# Reference Architectures 2017

## Deploying and Managing Red Hat OpenShift Container Platform 3.6 on Red Hat OpenStack Platform 10

Last Updated: 2018-07-26

# Reference Architectures 2017 Deploying and Managing Red Hat OpenShift Container Platform 3.6 on Red Hat OpenStack Platform 10

Roger Lopez

Ryan Cook

Eduardo Minguez
refarch-feedback@redhat.com

## Legal Notice

## Abstract

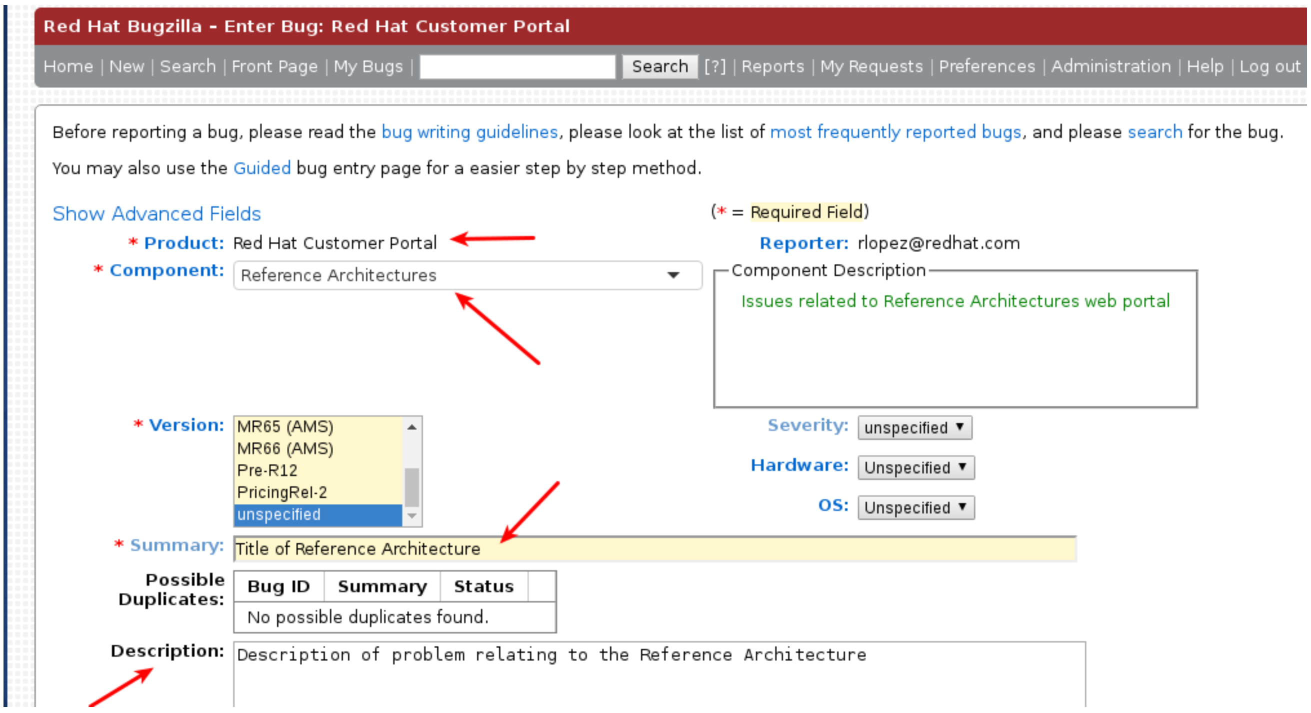
The purpose of this document is to provide guidelines and considerations for installing and configuring Red Hat OpenShift Container Platform 3.6 on Red Hat OpenStack Platform 10.

# Table of Contents

# COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

When filing a bug report for a reference architecture, create the bug under the Red Hat Customer Portal product and select the Component labeled Reference Architectures. Within the Summary, enter the title of the reference architecture. Within the Description, provide a URL (if available) along with any feedback.


Red Hat Bugzilla – Enter Bug: Red Hat Customer Portal

Home | New | Search | Front Page | My Bugs | Search [?] | Reports | My Requests | Preferences | Administration | Help | Log out

Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

You may also use the Guided bug entry page for a easier step by step method.

(* = Required Field)

* Product: Red Hat Customer Portal

Reporter: rlopez@redhat.com

* Component: Reference Architectures

Component Description

Issues related to Reference Architectures web portal

* Version: MR65 (AMS), MR66 (AMS), Pre-R12, PricingRel-2, unspecified

Severity: unspecified

Hardware: Unspecified

OS: Unspecified

* Summary: Title of Reference Architecture

Possible Duplicates:

| Bug ID | Summary | Status |
|---|---|---|
| No possible duplicates found. | | |

Description: Description of problem relating to the Reference Architecture

# CHAPTER 1. EXECUTIVE SUMMARY

As businesses focus on meeting customer demands and needs, building solutions that are not only robust, delivered in a timely manner, secure, and supported are of the utmost importance. In order to meet these requirements, organizations must provide the capabilities to facilitate faster development life cycles by managing and maintaining multiple products to meet each of their business needs. This process is simplified by the use of Red Hat solutions — specifically Red Hat OpenShift Container Platform on Red Hat OpenStack Platform. This is accomplished by Red Hat OpenShift Container Platform providing a Platform as a Service (PaaS) solution that allows for the development, deployment and management of container based applications while standing on top of a Red Hat OpenStack Platform framework that provides Infrastructure as a Service (IaaS) to run these container based applications securely within a privately owned cloud.

This reference architecture provides a methodology on how to deploy a highly available Red Hat OpenShift Container Platform on Red Hat OpenStack Platform environment by including a step-by-step solution along with best practices on customizing Red Hat OpenShift Container Platform.

This reference architecture is suited for system administrators, Openstack administrators running Openshift, Openshift administrators, and IT architects building Red Hat OpenShift Container Platform on Red Hat OpenStack Platform environments.

# CHAPTER 2. REFERENCE ARCHITECTURE OVERVIEW

Red Hat OpenShift Container Platform is a container application development and hosting platform that provides compute services on demand. It allows developers to create and deploy applications by delivering a consistent environment for both development and during the runtime life cycle that requires no server management.

The deployment of Red Hat OpenShift Container Platform varies among several factors that impact the installation process. Key considerations include:

- *Which installation method do you want to use?*
- *How many hosts do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other technologies?*

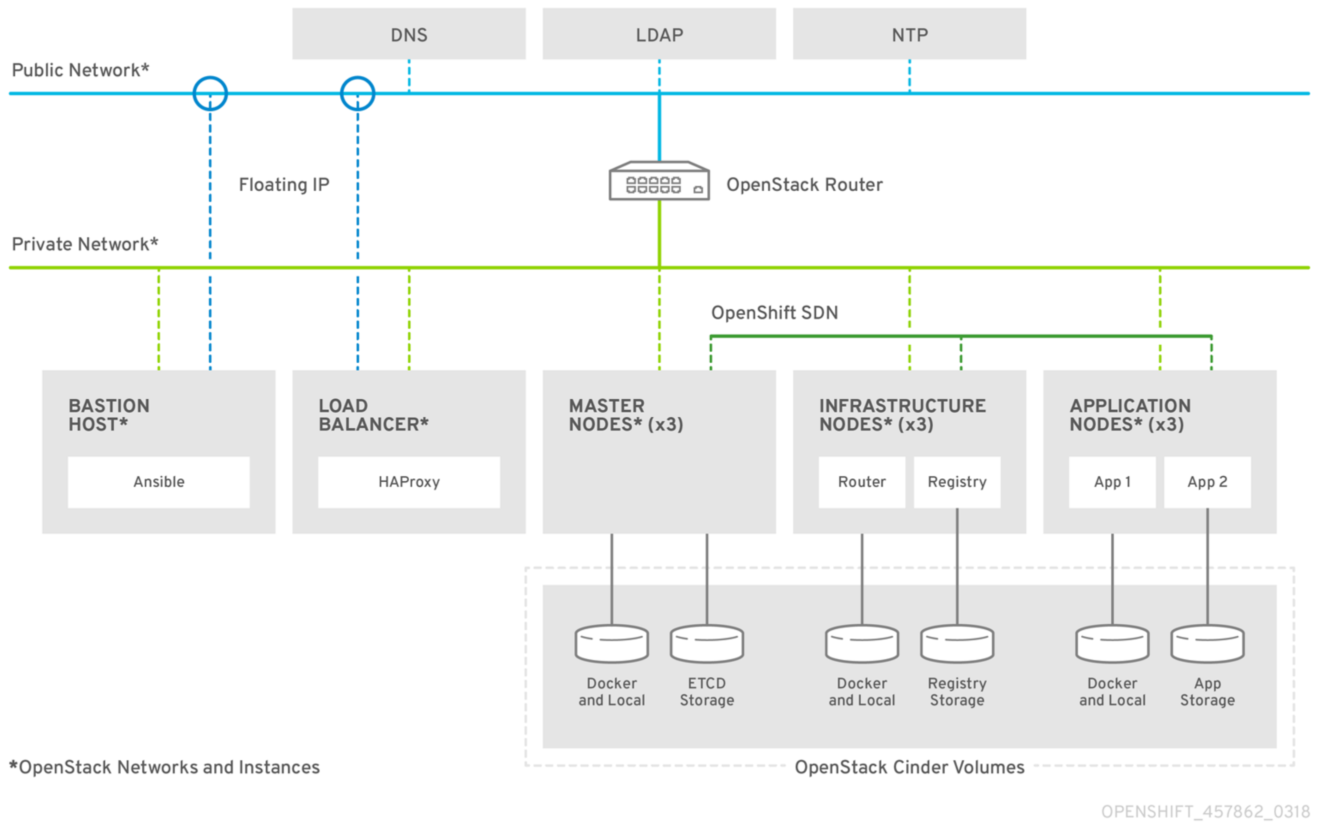
For more information regarding the different options in installing an Red Hat OpenShift Container Platform cluster visit: Red Hat OpenShift Container Platform Chapter 2. Installing a Cluster

The initial planning process for this reference architecture answers these questions for this environment as follows:

- *Which installation method do you want to use?* Advanced Installation
- *How many hosts do you require in the cluster?* 10
- *Is high availability required?* Yes
- *Which installation type do you want to use: RPM or containerized?* RPM
- *Is my installation supported if integrating with other technologies?* Yes

A pictorial representation of the environment in this reference environment is shown below.

**Figure 2.1. OpenShift Container Platform Architecture**



The Red Hat OpenShift Container Platform Architecture diagram shows the different hosts and networks involved in the reference architecture.

**NOTE**

The reference architecture contains the steps to create everything in the picture and an optional DNS within the internal network but it doesn’t include the creation of the NTP and LDAP servers nor the Red Hat OpenStack Platform environment.

There are three networks used in the reference architecture:

- OpenStack Internal network - provides an isolated network for Red Hat OpenShift Container Platform on its own Red Hat OpenStack Platform tenant. It routes the public network providing Red Hat OpenStack Platform instances the ability to access external resources if a floating IP is assigned.
- OpenStack Public network - provides floating IPs to the instances attached to the internal network, in this reference architecture, the bastion host and the load balancer.
- OpenShift SDN, a custom overlay network that uses Open vSwitch (OVS) that offers fine-grained dynamic routing. The SDN plug-in `ovs-mulitenant` allows for network isolation by default.

The environment prerequisites:

- LDAP host
- NTP host

- DNS host
- Load balancer host

Those instances are deployed in Red Hat OpenStack Platform using the instructions provided by this reference architecture (except for the LDAP and NTP servers) as a guide.

The Red Hat OpenShift Container Platform instances:

- Bastion instance
- Three master instances
- Three infrastructure instances
- Three application instances

## 2.1. RED HAT OPENSTACK PLATFORM INSTANCES

The reference environment in this document consists of the following OpenStack instances:

- a single *bastion* instance
- three *master* instance
- six *node* instance (3 *infrastructure* nodes and 3 *application* nodes)

The *bastion* host serves as the installer of the Ansible playbooks that deploy Red Hat OpenShift Container Platform as well as an entry point for management tasks.

The master nodes contain the master components, including the API server, controller manager server, and **`etcd`**. The master manages nodes in its Kubernetes cluster and schedules pods to run on nodes.

The nodes provide the runtime environments for the containers. Each node in a cluster has the required services to be managed by the master nodes. Nodes have the required services to run pods.

For more information about the differences between master and nodes see OpenShift Documentation - Kubernetes Infrastructure.

### 2.1.1. Master Instances and Components

**Table 2.1. Master Components[1]**

| Component | Description |
|---|---|
| API Server | The Kubernetes API server validates and configures the data for pods, services, and replication controllers. It also assigns pods to nodes and synchronizes pod information with service configuration. |

| Component | Description |
|---|---|
| **`etcd`** | **`etcd`** stores the persistent master state while other components watch **`etcd`** for changes to bring themselves into the desired state. **`etcd`** can be optionally configured for high availability, typically deployed with 2n+1 peer services. |
| Controller Manager Server | The controller manager server watches **`etcd`** for changes to replication controller objects and then uses the API to enforce the desired state. Can be run as a standalone process. Several such processes create a cluster with one active leader at a time. |

When using the **`native`** high availability method with HAProxy, master components have the following availability.

**Table 2.2. Availability Matrix with HAProxy[1]**

| Role | Style | Notes |
|---|---|---|
| API Server | Active-Active | Managed by HAProxy |
| Controller Manager Server | Active-Passive | One instance is elected as a cluster lead at a time |

1: OpenShift Documentation - Kubernetes Infrastructure

### NOTE

In environments with a single master instance, the API servers and controllers run in the same process (**`atomic-openshift-master`**). In HA environments, the API process is named **`atomic-openshift-master-api`** where the controllers **`atomic-openshift-master-controllers`**.

### 2.1.2. Node Instances and Components

The *node* instances run containers on behalf of its users and are separated into two functional classes: *infrastructure* and *application* nodes. The infrastructure (*infra*) nodes run the *OpenShift router* , *OpenShift logging* , *OpenShift metrics* , and the *OpenShift registry* while the application (*app*) nodes host the user container processes.

The Red Hat OpenShift Container Platform SDN requires Master instances to be considered nodes, therefore, all nodes run the **`atomic-openshift-node`** service.

The routers are an important component of Red Hat OpenShift Container Platform as they are the entry point to applications running in Red Hat OpenShift Container Platform. Developers can expose their applications running in Red Hat OpenShift Container Platform to be available from outside the Red Hat OpenShift Container Platform SDN creating routes that are published in the routers. Once the traffic reaches the router, the router forwards traffic to the containers on the *app* nodes using the Red Hat OpenShift Container Platform SDN.

