Red Hat OpenStack Platform 17.1

Customizing your Red Hat OpenStack Platform deployment

Customizing your core Red Hat OpenStack Platform deployment for your environment and requirements.

Last Updated: 2024-03-06

Red Hat OpenStack Platform 17.1 Customizing your Red Hat OpenStack Platform deployment

Customizing your core Red Hat OpenStack Platform deployment for your environment and requirements.

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Guidance on how to customize a basic Red Hat OpenStack Platform deployment for your environment and requirements, and how to use Ansible and the Orchestration service.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 5

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION 6

CHAPTER 1. PLANNING CUSTOM UNDERCLOUD FEATURES 7
1.1. CHARACTER ENCODING CONFIGURATION 7
1.2. CONSIDERATIONS WHEN RUNNING THE UNDERCLOUD WITH A PROXY 7

CHAPTER 2. COMPOSABLE SERVICES AND CUSTOM ROLES 9
2.1. SUPPORTED ROLE ARCHITECTURE 9
2.2. EXAMINING THE ROLES_DATA FILE 9
2.3. CREATING A ROLES_DATA FILE 10
2.4. SUPPORTED CUSTOM ROLES 11
2.5. EXAMINING ROLE PARAMETERS 14
2.6. CREATING A NEW ROLE 16
2.7. GUIDELINES AND LIMITATIONS 18
2.8. CONTAINERIZED SERVICE ARCHITECTURE 19
2.9. CONTAINERIZED SERVICE PARAMETERS 19
2.10. EXAMINING COMPOSABLE SERVICE ARCHITECTURE 20
2.11. ADDING AND REMOVING SERVICES FROM ROLES 22
2.12. ENABLING DISABLED SERVICES 23

CHAPTER 3. USING THE VALIDATION FRAMEWORK 24
3.1. ANSIBLE-BASED VALIDATIONS 24
3.2. CHANGING THE VALIDATION CONFIGURATION FILE 24
3.3. LISTING VALIDATIONS 25
3.4. RUNNING VALIDATIONS 26
3.5. CREATING A VALIDATION 27
3.6. VIEWING VALIDATION HISTORY 28
3.7. VALIDATION FRAMEWORK LOG FORMAT 28
3.8. VALIDATION FRAMEWORK LOG OUTPUT FORMATS 29
3.9. IN-FLIGHT VALIDATIONS 30

CHAPTER 4. ADDITIONAL INTROSPECTION OPERATIONS 31
4.1. PERFORMING INDIVIDUAL NODE INTROSPECTION 31
4.2. PERFORMING NODE INTROSPECTION AFTER INITIAL INTROSPECTION 31
4.3. PERFORMING NETWORK INTROSPECTION FOR INTERFACE INFORMATION 31
4.4. RETRIEVING HARDWARE INTROSPECTION DETAILS 33

CHAPTER 5. AUTOMATICALLY DISCOVERING BARE METAL NODES 38
5.1. ENABLING AUTO-DISCOVERY 38
5.2. TESTING AUTO-DISCOVERY 38
5.3. USING RULES TO DISCOVER DIFFERENT VENDOR HARDWARE 39

CHAPTER 6. CONFIGURING AUTOMATIC PROFILE TAGGING 41
6.1. POLICY FILE SYNTAX 41
6.2. POLICY FILE EXAMPLE 43
6.3. IMPORTING POLICY FILES INTO DIRECTOR 44

CHAPTER 7. CUSTOMIZING CONTAINER IMAGES 46
7.1. PREPARING CONTAINER IMAGES FOR DIRECTOR INSTALLATION 46
7.1.1. Container image preparation parameters 46
7.1.2. Guidelines for container image tagging 49

7.1.3. Excluding Ceph Storage container images 51
7.1.4. Modifying images during preparation 51
7.1.5. Updating existing packages on container images 52
7.1.6. Installing additional RPM files to container images 53
7.1.7. Modifying container images with a custom Dockerfile 54
7.1.8. Preparing a Satellite server for container images 54
7.1.9. Deploying a vendor plugin 59
7.2. PERFORMING ADVANCED CONTAINER IMAGE MANAGEMENT 60
7.2.1. Pinning container images for the undercloud 60
7.2.2. Pinning container images for the overcloud 61

CHAPTER 8. CUSTOMIZING NETWORKS FOR THE RED HAT OPENSTACK PLATFORM ENVIRONMENT 64
8.1. CUSTOMIZING UNDERCLOUD NETWORKS 64
8.1.1. Configuring undercloud network interfaces 64
8.1.2. Configuring the undercloud for bare metal provisioning over IPv6 66
8.2. CUSTOMIZING OVERCLOUD NETWORKS 69
8.2.1. Defining custom network interface templates 69
8.2.1.1. Creating a custom NIC template 69
8.2.1.2. Network interface configuration options 70
8.2.1.3. Example custom network interfaces 79
8.2.1.4. Customizing NIC mappings for pre-provisioned nodes 81
8.2.2. Composable networks 83
8.2.2.1. Adding a composable network 84
8.2.2.2. Including a composable network in a role 87
8.2.2.3. Assigning OpenStack services to composable networks 87
8.2.2.4. Enabling custom composable networks 88
8.2.2.5. Renaming the default networks 89
8.2.3. Additional overcloud network configuration 90
8.2.3.1. Configuring routes and default routes 90
8.2.3.2. Configuring policy-based routing 90
8.2.3.3. Configuring jumbo frames 92
8.2.3.4. Configuring ML2/OVN northbound path MTU discovery for jumbo frame fragmentation 93
8.2.3.5. Configuring the native VLAN on a trunked interface 94
8.2.3.6. Increasing the maximum number of connections that netfilter tracks 95
8.2.4. Network interface bonding 96
8.2.4.1. Network interface bonding for overcloud nodes 96
8.2.4.2. Creating Open vSwitch (OVS) bonds 97
8.2.4.3. Open vSwitch (OVS) bonding options 97
8.2.4.4. Using Link Aggregation Control Protocol (LACP) with Open vSwitch (OVS) bonding modes 98
8.2.4.5. Creating Linux bonds 100
8.2.5. Updating the format of your network configuration files 101
8.2.5.1. Updating the format of your network configuration file 102
8.2.5.2. Automatically converting NIC templates to Jinja2 Ansible format 102
8.2.5.3. Manually converting NIC templates to Jinja2 Ansible format 104
8.2.5.4. Heat parameter to Ansible variable mappings 105
8.2.5.5. Heat parameter to provisioning definition file mappings 108
8.2.5.6. Changes to the network data schema 109

CHAPTER 9. CONFIGURING AND MANAGING RED HAT OPENSTACK PLATFORM WITH ANSIBLE 111
9.1. ANSIBLE-BASED OVERCLOUD REGISTRATION 111
9.1.1. Red Hat Subscription Manager (RHSM) composable service 111
9.1.2. RhsmVars sub-parameters 113
9.1.3. Registering the overcloud with the rhsm composable service 114

9.1.4. Applying the rhsm composable service to different roles 116
9.1.5. Registering the overcloud to Red Hat Satellite Server 117
9.1.6. Switching to the rhsm composable service 118
9.1.7. rhel-registration to rhsm mappings 119
9.1.8. Deploying the overcloud with the rhsm composable service 119
9.1.9. Running Ansible-based registration manually 120
9.2. CONFIGURING THE OVERCLOUD WITH ANSIBLE 121
9.2.1. Ansible-based overcloud configuration (config-download) 121
9.2.2. config-download working directory 122
9.2.3. Checking config-download log 122
9.2.4. Performing Git operations on the working directory 122
9.2.5. Deployment methods that use config-download 123
9.2.6. Running config-download on a standard deployment 124
9.2.7. Running config-download with separate provisioning and configuration 124
9.2.8. Running config-download with the ansible-playbook-command.sh script 125
9.2.9. Running config-download with manually created playbooks 127
9.2.10. Limitations of config-download 130
9.2.11. config-download top level files 131
9.2.12. config-download tags 131
9.2.13. config-download deployment steps 132
9.3. MANAGING CONTAINERS WITH ANSIBLE 133
9.3.1. tripleo-container-manage role defaults and variables 134
9.3.2. tripleo-container-manage molecule scenarios 134
9.3.3. tripleo_container_manage role variables 135
9.3.4. tripleo-container-manage healthchecks 137
9.3.5. tripleo-container-manage debug 138

CHAPTER 10. CONFIGURING THE OVERCLOUD WITH THE ORCHESTRATION SERVICE (HEAT) 140
10.1. UNDERSTANDING HEAT TEMPLATES 140
10.1.1. heat templates 140
10.1.2. Environment files 141
10.1.3. Core overcloud heat templates 142
10.1.4. Including environment files in overcloud creation 143
10.1.5. Using customized core heat templates 144
10.1.6. Jinja2 rendering 147
10.2. HEAT PARAMETERS 149
10.2.1. Example 1: Configuring the time zone 149
10.2.2. Example 2: Configuring RabbitMQ file descriptor limit 150
10.2.3. Example 3: Enabling and disabling parameters 150
10.2.4. Example 4: Role-based parameters 150
10.2.5. Identifying parameters that you want to modify 150
10.3. CONFIGURATION HOOKS 152
10.3.1. Pre-configuration: customizing specific overcloud roles 152
10.3.2. Pre-configuration: customizing all overcloud roles 154
10.3.3. Post-configuration: customizing all overcloud roles 156
10.3.4. Puppet: Customizing hieradata for roles 158
10.3.5. Puppet: Customizing hieradata for individual nodes 159
10.3.6. Puppet: Applying custom manifests 160

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.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: Create Issue
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

CHAPTER 1. PLANNING CUSTOM UNDERCLOUD FEATURES

Before you configure and install director on the undercloud, you can plan to include custom features in your undercloud.

1.1. CHARACTER ENCODING CONFIGURATION

Red Hat OpenStack Platform has special character encoding requirements as part of the locale settings:

- Use UTF-8 encoding on all nodes. Ensure the **LANG** environment variable is set to **en_US.UTF-8** on all nodes.
- Avoid using non-ASCII characters if you use Red Hat Ansible Tower to automate the creation of Red Hat OpenStack Platform resources.

1.2. CONSIDERATIONS WHEN RUNNING THE UNDERCLOUD WITH A PROXY

Running the undercloud with a proxy has certain limitations, and Red Hat recommends that you use Red Hat Satellite for registry and package management.

However, if your environment uses a proxy, review these considerations to best understand the different configuration methods of integrating parts of Red Hat OpenStack Platform with a proxy and the limitations of each method.

System-wide proxy configuration

Use this method to configure proxy communication for all network traffic on the undercloud. To configure the proxy settings, edit the **/etc/environment** file and set the following environment variables:

http_proxy
: The proxy that you want to use for standard HTTP requests.

https_proxy
: The proxy that you want to use for HTTPs requests.

no_proxy
: A comma-separated list of domains that you want to exclude from proxy communications.

The system-wide proxy method has the following limitations:

- The maximum length of **no_proxy** is 1024 characters due to a fixed size buffer in the **pam_env** PAM module.

dnf proxy configuration

Use this method to configure **dnf** to run all traffic through a proxy. To configure the proxy settings, edit the **/etc/dnf/dnf.conf** file and set the following parameters:

proxy
: The URL of the proxy server.

proxy_username
: The username that you want to use to connect to the proxy server.

proxy_password

The password that you want to use to connect to the proxy server.

proxy_auth_method

The authentication method used by the proxy server.

For more information about these options, run **man dnf.conf**.

The **dnf** proxy method has the following limitations:

- This method provides proxy support only for **dnf**.
- The **dnf** proxy method does not include an option to exclude certain hosts from proxy communication.

Red Hat Subscription Manager proxy

Use this method to configure Red Hat Subscription Manager to run all traffic through a proxy. To configure the proxy settings, edit the **/etc/rhsm/rhsm.conf** file and set the following parameters:

proxy_hostname

Host for the proxy.

proxy_scheme

The scheme for the proxy when writing out the proxy to repo definitions.

proxy_port

The port for the proxy.

proxy_username

The username that you want to use to connect to the proxy server.

proxy_password

The password to use for connecting to the proxy server.

no_proxy

A comma-separated list of hostname suffixes for specific hosts that you want to exclude from proxy communication.

For more information about these options, run **man rhsm.conf**.

The Red Hat Subscription Manager proxy method has the following limitations:

- This method provides proxy support only for Red Hat Subscription Manager.
- The values for the Red Hat Subscription Manager proxy configuration override any values set for the system-wide environment variables.

Transparent proxy

If your network uses a transparent proxy to manage application layer traffic, you do not need to configure the undercloud itself to interact with the proxy because proxy management occurs automatically. A transparent proxy can help overcome limitations associated with client-based proxy configuration in Red Hat OpenStack Platform.

CHAPTER 2. 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.

2.1. 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.

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.

2.2. 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
```

2.3. 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 the **openstack overcloud role generate** command to join multiple predefined roles and automatically generate a custom **roles_data** file.

Procedure

1. List the default role templates:

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

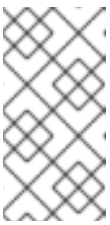
2. View the role definition:

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

3. Generate a custom **roles_data.yaml** file that contains the **Controller**, **Compute**, and **Networker** roles:

```
$ openstack overcloud roles \
 generate -o <custom_role_file> \
 Controller Compute Networker
```

- Replace **<custom_role_file>** with the name and location of the new role file to generate, for example, **/home/stack/templates/roles_data.yaml**.



NOTE

The **Controller** and **Networker** roles 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.yaml** file from your own custom roles.

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

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/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 role \
 generate -o my_roles_data.yaml \
 --roles-path /home/stack/templates/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.

2.4. 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**

Role	Description	File
ComputeHCI	Compute node with hyper-converged infrastructure. Includes Compute and Ceph OSD services.	**ComputeHCI.yaml**
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**
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**

2.5. 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 2.10, "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.

2.6. CREATING A NEW ROLE

You can use the composable service architecture to assign roles to bare-metal nodes 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**).

Procedure

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

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

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

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

4. Create a new file named **Horizon.yaml** in **home/stack/templates/roles**.
5. Add the following configuration to **Horizon.yaml** to create a new **Horizon** role that contains the base and core OpenStack Dashboard services:

```
- name: Horizon 1
  CountDefault: 1 2
  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
```

1 Set the **name** parameter to the name of the custom role. Custom role names have a maximum length of 47 characters.

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

6. 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
    ...
```

7. Generate a new roles data file named **roles_data_horizon.yaml** that includes the **Controller**, **Compute**, and **Horizon** roles:

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_horizon.yaml \
```

```
  --roles-path /home/stack/templates/roles \
  Controller Compute Horizon
```

8. Optional: Edit the **overcloud-baremetal-deploy.yaml** node definition file to configure the placement of the Horizon node:

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  ...
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  ...
- name: Horizon
  count: 1
  instances:
  - hostname: overcloud-horizon-0
    name: node07
```

2.7. 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.

2.8. 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
```

2.9. 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, 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.

2.10. 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([])}}
```

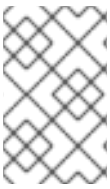
For the default roles, this creates the following service list parameters: **ControllerServices**, **ComputeServices**, **BlockStorageServices**, **ObjectStorageServices**, and **CephStorageServices**.

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.

