlimited number of storage devices to a single instance. To limit the maximum number of devices, add the **max_disk_devices_to_attach** parameter to your Compute environment file. Use the following example to change the value of **max_disk_devices_to_attach** to "30":

```
parameter_defaults:
   ComputeExtraConfig:
          nova::config::nova_config:
            compute/max_disk_devices_to_attach:
                value: '30'
```

Guidelines and considerations

- The number of storage disks supported by an instance depends on the bus that the disk uses. For example, the IDE disk bus is limited to 4 attached devices.
- Changing the **max_disk_devices_to_attach** on a Compute node with active instances can cause rebuilds to fail if the maximum number is lower than the number of devices already attached to instances. For example, if instance **A** has 26 devices attached and you change **max_disk_devices_to_attach** to 20, a request to rebuild instance **A** will fail.
- During cold migration, the configured maximum number of storage devices is enforced only on the source for the instance that you want to migrate. The destination is not checked before the

move. This means that if Compute node A has 26 attached disk devices, and Compute node B has a configured maximum of 20 attached disk devices, a cold migration of an instance with 26 attached devices from Compute node A to Compute node B succeeds. However, a subsequent request to rebuild the instance in Compute node B fails because 26 devices are already attached which exceeds the configured maximum of 20.

- The configured maximum is not enforced on shelved offloaded instances, as they have no Compute node.
- Attaching a large number of disk devices to instances can degrade performance on the instance. Tune the maximum number based on the boundaries of what your environment can support.
- Instances with machine type Q35 can attach a maximum of 500 disk devices.

17.7. IMPROVING SCALABILITY WITH IMAGE SERVICE CACHING

Use the glance-api caching mechanism to store copies of images on Image service (glance) API servers and retrieve them automatically to improve scalability. With Image service caching, glance-api can run on multiple hosts. This means that it does not need to retrieve the same image from back end storage multiple times. Image service caching does not affect any Image service operations.

Configure Image service caching with the Red Hat OpenStack Platform director (tripleo) heat templates:

Procedure

1. In an environment file, set the value of the **GlanceCacheEnabled** parameter to **true**, which automatically sets the **flavor** value to **keystone+cachemanagement** in the **glance-api.conf** heat template:

```
parameter_defaults:
    GlanceCacheEnabled: true
```

2. Include the environment file in the **openstack overcloud deploy** command when you redeploy the overcloud.

3. Optional: Tune the **glance_cache_pruner** to an alternative frequency when you redeploy the overcloud. The following example shows a frequency of 5 minutes:

```
parameter_defaults:
  ControllerExtraConfig:
    glance::cache::pruner::minute: '*/5'
```

   Adjust the frequency according to your needs to avoid file system full scenarios. Include the following elements when you choose an alternative frequency:

   - The size of the files that you want to cache in your environment.
   - The amount of available file system space.
   - The frequency at which the environment caches images.

17.8. CONFIGURING THIRD PARTY STORAGE

The following environment files are present in the core heat template collection **/usr/share/openstack-tripleo-heat-templates**.

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**.

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**.

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/storage/cinder-netapp-config.yaml**.

CHAPTER 18. SECURITY ENHANCEMENTS

The following sections provide some suggestions to harden the security of your overcloud.

18.1. 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.

Procedure

1. Log in to the undercloud node as the **stack** user.
2. Each overcloud node has a **heat-admin** user account. This user account contains the undercloud public SSH key, which provides SSH access without a password from the undercloud to the overcloud node. On the undercloud node, log in to the an overcloud node through SSH as the **heat-admin** user.
3. Switch to the **root** user with **sudo -i**.

18.2. 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 that can help with additional firewall management:

ManageFirewall
: Defines whether to automatically manage the firewall rules. Set this parameter 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 you set the **ManageFirewall** parameter 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. Each rule name starts with a three-digit prefix to help Puppet order all defined rules in the final **iptables** file. The default Red Hat OpenStack Platform rules use prefixes in the 000 to 200 range.

18.3. CHANGING THE SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) STRINGS

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.

NOTE

When you configure the **ExtraConfig** interface with a string parameter, you must use the following syntax to ensure that heat and Hiera do not interpret the string as a Boolean value: **'"<VALUE>"'**.

Set the following hieradata using the **ExtraConfig** hook in an environment file for your overcloud:

SNMP traditional access control settings

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**.

tripleo::profile::base::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.

SNMP view-based access control settings (VACM)

snmp::com2sec

IPv4 security name.

snmp::com2sec6

IPv6 security name.

