Reference Architectures 2017

Deploying and Managing OpenShift Container Platform 3.6 on VMware vSphere

Last Updated: 2017-11-27

Reference Architectures 2017 Deploying and Managing OpenShift Container Platform 3.6 on VMware vSphere

Davis Phillips

Annette Clewett
refarch-feedback@redhat.com

Legal Notice

Abstract

The purpose of this reference architecture is to provide a step-by-step on how to deploy and manage Red Hat OpenShift Container Platform 3.6 on VMware vSphere. Additionally, the document covers vSphere dynamic provisioning and Gluster for persistent container storage.

Table of Contents

COMMENTS AND FEEDBACK 5

CHAPTER 1. EXECUTIVE SUMMARY 6

CHAPTER 2. COMPONENTS AND CONFIGURATION 7
2.1. REFERENCE IMPLEMENTATION HARDWARE CONFIGURATION 8
2.2. VMWARE VCENTER AND VSPHERE DETAILS 9
2.3. VIRTUAL MACHINE INSTANCE DETAILS 9
2.4. HAPROXY LOAD BALANCER DETAILS 10
2.5. SOFTWARE VERSION DETAILS 10
2.6. REQUIRED CHANNELS 11
2.7. TOOLING PREREQUISITES 11
2.7.1. Git Repository 11
2.7.1.1. GitHub Repositories 11
2.7.1.2. Directory Setup 12
2.7.2. Ansible by Red Hat Setup 12
2.8. NETWORK COMPONENTS 15
2.8.1. DNS (Domain Name Server) Configuration 15
2.8.2. Hosted Zone Setup 16
2.8.3. Authentication 16
2.8.4. NFS (Network File System) Server 17
2.8.5. Load Balancer 18
2.9. VMWARE VCENTER PREREQUISITES 18
2.9.1. Networking 19
2.9.2. vCenter Shared Storage 19
2.9.3. Resource Pool, Cluster Name and Folder Location 20
2.9.4. SSH (Secure Shell) Prerequisite 20
2.9.4.1. SSH Configuration 20
2.9.5. VMware Template 21
2.9.6. Preparation Script 21
2.10. DYNAMIC INVENTORY 22
2.11. NODES 24
2.11.1. Master nodes 24
2.11.2. Infrastructure nodes 24
2.11.3. Application nodes 24
2.11.4. Node labels 25
2.12. RED HAT OPENSHIFT PODS 25
2.13. OPENSHIFT SDN 25
2.14. ROUTER 25
2.15. REGISTRY 26
2.16. OPENSHIFT METRICS 26

CHAPTER 3. PROVISIONING THE INFRASTRUCTURE 27
3.1. PROVISIONING THE INFRASTRUCTURE WITH ANSIBLE BY RED HAT 27
3.1.1. Authentication Prerequisite 27
3.1.1.1. Create a Red Hat OpenShift lightweight directory access protocol (LDAP) BIND user account 27
3.2. OCP-ON-VMWARE.PY - VARIABLES 28
3.2.1. Sample Red Hat OpenShift Install Variables 30
3.3. OCP-ON-VMWARE.PY - INVENTORY 31
3.3.1. Sample Red Hat OpenShift Inventory Variables 33
3.3.2. VMware Configuration Variables 34
3.3.2.1. VMware Authentication Variables 35

3.3.2.2. Remaining VMware Deployment Variables 35
3.3.3. Deploying the Environment 36
3.3.3.1. ocp-on-vmware.py - deployment 36
3.3.3.2. Greenfield Deployment 36
3.3.3.3. Brownfield Deployment 37
3.3.3.4. Post Ansible Deployment 39
3.3.3.4.1. Registry Console Selector (Optional) 42
3.4. POST PROVISIONING RESULTS 42

CHAPTER 4. OPERATIONAL MANAGEMENT 45
4.1. VALIDATING THE DEPLOYMENT 45
4.2. GATHERING HOSTNAMES 45
4.3. RUNNING DIAGNOSTICS 46
4.4. CHECKING THE HEALTH OF ETCD 47
4.5. EXAMINING DEFAULT NODE SELECTOR 48
4.6. MANAGING OF MAXIMUM POD SIZE 48
4.7. CHECKING THE YUM REPOSITORIES 49
4.8. CONSOLE ACCESS 50
4.9. LOGGING INTO GUI (GRAPHICAL USER INTERFACE) CONSOLE AND DEPLOY AN APPLICATION 50
4.10. LOGGING INTO THE CLI (COMMAND-LINE INTERFACE) AND DEPLOY AN APPLICATION 50
4.11. EXPLORING THE ENVIRONMENT 52
4.11.1. Listing Nodes and Set Permissions 52
4.11.2. Listing Router and Registry 53
4.11.3. Exploring the Docker Registry 53
4.11.4. Exploring Docker Storage 55
4.12. TESTING FAILURE 56
4.12.1. Generating a Master Outage 56
4.12.2. Observing the Behavior of etcd with a Failed Master Node 57
4.12.3. Generating an Infrastruture Node outage 57
4.12.3.1. Confirming Application Accessibility 57
4.12.3.2. Confirming Registry Functionality 58
4.12.3.3. Get Location of Router and Registry. 59
4.12.3.4. Initiate the Failure and Confirm Functionality 60
4.13. UPDATING THE OPENSHIFT DEPLOYMENT 60
4.13.1. Performing the Upgrade 60
4.13.2. Upgrading and Restarting the OpenShift Environment (Optional) 60
4.13.3. Specifying the OpenShift Version when Upgrading 61

CHAPTER 5. PERSISTENT STORAGE CONCEPTS 62
5.1. STORAGE CLASSES 62
5.2. PERSISTENT VOLUMES 62

CHAPTER 6. PERSISTENT STORAGE OPTIONS 63
6.1. VSPHERE CLOUD PROVIDER CONFIGURATION 63
6.1.1. vSphere Storage Class 63
6.1.2. VMDK Dynamic provisioning 64
6.2. CREATING AN NFS PERSISTENT VOLUME 65
6.2.1. Creating an NFS Persistent Volumes Claim 66
6.2.2. Deleting a Persistent Volumes Claim 66
6.3. CONTAINER-NATIVE STORAGE OVERVIEW 66
6.3.1. Prerequisites for Container-Native Storage 67
6.3.2. Deployment of CNS Infrastructure 67
6.3.3. Firewall Prerequisites 69
6.4. CNS INSTALLATION OVERVIEW 69

6.4.1. Creating CNS Project 70
6.4.2. Gluster Deployment Prerequisites 70
6.4.3. Deploying Container-Native Storage 70
6.4.4. Exploring Heketi 74
6.4.5. Store the Heketi Secret 75
6.4.6. Creating a Storage Class 75
6.5. CREATING A PERSISTENT VOLUME CLAIM 77
6.6. ADDITIONAL CNS STORAGE DEPLOYMENTS (OPTIONAL) 78
6.6.1. Deployment of a second Gluster Storage Pool 78
6.6.2. Modifying the Topology File 79
6.6.3. Creating an Additional Storage Class 81
6.7. CONTAINER-READY STORAGE OVERVIEW 82
6.7.1. Prerequisites for Container-Ready Storage 82
6.7.2. Deployment of CRS Infrastructure 82
6.7.3. CRS Subscription Prerequisites 85
6.7.4. Firewall and Security Group Prerequisites 85
6.7.5. CRS Package Prerequisites 85
6.7.6. Installing and Configuring Heketi 86
6.7.7. Loading Topology File 87
6.8. VALIDATING GLUSTER INSTALLATION(OPTIONAL) 89
6.9. DEPLOYING CRS FOR OPENSHIFT CONTAINER PLATFORM (OCP) 90
6.9.1. Store the heketi secret 90
6.9.2. Creating a Storage Class 91
6.9.3. Creating a Persistent Volume Claim 92
6.10. RWO PERSISTENT STORAGE EXAMPLE (OPTIONAL) 92
6.11. RWX PERSISTENT STORAGE (OPTIONAL) 94

CHAPTER 7. EXTENDING THE CLUSTER **101**
7.1. BEFORE ADDING A NODE 101
7.2. INTRODUCTION TO ADD-NODE.PY 101
7.3. ADDING AN APPLICATION NODE 102
7.4. ADDING AN INFRASTRUCTURE NODE 103
7.5. VALIDATING A NEWLY PROVISIONED NODE 103

CHAPTER 8. CONCLUSION **105**

APPENDIX A. REVISION HISTORY **106**

APPENDIX B. CONTRIBUTORS **107**

APPENDIX C. QUICK STEPS: HOW TO INSTALL RED HAT OPENSHIFT CONTAINER PLATFORM **108**

APPENDIX D. TROUBLESHOOTING ANSIBLE BY RED HAT **109**

APPENDIX E. INSTALLATION FAILURE **110**
E.1. INVENTORY 110
E.2. RUNNING THE UNINSTALL PLAYBOOK 111
E.3. MANUALLY LAUNCHING THE INSTALLATION OF RED HAT OPENSHIFT 111
E.4. STARTING COMPLETELY OVER 111
E.5. TROUBLESHOOTING CNS DEPLOYMENT FAILURES 111

APPENDIX F. REVISION HISTORY **113**

COMMENTS AND FEEDBACK

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

CHAPTER 1. EXECUTIVE SUMMARY

Red Hat® OpenShift Container Platform 3 is built around a core of application containers powered by Docker, with orchestration and management provided by Kubernetes, on a foundation of Red Hat Enterprise Linux® Atomic Host. OpenShift Origin is the upstream community project that brings it all together along with extensions to accelerate application development and deployment.

This reference environment provides a comprehensive example demonstrating how OpenShift Container Platform 3 can be set up to take advantage of the native high availability capabilities of Kubernetes and VMware in order to create a highly available OpenShift Container Platform 3 environment. The configuration consists of three OpenShift Container Platform masters, two OpenShift Container Platform infrastructure nodes, two OpenShift Container Platform application nodes, and native VMware integration. In addition to the configuration, operational management tasks are shown to demonstrate functionality.

The target audience for this reference architecture would be a system administrator or system architect with solid background with VMware. Some experience with Docker and OpenShift would be a positive, but it is not required. The subsequent chapters following the deployment will cover cluster administration and validation topics.

If reference architecture has already been read see the following steps in Appendix C, *Quick Steps: How to install Red Hat OpenShift Container Platform* section.