In this reference architecture a set of three routers are deployed on the *infra* nodes for high availability purposes. In order to have a single entry point for applications, a load balancer is created as part of the installation in the same Red Hat OpenStack Platform tenant where it can reach the infrastructure nodes for load balancing the incoming traffic to the routers running in the *infra* nodes.

**NOTE**

If using an external load balancer, the *infra* nodes require a public IP address in order to communicate with the load balancer host.

**Table 2.3. Node Instance types**

| Type | Node role |
|---|---|
| Master | None (to be part of the SDN) |
| Infrastructure node | Host Red Hat OpenShift Container Platform infrastructure pods (registry, router, logging, metrics, …) |
| Application node | Host applications deployed on Red Hat OpenShift Container Platform |

## 2.1.3. Instance Storage

**`Cinder`** volumes attach to instances to provide additional disk space. Red Hat OpenShift Container Platform requires at least 40 GB of space available on the master nodes and at least 15GB on the infrastructure and application nodes for the **`/var`** partition. This reference architecture breaks out **`/var/lib/etcd`**, **`docker`** storage, and ephemeral pod storage to provide individual storage for those specific directories.

Each *master* node Red Hat OpenStack Platform instance creates and mounts **`cinder`** volumes using the steps provided in this reference architecture. One volume contains a volume for **`etcd`** storage. The *master* nodes while not schedulable by default, contain **`cinder`** volumes for both **`docker`** storage and Red Hat OpenShift Container Platform pod storage.

Each *infra* and *app* node Red Hat OpenStack Platform instance creates and mounts **`cinder`** volumes using the steps provided in this reference architecture. One volume contains the **`docker`** runtime disk overhead to store the docker container images. The other **`cinder`** volume is created for pod storage. Each node offers **`cinder`** access to the containers it hosts through Red Hat OpenShift Container Platform.

**Table 2.4. Instances storage**

| Instance type | Role |
|---|---|
| Masters only | **`etcd`** data |
| Masters, infra and app nodes | Store docker images |
| Masters, infra and app nodes | Pod local storage |

## 2.2. LOAD BALANCERS

This guide uses an external load balancer running **`haproxy`** to offer a single entry point for the many Red Hat OpenShift Container Platform components. Organizations can provide their own currently deployed load balancers in the event that the service already exists.

**NOTE**

Red Hat OpenStack Platform 10 includes a Load Balancing-as-a-Service (LBaaS) that enables OpenStack Networking to distribute incoming requests evenly between designated instances, but there are some current issues preventing the use of LBaaS for Red Hat OpenShift Container Platform

The Red Hat OpenShift Container Platform console, provided by the Red Hat OpenShift Container Platform *master* nodes, can be spread across multiple instances to provide both load balancing and high availability properties.

Application traffic passes through the Red Hat OpenShift Container Platform Router on its way to the container processes. The Red Hat OpenShift Container Platform Router is a reverse proxy service container that multiplexes the traffic to multiple containers making up a scaled application running inside Red Hat OpenShift Container Platform. The load balancer used by *infra* nodes acts as the public view for the Red Hat OpenShift Container Platform applications.

The destination for the master and application traffic must be set in the load balancer configuration after each instance is created, the floating IP address is assigned and before the installation. A single **`haproxy`** load balancer can forward both sets of traffic to different destinations.

## 2.3. DNS

DNS service is an important component in the Red Hat OpenShift Container Platform environment. Regardless of the provider of DNS, an organization is required to have certain records in place to serve the various Red Hat OpenShift Container Platform components.

**NOTE**

Red Hat OpenStack Platform 10 includes a DNS-as-a-Service (DNSaaS) component, also known as Designate, but it is currently in Technology Preview phase

Since the load balancer values for the Red Hat OpenShift Container Platform master service and infrastructure nodes running router pods are known beforehand, entries must be configured into the DNS prior to starting the deployment procedure.

Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance* provides setup steps on how to configure a DNS service for the use of Red Hat OpenShift Container Platform.

## 2.4. RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKING

Red Hat OpenShift Container Platform offers the ability to specify how containers communicate with each other. This could be through the use of Red Hat provided Software-defined networks (SDN), third-party SDN, or **`flannel`**.

Deciding on the appropriate internal network for an Red Hat OpenShift Container Platform step is a crucial step. Unfortunately, there is no right answer regarding the appropriate network to chose, as this varies based upon the specific scenario requirements on how a Red Hat OpenShift Container Platform environment is to be used.

For the purposes of this reference environment, the Red Hat OpenShift Container Platform **`ovs-multitenant`** SDN plug-in is chosen due to its ability to provide tenant isolation. The following Section 2.4.1, "OpenShift SDN vs Flannel" section discusses pros and cons when deciding between two popular mechanisms for the internal networks - **`Openshift SDN`** and **`flannel`**.

### 2.4.1. OpenShift SDN vs Flannel

Red Hat OpenShift Container Platform offers two mechanisms to operate the internal network, **`OpenShift SDN`** and **`flannel`**.

The default mechanism is a software-defined network (SDN). This is a custom overlay network that uses Open vSwitch (OVS) that offers fine-grained dynamic routing. The SDN plug-in **`ovs-mulitenant`** allows for network isolation by default.

**`ovs-mulitenant`** SDN plug-in is a perfect choice as an internal network when traffic isolation between containers, services, and projects is important. However, this isolation comes at a cost in performance. Performance is degraded due to double encapsulation when deploying Red Hat OpenShift Container Platform on Red Hat OpenStack Platform because all Red Hat OpenShift Container Platform internode traffic is carried over a Red Hat OpenStack Platform **`neutron`** network. **`neutron`** is itself a SDN that provides network access to all the Red Hat OpenStack Platform instances.

**`flannel`** is a virtual networking layer designed for containers. Instead of having two SDN to route traffic, **`flannel`** runs in a mode labeled **`host-gw`** that routes packets by each node in the Red Hat OpenShift Container Platform environment running a daemon labeled **`flanneld`** that sends information to a centralized location (**`etcd`** store) thus allowing other nodes running the **`flanneld`** daemon to route packets to other containers as long as they are all on the **`flannel`** network. The main benefit to **`flannel`** is that it provides performance benefits over OVS double encapsulation since it does not create its own SDN on top of the existing RHOSP **`neutron`** network.

**Why not just default to `flannel`?**

**`flannel`** uses a single IP network space for all of the containers allocating a contiguous subset of the space to each instance. Consequently, nothing prevents a container from attempting to contact any IP address in the same network space. This hinders multi-tenancy because the network cannot be used to isolate containers in one application from another.

The key to making the right decision rests between mutli-tenancy isolation vs performance. Depending on the priority of each of these, should determine the appropriate choice when deciding between **`OpenShift SDN`** and **`flannel`** internal networks.

The steps to setup **`flannel`** instead of OpenShift **`ovs-multitenant`** can be found within the Appendix B, *Using Flannel*.

# CHAPTER 3. RED HAT OPENSHIFT CONTAINER PLATFORM PREREQUISITES

A successful deployment of Red Hat OpenShift Container Platform requires many prerequisites. This consists of a set of infrastructure and host configuration steps prior to the actual installation of Red Hat OpenShift Container Platform using Ansible. In the following subsequent sections, details regarding the prerequisites and configuration changes required for an Red Hat OpenShift Container Platform on a Red Hat OpenStack Platform environment are discussed in detail.

**NOTE**

All the Red Hat OpenStack Platform CLI commands in this reference environment are executed using the CLI **`openstack`** commands within the Red Hat OpenStack Platform director node. If using a workstation or laptop to execute these commands instead of the Red Hat OpenStack Platform director node, please ensure to install the **`python-openstackclient`** RPM found within the **`rhel-7-server-openstack-10-rpms`** repository.

Example:

Install **`python-openstackclient`** package.

```
$ sudo subscription-manager repos --enable rhel-7-server-openstack-10-rpms
$ sudo yum install -y python-openstackclient
```

Verify the package version for **`python-openstackclient`** is at least version **`3.2.1-1`**

```
$ rpm -q python-openstackclient
python-openstackclient-3.2.1-1.el7ost.noarch
```

To use bash completion feature, where the openstack commands are completed by pressing <tab> keystroke, use the following command in the host where the python-openstackclient has been installed:

```
$ openstack complete | sudo tee /etc/bash_completion.d/osc.bash_completion
> /dev/null
```

Logout and login again and the openstack commands can be autocompleted by pressing the <tab> keystroke as for **`openstack network list`**:

```
$ openstack netw<tab> li<tab>
```

## 3.1. CREATING OPENSTACK USER ACCOUNTS, PROJECTS AND ROLES

Before installing Red Hat OpenShift Container Platform, the Red Hat OpenStack Platform environment requires a project (tenant) that stores the Red Hat OpenStack Platform instances that are to install Red Hat OpenShift Container Platform. This project requires ownership by a user and role of that user to be set to *member*.

The following steps show how to accomplish the above.

As the Red Hat OpenStack Platform overcloud administrator,

1. Create a project (tenant) that is to store the RHOSP instances

   ```
   $ openstack project create <project>
   ```

2. Create a Red Hat OpenStack Platform user that has ownership of the previously created project

   ```
   $ openstack user create --password <password> <username>
   ```

3. Set the role of the user

   ```
   $ openstack role add --user <username> --project <project> member
   ```

Once the above is complete, an OpenStack administrator can create an RC file with all the required information to the user(s) implementing the Red Hat OpenShift Container Platform environment.

An example RC file:

```
$ cat path/to/examplerc
export OS_USERNAME=openshift
export OS_TENANT_NAME=openshift
export OS_PASSWORD=redhat
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://10.19.114.177:5000/v2.0
```

As the user(s) implementing the Red Hat OpenShift Container Platform environment, within the Red Hat OpenStack Platform director node or workstation, ensure to **`source`** the credentials as follows:

```
$ source path/to/examplerc
```

## 3.2. CREATING A RED HAT ENTERPRISE LINUX BASE IMAGE

After the user and project are created, a cloud image is required for all the Red Hat OpenStack Platform instances that are to install Red Hat OpenShift Container Platform. For this particular reference environment the image used is Red Hat Enterprise Linux 7.4.

**NOTE**

Red Hat Enterprise Linux 7.4 includes support for overlay2 file system used in this reference architecture for container storage purposes. If using prior Red Hat Enterprise Linux versions, do not use overlay2 file system.

The Red Hat Enterprise Linux 7.4 cloud image is located at Red Hat Enterprise Linux 7.4 KVM Guest Image Copy the image to the director or workstation node and store the image within **`glance`**. Glance image services provide the ability to discover, register and retrieve virtual machine (VM) images.

**NOTE**

Depending on the *glance* storage backend, it can be required to convert the image format from *qcow2* to a different format. For *Ceph* storage backends, it is required the image to converted to *raw* image format.

The following steps show the process of incorporating the Red Hat Enterprise Linux 7.4 image.

```
$ source /path/to/examplerc
$ mkdir ~/images
$ sudo cp /path/to/downloaded/rhel-server-7.4-x86_64-kvm.qcow2 ~/images
$ cd ~/images
$ qemu-img convert \
    -f qcow2 -O raw \
    rhel-server-7.4-x86_64-kvm.qcow2 ./<img-name>.raw
$ openstack image create <img-name> \
    --disk-format=raw \
    --container-format=bare < <img-name>.raw
```

An Red Hat OpenStack Platform administrator may allow this **`glance`** Red Hat Enterprise Linux image to be readily available to all projects within the Red Hat OpenStack Platform environment. This may be done by **`source`** an RC file with OpenStack administrator privileges and including the **`--public`** option prior to creating the **`raw`** image.

Confirm the creation of the image via **`openstack image list`**. An example output of the reference environment is shown.

```
$ openstack image list
+--------------------------------------+-------+--------+
| ID                                   | Name  | Status |
+--------------------------------------+-------+--------+
| 7ce292d7-0e7f-495d-b2a6-aea6c7455e8c | rhel7 | active |
+--------------------------------------+-------+--------+
```

## 3.3. CREATE AN OPENSTACK FLAVOR

Within OpenStack, flavors define the size of a virtual server by defining the compute, memory, and storage capacity of **`nova`** computing instances. Since the base image within this reference architecture is Red Hat Enterprise Linux 7.4, a **`m1.node`** and **`m1.master`** sized flavor is created with the following specifications as shown in Table 3.1, “Minimum System Requirements for OpenShift”.

**Table 3.1. Minimum System Requirements for OpenShift**

| Node Type | CPU | RAM | Root Disk | Flavor |
|---|---|---|---|---|
| Masters | 2 | 16 GB | 45 GB | **`m1.master`** |
| Nodes | 1 | 8 GB | 20 GB | **`m1.node`** |

As an OpenStack administrator,

```
$ openstack flavor create <flavor_name> \
    --id auto \
    --ram <ram_in_MB> \
    --disk <disk_in_GB> \
    --vcpus <num_vcpus>
```

An example below showing the creation of flavors within this reference environment.

```
$ openstack flavor create m1.master \
    --id auto \
    --ram 16384 \
    --disk 45 \
    --vcpus 2
$ openstack flavor create m1.node \
    --id auto \
    --ram 8192 \
    --disk 20 \
    --vcpus 1
```

**NOTE**

If access to OpenStack administrator privileges to create new flavors is unavailable, use existing flavors within the OpenStack environment that meet the requirements in Table 3.1, “Minimum System Requirements for OpenShift”

## 3.4. CREATING AN OPENSTACK KEYPAIR

Red Hat OpenStack Platform uses **`cloud-init`** to place an **`ssh`** public key on each instance as it is created to allow **`ssh`** access to the instance. Red Hat OpenStack Platform expects the user to hold the private key.

In order to generate a keypair use the following command

```
$ openstack keypair create <keypair-name> > /path/to/<keypair-name>.pem
```

Once the keypair is created, set the permissions to 600 thus only allowing the owner of the file to read and write to that file.

```
$ chmod 600 /path/to/<keypair-name>.pem
```

## 3.5. CREATION OF RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKS VIA OPENSTACK

When deploying Red Hat OpenShift Container Platform on Red Hat OpenStack Platform as described in this reference environment, the requirements are two networks — *public* and *internal* network.

### 3.5.1. Public Network

The *public* network is a network that contains external access and can be reached by the outside world. The *public* network creation can be only done by an OpenStack administrator.

The following commands provide an example of creating an OpenStack provider network for *public* network access.