2.11. 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
```

2.12. 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
```

CHAPTER 3. USING THE VALIDATION FRAMEWORK

Red Hat OpenStack Platform (RHOSP) includes a validation framework that you can use to verify the requirements and functionality of the undercloud and overcloud. The framework includes two types of validations:

- Manual Ansible-based validations, which you execute through the **validation** command set.
- Automatic in-flight validations, which execute during the deployment process.

You must understand which validations you want to run, and skip validations that are not relevant to your environment. For example, the pre-deployment validation includes a test for TLS-everywhere. If you do not intend to configure your environment for TLS-everywhere, this test fails. Use the **--validation** option in the **validation run** command to refine the validation according to your environment.

3.1. ANSIBLE-BASED VALIDATIONS

During the installation of Red Hat OpenStack Platform (RHOSP) director, director also installs a set of playbooks from the **openstack-tripleo-validations** package. Each playbook contains tests for certain system requirements and a set of groups that define when to run the test:

no-op

Validations that run a *no-op* (no operation) task to verify to workflow functions correctly. These validations run on both the undercloud and overcloud.

prep

Validations that check the hardware configuration of the undercloud node. Run these validation before you run the **openstack undercloud install** command.

openshift-on-openstack

Validations that check that the environment meets the requirements to be able to deploy OpenShift on OpenStack.

pre-introspection

Validations to run before the nodes introspection using Ironic Inspector.

pre-deployment

Validations to run before the **openstack overcloud deploy** command.

post-deployment

Validations to run after the overcloud deployment has finished.

pre-update

Validations to validate your RHOSP deployment before an update.

post-update

Validations to validate your RHOSP deployment after an update.

pre-upgrade

Validations to validate your RHOSP deployment before an upgrade.

post-upgrade

Validations to validate your RHOSP deployment after an upgrade.

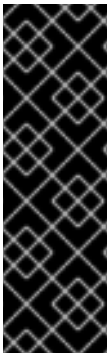
3.2. CHANGING THE VALIDATION CONFIGURATION FILE

The validation configuration file is a .ini file that you can edit to control every aspect of the validation execution and the communication between remote machines.

You can change the default configuration values in one of the following ways:

- Edit the default /etc/validations.cfg file.
- Make your own copy of the default **/etc/validations.cfg** file, edit the copy, and provide it through the CLI with the **--config** argument. If you create your own copy of the configuration file, point the CLI to this file on each execution with **--config**.

By default, the location of the validation configuration file is **/etc/validation.cfg**.



IMPORTANT

Ensure that you correctly edit the configuration file or your validation might fail with errors, for example:

- undetected validations
- callbacks written to different locations
- incorrectly-parsed logs

Prerequisites

- You have a thorough understanding of how to validate your environment.

Procedure

1. Optional: Make a copy of the validation configuration file for editing:
   a. Copy **/etc/validation.cfg** to your home directory.
   b. Make the required edits to the new configuration file.
2. Run the validation command:

```
$ validation run --config <configuration-file>
```

   - Replace <configuration-file> with the file path to the configuration file that you want to use.

NOTE

When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

3.3. LISTING VALIDATIONS

Run the **validation list** command to list the different types of validations available.

Procedure

1. Source the **stackrc** file.

```
$ source ~/stackrc
```

2. Run the **validation list** command:
   - To list all validations, run the command without any options:

     ```
     $ validation list
     ```

   - To list validations in a group, run the command with the **--group** option:

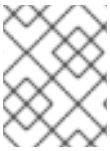
     ```
     $ validation list --group prep
     ```

NOTE

For a full list of options, run **validation list --help**.

3.4. RUNNING VALIDATIONS

To run a validation or validation group, use the **validation run** command. To see a full list of options, use the **validation run --help** command.



NOTE

When you run a validation, the **Reasons** column in the output is limited to 79 characters. To view the validation result in full, view the validation log files.

Procedure

1. Source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. Validate a static inventory file called **tripleo-ansible-inventory.yaml**.

   ```
   $ validation run --group pre-introspection -i tripleo-ansible-inventory.yaml
   ```

NOTE

You can find the inventory file in the **~/tripleo-deploy/<stack>** directory for a standalone or undercloud deployment or in the **~/overcloud-deploy/<stack>** directory for an overcloud deployment.

3. Enter the **validation run** command:
   - To run a single validation, enter the command with the **--validation** option and the name of the validation. For example, to check the memory requirements of each node, enter **--validation check-ram**:

     ```
     $ validation run --validation check-ram
     ```

To run multiple specific validations, use the **--validation** option with a comma-separated list of the validations that you want to run. For more information about viewing the list of available validations, see Listing validations.

- To run all validations in a group, enter the command with the **--group** option:

```
$ validation run --group prep
```

To view detailed output from a specific validation, run the **validation history get --full** command against the UUID of the specific validation from the report:

```
$ validation history get --full <UUID>
```

3.5. CREATING A VALIDATION

You can create a validation with the **validation init** command. Execution of the command results in a basic template for a new validation. You can edit the new validation role to suit your requirements.

IMPORTANT

Red Hat does not support user-created validations.

Prerequisites

- You have a thorough understanding of how to validate your environment.
- You have access rights to the directory where you run the command.

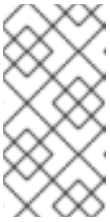
Procedure

1. Create your validation:

```
$ validation init <my-new-validation>
```

- Replace **<my-new-validation>** with the name of your new validation.
The execution of this command results in the creation of the following directory and sub-directories:

```
/home/stack/community-validations
├── library
├── lookup_plugins
├── playbooks
└── roles
```



NOTE

If you see the error message "The Community Validations are disabled by default, ensure that the **enable_community_validations** parameter is set to **True** in the validation configuration file. The default name and location of this file is **/etc/validation.cfg**.

2. Edit the role to suit your requirements.

Additional resources

- Section 3.2, “Changing the validation configuration file”.

3.6. VIEWING VALIDATION HISTORY

Director saves the results of each validation after you run a validation or group of validations. View past validation results with the **validation history list** command.

Prerequisites

- You have run a validation or group of validations.

Procedure

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

```
$ source ~/stackrc
```

3. You can view a list of all validations or the most recent validations:
   - View a list of all validations:

```
$ validation history list
```

   - View history for a specific validation type by using the **--validation** option:

```
$ validation history get --validation <validation-type>
```

     - Replace *<validation-type>* with the type of validation, for example, ntp.

4. View the log for a specific validation UUID:

```
$ validation show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

Additional resources

- assembly_using-the-validation-framework[Using the validation framework]

3.7. VALIDATION FRAMEWORK LOG FORMAT

After you run a validation or group of validations, director saves a JSON-formatted log from each validation in the **/var/logs/validations** directory. You can view the file manually or use the **validation history get --full** command to display the log for a specific validation UUID.

Each validation log file follows a specific format:

- **<UUID>_<Name>_<Time>**

  UUID

  The Ansible UUID for the validation.

Name

The Ansible name for the validation.

Time

The start date and time for when you ran the validation.

Each validation log contains three main parts:

- plays
- stats
- validation_output

plays

The **plays** section contains information about the tasks that the director performed as part of the validation:

play

A play is a group of tasks. Each **play** section contains information about that particular group of tasks, including the start and end times, the duration, the host groups for the play, and the validation ID and path.

tasks

The individual Ansible tasks that director runs to perform the validation. Each **tasks** section contains a **hosts** section, which contains the action that occurred on each individual host and the results from the execution of the actions. The **tasks** section also contains a **task** section, which contains the duration of the task.

stats

The **stats** section contains a basic summary of the outcome of all tasks on each host, such as the tasks that succeeded and failed.

validation_output

If any tasks failed or caused a warning message during a validation, the **validation_output** contains the output of that failure or warning.

3.8. VALIDATION FRAMEWORK LOG OUTPUT FORMATS

The default behaviour of the validation framework is to save validation logs in JSON format. You can change the output of the logs with the **ANSIBLE_STDOUT_CALLBACK** environment variable.

To change the validation output log format, run a validation and include the **--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** option:

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- Replace **<callback>** with an Ansible output callback. To view a list of the standard Ansible output callbacks, run the following command:

```
$ ansible-doc -t callback -l
```

The validation framework includes the following additional callbacks:

validation_json

The framework saves JSON-formatted validation results as a log file in **/var/logs/validations**. This is the default callback for the validation framework.

validation_stdout

The framework displays JSON-formatted validation results on screen.

http_json

The framework sends JSON-formatted validation results to an external logging server. You must also include additional environment variables for this callback:

HTTP_JSON_SERVER

The URL for the external server.

HTTP_JSON_PORT

The port for the API entry point of the external server. The default port in 8989.

Set these environment variables with additional **--extra-env-vars** options:

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
    --extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
    --extra-env-vars HTTP_JSON_PORT=8989 \
    --validation check-ram
```

IMPORTANT

Before you use the **http_json** callback, you must add **http_json** to the **callback_whitelist** parameter in your **ansible.cfg** file:

```
callback_whitelist = http_json
```

3.9. IN-FLIGHT VALIDATIONS

Red Hat OpenStack Platform (RHOSP) includes in-flight validations in the templates of composable services. In-flight validations verify the operational status of services at key steps of the overcloud deployment process.

In-flight validations run automatically as part of the deployment process. Some in-flight validations also use the roles from the **openstack-tripleo-validations** package.

CHAPTER 4. ADDITIONAL INTROSPECTION OPERATIONS

In some situations, you might want to perform introspection outside of the standard overcloud deployment workflow. For example, you might want to introspect new nodes or refresh introspection data after replacing hardware on existing unused nodes.

4.1. PERFORMING INDIVIDUAL NODE INTROSPECTION

To perform a single introspection on an available node, set the node to management mode and perform the introspection.

Procedure

1. Set all nodes to a **manageable** state:

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. Perform the introspection:

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

   After the introspection completes, the node changes to an **available** state.

4.2. PERFORMING NODE INTROSPECTION AFTER INITIAL INTROSPECTION

After an initial introspection, all nodes enter an **available** state due to the **--provide** option. To perform introspection on all nodes after the initial introspection, set the node to management mode and perform the introspection.

Procedure

1. Set all nodes to a **manageable** state

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do
openstack baremetal node manage $node ; done
```

2. Run the bulk introspection command:

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

   After the introspection completes, all nodes change to an **available** state.

4.3. PERFORMING NETWORK INTROSPECTION FOR INTERFACE INFORMATION

Network introspection retrieves link layer discovery protocol (LLDP) data from network switches. The following commands show a subset of LLDP information for all interfaces on a node, or full information for a particular node and interface. This can be useful for troubleshooting. Director enables LLDP data collection by default.

Procedure

1. To get a list of interfaces on a node, run the following command:

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

For example:

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-79f8b86c7cd9
+-----------+-------------------+------------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs   | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+------------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                  | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                  | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]             | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+------------------------+-------------------+----------------+
```

2. To view interface data and switch port information, run the following command:

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID] [INTERFACE]
```

For example:

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-907d-79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------------------------+
| Field                                | Value                                                                                          |
+--------------------------------------+------------------------------------------------------------------------------------------------+
| interface                            | p2p1                                                                                           |
| mac                                  | 00:0a:f7:79:93:18                                                                              |
| node_ident                           | c89397b7-a326-41a0-907d-79f8b86c7cd9                                                           |
| switch_capabilities_enabled          | [u'Bridge', u'Router']                                                                         |
| switch_capabilities_support          | [u'Bridge', u'Router']                                                                         |
| switch_chassis_id                    | 64:64:9b:31:12:00                                                                              |
| switch_port_autonegotiation_enabled  | True                                                                                           |
| switch_port_autonegotiation_support  | True                                                                                           |
| switch_port_description              | ge-0/0/2.0                                                                                     |
| switch_port_id                       | 507                                                                                            |
| switch_port_link_aggregation_enabled | False                                                                                          |
```

```
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type                 | Unknown
|
| switch_port_mtu                      | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids                 | [101]
|
| switch_port_vlans                    | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities           | None
|
| switch_system_name                   | rhos-compute-node-sw1
|
+--------------------------------------+----------------------------------------------------------------------------
----------------------------------------------+
```

4.4. RETRIEVING HARDWARE INTROSPECTION DETAILS

The Bare Metal service hardware-inspection-extras feature is enabled by default, and you can use it to retrieve hardware details for overcloud configuration. For more information about the **inspection_extras** parameter in the **undercloud.conf** file, see Director configuration parameters.

For example, the **numa_topology** collector is part of the hardware-inspection extras and includes the following information for each NUMA node:

- RAM (in kilobytes)
- Physical CPU cores and their sibling threads
- NICs associated with the NUMA node

Procedure

- To retrieve the information listed above, substitute <UUID> with the UUID of the bare-metal node to complete the following command:

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

The following example shows the retrieved NUMA information for a bare-metal node:

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0
    },
    {
      "cpu": 1,
      "thread_siblings": [
        9,
        25
      ],
```

```
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "numa_node": 1
  },
  {
    "cpu": 5,
    "thread_siblings": [
      5,
      21
    ],
    "numa_node": 0
  },
  {
    "cpu": 4,
    "thread_siblings": [
      12,
      28
    ],
    "numa_node": 1
  },
  {
    "cpu": 4,
    "thread_siblings": [
      4,
      20
    ],
    "numa_node": 0
  },
  {
    "cpu": 0,
    "thread_siblings": [
      8,
      24
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      14,
      30
    ],
```

```
    "numa_node": 1
   },
   {
    "cpu": 3,
    "thread_siblings": [
     3,
     19
    ],
    "numa_node": 0
   },
   {
    "cpu": 2,
    "thread_siblings": [
     2,
     18
    ],
    "numa_node": 0
   }
  ],
  "ram": [
   {
    "size_kb": 66980172,
    "numa_node": 0
   },
   {
    "size_kb": 67108864,
    "numa_node": 1
   }
  ],
  "nics": [
   {
    "name": "ens3f1",
    "numa_node": 1
   },
   {
    "name": "ens3f0",
    "numa_node": 1
   },
   {
    "name": "ens2f0",
    "numa_node": 0
   },
   {
    "name": "ens2f1",
    "numa_node": 0
   },
   {
    "name": "ens1f1",
    "numa_node": 0
   },
   {
    "name": "ens1f0",
    "numa_node": 0
   },
   {
    "name": "eno4",
```

```
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

CHAPTER 5. AUTOMATICALLY DISCOVERING BARE METAL NODES

You can use auto-discovery to register overcloud nodes and generate their metadata, without the need to create an **instackenv.json** file. This improvement can help to reduce the time it takes to collect information about a node. For example, if you use auto-discovery, you do not to collate the IPMI IP addresses and subsequently create the **instackenv.json**.

5.1. ENABLING AUTO-DISCOVERY

Enable and configure Bare Metal auto-discovery to automatically discover and import nodes that join your provisioning network when booting with PXE.

Procedure

1. Enable Bare Metal auto-discovery in the **undercloud.conf** file:

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

   - **enable_node_discovery** - When enabled, any node that boots the introspection ramdisk using PXE is enrolled in the Bare Metal service (ironic) automatically.
   - **discovery_default_driver** - Sets the driver to use for discovered nodes. For example, **ipmi**.

