Reference Architectures 2017

Deploying Red Hat OpenShift Container Platform 3.4 on Red Hat OpenStack Platform 10

Last Updated: 2017-11-30

Reference Architectures 2017 Deploying Red Hat OpenShift Container Platform 3.4 on Red Hat OpenStack Platform 10

Mark Lamourine

Roger Lopez
refarch-feedback@redhat.com

Legal Notice

Abstract

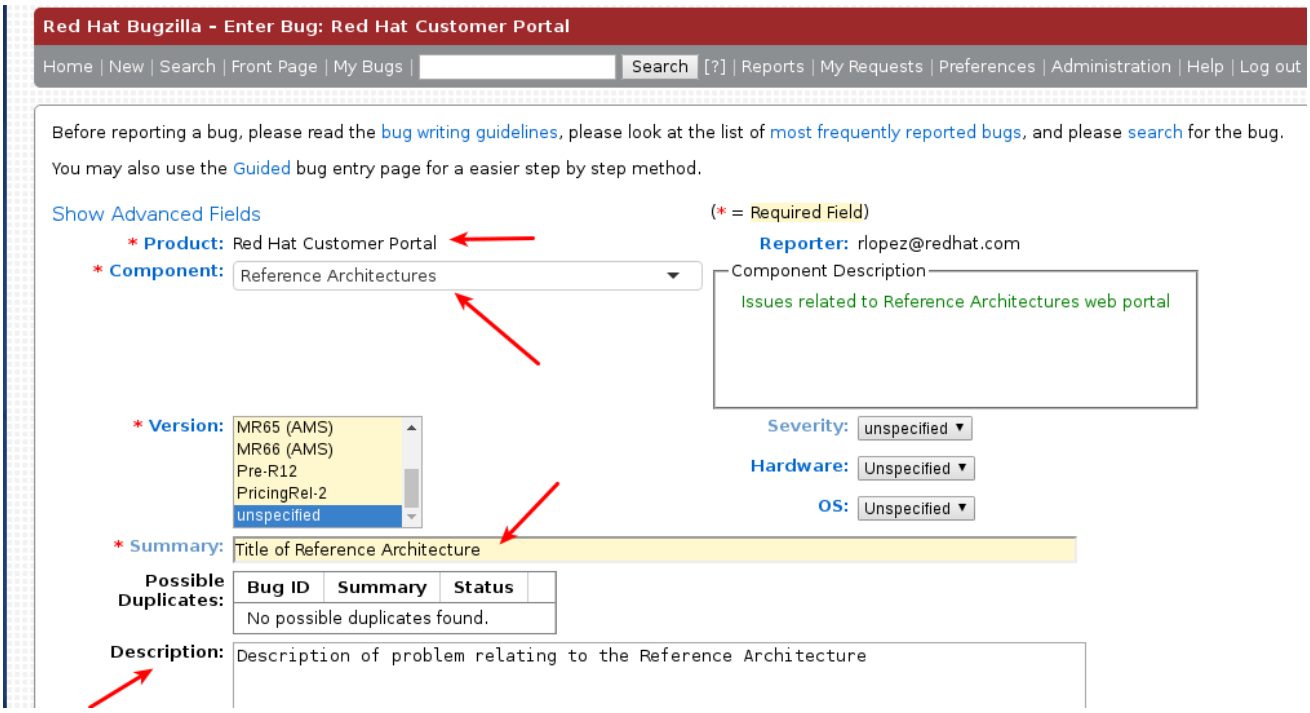
The purpose of this document is to provide guidelines and considerations for installing and configuring Red Hat OpenShift Container Platform 3.4 on Red Hat OpenStack Platform 10.

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.



CHAPTER 1. EXECUTIVE SUMMARY

With the evolving business requirements that IT organizations are challenged with daily, building solutions that are not only robust, but delivered in a timely manner is of the utmost importance. With the use of Red Hat OpenShift Container Platform (RHOCP), IT organizations can meet these requirements due to its ability of enabling developers to focus on application functionality and features while reducing time for infrastructure implementation.

This document describes a sample deployment of Red Hat OpenShift Container Platform (RHOCP) on Red Hat OpenStack Platform (RHOSP) highlighting each of the steps in the installation and the interaction between RHOCP and RHOSP.

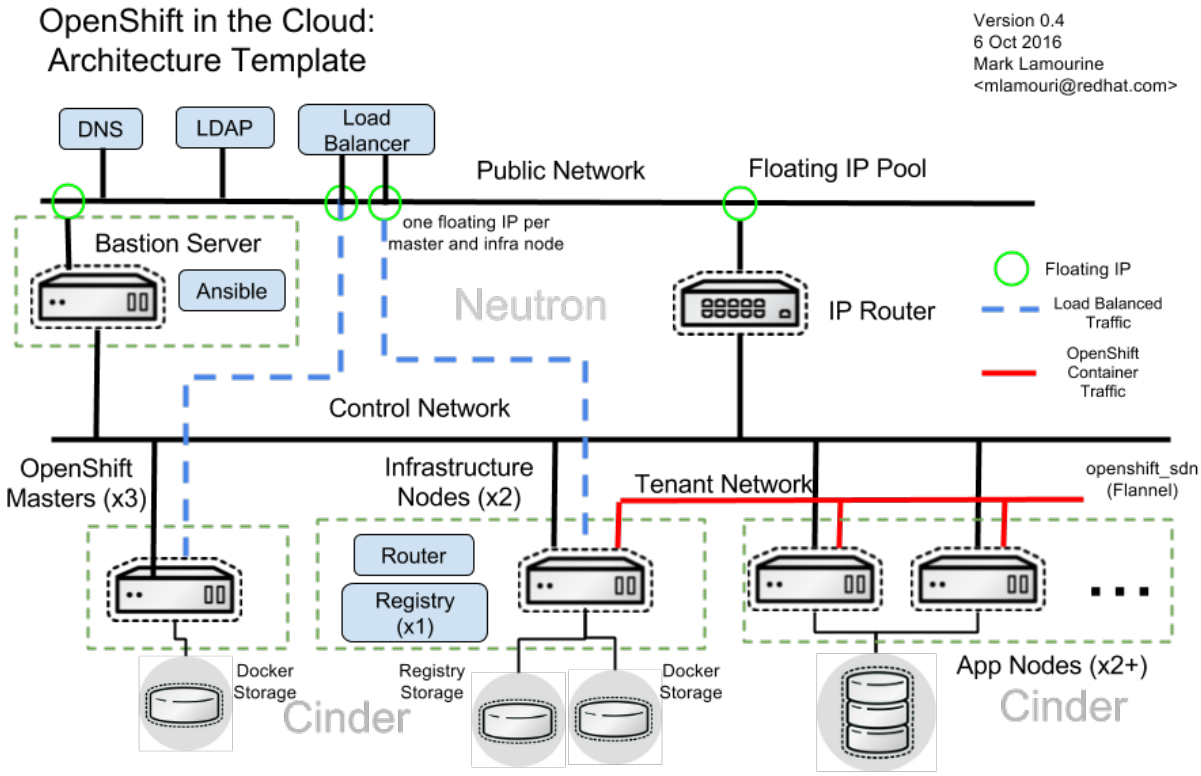
CHAPTER 2. REFERENCE ARCHITECTURE OVERVIEW

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 run-time life-cycle that requires no server management.

The reference environment described in this document consists of a single *bastion host*, three *master hosts*, and five *node hosts* running the Docker containers on behalf of its users. The *node hosts* are separated into two functional classes: *infrastructure nodes* and *app nodes*. The *infra nodes* run the internal RHOCP services, *OpenShift Router* and the *Local Registry*. The remaining three *app nodes* host the user container processes.

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

Figure 2.1. OpenShift Container Platform Architecture



When viewing the OpenShift Container Platform Architecture diagram, it is best seen as four layers.

The first layer consists of the prerequisites: DNS server, LDAP, Load-Balancer, Public Network, and Floating IP Pool that has readily available public IPs for the different nodes to be created.

The second layer consists of the *Bastion* server that runs *Ansible*, and the *IP Router* that routes traffic across all the nodes via the *Control network*.

The third layer consists of all the nodes: *master*, *infrastructure*, *app* nodes. This layer shows the connectivity of the OpenShift Master nodes, Infrastructure nodes, and App nodes.

The fourth layer consists of Docker storage and Cinder storage connected to their respective nodes.

The main distinction between the *infrastructure nodes* and the *app nodes* is the placement of the OpenShift Router and a public interface. The *infrastructure nodes* require a public IP address in order to communicate with the load-balancer. Once the traffic reaches the OpenShift router, the router forwards traffic to the containers on the *app nodes* thus eliminating the need for the *app nodes* to require a public interface.

All of the participating instances have two active network interfaces. These are connected to two networks with distinct purposes and traffic. The *control network* carries all of the communications between OpenShift service processes. The floating IPs of the *bastion* host, the *master nodes* and the *infrastructure nodes* come from a floating pool routed to this network. The *tenant network* carries and isolates the container-to-container traffic. This is directed through a *flannel* service which carries packets from one container to another.

Each node instance creates and mounts a *cinder* volume. The volume contains the Docker run-time disk overhead for normal container operations. Mounting from *cinder* allows the size of the disk space to be determined independently without the need to define a new OpenStack *flavor*. Each node offers Cinder access to the containers it hosts through RHOCP.

2.1. INTERNAL NETWORKS

RHOCP establishes a network for communication between containers on separate nodes. In the Architecture Diagram this is designated by the *tenant network*. However, the *tenant network* is actually two networks; one network encapsulated within another. The outer network (tenant network) is the *neutron* network that carries traffic between nodes. The second network carries container to container traffic.

RHOCP offers two mechanisms to operate the internal network; *openshift_sdn* and *flannel*. This reference environment uses the *flannel* network for the reasons detailed below.

The default mechanism is the *openshift_sdn*. This is a custom Open vSwitch (OVS) SDN that offers fine-grained dynamic routing and traffic isolation. On a bare-metal installation, *openshift_sdn* offers both control and high performance.

If the default mechanism (openshift_sdn) is used to operate the RHOCP internal network, all RHOCP internode traffic carried over OpenStack *neutron* undergoes *double encapsulation*, creating significant network performance degradation.

flannel replaces the default *openshift_sdn* mechanism thus eliminating the second OVS instance. *flannel* is used in *host-gateway* mode that routes packets using a lookup table for the destination instance and then forwards them directly to that instance encapsulated only with a single UDP header. This provides a performance benefit over OVS double-encapsulation. It, however, does come with a cost.

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.

In this reference environment, performance is given precedence over tenant isolation.

2.2. USERS AND VIEWS - THE FACES OF RHOCP

RHOCP is a software container service. When fully deployed in a cloud environment, it has three operational views for the different classes of users. These users work with RHOCP in different ways, performing distinct tasks. They see different views of RHOCP.

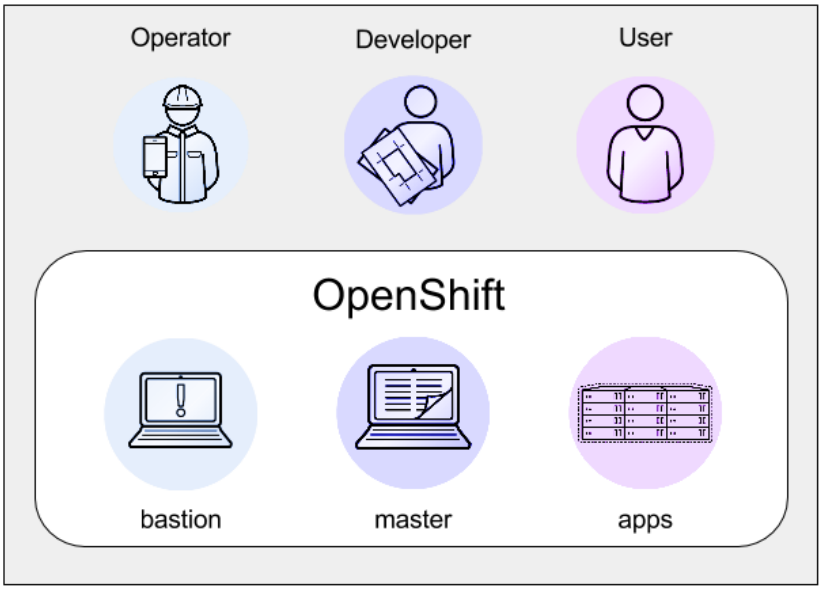
When deploying RHOCP, it is important to understand these views and how the service presents itself to different users.

Operations Users use a *bastion host* to reach the service internals. The *bastion host* serves as a stable trusted platform for service monitoring and managing updates.

Application Developers use the CLI or web interfaces of the RHOCP service to create and manage their applications.

Application Users see RHOCP applications as a series of web services published under a blanket DNS domain name. These all forward to the OpenShift Router an internal service that creates the connections between external users and internal containers.

Figure 2.2. RHOCP User Views



When the deployment process is complete, each type of user is able to reach and use their resources. The RHOCP service makes the best possible use of the OpenStack resources and where appropriate, making them available to the users.

Each of these views corresponds to a host or virtual host (via a load-balancer) that the user sees as their gateway to the RHOCP service.

Table 2.1. Sample User Portal Hostname

Role	Activity	Hostname
Operator	Monitoring and management of RHOCP service	`bastion.ocp3.example.com`

Role	Activity	Hostname
Application Developer	Creating, deploying and monitoring user services	`devs.ocp3.example.com`
Application User	Accessing apps and services to do business tasks	`*.apps.ocp3.example.com`

2.3. LOAD-BALANCERS AND HOSTNAMES

An external DNS server is used to make the service views visible. Most installations contain an existing DNS infrastructure capable of dynamic updates.

The developer interface, provided by the OpenShift *master* servers, can be spread across multiple instances to provide both load-balancing and high-availablity properties. This guide uses an external load-balancer running **haproxy** to offer a single entry point for developers.

Application traffic passes through the OpenShift Router on its way to the container processes inside the service. The OpenShift Router is really a reverse proxy service that multiplexes the traffic to multiple containers making up a scaled application running inside RHOCP. It itself can accept inbound traffic on multiple hosts as well. The load-balancer used for developer HA acts as the public view for the RHOCP applications, forwarding to the OpenShift Router endpoints.

Since the load-balancer is an external component and the hostnames for the OpenShift master service and application domain are known beforehand, it can be configured into the DNS prior to starting the deployment procedure.

The load-balancer can handle both the master and application traffic. The master web and REST interfaces are on port 8443/TCP. The applications use ports 80/TCP and 443/TCP. A single load-balancer can forward both sets of traffic to different destinations.

With a load-balancer host address of 10.19.x.y, the two DNS records can be added as follows:

Table 2.2. Load Balancer DNS records

IP Address	Hostname	Purpose
10.19.x.y	`devs.ocp3.example.com`	Developer access to OpenShift master web UI and REST API
10.19.x.y	`*.apps.ocp3.example.com`	User access to application web services

The second record is called a *wildcard* record. It allows all hostnames under that subdomain to have the same IP address without needing to create a separate record for each name.

This allows RHOCP to add applications with arbitrary names as long as they are under that subdomain. For example DNS name lookups for `tax.apps.ocp3.example.com` and `home-goods.apps.ocp3.example.com` would both get the same IP address: `10.19.x.y`. All of the traffic is forwarded to the OpenShift Routers that examines the HTTP headers of the queries and forward them to the correct destination.

The destination for the master and application traffic is set in the load-balancer configuration after each instance is created and the floating IP address is assigned.

2.4. DEPLOYMENT METHODS

This document describes two ways of deploying RHOCP. The first is a *manual* process to create and prepare the run-time environment in RHOSP and then install RHOCP on the instances. The manual process is presented first to provide a detailed explanation of all the steps involved in the deployment process and to provide a better understanding of all the components and their interworkings.

The second method uses **`Heat`** orchestration to deploy the RHOCP service. Heat is the OpenStack service that automates complex infrastructure and service deployments. With Heat, the user describes the structure of a set of OpenStack resources in a **`template`**, and the Heat Engine creates a **`stack`** from the template, configuring all of the networks, instances, volumes in a single step.

