Red Hat OpenStack Platform 14

Quick Start Guide

Creating an all-in-one OpenStack cloud for test and proof-of-concept environments

Last Updated: 2019-02-11

Red Hat OpenStack Platform 14 Quick Start Guide

Creating an all-in-one OpenStack cloud for test and proof-of-concept environments

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

This guide contains information about installing, configuring, and deploying Red Hat OpenStack Platform 14 in a test environment with the Red Hat OpenStack Platform standalone environment. Use this guide to deploy a simple single-node OpenStack environment.

Table of Contents

PREFACE

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

CHAPTER 1. ALL-IN-ONE OPENSTACK INSTALLATION

The all-in-one installation method uses TripleO to deploy OpenStack and related services with a simple single-node environment. Use this installation to enable proof-of-concept, development, and test deployments on a single node with limited or no follow-up operations.

1.1. PREREQUISITES

Before you begin, ensure that you have a system with two network interfaces and a base operating system installed.

Example network configuration

- Interface **`eth0`** assigned to network 192.168.122.0/24
- Interface **`eth1`** assigned to network 192.168.25.0/24

CHAPTER 2. OVERVIEW

This section contains information about installing, configuring, and deploying a simple single-node OpenStack environment. In this scenario, there is no pre-existing Undercloud dependency. Instead, the installer runs an inline heat-all instance to bootstrap the deployment process and convert the selected Heat templates into Ansible playbooks that you can execute on a local machine.

Use the all-in-one OpenStack installation for basic testing and development. The all-in-one installation is a good starting point and test environment for OpenStack, but if you want to perform complex operations, you must deploy a production-level scaled cloud installation.

Workflow

To install, configure, and deploy a simple single-node OpenStack environment, complete the tasks in the following basic workflow:

1. Enable the repositories necessary to deploy the all-in-one OpenStack installation.
2. Install the TripleO command line interface (CLI).
3. Configure basic network configuration and deployment parameters for your all-in-one OpenStack installation.
4. Deploy the all-in-one OpenStack installation.

Benefits of the all-in-one OpenStack installation

- Composable services.
- Pre-defined roles.
- Condensed single-node environment.
- Playbooks that you can use to run the small-footprint installer in a container and generate Ansible playbooks.

Configuration

The all-in-one Red Hat OpenStack installation does not require configuration pre-deployment or post-deployment. However, this guide also contains information about configuring roles and services that you can use to experiment with OpenStack.

Composable roles

You can create custom composable roles and deploy specific services for each role.

Ansible

This installation applies Ansible playbooks automatically with the deployment command. However, you can also direct the deployment command to output Ansible playbooks that you can use on other instances. For example, you can complete testing in the all-in-one installation, and then apply the verified Ansible playbook to other instances.

CHAPTER 3. INSTALLING THE ALL-IN-ONE OPENSTACK ENVIRONMENT

Before you can begin configuring, deploying, and testing your all-in-one OpenStack environment, you must install the necessary packages and dependencies.

1. Log in as a non-root user to the bare metal or virtual machine where you want to install the standalone services:

```
$ ssh <non-root-user>@<host-machine>
```

2. Register the machine with Red Hat Subscription Manager. Enter your Red Hat subscription credentials at the prompt:

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

3. Attach your Red Hat subscription to the entitlement server:

```
$ sudo subscription-manager attach --auto
```

4. Run the following commands to enable the necessary repositories:

```
$ sudo yum install -y yum-utils
$ sudo yum-config-manager --enable rhel-7-server-rpms
$ sudo yum-config-manager --enable rhel-7-server-optional-rpms
$ sudo yum-config-manager --enable rhel-7-server-extras-rpms
$ sudo yum-config-manager --enable rhel-ha-for-rhel-7-server-rpms
$ sudo yum-config-manager --enable rhel-7-server-openstack-
<VERSION>-rpms
```

   Replace **<VERSION>** with the version of your OpenStack installation.

Version	Name
14	rocky

NOTE

The all-in-one OpenStack installation is a Technology Preview feature in Red Hat OpenStack Platform 14.

