



Red Hat Reference Architecture Series

Deploying Red Hat OpenShift Container Platform 3 on Red Hat OpenStack Platform

Mark Lamourine, Ryan Cook, Scott Collier

Version 1.1 - 2016-11-16

Table of Contents

1. <i>Introduction</i>	2
1.1. Architecture	2
1.2. Internal Networks	4
1.3. Users and Views - the faces of OCP	4
1.4. Load-Balancers and Hostnames	6
1.5. Deployment Methods	7
2. <i>Deployment Process Overview</i>	8
2.1. Provisioning Steps	8
2.1.1. Configure OSP networks	8
2.1.2. Deploy instances and attach to networks	8
2.1.3. Update External Services (DNS, loadbalancer)	8
2.1.4. Configure and Tune instances	8
2.1.5. Define OCP Deployment Parameters - (Create Ansible Input Files)	9
2.1.6. Deploy OCP (execute Ansible)	9
2.1.7. Validate and Monitor	9
3. <i>Prerequisites and Preparation</i>	10
3.1. RHEL OpenStack Platform	10
3.2. OSP user account and credentials	10
3.3. RHEL 7 Base Image	12
3.4. SSH Key pair	12
3.5. Keystone: Increase token expiration	14
3.6. Neutron: <code>port_security</code> enabled	14
3.7. Public Network	15
3.8. DNS service	15
3.9. haproxy service	16
3.10. LDAP/AD service	16
3.11. Collected Information	16
4. <i>Manual Deployment</i>	19
4.1. OCP CLI clients	19
4.2. Script Fragments	19
4.2.1. Default Values and Environment Variables	19
4.2.2. Bastion Host Scripts and the Root User	20
4.3. Create Networks	20
4.4. Network Security Groups	22
4.4.1. Required Network Services	22
4.4.2. Bastion Host Security Group	22

4.4.3. Master Host Security Group	23
4.4.4. Node Host Security Group	24
4.4.5. Infrastructure Node Security Group	24
4.4.6. App Node Security Group	25
4.5. Host Instances	26
4.5.1. Host Names, DNS and cloud-init	26
4.5.2. Create Bastion Host Instance	29
4.5.3. Create Master Host instances.....	29
4.5.4. Cinder Volumes for <code>var/lib/docker</code>	29
4.5.5. Create the Node instances	30
4.5.6. Disable Port Security on Nodes	31
4.5.7. Adding Floating IP addresses.....	32
4.6. Publishing Host IP Addresses	33
4.7. Load-balancer	35
4.8. Host Preparation	37
4.8.1. Bastion Host	37
4.8.2. Prepare to access instances	39
4.8.3. instance Types and Environment Variables	40
4.8.4. Common Configuration Steps	42
4.8.5. Node Configuration Steps	44
4.9. Deploy OCP.....	46
4.9.1. Generate Ansible Inputs	46
4.9.2. Run Ansible Installer	50
5. Operational Management	52
5.1. Running Diagnostics	52
5.2. Checking the Health of ETCD.....	54
5.3. EmptyDir Quota	54
5.4. Yum Repositories	55
5.5. Console Access.....	55
5.5.1. Log into GUI console and deploy an application.....	55
5.5.2. Log into CLI and Deploy an Application	56
5.6. Explore the Environment.....	58
5.6.1. List Nodes and Set Permissions.....	58
5.6.2. List Router and Registry	59
5.6.3. Explore the Docker Registry.....	60
5.6.4. Explore Docker Storage.....	61
5.6.5. Explore Security Groups	64
5.7. Testing Failure	64

5.7.1. Generate a Master Outage	64
5.7.2. Observe the Behavior of ETCD with a Failed Master Node.....	64
6. Deploying Using Heat	66
6.1. Project Quotas	66
6.2. Benefits of Heat.....	67
6.3. Download the Heat Templates	67
6.4. Hostname Generation In Heat Stack	69
6.5. Creating the Stack.....	70
6.6. Observing Deployment	70
6.7. Verifying the OCP Service.....	71
6.7.1. Ops Access	71
6.7.2. WebUI Access.....	71
6.7.3. CLI Access	71
7. Conclusion	72
Appendix A: Revision History	73
Appendix B: Contributors.....	74

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
PO Box 13588
Research Triangle Park NC 27709 USA

Linux is a registered trademark of Linus Torvalds. Red Hat, Red Hat Enterprise Linux and the Red Hat "Shadowman" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

Ceph is a registered trademark of Red Hat, Inc.

UNIX is a registered trademark of The Open Group.

Docker and the Docker logo are trademarks or registered trademarks of Docker, Inc.

Intel, the Intel logo and Xeon are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. All other trademarks referenced herein are the property of their respective owners.

© 2016 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

The information contained herein is subject to change without notice. Red Hat, Inc. shall not be liable for technical or editorial errors or omissions contained herein.

Distribution of modified versions of this document is prohibited without the explicit permission of Red Hat Inc.

Distribution of this work or derivative of this work in any standard (paper) book form for commercial purposes is prohibited unless prior permission is obtained from Red Hat Inc.

The GPG fingerprint of the security@redhat.com key is: CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Send feedback to refarch-feedback@redhat.com

1. Introduction

This document describes a sample deployment of Red Hat OpenShift Container Platform (OCP) on Red Hat OpenStack Platform (RHOSP). While not a definitive configuration for all purposes, this guide highlights each of the steps in the installation and the interactions between OCP and OSP.

OCP provides compute services on-demand. It allows developers to create and deploy applications by delivering a consistent environment for both development and run-time life-cycle that requires no server management. It provides a consistent environment for both development and run-time. It enforces software management and aids in the development and operations management of services by providing visibility and stability for the users.

1.1. Architecture

The deployed OCP service consists of a single *bastion host*, three *master hosts* which runs the OCP master services, and five *node hosts* which run the Docker containers on behalf of its users. The node hosts are broken into two functional classes. Two *infra nodes* run internal OpenShift services; the *OpenShift Router* and the *Local Registry*. The remaining three *app nodes* host the user container processes.



The OCP service uses an external LDAP service for user identification and authentication.

1.2. Internal Networks

OCP establishes a network for communication between containers on separate nodes. In the [Architecture Diagram](#) this is designated the *tenant network*. This is actually two networks, one inside the other. The outer network is a Neutron network that carries traffic from node to node. The second network is encapsulated within that, with endpoints inside the containers.

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

The default mechanism is the *openshift_sdn*. This is a custom Open vSwitch (OVS) SDN which 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 OCP internal network, all OCP internode traffic carried over OpenStack Neutron undergoes *double encapsulation*, creating significant network performance issues.

In this document's design, flannel is used in place of the default *openshift_sdn*, 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 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. Nothing prevents a container from attempting to contact any IP address in the network space. That is: the network cannot be used to isolate containers in one application from those in another, hindering multitenancy.

In this reference architecture, the decision was made to give precedence to performance over tenant isolation.

1.3. Users and Views - the faces of OCP

OCP 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 OCP in different ways, performing distinct tasks. They see different faces of OCP.

When deploying OCP it is important to understand these faces 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 OCP service to create and manage their applications.

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

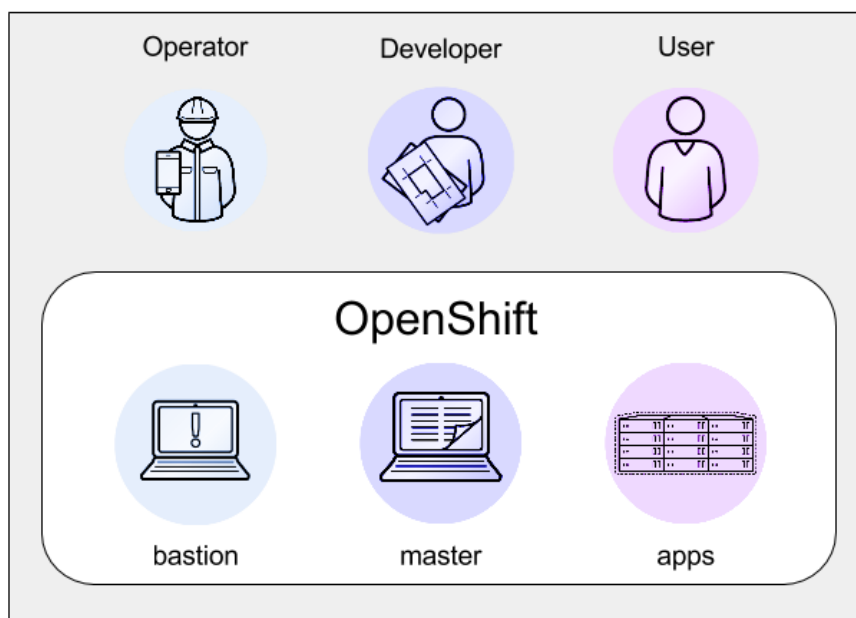


Figure 2. OCP User Views

When the deployment process is complete, each type of user is able to reach and use their resources. The OCP 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) which the user sees as their gateway to the OCP service.