The RHOCP on RHOSP Heat templates, along with some customization parameters, define the RHOSP service and the Heat engine executes the deployment process.

Heat orchestration offers serveral benefits including the removal of the operator from the process thus providing a consistent repeatable result.

CHAPTER 3. DEPLOYMENT PROCESS OVERVIEW

This section outlines the deployment process. This consists of a set of infrastructure and host configuration steps culminating with the actual installation of RHOCP using Ansible. The next section details the prerequisites and configuration values that are needed to execute these steps.

The deployment of RHOCP is executed by a set of Ansible playbooks created by the RHOCP team that install and configure RHOCP on a set of existing hosts or instances.

3.1. PROVISIONING STEPS

The high level steps include:

1. Configure RHOSP network
2. Deploy instances and attach to network
3. Update external services (DNS, load-balancer)
4. Configure and tune instances
5. Define deployment parameters (create Ansible input files)
6. Deploy RHOCP (execute Ansible playbooks)
7. Validate and Monitor

The following sections describe the steps in detail.

3.1.1. Configure Red Hat OpenStack Platform Networks

RHOCP runs on two networks. The *control network* is used for instance to instance communcations. The *tenant network* carries container-to-container traffic to isolate it from the operational communications.

3.1.2. Deploy Instances and Attach to Networks

This reference environment is made up of eight instances all attached to both the control and tenant networks. The bastion host, the OpenShift master hosts and the infrastructure nodes are assigned floating IP addresses in order to be accessible to the outside world.

3.1.3. Update External Services (DNS, Load-Balancer)

Once the floating IP addresses have been allocated and attached to the bastion, masters and infrastructure nodes these addresses are mapped to their host names and published via DNS. The master and infrastructure node IPs are configured into the load-balancer so that inbound connections are properly forwarded.

3.1.4. Configure and Tune Instances

The instances are created from stock RHEL images. The images need to be registered for software updates and install minimal package sets.

The bastion host gets the full complement of Ansible software. The remaining instances are configured to allow Ansible to run from the bastion host to execute the installation.

3.1.5. Define RHOCP Deployment Parameters - (Create Ansible Input Files)

Using the names and IP addresses of the new instances, create the Ansible **`inventory`** to define the layout of the service on the hosts.

Using the pre-defined values for the RHOCP service, fill in the deployment variables for RHOCP in the appropriate file.

3.1.6. Deploy RHOCP (Execute Ansible)

Execute the **`ansible-playbook`** to deploy RHOCP from the bastion host to the service instances.

3.1.7. Validate and Monitor

Verify that each of the users listed can access their services. Deploy a sample application as a developer and access it as a user.

CHAPTER 4. PREREQUISITES AND PREPARATION

This procedure depends on a number of services and settings that must be in place before proceeding.

This procedure assumes:

- Red Hat OpenStack Platform 10 or later
- OSP *port security* controls must be enabled in **neutron** service
- Increase **keystone** token expiration
- RHOSP user account and credentials
- A Red Hat Enterprise Linux 7.3 cloud image pre-loaded in **glance**
- A Red Hat Enterprise Linux update subscription credentials (user/password or Satellite server)
- An SSH keypair pre-loaded in **nova**
- A publicly available **neutron** network for inbound access
- A pool of floating IP addresses to provide inbound access points for instances
- A host running **haproxy** to provide load-balancing
- A host running **bind** with a properly delegated sub-domain for publishing, and a key for dynamic updates.
- An existing LDAP or Active Directory service for user identification and authentication

4.1. RED HAT OPENSTACK PLATFORM

This procedure was developed and tested on Red Hat OpenStack Platform 10. While it may apply to other releases, no claims are made for installations on versions other than RHOSP 10.

This procedure makes use of the following RHOSP service components:

- **keystone** - Identification and Authentication
- **nova**- Compute Resources an Virtual Machines
- **neutron**- Software Defined Networks
- **glance**- Bootable Image Storage
- **cinder**- Block Storage
- **ceilometer**- Resource Metering
- **gnocchi**- Time-Series Database
- **aodh**- Alarms and Responses
- **heat**- Deployment Orchestration

NOTE

Red Hat OpenStack Platform 10 requires several patches to support the process detailed here. See Appendix C

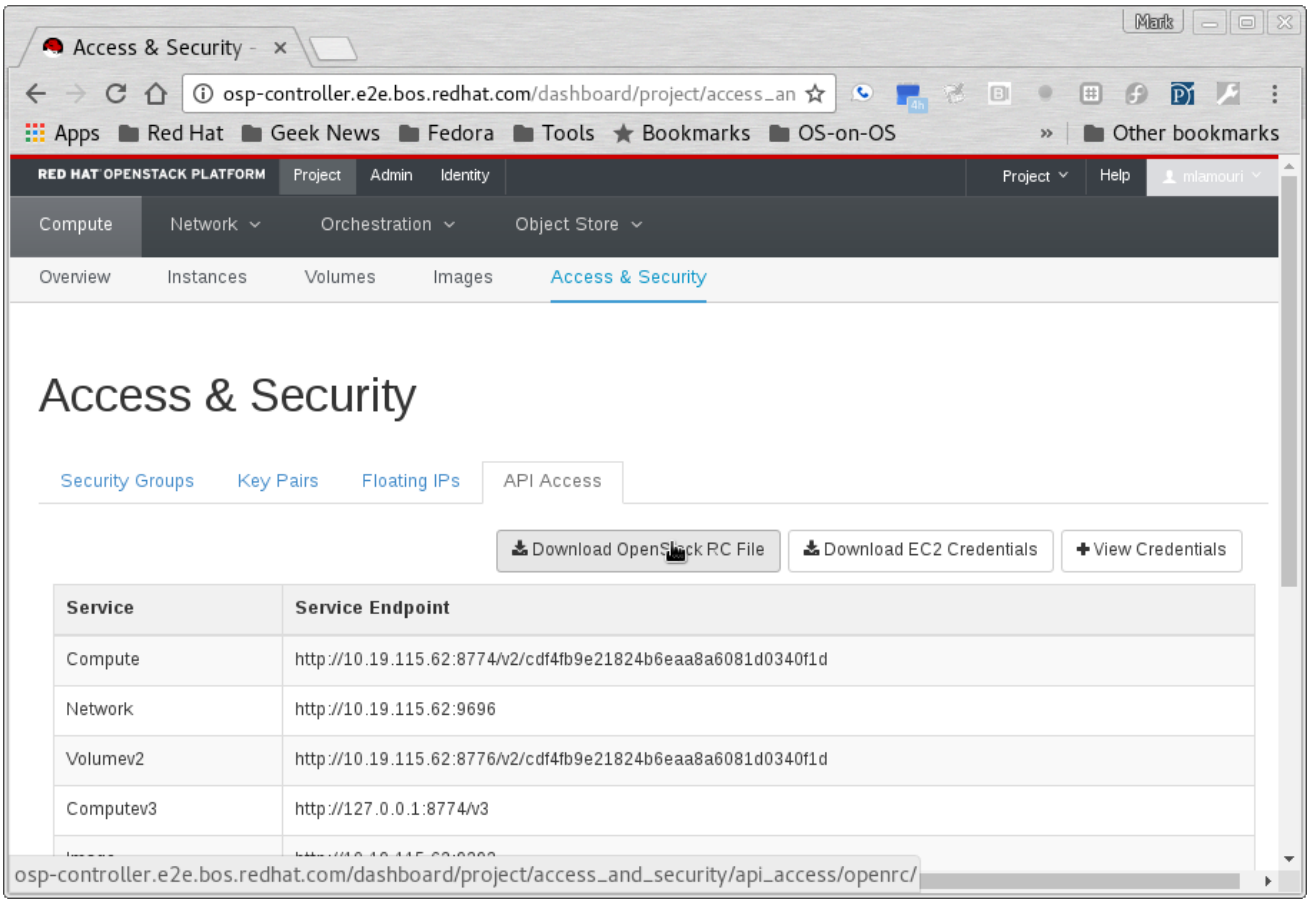
4.2. RED HAT OPENSTACK PLATFORM USER ACCOUNT AND CREDENTIALS

All of the RHOSP commands in this reference environment are executed using the CLI tools. The user needs a RHOSP account, and a project to contain the service.

For the manual procedure the user needs only the **member** role on the project. Running the installation using Heat requires the **heat_stack_owner** role.

The RHOSP credentials are commonly stored in a source file that sets the values of their shell environment.

A script fragement from the RHOSP **keystone** web console is available as shown below.



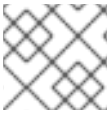
A simpler format can be used and is provided in the following code snippet:

sample ks_ocp3

```
unset OS_AUTH_URL OS_USERNAME OS_PASSWORD OS_PROJECT_NAME OS_REGION
# replace the IP address with the RHOSP10 keystone server IP
export OS_AUTH_URL=http://10.1.0.100:5000/v2.0
export OS_USERNAME=ocp3ops
```

```
export OS_PASSWORD=password
export OS_TENANT_NAME=ocp3
export OS_REGION=RegionOne
```

To use these values for RHOSP CLI commands, source the file into the environment.



NOTE

Replace the values here with those for the Red Hat OpenStack Platform account.

```
source ./ks_ocp3
```

4.3. RED HAT ENTERPRISE LINUX 7 BASE IMAGE

This procedure expects the instances to be created from the stock Red Hat Enterprise Linux 7.3 cloud image.

The Red Hat Enterprise Linux 7.3 cloud image is in the **rhel-guest-image-7** package that is located in the **rhel-7-server-rh-common-rpms** repository. Install the package on any Red Hat Enterprise Linux system and load the image file into glance. For the purposes of this procedure, the image is named **rhel7**.

Load the RHEL7 image into Glance

```
sudo subscription-manager repos --enable rhel-7-server-rh-common-rpms
sudo yum -y install rhel-guest-image-7
sudo cp /usr/share/rhel-guest-image7/*.qcow2 ~/images
cd ~/images
qemu-img convert -f qcow2 -O raw rhel-guest-image-<ver>.x86_64.qcow2
./rhel7.raw
openstack image create rhel7 \
--disk-format=raw \
--container-format=bare
--public < rhel7.raw
```

Confirm the image has been created as follows:

```
openstack image list
```

4.4. SSH KEY PAIR

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

Generate a Key Pair

```
openstack keypair create ocp3 > ~/ocp3.pem
chmod 600 ~/ocp3.pem
```

NOTE

ssh refuses to use private keys if the file permissions are too open. Be sure to set the permissions on **ocp3.pem** to **600** (user read/write).

4.5. KEYSTONE: INCREASE TOKEN EXPIRATION

NOTE

This step changes the global configuration of the RHOSP service. It must be done by an RHOSP service administrator.

When creating a heat stack, the heat engine gets a **keystone** token to continue taking actions on behalf of the user. By default the token is only good for 1 hour (3600 seconds). The token expires after 1 hour, and the heat engine can no longer continue.

The RHOCP 3 heat stack deployment may take more than one hour. To ensure deployment completion, increase the expiration time on the RHOSP controllers and restart the **httpd** service.

Increase Keystone token expiration

```
# On the RHOSP controller:
sudo sed -i -e 's/#expiration = .*/expiration = 7200/'
/etc/keystone/keystone.conf
sudo systemctl restart httpd
```

4.6. PUBLIC NETWORK

The RHOCP service must be connected to an external public network so that users can reach it. In general, this requires a physical network or at least a network that is otherwise out of the control of the RHOSP service.

The control network is connected to the public network via a router during the network setup.

Each instance is attached to the control network when it is created. Instances that accept direct inbound connections (the bastion, masters and infrastructure nodes) are given floating IP addresses from a pool on the public network.

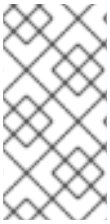
The provisioner must have permission necessary to create and attach the router to the public network and to create floating IPs from the pool on that network.

The floating IP pool must be large enough to accomodate all of the accessable hosts.

4.7. DNS SERVICE

Before beginning the provisioning process, the hostname for developer access and the *wild card* DNS entry for applications must be in place. Both of these should point to the IP address of the load balancer.

Both the IP address and the FQDN for the OpenShift master name and the application wild card entry are known before the provisioning process begins.



NOTE

A *wildcard* DNS entry is a record where, instead of a leading hostname, the record has an asterisk (*), ie. `*.wildcard.example.com`. The IP address associated with a wildcard DNS record returned for any query that matches the suffix. All of the following examples return the same IP address:

- mystuff.wildcard.example.com
- foobar.wildcard.example.com
- random.wildcard.example.com

Each host in the RHOCP service must have a DNS entry for the interfaces on the control and tenant networks. These names are only used internally. Additionally, each host with a floating IP must have a DNS entry with a public name in order to be found by the load-balancer host.

Since the instances are created and addressed dynamically, the name/IP pairs cannot be known before the provisioning process begins. The entries are added using *dynamic DNS* after each instance is created.

Dynamic DNS updates use a tool called *nsupdate* found in the *bind-utils* RPM. The update requests are authorized using a symmetric key found on the DNS host in `/etc/rndc.key`.

The dynamic domains must be created before the provisioning process begins and must be configured to allow dynamic updates. Configuring DNS zones and configuring them to allow dynamic updates is outside the scope of this paper.

Appendix B contains a procedure to create a heat stack running a compliant DNS service.

4.8. HAPROXY SERVICE

A load-balancer provides a single view of the OpenShift master services for the applications. The IP address of the load-balancer must be known before beginning. 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 RHOCP instances have been created and floating IP addresses assigned.

When installing RHOCP using Heat, the load-balancer can be created and configured as part of the heat stack.

4.9. LDAP/AD SERVICE

RHOCP can use an external LDAP or Active Directory service to manage developers from a single user base.

The hostname or IP address of the LDAP service must be known before commencing. This procees requires the *base_dn* and a small set of parameters to allow LDAP authentication by the RHOCP service.

Table 4.1. LDAP credentials example

Parameter	Value	Description
LDAP server	`dc.example.com`	An Active Directory domain controller or LDAP server
User DN	`cn=users,dc=example.dc=com`	The DN of the user database
Preferred Username	`sAMAccountName`	The field to use to find user entries
Bind DN	`cn=openshift,cn=users,dc=example,dc=com`	The user allowed to query
Bind Password	<password>	The password for the user allowed to query

4.10. COLLECTED INFORMATION

In summary, this table lists all of the information and settings that an operator requires to begin a RHOCP deployment on RHOSP.

Table 4.2. Configuration Values

Parameter	Value	Comments
DNS Domain	`ocp3.example.com`	Base domain for all hosts and services
Bastion Hostname	`bastion.ocp3.example.com`	
Load Balancer Hostname	`proxy1.ocp3.example.com`	
LDAP Server Hostname	`ldap1.ocp3.example.com`	
RHOCP Master hostname	`devs.ocp3.example.com`	Developer access to Web UI and API
RHOCP App sub-domain	`*.apps.ocp3.example.com`	All apps get a name under this
RHOSP User	user provided	User account for access to RHOSP
RHOSP Password	user provided	User password for RHOSP account

Parameter	Value	Comments
RHOSP Project	user provided	OSP project name for RHOCP deployment
RHOSP Roles (manual)	_member_	Roles required for the RHOSP user
RHOSP Roles (Heat)	_member_, heat_stack_owner	Roles required for the OSP user
RHEL Base Image	RHEL 7.3 guest	From **rhel-guest-image-7** RPM or download from the Customer Portal
Glance Image Name	rhel7	
SSH Keypair Name	ocp3	
Public Network Name	**public_network**	
Control Network Name	**control-network**	
Control Network Base	**172.18.10.0/24**	
Control Network DNS	X.X.X.X	External DNS IP address
Tenant Network Name	**tenant-network**	
Tenant Network Base	**172.18.20.0/24**	
DNS Update Host	**ns1.ocp3.example.com**	
DNS Update Key	TBD	
RHN Username	**<username>**	
RHN Password	**<password>**	
RHN Pool ID	**<pool id>**	
LDAP Host	**dc.example.com**	
LDAP User DNE	**cn=users,dc=example,dc=com**	
Preferred Username	**sAMAccountName**	

Parameter	Value	Comments
Bind DN	`cn=openshift.cn=users,dc=example,dc=com`	
Bind Password	`<password>`	

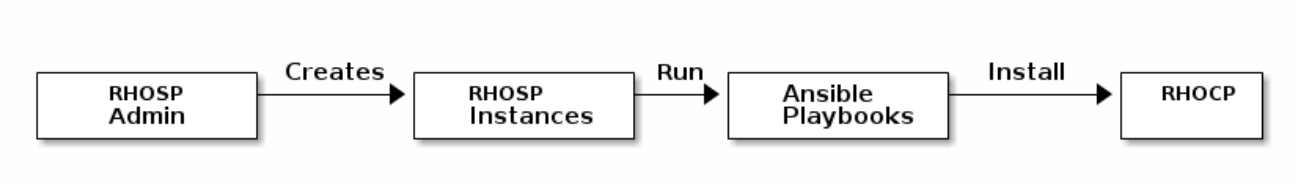
CHAPTER 5. *MANUAL DEPLOYMENT*

The following section is designed to provide a step-by-step of the installation of RHOCP on RHOSP.