5. Install the TripleO command line interface (CLI):

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

6. Copy your public SSH key to the all-in-one host machine:

```
$ sudo ssh-copy-id <user>@<host-machine>
```

CHAPTER 4. CONFIGURING THE ALL-IN-ONE OPENSTACK INSTALLATION

The all-in-one Red Hat OpenStack Platform installation does not require configuration pre-deployment or post-deployment. However, you must create the following configuration files manually before you can deploy OpenStack:

- **$HOME/containers-prepare-parameters.yaml**
- **$HOME/standalone_parameters.yaml**

If you want to customize, develop, and test your all-in-one installation, edit the following configuration files:

- **/usr/share/openstack-tripleo-heat-templates/environments/standalone.yaml**
- **/usr/share/openstack-tripleo-heat-templates/roles/Standalone.yaml**

4.1. GENERATING YAML FILES FOR THE ALL-IN-ONE INSTALLATION

To generate the **containers-prepare-parameters.yaml** and **standalone_parameters.yaml** files, complete the following steps:

1. Generate the **containers-prepare-parameters.yaml** file that contains the default **ContainerImagePrepare** parameters:

```
$ openstack tripleo container image prepare default --output-env-file $HOME/containers-prepare-parameters.yaml
```

2. Create the **$HOME/standalone_parameters.yaml** file and configure basic parameters for your standalone OpenStack installation, including network configuration and some deployment options. In this example, network interface **eth0** is a default interface, and network interface **eth1** is the interface that you use to deploy OpenStack:

```
export IP=192.168.25.2
export NETMASK=24
export INTERFACE=eth1

cat <<EOF > $HOME/standalone_parameters.yaml
parameter_defaults:
  CloudName: $IP
  ControlPlaneStaticRoutes: []
  Debug: true
  DeploymentUser: $USER
  DnsServers:
    - 1.1.1.1
    - 8.8.8.8
  DockerInsecureRegistryAddress:
    - $IP:8787
  NeutronPublicInterface: $INTERFACE
  NeutronDnsDomain: localdomain
  NeutronBridgeMappings: datacentre:br-ctlplane
  NeutronPhysicalBridge: br-ctlplane
```

```
  StandaloneEnableRoutedNetworks: false
  StandaloneHomeDir: $HOME
  StandaloneLocalMtu: 1500
    EOF
```

You must configure the **DnsServers** parameter with your DNS address. For example, use the address from the **/etc/resolv.conf** file:

```
$ cat /etc/resolv.conf
192.168.122.1
```

If you have an internal time source, or if your environment blocks access to external time sources, use the **NtpServer** parameter to define the time source that you want to use:

```
NtpServer: clock.example.com
```

If you want to use the all-in-one OpenStack installation in a virtual environment, you must define the virtualization type with the **StandaloneExtraConfig** parameter:

```
StandaloneExtraConfig:
  nova::compute::libvirt::services::libvirt_virt_type: qemu
  nova::compute::libvirt::libvirt_virt_type: qemu
```

CHAPTER 5. DEPLOYING THE ALL-IN-ONE OPENSTACK INSTALLATION

To deploy your all-in-one OpenStack environment, run the following command. Ensure that you include all **`.yaml`** files relevant to your environment.

- Run the deploy command:

```
$ sudo openstack tripleo deploy \
  --templates \
  --local-ip=$IP/$NETMASK \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/standalone.yaml \
  -r /usr/share/openstack-tripleo-heat-
templates/roles/Standalone.yaml \
  -e $HOME/containers-prepare-parameters.yaml \
  -e $HOME/standalone_parameters.yaml \
  --output-dir $HOME \
  --standalone
```

At the end of a successful deployment, you can use the **`clouds.yaml`** configuration file in the **`/home/$USER/.config/openstack`** directory to query and verify the OpenStack services:

```
export OS_CLOUD=standalone
openstack endpoint list
```

CHAPTER 6. CREATING ANSIBLE PLAYBOOKS WITH THE ALL-IN-ONE INSTALLATION

The deployment command applies Ansible playbooks to the environment automatically. However, you can modify the deployment command to generate Ansible playbooks without applying them to the deployment, and run the playbooks later.