CHAPTER 2. COMPONENTS AND CONFIGURATION

This chapter provides an overview and description of the reference architecture for a highly available Red Hat OpenShift Container Platform 3 environment deployed on a VMware private cloud.

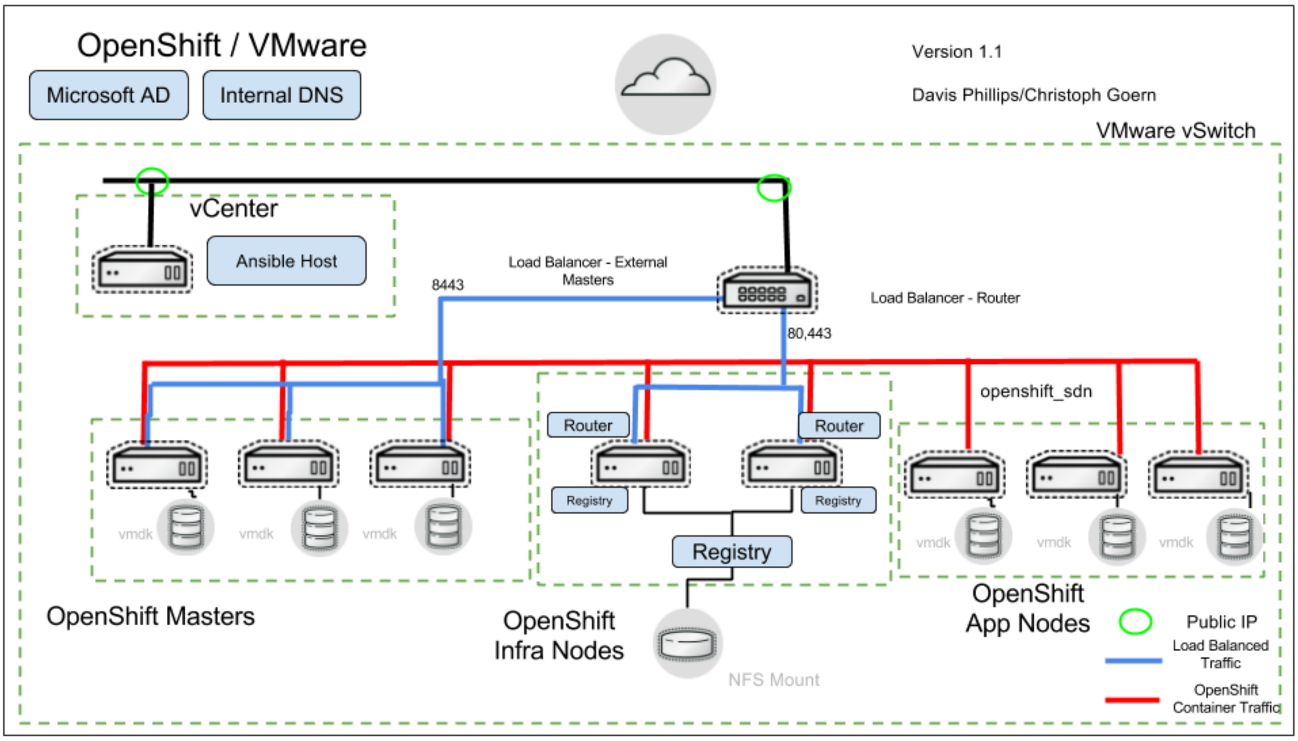
The image, shown below in Figure 1, provides a high-level representation of the components within this reference architecture. Virtual machine (VM) resources are highly available using VMware technologies; VMware HA (high availability), storage IO (input/output) control, and resource allocation via hypervisor affinity and anti-affinity rules. The Ansible host is a virtual machine and acts as the entrypoint for access to the hosts and performs configuration of the internal servers by ensuring that all Secure Shell (SSH) traffic passes through it.

The master instances host the OpenShift master components such as ETCD and the OpenShift API. The application instances are for users to deploy their containers while the infrastructure instances are used for the OpenShift router and registry roles. Authentication is managed by Microsoft Active Directory via lightweight directory access protocol (LDAP) authentication. OpenShift on VMware has two cloud native storage options; virtual machine persistent storage and network file system (NFS).

Virtual machine persistent storage is housed on virtual machine disk VMDKs on datastores located on external logical unit numbers (LUNs) or NFS shares.

The other storage utilized is NFS which is file based storage. NFS is used for the persistent storage of the OpenShift registry and used for persistent volume claims for containers. The network is configured to leverage a single load balancer for access to the OpenShift API & Console (8443/tcp) and the OpenShift routers (80/tcp, 443/tcp). Finally, the image shows that domain name system (DNS) is handled by an external DNS source. This DNS source should be pre-configured with the proper entries prior to deployment. In this case the system engineering team is managing all DNS entries through a BIND server and a conditional lookup zone in Microsoft DNS.

Figure 2.1. Red Hat OpenShift on VMware Architecture



This reference architecture breaks down the deployment into separate phases.

- Phase 1: Provision the infrastructure on VMware

- Phase 2: Provision Red Hat OpenShift Container Platform on VMware
- Phase 3: Post deployment activities

For Phase 1, the provisioning of the environment is done using a series of Ansible playbooks that are provided in the openshift-ansible-contrib github repo.

Once the infrastructure is deployed, the playbooks will flow automatically into Phase 2. Phase 2 is the provisioning of OpenShift Container Platform, which is done via the Ansible playbooks installed by the openshift-ansible-playbooks rpm package.

The playbooks in openshift-ansible-contrib utilize the playbooks included in the atomic-openshift-ansible-playbooks package to perform the installation of OpenShift and also to configure VMware specific parameters. During Phase 2 the router and registry are also deployed.

The last phase, Phase 3, concludes the deployment by confirming the environment was deployed properly. This is done by running command line tools and the systems engineering team's validation Ansible playbook.

NOTE

The scripts provided in the github repo are not supported by Red Hat. They merely provide a mechanism that can be used to build out your own infrastructure.

2.1. REFERENCE IMPLEMENTATION HARDWARE CONFIGURATION

This chapter describes the reference implementation environment that is deployed.

The reference architecture environment consists of a Dell m1000e chassis with a number of Dell PowerEdge M520 blades configured as described in the Table 2.1, "Hardware Details" table. This reference architecture creates by default:

- 3 OpenShift masters
- 3 OpenShift infrastructure nodes
- 3 OpenShift app nodes

To implement a highly available ETCD, each master will also be configured to run ETCD. The infrastructure nodes will host registry and router applications, while the application nodes will host end user applications.

Storage for OpenShift will be provisioned via NFS.

Table 2.1. Hardware Details

Hardware	Specifications
Dell PowerEdge M520	4 port Broadcom Gigabit Ethernet BCM5720
	2 CPU, 8 Core, Intel Xeon CPU E5-2450, 2.10GHz
	96GB of memory

Hardware	Specifications
	2 x 136GB SAS internal disk drives
V9.1.2	Equallogic PS4210
ose3-vmware-prod - 500GB	

Shared storage for the environment has been provisioned on a Dell Equallogic PS4210 array. Storage was presented to our VMware vSphere hosts via 2 iSCSI LUNs. SIOC Storage I/O Control was disabled on these datastores.

Table 2.2. iDRAC Connectivity

Hostname	iDRAC IP Address
vsphere1	10.x.x.88
vsphere2	10.x.x.89

2.2. VMWARE VCENTER AND VSPHERE DETAILS

This reference architecture utilizes the following versions of VMware software:

Table 2.3. VMware Software versions

Software	Version
vCenter Server via VCSA	6.5.0 Build 5973321
vSphere Server	6.5.0 Build 5969303

2.3. VIRTUAL MACHINE INSTANCE DETAILS

Within this reference environment, the virtual machines are deployed via a single VMware cluster in a single datacenter. The virtual machine template is based upon sizing requirements listed in the system requirements section of the docs.

Table 2.4. Virtual Machine Node Requirements

Node Type	Hardware
Master	2 vCPU
	16GB RAM
	1 x 60GB - OS RHEL 7.4

Node Type	Hardware
	1 x 40GB - Docker volume
	1 x 40Gb - EmptyDir volume
	1 x 40GB - ETCD volume
Node	2 vCPU
	8GB RAM
	1 x 60GB - OS RHEL 7.4
	1 x 40GB - Docker volume
	1 x 40Gb - EmptyDir volume

All instances for the OpenShift environment are built from the same template. The master instances contain three extra disks used for Docker storage and ETCD and OpenShift volumes. The application node instances use their additional disks for Docker storage and OpenShift volumes.

2.4. HAPROXY LOAD BALANCER DETAILS

The load balancers used in the reference environment use HAproxy. You can certainly use your own load balancer should one exist on premise. The table below describes the load balancer DNS name, the instances in which the haproxy load balancer is attached, and the port monitored by the load balancer to state whether an instance is in or out of service.

Table 2.5. Haproxy Load Balancers

HAproxy	Assigned Instances	Port
haproxy-0.vcenter.e2e.bos.redhat.com	master-0, master-1, master-2	8443
*.apps.vcenter.e2e.bos.redhat.com	infra-0, infra-1	80 and 443

The wildcard DNS *.apps uses the public subnets and maps to infrastructure nodes. The infrastructure nodes run the router pod which, then directs traffic directly from the outside world into OpenShift pods with external routes defined.

If the keepalived high availability is utilized the IP address of the wildcard DNS will need to point towards the virual IP address assigned to `lb-ha-ip` in the `ocp-on-vmware.ini`.

2.5. SOFTWARE VERSION DETAILS

Table 5 shows the installed software and versions installed on the servers in this Red Hat OpenShift highly available reference environment.

Table 2.6. Red Hat OpenShift Container Platform 3 Details

Software	Version
Red Hat Enterprise Linux 7.4 x86_64	kernel-3.10.0-693
Atomic-OpenShift{master/clients/node/sdn-ovs/utils}	3.6.x.x
Docker	1.12.x
Ansible by Red Hat	2.3.x

2.6. REQUIRED CHANNELS

A subscription to the following channels is required in order to deploy this reference environment's configuration.

Table 2.7. Required Channels – Red Hat OpenShift Container Platform 3 Master and Node Instances

Channel	Repository Name
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms
Red Hat OpenShift Container Platform 3.6 (RPMs)	rhel-7-server-ose-3.6-rpms
Red Hat Enterprise Linux Fast Datapath (RHEL 7 Server)	rhel-7-fast-datapath-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms
Red Hat Satellite Tools 6.2 (for Red Hat Enterprise Linux 7 Server) (RPMs)	rhel-7-server-satellite-tools-6.2-rpms

2.7. TOOLING PREREQUISITES

The following section provides an example of how to install Ansible by Red Hat to deploy and configure our infrastructure.