NOTE

It is intended as a learning tool in order to deploy and better understand the requirements of real-world installations. However, the manual installation process is not the recommended method for deployment.

In a manual deployment of RHOCP, the workflow is as follows:



It is important to note that a manual deployment does have limitations. These limitations include:

- cannot easily respond to load by auto-scaling
- no infrastructure dashboard or views that show details of instance network components within a RHOCP service.
- requires manual intervention for any changes made in the environment inviting human error in maintenance and monitoring.

5.1. OCP CLI CLIENTS

The manual deployment process starts with a series of steps that are executed using the RHOSP CLI tools. These CLI tools are available in the **`rhel-7-server-openstack-10-rpms`** repository via the package **`python-openstackclient`**. This package installs all the underlying clients that include **nova**, **neutron**, **`cinder`** and **`glance`**. The minimum package version for **`python-openstackclient`** recommended is version **`3.2.1-1`**. This can be verified via:

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

Using the **`openstack`** command, one can create and configure the infrastructure that hosts the RHOCP service.

5.2. SCRIPT FRAGMENTS

The procedures in this section consists of a set of RHOSP CLI script fragments. These fragments are to be executed on an operator's workstation that has the ability to install the **`python-openstackclient`** package. These script fragments allow the operator to create the infrastructure and the instances within the RHOSP environment as required for a successful RHOCP installation. Once the operator can create RHOSP instances, the remainder of the script fragments are run on the *bastion host* to complete the RHOCP deployment using Ansible.

5.2.1. Default Values and Environment Variables

Several of the script fragments set default values for the purpose of providing an example. In real deployments, replace those values with specific values for the target workload and environment.

For example, the **`control network`** script defines the **`OCP3_DNS_NAMESERVER`** variable with a default value. To override the default, set the **`OCP3_DNS_NAMESERVER`** variable in the workstation environment before executing the script fragment.

There are several fragments that execute on one or more of the instances. In order to keep the code size managable, these scripts use a set of variables that define the set of instance names to touch.

5.2.2. Bastion Host Scripts and the Root User

The commands to be executed on the *bastion host* are required to run as the **`root`** user, but the scripts do not have this same requirement. The instance's user is the **`cloud-user`**, a non-priviliged user. When required, the scripts run privileged commands with **`sudo`**.

5.3. CREATE NETWORKS

This section describes the RHOSP network components required by RHOCP and how to create them.

RHOCP depends on three networks.

- *public* network - provides external access controlled by the RHOSP operators
- *control* network - communication between RHOCP instances and service components
- *tenant* network - communications between RHOCP client containers

The public network is a prerequisite requirement prior to continuing with the RHOCP installation. Creation of a public network requires administrative access to the RHOSP environment. Consult the local RHOSP administrator if a public network is not currently available.

Once the public network is accessible, a pool of floating IP addresses is required for RHOSP instances that are to be created, specifically the *bastion host*, *master nodes* and *infrastructure nodes*. Consult the local RHOSP administrator if a pool of floating IP addresses is not currently available.

The control network and the tenant network can be created by a RHOSP user that within their respective RHOSP project thus not requiring any additional RHOSP administrator privileges.

This reference environment refers to the public network name as **`public_network`**

The following two tables describe the internal RHOCP networks.

Table 5.1. Control Network Specification

Name	Value
Network Name	control-network
Subnet Name	control-subnet
Network Address	172.18.10.0

Name	Value
CIDR Mask	/24
DNS Nameserver	X.X.X.X
Router Name	control-router

Table 5.2. Tenant Network Specification

Name	Value
Network Name	tenant-network
Subnet Name	tenant-subnet
Network address	172.18.20.0
CIDR Mask	/24
Flags	--no-gateway

Create the *control* network as indicated below. Update the default values as required.

NOTE

The control network is tied to the **`public network`** through a router labeled **`control-router`**.

Create Control Network

```
#!/bin/sh
OCP3_DNS_NAMESERVER=${OCP3_DNS_NAMESERVER:-8.8.8.8}
PUBLIC_NETWORK=${PUBLIC_NETWORK:-public_network}
CONTROL_SUBNET_CIDR=${CONTROL_SUBNET_CIDR:-172.18.10.0/24}

openstack network create control-network
openstack subnet create --network control-network --subnet-range
${CONTROL_SUBNET_CIDR} \
--dns-nameserver ${OCP3_DNS_NAMESERVER} control-subnet
openstack router create control-router
openstack router add subnet control-router control-subnet
neutron router-gateway-set control-router ${PUBLIC_NETWORK}
```

In a similar fashion, create the *tenant* network. This network is for internal use only and does not have a gateway to connect to other networks.

Create Tenant Network

```
#!/bin/sh
TENANT_SUBNET_CIDR=${TENANT_SUBNET_CIDR:-172.18.20.0/24}

openstack network create tenant-network
openstack subnet create --network tenant-network \
  --subnet-range ${TENANT_SUBNET_CIDR} --gateway none tenant-subnet
```

At this point, three networks exist. Two of those networks created by the operator (*control* and *tenant* network), while the *public* network is provided by the RHOSP administrator. The *control* network is bound to a router that provides external access.

Verify control and tenant networks

```
openstack network list
+--------------------------------------+-----------------+-----------
-------------------------+
| ID                                   | Name            | Subnets
|
+--------------------------------------+-----------------+-----------
-------------------------+
| c913bfd7-94e9-4caa-abe5-df9dcd0c0eab | control-network | e804600d-67fc-
40bb-a7a8-2f1ce49a81e6 |
| 53377111-4f85-401e-af9d-5166a4be7f99 | tenant-network  | 830845ca-f35c-
44d9-be90-abe532a32cbe |
| 957dae95-5b93-44aa-a2c3-37eb16c1bd2f | public_network  | 4d07867c-34d8-
4a7c-84e3-fdc485de66be |
+--------------------------------------+-----------------+-----------
-------------------------+
```

5.4. NETWORK SECURITY GROUPS

RHOSP 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 RHOSP.

The details of the communication ports and their purposes are defined in the Prerequisites section of the OpenShift Container Platform Installation and Configuration Guide.

5.4.1. Required Network Services

Each of the instance types in the RHOCP service communicate using a well defined set of network ports. This procedure includes defining a network *security_group* for each instance type.

All of these instances allow **icmp** (ping) packets to test that the hosts are booted and connected to their respective networks.

All the hosts accept inbound **ssh** connections for provisioning by Ansible and for ops access during normal operations.

5.4.2. Bastion Host Security Group

The *bastion host* only needs to allow inbound **ssh**. This host exists to give operators and automation a stable base to monitor and manage the rest of the services.

All other security groups accept **ssh** inbound from this security group.

The following commands create and define the bastion network security group.

Table 5.3. Bastion Security Group TCP ports

Port/Protocol	Service	Purpose
22/TCP	SSH	Secure shell login

Create Bastion Security Group

```
#!/bin/sh
openstack security group create bastion-sg
openstack security group rule create --ingress --protocol icmp bastion-sg
openstack security group rule create --protocol tcp \
--dst-port 22 bastion-sg
#Verification of security group
openstack security group show bastion-sg
```

5.4.3. Master Host Security Group

The RHOCP master service requires the most complex network access controls.

In addition to the secure **http** ports used by the developers, these hosts contain the **etcd** servers that form the cluster.

These commands create and define master host network security group.

Table 5.4. Master Host Security Group Ports

Port/Protocol	Service	Purpose
22/TCP	SSH	secure shell login
53/TCP	DNS	internal name services (pre 3.2)
53/UDP	DNS	internal name services (pre 3.2)
2379/TCP	etcd	client → server connections
2380/TCP	etcd	server → server cluster communications
4789/UDP	SDN	pod to pod communications
8053/TCP	DNS	internal name services (3.2+)

Port/Protocol	Service	Purpose
8053/UDP	DNS	internal name services (3.2+)
8443/TCP	HTTPS	Master WebUI and API
10250/TCP	kubernetes	kubelet communications
24224/TCP	fluentd	Docker logging

Create Master Security Group

```
#!/bin/sh
openstack security group create master-sg
openstack security group rule create --protocol icmp master-sg
neutron security-group-rule-create master-sg \
 --protocol tcp --port-range-min 22 --port-range-max 22 \
 --remote-group-id bastion-sg


neutron security-group-rule-create master-sg \
 --protocol tcp --port-range-min 2380 --port-range-max 2380 \
 --remote-group-id master-sg

for PORT in 53 2379 2380 8053 8443 10250 24224
do
  openstack security group rule create --protocol tcp --dst-port $PORT
master-sg
done

for PORT in 53 4789 8053 24224
do
  openstack security group rule create --protocol udp --dst-port $PORT
master-sg
done
```

5.4.4. Node Host Security Group

The node hosts execute the containers within the RHOCP service. The nodes are broken into two sets. The first set runs the Red Hat OpenShift Container Platform infrastructure, the Red Hat OpenShift Container Platform router, and a local docker registry. The second set runs the developer applications.

5.4.4.1. 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 5.5. Infrastructure Node Security Group Ports

Port/Protocol	Services	Purpose
22/TCP	SSH	secure shell login
80/TCP	HTTP	cleartext application web traffic
443/TCP	HTTPS	encrypted application web traffic
4789/UDP	SDN	pod to pod communications
10250/TCP	kubernetes	kubelet communications

Infrastructure Node Security Group

```
#!/bin/sh
openstack security group create infra-node-sg
openstack security group rule create --protocol icmp infra-node-sg
neutron security-group-rule-create infra-node-sg \
  --protocol tcp --port-range-min 22 --port-range-max 22 \
  --remote-group-id bastion-sg

for PORT in 80 443 10250 4789
do
 openstack security group rule create --protocol tcp --dst-port $PORT
infra-node-sg
done
```

5.4.4.2. App Node Security Group

The application nodes are only accept traffic from the *control* and *tenant* network. *app* nodes only accept the **etcd** control connections and the **flannel** SDN traffic for container communications.

Table 5.6. Application Node Security Group Ports

Port/Protocol	Services	Purpose
22/TCP	SSH	secure shell login
4789/UDP	SDN	pod to pod communications
10250/TCP	kubernetes	kubelet communications

Create App Node Security Group

```
#!/bin/sh
openstack security group create app-node-sg
openstack security group rule create --protocol icmp app-node-sg
neutron security-group-rule-create app-node-sg \
     --protocol tcp --port-range-min 22 --port-range-max 22 \
```

```
    --remote-group-id bastion-sg
openstack security group rule create --protocol tcp --dst-port 10250 app-
node-sg
openstack security group rule create --protocol udp --dst-port 4789 app-
node-sg
```

5.5. HOST INSTANCES

With the networks established, begin building the RHOSP instances to build the RHOCP environment.

This reference environment consists of:

- 1 *bastion host*
- 3 *master nodes*
- 2 *infrastructure nodes* (infra nodes)
- 3 *application nodes* (app nodes)

With three exceptions, the specifications all of the instances in this service are identical. Each instance has a unique name and each of the node hosts have external storage. Each one is assigned a security group based on its function. The common elements are presented first followed by the differences.

NOTE

The commands to create a new instance are only presented once. Repeat for each instance, altering the name for each instance.

5.5.1. Host Names, DNS and cloud-init

RHOCP uses the host names 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, RHOSP 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 be sufficient if RHOSP populated a DNS service with these names thus allowing each instance to find the IP addresses by name.

Using **`.novalocal`** domain name requires creating a zone in the external DNS service named **`.novalocal`**. However, since the RHOSP instance names are unique only within a project, this risks name collisions with other projects. Instead, create a subdomain for the *control* and *tenant* networks under the project domain, i.e. **`ocp3.example.com`**.

Table 5.7. Subdomains for RHOCP internal networks

Domain name	Description
`control.ocp3.example.com`	all interfaces on the control network
`tenant.ocp3.example.com`	all interfaces on the tenant network

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

Table 5.8. Sample FQDNs

Full Name	Description
master-0.control.ocp3.example.com	Name of the control network interface on the master-0 instance
master-0.tenant.ocp3.example.com	Name of the tenant network interface on the master-0 instance
master-0.ocp3.example.com	Name of the floating IP for the master-0 instance

RHOSP provides a way for users to pass in information to be applied when an instance boots. The **--user-data** switch to **openstack server create** 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. It queries a standard URL for the **user-data** file and processes the contents to initialize the OS.

This deployment process uses **cloud-init** to control three values:

1. hostname
2. fully qualified domain name (FQDN)
3. enable **sudo** via **ssh**

The **user-data** file is a multi-part MIME file with two parts. One is the *cloud-data* section that sets the hostname and FQDN. The other is a short one line shell script.

The **user-data** is placed in files named **<hostname>.yaml** where '<hostname>' is the name of the instance. The script below generates user-data for all of the instances.

Create user-data directory

```
mkdir -p ~/user-data
```

Generate user-data for instance hostnames

```
#!/bin/sh

#set DOMAIN and SUBDOMAIN to override
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
CONTROL_DOMAIN=${CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}

BASTION="bastion"
MASTERS=$(for M in $(seq 0 $(($MASTER_COUNT-1))) ; do echo master-$M ;
done)
INFRA_NODES=$(for I in $(seq 0 $(($INFRA_NODE_COUNT-1))) ; do echo infra-
node-$I ; done)
APP_NODES=$(for A in $(seq 0 $(($APP_NODE_COUNT-1))) ; do echo app-node-$A
```

```
; done)
ALL_NODES="$INFRA_NODES $APP_NODES"
ALL_HOSTS="$BASTION $MASTERS $ALL_NODES"

function generate_userdata_mime() {
  cat <<EOF
From nobody Fri Oct  7 17:05:36 2016
Content-Type: multipart/mixed;
boundary="===============6355019966770068462=="
MIME-Version: 1.0

--===============6355019966770068462==
MIME-Version: 1.0
Content-Type: text/cloud-config; charset="us-ascii"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="$1.yaml"

#cloud-config
hostname: $1
fqdn: $1.$2

write_files:
  - path: "/etc/sysconfig/network-scripts/ifcfg-eth1"
    permissions: "0644"
    owner: "root"
    content: |
      DEVICE=eth1
      TYPE=Ethernet
      BOOTPROTO=dhcp
      ONBOOT=yes
      DEFTROUTE=no
      PEERDNS=no

--===============6355019966770068462==
MIME-Version: 1.0
Content-Type: text/x-shellscript; charset="us-ascii"
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="allow-sudo-ssh.sh"

#!/bin/sh
sed -i "/requiretty/s/^/#/" /etc/sudoers

--===============6355019966770068462==--

EOF
}

[ -d user-data ] || mkdir -p user-data
for HOST in $ALL_HOSTS
do
  generate_userdata_mime ${HOST} ${CONTROL_DOMAIN} > user-
data/${HOST}.yaml
done
```

5.5.2. Create Bastion Host Instance

This script creates a single instance to access and control all the nodes within the RHOCP environment. Set the **DOMAIN** environment variable before invoking these scripts to customize the target domain for the entire deployment.