As an OpenStack administrator,

```
$ source /path/to/examplerc
$ neutron net-create public_network \
    --router:external \
    --provider:network_type flat \
    --provider:physical_network datacentre
$ neutron subnet-create \
    --name public_network \
    --gateway <ip> \
    --allocation-pool start=<float_start>,end=<float_end> \
    public_network <CIDR>
```

**NOTE**

**`<float_start>`** and **`<float_end>`** are the associated floating IP pool provided to the network labeled *public* network. The Classless Inter-Domain Routing (CIDR) uses the format <ip>/<routing_prefix>, i.e. 10.5.2.1/24

### 3.5.2. Internal Network

The *internal* network is connected to the *public* network via a router during the network setup. This allows each Red Hat OpenStack Platform instance attached to the *internal* network the ability to request a floating IP from the *public* network for public access.

The following commands create the *internal* network.

**WARNING**

If there currently is not a DNS service within the organization proceed to Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance* before continuing as this is required. The *internal* network is required for the DNS instance thus it can be created without specifying the *dns-nameserver* setting then add it as **`openstack subnet set --dns-nameserver <dns-ip> <subnet>`** once the DNS server is properly installed.

```
$ source /path/to/examplerc
$ openstack router create <router-name>
$ openstack network create <private-net-name>
$ openstack subnet create --net <private-net-name> \
    --dns-nameserver <dns-ip> \
    --subnet-range <cidr> \
    <private-net-name>
$ openstack subnet list
$ openstack router add subnet <router-name> <private-subnet-uuid>
$ neutron router-gateway-set <router-name> <public-network-uuid>
```

## 3.6. SETTING UP DNS FOR RED HAT OPENSHIFT CONTAINER PLATFORM

The installation process for Red Hat OpenShift Container Platform depends on a reliable name service that contains an address record for each of the target instances. If a DNS is currently not set, please refer to the appendix section on Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance*.

**NOTE**

Using **`/etc/hosts`** is not valid, a proper DNS service must exist.

## 3.7. CREATING RED HAT OPENSTACK PLATFORM SECURITY GROUPS

Red Hat OpenStack Platform networking allows the user to define inbound and outbound traffic filters that can be applied to each instance on a network. This allows the user to limit network traffic to each instance based on the function of the instance services and not depend on host based filtering.

This section describes the ports and services required for each type of host and how to create the security groups in Red Hat OpenStack Platform.

The following table shows the security group association to every instance type:

**Table 3.2. Security Group association**

| Instance type | Security groups associated |
|---|---|
| Bastion | **`<bastion-sg-name>`** |
| Masters | **`<master-sg-name>`** **`<node-sg-name>`** |
| Infra nodes | **`<infra-sg-name>`** **`<node-sg-name>`** |
| App nodes | **`<node-sg-name>`** |

As the <node-sg-name> security group is applied to all the instance types, the common rules are applied to it.

```
$ source /path/to/examplerc
$ for SG in <bastion-sg-name> <master-sg-name> <infra-sg-name> <node-sg-
name>
do
  openstack security group create $SG
done
```

The following tables and commands describe the security group network access controls for the *bastion* host, *master*, *infrastructure* and *app* instances.

### 3.7.1. Bastion Security Group

The *bastion* instance only needs to allow inbound **`ssh`**. This instance exists to give operators a stable base to deploy, monitor and manage the Red Hat OpenShift Container Platform environment.

**Table 3.3. Bastion Security Group TCP ports**

| Port/Protocol | Service | Remote source | Purpose |
|---|---|---|---|

| Port/Protocol | Service | Remote source | Purpose |
|---|---|---|---|
| ICMP | ICMP | Any | Allow ping, traceroute, etc. |
| 22/TCP | SSH | Any | Secure shell login |

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <bastion-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    <bastion-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group rule list <bastion-sg-name>
```

### 3.7.2. Master Security Group

The Red Hat OpenShift Container Platform master security group requires the most complex network access controls. In addition to the ports used by the API and master console, these nodes contain the **`etcd`** servers that form the cluster.

**Table 3.4. Master Host Security Group Ports**

| Port/Protocol | Service | Remote source | Purpose |
|---|---|---|---|
| 2379/TCP | etcd | Masters | Client → Server connections |
| 2380/TCP | etcd | Masters | Server → Server cluster communications |
| 8053/TCP | DNS | Masters and nodes | Internal name services (3.2+) |
| 8053/UDP | DNS | Masters and nodes | Internal name services (3.2+) |
| 8443/TCP | HTTPS | Any | Master WebUI and API |
| 24224/TCP | fluentd | Masters | master → master |

| Port/Protocol | Service | Remote source | Purpose |
|---|---|---|---|
| 24224/UDP | fluentd | Masters | master → master |

**NOTE**

As masters nodes are in fact nodes, the node security group is applied, as well as, the master security group.

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 2379:2380 \
    --src-group <master-sg-name> \
    <master-sg-name>

$ for PORT in 8053 24224;
do
  for PROTO in tcp udp;
  do
    openstack security group rule create \
      --ingress \
      --protocol $PROTO \
      --dst-port $PORT \
      --src-group <node-sg-name> \
      <master-sg-name>;
  done;
done
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 8443 \
    <master-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group rule list <master-sg-name>
```

### 3.7.3. Infrastructure Node Security Group

The infrastructure nodes run the Red Hat OpenShift Container Platform router and the local registry. It must accept inbound connections on the web ports that are forwarded to their destinations.

**Table 3.5. Infrastructure Node Security Group Ports**

| Port/Protocol | Services | Remote source | Purpose |
|---|---|---|---|
| 80/TCP | HTTP | Any | Cleartext application web traffic |
| 443/TCP | HTTPS | Any | Encrypted application web traffic |
| 9200/TCP | ElasticSearch | Any | ElasticSearch API |
| 9300/TCP | ElasticSearch | Any | Internal cluster use |

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ for PORT in 80 443 9200 9300;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port $PORT \
    <infra-sg-name>;
done
```

Verification of the security group is as follows:

```
$ openstack security group rule list <infra-sg-name>
```

### 3.7.4. Node Security Group

All instances run OpenShift node service. All instances should only accept ICMP from any source, **ssh** traffic from the *bastion* host or other nodes, pod to pod communication via SDN traffic and kubelet communication via Kubernetes.

**Table 3.6. Node Security Group Ports**

| Port/Protocol | Services | Remote source | Purpose |
|---|---|---|---|
| ICMP | Ping et al. | Any | Debug connectivity issues |
| 22/TCP | SSH | Bastion | Secure shell login |
| 4789/UDP | SDN | Nodes | Pod to pod communications |
| 10250/TCP | kubernetes | Nodes | Kubelet communications |

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <node-sg-name>
$ for SRC in <bastion-sg-name> <node-sg-name>;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    --src-group $SRC \
    <node-sg-name>;
done
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 10250 \
    --src-group <node-sg-name> \
    <node-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol udp \
    --dst-port 4789 \
    --src-group <node-sg-name> \
    <node-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group rule list <node-sg-name>
```

## 3.8. CREATING RED HAT OPENSTACK PLATFORM CINDER VOLUMES

A number of cinder volumes are required in this reference architecture, including cinder volumes for instances and a cinder volume to store the registry images.

The master instances contain volumes to store **`docker`** images, OpenShift local volumes, and **`etcd`** storage. Each volume has a role in keeping the **`/var`** filesystem partition from corruption and provides the ability for the use of **`cinder`** volume snapshots.

The node instances contain two volumes - a **`docker`** and OpenShift local storage volume. These volumes do not require snapshot capability, but serve the purpose of ensuring that a large image or container does not compromise node performance or abilities.

**Table 3.7. Cinder Volumes**

| Instance type | Volumes | Role | Minimum recommended size |
|---|---|---|---|
| Master | **`<instance_name>-etcd`** | **`etcd`** data | 25 GB |

| Instance type | Volumes | Role | Minimum recommended size |
|---|---|---|---|
| | **`<instance_name>-docker`** | Store docker images | 15 GB |
| | **`<instance_name>-openshift-local`** | Pod local storage | 30 GB |
| Infra and app nodes | **`<instance_name>-docker`** | Store docker images | 15 GB |
| | **`<instance_name>-openshift-local`** | Pod local storage | 30 GB |
| None | **`openshift-registry`** | Store images | 30 GB |

**NOTE**

The masters and nodes volumes are created at boot time, there is not a need to create them prior creating the instance. The **`openshift-registry`** volume creation is explained in the Section 3.8.4, "Registry volume" section.

### 3.8.1. Docker Volume

Containers and images are stored locally in a dedicated cinder volume on each instance able to run containers. The reason it must be a dedicate volume is to ensure there is not disk contention in the **`/var`** partition. The docker volume is customized using the **`docker-storage-setup`** at boot time using **`cloud-init`**.

**NOTE**

The value of the docker volume size should be at least 15 GB.

### 3.8.2. etcd Volume

A **`cinder`** volume is created on the *master* instances for the storage of **`/var/lib/etcd`**. Storing **`etcd`** allows the similar benefit of protecting **`/var`** but more importantly provides the ability to perform snapshots of the volume when performing **`etcd`** maintenance. The **`cinder etcd`** volume is created only on the *master* instances.

**NOTE**

The value of the etcd volume size should be at least 25 GB.

### 3.8.3. OpenShift Local Volume

A **`cinder`** volume is created for the directory of **`/var/lib/origin/openshift.local.volumes`** that is used with the **`perFSGroup`** setting at installation and with the mount option of **`gquota`**. These settings and volumes set a quota to ensure that containers cannot grow to an unreasonable size.

**NOTE**

The value of OpenShift local volume size should be at least 30 GB.

### 3.8.4. Registry volume

The Red Hat OpenShift Container Platform registry requires a cinder volume to ensure that images are saved in the event that the registry needs to migrate to another node. The volume is attached automatically by Red Hat OpenShift Container Platform to the node running the registry, there is no need to attach it manually.

```
$ source /path/to/examplerc
$ openstack volume create --size *<volume-size-in-GB>* openshift-registry
$ openstack volume list
```

**NOTE**

The registry volume size should be at least 30GB.

## 3.9. CREATING RHOSP INSTANCES FOR RHOCP

This reference environment consists of the following instances:

- one *bastion* instance
- three *master* instances
- three *infrastructure* instances
- three *application* instances

**`etcd`** requires that an odd number of cluster members exist. Three masters were chosen to support high availability and **`etcd`** clustering. Three infrastructure instances allow for minimal to zero downtime for applications running in the OpenShift environment. Applications instance can be one to many instances depending on the requirements of the organization.

**NOTE**

*infra* and *app* node instances can easily be added after the initial install.

The creation of each instance within Red Hat OpenStack Platform varies based upon the following:

- Hostname
- Attached **`cinder`** storage volumes
- Assigned security group based upon instance type

Red Hat OpenShift Container Platform uses the hostnames of all the nodes to identify, control and monitor the services on them. It is critical that the instance hostnames, the DNS hostnames and IP addresses are mapped correctly and consistently.

Without any inputs, Red Hat OpenStack Platform uses the **`nova`** instance name as the hostname and the domain as *novalocal*. The *bastion* host's FQDN would result in **`bastion.novalocal`**. This would suffice if Red Hat OpenStack Platform populated a DNS service with these names thus allowing each instance to find the IP addresses by name.

However, using the *novalocal* domain requires creating a zone in the external DNS service named *novalocal*. Since the RHOSP instance names are unique only within a project, this risks name collisions with other projects. To remedy this issue, creation of a subdomain for the *internal* network is implemented under the project domain, i.e. *ocp3.example.com*

**Table 3.8. Subdomain for RHOCP Internal Network**

| Domain Name | Description |
|---|---|
| **`internal.ocp3.example.com`** | All interfaces on the internal only network |

The **`nova`** instance name and the instance hostname are the same. Both are in the **`internal`** subdomain. The floating IPs are assigned to the top level domain **`ocp3.example.com`**.

**Table 3.9. Sample FQDNs**

| Fully Qualified Name | Description |
|---|---|
| **`master-0.internal.ocp3.example.com`** | Name of the *internal* network interface on the **`master-0`** instance |
| **`infra-node-0.internal.ocp3.example.com`** | Name of the *internal* network interface on the **`infra-node-0`** instance |
| **`app-node-0.internal.ocp3.example.com`** | Name of the *internal* network interface on the **`app-node-0`** instance |
| **`openshift.ocp3.example.com`** | Name of the Red Hat OpenShift Container Platform console using the address of the **`haproxy`** instance on the *public* network |

### 3.9.1. Cloud-Init

Red Hat OpenStack Platform provides a way for users to pass in information to be applied when an instance boots. The **`--user-data`** switch to **`openstack server create`** command makes the contents of the provided file available to the instance through **`cloud-init`**. **`cloud-init`** is a set of init scripts for cloud instances. It is available via the **`rhel-7-server-rh-common-rpms`** repository and queries a standard URL for the **`user-data`** file and processes the contents to initialize the OS of the deployed Red Hat OpenStack Platform instance.

The **`user-data`** is placed in files named *<hostname>.yaml* where *<hostname>* is the name of the instance.

An example of the *<hostname>.yaml* file is shown below. This is required for every Red Hat OpenStack Platform instance that is to be used for the Red Hat OpenShift Container Platform deployment.

Create a **`user-data`** directory to store the *<hostname>.yaml* files.

```
$ mkdir /path/to/user-data
```

#### 3.9.1.1. Master Cloud Init

The master instances require docker storage, partitions, and certain mounts be available for a successful deployment using the cinder volumes created in the previous steps.

```
#cloud-config
cloud_config_modules:
- disk_setup
- mounts

hostname: <hostname>
fqdn: <hostname>.<domain>

write_files:
  - path: "/etc/sysconfig/docker-storage-setup"
    permissions: "0644"
    owner: "root"
    content: |
      DEVS='/dev/vdb'
      VG=docker_vol
      DATA_SIZE=95%VG
      STORAGE_DRIVER=overlay2
      CONTAINER_ROOT_LV_NAME=dockerlv
      CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
      CONTAINER_ROOT_LV_SIZE=100%FREE

fs_setup:
- label: emptydir
  filesystem: xfs
  device: /dev/vdc
  partition: auto
- label: etcd_storage
  filesystem: xfs
  device: /dev/vdd
  partition: auto

runcmd:
- mkdir -p /var/lib/origin/openshift.local.volumes
- mkdir -p /var/lib/etcd

mounts:
- [ /dev/vdc, /var/lib/origin/openshift.local.volumes, xfs,
"defaults,gquota" ]
- [ /dev/vdd, /var/lib/etcd, xfs, "defaults" ]
```

#### 3.9.1.2. Node Cloud Init

The node instances, regardless if the instance is an *application* or *infrastructure* node require docker storage, partitions, and certain mounts be available for a successful deployment using the cinder volumes created in the previous steps.

```
#cloud-config
cloud_config_modules:
- disk_setup
- mounts

hostname: <hostname>
fqdn: <hostname>.<domain>

write_files:
  - path: "/etc/sysconfig/docker-storage-setup"
    permissions: "0644"
    owner: "root"
    content: |
      DEVS='/dev/vdb'
      VG=docker_vol
      DATA_SIZE=95%VG
      STORAGE_DRIVER=overlay2
      CONTAINER_ROOT_LV_NAME=dockerlv
      CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
      CONTAINER_ROOT_LV_SIZE=100%FREE

fs_setup:
- label: emptydir
  filesystem: xfs
  device: /dev/vdc
  partition: auto

runcmd:
- mkdir -p /var/lib/origin/openshift.local.volumes

mounts:
- [ /dev/vdc, /var/lib/origin/openshift.local.volumes, xfs,
"defaults,gquota" ]
```

Once the yaml files are created for each Red Hat OpenStack Platform instance, create the instances using the **`openstack`** command.