Table 1. Sample User Portal Hostname

Role	Activity	Hostname
Operator	Monitoring and management of OCP service	<code>bastion.ocp3.example.com</code>
Application Developer	Creating, deploying and monitoring user services	<code>devs.ocp3.example.com</code>

Role	Activity	Hostname
Application User	Accessing apps and services to do business tasks	*.apps.ocp3.example.com

1.4. Load-Balancers and Hostnames

An external DNS service is used to make the service views visible. Most installations will have 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-availability 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 which multiplexes the traffic to multiple containers which make up a scaled application running inside OCP. The OpenShift Router itself can accept inbound traffic on multiple hosts as well. The load-balancer used for developer HA acts as the public view for the OCP 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, they can be configured into the DNS before 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 10.19.1114.132, the two DNS records can be added as follows:

Table 2. Load Balancer DNS records

IP Address	Hostname	Purpose
10.19.xxx.132	devs.ocp3.example.com	Developer access to OpenShift master web UI and REST API
10.19.xxx.132	*.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 means that OCP can 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: *10.19.114.132*. All of the traffic is forwarded to the OpenShift Routers which examine 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.

1.5. Deployment Methods

This document describes two ways of deploying OCP. The first is a "manual" process to create and prepare the run-time environment in OpenStack and then install OCP on the instances. The manual process is presented first so that the reader gets some understanding of the components and their relationships in the deployed service.

The second method uses **Heat** orchestration to deploy the OCP service. Heat is the OpenStack service which 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 OpenShift on OpenStack Heat templates, along with some customization parameters, define the OSP service and the Heat engine then executes the deployment process.

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

2. 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 using Ansible. The next section will detail the prerequisites and configuration values that are needed to execute these steps.

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

2.1. Provisioning Steps

These are the high level steps:

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

Here's what each step entails.

2.1.1. Configure OSP networks

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

2.1.2. Deploy instances and attach to networks

This configuration is made up of 8 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 so they can be reached from the outside.

2.1.3. Update External Services (DNS, loadbalancer)

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

2.1.4. Configure and Tune instances

The instances are created from stock RHEL images. They need to be registered for software updates

and they need minimal package sets so that they can participate in the deployment.

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.

2.1.5. Define OCP Deployment Parameters - (Create Ansible Input Files)

Using the names and IP addresses of the new instances, create the Ansible `inventory` which defines the layout of the service on the hosts.

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

2.1.6. Deploy OCP (execute Ansible)

Execute the `ansible-playbook` to deploy OCP from the bastion to the service instances

2.1.7. Validate and Monitor

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

3. *Prerequisites and Preparation*

This procedure depends on a number of services and settings that must be in place before beginning. Some of these are related to OSP itself and others are external components that the procedure touches or updates.

This procedure presumes

- RHEL OSP Platform 8 or later
- OSP *port security* controls must be enabled in Neutron service
- Increase Keystone token expiration
- OSP user account and credentials
- A RHEL 7.2 cloud image pre-loaded in Glance
- A RHEL update subscription credentials (user/password or Satellite)
- 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

3.1. RHEL OpenStack Platform

This procedure was developed and tested on Red Hat OpenStack Platform 8 (OSP8). While it may apply to other releases, no claims are made for installations on versions other than OSP8.

This procedure makes use of the following OSP service components:

- Keystone
- Nova
- Neutron
- Glance
- Cinder

3.2. OSP user account and credentials

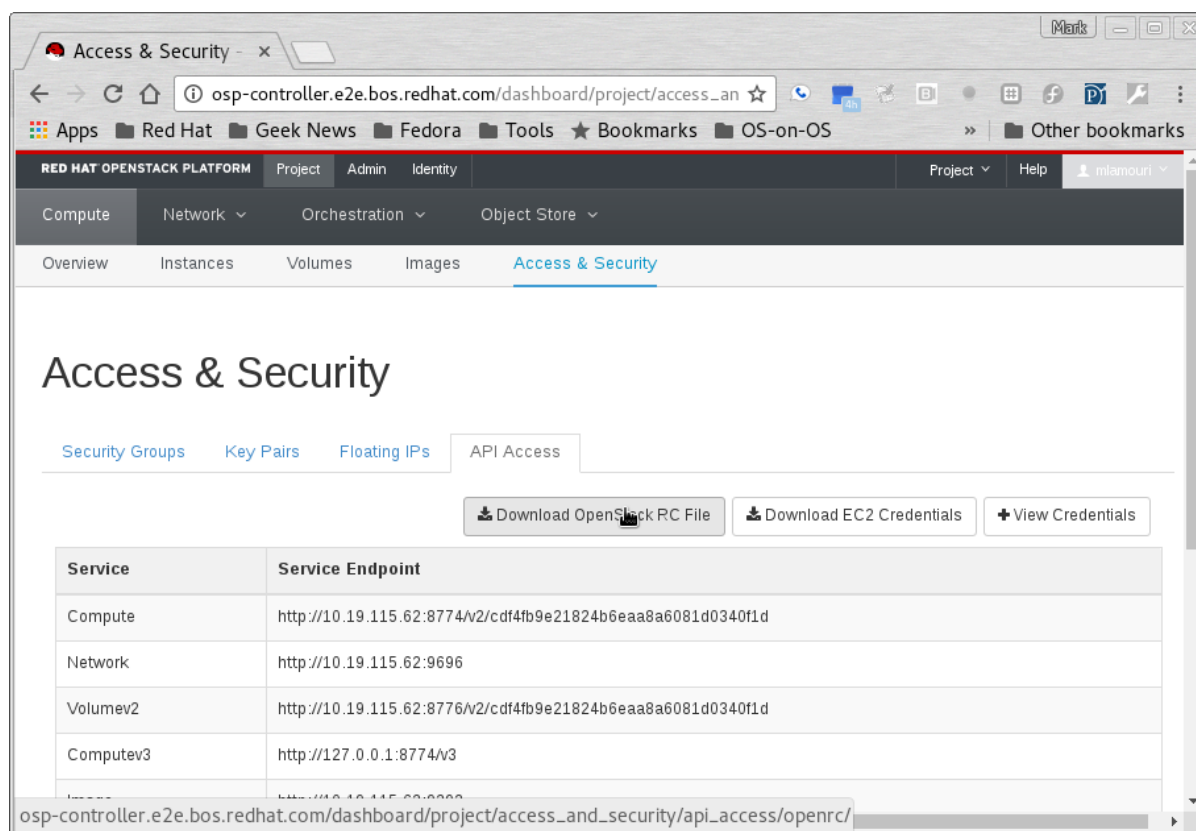
All of the OSP commands are executed using the CLI tools. These are easiest to use when the OSP login

credentials are saved as environment variables. The user needs an OSP 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 OSP credentials are commonly stored in a source file that sets the values of their shell environment.

You can get a script fragement like this from the OSP Keystone Web console.



A simpler form is here:

Listing 1. ks_ocp3

```
unset OS_AUTH_URL OS_USERNAME OS_PASSWORD OS_PROJECT_NAME OS_REGION
# replace the IP address with your OSP8 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_PROJECT_NAME=ocp3
export OS_TENANT_NAME=ocp3
export OS_REGION=RegionOne
```

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

```
source ./ks_ocp3
```

3.3. RHEL 7 Base Image

This procedure expects the instances to be created from the stock RHEL7.2 cloud image.