For example:

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: mysecurestring
    snmp::com2sec6: myv6securestring
```

This changes the read-only SNMP community string on all nodes.

For more information, see the **snmpd.conf** man page.

18.4. CHANGING THE SSL/TLS CIPHER AND RULES FOR HAPROXY

If you enabled SSL/TLS in the overcloud, consider hardening the SSL/TLS ciphers and rules that are used with the HAProxy configuration. By hardening the SSL/TLS ciphers, you help avoid SSL/TLS vulnerabilities, such as the POODLE vulnerability.

1. Create a heat template environment file called **tls-ciphers.yaml**:

```
touch ~/templates/tls-ciphers.yaml
```

2. Use the **ExtraConfig** hook in the environment file to apply values to the **tripleo::haproxy::ssl_cipher_suite** and **tripleo::haproxy::ssl_options** hieradata:

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite: 'DHE-RSA-AES128-CCM:DHE-RSA-AES256-
CCM:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES128-CCM:ECDHE-ECDSA-AES256-CCM:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-CHACHA20-POLY1305'

    tripleo::haproxy::ssl_options: 'no-sslv3 no-tls-tickets'
```

NOTE

The cipher collection is one continuous line.

3. Include the **tls-ciphers.yaml** environment file with the overcloud deploy command when deploying the overcloud:

```
openstack overcloud deploy --templates \
...
-e /home/stack/templates/tls-ciphers.yaml
...
```

18.5. USING THE OPEN VSWITCH FIREWALL

You can configure security groups to use the Open vSwitch (OVS) firewall driver in Red Hat OpenStack Platform director. Use the **NeutronOVSFirewallDriver** parameter to specify firewall driver that you want to use:

- **iptables_hybrid** - Configures the Networking service (neutron) to use the iptables/hybrid based implementation.
- **openvswitch** - Configures the Networking service 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.

IMPORTANT

Multicast traffic is handled differently by the Open vSwitch (OVS) firewall driver than by the iptables firewall driver. With iptables, by default, VRRP traffic is denied, and you must enable VRRP in the security group rules for any VRRP traffic to reach an endpoint. With OVS, all ports share the same OpenFlow context, and multicast traffic cannot be processed individually per port. Because security groups do not apply to all ports (for example, the ports on a router), OVS uses the **NORMAL** action and forwards multicast traffic to all ports as specified by RFC 4541.

NOTE

The **iptables_hybrid** option is not compatible with OVS-DPDK. The **openvswitch** option is not compatible with OVS Hardware Offload.

Configure the **NeutronOVSFirewallDriver** parameter in the **network-environment.yaml** file:

```
NeutronOVSFirewallDriver: openvswitch
```

- **NeutronOVSFirewallDriver** : Configures the name of the firewall driver that you want to use when you implement security groups. Possible values depend on your system configuration. Some examples are **noop**, **openvswitch**, and **iptables_hybrid**. The default value of an empty string results in a supported configuration.

CHAPTER 19. CONFIGURING NETWORK PLUGINS

Director includes environment files that you can use when you configure third-party network plugins:

19.1. FUJITSU CONVERGED FABRIC (C-FABRIC)

You can enable the Fujitsu Converged Fabric (C-Fabric) plugin by using the environment file located at **/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-cfab.yaml**.

Procedure

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:

```
$ 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 by using the environment file located at **/usr/share/openstack-tripleo-heat-templates/environments/neutron-ml2-fujitsu-fossw.yaml**.

Procedure

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:

```
$ openstack overcloud deploy --templates -e /home/stack/templates/neutron-ml2-fujitsu-fossw.yaml [OTHER OPTIONS] ...
```

CHAPTER 20. CONFIGURING IDENTITY

Director includes parameters to help configure Identity Service (keystone) settings:

20.1. REGION NAME

By default, your overcloud region is named **regionOne**. You can change this by adding a **KeystoneRegion** entry your environment file. You cannot modify this value after you deploy the overcloud.