NOTE

The script requires the operator to know the image name, key name and the network ids of the *control* and *tenant* network. The following commands provide that information.

```
$ openstack image list
$ openstack keypair list
$ openstack network list
```

Create Bastion Host instance

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
IMAGE=${IMAGE:-rhel7}
FLAVOR=${FLAVOR:-m1.small}
OCP3_KEY_NAME=${OCP3_KEY_NAME:-ocp3}
netid1=$(openstack network list | awk "/control-network/ { print \$2 }")
netid2=$(openstack network list | awk "/tenant-network/ { print \$2 }")
openstack server create --flavor ${FLAVOR} --image ${IMAGE} \
--key-name ${OCP3_KEY_NAME} \
--nic net-id=$netid1 \
--nic net-id=$netid2 \
--security-group bastion-sg --user-data=user-data/bastion.yaml \
bastion.${OCP3_CONTROL_DOMAIN}
```

NOTE

This reference environment uses *rhel7* as the image name, *ocp3* as the key name, and the network ids found within **`openstack network list`**.

5.5.3. Create Master Host instances

The following script creates three RHOCP master instances.

Create Master Host instances

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
OCP3_KEY_NAME=${OCP3_KEY_NAME:-ocp3}
IMAGE=${IMAGE:-rhel7}
FLAVOR=${FLAVOR:-m1.small}
MASTER_COUNT=${MASTER_COUNT:-3}
netid1=$(openstack network list | awk "/control-network/ { print \$2 }")
netid2=$(openstack network list | awk "/tenant-network/ { print \$2 }")
for HOSTNUM in $(seq 0 $(($MASTER_COUNT-1))) ; do
    openstack server create --flavor ${FLAVOR} --image ${IMAGE} \
      --key-name ${OCP3_KEY_NAME} \
```

```
      --nic net-id=$netid1 --nic net-id=$netid2 \
      --security-group master-sg --user-data=user-data/master-
${HOSTNUM}.yaml \
      master-${HOSTNUM}.${OCP3_CONTROL_DOMAIN}
done
```

NOTE

This reference environment uses *rhel7* as the image name, *ocp3* as the key name, and the network ids found within **`openstack network list`**.

5.5.4. Cinder Volumes for `/var/lib/docker`

All of the node instances need a **`cinder`** volume. These are mounted into the instance to provide additional space for **`docker`** image and container storage.

Create a volume for each node instance:

Create Node Volumes

```
#!/bin/sh
VOLUME_SIZE=${VOLUME_SIZE:-8}
BASTION="bastion"
INFRA_NODE_COUNT=${INFRA_NODE_COUNT:-2}
APP_NODE_COUNT=${APP_NODE_COUNT:-3}

INFRA_NODES=$(for I in $(seq 0 $(($INFRA_NODE_COUNT-1))) ; do echo infra-
node-$I ; done)
APP_NODES=$(for I in $(seq 0 $(($APP_NODE_COUNT-1))) ; do echo app-node-$I
; done)
ALL_NODES="$INFRA_NODES $APP_NODES"

for NODE in $ALL_NODES ; do
    cinder create --name ${NODE}-docker ${VOLUME_SIZE}
done

openstack volume list
```

5.5.5. Create the Node instances

The **`openstack server create`** command has an argument to mount **`cinder`** volumes into new instances. This argument requires the volume ID rather than the name. The following function returns the volume ID given a volume name.

The script below creates two *infrastructure nodes*, mounting the matching **`cinder`** volume on device **vdb**.

Create the Infrastructure Node instances

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
OCP3_KEY_NAME=${OCP3_KEY_NAME:-ocp3}
IMAGE=${IMAGE:-rhel7}
```

```
FLAVOR=${FLAVOR:-m1.small}
netid1=$(openstack network list | awk "/control-network/ { print \$2 }")
netid2=$(openstack network list | awk "/tenant-network/ { print \$2 }")
INFRA_NODE_COUNT=${INFRA_NODE_COUNT:-2}
for HOSTNUM in $(seq 0 $(($INFRA_NODE_COUNT-1))) ; do
    HOSTNAME=infra-node-$HOSTNUM
    VOLUMEID=$(cinder list | awk "/${HOSTNAME}-docker/ { print \$2 }")
    openstack server create --flavor ${FLAVOR} --image ${IMAGE} \
       --key-name ${OCP3_KEY_NAME} \
       --nic net-id=$netid1 --nic net-id=$netid2 \
       --security-group infra-node-sg --user-data=user-
data/${HOSTNAME}.yaml \
       --block-device-mapping vdb=${VOLUMEID} \
       ${HOSTNAME}.${OCP3_CONTROL_DOMAIN}
done
```

The script below creates three *application nodes*, mounting the matching Cinder volume on device **vdb**.

Create the App Node instances

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
OCP3_KEY_NAME=${OCP3_KEY_NAME:-ocp3}
IMAGE=${IMAGE:-rhel7}
FLAVOR=${FLAVOR:-m1.small}
netid1=$(openstack network list | awk "/control-network/ { print \$2 }")
netid2=$(openstack network list | awk "/tenant-network/ { print \$2 }")
APP_NODE_COUNT=${APP_NODE_COUNT:-3}
for HOSTNUM in $(seq 0 $(($APP_NODE_COUNT-1))) ; do
    HOSTNAME=app-node-${HOSTNUM}
    VOLUMEID=$(cinder list | awk "/${HOSTNAME}-docker/ { print \$2 }")
    openstack server create --flavor ${FLAVOR} --image ${IMAGE} \
        --key-name ${OCP3_KEY_NAME} \
        --nic net-id=$netid1 --nic net-id=$netid2 \
        --security-group app-node-sg --user-data=user-
data/${HOSTNAME}.yaml \
        --block-device-mapping vdb=${VOLUMEID} \
        ${HOSTNAME}.${OCP3_CONTROL_DOMAIN}
done
```

NOTE

This reference environment uses *rhel7* as the image name, *ocp3* as the key name, and the network ids found within **`openstack network list`**.

5.5.6. Disable Port Security on Nodes

In this installation **`flannel`** provides the software defined network (SDN). Current versions of **`neutron`** enforce *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. Port security must be disabled on the ports that carry **`flannel`** traffic.

This configuration runs `flannel` on a private network that is bound to the *eth1* device on the node instances.

Disable Port Security on Tenant Network Ports

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
TENANT_NETWORK=${TENANT_NETWORK:-tenant-network}

MASTER_COUNT=${MASTER_COUNT:-3}
INFRA_NODE_COUNT=${INFRA_NODE_COUNT:-2}
APP_NODE_COUNT=${APP_NODE_COUNT:-3}

MASTERS=$(for M in $(seq 0 $(($MASTER_COUNT-1))) ; do echo master-$M ;
done)
INFRA_NODES=$(for I in $(seq 0 $(($INFRA_NODE_COUNT-1))) ; do echo infra-
node-$I ; done)
APP_NODES=$(for A in $(seq 0 $(($APP_NODE_COUNT-1))) ; do echo app-node-$A
; done)

function tenant_ip() {
  # HOSTNAME=$1
  nova show ${1} | grep ${TENANT_NETWORK} | cut -d\| -f3 | cut -d, -f1 | tr
-d ' '
}

function port_id_by_ip() {
  # IP=$1
  neutron port-list --field id --field fixed_ips | grep $1 | cut -d' ' -f2
}

for NAME in $MASTERS $INFRA_NODES $APP_NODES
do
  TENANT_IP=$(tenant_ip ${NAME}.${OCP3_CONTROL_DOMAIN})
  PORT_ID=$(port_id_by_ip $TENANT_IP)
  neutron port-update $PORT_ID --no-security-groups --port-security-
enabled=False
done
```

5.5.7. Adding Floating IP addresses

The *bastion*, *master* and *infrastructure* nodes all need to be accessible from outside the *control* network. An IP address on the *public* network is required.

Create and Assign Floating IP Addresses

```
#!/bin/sh
OCP3_DOMAIN=${OCP3_DOMAIN:-ocp3.example.com}
OCP3_CONTROL_DOMAIN=${OCP3_CONTROL_DOMAIN:-control.${OCP3_DOMAIN}}
PUBLIC_NETWORK=${PUBLIC_NETWORK:-public_network}
MASTER_COUNT=${MASTER_COUNT:-3}
INFRA_NODE_COUNT=${INFRA_NODE_COUNT:-2}
APP_NODE_COUNT=${APP_NODE_COUNT:3}
```

```
BASTION="bastion"
MASTERS=$(for M in $(seq 0 $(($MASTER_COUNT-1))) ; do echo master-$M ;
done)
INFRA_NODES=$(for I in $(seq 0 $(($INFRA_NODE_COUNT-1))) ; do echo infra-
node-$I ; done)
for HOST in $BASTION $MASTERS $INFRA_NODES
do
  openstack floating ip create ${PUBLIC_NETWORK}
  FLOATING_IP=$(openstack floating ip list | awk "/None/ { print \$4 }")
  openstack server add floating ip ${HOST}.${OCP3_CONTROL_DOMAIN}
${FLOATING_IP}
done
```

5.6. PUBLISHING HOST IP ADDRESSES

While end users only interact with the RHOCP services through the public interfaces, it is important to assign a name to each interface to ensure each RHOCP component is able to identify and communicate across the environment. The simplest way to accomplish this is to update the DNS entries for each interface within each RHOSP instance.

All of the RHOSP instances connect to the **control-network** and the **tenant-network**. The *bastion*, *master* and *infra* nodes contain floating IPs that are assigned on the **public_network** and must be published in a zone. This reference environment uses the **ocp3.example.com** zone. The IP addresses on the *control* and *tenant* networks reside in a subdomain named for these networks. For example, DNS updates require the IP address of the primary DNS server for the zone and a key to enable updates.

When using BIND (Berkeley Internet Name Domain), the DNS server included in Red Hat Enterprise Linux, the key service it provides is the **named** program. When using the **named** service, the key is located in the **/etc/rndc.key** file. Create a copy of the contents of the file to a local file such as **ocp3_dns.key**. In order to implement updates, the **bind-utils** RPM must be installed. This contains the **/usr/bin/nsupdate** binary that implements dynamic updates.

Prior to making DNS updates, acquire the hostname, FQDN and floating IP for each instance.

On a server that has access to the key, i.e. **/etc/rndc.key**, perform the following for all nodes:

Change the IP address of a DNS name

```
nsupdate -k ocp3_dns.key <<EOF
server <dns-server-ip>
zone ocp3.example.com
update add <FQDN> 3600 A <ip-address>
.
.
.
update add <FQDN> 3600 A <ip-address>
send
quit
EOF
```

An example of adding the public, tenant and control networks of particular node.

Adding Master Node entries

```
nsupdate -k ocp3_dns.key <<EOF
server <dns-server-ip>
zone ocp3.example.com
update add master-0.control.ocp3.example.com 3600 A 172.x.10.7
update add master-0.ocp3.example.com 3600 A 10.x.0.166
send
quit
EOF
```

Table 5.9. Table Example Hostname/IP Address Mappings

ip-address	FQDN	Comment
10.x.0.151	**bastion.ocp3.example.com**	Provides deployement and operations access
10.x.0.166	**master-0.ocp3.example.com**	load balanced developer access
10.x.0.167	**master-1.ocp3.example.com**	load balanced developer access
10.x.0.168	**master-2.ocp3.example.com**	load balanced developer access
10.x.0.164	**infra-node-0.ocp3.example.com**	runs OpenShift router
10.x.0.164	**infra-node-1.ocp3.example.com**	runs OpenShift router
10.x.0.164	**infra-node-2.ocp3.example.com**	runs OpenShift router

Sample Host Table - DNS entries

```
10.19.114.153 bastion.ocp3.example.com bastion
172.18.10.6 bastion.control.example.com

10.19.114.114 master-0.ocp3.example.com
172.18.10.7 master-0.control.example.com

10.19.114.115 master-1.ocp3.example.com
172.18.10.8 master-1.control.example.com

10.19.114.116 master-2.ocp3.example.com
172.18.10.9 master-2.control.example.com

10.19.114.120 infra-node-0.ocp3.example.com
172.18.10.10 infra-node-0.control.example.com

10.19.114.121 infra-node-1.ocp3.example.com
```

```
172.18.10.11 infra-node-1.control.example.com

172.18.10.17 app-node-0.control.example.com
172.18.10.18 app-node-1.control.example.com
172.18.10.19 app-node-2.control.example.com
```

5.7. LOAD-BALANCER

This reference environment consists of a standalone server running a HAProxy instance with the following configuration. Please include the floating IPs for the *master* and *infra* nodes to the load balancer.

The file that follows is a sample configuration for HAProxy.

/etc/haproxy/haproxy.cfg

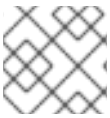
```
global
    log         127.0.0.1 local2

    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
    server infra-node-0.ocp3.example.com 10.19.114.120:80 check
    server infra-node-1.ocp3.example.com 10.19.114.121:80 check
    server infra-node-2.ocp3.example.com 10.19.114.122:80 check

frontend  main *:443
    default_backend             router443

backend router443
    balance source
    mode tcp
    server infra-node-0.ocp3.example.com 10.19.114.120:443 check
    server infra-node-1.ocp3.example.com 10.19.114.121:443 check
    server infra-node-2.ocp3.example.com 10.19.114.122:443 check


frontend  main *:8443
    default_backend             mgmt8443

backend mgmt8443
    balance source
    mode tcp
    server master-0.ocp3.example.com 10.19.114.114:8443 check
    server master-1.ocp3.example.com 10.19.114.115:8443 check
    server master-2.ocp3.example.com 10.19.114.116:8443 check
```

5.8. HOST PREPARATION

With access to the RHOSP instances, this section turns its focus to running the **`openshift-ansible`** playbooks. For a successful deployment of RHOCP, the RHOSP instances require additional configuration.

The additional configuration on the RHOCP nodes is all done via the *bastion* host. The *bastion* host acts as a jump server to access all RHOCP nodes via an **`ssh`** session using the control network.



NOTE

The *bastion* host is the only node that is accessible via **`ssh`** from outside the cluster.

THE RHOCP *master* nodes require software updates and accessibility to install **`ansible`**. The *infra* and *app* nodes require access to multiple software repositories, and a **`cinder`** volume i.e. /dev/vdb, to store the Docker container files.

Bastion Customization

- Register for Software Updates
- Install **`openshift-ansible-playbooks`** RPM package
- Copy the private key to the **`cloud-init`** user

Common Customization (*master*, *infra*, *app* nodes)

- Enable **sudo** via **ssh**
- Register for software repositories
- Enable **eth1** (tenant network)

***Infra* and *App* Node Customization**

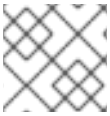
- Mount **cinder** volume
- Install **docker** RPM

5.8.1. Logging into the Bastion Host

Log into the *bastion* host as follows:

Log into Bastion Host

```
ssh -i ocp3.pem  cloud-user@bastion.control.ocp3.example.com
```



NOTE

This example assumes the private key is in a file named **ocp3.pem**.

5.8.1.1. Register for Software Updates

Red Hat software installation and updates requires a valid subscription and registration. Access to the RHOCP and RHOSP repositories may require a specific subscription pool.

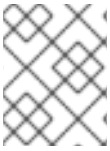
Register For Software Updates (RHN)

```
#!/bin/sh
# Set RHN_USERNAME, RHN_PASSWORD RHN_POOL_ID for your environment
sudo subscription-manager register \
  --username $RHN_USERNAME \
  --password $RHN_PASSWORD
sudo subscription-manager subscribe --pool $RHN_POOL_ID
```

Enable the standard server repositories and RHOCP software specific repository as follows:

Enable Required Repositories