2. Add your IPMI credentials to ironic:

   a. Add your IPMI credentials to a file named **ipmi-credentials.json**. Replace the **SampleUsername**, **RedactedSecurePassword**, and **bmc_address** values in this example to suit your environment:

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  }
]
```

3. Import the IPMI credentials file into ironic:

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

5.2. TESTING AUTO-DISCOVERY

PXE boot a node that is connected to your provisioning network to test the Bare Metal auto-discovery feature.

Procedure

1. Power on the required nodes.
2. Run the **openstack baremetal node list** command. You should see the new nodes listed in an **enrolled** state:

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. Set the resource class for each node:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. Configure the kernel and ramdisk for each node:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. Set all nodes to available:

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node provide $NODE ; done
```

5.3. USING RULES TO DISCOVER DIFFERENT VENDOR HARDWARE

If you have a heterogeneous hardware environment, you can use introspection rules to assign credentials and remote management credentials. For example, you might want a separate discovery rule to handle your Dell nodes that use DRAC.

Procedure

1. Create a file named **dell-drac-rules.json** with the following contents:

```
[
  {
    "description": "Set default IPMI credentials",
```

```
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver_info/ipmi_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/ipmi_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/ipmi_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    },
    {
        "description": "Set the vendor driver for Dell hardware",
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver", "value": "idrac"},
            {"action": "set-attribute", "path": "driver_info/drac_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/drac_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/drac_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    }
]
```

- Replace the user name and password values in this example to suit your environment:

2. Import the rule into ironic:

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

CHAPTER 6. CONFIGURING AUTOMATIC PROFILE TAGGING

The introspection process performs a series of benchmark tests. Director saves the data from these tests. You can create a set of policies that use this data in various ways:

- The policies can identify under-performing or unstable nodes and isolate these nodes from use in the overcloud.
- The policies can define whether to tag nodes into specific profiles automatically.

6.1. POLICY FILE SYNTAX

Policy files use a JSON format that contains a set of rules. Each rule defines a description, a condition, and an action. A **description** is a plain text description of the rule, a **condition** defines an evaluation using a key-value pattern, and an **action** is the performance of the condition.

Description

A description is a plain text description of the rule.

Example:

```
"description": "A new rule for my node tagging policy"
```

Conditions

A condition defines an evaluation using the following key-value pattern:

field

Defines the field to evaluate:

- **memory_mb** - The amount of memory for the node in MB.
- **cpus** - The total number of threads for the node CPU.
- **cpu_arch** - The architecture of the node CPU.
- **local_gb** - The total storage space of the node root disk.

op

Defines the operation to use for the evaluation. This includes the following attributes:

- **eq** - Equal to
- **ne** - Not equal to
- **lt** - Less than
- **gt** - Greater than
- **le** - Less than or equal to
- **ge** - Greater than or equal to
- **in-net** - Checks that an IP address is in a given network

- **matches** - Requires a full match against a given regular expression
- **contains** - Requires a value to contain a given regular expression
- **is-empty** - Checks that **field** is empty

invert

Boolean value to define whether to invert the result of the evaluation.

multiple

Defines the evaluation to use if multiple results exist. This parameter includes the following attributes:

- **any** - Requires any result to match
- **all** - Requires all results to match
- **first** - Requires the first result to match

value

Defines the value in the evaluation. If the field and operation result in the value, the condition return a true result. Otherwise, the condition returns a false result.

Example:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

Actions

If a condition is **true**, the policy performs an action. The action uses the **action** key and additional keys depending on the value of **action**:

- **fail** - Fails the introspection. Requires a **message** parameter for the failure message.
- **set-attribute** - Sets an attribute on an ironic node. Requires a **path** field, which is the path to an ironic attribute (for example, **/driver_info/ipmi_address**), and a **value** to set.
- **set-capability** - Sets a capability on an ironic node. Requires **name** and **value** fields, which are the name and the value for a new capability. This replaces the existing value for this capability. For example, use this to define node profiles.
- **extend-attribute** - The same as **set-attribute** but treats the existing value as a list and appends value to it. If the optional **unique** parameter is set to True, nothing is added if the given value is already in a list.

Example:

```
"actions": [
  {
    "action": "set-capability",
```

```
    "name": "profile",
    "value": "swift-storage"
  }
]
```

6.2. POLICY FILE EXAMPLE

The following is an example JSON file (**rules.json**) that contains introspection rules:

```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
        "op": "ge",
        "field": "local_gb",
        "value": 40
```

```
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

This example consists of three rules:

- Fail introspection if memory is lower than 4096 MiB. You can apply these types of rules if you want to exclude certain nodes from your cloud.
- Nodes with a hard drive size 1 TiB and bigger are assigned the swift-storage profile unconditionally.
- Nodes with a hard drive less than 1 TiB but more than 40 GiB can be either Compute or Controller nodes. You can assign two capabilities (**compute_profile** and **control_profile**) so that the **openstack overcloud profiles match** command can later make the final choice. For this process to succeed, you must remove the existing profile capability, otherwise the existing profile capability has priority.

The profile matching rules do not change any other nodes.

NOTE

Using introspection rules to assign the **profile** capability always overrides the existing value. However, **[PROFILE]_profile** capabilities are ignored for nodes that already have a profile capability.

6.3. IMPORTING POLICY FILES INTO DIRECTOR

To apply the policy rules you defined in your policy JSON file, you must import the policy file into director.

Procedure

1. Import the policy file into director:

```
$ openstack baremetal introspection rule import <policy_file>
```

- Replace **<policy_file>** with the name of your policy rule file, for example, **rules.json**.

2. Run the introspection process:

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

3. Retrieve the UUIDs of the nodes that the policy rules are applied to:

```
$ openstack baremetal node list
```

4. Confirm that the nodes have been assigned the profiles defined in your policy rule file:

```
$ openstack baremetal node show <node_uuid>
```

5. If you made a mistake in introspection rules, then delete all rules:

```
$ openstack baremetal introspection rule purge
```

CHAPTER 7. CUSTOMIZING CONTAINER IMAGES

Red Hat OpenStack Platform (RHOSP) services run in containers, therefore to deploy the RHOSP services you must obtain the container images. You can generate and customize the environment file that prepares the container images for your RHOSP deployment.

7.1. PREPARING CONTAINER IMAGES FOR DIRECTOR INSTALLATION

Red Hat supports the following methods for managing container images for your overcloud:

- Pulling container images from the Red Hat Container Catalog to the **image-serve** registry on the undercloud and then pulling the images from the **image-serve** registry. When you pull images to the undercloud first, you avoid multiple overcloud nodes simultaneously pulling container images over an external connection.
- Pulling container images from your Satellite 6 server. You can pull these images directly from the Satellite because the network traffic is internal.

The undercloud installation requires an environment file to determine where to obtain container images and how to store them. You generate a default container image preparation file when preparing for director installation. You can customize the default container image preparation file.

7.1.1. 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.
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.

Parameter	Description
modify_append_tag	String to append to the tag for the destination image. For example, if you pull an image with the tag 17.1.0-5.161 and set the **modify_append_tag** to **-hotfix**, the director tags the final image as 17.1.0-5.161-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.
pull_source	The source registry from where to pull the original container images.

Parameter	Description
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 17.1.0 and **release** might be set to **5.161**, which results in the tag 17.1.0-5.161. Director uses this parameter only if you have not defined **tag** in the **set** dictionary.

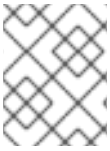
IMPORTANT

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.

7.1.2. 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 17.1.0 and the release for the container image is **5.161**, then the resulting tag for the container image is 17.1.0-5.161.

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 17.1 and 17.1.0 link to the latest **release** in the 17.1.0 container stream. If a new minor release of 17.1 occurs, the 17.1 tag links to the latest **release** for the new minor release stream while the 17.1.0 tag continues to link to the latest **release** within the 17.1.0 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 17.1. This always corresponds to the latest minor version and release.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1
      ...
```

- To change to a specific minor version for OpenStack Platform container images, set the tag to a minor version. For example, to change to 17.1.2, set **tag** to 17.1.2.

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1.2
      ...
```

- 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: 17.1
      ...
    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": "17.1.0",
  ...
}
```

- 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 17.1.0 set for **version** and 5.161 set for **release**, the resulting tag for the container image is 17.1.0-5.161.

- 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.

7.1.3. Excluding Ceph Storage container images

The default overcloud role configuration uses the default Controller, Compute, and Ceph Storage roles. However, if you use the default role configuration to deploy an overcloud without Ceph Storage nodes, director still pulls the Ceph Storage container images from the Red Hat Container Registry because the images are included as a part of the default configuration.

If your overcloud does not require Ceph Storage containers, you can configure director to not pull the Ceph Storage containers images from the Red Hat Container Registry.

Procedure

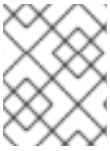
1. Edit the **containers-prepare-parameter.yaml** file and add the **ceph_images: false** parameter. The following is an example of this file with the parameter bolded:

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: {version}-{release}
    set:
      name_prefix: rhosp17-openstack-
      name_suffix: ''
      tag: 17.1_20231214.1
      rhel_containers: false
      neutron_driver: ovn
      ceph_images: false
    push_destination: true
```

2. Save the **containers-prepare-parameter.yaml** file.
3. Create a new container images file for use in the overcloud deployment:
   sudo openstack tripleo container image prepare -e containers-prepare-parameter.yaml --output-env-file <new_container_images_file>
   - Replace **<new_container_images_file>** with the output file that contains the new parameter.
4. Add the new container images file to the list of overcloud deployment environment files.

7.1.4. 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.

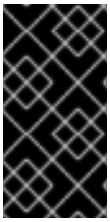
The Ansible role **tripleo-modify-image** conforms with the required role interface and provides the behaviour necessary for the modify use cases. Control the modification with the modify-specific keys in the **ContainerImagePrepare** parameter:

- **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
```

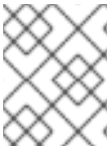


IMPORTANT

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.

7.1.5. Updating existing packages on container images

You can update the existing packages on the container images for Red Hat OpenStack Platform (RHOSP) containers.



NOTE

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

Procedure

1. Download the RPM packages for installation on the container images.
2. Edit the **containers-prepare-parameter.yaml** file to update all packages on the container images:

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

3. Save the **containers-prepare-parameter.yaml** file.
4. Include the **containers-prepare-parameter.yaml** file when you run the **openstack overcloud deploy** command.

7.1.6. 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 Performing a minor update of Red Hat OpenStack Platform .

Procedure

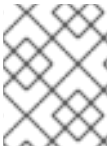
- The following example **ContainerImagePrepare** entry installs some hotfix packages on only the **nova-compute** image:

```
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
  ...
```

7.1.7. 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-rhel9/openstack-nova-compute:latest

USER "root"

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

USER "nova"
```

7.1.8. Preparing a Satellite server for container images

Red Hat Satellite 6 offers registry synchronization capabilities. This provides a method to pull multiple images into a Satellite server and manage them as part of an application life cycle. The Satellite also acts as a registry for other container-enabled systems to use. For more information about managing container images, see Managing Container Images in the *Red Hat Satellite 6 Content Management Guide*.

The examples in this procedure use the **hammer** command line tool for Red Hat Satellite 6 and an example organization called **ACME**. Substitute this organization for your own Satellite 6 organization.

NOTE

This procedure requires authentication credentials to access container images from **registry.redhat.io**. 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. For more information, see "Red Hat Container Registry Authentication".

Procedure

1. Create a list of all container images:

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel9" --format="{{ .Name }}" |
sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep
<ceph_dashboard_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep <ceph_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/openshift4" | grep ose-prometheus
```

   - Replace **<ceph_dashboard_image_file>** with the name of the image file for the version of Red Hat Ceph Storage that your deployment uses:
     - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
     - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**
   - Replace **<ceph_image_file>** with the name of the image file for the version of Red Hat Ceph Storage that your deployment uses:
     - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
     - Red Hat Ceph Storage 6: **rhceph-6-rhel9**

NOTE

The **openstack-ovn-bgp-agent** image is located at **registry.redhat.io/rhosp-rhel9/openstack-ovn-bgp-agent-rhel9:17.1**.

   - If you plan to install Ceph and enable the Ceph Dashboard, you need the following ose-prometheus containers:

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.12
registry.redhat.io/openshift4/ose-prometheus:v4.12
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.12
```

2. Copy the **satellite_images** file to a system that contains the Satellite 6 **hammer** tool. Alternatively, use the instructions in the *Hammer CLI Guide* to install the **hammer** tool to the undercloud.

3. Run the following **hammer** command to create a new product (**OSP Containers**) in your Satellite organization:

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

This custom product will contain your images.

4. Add the overcloud container images from the **satellite_images** file:

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

5. Add the Ceph Storage container image:

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

- Replace **<ceph_image_file>** with the name of the image file for the version of Red Hat Ceph Storage that your deployment uses:
  - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-rhel9**

NOTE

If you want to install the Ceph dashboard, include **--name <ceph_dashboard_image_name>** in the **hammer repository create** command:

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

- Replace **<ceph_dashboard_image_file>** with the name of the image file for the version of Red Hat Ceph Storage that your deployment uses:
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**

6. Synchronize the container images:

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

Wait for the Satellite server to complete synchronization.

NOTE

Depending on your configuration, **hammer** might ask for your Satellite server username and password. You can configure **hammer** to automatically login using a configuration file. For more information, see the Authentication section in the *Hammer CLI Guide*.

7. If your Satellite 6 server uses content views, create a new content view version to incorporate the images and promote it along environments in your application life cycle. This largely depends on how you structure your application lifecycle. For example, if you have an environment called **production** in your lifecycle and you want the container images to be available in that environment, create a content view that includes the container images and promote that content view to the **production** environment. For more information, see Managing Content Views.

8. Check the available tags for the **base** image:

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

This command displays tags for the OpenStack Platform container images within a content view for a particular environment.

9. Return to the undercloud and generate a default environment file that prepares images using your Satellite server as a source. Run the following example command to generate the environment file:

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

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

10. Edit the **containers-prepare-parameter.yaml** file and modify the following parameters:

- **push_destination** – Set this to **true** or **false** depending on your chosen container image management strategy. If you set this parameter to **false**, the overcloud nodes pull images directly from the Satellite. If you set this parameter to **true**, the director pulls the images from the Satellite to the undercloud registry and the overcloud pulls the images from the undercloud registry.

- **namespace** – The URL of the registry on the Satellite server.

- **name_prefix** – The prefix is based on a Satellite 6 convention. This differs depending on whether you use content views:

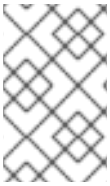
  - If you use content views, the structure is **[org]-[environment]-[content view]-[product]-**. For example: **acme-production-myosp17-osp_containers-**.

  - If you do not use content views, the structure is **[org]-[product]-**. For example: **acme-osp_containers-**.

- **ceph_namespace**, **ceph_image**, **ceph_tag** – If you use Ceph Storage, include these additional parameters to define the Ceph Storage container image location. Note that **ceph_image** now includes a Satellite-specific prefix. This prefix is the same value as the **name_prefix** option.

The following example environment file contains Satellite-specific parameters:

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp17_1-osp_containers-rhceph-6
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp17_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:5000
      neutron_driver: null
      tag: '17.1'
      ...
```



NOTE

To use a specific container image version stored on your Red Hat Satellite Server, set the **tag** key-value pair to the specific version in the **set** dictionary. For example, to use the 17.1.2 image stream, set **tag: 17.1.2** in the **set** dictionary.

You must define the **containers-prepare-parameter.yaml** environment file in the **undercloud.conf** configuration file, otherwise the undercloud uses the default values:

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

7.1.9. 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.