### 3.9.2. Master Instance Creation

The bash lines below are a loop that allow for all of the master instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *master* instances by filling in the bold items relevant to the current OpenStack values.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in master-{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <master-sg-name>,<node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
local-volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdd,size=<etcd-volume-
size>,shutdown=preserve \
    $node.$domain;
done
```

**NOTE**

Assuming the **`user-data`** yaml files are labeled **`master-<num>.yaml`**

**WARNING**

The volume device order is important as cloud-init creates filesystems based on the volume order.

### 3.9.3. Infrastructure Instance Creation

The bash lines below are a loop that allow for all of the infrastructure instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *infrastructure* instances by filling in the bold items relevant to the current OpenStack values.

**NOTE**

The default tenant limits only allow to create 10 volumes and 10 instances. It maybe required contact your OpenStack administrator to increase the tenant limits to fit the environment.

```
$ domain = <domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in infra-node-{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <infra-sg-name>,<node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
```

```
local-volume-size>,shutdown=preserve \
    $node.$domain;
done
```

**NOTE**

Assuming the **`user-data`** yaml files are labeled **`infra-node-<num>.yaml`**

**WARNING**

The volume device order is important as cloud-init creates filesystems based on the volume order.

### 3.9.4. Application Instance Creation

The bash lines below are a loop that allow for all of the application instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *application* instances by filling in the bold items relevant to the current OpenStack values.

```
$ domain = <domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in app-node-{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
local-volume-size>,shutdown=preserve \
      $node.$domain;
done
```

**NOTE**

Assuming the **`user-data`** yaml files are labeled **`app-node-<num>.yaml`**

**WARNING**

The volume device order is important as cloud-init creates filesystems based on the volume order.

### 3.9.5. Confirming Instance Deployment

The above creates 3 *master*, *infra* and *app* instances.

Verify the creation of the Red Hat OpenStack Platform instances via:

```
$ openstack server list
```

Using the values provided by the **`openstack server list`** command, update the DNS master **`zone.db`** file as shown in Section A.1, “Setting up the DNS Red Hat OpenStack Platform Instance” with the appropriate IP addresses. Do not proceed to the until the DNS resolution is configured.

### 3.9.6. Rename volumes (optional)

*cinder* volumes are created using the instance id making them confusing to identify.

To rename the *cinder* volumes and make them human readable, use the following snippet:

```
$ for node in master-{0..2} app-node-{0..2} infra-node-{0..2};
do
  dockervol=$(openstack volume list -f value -c ID -c "Attached to" | awk
"/$node/ && /vdb/ {print \$1}")
  ocplocalvol=$(openstack volume list -f value -c ID -c "Attached to" |
awk "/$node/ && /vdc/ {print \$1}")
  openstack volume set --name $node-docker $dockervol
  openstack volume set --name $node-ocplocal $ocplocalvol
done
for master in master-{0..2};
do
  etcdvol=$(openstack volume list -f value -c ID -c "Attached to" | awk
"/$master/ && /vdd/ {print \$1}")
  openstack volume set --name $master-etcd $etcdvol
done
```

## 3.10. CREATING AND CONFIGURING THE BASTION INSTANCE

The bastion servers two functions in this deployment of Red Hat OpenShift Container Platform. The first function is a **ssh** jump box allowing administrators to access instances within the private network. The other role of the *bastion* instance is to serve as a utility host for the deployment and management of Red Hat OpenShift Container Platform.

### 3.10.1. Deploying the Bastion Instance

Create the *bastion* instance via:

**NOTE**

If the *m1.small* flavor does not exist by default then create a flavor with 1 vCPU and 2GB of RAM.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-group <bastion-sg-name> \
    bastion.$domain
```

### 3.10.2. Creating and Adding Floating IPs to Instances

Once the instances are created, floating IPs must be created and then can be allocated to the instance. The following shows an example

```
$ source /path/to/examplerc
$ openstack floating ip create <public-network-name>
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-08-24T22:44:03Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 10.20.120.150                       |
| floating_network_id | 084884f9-d9d2-477a-bae7-26dbb4ff1873 |
| headers             |                                      |
| id                  | 2bc06e39-1efb-453e-8642-39f910ac8fd1 |
| port_id             | None                                 |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| revision_number     | 1                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2017-08-24T22:44:03Z                 |
+---------------------+--------------------------------------+
```

Within the above output, the **`floating_ip_address`** field shows that the floating IP **`10.20.120.150`** is created. In order to assign this IP to one of the Red Hat OpenShift Container Platform instances, run the following command:

```
$ source /path/to/examplerc
$ openstack server add floating ip <instance-name> <ip>
```

For example, if instance **`bastion.ocp3.example.com`** is to be assigned IP **`10.20.120.150`** the command would be:

```
$ source /path/to/examplerc
$ openstack server add floating ip bastion.ocp3.example.com 10.20.120.150
```

**NOTE**

Within this reference architecture, the only Red Hat OpenShift Container Platform instances requiring a floating IP are the *bastion* host and the *haproxy* host. In case the load balancer is located in a different Red Hat OpenStack Platform tenant or somewhere else, the masters and infra-nodes need to have a floating IP each as the load balancer needs to reach the masters and infra-nodes from outside.

## 3.11. CREATING AND CONFIGURING AN HAPROXY RED HAT OPENSTACK PLATFORM INSTANCE

If an organization currently does not have a load balancer in place then **`HAProxy`** can be deployed. A load balancer such as **`HAProxy`** provides a single view of the Red Hat OpenShift Container Platform master services for the applications. The master services and the applications use different TCP ports so a single TCP load balancer can handle all of the inbound connections.

The load balanced DNS name that developers use must be in a DNS A record pointing to the **`haproxy`** server before installation. For applications, a wildcard DNS entry must point to the **`haproxy`** host.

The configuration of the load balancer is created after all of the Red Hat OpenShift Container Platform instances have been created and floating IP addresses assigned.

The following steps show how to create the security group and add specific rules to the security group.

```
$ source /path/to/examplerc
$ openstack security group create <haproxy-sg-name>
$ for PORT in 22 80 443 8443;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port $PORT \
    <haproxy-sg-name>;
done
```

**NOTE**

If the m1.small flavor does not exist by default then create a flavor with 1vcpu and 2GB of RAM.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-group <haproxy-sg-name> \
    haproxy.$domain
```

Assign a floating ip

```
$ openstack floating ip create public_network
$ openstack server add floating ip haproxy.$domain <ip>
```

Using the floating IP, log into the Red Hat OpenStack Platform instance that shall become the the **`HAProxy`** server.

```
$ ssh -i /path/to/<keypair-name>.pem cloud-user@<IP>
```

### 3.11.1. Setting up the HAProxy Instance

The following steps configure a HAProxy server once logged into the instance.

Subscribe to the following repositories using **`subscription-manager`**

```
$ sudo subscription-manager register
$ sudo subscription-manager attach --pool <pool_id>
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms \
    --enable rhel-7-server-rh-common-rpms
```

Install and enable the following packages and services:

```
$ sudo yum -y install haproxy firewalld
$ sudo systemctl enable firewalld
$ sudo systemctl start firewalld
$ sudo systemctl enable haproxy
$ sudo systemctl start haproxy
```

Enable and permanently set the required services within firewalld

```
$ sudo firewall-cmd --zone public --add-service http
$ sudo firewall-cmd --zone public --add-service https
$ sudo firewall-cmd --zone public --add-port=8443/tcp
$ sudo firewall-cmd --zone public --add-service http --permanent
$ sudo firewall-cmd --zone public --add-service https --permanent
$ sudo firewall-cmd --zone public --add-port=8443/tcp --permanent
```

Modify the */etc/haproxy/haproxy.cfg* file with the appropriate Red Hat OpenShift Container Platform instance IPs

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon
    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

defaults
    log                     global
    option                  httplog
    option                  dontlognull
    option                  http-server-close
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
listen stats :9000
    stats enable
    stats realm Haproxy\ Statistics
    stats uri /haproxy_stats
    stats auth admin:password
    stats refresh 30
    mode http

frontend  main *:80
    default_backend router80

backend router80
    balance source
    mode tcp
    # INFRA_80
    server infra-node-0.internal.ocp3.example.com <IP>:80 check
    server infra-node-1.internal.ocp3.example.com <IP>:80 check
    server infra-node-2.internal.ocp3.example.com <IP>:80 check

frontend  main *:443
    default_backend router443

backend router443
    balance source
    mode tcp
    # INFRA_443
    server infra-node-0.internal.ocp3.example.com <IP>:443 check
    server infra-node-1.internal.ocp3.example.com <IP>:443 check
    server infra-node-2.internal.ocp3.example.com <IP>:443 check

frontend  main *:8443
    default_backend mgmt8443

backend mgmt8443
    balance source
    mode tcp
    # MASTERS 8443
    server master-0.internal.ocp3.example.com <IP>:8443 check
    server master-1.internal.ocp3.example.com <IP>:8443 check
    server master-2.internal.ocp3.example.com <IP>:8443 check
```

Restart the HAProxy service

```
$ sudo systemctl restart haproxy.service
```

To check connectivity to the instance services and DNS resolution:

```
$ for i in 0 1 2
do
  curl -v http://infra-node-$i.internal.ocp3.example.com
  curl -v -k https://infra-node-$i.internal.ocp3.example.com
  curl -v -k https://master-$i.internal.ocp3.example.com:8443
done
$ curl -v http://localhost
$ curl -v -k http://localhost:443
$ curl -v -k http://localhost:8443
```

# CHAPTER 4. DEPLOYMENT OF RED HAT OPENSHIFT CONTAINER PLATFORM 3.6

## 4.1. PREPARING THE BASTION INSTANCE

The following subsections register and install the appropriate prerequisites for the *bastion* instance to serve as a utility host.

### 4.1.1. Provide the SSH Private Key

The **ssh** private key is required to allow the *bastion* instance to connect to the other instances used for OpenShift. The following command copies the existing keypair from Section 3.4, “Creating an OpenStack Keypair” to the *bastion* instance and sets the appropriate permissions.

```
$ export KEY="/path/to/<keypair-name>.pem"
$ scp -i $KEY $KEY \
    cloud-user@bastion.internal.ocp3.example.com:~/.ssh/<keypair-name>.pem
$ ssh -i $KEY bastion.internal.ocp3.example.com \
    "chmod 0600 ~/.ssh/<keypair-name>.pem"
```

### 4.1.2. Registering the Bastion

Log into the *bastion* instance and perform the following steps.

Register and subscribe to the following repositories using **subscription-manager**.

```
$ sudo subscription-manager register
$ sudo subscription-manager attach --pool <pool_id>
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms \
    --enable=rhel-7-server-ose-3.6-rpms
```

Install the **ansible**, **git**, and **atomic-openshift-utils** packages that enable the installation and deployment of OpenShift.

```
$ sudo yum install -y ansible atomic-openshift-utils git
```

### 4.1.3. Configuring Ansible

**ansible** is installed on the *bastion* instance to perform the registration, installation of packages, and the deployment of the Red Hat OpenShift Container Platform environment on the *master* and *node* instances.

Before running playbooks, it is important to modify **ansible.cfg** to reflect the deployed environment:

```
$ sudo vi /etc/ansible/ansible.cfg
```

The code block below can overwrite the default values in the file. Ensure to populate **<keypair-name>** with the keypair that was copied to the *bastion* instance.

```
[defaults]
forks = 20
host_key_checking = False
remote_user = cloud-user
private_key_file = /home/cloud-user/.ssh/<keypair-name>.pem
roles_path = roles/
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
nocows = 1
callback_whitelist = profile_tasks

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o
UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=false
control_path = %(directory)s/%%h-%%r
pipelining = True
timeout = 10

[persistent_connection]
connect_timeout = 30
connect_retries = 30
connect_interval = 1
```

### 4.1.4. OpenShift Authentication

Red Hat OpenShift Container Platform provides the ability to use many different authentication platforms. For this reference architecture, LDAP is the preferred authentication mechanism. A listing of other authentication options are available at Configuring Authentication and User Agent.

When configuring LDAP as the authentication provider the following parameters can be added to the ansible inventory. An example is shown below.

```
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN':
'uid=admin,cn=users,cn=accounts,dc=ocp3,dc=openshift,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.ocp3.example.com/cn=users,cn=accounts,dc=ocp3,dc=openshift,dc
=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=ocp3,dc=openshift,dc=com)'}]
```

**NOTE**

If using LDAPS, all the masters must have the relevant *ca.crt* file for LDAP in place prior to the installation, otherwise the installation fails.

For more information about the required LDAP parameters, see LDAP Authentication

### 4.1.5. Preparing the Inventory

The inventory file contains both variables and instances used for the configuration and deployment of Red Hat OpenShift Container Platform. In the example below, some values are **bold** and must reflect the deployed environment from the previous chapter.