```
#!/bin/sh
OCP3_VERSION=${OCP3_VERSION:-3.4}
sudo subscription-manager repos --disable="*"
sudo subscription-manager repos \
  --enable=rhel-7-server-rpms \
  --enable=rhel-7-server-extras-rpms \
  --enable=rhel-7-server-optional-rpms \
  --enable=rhel-7-server-ose-${OCP3_VERSION}-rpms
```



NOTE

The RHOCP software is in a specific product repository. The version this reference environment uses is version 3.4.

5.8.1.2. Ansible Preperation

- Install **openshift-ansible-playbooks**

5.8.1.3. Install Ansible and OpenShift Ansible

Installing the Ansible components is requires installing the **openshift-ansible-playbooks** RPM. This package installs all of the dependencies required.

Install openshift-ansible-playbooks

```
#!/bin/sh
sudo yum install -y openshift-ansible-playbooks
```

5.8.2. Preparing to Access the RHOCP Instances

The *bastion* host requires access to all the RHOCP instances as the **cloud-user**. This is required to ensure that Ansible can log into each instance and make the appropriate changes when running the Ansible playbook. The following sections show how to enable access to all the RHOCP instances.

The first step is to install the RHOSP **ssh** key for the **cloud-user** account on the *bastion* host. The second step is to enable the **ssh** command to call **sudo** without opening a shell. This ensures Ansible to execute commands as **root** on all the RHOCP instances directly from the *bastion* host.

5.8.2.1. Set the Host Key On the Bastion

A copy of the private key is needed on the *bastion* host to allow **ssh** access from the *bastion* host to the RHOCP instances.

On the system with the private key file, copy the private key to the *bastion* host and place it in **.ssh/id_rsa**. An example is shown below.

Copy the SSH key to the bastion host

```
scp -i ocp3.pem ocp3.pem cloud-
user@bastion.control.ocp3.example.com:.ssh/id_rsa
```

ssh won't accept key files that are readable by other users. The key file permissions should only have read and write access to the owner. Log into the *bastion* host, and change the key file's permissions.

Restrict access to the SSH key on the bastion host

```
chmod 600 ~/.ssh/id_rsa
```

5.8.3. Instance Types and Environment Variables

The RHOCP instances are separated into three categories: *master*, *infrastructure* and *application* nodes. The following sections run a series of short script fragments that contain environment variables to allow for customization and to configure multiple hosts.

Below are the variables that are found in the script fragments and the conventions for their use.

Host Class Environment Variables

```
MASTERS="master-0 master-1 master-2"
INFRA_NODES="infra-node-0 infra-node-1"
APP_NODES="app-node-0 app-node-1 app-node-2"
ALL_HOSTS="$MASTERS $INFRA_NODES $APP_NODES"
```

5.8.3.1. Log into RHOCP Instances

The RHOCP service instances only allow inbound **ssh** from the *bastion* host. The connections are made over the *control* network interface.

In the DNS entries for each instance, the *control* network IP address is placed in the *.control* sub-domain in this reference environment. This means that the *control* network name for an instance is **<hostname>.control.ocp3.example.com**.

Once the hostname and FQDN DNS entries are defined, **cloud-init** puts the *control* network subdomain in the DNS search path in **/etc/resolv.conf**. This enables the *bastion* host to log into any instance using its **hostname**.

```
ssh master-0
```

NOTE

Log in to each instance to verify connectivity and to accept the host key.

5.8.3.2. Enable sudo via ssh

Attempting to run **sudo** commands via **ssh** by default is forbidden. However, it is necessary to relax this restriction to enable the Ansible playbook to run all the commands directly from the *bastion* host.

For each *master*, *infrastructure* and *app* node, log in and modify the **/etc/sudoers** file:

From the *bastion* host as the **cloud-user**,

Enable sudo via ssh

```
ssh <node-name>
sudo cp /etc/sudoers{,.orig}
sudo sed -i "/requiretty/s/^/#/" /etc/sudoers
```

To ensure the execution of command as the **root** user on the RHOCP instances is possible, run the following command on the *bastion* host for each RHOCP instance. An example below is shown on attempting to **ssh** to the **master-0** node while running the **id** command using **sudo**.

Demonstrate sudo via ssh

```
ssh master-0 sudo id
uid=0(root) gid=0(root) groups=0(root)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

5.8.4. Common Configuration Steps

The following steps within these subsections are to be executed on all of the RHOCP instances.

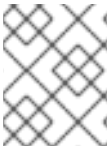
5.8.4.1. Enable Software Repositories

Register all of the RHOCP instances for software updates before attempting to install and update the environments.

Register All instances for Software Updates

```
#!/bin/sh
RHN_USERNAME=${RHN_USERNAME:-changeme}
RHN_PASSWORD=${RHN_PASSWORD:-changeme}
RHN_POOL_ID=${RHN_POOL_ID:-changeme}

for H in $ALL_HOSTS
do
  ssh $H sudo subscription-manager register \
      --username ${RHN_USERNAME} \
      --password ${RHN_PASSWORD}
  ssh $H sudo subscription-manager attach \
      --pool ${RHN_POOL_ID}
done
```

Disable the default enabled repositories, and re-enable the minimal standard set of repositories.



NOTE

The RHOCP software repository is set to the version that is used within this reference environment, version 3.4.

Enable Standard Software Repositories

```
#!/bin/sh
OCP3_VERSION=${OCP3_VERSION:-3.4}

for H in $ALL_HOSTS
do
  ssh $H sudo subscription-manager repos --disable="*"
  ssh $H sudo subscription-manager repos \
      --enable="rhel-7-server-rpms" \
      --enable="rhel-7-server-extras-rpms" \
      --enable="rhel-7-server-optional-rpms" \
      --enable="rhel-7-server-ose-${OCP3_VERSION}-rpms"
done
```

For all nodes, install the following base packages:

Install Base Packages

```
#!/bin/sh
for H in $ALL_HOSTS
do
  ssh $H sudo yum install -y \
    wget git net-tools bind-utils iptables-services \
    bridge-utils bash-completion atomic-openshift-excluder \
    atomic-openshift-docker-excluder
done
```

5.8.5. Node Configuration Steps

The *infrastructure* and *application* nodes require configuration updates not needed on the *bastion* nor the *master* nodes.

The *infrastructure* and *application* nodes must be able to communicate over a private network, and mount an external volume to hold the Docker container and image storage.

5.8.5.1. Enable Tenant Network Interface

The containers within the RHOCP service communicate over a private back-end network, called the *tenant network*. The *infrastructure* and *application* nodes are configured with a second interface , **`eth1`**, to carry this traffic. RHOCP uses **`flannel`** to route the traffic within and between nodes.

The file below configures that second interface on each node. RHOSP assigns each node an address on that subnet and provides a DHCP address to the hosts.

This file is copied to each instance and then the interface is started. When RHOCP is installed, the configuration file must specify this interface name for **`flannel`**.

ifcfg-eth1

```
# /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE="eth1"
BOOTPROTO="dhcp"
BOOTPROTOv6="dhcp"
ONBOOT="yes"
TYPE="Ethernet"
USERCTL="yes"
PEERDNS="no"
IPV6INIT="yes"
PERSISTENT_DHCLIENT="1"
```

The script below copies the **`ifcfg-eth1`** file to the hosts and brings interface up.

On the *bastion* host,

Copy and Install eth1 configuration

```
#!/bin/sh
for H in $ALL_HOSTS ; do
    scp ifcfg-eth1 $H:
```

```
    ssh $H sudo cp ifcfg-eth1 /etc/sysconfig/network-scripts
    ssh $H sudo ifup eth1
done
```

5.8.5.2. Add Docker Storage

The *infrastructure* and *application* nodes each get an external **cinder** volume that provides local space for the Docker images and live containers. Docker must be installed before the storage is configured and mounted.

5.8.5.3. Install Docker

Install the **docker** RPM package but do not enable on all the *infrastructure* and *app* nodes.

Install Docker

```
#!/bin/sh
for H in $INFRA_NODES $APP_NODES
do
  ssh $H sudo yum install -y docker
done
```

5.8.5.4. Configure and Enable Instance Metadata Daemon

The instance Metadata daemon discovers and maps the **cinder** volume into the device file system. Install and configure the instance Metadata daemon as follows.

Enable instance updates

```
#!/bin/sh
for H in $INFRA_NODES $APP_NODES
do
  ssh $H sudo systemctl enable lvm2-lvmetad
  ssh $H sudo systemctl start lvm2-lvmetad
done
```

5.8.5.5. Set the Docker Storage Location

The Docker storage configuration is set by a file named **/etc/sysconfig/docker-storage-setup**. This file specifies that the Docker storage is placed on the device **/dev/vdb** and creates an instance volume group called **docker-vg**.

docker-storage-setup

```
DEVS=/dev/vdb
VG=docker-vg
```

This script fragment copies the storage configuration to the *infrastructure* and *app* nodes then executes the Docker script to configure the storage.

Copy and Install Storage Setup Config

```
#!/bin/sh
for H in $INFRA_NODES $APP_NODES
do
  scp docker-storage-setup $H:
  ssh $H sudo cp ./docker-storage-setup /etc/sysconfig
  ssh $H sudo /usr/bin/docker-storage-setup
done
```

5.8.5.6. Enable and Start Docker

This script fragment starts and enables the **docker** service on all *infrastructure* and *app* nodes.

Enable and Start Docker

```
#!/bin/sh
for H in $INFRA_NODES $APP_NODES
do
  ssh $H sudo systemctl enable docker
  ssh $H sudo systemctl start docker
done
```

5.9. DEPLOY RED HAT OPENSHIFT CONTAINER PLATFORM

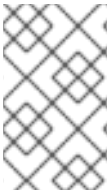
The OpenShift Ansible playbooks are separated into three input files.

- **inventory**
- **group_vars/OSEv3.yml**
- **ansible.cfg**

The *inventory* is an **ini** formatted set of host names, and groupings of hosts with common configuration attributes. It defines the configuration variables for each group and individual host. The user defines a set of groups at the top level and fills out a section for each group defined. Each group section list the hosts that are members of that group.

The **OSEv3.yml** file is a **yaml** formatted data file. It defines a structured set of key/value pairs that are used to configure all of the RHOSP instances.

The **ansible.cfg** file is an **ini** formatted file that defines Ansible run-time customizations.



NOTE

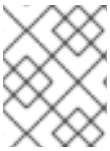
It is recommended to create a directory, i.e. **mkdir -p ~/ansible_files/group_vars**, within the *bastion* host as the **cloud-user** that contains all three files including the **group_vars** directory.

5.9.1. Ansible inventory File

The **inventory** file defines the set of servers for Ansible to configure. The servers are grouped into classes, and the members of a given class get the same configuration.

Ansible control file: inventory

```
[OSEv3:children]
masters
nodes
etcd

[masters]
master-[0:2].control.ocp3.example.com openshift_public_hostname=master-
[0:2].ocp3.example.com

[masters:vars]
openshift_schedulable=false
openshift_router_selector="region=infra"
openshift_registry_selector="region=infra"

[etcd]
master-[0:2].control.ocp3.example.com openshift_public_hostname=master-
[0:2].ocp3.example.com

[infra-nodes]
infra-node-[0:1].control.ocp3.example.com openshift_node_labels="
{'region': 'infra', 'zone': 'default'}" openshift_public_hostname=infra-
node-[0:1].ocp3.example.com

[app-nodes]
app-node-[0:2].control.ocp3.example.com openshift_node_labels="{'region':
'primary', 'zone': 'default'}"

[nodes]
master-[0:2].control.ocp3.example.com openshift_node_labels="{'region':
'primary', 'zone': 'default'}" openshift_schedulable=false
openshift_public_hostname=master-[0:2].ocp3.example.com

[nodes:children]
infra-nodes
app-nodes
```

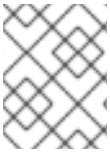


NOTE

The above is just a sample inventory file for reference. For more examples, please visit the Installation and Configuration Guide, Multiple-Masters

5.9.1.1. `group_vars` and the `OSv3.yml` file

The following defines the global values in the OSEv3 section and all of its children in a file named **`group_vars/OSEv3.yml`**.



NOTE

The **`group_vars/OSEv3.yml`** file exists in the same directory previously created within the *bastion* host file.

Global configuration values: `group_vars/OSEv3.yml`

```
deployment_type: openshift-enterprise
openshift_master_default_subdomain: apps.ocp3.example.com (1)

# Developer access to WebUI and API
openshift_master_cluster_hostname: devs.ocp3.example.com (2)
openshift_master_cluster_public_hostname: devs.ocp3.example.com (3)
openshift_master_cluster_method: native

openshift_override_hostname_check: true
openshift_set_node_ip: true
openshift_use_dnsmasq: false
osm_default_node_selector: 'region=primary'

# Enable Flannel and set interface
openshift_use_openshift_sdn: false
openshift_use_flannel: true
flannel_interface: eth1

openshift_cloudprovider_kind: openstack
openshift_cloudprovider_openstack_auth_url: http://10.0.0.1:5000/v2.0 (4)
openshift_cloudprovider_openstack_username: <username> (5)
openshift_cloudprovider_openstack_password: <password> (6)
openshift_cloudprovider_openstack_tenant_name: <tenant name> (7)

#If enabling a certificate, uncomment the following line.
#openshift_master_ldap_ca_file: /path/to/certificate.crt

# NOTE: Ensure to include the certificate name i.e certificate.crt within
the
# ca field under openshift_master_identity_providers
# Change insecure value from true to false if using a certificate in the
# openshift_master_identity_providers section

openshift_master_identity_providers:
  - name: ldap_auth
    kind: LDAPPasswordIdentityProvider
    challenge: true
    login: true
    bindDN: cn=openshift,cn=users,dc=example,dc=com (8)
    bindPassword: password (9)
    url: ldap://ad1.example.com:389/cn=users,dc=example,dc=com?
sAMAccountName (10)
    attributes:
      id: ['dn']
      email: ['mail']
      name: ['cn']
      preferredUsername: ['sAMAccountName']
    ca: ''
    insecure: True
```

(1) Apps are named **<name>.apps.ocp3.example.com**. Adjust as required.

NOTE

This name must resolve to the IP address of the app load-balancer.

2 3 Developers access the WebUI and API at this name. Adjust as required.

NOTE

This name must resolve to the IP address of the master load-balancer.

4 The URL of the RHOSP API service. Adjust as required.

5 The RHOSP username.

6 The RHOSP password.

7 The RHOSP project that contains the RHOCP service.

8 An LDAP user DN authorized to make LDAP user queries.

9 The password for the LDAP query user.

10 The query URL for LDAP authentication. Adjust the server name and the user base DN as required.

5.9.1.2. Ansible Configuration file: `ansible.cfg`

The final file is **`ansible.cfg`**. The following is a representation of the reference environment's **`ansible.cfg`** file.

NOTE

Within this reference environment, this file resides in the *bastion* host **`/home/cloud-user/ansible_files`** directory.

Ansible customization: `ansible.cfg`

```
# config file for ansible -- http://ansible.com/
# ==============================================
[defaults]
remote_user = cloud-user
forks = 50
host_key_checking = False
gathering = smart
retry_files_enabled = false
nocows = true

[privilege_escalation]
become = True

[ssh_connection]
```

```
ssh_args = -o ControlMaster=auto -o ControlPersist=900s -o
GSSAPIAuthentication=no
control_path = /var/tmp/%%h-%%r
```

5.9.2. Run Ansible Installer

Prior to deploying using OpenShift Ansible playbooks to deploy RHOCP, update all the RHOCP nodes, and install the **atomic-openshift-utils** package.

Preinstallation packages update and required packages install

```
#!/bin/sh
ansible nodes -i ~/ansible_files/inventory -b -m yum -a "name=*
state=latest"
ansible nodes -i ~/ansible_files/inventory -b -m yum -a "name=atomic-
openshift-utils state=latest"
```

With the configurations on all the nodes set, invoke the OpenShift Ansible playbooks to deploy RHOCP.