2.7.1. Git Repository

2.7.1.1. GitHub Repositories

The code in the openshift-ansible-contrib repository referenced below handles the installation of Red Hat OpenShift and the accompanying infrastructure. The openshift-ansible-contrib repository is not explicitly supported by Red Hat.

NOTE

The following task should be performed on the server that the Ansible playbooks will be launched from.

2.7.1.2. Directory Setup

The initial deployment setup can be largely automated by downloading the setup_ansible shell script. The content of the script is discussed in detail below.

```
$ wget https://raw.githubusercontent.com/openshift/openshift-ansible-
contrib/vmw-3.6/reference-architecture/vmware-
ansible/scripts/setup_ansible.sh
$ chmod +x setup_ansible.sh
```

The only prerequisite for running the setup script is to ensure that the system you are running from has a valid **RHN** subscription attached. That can be checked with the following command:

```
# subscription-manager version
server type: Red Hat Subscription Management
subscription management server: 0.9.51.24-1
subscription management rules: 5.15.1
subscription-manager: 1.19.21-1.el7
python-rhsm: 1.19.9-1.el7
```

2.7.2. Ansible by Red Hat Setup

The following section discusses the contents of setup_ansible.sh and provides an example of an Ansible by Red Hat source system and how to install Ansible by Red Hat on a virtual machine.

Install the following packages on the system performing the provisioning of VMware infrastructure and installation of Red Hat OpenShift.

NOTE

The following task should be performed on the server that the Ansible playbooks will be launched from.

```
# ./setup_ansible.sh

# git clone -b vmw-3.6 https://github.com/openshift/openshift-ansible-
contrib

# echo "Please fill in your variables ~/git/openshift-ansible-
contrib/reference-architecture/vmware-ansible/ocp-on-vmware.ini"
# echo "Create the initial inventory with the following command
~/git/openshift-ansible-contrib/reference-architecture/vmware-ansible/ocp-
on-vmware.py --create_inventory"
# echo "Create the OCP install vars with the following command
~/git/openshift-ansible-contrib/reference-architecture/vmware-ansible/ocp-
on-vmware.py --create_ocp_vars"
# echo "Lastly, run ~/git/openshift-ansible-contrib/reference-
architecture/vmware-ansible/ocp-on-vmware.py to complete and test the OCP
install"
```

To verify the repository was cloned the tree command can be used to display all of the contents of the git repository.

```
$ sudo yum -y install tree
$ tree ~/git/

... content abbreviated ...

|-- openshift-ansible-contrib
```

Before getting started with Ansible, first open ocp-on-vmware.ini and populate the configuration file with your environments information.

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

[vmware]
# unique cluster_id set during script run
cluster_id=
# console port and install type for OpenShift
console_port=8443
deployment_type=openshift-enterprise

# OpenShift Version
openshift_vers=v3_6

# vCenter host address/username and password
vcenter_host=
vcenter_username=administrator@vsphere.local
vcenter_password=

# name of RHEL template to use for OpenShift install
vcenter_template_name=ocp-server-template-2.0.2

# folder/cluster/resource pool in vCenter to organize VMs
vcenter_folder=ocp36
vcenter_datastore=
vcenter_cluster=
vcenter_datacenter=
vcenter_resource_pool=ocp36

# DNS zone where everything will be hosted and app wildcard prefix
public_hosted_zone=
app_dns_prefix=apps

# DNS/gateway/interface name the OpenShift nodes should utilize
vm_dns=
vm_gw=
vm_netmask=
vm_network="VM Network"

# red hat subscription name and password
rhel_subscription_user=
rhel_subscription_pass=
```

```
# Internal satellite 6 server
rhel_subscription_server=

# pool with openshift repo access
rhel_subscription_pool=Red Hat OpenShift Container Platform, Premium*

# bringing your own load balancer?
byo_lb=False
lb_host=haproxy-0

# bringing your own NFS server for registry?
byo_nfs=False
nfs_registry_host=nfs-0
nfs_registry_mountpoint=/registry

#create_inventory vars
# number of nodes of each type
master_nodes=3
infra_nodes=3
app_nodes=3
storage_nodes=0

# start node IP address *must be a contiguous space
vm_ipaddr_start=

# node hostname prefix
ocp_hostname_prefix=

# create_ocp_vars vars
# ldap bind user/password and FQDN ldap domain
ldap_user=openshift
ldap_user_password=
ldap_fqdn=

# Deploy OpenShift Metrics
openshift_hosted_metrics_deploy=false

# OpenShift SDN (default value redhat/openshift-ovs-subnet)
openshift_sdn=redhat/openshift-ovs-subnet

# Containerized installation of OpenShift
containerized=false

# persistent container storage: none, crs, cns
container_storage=none
```

The default values are pre-populated. But, the required values are:

- public_hosted_zone
- vcenter_host
- vcenter_password
- vcenter_datacenter

- vcenter_datastore
- vm_ipaddr_start
- ldap_fqdn
- ldap_user_password
- vm_dns
- vm_gw
- vm_netmask

Some explaination of the variables are available Table 3.1, “Red Hat OpenShift Installation Variables” Lastly, your satellite servers address or your Red Hat Network Classic1 username and password for the installation of OpenShift.

NOTE

The cluster_id value will be generated automatically when the inventory creation parameter is run via **`ocp-on-vmware.py`**.

2.8. NETWORK COMPONENTS

2.8.1. DNS (Domain Name Server) Configuration

DNS is an integral part of a successful Red Hat OpenShift Container Platform deployment/environment.

OpenShift Container Platform requires a properly configured wildcard DNS zone that resolves to the IP address of the OpenShift router. For more information, please refer to the OpenShift Container Platform installation guide. In this reference architecture the --create_inventory parameter on the ocp-on-vmware.py script will help manage DNS records for the OpenShift Container Platform environment.

If applications will be hosted externally, a public zone will be required for the setup. However an internal DNS zone can also be specified instead. This assumes the values have been populated in ocp-on-vmware.ini.

```
$ ./ocp-on-vmware.py --create_inventory

Configured inventory values:
 master_nodes: 3
 infra_nodes: 3
 app_nodes: 3
 public_hosted_zone: vmware.example.com
 app_dns_prefix: apps
 ocp_hostname_prefix: ocp3-
 byo_nfs: False
 nfs_host: nfs-0
 byo_lb: False
 lb_host: haproxy-0
 vm_ipaddr_start: 10.x.x.224
 Using values from: ./ocp-on-vmware.ini
```

```
Continue using these values? [y/N]: y

$ORIGIN apps.vmware.example.com.
* A 10.x.x.235
$ORIGIN vmware.example.com.
nfs-0 A 10.x.x.224
haproxy-0 A 10.x.x.235
ocp3-master-0 A 10.x.x.225
ocp3-master-1 A 10.x.x.226
ocp3-master-2 A 10.x.x.227
ocp3-app-0 A 10.x.x.228
ocp3-app-1 A 10.x.x.229
ocp3-app-2 A 10.x.x.230
ocp3-infra-0 A 10.x.x.231
ocp3-infra-1 A 10.x.x.232
ocp3-infra-2 A 10.x.x.233
```

2.8.2. Hosted Zone Setup

Some important things to note from the above configuration. An HAproxy server will be created for load balancing. That HAproxy server is being assigned the final IP address in the specified IP range as is the wildcard, "*.apps.vmware.example.com." The entries above can be copied and pasted into the DNS server or can be used them as a guideline for record creation.

```
$ORIGIN apps.vmware.example.com.
* A 10.x.x.235
$ORIGIN vmware.example.com.
haproxy-0 A 10.x.x.235
```

NOTE

The same steps listed above are applicable when using a subdomain as a subdomain can also be used.

2.8.3. Authentication

There are several options when it comes to authentication of users in Red Hat OpenShift Container Platform. OpenShift can leverage an existing identity provider within an organization such as LDAP, or OpenShift can use external identity providers like GitHub, Google, and GitLab. The configuration of identity providers occurs on the OpenShift master instances. OpenShift allows for multiple identity providers to be specified. This reference architecture document uses LDAP as the authentication provider but any of the other mechanisms would be an acceptable choice. Roles can be added to user accounts to allow for extra privileges, such as the ability to list nodes or assign persistent storage volumes to a project.

For more information on GitHub OAuth and other authentication methods see the Red Hat OpenShift documentation.

The pertinent OpenShift variables for LDAP authentication are listed below:

```
openshift_master_identity_providers:
- name: Active_Directory
  challenge: true
```

```
  login: true
  kind: LDAPPasswordIdentityProvider
  attributes:
    id:
    - dn
    email:
    - mail
    name:
    - cn
    preferredUsername:
    - uid
  insecure: true
  url: "ldap://example.com:389/cn=users,dc=example,dc=com?sAMAccountName"
  bindDN: "cn=openshift,cn=users,dc=example,dc=com"
  bindPassword: "password"
```

OpenShift needs the fully qualified domain name (FQDN) of the LDAP server in question. An OpenShift user was created in the users' organizational unit (OU) and assigned the password of password. To use a specific OU for users to authenticate against, create the BIND distinguished name in the desired OU. This provides a way to isolate logins to a specific group by adding an OU and restricting logins to members of that OU.

The **`ocp-on-vmware.py`** script that executes the install variables does an **`LDAP bind`** and verifies the information provided allows for a successful bind.

The LDAP server configuration can be manually tested (or validated) with the following query:

```
ldapsearch -x -w password -H ldap://dc1.example.com -b
cn=users,dc=example,dc=com -D cn=openshift,cn=users,dc=example,dc=com
cn=openshift
```

2.8.4. NFS (Network File System) Server

NFS is used to store container images in the registry. The NFS server will be created automatically in the playbooks unless it isn't needed. In that case use the byo_nfs = True option as described below.

There are two methods to provide NFS storage for the registry:

- Creating a custom NFS using the script: byo_nfs = False The playbooks will default to creating an NFS server for storage
- Precreated NFS: byo_nfs = True To leverage an existing on-premise NFS, define the following variables in the deployment script INI file:

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

... content abbreviated ...

# bringing your own NFS server for registry?
byo_nfs=False
nfs_registry_host=nfs-0
nfs_registry_mountpoint=/registry

... content abbreviated ...
```

2.8.5. Load Balancer

This environment provides an HAproxy service which enables uniform distribution of network traffic across the node instances deployed in VMware. The load balancer will distribute traffic across two different groups

The infra nodes will be utilized to load balance traffic from 80 and 443. The master nodes will be used to load balance our console port 8443. All three of these ports will be transmission control protocol (TCP) connections.