Include the **`--output-only`** option in the deploy command to generate the **`undercloud-ansible-XXXXX`** directory. This directory contains a set of Ansible playbooks that you can run on other hosts.

1. To generate the Ansible playbook directory, run the deploy command with the option **`--output-only`**:

```
$ sudo openstack tripleo deploy \
  --templates \
  --local-ip=$IP/$NETMASK \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/standalone.yaml \
  -r /usr/share/openstack-tripleo-heat-
templates/roles/Standalone.yaml \
  -e $HOME/containers-prepare-parameters.yaml \
  -e $HOME/standalone_parameters.yaml \
  --output-dir $HOME \
  --standalone
  --output-only
```

2. To run the Ansible playbooks, run the **`ansible-playbook`** command, and include the **`inventory.yaml`** file and the **`deploy_steps_playbook.yaml`** file:

```
$ cd undercloud-ansible-XXXXX
$ ansible-playbook -i inventory.yaml deploy_steps_playbook.yaml
```

CHAPTER 7. WORKING WITH HEAT TEMPLATES

The custom configurations in this guide use Heat templates and environment files to define certain aspects of the Overcloud. This chapter contains 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. The purpose of a template is to define and create a stack, which is a collection of resources that Heat creates, and the configuration of the resources. Resources are objects in OpenStack and can include compute resources, network configurations, security groups, scaling rules, and custom resources.

The structure of a Heat template has three main sections:

Parameters

Parameters are settings passed to Heat. Use these parameters to define and customize both default and non-default values. Define these parameters in the parameters section of a template.

Resources

Resources are the specific objects that you want to create and configure as part of a stack. OpenStack contains a set of core resources that span across all components. Define resources in the resources section of a template.

Output

These are values passed from Heat after the stack creation. You can access these values either through the Heat API or through the client tools. Define these values in the output section of a template.

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

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

7.1. CORE HEAT TEMPLATES

Red Hat OpenStack Platform contains a core Heat template collection for the Overcloud. You can find this collection in the **`/usr/share/openstack-tripleo-heat-templates`** directory.

There are many Heat templates and environment files in this collection. This section contains information about the main files and directories that you can use to customize your deployment.

overcloud.j2.yaml

This file is the main template file used to create the Overcloud environment. This file uses Jinja2 syntax and iterates 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 file is the main environment file used to create the Overcloud environment. This file contains a set of configurations for Puppet modules on the Overcloud image. After the director writes the Overcloud image to each node, Heat starts the Puppet configuration for each node using the resources registered in this environment file. This file uses Jinja2 syntax and iterates 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 definitions of the roles in an overcloud, and maps services to each role.

network_data.yaml

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

plan-environment.yaml

This file contains definitions of the metadata for your overcloud plan, including the plan name, the main template that you want to use, and environment files that you want to apply to the overcloud.

capabilities-map.yaml

This file contains a mapping of environment files for an overcloud plan. Use this file to describe and enable environment files in the director web UI. If you include custom environment files in the **`environments`** directory but do not define these files in the **`capabilities-map.yaml`** file, you can find these environment files in the **Other** sub-tab of the **Overall Settings** page on the web UI.

environments

This directory contains additional Heat environment files that you can use with your Overcloud creation. These environment files enable extra functions for your OpenStack environment. For example, you can use the **`cinder-netapp-config.yaml`** environment file to enable Cinder NetApp backend storage. If you include custom environment files in the **`environments`** directory but do not define these files in the **`capabilities-map.yaml`** file, you can find these environment files in the **Other** sub-tab of the **Overall Settings** page on the web UI.

network

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

puppet

This directory contains puppet templates. The **`overcloud-resource-registry-puppet.j2.yaml`** environment file uses the files in the **`puppet`** directory to drive the application of the Puppet configuration on each node.

puppet/services

This directory contains Heat templates for all services in the composable service architecture.

extraconfig

This directory contains templates that you can use to enable extra functionality. For example, you can use the **`extraconfig/pre_deploy/rhel-registration`** directory to register your nodes with the Red Hat Content Delivery network, or with your own Red Hat Satellite server.