Install OpenShift Container Platform

```
#!/bin/sh
export OCP_ANSIBLE_ROOT=${OCP_ANSIBLE_ROOT:-/usr/share/ansible}
export ANSIBLE_ROLES_PATH=${ANSIBLE_ROLES_PATH:-
${OCP_ANSIBLE_ROOT}/openshift-ansible/roles}
export ANSIBLE_HOST_KEY_CHECKING=False

ansible-playbook -i ~/ansible_files/inventory \
  ${OCP_ANSIBLE_ROOT}/openshift-ansible/playbooks/byo/config.yml
```

The final step is to configure **iptables** to allow traffic on the *master*, *infrastructure*, and *app* nodes. These commands makes use of Ansible to run the command on all the nodes.

1. Create a copy of the **iptables** rules

   #### Backup iptables rules

   ```
   #!/bin/sh
   ansible nodes -i ~/ansible_files/inventory -b -m copy -a
   "remote_src=true src=/etc/sysconfig/iptables
   dest=/etc/sysconfig/iptables.orig"
   ```

2. Run the following command that accepts traffic on the DOCKER chain, as well as allows **POSTROUTING MASQUERADING** on a specific Ethernet device, i.e. **eth1**, and persist the **iptables** rules

   #### Apply and persist iptables rules

   ```
   #!/bin/sh
   ansible nodes -i ~/ansible_files/inventory -b -m shell \
     -a 'iptables -A DOCKER -p all -j ACCEPT'
   ansible nodes -i ~/ansible_files/inventory -b -m shell \
     -a 'iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE'
   ```

```
ansible nodes -i ~/ansible_files/inventory -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

eth1 is the **flannel** interface within the RHOCP nodes.

5.9.3. Post Installation Tasks

Once OpenShift Container Platform has been installed, there are some post-installation tasks that need to be performed

5.9.3.1. Configure bastion host

To be able to do administrative tasks or interact with the OpenShift Container Platform cluster, the bastion host cloud-user can be configured to be logged as **system:admin** using the following commands:

Configure Bastion Host for oc cli

```
#!/bin/sh
mkdir ~/.kube/
ssh master-0 sudo cat /etc/origin/master/admin.kubeconfig > ~/.kube/config
sudo yum -y install atomic-openshift-clients
oc whoami
```

5.9.3.2. Create StorageClass

OpenShift Container Platform can provide dynamic storage to pods that requires persistent storage (**pvc**) using cinder volumes by creating **StorageClasses**.

NOTE

This process is performed transparently for the user.

This requires the OCP cluster to be configured to be able to interact with the OSP API, which is done by the ansible installer with the cloud parameters before.

The following command creates a StorageClass named "standard" in the "nova" OSP availability zone:

Configure StorageClass

```
#!/bin/sh
oc create -f - <<API
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: standard
provisioner: kubernetes.io/cinder
```

```
parameters:
  availability: nova
API
```

NOTE

Default **storageclass** is a feature included in future OCP releases.

CHAPTER 6. OPERATIONAL MANAGEMENT

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

6.1. RUNNING DIAGNOSTICS

Perform the following steps from the first master node.

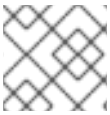
To run diagnostics, **SSH** into the the first master node. Direct access is provided to the first master node because of the configuration of the local **~/.ssh/config** file.

```
ssh master-0
```

Connectivity to the master00 host as the **root** user should have been established. Run the diagnostics that are included as part of the install.

```
sudo oadm diagnostics
[Note] Determining if client configuration exists for client/cluster
diagnostics
Info:  Successfully read a client config file at '/root/.kube/config'
Info:  Using context for cluster-admin access: 'default/devs-ocp3-example-
com:8443/system:admin'
[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/devs-ocp3-example-
com:8443/system:admin]
       Description: Validate client config context is complete and has
connectivity

Info:  The current client config context is 'default/devs-ocp3-example-
com:8443/system:admin':
       The server URL is 'https://devs.ocp3.example.com:8443'
       The user authentication is 'system:admin/devs-ocp3-example-
com:8443'
       The current project is 'default'
       Successfully requested project list; has access to project(s):
         [logging management-infra markllama openshift openshift-infra
default]

[Note] Running diagnostic: DiagnosticPod
       Description: Create a pod to run diagnostics from the application
standpoint


[Note] Running diagnostic: ClusterRegistry
       Description: Check that there is a working Docker registry

[Note] Running diagnostic: ClusterRoleBindings
       Description: Check that the default ClusterRoleBindings are present
and contain the expected subjects

Info:  clusterrolebinding/cluster-readers has more subjects than expected.

       Use the oadm policy reconcile-cluster-role-bindings command to
```

```
update the role binding to remove extra subjects.

Info:  clusterrolebinding/cluster-readers has extra subject
{ServiceAccount management-infra management-admin    }.

[Note] Running diagnostic: ClusterRoles
       Description: Check that the default ClusterRoles are present and
contain the expected permissions

[Note] Running diagnostic: ClusterRouterName
       Description: Check there is a working router

[Note] Skipping diagnostic: MasterNode
       Description: Check if master is also running node (for Open
vSwitch)
       Because: Network plugin does not require master to also run node:

[Note] Running diagnostic: NodeDefinitions
       Description: Check node records on master

[Note] Running diagnostic: AnalyzeLogs
       Description: Check for recent problems in systemd service logs

Info:  Checking journalctl logs for 'docker' service

[Note] Running diagnostic: MasterConfigCheck
       Description: Check the master config file

Info:  Found a master config file: /etc/origin/master/master-config.yaml

... output abbreviated ...

[Note] Running diagnostic: UnitStatus
       Description: Check status for related systemd units

[Note] Summary of diagnostics execution (version v3.2.1.15):
[Note] Warnings seen: 3
```



NOTE

The warnings do not cause issues in the environment

Based on the results of the diagnostics, actions can be taken to alleviate any issues.

6.2. CHECKING THE HEALTH OF ETCD

Perform the following steps from a local workstation.

This section focuses on the **ETCD** cluster. It describes the different commands to ensure the cluster is healthy. The internal **DNS** names of the nodes running **ETCD** must be used.

Issue the **etcdctl** command to confirm that the cluster is healthy.

```
sudo etcdctl \ --ca-file /etc/etcd/ca.crt \ --cert-
```

```
file=/etc/origin/master/master.etcd-client.crt \ --key-
file=/etc/origin/master/master.etcd-client.key \ --endpoints
https://master-0.ocp3.example.com:2379 \ --endpoints https://master-
1.ocp3.example.com:2379 \ --endpoints https://master-
2.ocp3.example.com:2379 \ cluster-health
member 9bd1d7731aa447e is healthy: got healthy result from
https://172.18.10.4:2379
member 2663a31f4ce5756b is healthy: got healthy result from
https://172.18.10.5:2379
member 3e8001b17125a44e is healthy: got healthy result from
https://172.18.10.6:2379
cluster is healthy
```

6.3. DOCKER STORAGE SETUP

The role docker-storage-setup tells the Docker service to use **/dev/vdb** and create the volume group of **docker-vol**. The extra Docker storage options ensures that a container can grow no larger than 3G. Docker storage setup is performed on all master, infrastructure, and application nodes.

```
# vi /etc/sysconfig/docker-storage-setup
DEVS=/dev/vdb
VG=docker-vg
```

6.4. YUM REPOSITORIES

The specfic repositories for a successful OpenShift installation are define inn section Register for Software Updates. All systems except for the bastion host must have the same subscriptions. To verify those subscriptions, match those defined earlier, perfom the following.

```
sudo yum repolist
Loaded plugins: search-disabled-repos, subscription-manager
repo id                                          repo name
status
rhel-7-server-extras-rpms/x86_64                 Red Hat Enterprise Linux 7
536
rhel-7-server-openstack-10-rpms/7Server/x86_64 Red Hat OpenStack Platform
1,179
rhel-7-server-optional-rpms/7Server/x86_64     Red Hat Enterprise Linux 7
11,098
rhel-7-server-ose-3.2-rpms/x86_64                Red Hat OpenShift Enterpri
847
rhel-7-server-rpms/7Server/x86_64                Red Hat Enterprise Linux 7
14,619
repolist: 28,279
```

6.5. CONSOLE ACCESS

This section covers logging into the OpenShift Container Platform management console via the GUI and the CLI. After logging in via one of these methods applications can then be deployed and managed.

6.5.1. Log into GUI console and deploy an application

Perform the following steps from the local workstation.

To log into the GUI console access the CNAME for the load balancer. Open a browser and access https://devs.ocp3.example.com:8443/console

To deploy an application, click on the **New Project** button. Provide a **Name** and click **Create**. Next, deploy the **jenkins-ephemeral** instant app by clicking the corresponding box. Accept the defaults and click **Create**. Instructions along with a URL are provided for how to access the application on the next screen. Click **Continue to Overview** and bring up the management page for the application. Click on the link provided and access the appliction to confirm functionality.

6.5.2. Log into CLI and Deploy an Application

Perform the following steps from a local workstation.

Install the **oc client** which can be installed by visiting the public URL of the OpenShift deployment. For example, https://devs.ocp3.example.com:8443/console/command-line and click latest release. When directed to https://access.redhat.com, login with the valid Red Hat customer credentials and download the client relevant to the current workstation. Follow the instructions located on the production documentation site for getting started.

Log in with a user that exists in the LDAP database. For this example use the **openshift** user we used as the LDAP BIND_DN user.

```
oc login --username openshift
Authentication required for https://devs.ocp3.example.com:8443 (openshift)
Username: openshift
Password:
Login successful.

You don't have any projects. You can try to create a new project, by
running

    $ oc new-project <projectname>
```

After access has been granted, create a new project and deploy an application.

```
$ oc new-project test-app

$ oc new-app https://github.com/openshift/cakephp-ex.git --name=php
--> Found image 2997627 (7 days old) in image stream "php" in project
"openshift" under tag "5.6" for "php"

    Apache 2.4 with PHP 5.6
    -----------------------
    Platform for building and running PHP 5.6 applications

    Tags: builder, php, php56, rh-php56

    * The source repository appears to match: php
    * A source build using source code from
https://github.com/openshift/cakephp-ex.git is created
      * The resulting image is pushed to image stream "php:latest"
    * This image is deployed in deployment config "php"
    * Port 8080/tcp is load balanced by service "php"
```

```
      * Other containers access this service through the hostname "php"

--> Creating resources with label app=php ...
    imagestream "php" created
    buildconfig "php" created
    deploymentconfig "php" created
    service "php" created
--> Success
    Build scheduled, use 'oc logs -f bc/php' to track its progress.
    Run 'oc status' to view apps.


$ oc expose service php
route "php" exposed
```

Display the status of the application.

```
$ oc status
In project test-app on server https://openshift-master.sysdeseng.com:8443

http://test-app.apps.sysdeseng.com to pod port 8080-tcp (svc/php)
  dc/php deploys istag/php:latest <- bc/php builds
https://github.com/openshift/cakephp-ex.git with openshift/php:5.6
    deployment #1 deployed about a minute ago - 1 pod

1 warning identified, use 'oc status -v' to see details.
```

Access the application by accessing the URL provided by **`oc status`**. The CakePHP application should be visible now.

6.6. EXPLORE THE ENVIRONMENT

6.6.1. List Nodes and Set Permissions

The following command should fail:

```
# oc get nodes --show-labels
Error from server: User "user@redhat.com" cannot list all nodes in the
cluster
```

The reason it is failing is because the permissions for that user are incorrect. Get the username and configure the permissions.

```
$ oc whoami
openshift
```

Once the username has been established, log back into a master node and enable the appropriate permissions for the user. Perform the following step from master00.

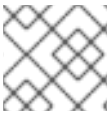
```
# oadm policy add-cluster-role-to-user cluster-admin openshift
```

Attempt to list the nodes again and show the labels.

```
# oc get nodes --show-labels
NAME                                         STATUS    AGE
LABELS
app-node-0.control.ocp3.example.com     Ready     5h        failure-
domain.beta.kubernetes.io/region=RegionOne,kubernetes.io/hostname=app-
node-0.control.ocp3.example.com,region=primary,zone=default
app-node-1.control.ocp3.example.com     Ready     5h        failure-
domain.beta.kubernetes.io/region=RegionOne,kubernetes.io/hostname=app-
node-1.control.ocp3.example.com,region=primary,zone=default
app-node-2.control.ocp3.example.com     Ready     5h        failure-
domain.beta.kubernetes.io/region=RegionOne,kubernetes.io/hostname=app-
node-2.control.ocp3.example.com,region=primary,zone=default
infra-node-0.control.ocp3.example.com   Ready     5h        failure-
domain.beta.kubernetes.io/region=RegionOne,kubernetes.io/hostname=infra-
node-0.control.ocp3.example.com,region=infra,zone=default
infra-node-1.control.ocp3.example.com   Ready     5h        failure-
domain.beta.kubernetes.io/region=RegionOne,kubernetes.io/hostname=infra-
node-1.control.ocp3.example.com,region=infra,zone=default
```

6.6.2. List Router and Registry

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



NOTE

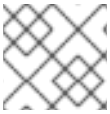
Perform the following steps from a workstation.

```
# oc project default
# oc get all
# oc status
In project default on server https://devs.ocp3.example.com:8443

svc/docker-registry - 172.30.110.31:5000
  dc/docker-registry deploys docker.io/openshift3/ocp-docker-
registry:v3.2.1.7
    deployment #2 deployed 41 hours ago - 2 pods
    deployment #1 deployed 41 hours ago

svc/kubernetes - 172.30.0.1 ports 443, 53->8053, 53->8053

svc/router - 172.30.235.155 ports 80, 443, 1936
  dc/router deploys docker.io/openshift3/ocp-haproxy-router:v3.2.1.7
    deployment #1 deployed 41 hours ago - 2 pods

View details with 'oc describe <resource>/<name>' or list everything with
'oc get all'.
```

Observe the output of **oc get all** and **oc status**. Notice that the registry and router information is clearly listed.

6.6.3. Explore the Docker Registry

The OpenShift Ansible playbooks configure two infrastructure nodes that have two registries running. 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.



NOTE

Perform the following steps from a workstation.

```
$ oc describe svc/docker-registry
Name:           docker-registry
Namespace:      default
Labels:         docker-registry=default
Selector:       docker-registry=default
Type:           ClusterIP
IP:         172.30.110.31
Port:           5000-tcp    5000/TCP
Endpoints:      172.16.4.2:5000
Session Affinity:   ClientIP
No events.
```

NOTE

Perform the following steps from the infrastructure node.

Once the endpoints are known, go to one of the infra nodes running a registry and grab some information about it. Capture the container UID in the leftmost column of the output.