There are clear advantages to doing this:

- It keeps applications highly available.
- It is elastic, so resources can scale up or down to handle capacity.

There are three methods to provide load balancing for the cluster:

Table 2.8. Load Balancer Options

INI File Option	Description
byo_lb = False	Creates a custom HAproxy VM instance using the Ansible playbooks.
byo_lb = False lb_ha_ip = Assign the floating VIP for keepalived	The Ansible playbooks will create two custom HAproxy VM instances and configure them as highly available utilizing keepalived.
byo_lb = True lb_host = Assign the FQDN of the existing load balancer	Leverages an existing on-premise loadbalaner, define the variables in the INI file.

```
$ *vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini*

... content abbreviated ...

# bringing your own load balancer?
byo_lb=False
lb_host=haproxy-0

# HA haproxy config
lb_ha_ip=
... content abbreviated ...
```

Ensure that the proper ports are configured on your existing load balancer and additionally make sure that your wildcard DNS record points to it as well.

2.9. VMWARE VCENTER PREREQUISITES

For simplicity sake, assume the vCenter environment is pre-existing and is configured with best practices for the infrastructure.

Technologies such as SIOC and VMware HA should already be configured. After the environment is provisioned, some anti-affinity rules will be established to ensure maximum uptime and optimal performance.

Table 2.9. Ansible by Red Hat vCenter settings

Parameter	Description	Defaults
vcenter_host	IP or Hostname of vCenter Server	
vcenter_username	Username of vCenter Administrator	'administrator@vsphere.local'
vcenter_password	Password of vCenter administrator	
vcenter_cluster	Cluster to place VM in	
vcenter_datacenter	Datacenter to place VM in	
vcenter_datastore	Datastore to place VM in and the target destination for the VMDK storage class	
vcenter_resource_pool	Resource pool to be used for newly created VMs	'ocp36'
vcenter_folder	Folder to place newly created VMs in	'ocp36'
vcenter_template_name	Template to clone new VM from	'ocp-server-template-2.0.2'
vm_network	Destination network for VMs. (vSwitch or VDS)	"VM Network"

2.9.1. Networking

An existing port group and virtual LAN (VLAN) are required for deployment. The initial configuration of the Red Hat OpenShift nodes in this reference architecture assumes you will be deploying VMs to a port group called "VM Network". This can be changed in the ocp-on-vmware.ini file under the variable: vm_network. Once deployed vmtoolsd is used to determine the static addresses.

The environment can utilize a virtual dedicated server (vDS) or vSwitch. The specifics of that are unimportant. However, should you wish to utilize network IO control and some of the quality of service (QoS) technologies that VMware employs, you would need to choose a vDS for the deployment.

2.9.2. vCenter Shared Storage

The vSphere hosts should ultimately have shared storage to house our VMware virtual machine disk files (VMDKs) for our templates. A best practice recommendation would be to enable storage I/O control (SIOC) to address any latency issues caused by performance. The following article discusses in

depth how to do this.

NOTE

Some storage providers such as Dell Equallogic advise to disabled storage I/O control (SIOC) as the array optimizes it. Check with your storage provider for details.

2.9.3. Resource Pool, Cluster Name and Folder Location

This reference architecture assumes some default names as per the ocp-on-vmware.py wrapper script. In particular, vcenter_resource_pool, vcenter_folder and vcenter_template_name all have default values.

The setup playbook in ansible creates a folder and resource pool via the defined values in ocp-on-vmware.ini the defaults are ocp36.

- Creates a resource pool named: "ocp36"
- Creates a folder for the Red Hat OpenShift VMs for logical organization named: "ocp36"
    - Ensure this folder exists under the datacenter and cluster you will use for deployment

This will allow you to use default names and not force you to customize outside of these entries. If you would like to customize the names feel free to but, remember to specify them later on during the Section 3.1, “Provisioning the Infrastructure with Ansible by Red Hat” section.

2.9.4. SSH (Secure Shell) Prerequisite

2.9.4.1. SSH Configuration

Before beginning the deployment of the VMware infrastructure and the deployment of Red Hat OpenShift, a specific SSH configuration must be in place to ensure that the proper SSH keys are used during the provisioning process.

NOTE

The following task should be performed on the server where the Ansible playbooks will be launched.

```
$ ssh-keygen -N '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/root/.ssh'.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:aaQHUf2rKHWvwwl4RmYcmCHswoouu3rdZiSH/BYgzBg root@ansible-test
The key's randomart image is:
+---[RSA 2048]----+
|    .. o=..      |
|E    ..o.. .     |
| * .   .... .    |
|. * o  +=.  .    |
|.. + o.=S    .   |
|o    + =o= . .   |
```

```
|.   . * = = +      |
|... . B . = .      |
+----[SHA256]-----+

$ ssh-copy-id root@ipaddress_of_template
```

NOTE

The ocp-on-vmware script will copy over the user's ssh id_rsa to the ssh key ansible expects at ~/git/openshift-ansible-contrib/reference-architecture/vmware-ansible/ssh_key/ocp-installer

Please note, that you are encouraged to copy over your public key "~/.ssh/id_rsa.pub" to the template's authorized_keys file ahead of time. The ssh-copy-id above accomplishes that task.

2.9.5. VMware Template

A prepared VMware template is used to deploy the environment. In this guide, the Red Hat Enterprise Linux 7 gold image is prepped by starting with an installation of open-vm-tools, perl, open-vm-tools-deploypkg and net-tools.

Any image that meets the prerequisites mentioned above should suffice. The Ansible playbooks will copy over keys or you could prepare the the authorized_keys section ahead of time. The default vcenter_template_name is "ocp3-server-template-2.0.2". Obviously, this can be customized to your environment at runtime.

Prepare the virtual machine to meet the specified requirements listed in the system requirements section of the docs:

- 2 vCPU
- 8GB RAM
  - Application nodes running **CNS** require 32GB of RAM. This will be discussed later in the storage section.
- 1 x 60GB drive
- 2 x 40GB drives (all thin provisioned)
  - The masters require an additional drive 40GB drive for **ETCD** storage

2.9.6. Preparation Script

A sample script is listed below for preparing the **`vcenter_template_name`**:

```
#!/bin/bash
#
# This file will prepare a RHEL7.4 instance for being converted to a
VMware template
# used as an OpenShift Enterprise host

yum install -y open-vm-tools perl open-vm-tools-deploypkg net-tools
python-six
```

```
systemctl enable vmtoolsd.service
# Remove all let the ansible roles configure
subscription-manager remove --all
subscription-manager unregister

# deploy default ssh key
ssh-keygen -N '' -f ~/.ssh/id_rsa

# This part is redundant provided the ssh-copy-id successfully ran
echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCuY6lBsj95cYTVmzqW2Xci37BJta+ZtNHIee8bFCDskK
x/qiH/kbceQGDPpAjREBQBpabhxwd8eUktqdiUI1aQIs7I0Om8XMNUd00Phsz69i8PDzVQyGcL
zdn4UPOpS89nFmyto0NJ1V5o4RoR3A7ENbzN7li34g5+zGSBXcVdHFFUErCfOnEtgXw5kltU1b
yv3+GGKAb+f0CL88BRowp35/NH9sRkP8fzWPS0hl+ofiay5H7xDDd6/nqx6OCd0YHfSEYSTMuR
KcM55IFDLLgVNLIumYqwDP6WdCw9dv2aSHPX0bpuLwIEgaMEfGgvTTNi8rZ/rC9Y9OYewfHYOp
r9 dphillip@dav1x-m" >> /root/.ssh/authorized_keys

# and run an update
yum update -y && yum clean all

# clean up the system to prepare for being a VMware template
rm -rf /etc/udev/rules.d/70* && rm -rf /etc/ssh/ssh_host_* && logrotate -f
/etc/logrotate.conf && rm -rf /var/log/audit/audit.log && history -c
systemctl halt
```

2.10. DYNAMIC INVENTORY

Ansible by Red Hat relies on inventory files and variables to perform playbook runs. The Ansible playbooks provided by the reference architecture will use a dynamic inventory script. The dynamic inventory script queries the VMware API to display information about VMware instances. This information is used to create an in-memory inventory file for Ansible to use.

The dynamic inventory script is also referred to as an Ansible inventory script and the VMware specific script is written in Python. The script can can be manually executed to provide information about the environment, but for this reference architecture it is automatically called to generate the Ansible inventory.

The inventory script is located here

But, is stored in the cloned repo here: ~/git/openshift-ansible-contrib/reference-architecture/vmware-ansible/inventory/vsphere/vms/vmware_inventory.py The `INI` config file is stored in the same directory.

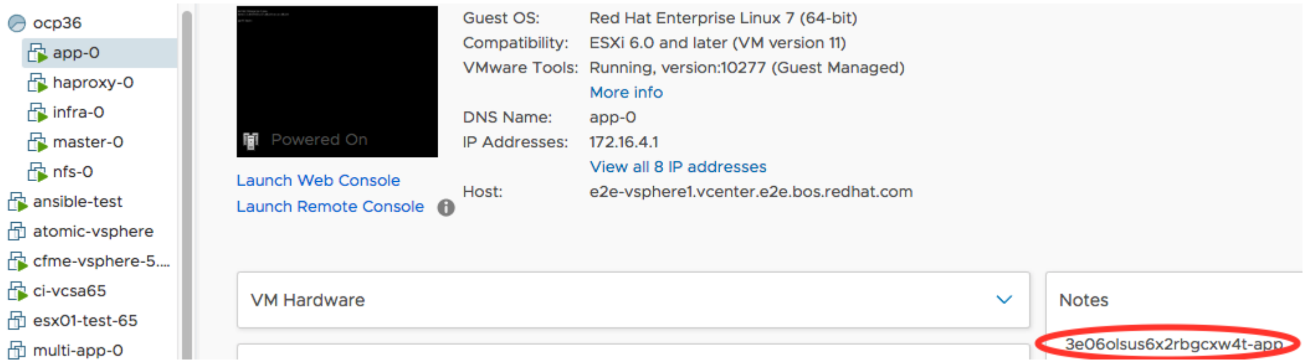
This particular inventory script utilizes VMware's pyVMomi Python software development kit (SDK) for querying vCenter. Some modifications to the default INI configuration file are shown below.

```
#Alias the VM by guestname alone
#alias_pattern={{ config.name + '_' + config.uuid }}
alias_pattern={{ config.name }}

# The host pattern is the ansible_ssh_host
# We want to leverage DNS because when we add virtual interfaces for
# things like docker this could get confusing for vmtools
host_pattern={{ guest.hostname }}
```

```
# The default is only gueststate of 'running'
# We just want to filter for our VMs
host_filters={{ guest.gueststate == "running" }}, {{ config.guestid ==
'rhel7_64Guest' }}, {{ config.template != 'templates' }}

# Lastly and most importantly, we filter by the VM applied annotation
# This allows us to apply the proper OpenShift labels based on our groupby
logic
#groupby_patterns={{ guest.guestid }},{{ 'templates' if config.template
else 'guests'}},
groupby_patterns={{ config.annotation }}
```

For the Red Hat OpenShift installation, the Python script and the Ansible module add_host allow for instances to be grouped based on their purpose to be used in later playbooks. During Phase 1, the infrastructure is provisioned with VMware annotations. The VMware annotations provide the ability to group instances.