CHAPTER 8. WORKING WITH CUSTOM ROLES AND SERVICES

Red Hat OpenStack Platform usually consists of nodes in pre-defined roles, for example, nodes in Controller roles, Compute roles, and different storage role types. Each of these default roles contain a set of services that you define in the core Heat template collection. However, the all-in-one OpenStack installation runs on a single node that contains all of the OpenStack services. The **`Standalone.yaml`** role file in the **`/usr/share/openstack-tripleo-heat-templates/roles`** directory is the configuration file that contains all of the services in the all-in-one installation. You can duplicate and modify the **`Standalone.yaml`** role file to enable and disable services in your installation.

The **`Standalone.yaml`** file contains a list of services in a role **`Standalone`**. Use the following example to understand the syntax of this file:

```
- name: Standalone
  description: |
    A standalone role that includes a minimal set of services. Use this
role for testing in a single node configuration with the 'openstack
tripleo deploy --standalone' command, or with the 'openstack overcloud
deploy' command.
  CountDefault: 1
  tags:
    - primary
    - controller
  disable_constraints: True
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AodhApi
    - OS::TripleO::Services::AodhEvaluator
    ...
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::Zaqar
```

Include this role file in the deployment command to configure your stack with the Standalone role that contains the services that you include in the **`ServicesDefault:`** section of the role file:

```
$ sudo openstack tripleo deploy --templates -r /usr/share/openstack-
tripleo-heat-templates/roles/Standalone.yaml
```

However, in a production multi-node OpenStack environment, you assign each node with a role that contains a portion of the OpenStack services, rather than including all services on a single node. For example, the default Controller role includes administration, networking, and high availability services, and the default Compute role includes computing services. The default role file in a multi-node environment is the **`/usr/share/openstack-tripleo-heat-templates/roles_data.yaml`** file. This file defines the following role types:

- Controller
- Compute
- BlockStorage
- ObjectStorage

- CephStorage

Use the following example to understand role syntax in a multi-node OpenStack environment:

```
- name: Controller
  description: |
    Controller role that contains all of the services for 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
    ...
```

You must include the role file each time you run the deployment command. You can use the **-r** argument in the deployment command to override this file and use a custom role file:

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

8.1. ENABLING AND DISABLING SERVICES WITH THE ALL-IN-ONE OPENSTACK DEPLOYMENT

The **Standalone.yaml** role file in the **/usr/share/openstack-tripleo-heat-templates/roles** directory is the configuration file that contains all of the services in the all-in-one OpenStack installation. To enable or disable services in your environment, complete the following steps:

Procedure

1. To disable a service, edit the **/usr/share/openstack-tripleo-heat-templates/roles/Standalone.yaml** file and add the value **OS::Heat::None** to the service that you want to disable:

   ```
       - OS::TripleO::Services::Aide: OS::Heat::None
   ```

2. To enable a service, edit the **/usr/share/openstack-tripleo-heat-templates/roles/Standalone.yaml** file and remove the value **OS::Heat::None** from the service that you want to enable:

   ```
       - OS::TripleO::Services::Aide
   ```

CHAPTER 9. EXAMPLES

Use the following examples to understand how to launch a compute instance post-deployment with various network configurations.

9.1. EXAMPLE 1: LAUNCHING A COMPUTE NODE WITH ONE NIC ON THE TENANT AND PROVIDER NETWORKS

Follow this example to understand how to launch a Compute node with the private tenant network and the provider network after you deploy the all-in-one OpenStack installation. This example is based on a single NIC configuration and requires at least three IP addresses.

Prerequisites

To complete this example successfully, you must have the following IP addresses available in your environment:

- One IP address for the OpenStack services.
- One IP address for the virtual router to provide connectivity to the tenant network. This IP address is assigned automatically in this example.
- At least one IP address for floating IPs on the provider network.