The RHEL7 cloud image is in the `rhel-guest-image-7` package which is in the `rhelosp-rhel-7-common` repository. Install the package on any RHEL system and load the image file into glance.

Listing 2. Load the RHEL7 image into Glance

```
sudo subscription-manager repos --enable rhelosp-rhel-7-common
sudo yum -y install rhel-guest-image-7
glance image-create \
    --name rhel72 \
    --disk-format qcow2 \
    --container-format bare \
    --visibility public \
    --file $(find /usr/share/rhel-guest-image-7 -type f -name \*.x86_64)
```

For the purposes of this procedure, the image is named `rhel72`.

3.4. SSH Key pair

OSP uses `cloud-init` to place an SSH public key on each instance as it is created to allow SSH access to the instance. OSP expects the user to hold the private key. On the RHEL image, the public key is installed under the `cloud-user` account.

Listing 3. Generate an SSH keypair

```
ssh-keygen -f ocp3_rsa -N ""
Generating public/private rsa key pair.
Your identification has been saved in ocp3_rsa.
Your public key has been saved in ocp3_rsa.pub.
The key fingerprint is:
SHA256:wfLUSBFfihZs8Tm6/PNjLZv+61DQ9avg1b/6q0a/tw testuser@testhost
The key's randomart image is:
+---[RSA 2048]-----+
|      += ...      |
|      o.+B..      |
|      . =Boo      |
|      +0.0  0  .  |
|      S  .  o  .  |
|      .   +  .  |
|      .  .+ +  |
|      o.+B=oo     |
|      =*B&OE      |
+-----[SHA256]-----+
```

This generates two files as noted above. The key files are the private and public keys respectively.

```
ocp3_rsa      # private key file
ocp3_rsa.pub  # public key file
```

The public key must be added to the OSP keypairs. OSP places that key on each instance as it boots so that the instance is accessible using SSH.

Listing 4. Add a keypair to Nova

```
nova keypair-add --pub-key ocp3_rsa.pub ocp3_key
```

and that the public key has been loaded into Nova and named *ocp3* then together these allow the user to log into an instance and get a shell.



SSH refuses to use private keys if the file permissions are too open. Be sure to set the permissions on *ocp3_rsa* to *0600* (user read/write).

The cloud-user account has permission to execute commands as root via *sudo*.

```
nova ssh -i ocp3_rsa cloud-user@<vm name>
```

3.5. Keystone: Increase token expiration



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

When creating a heat stack, the heat engine gets a Keystone token so that it can 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 OCP 3 heat stack deployment may take more than one hour. To ensure deployment completion, increase the expiration time on your OSP controllers and restart the `httpd` service.

Listing 5. Increase Keystone token expiration

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

3.6. Neutron: port_security enabled



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

The container to container communications is carried by *flannel*- a light weight virtual network service. Neutron *port security* controls prevent Flannel from working properly. The default configuration of OSP disables user control of *port_security*. To install OCP on OSP, Neutron must allow users to control the *port_security* settings on individual ports.

On the Neutron servers in the `/etc/neutron/plugins/ml2/ml2_conf.ini` file, make sure the file contains:

Listing 6. /etc/neutron/plugins/ml2/ml2_conf.ini

```
[ml2]
...
extension_drivers = port_security
```

Details of the configuration options for Neutron ML2 are in the [Neutron Controller Configuration](#) section of the [RDO Install Guide](#) for RHEL and CentOS.



Changes to this file require a restart of the Neutron services.

When port security is enabled, the network data structure displays a `port_security` field. This should be *true* for most networks, but is set to *false* for the tenant network.

3.7. Public Network

The OCP 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 OSP 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 which accepts direct inbound connections (the bastion, masters and infrastructure nodes) is 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 accessible hosts.

3.8. 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.



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 will be returned for any query which matches the suffix part. All these will return the same IP address:

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

Each host in the OCP 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 that it can 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. They 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.

3.9. haproxy service

A load-balancer provides a single view of the OpenShift master services and 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 will use must be in a DNS A record pointing to the haproxy server before installation. For applications, a wildcard DNS entry must also point to the haproxy host.

The configuration of the load balancer is created after all of the OCP instances have been created and floating IP addresses assigned.

3.10. LDAP/AD service

OCP 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 process requires the *base_dn* and a small set of parameters to allow LDAP authentication by the OCP service.

Table 3. LDAP credentials example

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

3.11. Collected Information

To summarize this table lists all of the information and settings that an operator will need to begin an OCP deployment on OSP.

Table 4. 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	
OCP Master hostname	devs.ocp3.example.com	Developer access to Web UI and API
OCP App sub-domain	*.apps.ocp3.example.com	All apps get a name under this
OSP User	user provided	User account for access to OSP
OSP Password	user provided	User password for OSP account
OSP Project	user provided	OSP project name for OCP deployment
OSP Roles (manual)	_member_	Roles required for the OSP user
OSP Roles (Heat)	_member_, heat_stack_owner	Roles required for the OSP user
RHEL Base Image	RHEL 7.2 guest	From rhel-guest-image-7 RPM or download from the Customer Portal
Glance Image Name	rhel72	
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	key text or file name? MAL
RHN Username	<username>	
RHN Password	<password>	
RHN Pool ID	<pool id>	

Parameter	Value	Comments
LDAP Host	dc.example.com	
LDAP User DNE	cn=users,dc=example,dc=com	
Preferred Username	sAMAccountName	
Bind DN	cn=openshift.cn=users,dc=example,dc=com	
Bind Password	<password>	

4. Manual Deployment

In a manual deployment of OCP, the operator interacts directly with OSP and then with the instances and finally with the Ansible playbooks for installing OCP. This method exposes the components and setup steps for the service to provide internal structure of the OCP service.

Manual deployment also has its limitations. It cannot easily respond to load by auto-scaling. There is no infrastructure "dashboard" or view which shows the instance and network components underneath the OCP service. It requires manual intervention for any changes, which may be spread over any or all of the instances inviting human error in maintenance and monitoring.

This reference design is meant as a learning tool and as a basis on which to build real-world installations.

4.1. OCP CLI clients

The manual deployment process starts with a series of steps that are executed using the OpenStack CLI tools. These CLI tools are available in the `rhel-7-server-openstack-8-rpms` repository.

Table 5. OSP CLI client packages

Client	Package Name	Service
Nova	python-novaclient	Virtual Machines
Neutron	python-neutronclient	Networks
Cinder	python-cinderclient	Block Storage
Glance	python-glanceclient	Image Storage

The client tools create and configure the infrastructure that hosts the OCP service.

4.2. Script Fragments

The procedure presented here consists of a set of CLI script fragments. These fragments are executed on the operator's workstation and then on the bastion host to complete the installation process.

The scripts which communicate with OSP are run on the operator's workstation. These scripts create the infrastructure and the instances to host the OCP service.

Once the bastion host exists the remainder of the work is run there. These scripts customize the host OS and prepare it to run Ansible to complete the deployment.

4.2.1. Default Values and Environment Variables

Several of the script fragments set default values for the purpose of providing an example. It real

deployments must replace those values with specific values for the target workload and environment.

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

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

4.2.2. Bastion Host Scripts and the Root User

Many of the commands which run on the bastion host must run as the `root` user, but the scripts do not need to be. The instance's user is `cloud-user`, a non-privileged user. Where the scripts run privileged commands, they call the commands with `sudo`.

4.3. Create Networks

This section describes the OSP network components that OCP needs, and how to create them.

OCP depends on three networks.

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

The public network must exist before starting the provisioning process. It must have a pool of floating IP addresses. The control network is tied to the public network through a router.

For this demonstration, the public network name is `public_network`

These two tables describe the internal OCP networks.

Table 6. Control Network Specification

Name	Value
Network Name	control_network
Subnet Name	control_subnet
Network Address	172.18.10.0
CIDR Mask	/24
DNS Nameserver	X.X.X.X
Router Name	control_router

Table 7. 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 values used below for your configuration.