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-rhel9
        name_prefix: openstack-
        name_suffix: ''
        tag: 17.1
        ...
      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: 16.2-2
    ...
```

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.

7.2. PERFORMING ADVANCED CONTAINER IMAGE MANAGEMENT

The default container image configuration suits most environments. In some situations, your container image configuration might require some customization, such as version pinning.

7.2.1. Pinning container images for the undercloud

In certain circumstances, you might require a set of specific container image versions for your undercloud. In this situation, you must pin the images to a specific version. To pin your images, you must generate and modify a container configuration file, and then combine the undercloud roles data with the container configuration file to generate an environment file that contains a mapping of services to container images. Then include this environment file in the **custom_env_files** parameter in the **undercloud.conf** file.

Procedure

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

2. Run the **openstack tripleo container image prepare default** command with the **--output-env-file** option to generate a file that contains the default image configuration:

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

3. Modify the **undercloud-container-image-prepare.yaml** file according to the requirements of your environment.

   a. Remove the **tag:** parameter so that director can use the **tag_from_label:** parameter. Director uses this parameter to identify the latest version of each container image, pull each image, and tag each image on the container registry in director.

   b. Remove the Ceph labels for the undercloud.

   c. Ensure that the **neutron_driver:** parameter is empty. Do not set this parameter to **OVN** because OVN is not supported on the undercloud.

   d. Include your container image registry credentials:

```
ContainerImageRegistryCredentials:
  registry.redhat.io:
    myser: 'p@55w0rd!'
```

NOTE

You cannot push container images to the undercloud registry on new underclouds because the **image-serve** registry is not installed yet. You must set the **push_destination** value to **false**, or use a custom value, to pull images directly from source. For more information, see Container image preparation parameters.

4. Generate a new container image configuration file that uses the undercloud roles file combined with your custom **undercloud-container-image-prepare.yaml** file:

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

   The **undercloud-container-images.yaml** file is an environment file that contains a mapping of service parameters to container images. For example, OpenStack Identity (keystone) uses the **ContainerKeystoneImage** parameter to define its container image:

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
```

   Note that the container image tag matches the **{version}-{release}** format.

5. Include the **undercloud-container-images.yaml** file in the **custom_env_files** parameter in the **undercloud.conf** file. When you run the undercloud installation, the undercloud services use the pinned container image mapping from this file.

7.2.2. Pinning container images for the overcloud

In certain circumstances, you might require a set of specific container image versions for your overcloud. In this situation, you must pin the images to a specific version. To pin your images, you must create the **containers-prepare-parameter.yaml** file, use this file to pull your container images to the undercloud registry, and generate an environment file that contains a pinned image list.

For example, your **containers-prepare-parameter.yaml** file might contain the following content:

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel9
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
```

The **ContainerImagePrepare** parameter contains a single rule **set**. This rule **set** must not include the **tag** parameter and must rely on the **tag_from_label** parameter to identify the latest version and release of each container image. Director uses this rule **set** to identify the latest version of each container image, pull each image, and tag each image on the container registry in director.

Procedure

1. Run the **openstack tripleo container image prepare** command, which pulls all images from the source defined in the **containers-prepare-parameter.yaml** file. Include the **--output-env-file** to specify the output file that will contain the list of pinned container images:

   ```
   $ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
   ```

   The **overcloud-images.yaml** file is an environment file that contains a mapping of service parameters to container images. For example, OpenStack Identity (keystone) uses the **ContainerKeystoneImage** parameter to define its container image:

   ```
   ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
   ```

   Note that the container image tag matches the **{version}-{release}** format.

2. Include the **containers-prepare-parameter.yaml** and **overcloud-images.yaml** files in that specific order with your environment file collection when you run the **openstack overcloud deploy** command:

   ```
   $ openstack overcloud deploy --templates \
       ...
       -e /home/stack/containers-prepare-parameter.yaml \
       -e /home/stack/overcloud-images.yaml \
       ...
   ```

The overcloud services use the pinned images listed in the **overcloud-images.yaml** file.

CHAPTER 8. CUSTOMIZING NETWORKS FOR THE RED HAT OPENSTACK PLATFORM ENVIRONMENT

You can customizing the undercloud and overcloud physical networks for your Red Hat OpenStack Platform (RHOSP)environment.

8.1. CUSTOMIZING UNDERCLOUD NETWORKS

You can customize the undercloud network configuration to install the undercloud with specific networking functionality. You can also configure the undercloud and the provisioning network to use IPv6 instead of IPv4 if you have IPv6 nodes and infrastructure.

8.1.1. Configuring undercloud network interfaces

Include custom network configuration in the **undercloud.conf** file to install the undercloud with specific networking functionality. For example, some interfaces might not have DHCP. In this case, you must disable DHCP for these interfaces in the **undercloud.conf** file so that **os-net-config** can apply the configuration during the undercloud installation process.

Procedure

1. Log in to the undercloud host.

2. Create a new file **undercloud-os-net-config.yaml** and include the network configuration that you require.
   In the **addresses** section, include the **local_ip**, such as **172.20.0.1/26**. If TLS is enabled in the undercloud, you must also include the **undercloud_public_host**, such as **172.20.0.2/32**, and the **undercloud_admin_host**, such as **172.20.0.3/32**.

   Here is an example:

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - type: interface
    name: nic2
```

   To create a network bond for a specific interface, use the following sample:

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
```

```
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
      name: nic2
    - type: interface
      name: nic3
```

3. Include the path to the **undercloud-os-net-config.yaml** file in the **net_config_override** parameter in the **undercloud.conf** file:

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

NOTE

Director uses the file that you include in the **net_config_override** parameter as the template to generate the **/etc/os-net-config/config.yaml** file. **os-net-config** manages the interfaces that you define in the template, so you must perform all undercloud network interface customization in this file.

4. Install the undercloud.

Verification

- After the undercloud installation completes successfully, verify that the **/etc/os-net-config/config.yaml** file contains the relevant configuration:

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
```

```
- ip_netmask: 172.20.0.1/26
- ip_netmask: 172.20.0.2/32
- ip_netmask: 172.20.0.3/32
members:
- type: interface
  name: nic2
```

8.1.2. Configuring the undercloud for bare metal provisioning over IPv6

If you have IPv6 nodes and infrastructure, you can configure the undercloud and the provisioning network to use IPv6 instead of IPv4 so that director can provision and deploy Red Hat OpenStack Platform onto IPv6 nodes. However, there are some considerations:

- Dual stack IPv4/6 is not available.
- Tempest validations might not perform correctly.
- IPv4 to IPv6 migration is not available during upgrades.

Modify the **undercloud.conf** file to enable IPv6 provisioning in Red Hat OpenStack Platform.

Prerequisites

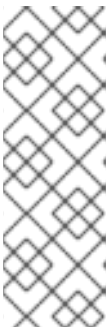
- An IPv6 address on the undercloud. For more information, see Configuring an IPv6 address on the undercloud in the *IPv6 networking for the overcloud* guide.

Procedure

1. Open your **undercloud.conf** file.
2. Specify the IPv6 address mode as either stateless or stateful:

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

   - Replace **<address_mode>** with **dhcpv6-stateless** or **dhcpv6-stateful**, based on the mode that your NIC supports.



NOTE

When you use the stateful address mode, the firmware, chain loaders, and operating systems might use different algorithms to generate an ID that the DHCP server tracks. DHCPv6 does not track addresses by MAC, and does not provide the same address back if the identifier value from the requester changes but the MAC address remains the same. Therefore, when you use stateful DHCPv6 you must also complete the next step to configure the network interface.

3. If you configured your undercloud to use stateful DHCPv6, specify the network interface to use for bare metal nodes:

```
[DEFAULT]
ipv6_address_mode = dhcpv6-stateful
```

```
ironic_enabled_network_interfaces = neutron,flat
...
```

4. Set the default network interface for bare metal nodes:

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

5. Specify whether or not the undercloud should create a router on the provisioning network:

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

- Replace **<true/false>** with **true** to enable routed networks and prevent the undercloud creating a router on the provisioning network. When **true**, the data center router must provide router advertisements.
- Replace **<true/false>** with **false** to disable routed networks and create a router on the provisioning network.

6. Configure the local IP address, and the IP address for the director Admin API and Public API endpoints over SSL/TLS:

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

- Replace **<ipv6_address>** with the IPv6 address of the undercloud.

7. Optional: Configure the provisioning network that director uses to manage instances:

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

- Replace **<ipv6_address>** with the IPv6 address of the network to use for managing instances when not using the default provisioning network.
- Replace **<ipv6_prefix>** with the IP address prefix of the network to use for managing instances when not using the default provisioning network.

8. Configure the DHCP allocation range for provisioning nodes:

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

- Replace **<ipv6_address_dhcp_start>** with the IPv6 address of the start of the network range to use for the overcloud nodes.
- Replace **<ipv6_address_dhcp_end>** with the IPv6 address of the end of the network range to use for the overcloud nodes.

9. Optional: Configure the gateway for forwarding traffic to the external network:

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

- Replace **<ipv6_gateway_address>** with the IPv6 address of the gateway when not using the default gateway.

10. Configure the DHCP range to use during the inspection process:

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

- Replace **<ipv6_address_inspection_start>** with the IPv6 address of the start of the network range to use during the inspection process.
- Replace **<ipv6_address_inspection_end>** with the IPv6 address of the end of the network range to use during the inspection process.

NOTE

This range must not overlap with the range defined by **dhcp_start** and **dhcp_end**, but must be in the same IP subnet.

11. Configure an IPv6 nameserver for the subnet:

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

- Replace **<ipv6_dns>** with the DNS nameservers specific to the subnet.

12. Use the **virt-customize** tool to modify the overcloud image to disable the **cloud-init** network configuration. For more information, see the Red Hat Knowledgebase solution Modifying the Red Hat Linux OpenStack Platform Overcloud Image with virt-customize.

8.2. CUSTOMIZING OVERCLOUD NETWORKS

You can customize the configuration of the physical network for your overcloud. For example, you can create configuration files for the network interface controllers (NICs) by using the NIC template file in Jinja2 ansible format, **j2**.

8.2.1. Defining custom network interface templates

You can create a set of custom network interface templates to define the NIC layout for each node in your overcloud environment. The overcloud core template collection contains a set of default NIC layouts for different use cases. You can create a custom NIC template by using a Jinja2 format file with a **.j2.yaml** extension. Director converts the Jinja2 files to YAML format during deployment.

You can then set the **network_config** property in the **overcloud-baremetal-deploy.yaml** node definition file to your custom NIC template to provision the networks for a specific node. For more information, see Provisioning bare metal nodes for the overcloud .

8.2.1.1. Creating a custom NIC template

Create a NIC template to customise the NIC layout for each node in your overcloud environment.

Procedure

1. Copy the sample network configuration template you require from **/usr/share/ansible/roles/tripleo_network_config/templates/** to your environment file directory:

```
$ cp /usr/share/ansible/roles/tripleo_network_config/templates/<sample_NIC_template>
/home/stack/templates/<NIC_template>
```

   - Replace **<sample_NIC_template>** with the name of the sample NIC template that you want to copy, for example, **single_nic_vlans/single_nic_vlans.j2**.
   - Replace **<NIC_template>** with the name of your custom NIC template file, for example, **single_nic_vlans.j2**.

2. Update the network configuration in your custom NIC template to match the requirements for your overcloud network environment. For information about the properties you can use to configure your NIC template, see Network interface configuration options . For an example NIC template, see Example custom network interfaces.

3. Create or update an existing environment file to enable your custom NIC configuration templates:

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
  CephStorageNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans_storage.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
```

4. If your overcloud uses the default internal load balancing, add the following configuration to your environment file to assign predictable virtual IPs for Redis and OVNDBs:

```
parameter_defaults:
  RedisVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
  OVNDBsVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
```

- Replace **<vip_address>** with an IP address from outside the allocation pool ranges.

8.2.1.2. Network interface configuration options

Use the following tables to understand the available options for configuring network interfaces.

interface

Defines a single network interface. The network interface **name** 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 such as **nic1** and **nic2**, instead of named interfaces such as **eth0** and **eno2**. 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.

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

Table 8.1. interface options

Option	Default	Description
name		Name of the interface. The network interface **name** uses either the actual interface name (**eth0**, **eth1**, **enp0s25**) or a set of numbered interfaces (**nic1**, **nic2**, **nic3**).
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.
routes		A list of routes assigned to the interface. For more information, see routes.

Option	Default	Description
mtu	1500	The maximum transmission unit (MTU) of the connection.
primary	False	Defines the interface as the primary interface.
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
    device: nic{{ loop.index + 1 }}
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

Table 8.2. 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.
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:

```
members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bond_interface_ovs_options }}
    members:
```

```
- type: interface
  name: nic2
  mtu: {{ min_viable_mtu }}
  primary: true
- type: interface
  name: nic3
  mtu: {{ min_viable_mtu }}
```

Table 8.3. ovs_bond options

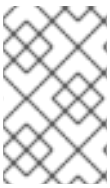
Option	Default	Description
name		Name of the bond.
use_dhcp	False	Use DHCP to get an IP address.
use_dhcpv6	False	Use DHCP to get a v6 IP address.
addresses		A list of IP addresses assigned to the bond.
routes		A list of routes assigned to the bond. For more information, see routes.
mtu	1500	The maximum 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.

Option	Default	Description
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.

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**.



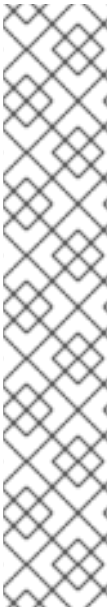
NOTE

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: br-bond
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bound_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
      primary: true
    - type: interface
      name: nic3
      mtu: {{ min_viable_mtu }}
```



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 8.4. 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
  mtu: {{ min_viable_mtu }}
  bonding_options: "mode=802.3ad lacp_rate=fast updelay=1000 miimon=100
xmit_hash_policy=layer3+4"
  members:
    type: interface
    name: ens1f0
    mtu: {{ min_viable_mtu }}
    primary: true
  type: interface
    name: ens1f1
    mtu: {{ min_viable_mtu }}
```

Table 8.5. 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
  mtu:
    get_attr: [MinViableMtu, value]
  use_dhcp: false
```

```
    dns_servers:
      get_param: DnsServers
    domain:
      get_param: DnsSearchDomains
    addresses:
    - ip_netmask:
        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
    routes:
      list_concat_unique:
        - get_param: ControlPlaneStaticRoutes
```

Table 8.6. 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: linux_bridge
  name: bridge_name
  ...
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
```

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.

8.2.1.3. Example custom network interfaces

The following examples illustrate how to customize network interface templates.

Separate control group and OVS bridge example

The following example Controller node NIC template configures the control group separate from the OVS bridge. The template uses five network interfaces and assigns a number of tagged VLAN devices to the numbered interfaces. The template creates the OVS bridges on **nic4** and **nic5**.

```
network_config:
- type: interface
  name: nic1
  mtu: {{ ctlplane_mtu }}
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
```

```
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
{% for network in role_networks if not network.startswith('Tenant') %}
- type: vlan
  device: bond_api
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: linux_bond
    name: bond-data
    mtu: {{ min_viable_mtu_dataplane }}
    bonding_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic4
      mtu: {{ min_viable_mtu_dataplane }}
      primary: true
    - type: interface
      name: nic5
      mtu: {{ min_viable_mtu_dataplane }}
{% for network in role_networks if network.startswith('Tenant') %}
  - type: vlan
    device: bond-data
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

Multiple NICs example

The following example uses a second NIC to connect to an infrastructure network with DHCP addresses and another NIC for the bond.