To gather the **`openshift_hosted_registry_storage_openstack_volumeID`** value use:

```
$ openstack volume show openshift-registry -f value -c id
```

The **`openshift_cloudprovider_openstack_*`** values are required for Red Hat OpenShift Container Platform to be able to create OpenStack resources such as cinder volumes for persistent volumes. It is recommended to create a dedicated OpenStack user within the tenant instead of using a personal account.

```
$ sudo vi /etc/ansible/hosts
```

```
[OSEv3:children]
masters
etcd
nodes

# Set variables common for all OSEv3 hosts
[OSEv3:vars]
ansible_ssh_user=cloud-user
deployment_type=openshift-enterprise
debug_level=2
ansible_become=true
console_port=8443
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level | default(debug_level,
true) }}"
openshift_master_debug_level="{{ master_debug_level | default(debug_level,
true) }}"

openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN':
'uid=admin,cn=users,cn=accounts,dc=ocp3,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.ocp3.example.com/cn=users,cn=accounts,dc=ocp3,dc=example,dc=c
om?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=ocp3,dc=openshift,dc=com)'}]

openshift_master_access_token_max_seconds=2419201
openshift_registry_selector="role=infra"
openshift_router_selector="role=infra"
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi
openshift_cloudprovider_kind=openstack
```

```
openshift_cloudprovider_openstack_auth_url=http://10.19.114.177:5000/v2.0
openshift_cloudprovider_openstack_username=openshift
openshift_cloudprovider_openstack_password=redhat
openshift_cloudprovider_openstack_tenant_name=openshift-tenant
openshift_master_cluster_hostname=openshift.ocp3.example.com
openshift_master_cluster_public_hostname=openshift.ocp3.example.com
openshift_master_default_subdomain=apps.ocp3.example.com

osm_default_node_selector="role=app"
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
osm_use_cockpit=true

containerized=false

# registry
openshift_hosted_registry_storage_kind=openstack
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
openshift_hosted_registry_storage_openstack_filesystem=ext4

openshift_hosted_registry_storage_openstack_volumeID=3b5c9517-abd1-4364-
80b9-a1469cae8fd8
openshift_hosted_registry_storage_volume_size=15Gi

# host group for masters
[masters]
master-0.internal.ocp3.example.com master-1.internal.ocp3.example.com
master-2.internal.ocp3.example.com

# host group for etcd
[etcd]
master-0.internal.ocp3.example.com master-1.internal.ocp3.example.com
master-2.internal.ocp3.example.com

# host group for nodes, includes region info
[nodes]
master-0.internal.ocp3.example.com openshift_node_labels="{'role':
'master'}" openshift_hostname=master-0.internal.ocp3.example.com
openshift_schedulable=false master-1.internal.ocp3.example.com
openshift_node_labels="{'role': 'master'}" openshift_hostname=master-
1.internal.ocp3.example.com openshift_schedulable=false master-
2.internal.ocp3.example.com openshift_node_labels="{'role': 'master'}"
openshift_hostname=master-2.internal.ocp3.example.com
openshift_schedulable=false app-node-0.internal.ocp3.example.com
openshift_node_labels="{'role': 'app'}" openshift_hostname=app-node-
0.internal.ocp3.example.com app-node-1.internal.ocp3.example.com
openshift_node_labels="{'role': 'app'}" openshift_hostname=app-node-
1.internal.ocp3.example.com app-node-2.internal.ocp3.example.com
openshift_node_labels="{'role': 'app'}" openshift_hostname=app-node-
2.internal.ocp3.example.com infra-node-0.internal.ocp3.example.com
openshift_node_labels="{'role': 'infra'}" openshift_hostname=infra-node-
0.internal.ocp3.example.com infra-node-1.internal.ocp3.example.com
openshift_node_labels="{'role': 'infra'}" openshift_hostname=infra-node-
1.internal.ocp3.example.com infra-node-2.internal.ocp3.example.com
openshift_node_labels="{'role': 'infra'}" openshift_hostname=infra-node-
2.internal.ocp3.example.com
```

### 4.1.6. Check instances

It can be useful to check for potential issues or misconfigurations in the instances before continuing the installation process. Connect to every instance using the bastion host and verify the disks are properly created and mounted, the cloud-init process finished successfully and check for potential errors in the log files to ensure everything is ready for the Red Hat OpenShift Container Platform installation:

```
$ ssh -i $KEY bastion.internal.ocp3.example.com
$ ssh <instance>
$ df -h
$ mount
$ sudo journalctl
$ sudo less /var/log/cloud-init.log
$ free -m
$ sudo yum repolist
```

#### 4.1.6.1. Instance setup

Using the inventory from above register the instances via Red Hat Subscription Manager. This is accomplished via using credentials or an activation key.

Via credentials the **`ansible`** command is as follows:

**NOTE**

This reference environment requires internet access to the Red Hat OpenShift Container Platform instances for package updates via https://access.redhat.com However, Red Hat Satellite may be used for infrastructure management if providing internet access to the Red Hat OpenShift Container Platform instances is not available.

```
$ ansible all -b -i /etc/ansible/hosts -m command -a "subscription-manager
register --username <user> --password '<password>'"
```

Via activation key, the **`ansible`** command is as follows:

```
$ ansible all -b -i /etc/ansible/hosts -m command -a "subscription-manager
register --org=<org_id> --activationkey=<keyname>"
```

where the following options:

- -b - referes to privileged escalation
- -i - location of inventory file
- -m - module to use
- -a - module argument

Once all the instances have been successfully registered, enable all the required RHOCP repositories on all the instances via:

The repositories required for Red Hat OpenShift Container Platform instances are:

- rhel-7-server-rpms

- rhel-7-server-extras-rpms
- rhel-7-server-ose-3.6-rpms
- rhel-7-fast-datapath-rpms

```
$ ansible -i /etc/ansible/hosts all -b \
    -m command -a 'subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.6-rpms" \
    --enable="rhel-7-fast-datapath-rpms"'
```

After installing the prerrequisites, update all the Red Hat Enterprise Linux packages to the latest version in all the instances as:

```
$ ansible -i /etc/ansible/hosts all -b -m yum -a "name=* state=latest"
```

A reboot is required to implement the latest kernel on all nodes and can be accomplished via command:

```
$ ansible -i /etc/ansible/hosts all -b -a "reboot"
```

Once the reboot has been completed install the following packages to be used by OpenShift:

```
$ ansible -i /etc/ansible/hosts all -b \
    -m command -a 'yum -y install iptables NetworkManager iptables-
services iptables docker'
```

With the packages installed the next step is to start the NetworkManager, docker, and iptables services.

```
$ ansible -i /etc/ansible/hosts all -b \
    -m service -a 'name=iptables state=started enabled=true'
$ ansible -i /etc/ansible/hosts all -b \
    -m service -a 'name=docker state=started enabled=true'
$ ansible -i /etc/ansible/hosts all -b \
    -m service -a 'name=NetworkManager state=started enabled=true'
```

## 4.2. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM

With the prerequisites met, the focus shifts to the installation and configuration Red Hat OpenShift Container Platform. The installation and configuration is done via a series of **`Ansible`** playbooks and roles provided by the **`atomic-openshift`** packages.

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/config.yml
```

The playbooks runs through the complete process of installling Red Hat OpenShift Container Platform and reports a playbook recap showing the number of changes and errors (if any).

```
PLAY RECAP
************************************************************************
************************************************************************
***************************************************
```

```
app-node-0.internal.ocp3.example.com : ok=233  changed=40   unreachable=0
failed=0
app-node-1.internal.ocp3.example.com : ok=233  changed=40   unreachable=0
failed=0
app-node-2.internal.ocp3.example.com : ok=233  changed=40   unreachable=0
failed=0
infra-node-0.internal.ocp3.example.com : ok=233  changed=40
unreachable=0    failed=0
infra-node-1.internal.ocp3.example.com : ok=233  changed=40
unreachable=0    failed=0
infra-node-2.internal.ocp3.example.com : ok=233  changed=40
unreachable=0    failed=0
localhost                  : ok=12   changed=0    unreachable=0
failed=0
master-0.internal.ocp3.example.com : ok=674  changed=161  unreachable=0
failed=0
master-1.internal.ocp3.example.com : ok=442  changed=103  unreachable=0
failed=0
master-2.internal.ocp3.example.com : ok=442  changed=103  unreachable=0
failed=0

Tuesday 29 August 2017  10:34:49 -0400 (0:00:01.002)       0:29:54.775
********
===========================================================================
=====
openshift_hosted : Ensure OpenShift router correctly rolls out (best-
effort today) -- 92.44s
openshift_hosted : Ensure OpenShift registry correctly rolls out (best-
effort today) -- 61.93s
openshift_health_check -----------------------------------------------------
- 53.92s
openshift_common : Install the base package for versioning -------------
42.15s
openshift_common : Install the base package for versioning -------------
36.36s
openshift_hosted : Sanity-check that the OpenShift registry rolled out
correctly -- 31.43s
cockpit : Install cockpit-ws --------------------------------------------
27.65s
openshift_version : Get available atomic-openshift version -------------
25.27s
etcd_server_certificates : Install etcd --------------------------------
15.53s
openshift_master : Wait for master controller service to start on first
master -- 15.21s
openshift_master : pause -----------------------------------------------
- 15.20s
openshift_node : Configure Node settings -------------------------------
13.56s
openshift_excluder : Install openshift excluder ------------------------
13.54s
openshift_node : Install sdn-ovs package -------------------------------
13.45s
openshift_master : Create master config --------------------------------
11.92s
openshift_master : Create the scheduler config -------------------------
```

```
10.92s
openshift_master : Create the policy file if it does not already exist --
10.65s
openshift_node : Install Node service file -----------------------------
10.43s
openshift_node : Install Node package ----------------------------------
10.39s
openshift_node : Start and enable node ---------------------------------
- 8.96s
```

## 4.3. POST INSTALLATION STEPS

The following subsections discuss the post installation steps required once the Red Hat OpenShift Container Platform environment has been successfully deployed.

### 4.3.1. Configure ~/.ssh/config to use Bastion as Jumphost (Optional)

To easily connect to the Red Hat OpenShift Container Platform environment, the following snippet can be added to the **`~/.ssh/config`** file in the host used to administer the Red Hat OpenShift Container Platform platform:

```
Host osbastion
    HostName        <bastion-ip>
    User            cloud-user
    IdentityFile    /path/to/<keypair-name>.pem

Host osmaster-? osapp-node-? osinfra-node-?
    ProxyCommand    ssh osbastion -W $(sed -e "s/os//" <<<
"%h").internal.ocp3.example.com:%p
    IdentityFile    /path/to/<keypair-name>.pem
    User            cloud-user
```

As an example, to access the second master, this can be used:

```
$ ssh osmaster-0
```

### 4.3.2. Red Hat OpenShift Container Platform Registry Console Selector

The Red Hat OpenShift Container Platform Registry Console deployment is deployed on any Red Hat OpenShift Container Platform node by default, so the container may end up running on any of the application nodes.

From the first *master* instance(master-0.internal.ocp3.example.com), ensure the Red Hat OpenShift Container Platform Registry Console pod runs on the *infra* nodes by modifying the *nodeSelector* as follows:

```
$ oc patch dc registry-console \
    -n default \
    -p '{"spec":{"template":{"spec":{"nodeSelector":{"role":"infra"}}}}}'
```

**NOTE**

There is a bugzilla ID: 1425022 being investigated by Red Hat at the time of writing this paper to fix this issue.

### 4.3.3. OpenShift Storage Class

The **`StorageClass`** resource object describes and classifies storage that can be requested and provide a means for passing parameters for dynamically provisioned storage on demand. A **`StorageClass`** object can serve as a management mechanism for controlling various levels of access to storage that are defined and created by either a cluster or storage administrator.

With regards to Red Hat OpenStack Platform, the storage type that allows for dynamic provisioning is OpenStack **`cinder`** using the Provisioner Plug-in Name labeled **`kubernetes.io/cinder`**

The following is an example of **`storage-class.yaml`** that is required for dynamic provisioning using OpenStack **`cinder`**. **`ssh`** into the first master instance (*master-0.internal.ocp3.example.com*) and create the storage class file.

```
$ vi storage-class.yaml
```

#### Storage-Class YAML file without `cinder` Volume Type (default)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/cinder
parameters:
  availability: nova
```

Use the **`oc`** client to create the **`StorageClass`**.

```
$ oc create -f storage-class.yaml
```

In the **`storage-class.yaml`** file example, the cluster or storage administrator requires to input a name for the **`StorageClass`**. One parameter option not shown in the above example is **`type`**. Type refers to the volume type created in **`cinder`**. By default, the value for cinder is empty, thus it is not included in the above example.

However, if the **`cinder`** volumes created by Red Hat OpenStack Platform contain a volume type, a **`storage-class.yaml`** file with the additional **`type`** parameter is required as shown below:

#### Storage-Class YAML file with `cinder` Volume Type (custom)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/cinder
```

```
parameters:
  type: fast
  availability: nova
```

### 4.3.4. Adding OpenShift Logging (Optional)

Red Hat OpenShift Container Platform allows the option to deploy aggregate logging for containers running in the OpenShift environment. OpenShift uses the EFK stack (**`Elasticsearch`**, **`Fluentd`**, **`Kibana`**) to collect logs from applications and present them to OpenShift users. Cluster administrators can view all logs, but application developers can only view logs for projects they have permission to view. The EFK stack consists of the following components:

- **`Elasticsearch`** - object store where logs are stored and provides the ability to search logs
- **`Fluentd`** - gathers and sends logs to Elasticsearch
- **`Kibana`** - a web interface to allow for easy interaction with Elasticsearch
- **`Curator`** - schedule Elasticsearch maintenance operations automatically

Below is an example of some of the best practices when deploying OpenShift logging.

**`Elasticsearch`**, **`Kibana`**, and **`Curator`** are deployed on nodes with the label of "role=infra". Specifying the node label ensures that the **`Elasticsearch`** and **`Kibana`** components are not competing with applications for resources. The **`Elasticsearch`** cluster size (**`openshift_logging_es_cluster_size`**) of three is the minimum number of nodes to ensure data durability and redundancy.

**NOTE**

**`Fluentd`** runs on all OpenShift nodes regardless of the node label.

The logging project created during installation must be modified to bypass the default node selector. If this is not done then the **`Elasticsearch`**, **`Kibana`**, and **`Curator`** components cannot be started. There are two options in making the appropriate edit.

Option 1:

**`ssh`** into the first master instance (*master-0.internal.ocp3.example.com*) or the *bastion* host and modify the logging project.

Add the following line **`openshift.io/node-selector: ""`** under annotations but do not remove any lines.

```
$ oc edit project logging
... omitted ...
  annotations:
    openshift.io/node-selector: ""
...ommitted...
project "logging" edited
```

Option 2:

Using **`oc patch`** command

```
oc patch namespace logging \
    -p "{\"metadata\":{\"annotations\":{\"openshift.io/node-
selector\":\"\"}}}"
```

Log out of the *master-0* instance and on the *bastion* instance add the following lines to the **`/etc/ansible/hosts`** file below the registry and above the [masters] entry.