Listing 7. Create Control Network

```
#!/bin/sh
# Set NAMESERVER to override
NAMESERVER=${NAMESERVER:-8.8.8.8}
neutron net-create control-network
neutron subnet-create --name control-subnet --dns-nameserver ${NAMESERVER} \
    control-network 172.18.10.0/24
neutron router-create control-router
neutron router-interface-add control-router control-subnet
neutron router-gateway-set control-router public_network
```

In the same way, create the tenant network. This network is for internal use only. It does not have a gateway to connect it to other networks.

Listing 8. Create Tenant Network

```
#!/bin/sh
neutron net-create tenant-network
neutron subnet-create --name tenant-subnet tenant-network 172.18.20.0/24
```

At this point, two networks and two subnets have been created. The control network is bound to a router that provides external access.

Listing 9. Verify control and tenant networks

```
nova net-list
```

ID	Label	CIDR
...		
957dae95-5b93-44aa-a2c3-37eb16c1bd2f	public_network	None
53377111-4f85-401e-af9d-5166a4be7f99	tenant-network	None
c913bfd7-94e9-4caa-abe5-df9dcd0c0eab	control-network	None
...		

4.4. Network Security Groups

OSP networking allows the user to define inbound and outbound traffic filters which 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 the host based filtering.

This section describes what ports and services are needed for each type of host and how to create the security groups in OSP. These will be used later when creating the instances.

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

4.4.1. Required Network Services

Each of the instance types in the OCP 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 that are used to test that the hosts are booted and are connected to their respective networks.

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

4.4.2. 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 service.

All of the other security groups accept SSH inbound from this security group.

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

Table 8. Bastion Security Group TCP ports

Port	Protocol	Purpose
22/TCP	SSH	Secure shell login
53/TCP	DNS	Internal name services
53/UDP	DNS	Internal name services

Listing 10. Create Bastion Security Group

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

# From
neutron security-group-rule-create bastion-sg \
    --protocol tcp --port-range-min 53 --port-range-max 53
```

4.4.3. Master Host Security Group

The OCP 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 which form the cluster. These servers must be accessible to each other within the cluster and to the node hosts.

These commands create and define master host network security group.

Table 9. Master Host Security Group Ports

Port	Protocol	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
4001/TCP	etcd	state changes
4789/UDP	SDN	pod to pod communications
8053/TCP	DNS	internal name services (3.2+)
8053/UDP	DNS	internal name services (3.2+)

Port	Protocol	Purpose
8443/TCP	HTTPS	Master WebUI and API
10250/TCP	kubernetes	kubelet communications
24224/TCP	fluentd	Docker logging

Listing 11. Create Master Security Group

```
#!/bin/sh
neutron security-group-create master-sg
neutron security-group-rule-create master-sg --protocol icmp
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 4001 8053 8443 10250 24224
do
  neutron security-group-rule-create master-sg \
    --protocol tcp --port-range-min $PORT --port-range-max $PORT
done

for PORT in 53 4789 8053 24224
do
  neutron security-group-rule-create master-sg \
    --protocol udp --port-range-min $PORT --port-range-max $PORT
done
```

4.4.4. Node Host Security Group

The node hosts execute the containers within the OCP service. The nodes are broken into two sets. The first set runs some OpenShift infrastructure, the OpenShift Router, and a local docker registry. The second set runs the developer applications.

4.4.5. Infrastructure Node Security Group

The infrastructure nodes run the OpenShift router and the local registry. It must accept inbound connections on the web ports which are then forwarded to their destinations.

Table 10. Infrastructure Node Security Group Ports

Port	Protocol	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
5000/TCP	HTTP	Docker Registry
10250/TCP	kubernetes	kubelet communications

Listing 12. Infrastructure Node Security Group

```
#!/bin/sh
neutron security-group-create infra-node-sg
neutron security-group-rule-create infra-node-sg --protocol icmp
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 5000 10250
do
  neutron security-group-rule-create infra-node-sg \
    --protocol tcp --port-range-min $PORT --port-range-max $PORT
done
```

4.4.6. App Node Security Group

The application nodes are isolated from random traffic. They only need to accept the **etcd** control connections and the **flannel** SDN for container communications.

Table 11. Application Node Security Group Ports

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

Listing 13. Create App Node Security Group

```
#!/bin/sh
neutron security-group-create app-node-sg
neutron security-group-rule-create app-node-sg --protocol icmp
neutron security-group-rule-create app-node-sg \
    --protocol tcp --port-range-min 22 --port-range-max 22 \
    --remote-group-id bastion-sg
neutron security-group-rule-create app-node-sg \
    --protocol tcp --port-range-min 10250 --port-range-max 10250
neutron security-group-rule-create app-node-sg \
    --protocol udp --port-range-min 4789 --port-range-max 4789
```

4.5. Host Instances

With the networks established we can begin building the virtual machines that host the OCP service.

This design includes eleven instances

- 1 Bastion host
- 3 OpenShift master hosts
- Openshift node hosts (7 total)
 - 3 Infrastructure nodes (infra)
 - 4 Application nodes (app)

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. This means that the command to create each instance is the same except for those three arguments. The common elements are presented first and then the differences.



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

4.5.1. Host Names, DNS and cloud-init

OCP uses the host names of the master and node hosts to identify each in the internal databases and to control and monitor the services on them. This makes it very important that the instance hostnames, the DNS hostnames and IP addresses are mapped correctly and consistently.

Without any inputs, OSP uses the Nova instance name as the hostname and the domain is *novalocal*. The bastion host's FQDN would be **bastion.novalocal**. This would be sufficient if OSP populated a DNS service with these names as the current cloud providers do. Each instance could find the IP addresses by name.

Using this name would require creating a zone in the external DNS service named '.novalocal'. Because the instance names are unique only within a project this risks name collisions. Instead create a subdomain for the control and tenant networks under the project domain `ocp3.example.com`.



Actually this is not a true subdomain. It is a *hostname suffix*. True domains are delegated. These names are not.

Table 12. Subdomains for OCP internal networks

Domain name	Description
<code>control.ocp3.example.com</code>	all interfaces on the control network
<code>tenant.ocp3.example.com</code>	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 13. Sample FQDNs

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

OSP provides a way for users to pass in information to be applied when an instance boots. The `--user-data` switch to `nova boot` makes the contents of the provided file available to the instance through `cloud-init`. `cloud-init` is a boot-time mechanism built into the RHEL operating system. 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 which sets the hostname and FQDN and the other is a short one line bourne 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.

Listing 14. Generate user-data for instance hostnames

```
#!/bin/sh
```

```
#set DOMAIN and SUBDOMAIN to override
```

```
DOMAIN=${DOMAIN:-ocp3.example.com}
```

```
BASTION=bastion
```

```
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_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="bastion.yaml"
```

```
#cloud-config
```

```
hostname: $1
```

```
fqdn: $1.$2
```

```
--=====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
```

```
}
```

```
for HOST in $ALL_HOSTS
```

```
do
```

```
    generate_userdata_mime ${HOST} ${DOMAIN} > user-data/${HOST}.yaml
```

```
done
```


4.5.2. Create Bastion Host Instance

This script creates a single instance which will be used to access and control all of the rest. Set the **DOMAIN** environment variable before invoking these scripts to customize the target domain for the entire deployment.

Listing 15. Create Bastion Host instance

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}
nova boot --flavor m1.small --image rhel72 --key-name ocp3 \
  --nic net-name=control-network --nic net-name=tenant-network \
  --security-groups bastion-sg \
  --user-data=user-data/bastion.yaml \
  bastion.${DOMAIN}
```

4.5.3. Create Master Host instances

This script creates three OCP master instances.

Listing 16. Create Master Host instances

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}
for HOSTNUM in 0 1 2 ; do
  nova boot --flavor m1.small --image rhel72 --key-name ocp3 \
    --nic net-name=control-network --nic net-name=tenant-network \
    --security-groups master-sg \
    --user-data=user-data/master-${HOSTNUM}.yaml \
    master-${HOSTNUM}.${DOMAIN}
done
```

4.5.4. 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:

Listing 17. Create Node Volumes

```
#!/bin/sh
VOLUME_SIZE=${VOLUME_SIZE:-8}
BASTION="bastion"
MASTERS="master-0 master-1 master-2"
INFRA_NODES="infra-node-0 infra-node-1"
APP_NODES="app-node-0 app-node-1 app-node-2"
ALL_NODES="$INFRA_NODES $APP_NODES"
ALL_HOSTS="$BASTION $MASTERS $ALL_NODES"

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

4.5.5. Create the Node instances

The `nova boot` 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.