Procedure

1. Create configuration helper variables:

```
# standalone with tenant networking and provider networking
export OS_CLOUD=standalone
export GATEWAY=192.168.24.1
export STANDALONE_HOST=192.168.24.2
export PUBLIC_NETWORK_CIDR=192.168.24.0/24
export PRIVATE_NETWORK_CIDR=192.168.100.0/24
export PUBLIC_NET_START=192.168.24.4
export PUBLIC_NET_END=192.168.24.5
export DNS_SERVER=1.1.1.1
```

2. Create a basic flavor:

```
$ openstack flavor create --ram 512 --disk 1 --vcpu 1 --public tiny
```

3. Download CirrOS and create an OpenStack image:

```
$ wget https://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-
disk.img
$ openstack image create cirros --container-format bare --disk-
format qcow2 --public --file cirros-0.4.0-x86_64-disk.img
```

4. Configure SSH:

```
$ ssh-keygen
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub default
```

5. Create a simple network security group:

```
$ openstack security group create basic
```

6. Configure the new network security group:

   a. Enable SSH:

   ```
   $ openstack security group rule create basic --protocol tcp --
   dst-port 22:22 --remote-ip 0.0.0.0/0
   ```

   b. Enable ping:

   ```
   $ openstack security group rule create --protocol icmp basic
   ```

   c. Enable DNS:

   ```
   $ openstack security group rule create --protocol udp --dst-port
   53:53 basic
   ```

7. Create Neutron networks:

```
$ openstack network create --external --provider-physical-network
datacentre --provider-network-type flat public
$ openstack network create --internal private
$ openstack subnet create public-net \
    --subnet-range $PUBLIC_NETWORK_CIDR \
    --no-dhcp \
    --gateway $GATEWAY \
    --allocation-pool start=$PUBLIC_NET_START,end=$PUBLIC_NET_END \
    --network public
$ openstack subnet create private-net \
    --subnet-range $PRIVATE_NETWORK_CIDR \
    --network private
```

8. Create a virtual router:

```
# NOTE: In this case an IP will be automatically assigned
# from the allocation pool for the subnet.
$ openstack router create vrouter
$ openstack router set vrouter --external-gateway public
$ openstack router add subnet vrouter private-net
```

9. Create a floating IP:

```
$ openstack floating ip create public
```

10. Launch the instance:

```
$ openstack server create --flavor tiny --image cirros --key-name
default --network private --security-group basic myserver
```

11. Assign the floating IP:

```
$ openstack server add floating ip myserver <FLOATING_IP>
```

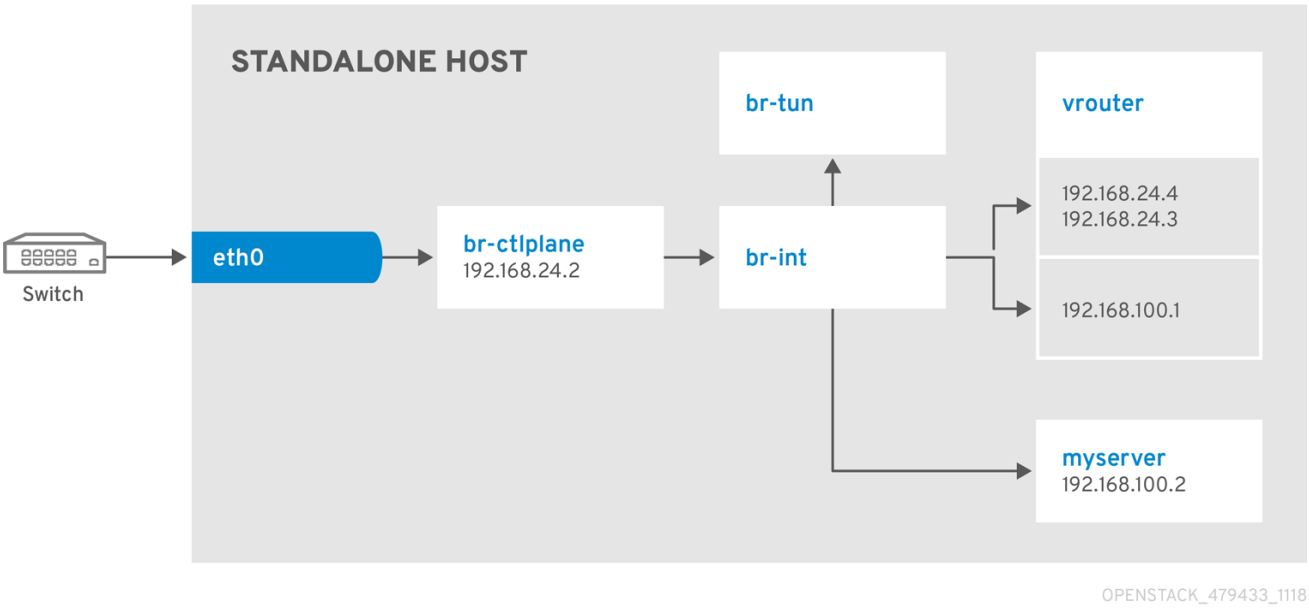
Replace **FLOATING_IP** with the address of the floating IP that you create in a previous step.