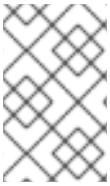
```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    mtu: {{ tenant_mtu }}
    use_dhcp: true
```

```
    primary: true
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: ovs_bridge
    name: br-bond
    mtu: {{ external_mtu }}
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
    members:
      - type: interface
        name: nic10
        mtu: {{ external_mtu }}
        use_dhcp: false
        primary: true
      - type: vlan
        mtu: {{ external_mtu }}
        vlan_id: {{ external_vlan_id }}
        addresses:
        - ip_netmask: {{ external_ip }}/{{ external_cidr }}
        routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

8.2.1.4. Customizing NIC mappings for pre-provisioned nodes

If you are using pre-provisioned nodes, you can specify the **os-net-config** mappings for specific nodes by using one of the following methods:

- Configure the **NetConfigDataLookup** heat parameter in an environment file, and run the **openstack overcloud node provision** command without **--network-config**.
- Configure the **net_config_data_lookup** property in your node definition file, **overcloud-baremetal-deploy.yaml**, and run the **openstack overcloud node provision** command with **--network-config**.



NOTE

If you are not using pre-provisioned nodes, you must configure the NIC mappings in your node definition file. For more information on configuring the **net_config_data_lookup** property, see Bare-metal node provisioning attributes.

You can assign aliases to the physical interfaces on each node to pre-determine which physical NIC maps to specific aliases, such as **nic1** or **nic2**, and you can map a MAC address to a specified alias. You can map specific nodes by using the MAC address or DMI keyword, or you can map a group of nodes by using a DMI keyword. The following examples configure three nodes and two node groups with aliases to the physical interfaces. 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.

Example 1: Configuring the NetConfigDataLookup parameter in os-net-config-mappings.yaml

```
NetConfigDataLookup:
```

```
node1: (1)
  nic1: "00:c8:7c:e6:f0:2e"
node2:
  nic1: "00:18:7d:99:0c:b6"
node3: (2)
  dmiString: "system-uuid" (3)
  id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
  nic1: em3
# Dell PowerEdge
nodegroup1: (4)
  dmiString: "system-product-name"
  id: "PowerEdge R630"
  nic1: em3
  nic2: em1
  nic3: em2
# Cisco UCS B200-M4"
nodegroup2:
  dmiString: "system-product-name"
  id: "UCSB-B200-M4"
  nic1: enp7s0
  nic2: enp6s0
```

(1) Maps **node1** to the specified MAC address, and assigns **nic1** as the alias for the MAC address on this node.

(2) Maps **node3** to the node with the system UUID "A8C85861-1B16-4803-8689-AFC62984F8F6", and assigns **nic1** as the alias for **em3** interface on this node.

(3) The **dmiString** parameter must be set to a valid string keyword. For a list of the valid string keywords, see the DMIDECODE(8) man page.

(4) Maps all the nodes in **nodegroup1** to nodes with the product name "PowerEdge R630", and assigns **nic1**, **nic2**, and **nic3** as the alias for the named interfaces on these nodes.

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.

Example 2: Configuring the net_config_data_lookup property in overcloud-baremetal-deploy.yaml - specific nodes

```
- name: Controller
  count: 3
  defaults:
    network_config:
      net_config_data_lookup:
        node1:
          nic1: "00:c8:7c:e6:f0:2e"
        node2:
          nic1: "00:18:7d:99:0c:b6"
        node3:
```

```
      dmiString: "system-uuid"
      id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
      nic1: em3
    # Dell PowerEdge
    nodegroup1:
      dmiString: "system-product-name"
      id: "PowerEdge R630"
      nic1: em3
      nic2: em1
      nic3: em2
    # Cisco UCS B200-M4"
    nodegroup2:
      dmiString: "system-product-name"
      id: "UCSB-B200-M4"
      nic1: enp7s0
      nic2: enp6s0
```

Example 3: Configuring the net_config_data_lookup property in overcloud-baremetal-deploy.yaml - all nodes for a role

```
- name: Controller
  count: 3
  defaults:
    network_config:
      template: templates/net_config_bridge.j2
      default_route_network:
      - external
  instances:
  - hostname: overcloud-controller-0
    network_config:
      <name/groupname>:
        nic1: 'XX:XX:XX:XX:XX:XX'
        nic2: 'YY:YY:YY:YY:YY:YY'
        nic3: 'ens1f0'
```

8.2.2. Composable networks

You can create custom composable networks if you want to host specific network traffic on different networks. Director provides a default network topology with network isolation enabled. You can find this configuration in the **/usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml**.

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

- Internal API
- Storage
- Storage management
- Tenant
- External

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 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.

8.2.2.1. 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. List the available sample network configuration files:

```
$ ll /usr/share/openstack-tripleo-heat-templates/network-data-samples/
-rw-r--r--. 1 root root 1554 May 11 23:04 default-network-isolation-ipv6.yaml
-rw-r--r--. 1 root root 1181 May 11 23:04 default-network-isolation.yaml
-rw-r--r--. 1 root root 1126 May 11 23:04 ganesha-ipv6.yaml
-rw-r--r--. 1 root root 1100 May 11 23:04 ganesha.yaml
-rw-r--r--. 1 root root 3556 May 11 23:04 legacy-routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2929 May 11 23:04 legacy-routed-networks.yaml
-rw-r--r--. 1 root root  383 May 11 23:04 management-ipv6.yaml
-rw-r--r--. 1 root root  290 May 11 23:04 management.yaml
-rw-r--r--. 1 root root  136 May 11 23:04 no-networks.yaml
-rw-r--r--. 1 root root 2725 May 11 23:04 routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2033 May 11 23:04 routed-networks.yaml
-rw-r--r--. 1 root root  943 May 11 23:04 vip-data-default-network-isolation.yaml
-rw-r--r--. 1 root root  848 May 11 23:04 vip-data-fixed-ip.yaml
-rw-r--r--. 1 root root 1050 May 11 23:04 vip-data-routed-networks.yaml
```

2. Copy the sample network configuration file you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory:

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml /home/stack/templates/network_data.yaml
```

3. Edit your **network_data.yaml** configuration file and add a section for your new network:

```
- name: StorageBackup
  vip: false
  name_lower: storage_backup
  subnets:
    storage_backup_subnet:
      ip_subnet: 172.16.6.0/24
      allocation_pools:
        - start: 172.16.6.4
        - end: 172.16.6.250
      gateway_ip: 172.16.6.1
```

Configure any other network attributes for your environment. For more information about the properties you can use to configure network attributes, see link:https://access.redhat.com/documentation/en-

us/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_
overcloud-networking_installing-director-on-the-undercloud#ref_network-definition-file-
configuration-options_overcloud_networking [Network definition file configuration options].

4. When you add an extra composable network that contains a virtual IP, and want to map some API services to this network, use the **CloudName{network.name}** definition to set the DNS name for the API endpoint:

```
CloudName{{network.name}}
```

Here is an example:

```
parameter_defaults:
  ...
  CloudNameOcProvisioning: baremetal-vip.example.com
```

5. Copy the sample network VIP definition template you require from **/usr/share/openstack-tripleo-heat-templates/network-data-samples** to your environment file directory. The following example copies the **vip-data-default-network-isolation.yaml** to a local environment file named **vip_data.yaml**:

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/vip-data-default-
network-isolation.yaml  /home/stack/templates/vip_data.yaml
```

6. Edit your **vip_data.yaml** configuration file. The virtual IP data is a list of virtual IP address definitions, each containing the name of the network where the IP address is allocated:

```
- network: storage_mgmt
  dns_name: overcloud
- network: internal_api
  dns_name: overcloud
- network: storage
  dns_name: overcloud
- network: external
  dns_name: overcloud
  ip_address: <vip_address>
- network: ctlplane
  dns_name: overcloud
```

   - Replace **<vip_address>** with the required virtual IP address.

   For more information about the properties you can use to configure network VIP attributes in your VIP definition file, see Network VIP attribute properties.

7. Copy a sample network configuration template. Jinja2 templates are used to define NIC configuration templates. Browse the examples provided in the **/usr/share/ansible/roles/tripleo_network_config/templates/** directory, if one of the examples matches your requirements, use it. If the examples do not match your requirements, copy a sample configuration file, and modify it for your needs:

```
$ cp
/usr/share/ansible/roles/tripleo_network_config/templates/single_nic_vlans/single_nic_vlans.j2
/home/stack/templates/
```

8. Edit your **single_nic_vlans.j2** configuration file:

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    # force the MAC address of the bridge to this interface
    primary: true
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
```

9. Set the **network_config** template in **overcloud-baremetal-deploy.yaml** configuration file:

```
- name: CephStorage
  count: 3
  defaults:
    networks:
    - network: storage
    - network: storage_mgmt
    - network: storage_backup
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
```

10. Provision the overcloud networks. This action generates an output file which will be used an an environment file when deploying the overcloud:

```
(undercloud)$ openstack overcloud network provision --output <deployment_file>
/home/stack/templates/<networks_definition_file>.yaml
```

- Replace **<networks_definition_file>** with the name of your networks definition file, for example, **network_data.yaml**.

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-networks-deployed.yaml**.

11. Provision the network VIPs and generate the **vip-deployed-environment.yaml** file. You use this file when you deploy the overcloud:

```
(overcloud)$ openstack overcloud network vip provision  --stack <stack> --output
<deployment_file> /home/stack/templates/vip_data.yaml
```

- Replace **<stack>** with the name of the stack for which the network VIPs are provisioned. If not specified, the default is overcloud.

- Replace **<deployment_file>** with the name of the heat environment file to generate for inclusion in the deployment command, for example **/home/stack/templates/overcloud-vip-deployed.yaml**.

8.2.2.2. 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/templates/roles_data.yaml
```

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
      subnet: storage_subnet
    StorageMgmt
      subnet: storage_mgmt_subnet
    StorageBackup
      subnet: storage_backup_subnet
```

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.

8.2.2.3. 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:
    SwiftStorageNetwork: 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.

8.2.2.4. Enabling custom composable networks

Use one of the default NIC templates to enable custom composable networks. In this example, use the Single NIC with VLANs template, (**custom_single_nic_vlans**).

Procedure

1. Source the **stackrc** undercloud credentials file:

```
$ source ~/stackrc
```

2. Provision the overcloud networks:

```
$ openstack overcloud network provision \
  --output overcloud-networks-deployed.yaml \
  custom_network_data.yaml
```

3. Provision the network VIPs:

```
$ openstack overcloud network vip provision \
```

```
  --stack overcloud \
  --output overcloud-networks-vips-deployed.yaml \
   custom_vip_data.yaml
```

4. Provision the overcloud nodes:

```
$ openstack overcloud node provision \
   --stack overcloud \
   --output overcloud-baremetal-deployed.yaml \
   overcloud-baremetal-deploy.yaml
```

5. Construct your **openstack overcloud deploy** command, specifying the configuration files and templates in the required order, for example:

```
$ openstack overcloud deploy --templates \
  --networks-file network_data_v2.yaml \
  -e overcloud-networks-deployed.yaml \
  -e overcloud-networks-vips-deployed.yaml \
  -e overcloud-baremetal-deployed.yaml
  -e custom-net-single-nic-with-vlans.yaml
```

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

8.2.2.5. 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
```

8.2.3. Additional overcloud network configuration

This chapter follows on from the concepts and procedures outlined in Section 8.2.1, "Defining custom network interface templates" and provides some additional information to help configure parts of your overcloud network.

8.2.3.1. 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: 9
  addresses:
  - ip_netmask: 172.17.0.100/16
  routes:
  - ip_netmask: 10.1.2.0/24
    next_hop: 172.17.0.1
```

8.2.3.2. Configuring policy-based routing

To configure unlimited access from different networks on Controller nodes, 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 on the undercloud in the *Installing and managing Red Hat OpenStack Platform with director* guide.

Procedure

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

```
network_config:
- type: interface
  name: em1
  use_dhcp: false
  addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
  routes:
    - default: true
      next_hop: {{ external_gateway_ip }}
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
      next_hop: {{ external_gateway_ip }}
      table: 2
      route_options: metric 100
  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. 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 /home/stack/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
```

8.2.3.3. 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 can all benefit from jumbo frames.

You can alter the value of the **mtu** in the **jinja2** template or in the **network_data.yaml** file. If you set the value in the **network_data.yaml** file it is rendered during deployment.

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.

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: bridge_name
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
```

```
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: vlan
    mtu: 9000 (1)
    vlan_id: {{ storage_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_ip }}/{{ storage_cidr }}
    routes: {{ [storage_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ storage_mgmt_mtu }} (2)
    vlan_id: {{ storage_mgmt_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_mgmt_ip }}/{{ storage_mgmt_cidr }}
    routes: {{ [storage_mgmt_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}
    routes: {{ [internal_api_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ external_mtu }}
    vlan_id: {{ external_vlan_id }}
    addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr }}
    routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

1. mtu value updated directly in the **jinja2** template.
2. mtu value is taken from the **network_data.yaml** file during deployment.

8.2.3.4. 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.

Procedure

1. Set the following value in the [ovn] section of ml2_conf.ini:

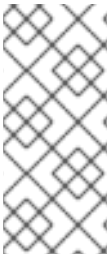
```
ovn_emit_need_to_frag = True
```

8.2.3.5. 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.

The following example configures a bonded interface where the External network is on the native VLAN:

```
network_config:
- type: ovs_bridge
  name: br-ex
  addresses:
  - ip_netmask: {{ external_ip }}/{{ external_cidr }}
  routes: {{ external_host_routes }}
  members:
  - type: ovs_bond
    name: bond1
    ovs_options: {{ bond_interface_ovs_options }}
    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.

8.2.3.6. 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/custom-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

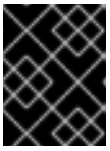
  In this example, you can set the conntrack limit for only the Controller role in your RHOSP deployment:

```
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/custom-environment.yaml
```

Additional resources

- Environment files
- Including environment files in overcloud creation

8.2.4. Network interface bonding

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

8.2.4.1. 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 8.7. 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.

8.2.4.2. 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:

```
- type: ovs_user_bridge
  name: br-dpdk0
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: {{ num_dpdk_interface_rx_queues }}
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      members:
      - type: interface
        name: nic4
    - type: ovs_dpdk_port
      name: dpdk1
      members:
      - type: interface
        name: nic5
```

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:

```
environment_parameters:
  BondInterfaceOvsOptions: "bond_mode=active_backup"
```

8.2.4.3. 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.

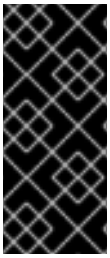
other_config:bond-rebalance-interval=10000

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

8.2.4.4. 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

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 8.8. 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**	• Performance is affected by extra parsing per packet. • There is a potential for vhost-user lock contention.
	balance-tcp	**active** or **passive**	• As with balance-slb, performance is affected by extra parsing per packet and there is a potential for vhost-user lock contention. • LACP must be configured and enabled. • Set **lb-output-action=true**. For example: `ovs-vsctl set port <bond port> other_config:lb-output-action=true`

8.2.4.5. Creating Linux bonds

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

```
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
```

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 9 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:

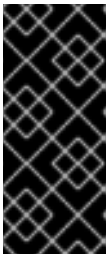
```
....