Listing 18. A Function to get Cinder Volume IDs by name

```
#!/bin/sh
function vol-id-by-name() {
    # VOLNAME=$1
    cinder show $1 | grep ' id ' | awk '{print $4}';
}
```

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

Listing 19. Create the Infrastructure Node instances

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}
for HOSTNAME in infra-node-0 infra-node-1
do
    VOLUMEID=$(cinder show ${HOSTNAME}-docker | grep ' id ' | awk '{print $4}')

    nova boot --flavor m1.medium --image rhel72 --key-name ocp3 \
        --nic net-name=control-network --nic net-name=tenant-network \
        --security-groups infra-sg \
        --block-device source=volume,dest=volume,device=vdb,id=${VOLUMEID} \
        --user-data=user-data/${HOSTNAME}.yaml \
        ${HOSTNAME}.${DOMAIN}
done
```

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

Listing 20. Create the App Node instances

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}
for HOSTNAME in app-node-0 app-node-1 app-node-2
do
    VOLUMEID=$(cinder show ${HOSTNAME}-docker | grep ' id ' | awk '{print $4}')

    nova boot --flavor m1.medium --image rhel72 --key-name ocp3 \
        --nic net-name=control-network --nic net-name=tenant-network \
        --security-groups node-sg \
        --block-device source=volume,dest=volume,device=vdb,id=${VOLUMEID} \
        --user-data=user-data/${HOSTNAME}.yaml \
        ${HOSTNAME}.${DOMAIN}
done
```

4.5.6. 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 MAC and IP addresses and must send and receive packets on the port. Port security must be disabled on the ports which carry Flannel traffic.

This configuration runs Flannel on a private network which is bound to the *eth1* device on the node instances.

Listing 21. Disable Port Security on Tenant Network Ports

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}

BASTION="bastion"
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_NODES="$INFRA_NODES $APP_NODES"
ALL_HOSTS="$BASTION $MASTERS $ALL_NODES"

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 ${NAME}.${DOMAIN})
    PORT_ID=$(port_id_by_ip $TENANT_IP)
    neutron port-update $PORT_ID --no-security-groups --port-security-enabled=False
done
```

4.5.7. Adding Floating IP addresses

The Bastion host, the masters and the infrastructure nodes all need to be accessible from outside the control network. To make them accessible, give them an IP address on the public network.

Listing 22. Create and Assign Floating IP Addresses

```
#!/bin/sh
DOMAIN=${DOMAIN:-ocp3.example.com}

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_NODES="$INFRA_NODES $APP_NODES"
ALL_HOSTS="$BASTION $MASTERS $ALL_NODES"

for HOST in bastion $MASTERS $INFRA_NODES
do
    FLOATING_IP=$(nova floating-ip-create public_network | \
        grep public_network | awk '{print $4}')
    nova floating-ip-associate ${HOST}.${DOMAIN} ${FLOATING_IP}
done
```

4.6. Publishing Host IP Addresses

Though the users only interact with the OCP service through the public interfaces, the components need to be able to identify and communicate with each other. It's important to assign a name to each of the interfaces on an instance and then to publish the IP mappings so that processes on all the instances can find them. The simplest way is to update the DNS service used by the service.

All of the instances are connected to the **control-net** and the **tenant-net**. The bastion, masters and infra-nodes have floating IPs that are assigned on the **public_network** and must be published in the **ocp3.example.com** zone. The IP addresses on the control and tenant nets are placed in sub-domains named for these nets. For example:

To make DNS updates we need the IP address of the primary DNS server for the zone and a key that allows us to make updates. For **named** installations, the key is on the DNS server in **/etc/rndc.key**. Copy the contents of that file to a local file named **ocp3_dns.key**. Make sure you have **bind-utils** RPM installed. This contains the **/usr/bin/nsupdate** binary that is used to make dynamic updates.

Determine the hostname and FQDN for each instance and then get the floating IP it has been assigned.

Perform the following for all nodes:

Change the IP address of a DNS name

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

Table 14. Table Example Hostname/IP Address Mappings

ip-address	FQDN	Comment
10.x.0.151	bastion.ocp3.example.com	Provides deployment 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 from real-world deploy

Listing 23. 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
```

4.7. Load-balancer

The expectation in this section is that an existing load-balancer already exists. This reference environment is using a HAProxy instance with the following configuration. Please include the floating IPs for the masters to the load balancer.

The file that follows is a sample configuration for HAProxy.

Listing 24. /etc/htproxy/htproxy.conf

```
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.x.0.166:80 check
    server infra-node-1.ocp3.example.com 10.x.0.167:80 check

frontend main *:443
    default_backend            router443

backend router443
    balance source
    mode tcp
    server infra-node-0.ocp3.example.com 10.x.0.166:443 check
    server infra-node-1.ocp3.example.com 10.x.0.167:443 check

frontend main *:8443
    default_backend            mgmt8443

backend mgmt8443
    balance source
    mode tcp
```



```
server master-1.ocp3.example.com 10.x.0.168:8443 check
server master-2.ocp3.example.com 10.x.0.169:8443 check
server master-3.ocp3.example.com 10.x.0.170:8443 check
```

4.8. Host Preparation

At this point the host instances have been created, but they are not yet ready to run the `openshift-ansible` playbooks to deploy OCP. All of the instances need additional configuration to prepare them to run OCP.

Only the bastion host is accessible by SSH from outside the cluster. All of the work on the OCP hosts (masters and nodes) is done from the bastion host over the control network via SSH. The upcoming steps do all the work on the bastion host first, followed by customization of the OCP instances.

The OCP masters require software updates and accessibility to Ansible. The nodes need a little more work. Each node requires a Cinder volumes to hold the Docker container files and each node instance must be configured to mount that volume.

Bastion Customization

- Register for Software Updates
- Add `/usr/local/bin` to secure path for OpenShift Ansible
- Install ansible
- Install Openshift Ansible
- Copy private key to cloud-init user

Common Customization (master and node)

- Enable `sudo` via `ssh`
- Register for Software Repositories
- Enable eth1 (tenant network)
- Install and enable Openstack Config Agent

Node Customization

- Start and enable tenant network on boot
- Mount Cinder volume
- Install Docker

4.8.1. Bastion Host

The bastion host is the gateway to the rest of the instances for deployment and operations. The first step in deployment is to prepare the bastion host to act as a launching pad for the configuration of the other instances.

Logging into the Bastion Host

All of these steps are run from the bastion host. Most of them involve running SSH to execute the commands remotely on the masters and nodes. The neutron security groups limit SSH access inbound so that only the bastion host is allowed.

Log into the bastion host:

Listing 25. Log into Bastion Host

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

This example assumes you have the private key in a file named `./ocp3_rsa`. Replace that name with the file name of your private key if it is different.

Register for Software Updates

Red Hat software installation and updates require a valid subscription and registration. Access to the OCP and OSP repositories may also require a specific subscription pool.

Listing 26. Register For Software Updates (RHN)

```
subscription-manager register \  
  --username <username> \  
  --password <password>  
subscription-manager subscribe --pool <subscription pool id>
```

First, enable the standard server repositories:

Listing 27. Enable Required Repositories

```
subscription-manager repos --disable="*"
subscription-manager repos \  
  --enable="rhel-7-server-rpms" \  
  --enable=rhel-7-server-extras-rpms \  
  --enable=rhel-7-server-optional-rpms
```

The OCP software is in a specific product repository. Note the version number in the repository name.

Listing 28. Enable OCP Repositories

```
subscription-manager repos  
  --enable="rhel-7-server-ose-3.2-rpms"
```

The OSP repositories are required on all of the instances. The OSP repositories contain the `os-collect-config` package and its dependencies. This package contains utilities to collect and report the details of

an OpenStack instance which are used by the OpenShift Ansible playbooks.