12. Test SSH:

```
ssh cirros@<FLOATING_IP>
```

Replace **FLOATING_IP** with the address of the floating IP that you create in a previous step.

Network Architecture



9.2. EXAMPLE 2: LAUNCHING A COMPUTE NODE WITH ONE NIC ON THE PROVIDER NETWORK

Follow this example to understand how to launch a Compute node with the provider network after you deploy the all-in-one OpenStack installation. This example is based on a single NIC configuration and requires at least four IP addresses.

Prerequisites

To complete this example successfully, you must have the following IP addresses available in your environment:

- One IP address for the OpenStack services.
- One IP address for the virtual router to provide connectivity to the tenant network. This IP address is assigned automatically in this example.
- One IP address for DHCP on the provider network.
- At least one IP address for floating IPs on the provider network.

Procedure

1. Create configuration helper variables:

```
# standalone with tenant networking and provider networking
export OS_CLOUD=standalone
```

```
export GATEWAY=192.168.24.1
export STANDALONE_HOST=192.168.24.2
export VROUTER_IP=192.168.24.3
export PUBLIC_NETWORK_CIDR=192.168.24.0/24
export PUBLIC_NET_START=192.168.24.4
export PUBLIC_NET_END=192.168.24.5
export DNS_SERVER=1.1.1.1
```

2. Create a basic flavor:

```
$ openstack flavor create --ram 512 --disk 1 --vcpu 1 --public tiny
```

3. Download CirrOS and create an OpenStack image:

```
$ wget https://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-
disk.img
$ openstack image create cirros --container-format bare --disk-
format qcow2 --public --file cirros-0.4.0-x86_64-disk.img
```

4. Configure SSH:

```
$ ssh-keygen
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub default
```

5. Create a simple network security group:

```
$ openstack security group create basic
```

6. Configure the new network security group:

   a. Enable SSH:

```
$ openstack security group rule create basic --protocol tcp --
dst-port 22:22 --remote-ip 0.0.0.0/0
```

   b. Enable ping:

```
$ openstack security group rule create --protocol icmp basic
```

   c. Enable DNS:

```
$ openstack security group rule create --protocol udp --dst-port
53:53 basic
```

7. Create Neutron networks:

```
$ openstack network create --external --provider-physical-network
datacentre --provider-network-type flat public
$ openstack network create --internal private
$ openstack subnet create public-net \
    --subnet-range $PUBLIC_NETWORK_CIDR \
    --gateway $GATEWAY \
    --allocation-pool start=$PUBLIC_NET_START,end=$PUBLIC_NET_END \
```

```
    --network public \
    --host-route destination=169.254.169.254/32,gateway=$VROUTER_IP
\
    --host-route destination=0.0.0.0/0,gateway=$GATEWAY \
    --dns-nameserver $DNS_SERVER
```

8. Create a virtual router:

```
# NOTE: In this case an IP will be automatically assigned
# from the allocation pool for the subnet.
$ openstack router create vrouter
$ openstack port create --network public --fixed-ip subnet=public-
net,ip-address=$VROUTER_IP vrouter-port
$ openstack router add port vrouter vrouter-port
```