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: "mode=active-backup"
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
```

```
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask:
        {{ internal_api_ip }}/{{ internal_api_cidr }}
      routes:
        {{ internal_api_host_routes }}
```

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

```
- type: linux_bond
  name: bond_tenant
  mtu: {{ min_viable_mtu_ctlplane }}
  bonding_options: "mode=802.3ad updelay=1000 miimon=100"
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameserver }}
  domain: {{ dns_search_domains }}
  members:
    - type: interface
      name: p1p1
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: interface
      name: p1p2
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: vlan
      mtu: {{ tenant_mtu }}
      vlan_id: {{ tenant_vlan_id }}
      addresses:
        - ip_netmask:
            {{ tenant_ip }}/{{ tenant_cidr }}
      routes:
        {{ tenant_host_routes }}
```



IMPORTANT

You must set up **min_viable_mtu_ctlplane** before you can use it. Copy **/usr/share/ansible/roles/tripleo_network_config/templates/2_linux_bonds_vlans.j2** to your templates directory and modify it for your needs. For more information, see Composable networks, and refer to the steps that pertain to the network configuration template.

8.2.5. Updating the format of your network configuration files

The format of the network configuration **yaml** files has changed in Red Hat OpenStack Platform (RHOSP) 17.0. The structure of the network configuration file **network_data.yaml** has changed, and the NIC template file format has changed from **yaml** file format to Jinja2 ansible format, **j2**.

You can convert your existing network configuration file in your current deployment to the RHOSP 17+ format by using the following conversion tools:

- **convert_v1_net_data.py**
- **convert_heat_nic_config_to_ansible_j2.py**

You can also manually convert your existing NIC template files.

The files you need to convert include the following:

- **network_data.yaml**
- Controller NIC templates
- Compute NIC templates
- Any other custom network files

8.2.5.1. Updating the format of your network configuration file

The format of the network configuration **yaml** file has changed in Red Hat OpenStack Platform (RHOSP) 17.0. You can convert your existing network configuration file in your current deployment to the RHOSP 17+ format by using the **convert_v1_net_data.py** conversion tool.

Procedure

1. Download the conversion tool:
   - **/usr/share/openstack-tripleo-heat-templates/tools/convert_v1_net_data.py**
2. Convert your RHOSP 16+ network configuration file to the RHOSP 17+ format:

```
$ python3 convert_v1_net_data.py <network_config>.yaml
```

   - Replace **<network_config>** with the name of the existing configuration file that you want to convert, for example, **network_data.yaml**.

8.2.5.2. Automatically converting NIC templates to Jinja2 Ansible format

The NIC template file format has changed from **yaml** file format to Jinja2 Ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.0.

You can convert your existing NIC template files in your current deployment to the Jinja2 format by using the **convert_heat_nic_config_to_ansible_j2.py** conversion tool.

You can also manually convert your existing NIC template files. For more information, see Manually converting NIC templates to Jinja2 Ansible format.

The files you need to convert include the following:

- Controller NIC templates
- Compute NIC templates
- Any other custom network files

Procedure

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

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

3. Copy the conversion tool to your current directory on the undercloud:

```
$ cp /usr/share/openstack-tripleo-heat-
templates/tools/convert_heat_nic_config_to_ansible_j2.py .
```

4. Convert your Compute and Controller NIC tempate files, and any other custom network files, to the Jinja2 Ansible format:

```
$ python3 convert_heat_nic_config_to_ansible_j2.py \
 [--stack <overcloud> | --standalone] --networks_file <network_config.yaml> \
 <network_template>.yaml
```

   - Replace **<overcloud>** with the name or UUID of the overcloud stack. If **--stack** is not specified, the stack defaults to **overcloud**.

NOTE

You can use the **--stack** option only on your RHOSP 16 deployment because it requires the Orchestration service (heat) to be running on the undercloud node. Starting with RHOSP 17, RHOSP deployments use ephemeral heat, which runs the Orchestration service in a container. If the Orchestration service is not available, or you have no stack, then use the **--standalone** option instead of **--stack**.

   - Replace **<network_config.yaml>** with the name of the configuration file that describes the network deployment, for example, **network_data.yaml**.
   - Replace **<network_template>** with the name of the network configuration file you want to convert.

   Repeat this command until you have converted all your custom network configuration files. The **convert_heat_nic_config_to_ansible_j2.py** script generates a **.j2** file for each **yaml** file you pass to it for conversion.

5. Inspect each generated **.j2** file to ensure the configuration is correct and complete for your environment, and manually address any comments generated by the tool that highlight where the configuration could not be converted. For more information about manually converting the NIC configuration to Jinja2 format, see Heat parameter to Ansible variable mappings .
6. Configure the ***NetworkConfigTemplate** parameters in your **network-environment.yaml** file to point to the generated **.j2** files:

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

7. Delete the **resource_registry** mappings from your **network-environment.yaml** file for the old network configuration files:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
```

8.2.5.3. Manually converting NIC templates to Jinja2 Ansible format

The NIC template file format has changed from **yaml** file format to Jinja2 Ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.0.

You can manually convert your existing NIC template files.

You can also convert your existing NIC template files in your current deployment to the Jinja2 format by using the **convert_heat_nic_config_to_ansible_j2.py** conversion tool. For more information, see Automatically converting NIC templates to Jinja2 ansible format .

The files you need to convert include the following:

- Controller NIC templates
- Compute NIC templates
- Any other custom network files

Procedure

1. Create a Jinja2 template. You can create a new template by copying an example template from the **/usr/share/ansible/roles/tripleo_network_config/templates/** directory on the undercloud node.
2. Replace the heat intrinsic functions with Jinja2 filters. For example, use the following filter to calculate the **min_viable_mtu**:

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
```

3. Use Ansible variables to configure the network properties for your deployment. You can configure each individual network manually, or programatically configure each network by iterating over **role_networks**:
   - To manually configure each network, replace each **get_param** function with the equivalent Ansible variable. For example, if your current deployment configures **vlan_id** by using **get_param: InternalApiNetworkVlanID**, then add the following configuration to your template:

     ```
     vlan_id: {{ internal_api_vlan_id }}
     ```

     Table 8.9. Example network property mapping from heat parameters to Ansible vars

yaml file format	Jinja2 ansible format, **j2**
- type: vlan device: nic2 vlan_id: get_param: InternalApiNetworkVlanID addresses: - ip_netmask: get_param: InternalApiIpSubnet	- type: vlan device: nic2 vlan_id: {{ internal_api_vlan_id }} addresses: - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}

- To programatically configure each network, add a Jinja2 for-loop structure to your template that retrieves the available networks by their role name by using **role_networks**.

  Example

```
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{%- endfor %}
```

  For a full list of the mappings from the heat parameter to the Ansible **vars** equivalent, see Heat parameter to Ansible variable mappings.

4. Configure the ***NetworkConfigTemplate** parameters in your **network-environment.yaml** file to point to the generated **.j2** files:

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

5. Delete the **resource_registry** mappings from your **network-environment.yaml** file for the old network configuration files:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
```

8.2.5.4. Heat parameter to Ansible variable mappings

The NIC template file format has changed from **yaml** file format to Jinja2 ansible format, **j2**, in Red Hat OpenStack Platform (RHOSP) 17.x.

To manually convert your existing NIC template files to Jinja2 ansible format, you can map your heat parameters to Ansible variables to configure the network properties for pre-provisioned nodes in your deployment. You can also map your heat parameters to Ansible variables if you run **openstack overcloud node provision** without specifying the **--network-config** optional argument.

For example, if your current deployment configures **vlan_id** by using **get_param: InternalApiNetworkVlanID**, then replace it with the following configuration in your new Jinja2 template:

```
vlan_id: {{ internal_api_vlan_id }}
```

NOTE

If you provision your nodes by running **openstack overcloud node provision** with the **--network-config** optional argument, you must configure the network properties for your deploying by using the parameters in **overcloud-baremetal-deploy.yaml**. For more information, see Heat parameter to provisioning definition file mappings .

The following table lists the available mappings from the heat parameter to the Ansible **vars** equivalent.

Table 8.10. Mappings from heat parameters to Ansible vars

Heat parameter	Ansible **vars**
BondInterfaceOvsOptions	**{{ bond_interface_ovs_options }}**
ControlPlaneIp	**{{ ctlplane_ip }}**
ControlPlaneDefaultRoute	**{{ ctlplane_gateway_ip }}**
ControlPlaneMtu	**{{ ctlplane_mtu }}**
ControlPlaneStaticRoutes	**{{ ctlplane_host_routes }}**
ControlPlaneSubnetCidr	**{{ ctlplane_subnet_cidr }}**
DnsSearchDomains	**{{ dns_search_domains }}**

Heat parameter	Ansible **vars**
DnsServers	**{{ ctlplane_dns_nameservers }}** **NOTE** This Ansible variable is populated with the IP address configured in **undercloud.conf** for **DEFAULT/undercloud_nameservers** and **%SUBNET_SECTION%/dns_nameservers**. The configuration of **%SUBNET_SECTION%/dns_nameservers** overrides the configuration of **DEFAULT/undercloud_nameservers**, so that you can use different DNS servers for the undercloud and the overcloud, and different DNS servers for nodes on different provisioning subnets.
NumDpdkInterfaceRxQueues	**{{ num_dpdk_interface_rx_queues }}**

Configuring a heat parameter that is not listed in the table

To configure a heat parameter that is not listed in the table, you must configure the parameter as a **{{role.name}}ExtraGroupVars**. After you have configured the parameter as a **{{role.name}}ExtraGroupVars** parameter, you can then use it in your new template. For example, to configure the **StorageSupernet** parameter, add the following configuration to your network configuration file:

```
parameter_defaults:
  ControllerExtraGroupVars:
    storage_supernet: 172.16.0.0/16
```

You can then add **{{ storage_supernet }}** to your Jinja2 template.

WARNING

This process will not work if the **--network-config** option is used with node provisioning. Users requiring custom vars should not use the **--network-config** option. Instead, after creating the Heat stack, apply the node network configuration to the **config-download** ansible run.

Converting the Ansible variable syntax to programmatically configure each network

When you use a Jinja2 for-loop structure to retrieve the available networks by their role name by iterating over **role_networks**, you need to retrieve the lower case name for each network role to prepend to each property. Use the following structure to convert the Ansible **vars** from the above table to the required syntax:

{{ lookup('vars', networks_lower[network] ~ '_<property>') }}

- Replace **<property>** with the property that you are setting, for example, **ip**, **vlan_id**, or **mtu**.

For example, to populate the value for each **NetworkVlanID** dynamically, replace **{{ <network_name>_vlan_id }}** with the following configuration:

```
{{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}`
```

8.2.5.5. Heat parameter to provisioning definition file mappings

If you provision your nodes by running the **openstack overcloud node provision** command with the **--network-config** optional argument, you must configure the network properties for your deployment by using the parameters in the node definition file **overcloud-baremetal-deploy.yaml**.

If your deployment uses pre-provisioned nodes, you can map your heat parameters to Ansible variables to configure the network properties. You can also map your heat parameters to Ansible variables if you run **openstack overcloud node provision** without specifying the **--network-config** optional argument. For more information about configuring network properties by using Ansible variables, see Heat parameter to Ansible variable mappings.

The following table lists the available mappings from the heat parameter to the **network_config** property equivalent in the node definition file **overcloud-baremetal-deploy.yaml**.

Table 8.11. Mappings from heat parameters to node definition file overcloud-baremetal-deploy.yaml

Heat parameter	**network_config** property
BondInterfaceOvsOptions	**bond_interface_ovs_options**
DnsSearchDomains	**dns_search_domains**
NetConfigDataLookup	**net_config_data_lookup**
NeutronPhysicalBridge	**physical_bridge_name**
NeutronPublicInterface	**public_interface_name**
NumDpdkInterfaceRxQueues	**num_dpdk_interface_rx_queues**
{{role.name}}NetworkConfigUpdate	**network_config_update**

The following table lists the available mappings from the heat parameter to the property equivalent in the networks definition file **network_data.yaml**.

Table 8.12. Mappings from heat parameters to networks definition file network_data.yaml

Heat parameter	IPv4 **network_data.yaml** property	IPv6 **network_data.yaml** property
<network_name>IpSubnet	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.1.0/24	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64
<network_name>NetworkVlanID	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>
<network_name>Mtu	- name: <network_name> mtu:	- name: <network_name> mtu:
<network_name>InterfaceDefaultRoute	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.16.0/24 gateway_ip: 172.16.16.1	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64 gateway_ipv6: 2001:db8:a::1
<network_name>InterfaceRoutes	- name: <network_name> subnets: subnet01: ... routes: - destination: 172.18.0.0/24 nexthop: 172.18.1.254	- name: <network_name> subnets: subnet01: ... routes_ipv6: - destination: 2001:db8:b::/64 nexthop: 2001:db8:a::1

8.2.5.6. Changes to the network data schema

The network data schema was updated in Red Hat OpenStack Platform (RHOSP) 17. The main differences between the network data schema used in RHOSP 16 and earlier, and network data schema used in RHOSP 17 and later, are as follows:

- The base subnet has been moved to the **subnets** map. This aligns the configuration for non-routed and routed deployments, such as spine-leaf networking.

- The **enabled** option is no longer used to ignore disabled networks. Instead, you must remove disabled networks from the configuration file.
- The **compat_name** option is no longer required as the heat resource that used it has been removed.
- The following parameters are no longer valid at the network level: **ip_subnet**, **gateway_ip**, **allocation_pools**, **routes**, **ipv6_subnet**, **gateway_ipv6**, **ipv6_allocation_pools**, and **routes_ipv6**. These parameters are still used at the subnet level.
- A new parameter, **physical_network**, has been introduced, that is used to create ironic ports in **metalsmith**.
- New parameters **network_type** and **segmentation_id** replace **{{network.name}}NetValueSpecs** used to set the network type to **vlan**.
- The following parameters have been deprecated in RHOSP 17:
  - **{{network.name}}NetCidr**
  - **{{network.name}}SubnetName**
  - **{{network.name}}Network**
  - **{{network.name}}AllocationPools**
  - **{{network.name}}Routes**
  - **{{network.name}}SubnetCidr_{{subnet}}**
  - **{{network.name}}AllocationPools_{{subnet}}**
  - **{{network.name}}Routes_{{subnet}}**

CHAPTER 9. CONFIGURING AND MANAGING RED HAT OPENSTACK PLATFORM WITH ANSIBLE

You can use Ansible to configure and register the overcloud, and to manage containers.

9.1. 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.

I:f 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 Section 9.1.6, "Switching to the rhsm composable service" and Section 9.1.7, "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 9.1.9, "Running Ansible-based registration manually"

9.1.1. 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:

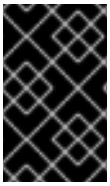
```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 9.2
```

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.

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_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.
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 9.2 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**.

9.1.2. 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_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.
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 9.2 for Red Hat OpenStack Platform

rhsm	Description
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**.

9.1.3. 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-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

- 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. To apply the **rhsm** composable service on a per-role basis, include your configuration in the environment file. For example, you can apply different sets of configurations to Controller nodes, Compute nodes, and Ceph Storage nodes:

```
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-rpms
        - rhel-9-for-x86_64-appstream-rpms
        - rhel-9-for-x86_64-highavailability-rpms
        - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
      rhsm_method: "portal"
      rhsm_release: 9.2
```

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.