Listing 29. Enable OSP8 Repos

```
subscription-manager repos \
  --enable="rhel-7-server-openstack-8-director-rpms" \
  --enable="rhel-7-server-openstack-8-rpms"
```

Prepare to run Ansible

- Update sudo secure path (check)
- Install Ansible and openshift-ansible

Update sudo secure path

Ansible installation can place files in `/usr/local/bin` that must be run as root. Update the `secure_path` value to allow commands there to be run by `sudo`.

Listing 30. Add `/usr/local/bin` to secure path

```
#!/bin/sh
sudo sed -i \
  '/secure_path = /s|=.*= /sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin|' \
  /etc/sudoers
```

Install Ansible and OpenShift Ansible

Installing the ansible components is just a matter of installing the `openshift-ansible-playbooks` RPM. This package calls in all of the dependancies required.

Listing 31. Install `openshift-ansible-playbooks`

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

4.8.2. Prepare to access instances

The Bastion host is now ready to run Ansible, but the OCP instances are not ready to accept it. The next few steps allow the `cloud-user` on the bastion host to access all of the instances. Ansible uses this ability to log into each instance and make the needed changes during the playbook run.

We'll install the OSP SSH key for the `cloud-user` account on the bastion so that it can log into each of the instances on the same account. Then we'll make it so that an SSH command is allowed to call `sudo` without opening a shell. This allows Ansible (and us) to execute commands as root on all of the instances directly from the bastion host.

- copy SSH key to bastion
- on each instance: allow sudo via SSH

Set the Host Key On the Bastion

A copy of the private key is needed on the bastion host to allow SSH access from there to the other instances. Log out of the bastion and copy the private key file to the bastion host and place it in `.ssh/id_rsa`. Adjust the bastion host name as needed.

Listing 32. Copy the SSH key to the bastion host

```
#!/bin/sh
scp -i ./ocp3_rsa ./ocp3_rsa cloud-user@bastion.ocp3.example.com:~/.ssh/id_rsa
```

SSH won't accept key files that are readable by other users. Log back into the bastion host and set the key file permissions

Listing 33. Restrict access to the SSH key on the bastion host

```
#!/bin/sh
chmod 600 ~/.ssh/id_rsa
```

4.8.3. instance Types and Environment Variables

There are three types of instances that require preparation. Some steps are executed on all of them, others just on one type. These steps are presented as a series of short script fragments. Some of these fragments contain environment variables to allow for customization and to allow them to be used to configure multiple hosts.

This section describes the variables and the conventions for their use.

The first set of variables define a several classes of hosts by name:

Listing 34. 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"
```

With these set it is possible to execute commands on multiple hosts by looping over the appropriate class.

Listing 35. Sample Loop Fragment

```
for H in $MASTERS
do
    ssh $H sudo <remote command>
done
```

Log into OCP instances

The OCP 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 was placed in the *.control* sub-domain. This means that the control network name for an instance is `<hostname>.control.ocp3.example.com`.

The hostname and FQDN are set, cloud-init puts the control network subdomain in the DNS search path in `/etc/resolv.conf`. It is possible to log from the bastion to any instance using its hostname only.

```
ssh master-0
```

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

Enable sudo via ssh

It is necessary to run commands on the instances as the root user. Root logins to the instances are disabled. Use the *cloud-user* account instead. This account has `sudo` enabled with no password so that it is possible to execute privileged commands. However, by default, running `sudo` commands via SSH is forbidden. It is necessary to relax this restriction so that it is possible to execute all of the commands directly from the bastion host. This is the one time that it is necessary to log onto each instance and execute a command directly.

For each instance, log in and modify the `/etc/sudoers` file like this:

Listing 36. Enable sudo via ssh

```
sudo sed -i "/requiretty/s/^/#/" /etc/sudoers
```

From this point on it is possible to execute commands as root on the instances directly from the bastion by combining `ssh` and `sudo`.

Listing 37. 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
```

In the examples that follow, unless otherwise indicated, execute the commands on each host using `ssh <hostname> sudo <command>`.

Each step should be run on the bastion and then on all OCP instances unless otherwise indicated.

4.8.4. Common Configuration Steps

These steps are executed on all of the OCP instances

Enable Software Repositories

These are the same steps which were executed on the bastion host. This time they are be executed on all of the instances using a shell loop.

Register all of the instances for software updates before attempting to install the OSC and OSP packages.

Listing 38. Register All instances for Software Updates

```
for H in $ALL_HOSTS  
do  
    ssh $H sudo subscription-manager register \  
        --username <username> --password <password>  
    ssh $H sudo subscription-manager subscribe --pool <pool id>  
done
```

Make sure that any default repositories are disabled and then re-enable the minimal standard set of repositories.

Listing 39. Enable Standard Software Repositories

```
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  
done
```

Now enable the OCP software repository. Update the version number to the current value if needed.

Listing 40. Enable OCP Software Repository

```
for H in $ALL_HOSTS
do
  ssh $H sudo subscription-manager repos --enable="rhel-7-server-ose-3.2-rpms"
done
```

Enable the OSP repositories to access the `os-collect` package and dependencies.

Listing 41. Enable OSP Software Repositories

```
for H in $ALL_HOSTS
do
  ssh $H sudo subscription-manager repos \
    --enable="rhel-7-server-openstack-8-director-rpms" \
    --enable="rhel-7-server-openstack-8-rpms"
done
```

Install OSP Data Collection

OpenShift Ansible on OSP depends on the ability to collect and update instance configuration and status information. Install the `os-*-config` packages and their dependencies on all instances.

Listing 42. Install OSP Data Collection RPMs

```
for H in $ALL_HOSTS
do
  ssh $H sudo yum -y install \
    os-collect-config \
    python-zaqarclient \
    os-refresh-config \
    os-apply-config \
    openstack-heat-templates # only on bastion and only for heat?
done
```

The OSP data collection software runs as a service. Enable and start it on all instances.

Listing 43. Enable and Start OSP Data Collection

```
for H in $ALL_HOSTS
do
  ssh $H sudo systemctl enable os-collect-config
  ssh $H sudo systemctl start --no-block os-collect-config
done
```

4.8.5. Node Configuration Steps

The OpenShift node instances require a few configuration updates that are not needed on the bastion and OpenShift master instances.

The nodes must be able to communicate over a private network, and they each get an external volume mounted to hold the Docker container and image storage.

Enable Tenant Network Interface

The containers within the OCP service communicate over a private back-end network, called the *tenant network*. The node instances are configured with a second interface to carry this private traffic. OCP uses Flannel to route the traffic within and between nodes.

The file below configures that second interface on each node. OSP assigns each node an address on that subnet and provides DHCP so that the hosts can configure the IP address automatically.

This file is copied to each instance and then the interface is started. When OCP is installed the configuration file must specify this interface name for Flannel.

Listing 44. 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 short script below copies the `ifcfg-eth1` file to the hosts and brings interface up.