9. Launch the instance:

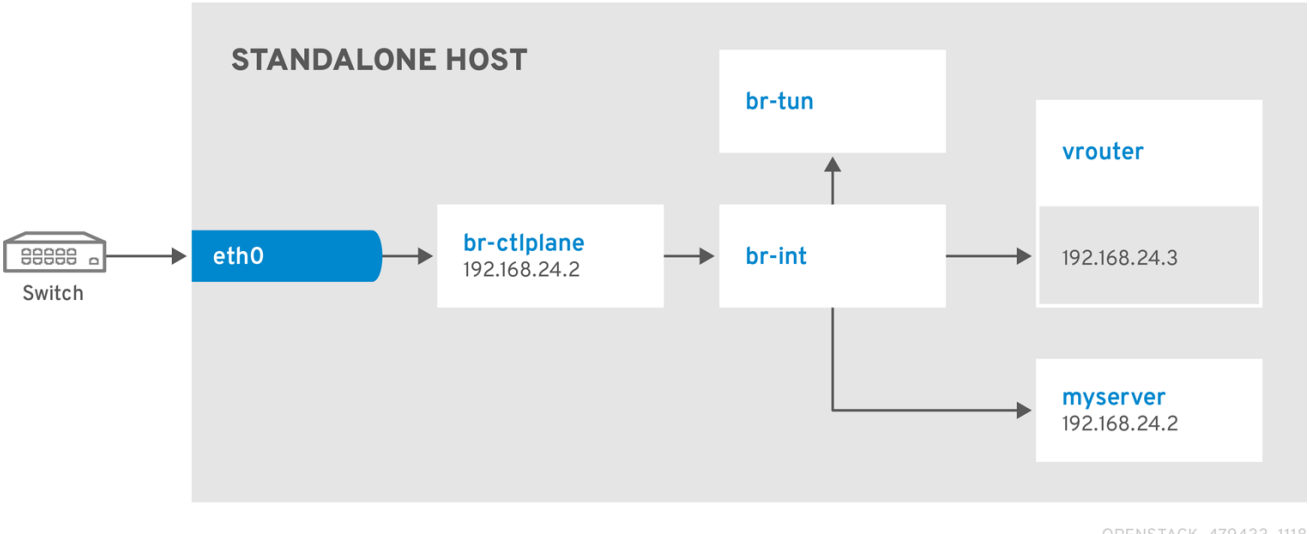
```
$ openstack server create --flavor tiny --image cirros --key-name
default --network public --security-group basic myserver
```

10. Test SSH:

```
ssh cirros@<VM_IP>
```

Replace **VM_IP** with the address of the virtual machine that you create in the previous step.

Network Architecture



9.3. EXAMPLE 3: LAUNCHING A COMPUTE NODE WITH TWO NICS ON THE TENANT AND PROVIDER NETWORKS

Follow this example to understand how to launch a Compute node with the private tenant network and the provider network after you deploy the all-in-one OpenStack installation. This example is based on a dual NIC configuration and requires at least three IP addresses on the provider network.

Prerequisites

- One IP address for a gateway on the provider network.
- One IP address for OpenStack endpoints.
- One IP address for the virtual router to provide connectivity to the tenant network. This IP address is assigned automatically in this example.
- At least one IP address for floating IPs on the provider network.

Procedure

1. Create configuration helper variables:

```
# standalone with tenant networking and provider networking
export OS_CLOUD=standalone
export GATEWAY=192.168.24.1
export STANDALONE_HOST=192.168.0.2
export PUBLIC_NETWORK_CIDR=192.168.24.0/24
export PRIVATE_NETWORK_CIDR=192.168.100.0/24
export PUBLIC_NET_START=192.168.0.3
export PUBLIC_NET_END=192.168.24.254
export DNS_SERVER=1.1.1.1
```

2. Create a basic flavor:

```
$ openstack flavor create --ram 512 --disk 1 --vcpu 1 --public tiny
```

3. Download CirrOS and create an OpenStack image:

```
$ wget https://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-
disk.img
$ openstack image create cirros --container-format bare --disk-
format qcow2 --public --file cirros-0.4.0-x86_64-disk.img
```

4. Configure SSH:

```
$ ssh-keygen
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub default
```

5. Create a simple network security group:

```
$ openstack security group create basic
```

6. Configure the new network security group:

   a. Enable SSH:

```
$ openstack security group rule create basic --protocol tcp --
dst-port 22:22 --remote-ip 0.0.0.0/0
```

   b. Enable ping:

```
$ openstack security group rule create --protocol icmp basic
```

c. Enable DNS:

```
$ openstack security group rule create --protocol udp --dst-port
53:53 basic
```