Figure 2.2. VMware VM annotations example



The annotation uses the following layout. A randomly generated **cluster_id** plus master, app, infra, loadbalancer or networkfs.

Table 2.10. Node Annotations

Node Type	Annotation
master	master
app	app
infra	infra
NFS	networkfs
HAproxy	loadbalancer

The interesting parts of output from our inventory script are shown below:

```
"3e06olsus6x2rbgcxw4t-app": {
  "hosts": [
    "app-2",
    "app-1",
    "app-0"
```

```
    ]
  },
  "3e06olsus6x2rbgcxw4t-infra": {
    "hosts": [
      "infra-0",
      "infra-1",
      "infra-2"
    ]
  },
  "3e06olsus6x2rbgcxw4t-master": {
    "hosts": [
      "master-1",
      "master-0",
      "master-2"
```


For more information see: http://docs.ansible.com/ansible/intro_dynamic_inventory.html

2.11. NODES

Nodes are VMware virtual machines that serve a specific purpose for Red Hat OpenShift. OpenShift masters are also considered nodes. Nodes deployed on VMware can be horizontally scaled or scaled out before or after the OpenShift installation using the create_inventory flag in the deployment script.

There are three types of nodes as described below.

2.11.1. Master nodes

The master nodes contain the master components, including the API server, controller manager server and ETCD.

The masters:

- Maintain the clusters configuration.
- Manages nodes in its Red Hat OpenShift cluster.
- Assigns pods to nodes.
- Synchronize pod information with service configuration

The masters are used to define routes, services, and volume claims for pods deployed within the OpenShift environment.

2.11.2. Infrastructure nodes

The infrastructure nodes are used for the router and registry pods. These nodes could be used if the optional components Kibana and Hawkular metrics are required. The storage for the Docker registry that is deployed on the infrastructure nodes is NFS which allows for multiple pods to use the same storage. NFS storage is used because it largely prevalent in most VMware environments.

2.11.3. Application nodes

The Application nodes are the instances where non-infrastructure based containers run. Depending on the application, VMware specific storage can be applied such as a VMDK which can be assigned using a persistent volume claim for application data that needs to persist between container restarts. A

configuration parameter is set on the master which ensures that Red Hat OpenShift Container Platform user containers will be placed on the application nodes by default.

2.11.4. Node labels

All Red Hat OpenShift Container Platform nodes are assigned a label. This allows certain pods to be deployed on specific nodes. For example, nodes labeled infra are infrastructure nodes. These nodes run the router and registry pods. Nodes with the label app are nodes used for end user application pods.

The configuration parameter `'defaultNodeSelector: "role=app"` in /etc/origin/master/master-config.yaml ensures all pods automatically are deployed on application nodes.

2.12. RED HAT OPENSHIFT PODS

Red Hat OpenShift leverages the Kubernetes concept of a pod, which is one or more containers deployed together on a single host. The container is the smallest compute unit that can be defined, deployed, and managed. For example, a pod could be just a single PHP application connecting to a database outside of the OpenShift environment or a pod could be a PHP application that's connected to another ephemeral database container.

Red Hat OpenShift pods have the ability to be scaled at runtime or at the time of launch using the OpenShift console or the oc CLI tool. Any container running in the environment is considered a pod. The pods containing the OpenShift router and registry are required to be deployed in the OpenShift environment.

2.13. OPENSHIFT SDN

OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network that enables communication between pods across the OpenShift Container Platform cluster. The reference architecture scripts deploy the ovs-multitenant plug-in by default but the option exists to deploy the ovs-subnet plug-in.

- The ovs-subnet plug-in is the original plug-in which provides a "flat" pod network where every pod can communicate with every other pod and service.

- The ovs-multitenant plug-in provides OpenShift Container Platform project level isolation for pods and services. Each project receives a unique Virtual Network ID (VNID) that identifies traffic from pods assigned to the project. Pods from different projects cannot send packets to or receive packets from pods and services of a different project.

2.14. ROUTER

Pods inside of a Red Hat OpenShift cluster are only reachable via their IP addresses on the cluster network. An edge load balancer can be used to accept traffic from outside networks and proxy the traffic to pods inside the OpenShift cluster.

An OpenShift administrator can deploy routers in an OpenShift cluster. These enable routes created by developers to be used by external clients.

OpenShift routers provide external hostname mapping and load balancing to services over protocols that pass distinguishing information directly to the router; the hostname must be present in the protocol in order for the router to determine where to send it. Routers support the following protocols:

- HTTP

- HTTPS (with SNI)
- WebSockets
- Transport layer security (TLS) with Server Name Indication (SNI)

The router utilizes the wildcard zone specified during the installation and configuration of OpenShift. This wildcard zone is used by the router to create routes for a service running within the OpenShift environment to a publicly accessible URL. The wildcard zone itself is a wildcard entry in DNS which is linked using a Canonical Name record (CNAME) to a load balancer, which performs a health check and forwards traffic to router pods on port 80 and 443.

2.15. REGISTRY

Red Hat OpenShift can build Docker images from your source code, deploy them, and manage their life cycle. To enable this, OpenShift provides an internal, integrated Docker registry that can be deployed in your OpenShift environment to manage images.

The registry stores Docker images and metadata. For production environment, you should use persistent storage for the registry, otherwise any images anyone has built or pushed into the registry would disappear if the pod were to restart.

Using the installation methods described in this document the registry is deployed using an NFS share. The NFS share allows for multiple pods to be deployed at once for HA but also use the same persistent backend storage. NFS is file based storage which does not get assigned to nodes in the same way that VMDK volumes are attached and assigned to a node. The share does not mount as block-based storage to the node so commands like fdisk or lsblk will not show information in regards to the NFS share.

The configuration for the NFS share is completed via an Ansible playbook. Users of this reference architecture can also bring their own NFS server and share name should an existing resource exists on-premise.

2.16. OPENSHIFT METRICS

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

When metrics are deployed, persistent storage should be use to allow for metrics to be preserved. Metrics data by default is stored for 7 days unless specified.

CHAPTER 3. PROVISIONING THE INFRASTRUCTURE

This chapter focuses on Phase 1 of the process. The prerequisites defined below are required for a successful deployment of infrastructure and the installation of OpenShift.

3.1. PROVISIONING THE INFRASTRUCTURE WITH ANSIBLE BY RED HAT

The script and playbooks provided within the git repository deploys infrastructure, installs and configures Red Hat OpenShift, and performs post installation tasks such as scaling the router and registry. The playbooks create specific roles, policies, and users required for cloud provider configuration in OpenShift and management of a newly created NFS share to manage container images.

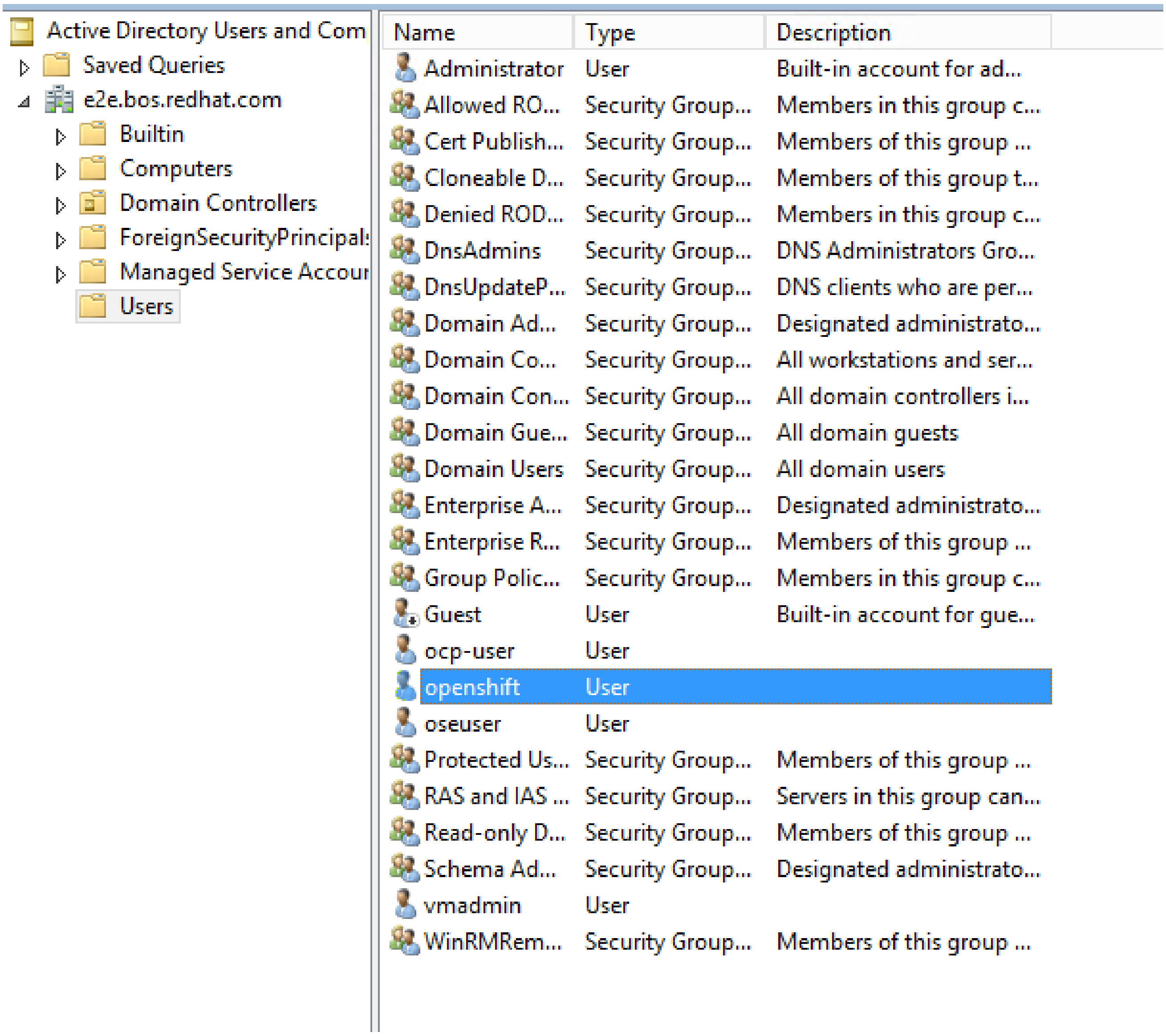
3.1.1. Authentication Prerequisite

As mentioned in the Section 2.8.3, “Authentication” section, authentication for the reference architecture deployment is handled by Microsoft’s Active Directory LDAP (lightweight directory access protocol). The steps below describe how to connect your OpenShift deployment to an LDAP server.