Listing 45. 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
```

Add Docker Storage

The 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.

Install Docker

Install Docker but do not enable it yet.

Listing 46. Install Docker

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

Configure and enable Linstance Metadata Daemon

The Linstance Metadata daemon is used to discover and map the Cinder volume into the device file system. Install and configure the Linstance Metadata daemon.

Listing 47. Enable Linstance updates

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

Set the Docker Storage Location

The Docker storage configuration is set by a file named `/etc/sysconfig/docker-storage-config`. This file specifies that the Docker storage will be on the device `/dev/vdb` and will create an Linstance volume group called `docker-vg`.

Listing 48. docker-storage-setup

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

This script fragment copies the storage configuration to all nodes then executes the Docker script to configure the storage.

Listing 49. Copy and Install Storage Setup Config

```
#!/bin/sh
for H in $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
```

Enable and Start Docker

Listing 50. Enable and Start Docker

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

4.9. Deploy OCP

The OpenShift Ansible playbooks require three input files.

- `inventory`
- `group_vars/OSv3.yml`
- `ansible.cnf`

The *inventory* is an **INI** formatted set of host names, and groupings of hosts which have 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 lists the hosts that are members of that group.

The `OSv3.yml` file is a **YAML** formatted data file. It defines a structured set of key/value pairs which are used to configure all of the OSP instances.

The `ansible.cfg` file is an **INI** formatted file which defines Ansible run-time customizations.

4.9.1. Generate Ansible Inputs

There are three input files for the OpenShift Ansible scripts:

1. `inventory`
2. `group_vars/OSv3.yml`

3. `ansible.cfg`

The `inventory` and `OSv3.yml` file must be edited to suit the deployment environment.

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 will get the same configuration.

Listing 51. Ansible control file: `inventory`

```
[OSv3:children]
infra
masters
nodes
etcd

[infra]
localhost

[masters]
master-0.control.ocp3.example.com openshift_hostname=master-0.control.ocp3.example.com
openshift_public_hostname=master-0.ocp3.example.com
master-1.control.ocp3.example.com openshift_hostname=master-1.control.ocp3.example.com
openshift_public_hostname=master-1.ocp3.example.com
master-2.control.ocp3.example.com openshift_hostname=master-2.control.ocp3.example.com
openshift_public_hostname=master-2.ocp3.example.com

master-1.control.ocp3.example.com
master-2.control.ocp3.example.com

[masters:vars]
openshift_schedulable=true
openshift_router_selector="region=infra"

[etcd]
master-0.control.ocp3.example.com
master-1.control.ocp3.example.com
master-2.control.ocp3.example.com

[nodes]
infra-node-0.ocp3.example.com openshift_node_labels="{ 'region': 'infra', 'zone':
'default' }" openshift_hostname=infra-node-0.ocp3.example.com
openshift_public_hostname=infra-node-0.ocp3.example.com openshift_ip=192.168.0.50
infra-node-1.ocp3.example.com openshift_node_labels="{ 'region': 'infra', 'zone':
'default' }" openshift_hostname=infra-node-1.ocp3.example.com
openshift_public_hostname=infra-node-1.ocp3.example.com openshift_ip=192.168.0.50
infra-node-2.ocp3.example.com openshift_node_labels="{ 'region': 'infra', 'zone':
```

```
'default'}" openshift_hostname=infra-node-2.ocp3.example.com
openshift_public_hostname=infra-node-2.ocp3.example.com openshift_ip=192.168.0.50

app-node-0.ocp3.example.com openshift_node_labels="{ 'region': 'primary', 'zone':
'default'}" openshift_hostname=app-node-0.ocp3.example.com openshift_public_hostname=app-
node-0.ocp3.example.com openshift_ip=192.168.0.48
app-node-1.ocp3.example.com openshift_node_labels="{ 'region': 'primary', 'zone':
'default'}" openshift_hostname=app-node-1.ocp3.example.com openshift_public_hostname=app-
node-1.ocp3.example.com openshift_ip=192.168.0.48
app-node-2.ocp3.example.com openshift_node_labels="{ 'region': 'primary', 'zone':
'default'}" openshift_hostname=app-node-2.ocp3.example.com openshift_public_hostname=app-
node-2.ocp3.example.com openshift_ip=192.168.0.48
app-node-3.ocp3.example.com openshift_node_labels="{ 'region': 'primary', 'zone':
'default'}" openshift_hostname=app-node-3.ocp3.example.com openshift_public_hostname=app-
node-3.ocp3.example.com openshift_ip=192.168.0.48

[ dns ]
localhost
```

The sample above includes examples of both group variables and host variables in-line. These are all simple key/value pairs. For more complex data structures it is easier to place the settings in an external file.

group_vars and the OSv3.yml file

Ansible checks two subdirectories under the directory which contains the inventory file: **group_vars/** and **host_vars/**. The files there are YAML format, using the **.yml** suffix. The file name must match the name of the section to which the values apply.

The following defines the global values in the OSv3 section and all of its children in a file named **group_vars/OSv3.yml**.

Listing 52. Global configuration values: `group_vars/OSv3.yml`

```

deployment_type: openshift-enterprise
osm_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 ②
openshift_master_cluster_method: native

openshift_override_hostname_check: true
openshift_set_node_ip: true
openshift_use_dnsmasq: false

# 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 ③
openshift_cloudprovider_openstack_username: <username> ④
openshift_cloudprovider_openstack_password: <password> ⑤
openshift_cloudprovider_openstack_tenant_name: <tenant name> ⑥
openshift_cloudprovider_openstack_region: RegionOne ⑦

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 will be named `<name>.apps.ocp3.example.com`. Adjust as needed. This name must resolve to the IP address of the app load balancer.
- ② Developers will access the WebUI and API at this name. Adjust as needed. This name must resolve to the IP address of the master load balancer.

- ③ The URL of the OSP API service. Adjust as needed.
- ④ The OSP username.
- ⑤ The OSP password.
- ⑥ The OSP project which contains the OCP service.
- ⑦ The OSP deployment region.
- 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 needed.

ansible.cfg

The final file is `ansible.cfg`. This file defines Ansible global customization values. Most important here are the `remote-user` and `become` values. These indicate that Ansible should log into instances as the `cloud-user` user and then use `sudo` to execute scripts with root privileges.

Listing 53. Ansible customization: ansible.cfg

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

[privilege_escalation]
become = True

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=900s -o GSSAPIAuthentication=no
control_path = /var/tmp/ssh-%h-%r
#pipelining = True
```

4.9.2. Run Ansible Installer

At this point all of the configuration inputs are defined. It is finally time to execute the OpenShift Ansible playbooks and deploy OCP.

```
export ANSIBLE_ROLES_PATH=/usr/share/ansible/openshift-ansible/roles
export ANSIBLE_HOST_KEY_CHECKING=False

ansible-playbook -vvv --inventory inventory \
  /usr/share/ansible/openshift-ansible/playbooks/byo/config.yml
```

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

```
ansible masters -i /var/lib/ansible/inventory -m shell -a 'iptables -A DOCKER -p tcp -j ACCEPT'
ansible nodes -i /var/lib/ansible/inventory -m shell -a 'iptables -A DOCKER -p tcp -j ACCEPT'
```

5. Operational Management

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

5.1. 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-e2e-bos-redhat-com:8443/system:admin'
```

```
[Note] Performing systemd discovery
```

```
[Note] Running diagnostic: ConfigContexts[default/devs-ocp3-e2e-bos-redhat-com:8443/system:admin]
```

```
    Description: Validate client config context is complete and has connectivity
```

```
Info: The current client config context is 'default/devs-ocp3-e2e-bos-redhat-com:8443/system:admin':
```

```
    The server URL is 'https://devs.ocp3.example.com:8443'
```

```
    The user authentication is 'system:admin/devs-ocp3-e2e-bos-redhat-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



The warnings will not cause issues in the environment

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

5.2. 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
```

5.3. EmptyDir Quota

Quotas are set on ephemeral volumes within pods to prohibit a pod from becoming too large and impacting the node. There are three places where the quota information is required to be set. When locations are not set a pod has the ability to grow as large as the underlying filesystem will allow. The required modifications are set by Ansible roles below will be the specific Ansible role that defines the parameters along with the locations on the nodes in which the parameters are set.

openshift-emptydir-quota



This is not set the same way in Heat

This role creates a xfs partition on the **/dev/xvdc** block device, adds an entry in fstab, and mounts the volume with the option of gquota. If gquota is not set the OpenShift node will not be able to start with the "perFSGroup" parameter defined below. This disk and configuration is done on the infrastructure and application nodes. The configuration is not done on the masters due to the master nodes being unschedulable.

```
# vi /etc/fstab
/dev/xvdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0
```

docker-storage-setup

The role `docker-storage-setup` tells the Docker service to use `/dev/xvdb` 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
```

5.4. Yum Repositories

In section 2.3 Required Channels the specific repositories for a successful OpenShift installation were defined. All systems except for the bastion host should have the same subscriptions. To verify those subscriptions match those defined in Required Channels perform the following. The repositories below are enabled during the `rhsm-repos` playbook during the installation. The installation will be unsuccessful if the repositories are missing from the system.