```
openshift_hosted_logging_deploy=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=10Gi
openshift_logging_es_cluster_size=3
openshift_logging_es_nodeselector={"role":"infra"}
openshift_logging_kibana_nodeselector={"role":"infra"}
openshift_logging_curator_nodeselector={"role":"infra"}
```

**NOTE**

A **`StorageClass`** must be defined when requesting dynamic storage.

Run the deployment playbooks to deploy logging using the parameters defined in the inventory file.

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-cluster/openshift-logging.yml
```

An example of a successful playbook run is shown below.

```
PLAY RECAP
********************************************************************
localhost                  : ok=11   changed=0    unreachable=0
failed=0
master-0.internal.ocp3.example.com : ok=237  changed=94   unreachable=0
failed=0
master-1.internal.ocp3.example.com : ok=16   changed=4    unreachable=0
failed=0
master-2.internal.ocp3.example.com : ok=16   changed=4    unreachable=0
failed=0
Wednesday 30 August 2017  12:34:47 -0400 (0:00:00.415)       0:03:58.679
******
=========================================================================
=====
openshift_facts : Ensure various deps are installed --------------------
46.27s
openshift_facts : Ensure various deps are installed --------------------
38.13s
openshift_logging : Run JKS generation script --------------------------
8.98s
openshift_logging : restart master api ---------------------------------
- 3.98s
openshift_logging : Gather OpenShift Logging Facts ---------------------
3.69s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.60s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.13s
```

```
openshift_logging : restart master api -------------------------------
- 2.90s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.74s
openshift_logging : include_role --------------------------------------
- 1.60s
openshift_logging_elasticsearch : Set logging-elasticsearch-view-role role
--- 1.51s
openshift_logging_kibana : Set logging-kibana service ------------------
1.46s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.45s
openshift_logging_elasticsearch : Set logging-es service ----------------
1.43s
openshift_logging_elasticsearch : Set logging-es service ----------------
1.41s
openshift_logging_elasticsearch : Set logging-elasticsearch-view-role role
--- 1.40s
openshift_logging_elasticsearch : Create rolebinding-reader role --------
1.40s
openshift_logging_elasticsearch : Set logging-es service ----------------
1.39s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.37s
openshift_logging_elasticsearch : Create rolebinding-reader role --------
1.36s
```

Once the playbook finishes **ssh** into the first master instance ( *master-0.internal.ocp3.example.com*) or the *bastion* host, and view the pods in the logging project.

```
$ oc get pods -n logging
NAME                                       READY     STATUS    RESTARTS
AGE
logging-curator-1-tzrsx                    1/1       Running   2
4m
logging-es-data-master-9uq6xi6z-1-dw6vq    1/1       Running   0
5m
logging-es-data-master-mcanh3m7-1-deploy   1/1       Running   0
5m
logging-es-data-master-mcanh3m7-1-vfkt2    1/1       Running   0
4m
logging-es-data-master-qbwcw4j6-1-227gj    1/1       Running   0
4m
logging-es-data-master-qbwcw4j6-1-deploy   1/1       Running   0
4m
logging-fluentd-49hfs                      1/1       Running   0
4m
logging-fluentd-4jwd0                      1/1       Running   0
4m
logging-fluentd-8wtph                      1/1       Running   0
4m
logging-fluentd-bnld6                      1/1       Running   0
4m
logging-fluentd-dp89n                      1/1       Running   0
4m
logging-fluentd-hsgl8                      1/1       Running   0
```

```
4m
logging-fluentd-htgx0                      1/1       Running   0
4m
logging-fluentd-s9jp0                      1/1       Running   0
4m
logging-kibana-1-76wxp                     2/2       Running   0
4m
```

**NOTE**

The curator pod restarts are normal as it tries to connect to the elasticsearch cluster before it is created.

### 4.3.5. Adding OpenShift Metrics (Optional)

Red Hat OpenShift Container Platform has the ability to gather metrics from the pods running by querying the kubelet and storing the values in **`Heapster`**. OpenShift Cluster Metrics provide the ability to view CPU, memory, and network-based metrics and display the values in the user interface. These metrics can allow for the horizontal autoscaling of pods based on parameters provided by an OpenShift user. It is important to understand capacity planning when deploying metrics into an OpenShift environment.

As best practices when metrics are deployed, persistent storage should be used to allow for metrics to be preserved. Node selectors should be used to specify where the metrics components should run. In the reference architecture environment, the components are deployed on nodes with the label of "role=infra".

Add the following lines to the **`/etc/ansible/hosts`** file below the registry and above the [masters] entry.

```
openshift_hosted_metrics_deploy=true
openshift_hosted_metrics_storage_kind=dynamic
openshift_hosted_metrics_storage_volume_size=10Gi
openshift_metrics_hawkular_nodeselector={"role":"infra"}
openshift_metrics_cassandra_nodeselector={"role":"infra"}
openshift_metrics_heapster_nodeselector={"role":"infra"}
```

**NOTE**

A **`StorageClass`** must be defined when requesting dynamic storage.

Run the deployment playbooks to deploy metrics using the parameters defined in the inventory file.

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-cluster/openshift-metrics.yml
```

An example of a successful playbook run is shown below.

```
PLAY RECAP
*********************************************************************
localhost                  : ok=11   changed=0    unreachable=0
failed=0
master-0.internal.ocp3.example.com : ok=177  changed=47   unreachable=0
failed=0
```

```
master-1.internal.ocp3.example.com : ok=16   changed=3    unreachable=0
failed=0
master-2.internal.ocp3.example.com : ok=16   changed=3    unreachable=0
failed=0

Wednesday 30 August 2017  12:49:12 -0400 (0:00:00.296)       0:01:54.693
******
===========================================================================
=====
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.87s
openshift_metrics : restart master api ---------------------------------
- 3.33s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.19s
openshift_facts : Ensure various deps are installed --------------------
3.09s
openshift_facts : Ensure various deps are installed --------------------
2.74s
openshift_metrics : slurp ----------------------------------------------
- 2.56s
openshift_metrics : Set serviceaccounts for hawkular metrics/cassandra ---
2.38s
openshift_metrics : restart master api ---------------------------------
- 2.25s
openshift_metrics : Stop Heapster --------------------------------------
- 1.89s
openshift_metrics : Stop Hawkular Metrics ------------------------------
1.60s
openshift_metrics : Start Hawkular Metrics -----------------------------
1.57s
openshift_metrics : Start Hawkular Cassandra ---------------------------
1.56s
openshift_metrics : command --------------------------------------------
- 1.53s
openshift_metrics : Start Heapster -------------------------------------
- 1.51s
openshift_metrics : read files for the hawkular-metrics secret ----------
1.46s
openshift_metrics : Generate services for cassandra --------------------
1.24s
openshift_metrics : Generating serviceaccounts for hawkular
metrics/cassandra --- 1.19s
openshift_metrics : Set hawkular cluster roles -------------------------
1.14s
openshift_metrics : generate hawkular-cassandra keys --------------------
1.02s
Gathering Facts -------------------------------------------------------
-- 0.83s
```

Once the playbook finishes **`ssh`** into the first master instance ( *master-0.internal.ocp3.example.com*) or the *bastion* and view the pods in the **`openshift-infra`** project.

```
$ oc get pods -n openshift-infra
NAME                         READY     STATUS    RESTARTS   AGE
hawkular-cassandra-1-4q46f   1/1       Running   0          3m
```

```
hawkular-metrics-0w46z       1/1       Running   0          3m
heapster-gnhsh               1/1       Running   0          3m
```

# CHAPTER 5. OPERATIONAL MANAGEMENT

With the successful deployment of OpenShift, the following section demonstrates how to confirm proper functionality of the Red Hat OpenShift Container Platform.

## 5.1. VALIDATE THE DEPLOYMENT

Once the installation is completed a simple way to test the deployed environment is by deploying an application, adding a route, and verifying the application is available. This tests that the router, registry, and scheduling work for the newly deployed OpenShift environment.

**Environment Validation**

- Create a project in OpenShift called validate
- Create an OpenShift application
- Add a route for the application
- Validate the URL returns a status code of 200 or healthy
- Delete the validation project

**NOTE**

Ensure the URLs and values below match the values in the inventory file.

```
$ oc new-project validate
$ oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-
ex.git
--> Found Docker image e42d0dc (7 weeks old) from Docker Hub for
"centos/ruby-22-centos7"

    Ruby 2.2
    --------
    Ruby 2.2 available as container is a base platform for building and
running various Ruby 2.2 applications and frameworks. Ruby is the
interpreted scripting language for quick and easy object-oriented
programming. It has many features to process text files and to do system
management tasks (as in Perl). It is simple, straight-forward, and
extensible.

    Tags: builder, ruby, ruby22

    * An image stream will be created as "ruby-22-centos7:latest" that
will track the source image
    * A source build using source code from
https://github.com/openshift/ruby-ex.git will be created
      * The resulting image will be pushed to image stream "ruby-
ex:latest"
      * Every time "ruby-22-centos7:latest" changes a new build will be
triggered
    * This image will be deployed in deployment config "ruby-ex"
    * Port 8080/tcp will be load balanced by service "ruby-ex"
      * Other containers can access this service through the hostname
```

```
"ruby-ex"

--> Creating resources ...
    imagestream "ruby-22-centos7" created
    imagestream "ruby-ex" created
    buildconfig "ruby-ex" created
    deploymentconfig "ruby-ex" created
    service "ruby-ex" created
--> Success
    Build scheduled, use 'oc logs -f bc/ruby-ex' to track its progress.
    Run 'oc status' to view your app.

$ oc get svc
NAME      CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
ruby-ex   172.30.152.72   <none>        8080/TCP   45s
$ oc expose svc/ruby-ex
route "ruby-ex" exposed
```

Check to see if the application has been deployed.

```
$ oc get pod
NAME                READY     STATUS      RESTARTS   AGE
ruby-ex-1-build     0/1       Completed   0          2m
ruby-ex-1-slk9l     1/1       Running     0          9s


$ oc get routes
NAME      HOST/PORT                       PATH       SERVICES   PORT
TERMINATION   WILDCARD
ruby-ex   ruby-ex-test.apps.example.com              ruby-ex    8080-tcp
None
```

Using either a browser or curl access the route to ensure that the content appears. If the page returns the content then this means that the router and registry are operating as expected.

Once this test has been completed then delete the **`validate`** project.

```
$ oc delete project validate
```

## 5.2. CHECKING THE HEALTH OF ETCD

This section focuses on the **`etcd`** cluster. It describes the different commands to ensure the cluster is healthy.

**`ssh`** into the first master node (*master-0.internal.ocp3.example.com*). Using the output of the command **`hostname`** issue the **`etcdctl`** command to confirm that the cluster is healthy.

```
$ ssh cloud-user@master-0.internal.ocp3.example.com
```

Using the **`ip`** command identify the **`IP`** address of the **`master-0`** instance. This value is used with **`etcdctl`**. Replace the value of **`-C`** **`https://172.16.77.4:2379`** with the output of the **`ip`** command.

```
$ ip a | grep eth0
$ sudo etcdctl --ca-file /etc/etcd/ca.crt -C https://172.16.77.4:2379 --
```

```
cert-file=/etc/origin/master/master.etcd-client.crt --key-
file=/etc/origin/master/master.etcd-client.key cluster-health
2017-08-29 14:53:45.986526 I | warning: ignoring ServerName for user-
provided CA for backwards compatibility is deprecated
2017-08-29 14:53:45.987228 I | warning: ignoring ServerName for user-
provided CA for backwards compatibility is deprecated
member 2d0c4c13c58032a is healthy: got healthy result from
https://172.16.77.11:2379
member 4e2e806d50a811ec is healthy: got healthy result from
https://172.16.77.3:2379
member 6c0c26364d230569 is healthy: got healthy result from
https://172.16.77.4:2379
```

**NOTE**

In this configuration the **etcd** services are distributed among the OpenShift master nodes.

## 5.3. DEFAULT NODE SELECTOR

The default node selector within this reference environment is set to "role=app" in **/etc/origin/master/master-config.yaml** on all of the master nodes. This configuration parameter is set by the OpenShift installation playbooks on all masters.

**ssh** into the first master node (*ose-master01.ocp3.example.com*) to verify the **defaultNodeSelector** is defined.

```
$ sudo vi /etc/origin/master/master-config.yaml
...omitted...
projectConfig:
  defaultNodeSelector: "role=app"
  projectRequestMessage: ""
  projectRequestTemplate: ""
...omitted...
```

If making any changes to the master configuration then the master API service must be restarted otherwise the changes are not applied. Any changes and the subsequent restart must be done on all masters.

```
$ sudo systemctl restart atomic-openshift-master-api
```

## 5.4. MANAGEMENT OF MAXIMUM POD SIZE

Quotas are set on ephemeral volumes within pods to prohibit a pod from becoming too large and impacting the node. There are three places where sizing restrictions should be set. When persistent volume claims are not set, a pod has the ability to grow as large as the underlying filesystem allows. The required modifications are set using a combination of user-data and Ansible.

**Openshift Volume Quota**

At launch time, user-data creates a xfs partition on the **/dev/vdc** block device, adds an entry in fstab, and mounts the volume with the option of gquota. If gquota is not set, the OpenShift node is unable to start with the **perFSGroup** parameter defined below. This disk and configuration is done on the infrastructure and application instances.

**ssh** into the first infrastructure node (*infra-node-0.internal.ocp3.example.com*) to verify the entry exists within fstab.

```
$ sudo grep "local.volumes" /etc/fstab
/dev/vdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0
```

**Docker Storage Setup**

The docker-storage-setup file is created at launch time by cloud-init. This file configures the Docker service to use **/dev/vdb** and creates the volume group of **docker-vol**. Docker storage setup is performed on all master, infrastructure, and application nodes.

**ssh** into the first infrastructure node (*infra-node-0.internal.ocp3.example.com*) to verify **/etc/sysconfig/docker-storage-setup** matches the information below.

```
$ sudo cat /etc/sysconfig/docker-storage-setup
DEVS='/dev/vdb'
VG=docker_vol
DATA_SIZE=95%VG
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME=dockerlv
CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
CONTAINER_ROOT_LV_SIZE=100%FREE
```

**OpenShift Emptydir Quota**

The parameter **openshift_node_local_quota_per_fsgroup** in the file **playbooks/openshift-setup.yaml** configures **perFSGroup** on all nodes. The **perFSGroup** setting restricts the ephemeral emptyDir volume from growing larger than 512Mi. This empty dir quota is done on the master, infrastructure, and application nodes.

**SSH** into the first infrastructure node (infra-node-0.internal.ocp3.example.com) to verify **/etc/origin/node/node-config.yaml** matches the information below.