```
# docker ps | grep ocp-docker-registry
073d869f0d5f        openshift3/ocp-docker-registry:v3.2.1.9   "/bin/sh -c
'DOCKER_R"   6 hours ago         Up 6 hours
k8s_registry.90479e7d_docker-registry-2-jueep_default_d5882b1f-5595-11e6-
a247-0eaf3ad438f1_ffc47696
```

```
sudo docker exec -it a637d95aa4c7 cat /config.yml
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    layerinfo: inmemory
  filesystem:
    rootdirectory: /registry
  delete:
    enabled: true
auth:
  openshift:
    realm: openshift
middleware:
  repository:
```

```
    - name: openshift
      options:
        pullthrough: true
```

6.6.4. Explore Docker Storage

This section explores the Docker storage on an infrastructure node.

NOTE

The example below can be performed on any node but for this example the infrastructure node is used

```
sudo docker info
Containers: 4
 Running: 4
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: docker-253:1-75502733-pool
 Pool Blocksize: 65.54 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 1.118 GB
 Data Space Total: 107.4 GB
 Data Space Available: 39.96 GB
 Metadata Space Used: 1.884 MB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.146 GB
 Udev Sync Supported: true
 Deferred Removal Enabled: false
 Deferred Deletion Enabled: false
 Deferred Deleted Device Count: 0
 Data loop file: /var/lib/docker/devicemapper/devicemapper/data
 WARNING: Usage of loopback devices is strongly discouraged for production
use. Either use --storage-opt dm.thinpooldev or use --storage-opt
dm.no_warn_on_loop_devices=true to suppress this warning.
 Metadata loop file: /var/lib/docker/devicemapper/devicemapper/metadata
 Library Version: 1.02.107-RHEL7 (2016-06-09)
Execution Driver: native-0.2
Logging Driver: json-file
Plugins:
 Volume: local
 Network: host bridge null
 Authorization: rhel-push-plugin
Kernel Version: 3.10.0-327.10.1.el7.x86_64
Operating System: Employee SKU
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 2
CPUs: 2
```

```
Total Memory: 3.702 GiB
Name: infra-node-0.control.ocp3.example.com
ID: AVUO:RUKL:Y7NZ:QJKC:KIMX:5YXG:SJUY:GGH2:CL3P:3BTO:6A74:4KYD
WARNING: bridge-nf-call-ip6tables is disabled
Registries: registry.access.redhat.com (secure), docker.io (secure)
```

```
$ fdisk -l

Disk /dev/vda: 42.9 GB, 42949672960 bytes, 83886080 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: 0x0000b3fd

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    83884629    41941291   83  Linux

Disk /dev/vdb: 8589 MB, 8589934592 bytes, 16777216 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 /dev/mapper/docker-253:1-75502733-pool: 107.4 GB, 107374182400 bytes,
209715200 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 65536 bytes


Disk /dev/mapper/docker-253:1-75502733-
0f03fdafb3f541f0ba80fa40b28355cd78ae2ef9f5cab3c03410345dc97835f0: 10.7 GB,
10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 65536 bytes


Disk /dev/mapper/docker-253:1-75502733-
e1b70ed2deb6cd2ff78e37dd16bfe356504943e16982c10d9b8173d677b5c747: 10.7 GB,
10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 65536 bytes


Disk /dev/mapper/docker-253:1-75502733-
c680ec6ec5d72045fc31b941e4323cf6c17b8a14105b5b7e142298de9923d399: 10.7 GB,
10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 65536 bytes


Disk /dev/mapper/docker-253:1-75502733-
```

```
80f4398cfd272820d625e9c26e6d24e57d7a93c84d92eec04ebd36d26b258533: 10.7 GB,
10737418240 bytes, 20971520 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 65536 bytes / 65536 bytes
```

```
$ cat /etc/sysconfig/docker-storage-setup
DEVS=/dev/xvdb
VG=docker-vol
```

6.6.5. Explore Security Groups

As mentioned earlier in the document several security groups have been created. The purpose of this section is to encourage exploration of the security groups that were created.

NOTE

Perform the following steps from the OpenStack web console.

Select the **Compute** tab and the **Access and Security** sub-tab in the upper left of the web console. Click through each group using the **Manage Rules** button in the right column and check out both the **Inbound** and **Outbound** rules that were created as part of the infrastructure provisioning. For example, notice how the **Bastion** security group only allows **SSH** traffic inbound. That can be further restricted to a specific network or host if required. Next take a look at the **Master** security group and explore all the **Inbound** and **Outbound** TCP and UDP rules and the networks from which traffic is allowd.

6.7. TESTING FAILURE

In this section, reactions to failure are explored. After a sucessful install and some of the smoke tests noted above have been completed, failure testing is executed.

6.7.1. Generate a Master Outage

When a master instance fails, the service should remain available.

Stop one of the master instances:

Stop a Master instance

```
nova stop master-0.control.ocp3.example.com
Request to stop server master-0.control.ocp3.example.com has been
accepted.
nova list --field name,status,power_state | grep master
| 4565505c-e48b-43e7-8c77-da6c1fc3d7d8 | master-0.control.ocp3.example.com
| SHUTOFF | Shutdown     |
| 12692288-013b-4891-a8a0-71e6967c656d | master-1.control.ocp3.example.com
| ACTIVE | Running     |
| 3cc0c6f0-59d8-4833-b294-a3a47c37d268 | master-2.control.ocp3.example.com
| ACTIVE | Running     |
```

Ensure the console can still be accessed by opening a browser and accessing devs.ocp3.example.com. At this point, the cluster is in a degraded state because only 2/3 master nodes are running, but complete funcionality remains.

6.7.2. Observe the Behavior of ETCD with a Failed Master Node

One master instance is down. The master instances contain the **etcd** daemons.

NOTE

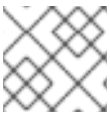
Run this commands on one of the active master servers.

Check etcd cluster health

```
# etcdctl -C https://master-
0.control.ocp3.example.com:2379,https://master-
1.control.ocp3.example.com:2379,https://master-
2.control.ocp3.example.com:2379 --ca-file /etc/etcd/ca.crt --cert-
file=/etc/origin/master/master.etcd-client.crt --key-
file=/etc/origin/master/master.etcd-client.key cluster-health
failed to check the health of member 82c895b7b0de4330 on
https://10.30.1.251:2379: Get https://10.30.1.251:2379/health: dial tcp
10.30.1.251:2379: i/o timeout
member 82c895b7b0de4330 is unreachable: [https://10.30.1.251:2379] are all
unreachable
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.30.3.74:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.30.2.157:2379
cluster is healthy
```

Notice how one member of the **ETCD** cluster is now unreachable.

Restart master-0.



NOTE

Run this command from a workstation.

Restart master instance

```
nova start master-0.control.ocp3.example.com
Request to start server master-0.control.example.com has been accepted.
nova list --field name,status,power_state | grep master
| 4565505c-e48b-43e7-8c77-da6c1fc3d7d8 | master-0.control.ocp3.example.com
| ACTIVE | Running     |
| 12692288-013b-4891-a8a0-71e6967c656d | master-1.control.ocp3.example.com
| ACTIVE | Running     |
| 3cc0c6f0-59d8-4833-b294-a3a47c37d268 | master-2.control.ocp3.example.com
| ACTIVE | Running     |
```

Verify etcd cluster health

```
# *etcdctl -C https://master-
0.control.ocp3.example.com:2379,https://master-
1.control.ocp3.example.com:2379,https://master-
2.control.ocp3.example.com:2379 --ca-file /etc/etcd/ca.crt --cert-
file=/etc/origin/master/master.etcd-client.crt --key-
file=/etc/origin/master/master.etcd-client.key cluster-health*
member 82c895b7b0de4330 is healthy: got healthy result from
https://10.30.1.251:2379
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.30.3.74:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.30.2.157:2379
cluster is healthy
```

6.8. DYNAMIC PROVISIONED STORAGE

Persistent volumes (pv) are OpenShift objects that allow for storage to be defined and then claimed by pods to allow for data persistence. Mounting of persistent volumes is done by using a **persistent volume claim** (pvc). This claim mounts the persistent storage to a specific directory within a pod referred to as the **mountpath**.

6.8.1. Creating a 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 RHOSP, 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**.

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

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: <name>
provisioner: kubernetes.io/cinder
parameters:
  availability: nova
```

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 RHOSP 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/v1beta1
metadata:
  name: <name>
provisioner: kubernetes.io/cinder
parameters:
  type: <volumetypename>
  availability: nova
```

NOTE

StorageClass is only defined and maintained on a per project basis.

6.8.2. Creating a Persistent Volumes Claim

When creating a new application, a pod is created using non-persistent storage labeled as **EmptyDir**. In order to provide persistent storage to the application just created, a persistent volume claim must be created. The following example shows the creation of a MySQL application that initially uses non-persistent storage, but is then assigned a persistent volume using a persistent storage volume claim.

The following command creates the application. In this example, the application created is a MySQL application.

```
$ oc new-app --docker-image registry.access.redhat.com/openshift3/mysql-
55-rhel7 --name=db -e 'MYSQL_USER=myuser' -e 'MYSQL_PASSWORD=d0nth@x' -e
'MYSQL_DATABASE=persistent'
```

The following shows that the application pod is currenty running.

```
$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
db-1-6kn6c                 1/1       Running   0          5m
```

With no persistent storage in place, the volumes section when describing the pod shows that the volume is in fact non-persistent storage. This is shown by the temporary directory that shares a pod's lifetime labeled **EmptyDir**.

```
$ oc describe pod db-1-6kn6c | grep -i volumes -A3
Volumes:
  db-volume-1:
    Type:   EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
```

Verify that the **StorageClass** has been created

```
$ oc get storageclass
NAME      TYPE
gold      kubernetes.io/cinder
```

Create a persistent storage claim yaml file. In this example the file consists of application labeled db, a storage class labeled gold, and the amount of storage requested at 10GB.

```
$ cat db-claim.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 annotations:
   volume.beta.kubernetes.io/storage-class: gold
spec:
 accessModes:
  - ReadWriteMany
 resources:
   requests:
     storage: 10Gi
```

Execute the persistent volume claim yaml file.

```
$ oc create -f db-claim.yaml
persistentvolumeclaim "db" created
```

Verify the persistent volume claim has been created and is bound.

```
$ oc get pvc
NAME      STATUS    VOLUME                                     CAPACITY
ACCESSMODES   AGE
db        Bound     pvc-cd8f0e34-02b6-11e7-b807-fa163e5c3cb8   10Gi
RWX
10s
```

Provide the persistent volume to the MySQL application pod labeled 'db'.

```
$ oc volume dc/db --add --overwrite --name=db-volume-1 --
type=persistentVolumeClaim --claim-name=db
deploymentconfig "db" updated
```

Describe the db pod to ensure it is using the persistent volume claim.

```
$ oc describe dc/db | grep -i volumes -A3
  Volumes:
   db-volume-1:
    Type:   PersistentVolumeClaim (a reference to a PersistentVolumeClaim
in the same namespace)
    ClaimName:  db
    ReadOnly:   false
```

CHAPTER 7. DEPLOYING USING HEAT

The installation created earlier shows the creation and configuration of the OSP resources to host an OCP service. It also shows how to run the **`openshift-ansible`** installation process directly.

NOTE

Heat is the OpenStack *orchestration system*.

Orchestration allows the end user to describe the system to their specfications rather than the process to create it. The OCP RPM suite includes a set of templates for an OCP service. Using these templates it is possible to create a working OCP service on OSP with a single file input.

7.1. PROJECT QUOTAS

Each project in OSP has a set of resource quotas which are set by default. Several of these values should be increased to allow the OCP stack to fit.

Table 7.1. OSP Resource Minimum Quotas

Resource	Minimum	Recommended
Instances	9	20
VCPUs	20	60
RAM (GB)	50	450
Floating IPs	9	15
Security Groups	5	5
Volumes	10	30
Volume Storage (GB)	800	2000

These numbers are for a basic general-purpose installation with low to moderate use. It allows for scaling up to 10 more application nodes. The correct values for a specific installation depends on the expected use. They are calculated using a detailed analysis of the actual resources needed and available.

NOTE

The number of nodes, selection of instance flavors and disk volume sizes here are for demonstration purposes. To deploy a production service consult the OCP sizing guidelines for each resource.

7.2. BENEFITS OF HEAT

Heat orchestration and the heat engine offer on-demand detection that can respond to automatic

adding and removing nodes based upon the running workloads. The Heat templates integrate the RHOCP service with the Ceilometer monitoring service in RHOSP. While Ceilometer monitors the environment, it can signal the Heat stack to appropriate increase or decrease nodes upon workload requirements.

7.3. INSTALLATION OF THE HEAT TEMPLATES

The installation of the Heat Templates is best approached using the RHOSP heat CLI tool in conjunction with the OpenShift Heat templates. A successful installation depends on the enablement of the rhel-7-server-openstack-10-rpms repository. Once enabled, the heat stack installation process can be initiated from any host that can access the RHOSP service using an RHOSP CLI client.

Enable RHOSP repository & install Heat RPMs.

```
sudo subscription-manager repos --enable rhel-7-server-openstack-10-rpms
sudo yum -y install python-heatclient openshift-heat-templates
```

The template files are installed at **`/usr/share/openshift/heat/templates`**.

7.4. HEAT STACK CONFIGURATION

Heat uses one or more YAML input files to customize the stacks it creates.

This configuration uses an external DNS service as detailed in Appendix B and a dedicated loadbalancer created as part of the Heat stack. DNS records are created to direct inbound traffic for the masters and the OpenShift router through this loadbalancer. This configuration uses the *flannel* SDN. The Docker storage is set to a low value as this reference environment is for demonstration purposes only. When deploying in production environments, ensure to tune these values to accomodate for appropriate container density.

OpenShift Heat Stack Confguration File - `openshift_parameters.yaml`

```
# Invoke:
# heat stack-create ocp3-heat \
#      -e openshift_parameters.yaml \
#      -e /usr/share/openshift-heat-
templates/env_loadbalancer_dedicated.yaml \
#      -t /usr/share/openshift-heat-templates/openshift.yaml
#
parameters:
  # OpenShift service characteristics (1)
  deployment_type: openshift-enterprise
  domain_name: "ocp3.example.com"
  app_subdomain: "apps.ocp3.example.com"
  lb_hostname: "devs"
  loadbalancer_type: dedicated
  openshift_sdn: flannel
  deploy_router: true
  deploy_registry: true

  # Number of each server type (2)
  master_count: 3
  infra_count: 3
  node_count: 2
```

```
  # OpenStack network characteristics 3
  external_network: public_network
  internal_subnet: "172.22.10.0/24"
  container_subnet: "172.22.20.0/24"

  # DNS resolver and updates 4
  dns_nameserver: 10.x.x.130,10.x.x.29,10.x.x.19
  dns_update_key: <HMAC:MD5 string>

  # Instance access 5
  ssh_key_name: ocp3
  ssh_user: cloud-user

  # Image selection 6
  bastion_image: rhel7
  master_image: rhel7
  infra_image: rhel7
  node_image: rhel7
  loadbalancer_image: rhel7

  # Docker Storage controls 7
  master_docker_volume_size_gb: 10
  infra_docker_volume_size_gb: 20
  node_docker_volume_size_gb: 100

  # OpenStack user credentials 8
  os_auth_url: http://10.x.x.62:5000/v2.0
  os_username: <username>
  os_password: <password>
  os_region_name: <region name>
  os_tenant_name: <project name>

  # Red Hat Subscription information 9
  rhn_username: "<username>"
  rhn_password: "<password>"
  rhn_pool: '<pool id containing openshift>'

parameter_defaults:
  # Authentication service information 10
  ldap_url: "ldap://ad.example.com.com:389/cn=users,dc=example,dc=com?
sAMAccountName"
  ldap_preferred_username: "sAMAccountName"
  ldap_bind_dn: "cn=openshift,cn=users,dc=example,dc=com"
  ldap_bind_password: "password"
  ldap_insecure: true

resource_registry: 11
  # Adjust path for each entry
  OOShift::LoadBalancer: /usr/share/openshift-heat-
templates/loadbalancer_dedicated.yaml
  OOShift::ContainerPort: /usr/share/openshift-heat-
templates/sdn_flannel.yaml
  OOShift::IPFailover: /usr/share/openshift-heat-
templates/ipfailover_keepalived.yaml
```