7. Create Neutron networks:

```
$ openstack network create --external --provider-physical-network
datacentre --provider-network-type flat public
$ openstack network create --internal private
$ openstack subnet create public-net \
    --subnet-range $PUBLIC_NETWORK_CIDR \
    --no-dhcp \
    --gateway $GATEWAY \
    --allocation-pool start=$PUBLIC_NET_START,end=$PUBLIC_NET_END \
    --network public
$ openstack subnet create private-net \
    --subnet-range $PRIVATE_NETWORK_CIDR \
    --network private
```

8. Create a virtual router:

```
# NOTE: In this case an IP will be automatically assigned
# from the allocation pool for the subnet.
$ openstack router create vrouter
$ openstack router set vrouter --external-gateway public
$ openstack router add subnet vrouter private-net
```

9. Create a floating IP:

```
$ openstack floating ip create public
```

10. Launch the instance:

```
$ openstack server create --flavor tiny --image cirros --key-name
default --network private --security-group basic myserver
```

11. Assign the floating IP:

```
$ openstack server add floating ip myserver <FLOATING_IP>
```

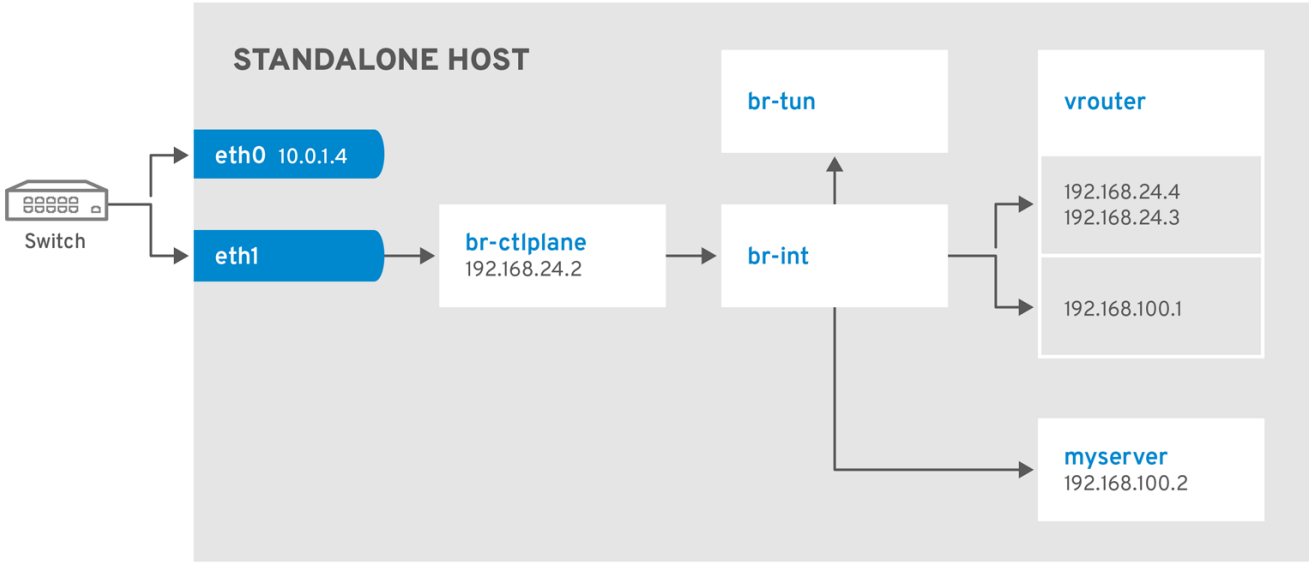
Replace **FLOATING_IP** with the address of the floating IP that you create in a previous step.

12. Test SSH:

```
ssh cirros@<FLOATING_IP>
```

Replace **FLOATING_IP** with the address of the floating IP that you create in a previous step.

Network Architecture



OPENSTACK_479433_1118