4. Save the environment file.

9.1.4. 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-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
```

```
CephStorageParameters:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-rpms
      - rhel-9-for-x86_64-appstream-rpms
      - rhel-9-for-x86_64-highavailability-rpms
      - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

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.

9.1.5. 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: 9.2
```

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.

9.1.6. 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.

9.1.7. 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_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**

9.1.8. 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
```

9.1.9. 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-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 9.2
        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.

9.2. CONFIGURING THE OVERCLOUD WITH ANSIBLE

Ansible is the main method to apply the overcloud configuration. This chapter provides information about how to interact with the overcloud Ansible configuration.

Although director generates the Ansible playbooks automatically, it is a good idea to familiarize yourself with Ansible syntax. For more information about using Ansible, see https://docs.ansible.com/.

NOTE

Ansible also uses the concept of roles, which are different to OpenStack Platform director roles. **Ansible roles** form reusable components of playbooks, whereas director roles contain mappings of OpenStack services to node types.

9.2.1. Ansible-based overcloud configuration (config-download)

The **config-download** feature is the method that director uses to configure the overcloud. Director uses **config-download** in conjunction with OpenStack Orchestration (heat) to generate the software configuration and apply the configuration to each overcloud node. Although heat creates all deployment data from **SoftwareDeployment** resources to perform the overcloud installation and configuration, heat does not apply any of the configuration. Heat only provides the configuration data through the heat API.

As a result, when you run the **openstack overcloud deploy** command, the following process occurs:

- Director creates a new deployment plan based on **openstack-tripleo-heat-templates** and includes any environment files and parameters to customize the plan.
- Director uses heat to interpret the deployment plan and create the overcloud stack and all descendant resources. This includes provisioning nodes with the OpenStack Bare Metal service (ironic).
- Heat also creates the software configuration from the deployment plan. Director compiles the Ansible playbooks from this software configuration.
- Director generates a temporary user (**tripleo-admin**) on the overcloud nodes specifically for Ansible SSH access.
- Director downloads the heat software configuration and generates a set of Ansible playbooks using heat outputs.
- Director applies the Ansible playbooks to the overcloud nodes using **ansible-playbook**.

9.2.2. config-download working directory

The **ansible-playbook** command creates an Ansible project directory, default name **~/config-download/overcloud**. This project directory stores downloaded software configuration from heat. It includes all Ansible-related files which you need to run **ansible-playbook** to configure the overcloud.

The contents of the directory include:

- tripleo-ansible-inventory.yaml - Ansible inventory file containing **hosts** and **vars** for all the overcloud nodes.
- ansible.log - Log file from the most recent run of **ansible-playbook**.
- ansible.cfg - Configuration file used when running **ansible-playbook**.
- ansible-playbook-command.sh - Executable script used to rerun **ansible-playbook**.
- ssh_private_key - Private ssh key used to access the overcloud nodes.

1. Reproducing ansible-playbook

After the project directory is created, run the **ansible-playbook-command.sh** command to reproduce the deployment.

```
$ ./ansible-playbook-command.sh
```

You can run the script with additional arguments, such as check mode **--check**, limiting hosts **--limit**, and overriding variables **-e**.

```
$ ./ansible-playbook-command.sh --check
```

9.2.3. Checking config-download log

During the **config-download** process, Ansible creates a log file, named **ansible.log**, in the **/home/stack** directory on the undercloud.

Procedure

1. View the log with the **less** command:

```
$ less ~/ansible.log
```

9.2.4. Performing Git operations on the working directory

The **config-download** working directory is a local Git repository. Every time a deployment operation runs, director adds a Git commit to the working directory with the relevant changes. You can perform Git operations to view configuration for the deployment at different stages and compare the configuration with different deployments.

Be aware of the limitations of the working directory. For example, if you use Git to revert to a previous version of the **config-download** working directory, this action affects only the configuration in the working directory. It does not affect the following configurations:

- **The overcloud data schema:** Applying a previous version of the working directory software configuration does not undo data migration and schema changes.

- **The hardware layout of the overcloud:**Reverting to previous software configuration does not undo changes related to overcloud hardware, such as scaling up or down.
- **The heat stack:** Reverting to earlier revisions of the working directory has no effect on the configuration stored in the heat stack. The heat stack creates a new version of the software configuration that applies to the overcloud. To make permanent changes to the overcloud, modify the environment files applied to the overcloud stack before you rerun the **openstack overcloud deploy** command.

Complete the following steps to compare different commits of the **config-download** working directory.

Procedure

1. Change to the **config-download** working directory for your overcloud, usually named **overcloud**:

```
$ cd ~/config-download/overcloud
```

2. Run the **git log** command to list the commits in your working directory. You can also format the log output to show the date:

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

   By default, the most recent commit appears first.

3. Run the **git diff** command against two commit hashes to see all changes between the deployments:

```
$ git diff a7e9063 dfb9d12
```

9.2.5. Deployment methods that use config-download

There are four main methods that use **config-download** in the context of an overcloud deployment:

Standard deployment

Run the **openstack overcloud deploy** command to automatically run the configuration stage after the provisioning stage. This is the default method when you run the **openstack overcloud deploy** command.

Separate provisioning and configuration

Run the **openstack overcloud deploy** command with specific options to separate the provisioning and configuration stages.

Run the ansible-playbook-command.sh script after a deployment

Run the **openstack overcloud deploy** command with combined or separate provisioning and configuration stages, then run the **ansible-playbook-command.sh** script supplied in the **config-download** working directory to re-apply the configuration stage.

Provision nodes, manually create config-download, and run Ansible

Run the **openstack overcloud deploy** command with a specific option to provision nodes, then run the **ansible-playbook** command with the **deploy_steps_playbook.yaml** playbook.

9.2.6. Running config-download on a standard deployment

The default method for executing **config-download** is to run the **openstack overcloud deploy** command. This method suits most environments.

Prerequisites

- A successful undercloud installation.
- Overcloud nodes ready for deployment.
- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

```
$ source ~/stackrc
```

3. Run the deployment command. Include any environment files that you require for your overcloud:

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

4. Wait until the deployment process completes.

During the deployment process, director generates the **config-download** files in a **~/config-download/overcloud** working directory. After the deployment process finishes, view the Ansible playbooks in the working directory to see the tasks director executed to configure the overcloud.

9.2.7. Running config-download with separate provisioning and configuration

The **openstack overcloud deploy** command runs the heat-based provisioning process and then the **config-download** configuration process. You can also run the deployment command to execute each process individually. Use this method to provision your overcloud nodes as a distinct process so that you can perform any manual pre-configuration tasks on the nodes before you run the overcloud configuration process.

Prerequisites

- A successful undercloud installation.
- Overcloud nodes ready for deployment.
- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

2. Source the **stackrc** file:

```
$ source ~/stackrc
```

3. Run the deployment command with the **--stack-only** option. Include any environment files you require for your overcloud:

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

4. Wait until the provisioning process completes.

5. Enable SSH access from the undercloud to the overcloud for the **tripleo-admin** user. The **config-download** process uses the **tripleo-admin** user to perform the Ansible-based configuration:

```
$ openstack overcloud admin authorize
```

6. Perform any manual pre-configuration tasks on nodes. If you use Ansible for configuration, use the **tripleo-admin** user to access the nodes.

7. Run the deployment command with the **--config-download-only** option. Include any environment files required for your overcloud:

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --config-download-only
```

8. Wait until the configuration process completes.

During the configuration stage, director generates the **config-download** files in a **~/config-download/overcloud** working directory. After the deployment process finishes, view the Ansible playbooks in the working directory to see the tasks director executed to configure the overcloud.

9.2.8. Running config-download with the ansible-playbook-command.sh script

When you deploy the overcloud, either with the standard method or a separate provisioning and configuration process, director generates a working directory in **~/config-download/overcloud**. This directory contains the playbooks and scripts necessary to run the configuration process again.

Prerequisites

- An overcloud deployed with the one of the following methods:
  - Standard method that combines provisioning and configuration process.
  - Separate provisioning and configuration processes.

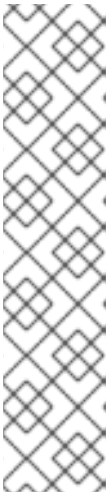
Procedure

1. Log in to the undercloud host as the **stack** user.
2. Run the **ansible-playbook-command.sh** script.
   You can pass additional Ansible arguments to this script, which are then passed unchanged to the **ansible-playbook** command. This makes it possible to take advantage of Ansible features, such as check mode (**--check**), limiting hosts (**--limit**), or overriding variables (**-e**). For example:

```
$ ./ansible-playbook-command.sh --limit Controller
```

WARNING

When **--limit** is used to deploy at scale, only hosts included in the execution are added to the SSH **known_hosts** file across the nodes. Therefore, some operations, such as live migration, may not work across nodes that are not in the **known_hosts** file.



NOTE

To ensure that the **/etc/hosts** file, on all nodes, is up-to-date, run the following command as the **stack** user:

```
(undercloud)$ cd /home/stack/overcloud-deploy/overcloud/config-download/overcloud
(undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible allovercloud \
  -i /home/stack/overcloud-deploy/overcloud/tripleo-ansible-inventory.yaml \
  -m include_role \
  -a name=tripleo_hosts_entries \
  -e @global_vars.yaml
```

3. Wait until the configuration process completes.

Additional information

- The working directory contains a playbook called **deploy_steps_playbook.yaml**, which manages the overcloud configuration tasks. To view this playbook, run the following command:

```
$ less deploy_steps_playbook.yaml
```

  The playbook uses various task files contained in the working directory. Some task files are common to all OpenStack Platform roles and some are specific to certain OpenStack Platform roles and servers.

- The working directory also contains sub-directories that correspond to each role that you define in your overcloud **roles_data** file. For example:

```
$ ls Controller/
```

Each OpenStack Platform role directory also contains sub-directories for individual servers of that role type. The directories use the composable role hostname format:

```
$ ls Controller/overcloud-controller-0
```

- The Ansible tasks in **deploy_steps_playbook.yaml** are tagged. To see the full list of tags, use the CLI option **--list-tags** with **ansible-playbook**:

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags
deploy_steps_playbook.yaml
```

Then apply tagged configuration using the **--tags**, **--skip-tags**, or **--start-at-task** with the **ansible-playbook-command.sh** script:

```
$ ./ansible-playbook-command.sh --tags overcloud
```

4. When you run the **config-download** playbooks against the overcloud, you might receive a message regarding the SSH fingerprint for each host. To avoid these messages, include **--ssh-common-args="-o StrictHostKeyChecking=no"** when you run the **ansible-playbook-command.sh** script:

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

9.2.9. Running config-download with manually created playbooks

You can create your own **config-download** files outside of the standard workflow. For example, you can run the **openstack overcloud deploy** command with the **--stack-only** option to provision the nodes, and then manually apply the Ansible configuration separately.

Prerequisites

- A successful undercloud installation.
- Overcloud nodes ready for deployment.
- Heat environment files that are relevant to your specific overcloud customization.

Procedure

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

```
$ source ~/stackrc
```

3. Run the deployment command with the **--stack-only** option. Include any environment files required for your overcloud:

```
$ openstack overcloud deploy \
 --templates \
```

```
-e environment-file1.yaml \
-e environment-file2.yaml \
...
--stack-only
```

4. Wait until the provisioning process completes.

5. Enable SSH access from the undercloud to the overcloud for the **tripleo-admin** user. The **config-download** process uses the **tripleo-admin** user to perform the Ansible-based configuration:

```
$ openstack overcloud admin authorize
```

6. Generate the **config-download** files:

```
$ openstack overcloud deploy \
  --stack overcloud --stack-only \
  --config-dir  ~/overcloud-deploy/overcloud/config-download/overcloud/
```

   - **--stack** specifies the name of the overcloud.
   - **--stack-only** ensures that the command only deploys the heat stack and skips any software configuration.
   - **--config-dir** specifies the location of the **config-download** files.

7. Change to the directory that contains your **config-download** files:

```
$ cd ~/config-download
```

8. Generate a static inventory file:

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user tripleo-admin \
  --static-yaml-inventory inventory.yaml
```

   - Replace **<overcloud>** with the name of your overcloud.

9. Use the **~/overcloud-deploy/overcloud/config-download/overcloud** files and the static inventory file to perform a configuration. To execute the deployment playbook, run the **ansible-playbook** command:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/overcloud-deploy/overcloud/config-download/overcloud/deploy_steps_playbook.yaml
```

NOTE

When you run the **config-download/overcloud** playbooks against the overcloud, you might receive a message regarding the SSH fingerprint for each host. To avoid these messages, include **--ssh-common-args="-o StrictHostKeyChecking=no"** in your **ansible-playbook** command:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

10. Wait until the configuration process completes.

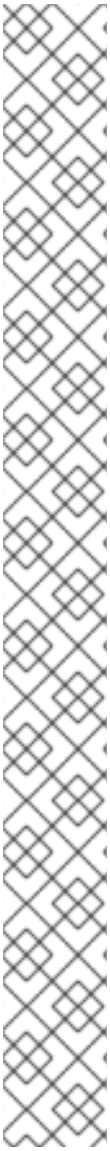
11. Generate an **overcloudrc** file manually from the ansible-based configuration:

```
$ openstack action execution run \
  --save-result \
  --run-sync \
  tripleo.deployment.overcloudrc \
  '{"container":"overcloud"}' \
  | jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. Manually set the deployment status to success:

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<overcloud>"}'
```

- Replace **<overcloud>** with the name of your overcloud.



NOTE

The ~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ directory contains a playbook called **deploy_steps_playbook.yaml**. The playbook uses various task files contained in the working directory. Some task files are common to all Red Hat OpenStack Platform (RHOSP) roles and some are specific to certain RHOSP roles and servers.

The ~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ directory also contains sub-directories that correspond to each role that you define in your overcloud **roles_data** file. Each RHOSP role directory also contains sub-directories for individual servers of that role type. The directories use the composable role hostname format, for example **Controller**/**overcloud-controller-0**.