```
$ sudo cat /etc/origin/node/node-config.yaml
...omitted...
volumeConfig:
  localQuota:
     perFSGroup: 512Mi
```

## 5.5. YUM REPOSITORIES

All systems except for the *bastion* instance should have the same subscriptions. To verify subscriptions match those defined in Required Channels perform the following. The repositories below are enabled while running the **prerequisite.yaml** playbook during the installation. The installation is unsuccessful if the repositories are missing from the system.

```
$ sudo yum repolist
Loaded plugins: search-disabled-repos, subscription-manager
repo id                                  repo name
status
rhel-7-fast-datapath-rpms/7Server/x86_64 Red Hat Enterprise Linux Fast
```

```
Datapath (RHEL 7 Server) (RPMs) 38
rhel-7-server-extras-rpms/x86_64         Red Hat Enterprise Linux 7 Server
- Extras (RPMs)              587+16
rhel-7-server-ose-3.6-rpms/x86_64        Red Hat OpenShift Container
Platform 3.6 (RPMs)                472+10
rhel-7-server-rpms/7Server/x86_64        Red Hat Enterprise Linux 7 Server
(RPMs)                       16,957
```

Or using Ansible:

```
$ ansible all -b -a "yum repolist"
```

### 5.5.1. List Router and Registry

List the router and registry by changing to the **`default`** project.

**`ssh`** into the first master instance ( *master-0.internal.ocp3.example.com*) or the *bastion* instance to verify the router and registry in the default project.

The command **`oc get all`** is used to observe the values.

```
$ oc get all -n default
NAME                  DOCKER REPO
TAGS      UPDATED
is/registry-console   docker-registry.default.svc:5000/default/registry-
console   v3.6      5 hours ago

NAME                  REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/docker-registry    1          1         1         config
dc/registry-console   1          1         1         config
dc/router             1          1         1         config

NAME                    DESIRED   CURRENT   READY     AGE
rc/docker-registry-1    1         1         1         4h
rc/registry-console-1   1         1         1         4h
rc/router-1             1         1         1         4h

NAME                        HOST/PORT
PATH      SERVICES            PORT      TERMINATION   WILDCARD
routes/docker-registry    docker-registry-default.apps.ocp3.example.com
docker-registry    <all>     passthrough   None
routes/registry-console   registry-console-default.apps.ocp3.example.com
registry-console   <all>     passthrough   None

NAME                   CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
svc/docker-registry    172.30.7.33     <none>        5000/TCP
4h
svc/kubernetes         172.30.0.1      <none>
443/TCP,53/UDP,53/TCP     4h
svc/registry-console   172.30.174.69   <none>        9000/TCP
4h
svc/router             172.30.83.208   <none>
80/TCP,443/TCP,1936/TCP   4h
```

```
NAME                           READY     STATUS    RESTARTS   AGE
po/docker-registry-1-ggpsg     1/1       Running   0          4h
po/registry-console-1-jcgn5    1/1       Running   0          4h
po/router-1-fhrj1              1/1       Running   0          4h
NAME                   DOCKER REPO
TAGS      UPDATED
is/registry-console    docker-registry.default.svc:5000/default/registry-
console   v3.6      5 hours ago

NAME                   REVISION   DESIRED   CURRENT   TRIGGERED BY
dc/docker-registry     1          1         1         config
dc/registry-console    1          1         1         config
dc/router              1          1         1         config

NAME                     DESIRED   CURRENT   READY     AGE
rc/docker-registry-1     1         1         1         4h
rc/registry-console-1    1         1         1         4h
rc/router-1              1         1         1         4h

NAME                          HOST/PORT
PATH      SERVICES            PORT      TERMINATION   WILDCARD
routes/docker-registry    docker-registry-default.apps.ocp3.example.com
docker-registry    <all>     passthrough   None
routes/registry-console   registry-console-default.apps.ocp3.example.com
registry-console   <all>     passthrough   None

NAME                    CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
svc/docker-registry     172.30.7.33     <none>        5000/TCP
4h
svc/kubernetes          172.30.0.1      <none>
443/TCP,53/UDP,53/TCP     4h
svc/registry-console    172.30.174.69   <none>        9000/TCP
4h
svc/router              172.30.83.208   <none>
80/TCP,443/TCP,1936/TCP   4h

NAME                           READY     STATUS    RESTARTS   AGE
po/docker-registry-1-ggpsg     1/1       Running   0          4h
po/registry-console-1-jcgn5    1/1       Running   0          4h
po/router-1-fhrj1              1/1       Running   0          4h
```

### 5.5.2. Explore the Registry

The OpenShift Ansible playbooks configure three infrastructure instances to have the ability to run the registry. Only one registry pod is running due to being backed by a **`cinder`** volume. In order to understand the configuration and mapping process of the registry pods, the command 'oc describe' is used. **`oc describe`** details how registries are configured and mapped to the Amazon **`S3`** buckets for storage. Using **`oc describe`** should help explain how HA works in this environment.

**`ssh`** into the first master instance ( *master-0.internal.ocp3.example.com*) or the *bastion* instance and use the default project to describe the registry.

```
$ oc describe svc/docker-registry -n default
Name:           docker-registry
```

```
Namespace:      default
Labels:         <none>
Annotations:        <none>
Selector:       docker-registry=default
Type:           ClusterIP
IP:         172.30.7.33
Port:           5000-tcp    5000/TCP
Endpoints:      172.45.16.10:5000
Session Affinity:   ClientIP
Events:         <none>
```

```
$ oc get pods -n default
NAME                        READY     STATUS    RESTARTS   AGE
docker-registry-1-ggpsg     1/1       Running   0          4h
```

```
$ oc describe pod docker-registry-1-ggpsg -n default
Name:           docker-registry-1-ggpsg
Namespace:      default
Security Policy:    hostnetwork
Node:           infra-node-1.internal.ocp3.example.com/172.16.77.13
Start Time:     Tue, 29 Aug 2017 10:32:43 -0400
Labels:         deployment=docker-registry-1
            deploymentconfig=docker-registry
            docker-registry=default
Annotations:        kubernetes.io/created-by=
{"kind":"SerializedReference","apiVersion":"v1","reference":
{"kind":"ReplicationController","namespace":"default","name":"docker-
registry-1","uid":"ea579212-8cc6-11e7-bd52-fa1...
            openshift.io/deployment-config.latest-version=1
            openshift.io/deployment-config.name=docker-registry
            openshift.io/deployment.name=docker-registry-1
            openshift.io/scc=hostnetwork
Status:         Running
IP:         172.45.16.10
Controllers:        ReplicationController/docker-registry-1
Containers:
  registry:
    Container ID:
docker://1ce05dc75aa9403a36f5239c49ab4c4ec11ec8dc58486d1744dd28d0ea44926e
    Image:      openshift3/ose-docker-registry:v3.6.173.0.5
    Image ID:       docker-
pullable://registry.access.redhat.com/openshift3/ose-docker-
registry@sha256:cb0a9039b7f037a3cca476f3b0f2fe32b9f21a37f15670e270bd01db21
b66c19
    Port:       5000/TCP
    State:      Running
      Started:      Tue, 29 Aug 2017 10:34:11 -0400
    Ready:      True
    Restart Count:  0
    Requests:
      cpu:  100m
      memory:   256Mi
    Liveness:   http-get https://:5000/healthz delay=10s timeout=5s
period=10s #success=1 #failure=3
    Readiness:  http-get https://:5000/healthz delay=0s timeout=5s
```

```
period=10s #success=1 #failure=3
    Environment:
      REGISTRY_HTTP_ADDR:                      :5000
      REGISTRY_HTTP_NET:                       tcp
      REGISTRY_HTTP_SECRET:
HQS/7MfynpqP911Mfqqrmw9rAA1W9DpwhqINZuGUzZk=
      REGISTRY_MIDDLEWARE_REPOSITORY_OPENSHIFT_ENFORCEQUOTA:    false
      OPENSHIFT_DEFAULT_REGISTRY:                  docker-
registry.default.svc:5000
      REGISTRY_HTTP_TLS_KEY:                       /etc/secrets/registry.key
      REGISTRY_HTTP_TLS_CERTIFICATE:
/etc/secrets/registry.crt
    Mounts:
      /etc/secrets from registry-certificates (rw)
      /registry from registry-storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from registry-token-
9379b (ro)
Conditions:
  Type      Status
  Initialized   True
  Ready     True
  PodScheduled  True
Volumes:
  registry-storage:
    Type:   PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName:  registry-claim
    ReadOnly:   false
  registry-certificates:
    Type:   Secret (a volume populated by a Secret)
    SecretName: registry-certificates
    Optional:   false
  registry-token-9379b:
    Type:   Secret (a volume populated by a Secret)
    SecretName: registry-token-9379b
    Optional:   false
QoS Class:  Burstable
Node-Selectors: role=infra
Tolerations:    <none>
Events:     <none>
```

### 5.5.3. Explore Docker Storage

This section explores the Docker storage on an infrastructure node.

The example below can be performed on any node but for this example the infrastructure instance(*infra-node-0.internal.ocp3.example.com*) is used.

The output below describing the **`Backing Filesystem: xfs`** states that the docker service is not using a loop-back device.

```
$ sudo docker info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 2
Server Version: 1.12.6
Storage Driver: overlay2
 Backing Filesystem: xfs
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: bridge host overlay null
 Authorization: rhel-push-plugin
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
Security Options: seccomp selinux
Kernel Version: 3.10.0-693.el7.x86_64
Operating System: OpenShift Enterprise
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 4
Total Memory: 15.51 GiB
Name: ip-10-20-5-71.ec2.internal
ID: UJZA:WUBN:J6ZI:4ZJ2:WZVL:EVJX:3JXX:2ZN7:VGFM:UBTG:UMOJ:G54U
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://registry.access.redhat.com/v1/
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Insecure Registries:
 127.0.0.0/8
Registries: registry.access.redhat.com (secure),
registry.access.redhat.com (secure), docker.io (secure)
```

Verify three disks are attached to the instance. The disk **/dev/vda** is used for the OS, **/dev/vdb** is used for docker storage, and **/dev/vdc** is used for emptyDir storage for containers that do not use a persistent volume.

```
$ fdisk -l

Disk /dev/vda: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000a73bb

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    41943006    20970479+  83  Linux

Disk /dev/vdb: 32.2 GB, 32212254720 bytes, 62914560 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000093ab

   Device Boot      Start         End      Blocks   Id  System
/dev/vdb1            2048    62914559    31456256   8e  Linux LVM

Disk /dev/vdc: 16.1 GB, 16106127360 bytes, 31457280 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

## 5.6. MANUALLY DEPLOYING AN APPLICATION

Besides of the use of the automated validation playbook that was described above, this section deploys an application and sets a route allowing for the application to be accessed by the web browser.

The example below can be performed on any of the master instance but for this example the first master instance(*master-0.internal.ocp3.example.com*) or the *bastion* instance is used.

```
$ oc new-project validate
$ oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-
ex.git
$ oc get svc
NAME      CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
ruby-ex   172.30.15.38   <none>        8080/TCP   15s
$ oc get route
No resources found.
$ oc expose svc ruby-ex
$ oc get route
NAME      HOST/PORT                                   PATH      SERVICES
PORT       TERMINATION   WILDCARD
ruby-ex   ruby-ex-validate.apps.ocp3.example.com             ruby-ex
8080-tcp                 None
$ curl ruby-ex-validate.apps.ocp3.example.com
```

# CHAPTER 6. CONCLUSION

Red Hat solutions involving the Red Hat OpenShift Container Platform provide an excellent foundation for building a production ready environment that simplifies the deployment process, provides the latest best practices, and stability in running applications in a highly available environment.

The steps and procedures described in this reference architecture provide OpenStack, system, storage and OpenShift administators the blueprint required to create a solution to meet their business needs. Administrators may reference this document to simplify and optimize their Red Hat OpenShift Container Platform on Red Hat OpenStack Platform environments with the following tasks:

- Deciding between different internal network technologies
- Provisioning a Red Hat OpenStack Platform for Red Hat OpenShift Container Platform readiness
- Deploying Red Hat OpenShift Container Platform 3.6
- Using dynamic provisioned storage
- Verifying a successful installation
- Troubleshooting common pitfalls

For any questions or concerns, please email refarch-feedback@redhat.com and ensure to visit the Red Hat Reference Architecture page to find about all of our Red Hat solution offerings.

# APPENDIX A. CREATING THE DNS RED HAT OPENSTACK PLATFORM INSTANCE

The installation process for Red Hat OpenShift Container Platform depends on a reliable name service that contains an address record for each of the target instances. When installing Red Hat OpenShift Container Platform in a cloud environment, the IP address of each instance is not known at the beginning. The address record for each instance must be created dynamically by the orchestration system as the instances themselves are created.

Red Hat OpenStack Platform does not have an integrated DNS service, but it is possible to create new records in a DNS service using dynamic updates as defined in RFC2137. This method uses a symmetric key to authenticate and authorize updates to a specific zone.

Installation of Red Hat OpenShift Container Platform on Red Hat OpenStack Platform requires a DNS service capable of accepting DNS update records for the new instances. This section describes a method to create a suitable DNS service within Red Hat OpenStack Platform. The DNS is capable of accepting DNS queries and update requests for the Red Hat OpenStack Platform service. It is suitable for delegation to make the Red Hat OpenShift Container Platform service accessible to users.

The following steps create a DNS Red Hat OpenStack Platform instance and configure it.

1. Source the RC file for the project

```
$ source /path/to/examplerc
```

2. Create a security group to allow incoming connections to the DNS service (ports 53 UDP and TCP), **ssh** to manage the instance and ICMP for debug purposes.

```
$ openstack security group create <dns-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <dns-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    <dns-sg-name>
$ for PROTO in udp tcp;
do
  openstack security group rule create \
    --ingress \
    --protocol $PROTO \
    --dst-port 53 \
    <dns-sg-name>;
done
```

3. Create the Red Hat OpenStack Platform instance

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -
c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --security-group=<dns-sg-name> \
    --flavor <flavor-name> \
    --image <image-name> \
    --key-name <keypair-name> <instance-name>
```

4. Assign and attach a floating IP to the instance

```
$ openstack floating ip create <public-network-name>
$ openstack server add floating ip <instance-name> <ip>
```

**NOTE**

If the DNS server doesn't need to be accessed from outside, the floating ip can be safely removed once the bastion host is configured.

5. Log into the Red Hat OpenStack Platform instance that shall become the the DNS server.