3.1.1.1. Create a Red Hat OpenShift lightweight directory access protocol (LDAP) BIND user account

An existing user account can be utilized or a new account can be created. Below, we have created the user openshift in our default users organizational unit or OU. The location does not matter as our wrapper script will search LDAP for the distinguished name and return our full path to the user account.

Figure 3.1. Active Directory Users and Computers



In our example above, our important OpenShift authentication variables would be:

```
url:
ldap://e2e.bos.redhat.com:389/CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com?
sAMAccountName
bindDN: CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com
bindPassword: password
```

3.2. OCP-ON-VMWARE.PY - VARIABLES

NOTE

The following task should be performed on the server that the Ansible playbooks will be launched.

This will be our first hands on experience with ocp-on-vmware.py. Within the openshift-ansible-contrib git repository is a python script called ocp-on-vmware.py that launches VMware resources and installs Red Hat OpenShift on the new resources. The OpenShift install Ansible playbook requires a few

variables for a successful installation. This will include both authentication variables and the network FQDN variables.

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ ./ocp-on-vmware.py --create_ocp_vars
Configured OCP variables:
  auth_type: ldap
  ldap_fqdn: e2e.bos.redhat.com
  ldap_user: openshift
  ldap_user_password: password
  public_hosted_zone: vcenter.example.com
  app_dns_prefix: apps
  byo_lb: False
  lb_fqdn: haproxy-0
  Using values from: ./ocp-on-vmware.ini
Continue using these values? [y/N]:
```

Table 3.1. Red Hat OpenShift Installation Variables

Variable	Purpose	Defaults
auth_type	The type of authentication used with OpenShift valid options are ldap, none.	ldap
ldap_fqdn	The location of your LDAP server. This could be a forest domain for AD.	
ldap_user	The user account you will bind to LDAP with.	openshift
ldap_user_password	The LDAP password for the user account.	
public_hosted_zone	The DNS zone for your OpenShift configuration	
app_dns_prefix	The wildcard prefix for that DNS zone.	apps
byo_lb	Will you be bring an on premise load balancer?	False
lb_host	If you are bringing your own, what is the hostname?	haproxy-0
lb_ha_ip	Virtual IP address to assign for keepalived. This should be a different IP than the two HAproxy instances	

As noted above we will need the username and password created to bind to LDAP. We will also need your LDAP FQDN to locate your server. If the LDAP source is Microsoft's Active Directory, the location where you are running the ocp-on-vmware.py will need the ability to locate service records (SRVs) for your domain using DNS.

The wildcard zone is a combination of the app_dns_prefix and the public_hosted_zone. This will be the FQDN that OpenShift uses for deployed applications and the load balancer FQDN will serve as the OpenShift cluster's hostname and public hostname for cluster configuration purposes.

3.2.1. Sample Red Hat OpenShift Install Variables

Let's assume that the variables will be the following:

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

... content abbreviated ...

# DNS zone where everything will be hosted and app wildcard prefix
public_hosted_zone=example.com
app_dns_prefix=apps

... content abbreviated ...

# create_ocp_vars vars
# ldap bind user/password and FQDN ldap domain
ldap_user=openshift
ldap_user_password=password
ldap_fqdn=example.com

# Deplot OpenShift Metrics
openshift_hosted_metrics_deploy=false

# OpenShift SDN (default value redhat/openshift-ovs-subnet)
openshift_sdn=redhat/openshift-ovs-subnet

... content abbreviated ...

$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
./ocp-on-vmware.py --create_ocp_vars
```

Once the script runs the ocp-install.yaml file that contains the Ansible variables for the OCP installation is modified with above variables

An example is below:

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/playbooks/ocp-install.yaml
... ommitted ...
wildcard_zone: apps.example.com
osm_default_subdomain: "{{ wildcard_zone }}"
load_balancer_hostname: haproxy-0.example.com
openshift_master_cluster_hostname: "{{ load_balancer_hostname }}"
openshift_master_cluster_public_hostname: "{{ load_balancer_hostname }}"
```

```
openshift_master_identity_providers:
- name: Active_Directory
  challenge: true
  login: true
  kind: LDAPPasswordIdentityProvider
  attributes:
    id:
    - dn
    email:
    - mail
    name:
    - cn
    preferredUsername:
    - uid
  insecure: true
  url: ldap://example.com:389/CN=Users,DC=example,DC=com?sAMAccountName
  bindDN: CN=openshift,CN=Users,DC=example,DC=com
  bindPassword: password
... ommitted ...
```

3.3. OCP-ON-VMWARE.PY - INVENTORY

NOTE

The following task should be performed on the server where the Ansible playbooks will be launched.

Now that our installation variables are complete, define the infrastructure requirements.

WARNING

Before deployment the VLAN used as a deploy target, should have a portgroup named "VM Network" available. Also in the VLAN should be a set of contiguous IP Addresses with enough addresses to cover our deployment components. If you are bringing your own NFS or loadbalancer those IPs could be omitted.

- Wildcard IP Address, this will also be the target for our load balancer
- The NFS Server
- The number of master nodes you wish to install
- The number of infra nodes you wish to install
- The number of app nodes you wish to install
- The number of storage nodes you wish to install (This will be explained in the storage chapter)

Provided these requirements are defined, populate the entries in the VMware guest inventory file.

—

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

... content abbreviated ...

#create_inventory vars
# number of nodes of each type
master_nodes=3
infra_nodes=3
app_nodes=3

# start node IP address must be a contiguous space
vm_ipaddr_start=10.x.x.224

... content abbreviated ...

$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
./ocp-on-vmware.py --create_inventory
Configured inventory values:
 master_nodes: 3
 infra_nodes: 3
 app_nodes: 3
 public_hosted_zone: example.com
 app_dns_prefix: apps
 ocp_hostname_prefix:
 byo_nfs: False
 nfs_host: nfs-0
 byo_lb: False
 lb_host: haproxy-0
 vm_ipaddr_start: 10.x.x.224
  Using values from: ./ocp-on-vmware.ini
```

Table 3.2. Red Hat OpenShift Inventory Variables

Variable	Purpose	Defaults
master_nodes	The number of master nodes to create.	3
infra_nodes	The number of infra nodes to create.	3
app_nodes	The number of app nodes to create.	3
storage_nodes	The number of storage nodes to create.	0
public_hosted_zone	The DNS zone for your OpenShift configuration.	

Variable	Purpose	Defaults
app_dns_prefix	The wildcard prefix for that DNS zone.	apps
ocp_hostname_prefix	Any prefix for the guestname and hostname of your OpenShift nodes.	
byo_nfs	Will you be bring an on-premise NFS server?	False
nfs_host	If you are bringing your own, what is the hostname?	nfs-0
byo_lb	Will you be bring an on-premise load balancer?	False
lb_host	If you are bringing your own, what is the hostname?	haproxy-0
vm_ipaddr_start	The starting IP address for your range of contiguous addresses.	

A possibility for scaling against your environment could be to create an equal number of master nodes per the number of hypervisors. Remember that the ETCD cluster requires an odd number of nodes for cluster election This would allow us to create anti-affinity rules separating the master nodes and allowing for maximum uptime. However, you can customize the number of all your nodes for the environment by modifying the value of the app_nodes, master_nodes, and infra_nodes variables.

Our wrapper script basically takes the number of nodes you will be building and increments your vm_ipaddr_start with that. Let's build a sample configuration that matches our OpenShift install variables above.

3.3.1. Sample Red Hat OpenShift Inventory Variables

Let's assume that our variables will be the following:

```
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

... content abbreviated ...
public_hosted_zone=example.com

#create_inventory vars
# number of nodes of each type
master_nodes=3
infra_nodes=3
app_nodes=3
```

```
# start node IP address must be a contiguous space
vm_ipaddr_start=10.x.x.224

# node hostname prefix
ocp_hostname_prefix=ocp3-

... content abbreviated ...

$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
./ocp-on-vmware.py --create_inventory --no-confirm

# Here is what should go into your DNS records
$ORIGIN apps.example.com.
*                   A       10.x.x.234
$ORIGIN example.com.
nfs-0               A       10.x.x.224
haproxy-0           A       10.x.x.234
ocp3-master-0      A       10.x.x.225
ocp3-master-1      A       10.x.x.226
ocp3-master-2      A       10.x.x.227
ocp3-app-0         A       10.x.x.228
ocp3-app-1         A       10.x.x.229
ocp3-app-2        A       10.x.x.230
ocp3-infra-0        A       10.x.x.231
ocp3-infra-1        A       10.x.x.232
ocp3-infra-2        A       10.x.x.233
# Please note, if you have chosen to bring your own load balancer and NFS
Server you will need to ensure that
# these records are added to DNS and properly resolve.
```

As you can see based on our input we have the guidelines for our DNS zone creation or modification. Also, the wildcard_zone record is shown with our supplied public_hosted_zone and the default app_dns_prefix of apps. Additionally, in that directory we have created a dynamic base inventory file infrastructure.json with the specifics to provisioning the environment.