The Ansible tasks in **deploy_steps_playbook.yaml** are tagged. To see the full list of tags, use the CLI option **--list-tags** with **ansible-playbook**:

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags
deploy_steps_playbook.yaml
```

You can apply tagged configuration using the **--tags**, **--skip-tags**, or **--start-at-task** with the **ansible-playbook-command.sh** script:

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

9.2.10. Limitations of config-download

The **config-download** feature has some limitations:

- When you use ansible-playbook CLI arguments such as **--tags**, **--skip-tags**, or **--start-at-task**, do not run or apply deployment configuration out of order. These CLI arguments are a convenient way to rerun previously failed tasks or to iterate over an initial deployment. However, to guarantee a consistent deployment, you must run all tasks from **deploy_steps_playbook.yaml** in order.

- You can not use the **--start-at-task** arguments for certain tasks that use a variable in the task name. For example, the **--start-at-task** arguments does not work for the following Ansible task:

  ```
  - name: Run puppet host configuration for step {{ step }}
  ```

- If your overcloud deployment includes a director-deployed Ceph Storage cluster, you cannot skip **step1** tasks when you use the **--check** option unless you also skip **external_deploy_steps** tasks.

- You can set the number of parallel Ansible tasks with the **--forks** option. However, the performance of **config-download** operations degrades after 25 parallel tasks. For this reason, do not exceed 25 with the **--forks** option.

9.2.11. config-download top level files

The following file are important top level files within a **config-download** working directory.

Ansible configuration and execution

The following files are specific to configuring and executing Ansible within the **config-download** working directory.

ansible.cfg

Configuration file used when running **ansible-playbook**.

ansible.log

Log file from the last run of **ansible-playbook**.

ansible-errors.json

JSON structured file that contains any deployment errors.

ansible-playbook-command.sh

Executable script to rerun the **ansible-playbook** command from the last deployment operation.

ssh_private_key

Private SSH key that Ansible uses to access the overcloud nodes.

tripleo-ansible-inventory.yaml

Ansible inventory file that contains hosts and variables for all the overcloud nodes.

overcloud-config.tar.gz

Archive of the working directory.

Playbooks

The following files are playbooks within the **config-download** working directory.

deploy_steps_playbook.yaml

Main deployment steps. This playbook performs the main configuration operations for your overcloud.

pre_upgrade_rolling_steps_playbook.yaml

Pre upgrade steps for major upgrade

upgrade_steps_playbook.yaml

Major upgrade steps.

post_upgrade_steps_playbook.yaml

Post upgrade steps for major upgrade.

update_steps_playbook.yaml

Minor update steps.

fast_forward_upgrade_playbook.yaml

Fast forward upgrade tasks. Use this playbook only when you want to upgrade from one long-life version of Red Hat OpenStack Platform to the next.

9.2.12. config-download tags

The playbooks use tagged tasks to control the tasks that they apply to the overcloud. Use tags with the **ansible-playbook** CLI arguments **--tags** or **--skip-tags** to control which tasks to execute. The following list contains information about the tags that are enabled by default:

facts

Fact gathering operations.

common_roles

Ansible roles common to all nodes.

overcloud

All plays for overcloud deployment.

pre_deploy_steps

Deployments that happen before the **deploy_steps** operations.

host_prep_steps

Host preparation steps.

deploy_steps

Deployment steps.

post_deploy_steps

Steps that happen after the **deploy_steps** operations.

external

All external deployment tasks.

external_deploy_steps

External deployment tasks that run on the undercloud only.

9.2.13. config-download deployment steps

The **deploy_steps_playbook.yaml** playbook configures the overcloud. This playbook applies all software configuration that is necessary to deploy a full overcloud based on the overcloud deployment plan.

This section contains a summary of the different Ansible plays used within this playbook. The play names in this section are the same names that are used within the playbook and that are displayed in the **ansible-playbook** output. This section also contains information about the Ansible tags that are set on each play.

Gather facts from undercloud

Fact gathering for the undercloud node.
Tags: **facts**

Gather facts from overcloud

Fact gathering for the overcloud nodes.
Tags: **facts**

Load global variables

Loads all variables from **global_vars.yaml**.
Tags: **always**

Common roles for TripleO servers

Applies common Ansible roles to all overcloud nodes, including tripleo-bootstrap for installing bootstrap packages, and tripleo-ssh-known-hosts for configuring ssh known hosts.
Tags: **common_roles**

Overcloud deploy step tasks for step 0

Applies tasks from the deploy_steps_tasks template interface.
Tags: **overcloud**, **deploy_steps**

Server deployments

Applies server-specific heat deployments for configuration such as networking and hieradata. Includes NetworkDeployment, <Role>Deployment, <Role>AllNodesDeployment, etc.
Tags: **overcloud**, **pre_deploy_steps**

Host prep steps

Applies tasks from the host_prep_steps template interface.
Tags: **overcloud**, **host_prep_steps**

External deployment step [1,2,3,4,5]

Applies tasks from the external_deploy_steps_tasks template interface. Ansible runs these tasks only against the undercloud node.
Tags: **external**, **external_deploy_steps**

Overcloud deploy step tasks for [1,2,3,4,5]

Applies tasks from the deploy_steps_tasks template interface.
Tags: **overcloud**, **deploy_steps**

Overcloud common deploy step tasks [1,2,3,4,5]

Applies the common tasks performed at each step, including puppet host configuration, **container-puppet.py**, and **tripleo-container-manage** (container configuration and management).
Tags: **overcloud**, **deploy_steps**

Server Post Deployments

Applies server specific heat deployments for configuration performed after the 5-step deployment process.
Tags: **overcloud**, **post_deploy_steps**

External deployment Post Deploy tasks

Applies tasks from the external_post_deploy_steps_tasks template interface. Ansible runs these tasks only against the undercloud node.
Tags: **external**, **external_deploy_steps**

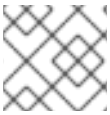
9.3. MANAGING CONTAINERS WITH ANSIBLE

Red Hat OpenStack Platform 17.1 uses the **tripleo_container_manage** Ansible role to perform management operations on containers. You can also write custom playbooks to perform specific container management operations:

- Collect the container configuration data that heat generates. The **tripleo_container_manage** role uses this data to orchestrate container deployment.
- Start containers.
- Stop containers.

- Update containers.
- Delete containers.
- Run a container with a specific configuration.

Although director performs container management automatically, you might want to customize a container configuration, or apply a hotfix to a container without redeploying the overcloud.



NOTE

This role supports only Podman container management.

9.3.1. tripleo-container-manage role defaults and variables

The following excerpt shows the defaults and variables for the **tripleo_container_manage** Ansible role.

```
# All variables intended for modification should place placed in this file.
tripleo_container_manage_hide_sensitive_logs: '{{ hide_sensitive_logs | default(true)
  }}'
tripleo_container_manage_debug: '{{ ((ansible_verbosity | int) >= 2) | bool }}'
tripleo_container_manage_clean_orphans: true

# All variables within this role should have a prefix of "tripleo_container_manage"
tripleo_container_manage_check_puppet_config: false
tripleo_container_manage_cli: podman
tripleo_container_manage_concurrency: 1
tripleo_container_manage_config: /var/lib/tripleo-config/
tripleo_container_manage_config_id: tripleo
tripleo_container_manage_config_overrides: {}
tripleo_container_manage_config_patterns: '*.json'
# Some containers where Puppet is run, can take up to 10 minutes to finish
# in slow environments.
tripleo_container_manage_create_retries: 120
# Default delay is 5s so 120 retries makes a timeout of 10 minutes which is
# what we have observed a necessary value for nova and neutron db-sync execs.
tripleo_container_manage_exec_retries: 120
tripleo_container_manage_healthcheck_disabled: false
tripleo_container_manage_log_path: /var/log/containers/stdouts
tripleo_container_manage_systemd_teardown: true
```

9.3.2. tripleo-container-manage molecule scenarios

Molecule is used to test the **tripleo_container_manage** role. The following shows a **molecule** default inventory:

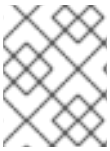
```
hosts:
  all:
    hosts:
      instance:
        ansible_host: localhost
        ansible_connection: local
        ansible_distribution: centos8
```

Usage

Red Hat OpenStack 17.1 supports only Podman in this role. Docker support is on the roadmap.

The **Molecule** Ansible role performs the following tasks:

- Collect container configuration data, generated by the TripleO Heat Templates. This data is used as a source of truth. If a container is already managed by **Molecule**, no matter its present state, the configuration data will reconfigure the container as needed.
- Manage **systemd** shutdown files. It creates the **TripleO Container systemd** service, required for service ordering when shutting down or starting a node. It also manages the **netns-placeholder** service.
- Delete containers that are nonger needed or that require reconfiguration. It uses a custom filter, named **needs_delete()** which has a set of rules to determine if the container needs to be deleted.
  - A container will not be deleted if, the container is not managed by **tripleo_ansible** or the container **config_id** does not match the input ID.
  - A container will be deleted, if the container has no **config_data** or the container has **config_data** which does not match data in input. Note that when a container is removed, the role also disables and removes the **systemd** services and healtchecks.
- Create containers in a specific order defined by **start_order** container config, where the default is 0.
  - If the container is an **exec**, a dedicated playbook for **execs** is run, using **async** so multiple **execs** can be run at the same time.
  - Otherwise, the **podman_container** is used, in async, to create the containers. If the container has a **restart policy**, **systemd** service is configured. If the container has a healthcheck script, **systemd healthcheck** service is configured.



NOTE

tripleo_container_manage_concurrency parameter is set to 1 by default, and putting a value higher than 2 can expose issues with Podman locks.

Example of a playbook:

```
- name: Manage step_1 containers using tripleo-ansible
  block:
    - name: "Manage containers for step 1 with tripleo-ansible"
      include_role:
        name: tripleo_container_manage
      vars:
        tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
        tripleo_container_manage_config_id: "tripleo_step1"
```

9.3.3. tripleo_container_manage role variables

The **tripleo_container_manage** Ansible role contains the following variables:

Table 9.1. Role variables

Name	Default value	Description
tripleo_container_manage_check_puppet_config	false	Use this variable if you want Ansible to check Puppet container configurations. Ansible can identify updated container configuration using the configuration hash. If a container has a new configuration from Puppet, set this variable to **true** so that Ansible can detect the new configuration and add the container to the list of containers that Ansible must restart.
tripleo_container_manage_cli	podman	Use this variable to set the command line interface that you want to use to manage containers. The **tripleo_container_manage** role supports only Podman.
tripleo_container_manage_concurrency	1	Use this variable to set the number of containers that you want to manage concurrently.
tripleo_container_manage_config	/var/lib/tripleo-config/	Use this variable to set the path to the container configuration directory.
tripleo_container_manage_config_id	tripleo	Use this variable to set the ID of a specific configuration step. For example, set this value to **tripleo_step2** to manage containers for step two of the deployment.
tripleo_container_manage_config_patterns	*.json	Use this variable to set the bash regular expression that identifies configuration files in the container configuration directory.

Name	Default value	Description
tripleo_container_manage_debug	false	Use this variable to enable or disable debug mode. Run the **tripleo_container_manage** role in debug mode if you want to run a container with a specific one-time configuration, to output the container commands that manage the lifecycle of containers, or to run no-op container management operations for testing and verification purposes.
tripleo_container_manage_healthcheck_disable	false	Use this variable to enable or disable healthchecks.
tripleo_container_manage_log_path	/var/log/containers/stdouts	Use this variable to set the stdout log path for containers.
tripleo_container_manage_systemd_order	false	Use this variable to enable or disable systemd shutdown ordering with Ansible.
tripleo_container_manage_systemd_teardown	true	Use this variable to trigger the cleanup of obsolete containers.
tripleo_container_manage_config_overrides	{}	Use this variable to override any container configuration. This variable takes a dictionary of values where each key is the container name and the parameters that you want to override, for example, the container image or user. This variable does not write custom overrides to the JSON container configuration files and any new container deployments, updates, or upgrades revert to the content of the JSON configuration file.
tripleo_container_manage_valid_exit_code	[]	Use this variable to check if a container returns an exit code. This value must be a list, for example, **[0,3]**.

9.3.4. tripleo-container-manage healthchecks

Until Red Hat OpenStack 17.1, container healthcheck was implemented by a **systemd** timer which would run **podman exec** to determine if a given container was healthy. Now, it uses the native healthcheck interface in **Podman** which is easier to integrate and consume.

To check if a container (for example, keystone) is healthy, run the following command:

```
$ sudo podman healthcheck run keystone
```

The return code should be **0** and **"healthy"**.

```
"Healthcheck": {
    "Status": "healthy",
    "FailingStreak": 0,
    "Log": [
        {
            "Start": "2020-04-14T18:48:57.272180578Z",
            "End": "2020-04-14T18:48:57.806659104Z",
            "ExitCode": 0,
            "Output": ""
        },
        (...)
    ]
}
```

9.3.5. tripleo-container-manage debug

The **tripleo_container_manage** Ansible role allows you to perform specific actions on a given container. This can be used to:

- Run a container with a specific one-off configuration.
- Output the container commands to manage containers lifecycle.
- Output the changes made on containers by Ansible.

NOTE

To manage a single container, you need to know two things:

- At which step during the overcloud deployment was the container deployed.
- The name of the generated JSON file containing the container configuration.

The following is an example of a playbook to manage **HAproxy** container at **step 1** which overrides the image setting:

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo_container_manage
```

```
    vars:
      tripleo_container_manage_config_patterns: 'haproxy.json'
      tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
      tripleo_container_manage_config_id: "tripleo_step1"
      tripleo_container_manage_clean_orphans: false
      tripleo_container_manage_config_overrides:
        haproxy:
          image: quay.io/tripleomastercentos9/centos-binary-haproxy:hotfix
```

If Ansible is run in **check mode**, no container is removed or created, however at the end of the playbook run a list of commands is displayed to show the possible outcome of the playbook. This is useful for debugging purposes.

```
$ ansible-playbook haproxy.yaml --check
```

Adding the **diff mode** will show the changes that would have been made on containers by Ansible.

```
$ ansible-playbook haproxy.yaml --check --diff
```

The **tripleo_container_manage_clean_orphans** parameter is optional. It can be set to false meaning orphaned containers, with a specific **config_id**, will not be removed. It can be used to manage a single container without impacting other running containers with same **config_id**.

The **tripleo_container_manage_config_overrides** parameter is optional and can be used to override a specific container attribute, for example the image or the container user. The parameter creates dictionary with container name and the parameters to override. These parameters have to exist and they define the container configuration in TripleO Heat Templates.

Note the dictionary does not update the overrides in the JSON file so if an update or upgrade is executed, the container will be re-configured with the configuration in the JSON file.

CHAPTER 10. CONFIGURING THE OVERCLOUD WITH THE ORCHESTRATION SERVICE (HEAT)

You can use the Orchestration service (heat) to create custom overcloud configurations in heat templates and environment files.

10.1. 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.

10.1.1. 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
```

10.1.2. 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.

10.1.3. 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.

10.1.4. 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.

10.1.5. 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.

a. Stage all templates for the initial commit:

```
$ git add *
```

b. 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:

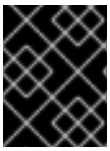
```
git checkout my-customizations
```

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 Section 10.3, "Configuration hooks" instead of modifying the heat template collection.

10.1.6. 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/ssl/enable-internal-tls.j2.yaml

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

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

Use case 2: Custom core templates

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

Environment file: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

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

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

Use case 3: Incorrect usage

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

Environment file: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

Director uses a custom core template location (**--templates /home/stack/tripleo-heat-installer-templates**). However, the chosen **enable-internal-tls.j2.yaml** is not located within the custom core templates, so it will not render into **enable-internal-tls.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**.

10.2. 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.

10.2.1. 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'
```

10.2.2. 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
```

10.2.3. 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.

10.2.4. 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"
```

10.2.5. 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 10.3.4, "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**

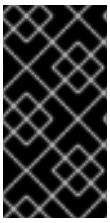
You set the **KeystoneNotificationFormat** in an overcloud environment file, which then sets the **notification_format** option in the **keystone.conf** file during the overcloud configuration.

10.3. 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.

10.3.1. 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 before the core configuration begins. These hooks include the following configurations:



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']
      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:

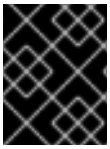
```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

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.

10.3.2. 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 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']
      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.

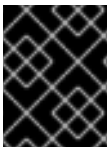
```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

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 \
    ...
```

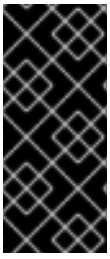
This applies the configuration to all nodes before the core configuration begins on either the initial overcloud creation or subsequent updates.



IMPORTANT

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

10.3.3. 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']
      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.

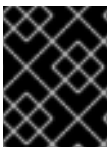
```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

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.

10.3.4. 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.

10.3.5. 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.

10.3.6. 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.