```
$ ssh -i /path/to/<keypair-name>.pem cloud-user@<IP>
```

## A.1. SETTING UP THE DNS RED HAT OPENSTACK PLATFORM INSTANCE

The following steps configure a DNS server once logged into the instance.

1. Subscribe to the following repositories using **`subscription-manager`**

```
$ sudo subscription-manager register
$ sudo subscription-manager attach --pool <pool_id>
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms \
    --enable rhel-7-server-rh-common-rpms
```

2. Install the following packages: **`firewalld`**, **`python-firewall`**, **`bind-utils`** and **`bind`**:

```
$ sudo yum install -y firewalld python-firewall bind-utils bind
```

3. Enable the **`firewalld`** service and start it

```
$ sudo systemctl enable firewalld && sudo systemctl start firewalld
```

4. Add the DNS service to the **`firewalld`** public zone

```
$ sudo firewall-cmd --zone public --add-service dns
$ sudo firewall-cmd --zone public --add-service dns --permanent
```

5. Copy the **`named.conf`** prior to modification

```
$ sudo cp /etc/named.conf{,.orig}
```

The following **`diff`** command shows the difference between the original *named.conf* file and newest *named.conf* file. The **-** means the line was removed, while the **+** means the line has been added to the newest *named.conf*. When modifying the *named.conf* remove all lines with a **-** and include all lines with **+**.

```
$ diff -u /etc/named.conf.orig /etc/named.conf
--- /etc/named.conf.orig        2016-03-22 14:15:45.000000000 +0000
+++ /etc/named.conf     2017-05-03 16:21:00.579000000 +0000
@@ -6,17 +6,15 @@
 //
 // See /usr/share/doc/bind*/sample/ for example named configuration
files.
 //
-// See the BIND Administrator's Reference Manual (ARM) for details about
the
-// configuration located in /usr/share/doc/bind-{version}/Bv9ARM.html

 options {
-   listen-on port 53 { 127.0.0.1; };
-   listen-on-v6 port 53 { ::1; };
+   listen-on port 53 { any ; };
+   // listen-on-v6 port 53 { ::1; };
    directory   "/var/named";
    dump-file   "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
-   allow-query     { localhost; };
+   allow-query     { any ; };

    /*
     - If you are building an AUTHORITATIVE DNS server, do NOT enable
recursion.
@@ -28,18 +26,27 @@
       attacks. Implementing BCP38 within your network would greatly
       reduce such attack surface
    */
-   recursion yes;
+   // recursion yes;
+  forward only;
+  forwarders { <IP of DNS1 forwarder> ; <IP of DNS2 forwarder> ; } ;

-   dnssec-enable yes;
-   dnssec-validation yes;
+   // dnssec-enable yes;
+   // dnssec-validation yes;

    /* Path to ISC DLV key */
-   bindkeys-file "/etc/named.iscdlv.key";
+   /* In case you want to use ISC DLV, please uncomment the following
line. */
+   //bindkeys-file "/etc/named.iscdlv.key";

    managed-keys-directory "/var/named/dynamic";

    pid-file "/run/named/named.pid";
    session-keyfile "/run/named/session.key";
+
+   /* https://fedoraproject.org/wiki/Changes/CryptoPolicy */
+  //include "/etc/crypto-policies/back-ends/bind.config";
+
+        /* tickle slaves to pull updates */
+        notify yes ;
 };

 logging {
@@ -54,6 +61,10 @@
    file "named.ca";
 };

+
 include "/etc/named.rfc1912.zones";
 include "/etc/named.root.key";

+include "/etc/named/zones.conf" ;
```

Once the changes have been made, save the the *named.conf* file.

Next, create a *zones.conf* file that provides the zone that is to be used for the RHOCP installation. An example of a *zones.conf* is shown below.

### /etc/named/zones.conf

```
include "/etc/named/update.key" ;

zone ocp3.example.com {
  type master ;
  file "/var/named/dynamic/zone.db" ;
  allow-update { key update-key ; } ;
};
```

Once the *zones.conf* file has been created, one must update the */var/named/dynamic/zone.db* to contain all the Red Hat OpenShift Container Platform hosts and their IP addresses.

**NOTE**

Due to not having any Red Hat OpenStack Platform instances created yet, thus no IP addresses, return to this section for completing the input of IPs required within the *zones.db* file once Section 3.9, “Creating RHOSP Instances for RHOCP” section is completed.

Example of the */var/named/dynamic/zone.db* file:

### /var/named/dynamic/zone.db

```
$ORIGIN .
$TTL 300        ; 5 minutes
ocp3.example.com IN SOA ns-master.ocp3.example.com. dnsadmin.example.com.
(
                                3          ; serial
                                28800      ; refresh (8 hours)
                                3600       ; retry (1 hour)
                                604800     ; expire (1 week)
                                86400      ; minimum (1 day)
                                )
                        NS      ns-master.ocp3.example.com.
$ORIGIN apps.ocp3.example.com.
*                       A       <public_IP_of_haproxy_server>
$ORIGIN control.ocp3.example.com.
app-node-0              A       <control-net-IP>
app-node-1              A       <control-net-IP>
app-node-2              A       <control-net-IP>
bastion                 A       <control-net-IP>
infra-node-0            A       <control-net-IP>
infra-node-1            A       <control-net-IP>
infra-node-2            A       <control-net-IP>
master-0                A       <control-net-IP>
master-1                A       <control-net-IP>
master-2                A       <control-net-IP>
$ORIGIN ocp3.example.com.
bastion                 A       <public_IP>
dns                     A       <public_IP>
ns-master               A       <public_IP>
ns0                     A       <public_IP>
ns1                     A       <public_IP>
ocp3                    A       <public_IP_of_haproxy_server>
```

In order to update the DNS, the DNS requires a special key string that can be generated by **`rndc-confgen`** which is part of the **`bind`** RPM. The *update.key* file should exist within the */etc/named* directory.

To create the *update.key* file run the following command:

```
$ sudo rndc-confgen -a -c /etc/named/update.key -k update-key -r
/dev/urandom
$ sudo chown root.named /etc/named/update.key
$ sudo chmod 640 /etc/named/update.key
```

Example output:

```
$ sudo cat /etc/named/update.key
key "update-key" {
    algorithm hmac-md5;
    secret "key string"; # don't use this
};
```

**WARNING**

The significant part of the key output is the secret field. This should be kept secret as someone may make edits to the DNS entries with the key.

Once everything is set, configure *named* service to start at boot and start it:

```
$ sudo systemctl enable named --now
```

Verification of the entire zone can be done using the **`dig`** command. The example below shows the records of the DNS master **`ns-master.ocp3.example.com`**

```
$ dig @<DNS_IP> ocp3.example.com axfr

; <<>> DiG 9.9.4-RedHat-9.9.4-38.el7_3.3 <<>> @<IP> ocp3.example.com axfr
; (1 server found)
;; global options: +cmd
ocp3.example.com.   300 IN  SOA ns-master.ocp3.example.com.
admin.ocp3.example.com.ocp3.example.com. 0 28800 3600 604800 86400
ocp3.example.com.   300 IN  NS  ns-master.ocp3.example.com.
ns-master.ocp3.example.com. 300 IN  A   <IP>
ocp3.example.com.   300 IN  SOA ns-master.ocp3.example.com.
admin.ocp3.example.com.ocp3.example.com. 0 28800 3600 604800 86400
;; Query time: 2 msec
;; SERVER: <DNS_IP>#53(<DNS_IP>)
;; WHEN: Wed Jun 07 11:46:33 EDT 2017
```

**NOTE**

Creating a reverse DNS zone is out of scope of this guide. Refer to the official documentation for proper instructions.

# APPENDIX B. USING FLANNEL

*flannel* replaces the default *openshift_sdn* mechanism thus eliminating the second OVS instance. An extra Red Hat OpenStack Platform network is required to carry container-to-container traffic to isolate it from the operational communications where this network is for internal use only and does not have a gateway to connect to other networks. All instances that are able to run containers must have a network interface connected to the extra network for container traffic.

The current version of *neutron* enforces port security on ports by default. This prevents the port from sending or receiving packets with a MAC address different from that on the port itself. *flannel* creates virtual MACs and IP addresses and must send and receive packets on the port so port security must be disabled on the ports that carry *flannel* traffic.

The Red Hat OpenShift Container Platform installation process needs to be tweaked to reflect those architecture changes as well as some networking rules need to be added to allow *flannel* traffic between hosts.

## B.1. FLANNEL RED HAT OPENSTACK PLATFORM NETWORK

After the Red Hat OpenStack Platform instances are created, the internal network to carry the inter container traffic is created.

As the user(s) implementing the Red Hat OpenShift Container Platform environment, within the Red Hat OpenStack Platform director node or workstation, ensure to **`source`** the credentials as follows:

```
$ source path/to/examplerc
```

Create the internal network and the associated subnet

```
$ openstack network create <internal-net-name>
$ openstack subnet create \
    --network <internal-net-name> \
    --subnet-range <cidr> \
    --gateway none \
    <internal-subnet-name>
```

**NOTE**

Use a different CIDR for the internal network and the private network.

Verify the network and subnet has been created

```
$ openstack network list
$ openstack subnet list
```

## B.2. PORTS

Once the network and instances have been created, the instances need to be attached to the new network. Red Hat OpenStack Platform command line requires to create a network port in the internal network and then attach the port to the required instance.

```
$ domain=<domain>
$ network=$(openstack network show <internal-net-name> -f value -c id)
$ for instance in master-{0..2} app-node-{0..2} infra-node-{0..2};
do
  instanceid=$(openstack server show $instance.$domain -f value -c id);
  port=$(openstack port create --network $network $instance-port -f value
-c id);
  neutron port-update $port --no-security-groups --port-security-
enabled=False;
  nova interface-attach --port-id $port $instanceid;
done
```

The interfaces are created as *eth1* device in the Red Hat OpenShift Container Platform instances with a private IP from the internal network CIDR range specified for the subnet.

## B.3. RED HAT OPENSHIFT CONTAINER PLATFORM PREINSTALLATION TASKS

Using *flannel* requires some prerequisite steps to ensure the instances are properly setup before running the Red Hat OpenShift Container Platform installer.

**NOTE**

These pre installation steps should be executed just before the installation

### B.3.1. Enable Flannel ETCD Query

Flannel requires to gather information from *etcd* to configure and assign the subnets in the nodes, therefore, the master security group should allow access from nodes to port 2379/tcp

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 2379 \
    --src-group <node-sg-name>
    <master-sg-name>
```

### B.3.2. Enable eth1 Interface at Boot Time

It is required that the new interfaces be enabled at boot time for *flannel* and Red Hat OpenShift Container Platform to work. Red Hat Enterprise Linux requires a new file in the */etc/sysconfig/network-scripts* directory named *ifcfg-<interface_name>* with the proper configuration for enabling the interface at boot time.

The following lines show how to create the file with the appropriate content:

```
$ export KEY=/path/to/key.pem
$ for instance in master-{0..2} app-node-{0..2} infra-node-{0..2};
do
  cat << EOF | ssh -i $KEY cloud-user@$instance 'sudo tee
/etc/sysconfig/network-scripts/ifcfg-eth1'
DEVICE=eth1
TYPE=Ethernet
```

```
BOOTPROTO=dhcp
ONBOOT=yes
DEFTROUTE=no
PEERDNS=no
EOF
  ssh -i $KEY cloud-user@$instance 'sudo ifup eth1'
done
```

### B.3.3. Ansible Inventory Modifications

Using *flannel* requires some parameters to be modified from the Ansible inventory to enable the flannel installation and configuration of Red Hat OpenShift Container Platform.

The following **diff** command shows the difference between the original */etc/ansible/hosts* file and newest */etc/ansible/hosts* file. The **-** means the line was removed, while the **+** means the line has been added to the newest */etc/ansible/hosts*. When modifying the */etc/ansible/hosts* remove all lines with a **-** and include all lines with **+**.

```
$ diff -u hosts.sdn hosts
--- hosts.sdn   2017-09-05 09:16:14.389275712 -0400
+++ hosts   2017-09-05 09:16:57.796275712 -0400
@@ -32,8 +32,10 @@
 openshift_master_default_subdomain=apps.ocp3.example.com

 osm_default_node_selector="role=app"
-os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
 osm_use_cockpit=true
+openshift_use_openshift_sdn=false
+openshift_use_flannel=true
+flannel_interface=eth1

 containerized=false
```

**NOTE**

In order to provide custom networking CIDR for pods and services using flannel, there is a bugzilla being invested by Red Hat Bugzilla ID 1473858 at the time of writing this paper.

### B.3.4. Installation

After properly configuring the Ansible inventory, the standard Red Hat OpenShift Container Platform installation procedure is used to deploy the Red Hat OpenShift Container Platform, including accomplish all the Section 4.1.6, “Check instances” and the Red Hat OpenShift Container Platform ocp_installation.

### B.3.5. Post Installation Tasks

After the Red Hat OpenShift Container Platform installation, *iptables* rules should be applied in all the hosts to allow container traffic to work.

**NOTE**

These steps should be done just after the Red Hat OpenShift Container Platform installation as the installation process overwrite the *iptables* rules.

1. Backup the */etc/sysconfig/iptables* file

   ```
   $ ansible nodes -b -m copy \
       -a "src=/etc/sysconfig/iptables
   dest=/etc/sysconfig/iptables.orig remote_src=true"
   ```

2. Add *iptables* rules to the running config

   ```
   $ ansible nodes -b -m shell \
       -a 'iptables -A DOCKER -p all -j ACCEPT'
   $ ansible nodes -b -m shell \
       -a 'iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE'
   ```

3. Persist *iptables* rules

   ```
   $ ansible nodes -b -m shell \
       -a "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/
   s/:DOCKER -/:DOCKER ACCEPT/' | awk '"\!"p && /POSTROUTING/{print \"-
   A POSTROUTING -o eth1 -j MASQUERADE\"; p=1} 1' | tac >
   /etc/sysconfig/iptables"
   ```

**NOTE**

Instead accepting all the protocols traffic, the DOCKER iptables chain is created with ACCEPT policy by default.

After running the post installation tasks, the Red Hat OpenShift Container Platform environment is ready to run applications. The steps provided in the Chapter 5, *Operational Management* can be executed to ensure it is properly running.

# APPENDIX C. REVISION HISTORY

| | | |
|---|---|---|
| **Revision 3.6.2-0** | **2018-03-05** | **Eduardo Minguez (edu@redhat.com)** |

- **Added new diagram.**

| | | |
|---|---|---|
| **Revision 3.6.1-0** | **2018-02-09** | **Eduardo Minguez (edu@redhat.com)** |

- **Added some fixes and clarifications.**

| | | |
|---|---|---|
| **Revision 3.6.0-0** | **2017-09-18** | **Ryan Cook (rcook@redhat.com), , Roger Lopez (rlopez@redhat.com), , Eduardo Minguez (edu@redhat.com)** |

- **Initial version.**