Here is an interesting excerpt from infrastructure.json converted to YAML:

```
ocp3-master-0:
  guestname: ocp3-master-0
  tag: 3e06olsus6x2rbgcxw4t-master
  ip4addr: 10.19.114.225
```

Note the "tag: 3e06olsus6x2rbgcxw4t-master" in the entry. This will be the annotation created on the virtual machine and is how OpenShift labels are generated for the VMware virtual machines.

3.3.2. VMware Configuration Variables

VMware vCenter Deployment Variables

-

Variable	Purpose	Defaults
vcenter_host	vCenter IP Address	
vcenter_username	vCenter Username	administrator@vsphere.local
vcenter_password	Password for connecting to vCenter	
vcenter_template_name	Pre-created VMware template with Red Hat Enterprise Linux	ocp3-server-template-2.0.2
vcenter_folder	Logical target for VM creation	ocp3
vcenter_cluster	vCenter cluster for VMs	
vcenter_datacenter	vCenter datacenter for VMs	
vcenter_datastore	vCenter datastore for VMs	
vcenter_resource_pool	Resource Pool to use in vCenter	OCP3

3.3.2.1. VMware Authentication Variables

To connect to our vCenter server and be able to provision VMs with our Ansible playbooks, we will need three components:

- vcenter_host - vCenter IP Address
- vcenter_username - vCenter Username, this defaults to Administrator@vsphere.local but can be anything
- vcenter_password - Password for connecting to vCenter

3.3.2.2. Remaining VMware Deployment Variables

Once we have these three authentication variables, the remaining variables as described in Section 2.9, “VMware vCenter Prerequisites” are:

- vcenter_template_name
- vcenter_folder
- vcenter_cluster
- vcenter_resource_pool
- vcenter_datacenter
- vcenter_datastore
- vm_gw

- vm_dns
- vm_netmask

3.3.3. Deploying the Environment

Intelligence is built into the playbooks to allow for certain variables to be set using options provided by the ocp-on-vmware.py script. The script allows for deployment into an existing environment (brownfield) or a new environment (greenfield) using a series of Ansible playbooks. Once the Ansible playbooks begin, the installation automatically flows from the VMware deployment to the Red Hat OpenShift deployment and post installation tasks.

3.3.3.1. ocp-on-vmware.py - deployment

The ocp-on-vmware.py script contains many different configuration options such as configuring Red Hat Network Classic1 vs. Red Hat Satellite Server.

To see all of the potential options, check out the INI file.

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

[vmware]
# console port and install type for OpenShift
console_port=8443
deployment_type=openshift-enterprise

# vCenter host address/username and password
vcenter_host=
vcenter_username=administrator@vsphere.local
vcenter_password=

... ommitted ...
```

3.3.3.2. Greenfield Deployment

For deploying Red Hat OpenShift into a new environment, ocp-on-vmware.py creates our virtual machines, deploys and configures an HAproxy load balancer and an NFS server for registry storage. Once the values have been entered into the ocp-on-vmware.py script, all values will be presented from the configuration file and the script will prompt to continue with the values or exit.

By default, the Red Hat gold image vcenter_template_name ocp3-server-template-2.0.2, as described in the Section 2.9.5, “VMware Template” section, is used when provisioning VMs but can be changed when executing the script.

Additionally, the VMware specific variables below are using the defaults as discussed in Section 2.9, “VMware vCenter Prerequisites”.

Example of Greenfield Deployment values

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ ./ocp-on-vmware.py
```

```
Configured values:

console_port: 8443
deployment_type: openshift-enterprise
vcenter_host: 10.x.x.25
vcenter_username: administrator@vsphere.local
vcenter_password:
vcenter_template_name: ocp-server-template-2.0.2
vcenter_folder: ocp36
vcenter_cluster: devel
vcenter_datacenter: Boston
vcenter_datastore: ose3-vmware-prod
vcenter_resource_pool: ocp36
public_hosted_zone: vcenter.example.com
app_dns_prefix: apps
vm_dns: 10.x.x.5
vm_gw: 10.x.x.254
vm_netmask: 255.255.255.0
vm_network: VM Network
byo_lb: False
lb_host: haproxy-0.vcenter.example.com
byo_nfs: False
nfs_registry_host: nfs-0.vcenter.example.com
nfs_registry_mountpoint: /registry
apps_dns: apps.vcenter.example.com
openshift_sdn: redhat/openshift-ovs-subnet
openshift_hosted_metrics_deploy: false
container_storage: none

Using values from: ./ocp-on-vmware.ini

Continue using these values? [y/N]:
```

3.3.3.3. Brownfield Deployment

The ocp-on-vmware.py script allows for deployments into an existing environment in which VMs already exists and are subscribed to the proper Red Hat Enterprise Linux channels. The prerequisite packages will be installed. The script expects the proper VM annotations are created on your VMs as described in Section 2.10, "Dynamic Inventory". A **`cluster_id`** can be manually created. Then App nodes will be labeled "app", infra nodes labeled "infra" and master nodes labeled as "master."

Lastly, the prepared VMs must also have two additional hard disks as the OpenShift setup needs those for both docker storage and OpenShift volumes.

Let's take a look at the tags in the last line below:

- nfs - This tag will install an NFS server using the playbooks and variables given
- HAproxy - This tag installs our load balancer service
- ocp-install - This will install OpenShift on your pre-existing environment
    - The dynamic inventory script uses your pre-existing annotations to determine labels
- ocp-configure - The final tag listed will configure your persistent registry and scale

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ ./ocp-on-vmware.py --tag nfs,haproxy,ocp-install,ocp-configure

Configured values:

cluster_id: my_custom_id
console_port: 8443
deployment_type: openshift-enterprise
vcenter_host: 10.x.x.25
vcenter_username: administrator@vsphere.local
vcenter_password:
vcenter_template_name: ocp-server-template-2.0.2
vcenter_folder: ocp36
vcenter_cluster: devel
vcenter_datacenter: Boston
vcenter_datastore: ose3-vmware-prod
vcenter_resource_pool: ocp36
public_hosted_zone: vcenter.example.com
app_dns_prefix: apps
vm_dns: 10.x.x.5
vm_gw: 10.x.x.254
vm_netmask: 255.255.255.0
vm_network: VM Network
rhel_subscription_user: rhn_user
rhel_subscription_password:
byo_lb: False
lb_host: haproxy-0.vcenter.example.com
byo_nfs: False
nfs_registry_host: nfs-0.vcenter.example.com
nfs_registry_mountpoint: /registry
apps_dns: apps.vcenter.example.com
Using values from: ./ocp-on-vmware.ini

Continue using these values? [y/N]:
```

In the case where NFS and load balancer instance(s) have already been deployed, an option exists within our INI file to not deploy those services.

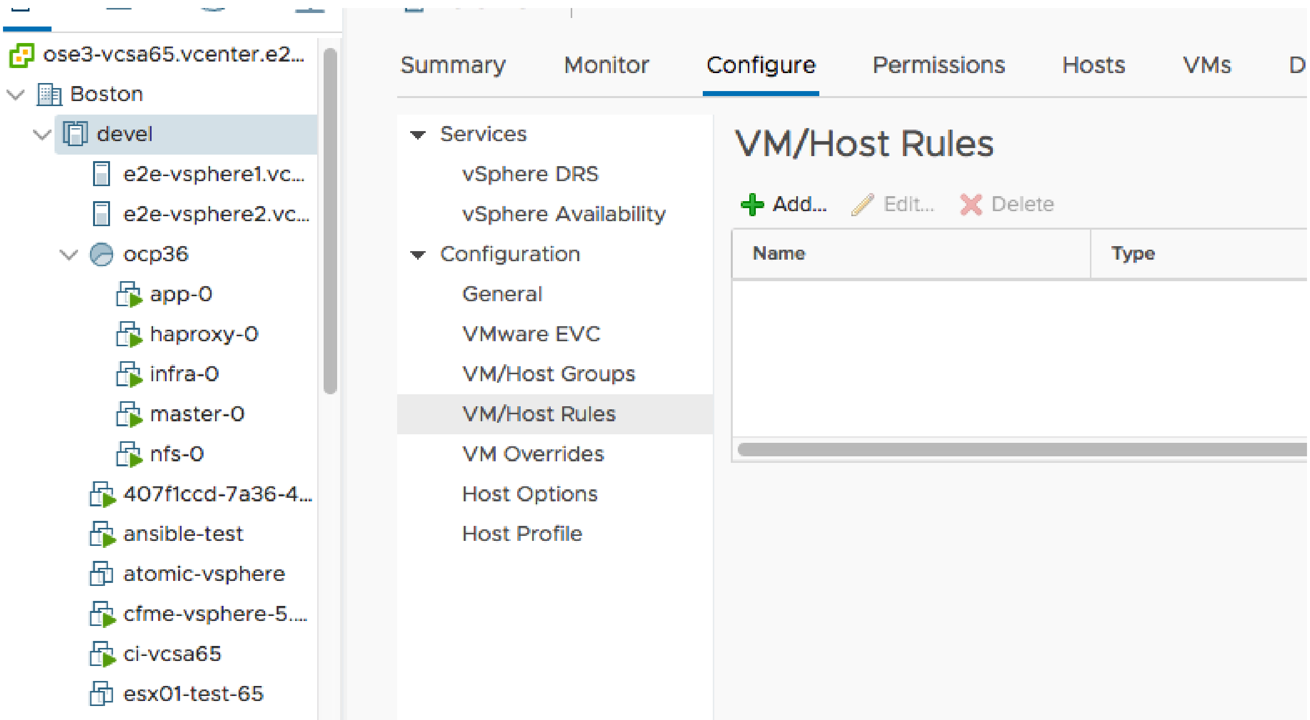
```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ vim ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini

... content abbreviated ...