```
sudo yum repolist
Loaded plugins: search-disabled-repos
repo id                                repo name                                status
!rhel-7-server-extras-rpms/x86_64      Red Hat Enterprise Linux 7 Se          297
!rhel-7-server-optional-rpms/7Server/x86_64 Red Hat Enterprise Linux 7 Se    8,887
!rhel-7-server-ose-3.2-rpms/x86_64      Red Hat OpenShift Enterprise          666
!rhel-7-server-rpms/7Server/x86_64      Red Hat Enterprise Linux 7 Se   11,375
repolist: 21,225
```



All rhui repositories are disabled and only those repositories defined in the Ansible role `rhsm-repos` are enabled.

5.5. Console Access

This section will cover 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.

5.5.1. 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 will be 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 application to confirm functionality.

5.5.2. 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 with the cli](#).

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
will be created
      * The resulting image will be pushed to image stream "php:latest"
      * This image will be deployed in deployment config "php"
      * Port 8080/tcp will be load balanced by service "php"
      * Other containers can 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 your app.

$ 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.

5.6. Explore the Environment

5.6.1. 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			

5.6.2. List Router and Registry

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



Perform the following steps from a workstation.

```
# oc project default
```

```
# oc get all
```

```
# oc status
```

In project default on server <https://prod-openshift-master.prod-aws.sysdeseng.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.

5.6.3. 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 and mapped to the Amazon S3 buckets for storage. Using Oc describe should help explain how HA works in this environment.



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.
```



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
```

5.6.4. Explore Docker Storage

This section will explore the Docker storage on an infrastructure node.



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: AVU0: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
```

5.6.5. 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.



Perform the following steps from the OpenStack web console.

On the main **AWS** console, click on **EC2**. Next on the left hand navigation panel select the **Security Groups**. Click through each group 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 allowed.

5.7. Testing Failure

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

5.7.1. Generate a Master Outage

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

Stop one of the master instances:

Listing 54. 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 `openshift-master.sysdeseng.com`. At this point, the cluster is in a degraded state because only 2/3 master nodes are running, but complete functionality remains.

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

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



Run this commands on one of the active master servers.

Listing 55. 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.



Run this command from a workstation.

Listing 56. 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 |
```

6. 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.

Heat is the OpenStack orchestration system.

Orchestration allows the end user to describe the system to their specifications 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.



The Heat template software is pre-release and subject to rapid development and change. This software is unsupported and any questions or queries should be directed to the Github issues for the repository rather than as Red Hat bugs. Support will be available when the templates have been released for General Availability.

6.1. 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 15. 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.



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.

6.2. Benefits of Heat

Manual installation can only create a static service configuration. Any changes, such as adding app nodes to support greater demand, must also be done manually.

Heat orchestration and the `heat-engine` offer a way to detect and respond to loading on the app nodes, automatically adding or removing nodes as demand requires. The templates integrate the OCP service with the Ceilometer monitoring service in OSP. When Ceilometer indicates an increase in load, it signals the Heat stack and Heat can add nodes as needed.

6.3. Download the Heat Templates

At the time of this writing the OCP heat templates have not been released. They are available from the Github project repository:

Github Repository

<https://github.com/redhat-openstack/openshift-on-openstack>

Clone a copy of the Git workspaces like this:

Listing 57. Clone Git Repository

```
git clone https://github.com/redhat-openstack/openshift-on-openstack
cd openshift-on-openstack
git checkout -b v0.9.2 v0.9.2
```

In this example we are using the pre-release tag 0.9.2.

It is easiest to create a Heat 'stack' using the OSP `heat` CLI tool. These are available in the `rhel-7-server-openstack-8-rpms` repository. Be sure to enable that repository before trying to install the OSP CLI tools.

Listing 58. Install Heat Client

```
sudo subscription-manager repos --enable rhel-7-server-openstack-8-rpms
sudo yum -y install python-heatclient
```

Heat uses a YAML input file to customize the stacks it creates.

Listing 59. openshift_parameters.yaml

```
# Invoke:
# heat stack-create ocp3-heat -f openshift.yaml
#     -e openshift_parameters.yaml \
parameters:
  # OpenStack user credentials - Adjust as needed
```

```
os_auth_url: http://10.19.115.62:5000/v2.0 ①
os_username: <username> ②
os_password: <password> ②
os_region_name: default
os_tenant_name: <project name> ②

# Red Hat Subscription information - Adjust as needed
rhn_username: "<username>" ③
rhn_password: "<password>" ③
rhn_pool: '<pool id containing openshift>' ③

external_network: public_network
dns_nameserver: 10.19.114.130,10.16.36.29,10.11.5.19 ④

deployment_type: openshift-enterprise
ose_version: "3.2" ⑤

domain_name: "ocp3.example.com" ⑥
lb_hostname: "devs" ⑦
loadbalancer_type: external

ssh_key_name: ocp3
ssh_user: cloud-user

bastion_hostname: "bastion"
bastion_image: rhel72
master_hostname: "master"
master_image: rhel72
master_count: 3
master_docker_volume_size_gb: 5
infra_hostname: "infra"
infra_image: rhel72
infra_count: 3
infra_docker_volume_size_gb: 5
node_hostname: "node"
node_image: rhel72
node_count: 2
node_docker_volume_size_gb: 5

openshift_sdn: flannel

deploy_router: true
deploy_registry: true

parameter_defaults:
# # Authentication service information
ldap_url: "ldap://ad.example.com.com:389/cn=users,dc=example,dc=com?sAMAccountName" ⑧
ldap_preferred_username: "sAMAccountName"
```



```

ldap_bind_dn: "cn=openshift,cn=users,dc=example,dc=com" ⑨
ldap_bind_password: "password" ⑩
ldap_insecure: true

# Adjust location
# Package location: file:///usr/share/openshift-on-openstack/
resource_registry: ⑪
  OOShift::LoadBalancer: openshift-on-openstack/loadbalancer_external.yaml
  OOShift::ContainerPort: openshift-on-openstack/sdn_flannel.yaml
  OOShift::IPFailover: openshift-on-openstack/ipfailover_keepalived.yaml
  OOShift::DockerVolume: openshift-on-openstack/volume_docker.yaml
  OOShift::DockerVolumeAttachment: openshift-on-openstack/volume_attachment_docker.yaml
  OOShift::RegistryVolume: openshift-on-openstack/registry_ephemeral.yaml

```

- ① Replace with the IP or hostname of the OSP controller service.
- ② Replace the username, password and tenant/project name with the user's credentials
- ③ Replace with the RHN/Access credentials for software install and updates
- ④ Replace with the local DNS server IP addresses for name resolution
- ⑤ Replace to instal a more recent release of OCP
- ⑥ Set the suffix of all DNS names for this OCP service
- ⑦ Set the hostname label for developer access to the OCP masters. This defines the loadbalancer FQDN.
- ⑧ The LDAP auth search path: Host, port, group DN and user name field.
- ⑨ This is an LDAP user that has permission to query for other user information
- ⑩ The password for the LDAP bind dn user.
- ⑪ This assumes the `heat` command is run from the directory above the `openshift-on-openstack` git workspace. Adjust the file path as needed. The file paths can be valid `file::` or `http:` URLs.

6.4. 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`: `mystack`
- `lb_hostname`: `devs`

Table 16. OCP Service Host Names

host	FQDN
suffix	ocp3.example.com
master LB	mystack-devs.ocp3.example.com
application LB	*.cloudapps.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 distinguish them.

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

Table 17. OCP instance names in OSP Nova

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

6.5. Creating the Stack

Listing 60. Create the Heat Stack

```
heat stack-create ocp3-heat \
  --timeout 120 \
  -e openshift_parameters.yaml \
  -f openshift-on-openstack/openshift.yaml
```

6.6. Observing Deployment

Listing 61. Observe the Stack Creation

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

6.7. Verifying the OCP Service

6.7.1. Ops Access

Log into the bastion host through `nova ssh`

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

6.7.2. WebUI Access

Browse to <https://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"

6.7.3. 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".
```

7. 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 your 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 will allow 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)
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: Contributors

1. Jan Provaznik, Heat template developer
2. Sylvain Baubeau, Heat template developer
3. Jason DeTiberus, content provider
4. Matthew Farrellee, content provider