```
  OOShift::DockerVolume: /usr/share/openshift-heat-
templates/volume_docker.yaml
  OOShift::DockerVolumeAttachment: /usr/share/openshift-heat-
templates/volume_attachment_docker.yaml
  OOShift::RegistryVolume: /usr/share/openshift-heat-
templates/registry_ephemeral.yaml
```

1. RHOCP Service Configuration
   This section defines the RHOCP service itself. The parameters here set the public view for the developers and application users.

2. Number of each instance type
   This section determines the number of each type of component of the RHOCP service.

3. RHOSP Network Definition
   This section defines the internal networks and how the RHOCP servers connect to the public network.

4. DNS Services and Updates
   This section defines DNS servers and provides the means for the Heat templates to populate name services for the new instances as they are created. See Generating an Update Key in the appendices for details.

5. Instance access
   This section defines how Ansible accesses the instances to configure and manage them.

6. Base image for each instance type
   This section selects the base image for each instance within the RHOCP deployment.

7. Docker Storage Sizing
   This section controls the amount of storage allocated on each instance to contain Docker images and container runtime files.

8. RHOSP credentials
   This section defines the credentials which allows the bastion host and kubernetes on each node to communicate with the cloud provider to manage storage for containers.
   NOTE: It is **critical** that the values here match those in the Red Hat OpenStack Platform Credentials section. Incorrect values can result in installation failures that are difficult to diagnose.

9. Red Hat Subscription Credentials
   This section defines the credentials used to enable software subscription and updates.

10. LDAP Authentication
    This section defines the credentials that enable RHOCP to authenticate users from an existing LDAP server.

11. Sub-templates
    This section defines a set of sub-templates that enable extensions or variations like dedicated loadbalancer or SDN selection.

7.5. HOSTNAME GENERATION IN HEAT STACK

Two of the hostnames generated by the heat stack installation are significant for users who need to find the service. The hostnames are generated from the `domain_name` and the `lb_hostname`

parameters in the YAML file and from the heat stack name given on the CLI when the stack is created. This is to avoid naming conflicts if multiple stacks are created.

- domain_name: **`ocp3.example.com`**
- stack name: **`ocp3-heat`**
- lb_hostname: devs

Table 7.2. OCP Service Host Names

host	FQDN
suffix	**`ocp3.example.com`**
master LB	**`ocp3-heat-devs.ocp3.example.com`**
application LB	**`*.apps.ocp3.example.com`**

The instances that make up the Heat deployment get names composed from the stack name and domain. The master and infrastructure nodes are distinguished from each other by simple integer serial numbers. The nodes are handled differently as they can be created on demand when a load trigger event occurs. The heat stack assigns each node a random string to distingush them.

A listing of all the instance names can be found using **`nova list --field name`**.

Table 7.3. OCP instance names in OSP Nova

instance Type	Name Template	Example
bastion	<stackname>-bastion.<domain>	ocp3-heat-bastion.ocp3.example.com
master	<stackname>-master-<num>.<domain>	ocp3-heat-master-0.ocp3.example.com
infrastructure node	<stackname>-infra-<num>.<domain>	ocp3-heat-infra-0.ocp3.example.com
application node	<stackname>-node-<hash>.<domain>	ocp3-heat-node-12345678

These are the names developers and application users use to reach the OCP service. These two domain names must be registered in DNS and must point to the load-balancer. The load-balancer must be configured with the floating IP addresses of the master instances for port 8443 and with the addresses of the infrastructure nodes on ports 80 and 443 for access to the OCP service.

7.6. CREATING THE STACK

Create the Heat Stack

```
heat stack-create ocp3-heat \ --timeout 120 \ -e openshift_parameters.yaml
\ -e /usr/share/openshift-heat-templates/env_loadbalancer_dedicated.yaml \
-f /usr/share/openshift-heat-templates/openshift.yaml
```

7.7. OBSERVING DEPLOYMENT

Observe the Stack Creation

```
heat stack-list ocp3-heat
heat resource-list ocp3-heat | grep CREATE_IN_PROGRESS
```

7.8. VERIFYING THE OCP SERVICE

7.8.1. Ops Access

Log into the bastion host through **nova ssh**

```
nova ssh -i ocp3_rsa cloud-user@openshift-bastion.ocp3.example.com
```

7.8.2. WebUI Access

Browse to **https://ocp3-heat-devs.ocp3.example.com:8443** and log in with user credentials from the LDAP/AD service. For this example, username = "openshift" and password is "password"

7.8.3. CLI Access

```
oc login ocp3-heat-devs.ocp3.example.com --username openshift --insecure-
skip-tls-verify
Authentication required for https://ocp3-heat-devs.ocp3.example.com:8443
(openshift)
Username: openshift
Password:
Login successful.

Using project "test-project".
```

CHAPTER 8. CONCLUSION

Red Hat solutions involving the OpenShift Container Platform are created to deliver a production-ready foundation that simplifies the deployment process, shares the latest best practices, and provides a stable highly available environment on which to run production applications.

This reference architecture covered the following topics:

- A completely provisioned infrastructure in OpenStack using both manual and Heat orchestration.
- Native integration with OpenStack services like Heat, Neutron, Cinder and Ceilometer
  - Cinder storage for `/var/lib/docker` on each node
  - A role assigned to instances that allows OCP to mount Cinder volumes
- Creation of applications
- Validating the environment
- Testing failover
- Auto-scaling OpenShift nodes with Heat and Ceilometer

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. REVISION HISTORY

Revision	Release Date	Author(s)
2.0	Wednesday May 3, 2017	Mark Lamourine, Roger Lopez
1.4	Wednesday January 4, 2017	Mark Lamourine
1.3	Thursday December 22, 2016	Mark Lamourine
1.2	Friday December 1, 2016	Mark Lamourine, Didier Wojciechowski
1.1	Wednesday November 16, 2016	Mark Lamourine, Ryan Cook, Scott Collier
1.0	Tuesday November 1, 2016	Mark Lamourine, Ryan Cook, Scott Collier
PDF generated by Asciidoctor PDF		
Reference Architecture Theme version 1.0		

APPENDIX B. DYNAMIC DNS SERVICE

The installation process for RHOCP depends on a reliable name service that contains an address record for each of the target instances. When installing RHOCP 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 dyamically by the orchestration system as the instances themselves are created.

RHOSP 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 RHOCP on RHOSP 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 RHOSP as a heat stack. The resulting stack contains a DNS master and two DNS slaves. The DNS is capable of accepting DNS queries and update requests for the RHOSP service. It is suitable for delegation to make the RHOCP service accessable to users.

The Ansible playbook that creates and configures the heat stack and configures the DNS service is part of the `openshift-ansible-contrib` repository on on Github. The RHOSP DNS service is in the `osp-dns` section for reference architectures.

B.1. AUTOMATED DEPLOYMENT OF A DNS SERVICE

NOTE

Before running this playbook, be sure to initialize the RHOSP credentials environment variables.

The script invocation below is a sample that shows the required input parameters. The callouts lead to details of each parameter.

Download DNS service playbook

```
git clone https://github.com/openshift/openshift-ansible-contrib
```

B.1.1. Configure the DNS Service

The Ansible playbook requires a set of inputs that define the characteristics of the DNS service. These inputs are provided as a YAML formatted data file. The input parameters are broadly divided into three groups:

- DNS Service settings
- OpenStack host parameters
- Software update subscription

In the sample file below the values must be updated for the target environment. Create a file from this template and replace the values as needed.

`dns-vars.yaml`

```
---
```

```
domain_name: ocp3.example.com                    1
contact: admin@ocp3.example.com
dns_forwarders: [10.x.x.41, 10.x.x.2]            2
update_key: "NOT A REAL KEY"                     3
slave_count: 2                                   4

stack_name: dns-service                          5
external_network: public                         6

image: rhel7                                     7
flavor: m1.small
ssh_user: cloud-user
ssh_key_name: ocp3

# NOTE: For Red Hat Enterprise Linux:            8
rhn_username: "rhnusername"
rhn_password: "NOT A REAL PASSWORD"
rhn_pool: "pool id string"
# Either RHN or Sat6
# sat6_hostname: ""
# sat6_organization: ""
# sat6_activationkey: ""
```

1. DNS Domain Name
This is the domain that the server manages. It must be a fully-qualified domain name. If it is delegated then it must be a proper subdomain of an established parent domain.

2. DNS Forwarders
This is a comma separated list of IP addresses enclosed in square brackets. Each must be the address of a DNS server that is capable of accepting forwarded DNS queries.

3. DNS Update Key
This is a DNS TSIG signing key. The key can be generated using **`ddns-confgen`** or **`rndc-confgen`**. See Generating an Update Key

4. Slave Count
The slave count defines the number of DNS slaves. A functional DNS service can have no slaves, but for legal delegation a service must have a master and at least 1 slave. The default is 2.

5. Stack Name
This value defines the name of the Heat stack that is created. It defaults to **`dns-service`**.

6. External Network Name
This value must be the name of the public or external network in the Red Hat OpenStack Platform service.

7. Instance Creation Parameters
These parameters set the characteristics of the instances which host the DNS service. The image and flavor must already be defined in the Red Hat OpenStack Platform service. The ssh_user is the user which allows login with the matching key.

8. Software Update Subscription
These parameters enable access to software updates. Use either the rhn_ or *sat6* values, but not both.

B.1.2. Deploy the DNS service

The DNS service is defined as an Ansible playbook. This playbook creates a Heat stack that implements the network and the host instances for the DNS service. It then applies the DNS service configuration to each instance to complete the process.

Deploy the DNS service

```
export ANSIBLE_HOST_KEY_CHECKING=False
ansible-playbook --private-key <keyfile> -e @dns-vars.yaml \ openshift-
ansible-contrib/reference-architecture/osp-dns/deploy-dns.yaml
```

Where the **<keyfile>** is the file name of the private key file matching the OSP keypair name. If needed, adjust the location of the root of the **openshift-ansible-contrib** repository.

The Ansible playbook runs in two distinct phases. The first phase creates the instances as a Heat stack. The second phase configures the instances once they are ready, registering for software repositories, installing and updating packages and finally applying the named configuration.

The playbook creates a dynamic inventory based on the result of the stack, adding the new instances. No inventory file is needed to start. A warning indicating either that the inventory is not present or that the host list is empty can be disregarded.

A successful deployment displays an Ansible "PLAY RECAP" which show a count of the applied plays. It includes the IP address of the DNS server instances.

Playbook Result

```
...
PLAY RECAP
*********************************************************************
10.0.x.187             : ok=16   changed=12   unreachable=0    failed=0
10.0.x.209             : ok=15   changed=11   unreachable=0    failed=0
10.0.x.210             : ok=15   changed=11   unreachable=0    failed=0
localhost                  : ok=7    changed=5    unreachable=0
failed=0
```

The first IP address is the master named. The following addresses are the slaves. Any address can be used for dynamic DNS updates, but the master is the recommended target.

B.2. GENERATING AN UPDATE KEY

DNS updates require a special key string that can be generated by **ddns-confgen** which is part of the **bind** RPM. The only element needed is the *secret* string.

Create a DNS update key

```
rndc-confgen -a -c update.key -k update-key -r /dev/urandom
cat update.key
key "update-key" {
    algorithm hmac-md5;
    secret "key string"; # don't use this
};
```

The significant part of the key output is the **secret** field. The value can be passed via command line or by setting and exporting the **DNS_UPDATE_KEY** environment variable for the deploy script that follows.

B.3. VERIFYING THE DNS SERVICE

Verify both the operation of the DNS service and the ability to update the zone contents using **nsupdate** are part of the *bind-utils* rpm.

The installation process reports the IP addresses of the DNS servers. The IP adresses can be retrieved using **openstack server list** command as well.

Verify that the queries work and that the nameserver NS and A records are present. Query the service with the IP address of the master and each of the slaves in turn to be sure that all return the same results.

B.3.1. Simple Query and Zone Contents

The query below returns the contents of the entire zone. It shows the SOA, NS and A records for the zone and the DNS servers.

```
dig @10.x.0.187 ocp3.example.com axfr

; <<>> DiG 9.9.4-RedHat-9.9.4-38.el7_3.3 <<>> @10.x.0.187 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 (1)
ocp3.example.com.   300 IN  NS  ns-master.ocp3.example.com. (2)
ocp3.example.com.   300 IN  NS  ns0.ocp3.example.com. (3)
ocp3.example.com.   300 IN  NS  ns1.ocp3.example.com. (4)
ns-master.ocp3.example.com. 300 IN  A   10.x.0.187 (5)
ns0.ocp3.example.com.   300 IN  A   10.x.0.209 (6)
ns1.ocp3.example.com.   300 IN  A   10.x.0.210 (7)
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: 10.19.114.187#53(10.x.0.187)
;; WHEN: Wed Jun 07 11:46:33 EDT 2017
;; XFR size: 8 records (messages 1, bytes 237)
```

(1) OCP SOA record

(2) NS record for the master server

(3) (4) NS records for the slave servers

(5) A record for the master server

(6) (7) A records for the slave servers

B.3.2. Dynamic Updates

The last test is to verify that dynamic updates work. The fragments below demonstrate adding, verifying and removing a test record. The actual IP address provided does not need to refer to a real host.

```
nsupdate -k update.key
server 10.x.0.187
zone ocp3.example.com
update add add-test.ocp3.example.com 300 A 1.2.3.4
send
quit
```

Verify that the new address is present:

```
dig @10.x.0.187 add-test.ocp3.example.com
```

Once the records are verified for accuracy, remove the test record.

```
nsupdate -k update.key
server 10.x.0.187
zone ocp3.example.com
update delete add-test.ocp3.example.com A
send
quit
```

B.4. RFCS

- RFC2137 Secure Domain Name System Dynamic Update
- RFC2535 Domain Name System Security Extensions

APPENDIX C. TUNING AND PATCHING RED HAT OPENSTACK PLATFORM FOR RED HAT OPENSHIFT CONTAINER PLATFORM

Red Hat OpenStack Platform may require tuning or patching for correct operation of Red Hat OpenShift Container Platform.

C.1. KEYSTONE TOKEN EXPIRATION

The Heat installation of Red Hat OpenShift Container Platform can take longer than one hour. The default Keystone token expiration time is 3600 seconds (one hour). If the stack creation token expires before the stack creation completes, Heat times-out the active task and fails the stack.

The Keystone token expiration time is set in the keystone configuration on all of the Red Hat OpenStack Platform controllers. The value that controls the expiration time is set in **`/etc/keystone/keystone.conf`**, in the **`[tokens]`** section.

Find and update the **`expiration`** value in the **`[tokens]`** section as indicated below and restart the httpd services on all of the Red Hat OpenStack Platform controller hosts.

`/etc/keystone/keystone.conf` fragment

```
...
[token]

 #
 # Options defined in keystone
 #

 # External auth mechanisms that should add bind information to
 # token e.g. kerberos, x509. (list value)
 #bind=

 # Enforcement policy on tokens presented to keystone with bind
 # information. One of disabled, permissive, strict, required
 # or a specifically required bind mode e.g. kerberos or x509
 # to require binding to that authentication. (string value)
 #enforce_token_bind=permissive

 # Amount of time a token should remain valid (in seconds).
 # (integer value)
 #expiration=3600
expiration=7200
...
```

restart httpd service

```
systemctl restart httpd
```

Details of Keystone identity service configuration can be find in the OpenStack Configuration Overview

C.2. HEAT SERVICE METADATA URL PUBLICATION

OSP10 Heat versions prior to **`openstack-heat-7.0.3-2.el7ost`** report metadata lookup URLs to **`oc-collect-config`** which point to the IPv4 localhost 127.0.0.1. The problem is described in BZ1452677 - overcloud heat metadata endpoints are incorrectly set to localhost .

For affected versions of openstack-heat the **`/etc/heat/heat.conf`** file must be modified to reflect the actual metadata server url. Specifically the values for **`heat_metadata_server_url`**, **`heat_waitcondition_server_url`** and **`heat_watch_server_url`** must contain the IP address of the actual controller host.

See solution overcloud heat metadata endpoints are incorrectly set to localhost for instructions. Replace the IP address in the example with the actual value for the target service.

- Bug Report: https://bugzilla.redhat.com/show_bug.cgi?id=1395139
- Solution: https://access.redhat.com/solutions/2868471 (requires subscription)

C.3. GNOCCHI USER PERMISSIONS

In OSP10 the metering, time series data and alarm systems have been split into three services. Prior to OSP10 these were all done by the Ceilometer service. Now the metering is still done by Ceilometer but the time-series data are stored by Gnocchi and alarms are managed by Aodh. Access to the time-series data must be restricted only to the owners of the servers and resources which generated them.

In OSP10, alarms created as Heat resources cannot access the Gnocchi data for the same project. This prevents autoscaling. The problem is described in BZ 1470134 - Unprivileged user can't access to its Gnocchi resources created by Ceilometer.

This is fixed in **`openstack-aodh-3.0.3-1.el7ost`**. Versions before this run but auto-scaling does not work. At the time of this writing the fix expected late summer 2017.

APPENDIX D. CONTRIBUTORS

1. Jan Provaznik, Heat template developer
2. Sylvain Baubeau, Heat template developer
3. Jason DeTiberus, content provider
4. Matthew Farrellee, content provider
5. Eduardo Minguez, content reviewer
6. Tomas Sedovic, Heat template developer
7. Ryan Cook, content reviewer
8. Scott Collier, content reviewer
9. Gan Huang, quality assurance testing