```
parameter_defaults:
  KeystoneRegion: 'SampleRegion'
```

CHAPTER 21. MISCELLANEOUS OVERCLOUD CONFIGURATION

Use the following configurations to configure miscellaneous features in the overcloud.

21.1. 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.

Procedure

- Set the parameter in the **parameter_defaults** section of an environment file:

```
parameter_defaults:
  KeystoneDebug: True
```

After you have set the **KeystoneDebug** parameter to **True**, the **/var/log/containers/keystone/keystone.log** standard keystone log file is updated with **DEBUG** level logs.

For a full list of debug parameters, see "Debug Parameters" in the *Overcloud Parameters* guide.

21.2. CONFIGURING THE KERNEL ON OVERCLOUD NODES

Red Hat OpenStack Platform director includes parameters that configure the kernel on overcloud nodes.

ExtraKernelModules

Kernel modules to load. The modules names are listed as a hash key with an empty value:

```
ExtraKernelModules:
  <MODULE_NAME>: {}
```

ExtraKernelPackages

Kernel-related packages to install prior to loading the kernel modules from **ExtraKernelModules**. The package names are listed as a hash key with an empty value.

```
ExtraKernelPackages:
  <PACKAGE_NAME>: {}
```

ExtraSysctlSettings

Hash of sysctl settings to apply. Set the value of each parameter using the **value** key.

```
ExtraSysctlSettings:
  <KERNEL_PARAMETER>:
    value: <VALUE>
```

This example shows the syntax of these parameters in an environment file:

```
parameter_defaults:
  ExtraKernelModules:
    iscsi_target_mod: {}
  ExtraKernelPackages:
    iscsi-initiator-utils: {}
  ExtraSysctlSettings:
    dev.scsi.logging_level:
      value: 1
```

21.3. CONFIGURING THE SERVER CONSOLE

Console output from overcloud nodes is not always sent to the server console. If you want to view this output in the server console, you must configure the overcloud to use the correct console for your hardware. Use one of the following methods to perform this configuration:

- Modify the **KernelArgs** heat parameter for each overcloud role.
- Customize the **overcloud-hardened-uefi-full.qcow2** image that director uses to provision the overcloud nodes.

Prerequisites

- A successful undercloud installation. For more information, see the Director Installation and Usage guide.
- Overcloud nodes ready for deployment.

Modifying KernelArgs with heat during deployment

1. Log in to the undercloud host as the **stack** user.
2. Source the **stackrc** credentials file:

   ```
   $ source stackrc
   ```

3. Create an environment file **overcloud-console.yaml** with the following content:

   ```
   parameter_defaults:
     <role>Parameters:
       KernelArgs: "console=<console-name>"
   ```

   Replace **<role>** with the name of the overcloud role that you want to configure, and replace **<console-name>** with the ID of the console that you want to use. For example, use the following snippet to configure all overcloud nodes in the default roles to use **tty0**:

   ```
   parameter_defaults:
     ControllerParameters:
       KernelArgs: "console=tty0"
     ComputeParameters:
       KernelArgs: "console=tty0"
     BlockStorageParameters:
       KernelArgs: "console=tty0"
   ```

```
ObjectStorageParameters:
  KernelArgs: "console=tty0"
CephStorageParameters:
  KernelArgs: "console=tty0"
```

4. Include the **overcloud-console-tty0.yaml** file in your deployment command with the **-e** option.

Modifying the **overcloud-hardened-uefi-full.qcow2** image

1. Log in to the undercloud host as the **stack** user.
2. Source the **stackrc** credentials file:

```
$ source stackrc
```

3. Modify the kernel arguments in the **overcloud-hardened-uefi-full.qcow2** image to set the correct console for your hardware. For example, set the console to **tty1**:

```
$ virt-customize --selinux-relabel -a overcloud-hardened-uefi-full.qcow2 --run-command
'grubby --update-kernel=ALL --args="console=tty1"'
```

4. Import the image into director:

```
$ openstack overcloud image upload --image-path overcloud-hardened-uefi-full.qcow2
```

5. Deploy the overcloud.

Verification

1. Log in to an overcloud node from the undercloud:

```
$ ssh tripleo-admin@<IP-address>
```

Replace **<IP-address>** with the IP address of an overcloud node.

2. Inspect the contents of the **/proc/cmdline** file and ensure that **console=** parameter is set to the value of the console that you want to use:

```
[tripleo-admin@controller-0 ~]$ cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos2)/boot/vmlinuz-4.18.0-193.29.1.el8_2.x86_64
root=UUID=0ec3dea5-f293-4729-b676-5d38a611ce81 ro console=tty0
console=ttyS0,115200n81 no_timer_check crashkernel=auto rhgb quiet
```

21.4. 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. You can also use an external load balancer to perform this distribution. For example, you can use your 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.

21.5. CONFIGURING IPV6 NETWORKING

This section examines the network configuration for the overcloud. This includes isolating the OpenStack services to use specific network traffic and configuring the overcloud with IPv6 options.