# bringing your own load balancer?
byo_lb=True
lb_host=my-load-balancer.lb.example.com

# bringing your own NFS server for registry?
byo_nfs=True
nfs_registry_host=my-nfs-server.nfs.example.com
nfs_registry_mountpoint=/my-registry

... content abbreviated ...
```

```
$ ./ocp-on-vmware.py --tag ocp-install,ocp-configure

Configured values:

cluster_id: my_custom_id
console_port: 8443
deployment_type: openshift-enterprise
vcenter_host: 10.19.114.25
vcenter_username: administrator@vsphere.local
vcenter_password:
vcenter_template_name: ocp-server-template-2.0.2
vcenter_folder: ocp
vcenter_cluster: devel
vcenter_datacenter: Boston
vcenter_resource_pool: OCP3
public_hosted_zone: vcenter.e2e.bos.redhat.com
app_dns_prefix: apps
vm_dns: 10.19.114.5
vm_gw: 10.19.115.254
vm_netmask: 255.255.254.0
vm_network: VM Network
byo_lb: True
lb_host: my-load-balancer
byo_nfs: True
nfs_host: my-nfs-server
nfs_registry_mountpoint: /my-registry
apps_dns: apps.vcenter.e2e.bos.redhat.com
Using values from: ./ocp-on-vmware.ini

Continue using these values? [y/N]:
```

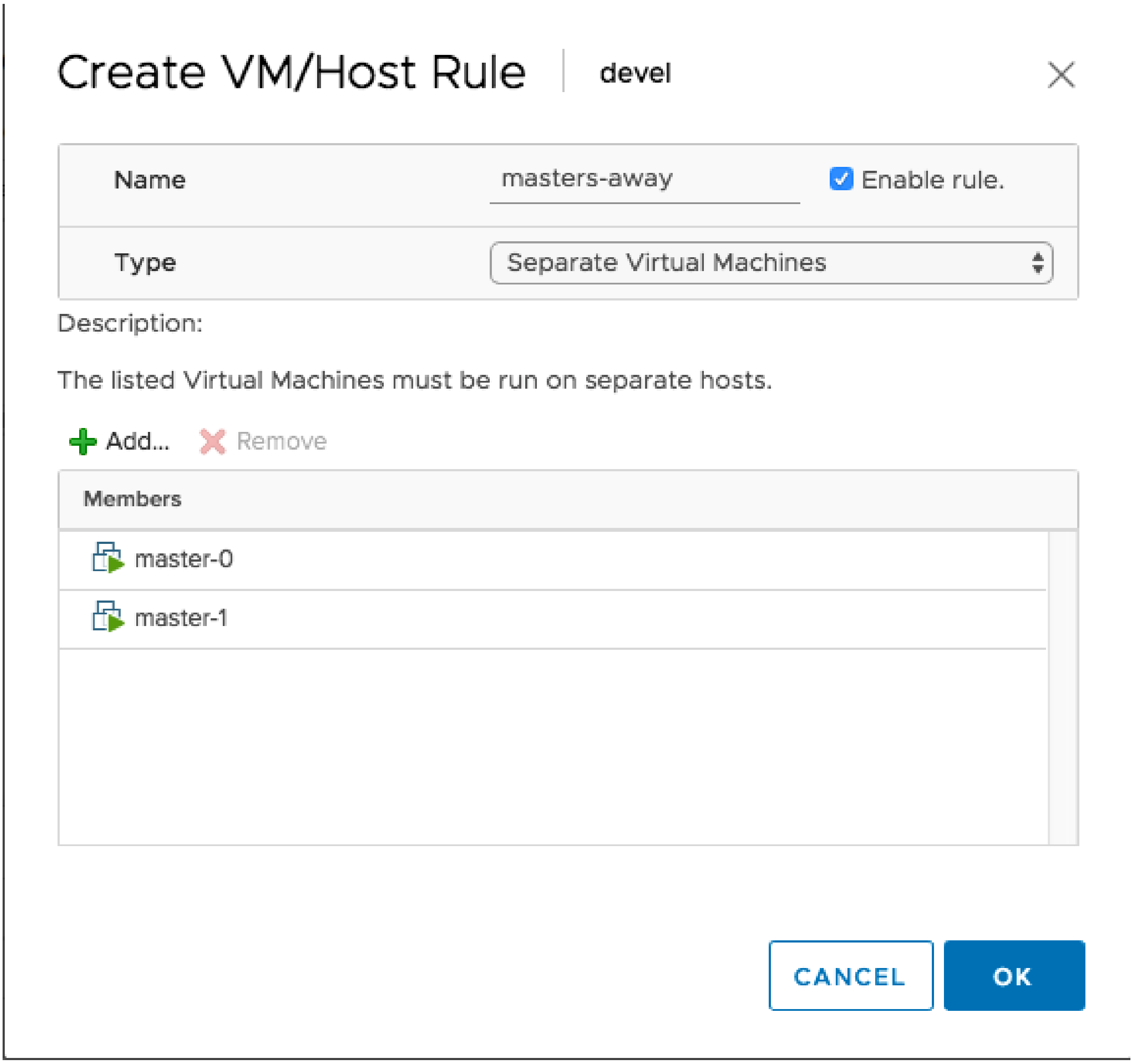
3.3.3.4. Post Ansible Deployment

Prior to Chapter 4, *Operational Management*, create DRS anti-affinity rules to ensure maximum availability for our cluster.

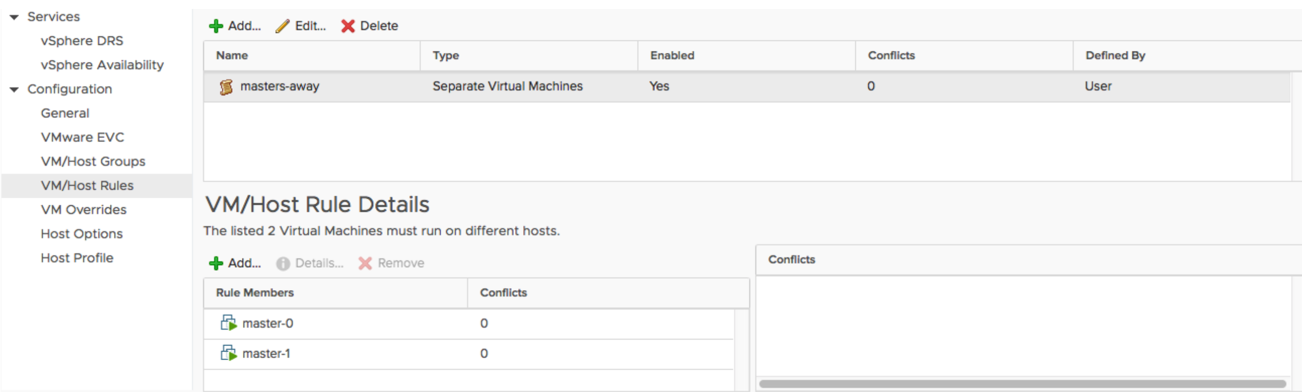
1. Open the VMware vCenter web client, select the cluster, choose configure.



1. Under Configuration, select VM/Host Rules.



1. Click add, and create a rules to keep the masters separate.



The following VMware documentation goes over creating and configuring anti-affinity rules in depth.

Once the playbooks have successfully completed, the next steps will be to perform the steps defined in Chapter 4, *Operational Management*. In the event that OpenShift failed to install, follow the steps in Appendix C: Appendix E, *Installation Failure* to restart the installation of OpenShift.

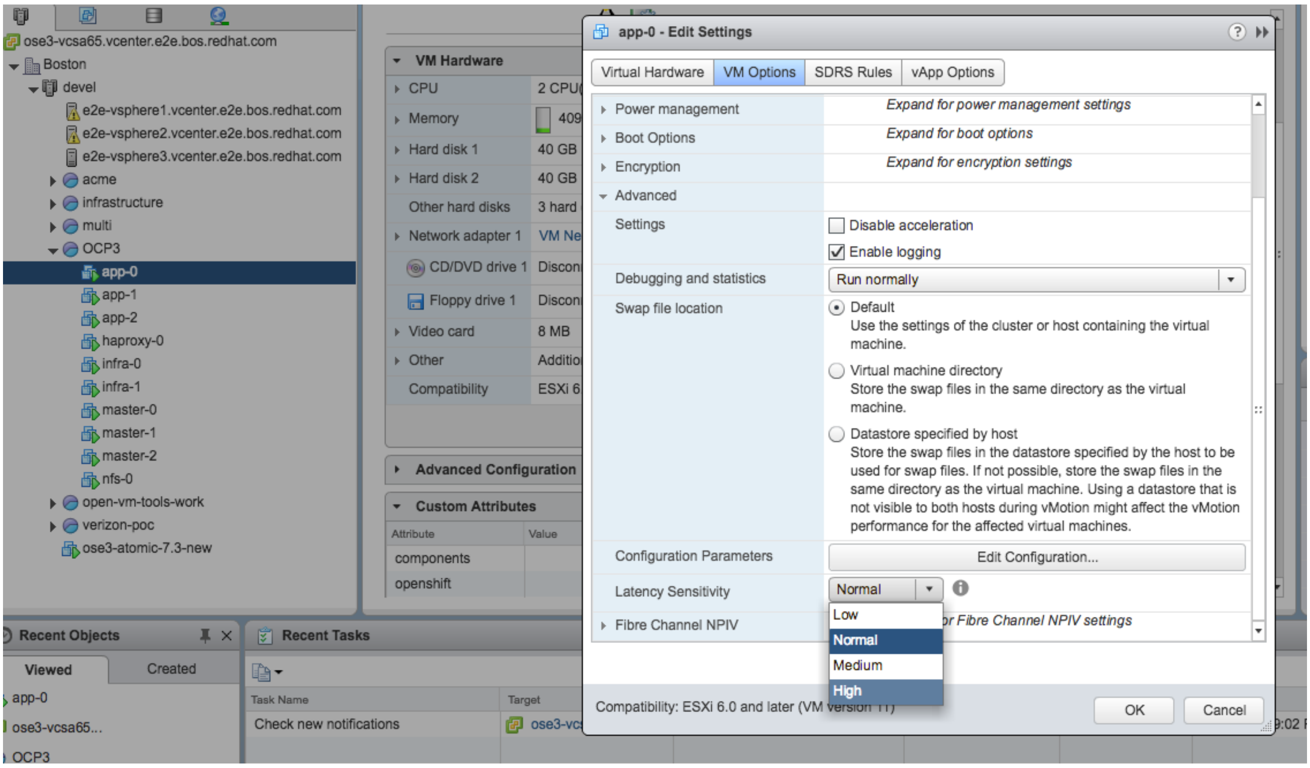
Lastly, set all of the VMs created to High VM Latency to ensure some additional tuning recommended by VMware for latency sensitive workloads as described here.

1. Open the VMware vCenter web client and under the virtual machines summary tab, in the 'VM Hardware' box select 'Edit Settings'.

2. Under, 'VM Options', expand 'Advanced'.

3. Select the 'Latency Sensitivity' dropbox and select 'High'.

Figure 3.2. VMware High Latency



3.3.3.4.1. Registry Console Selector (Optional)

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

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

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

NOTE

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

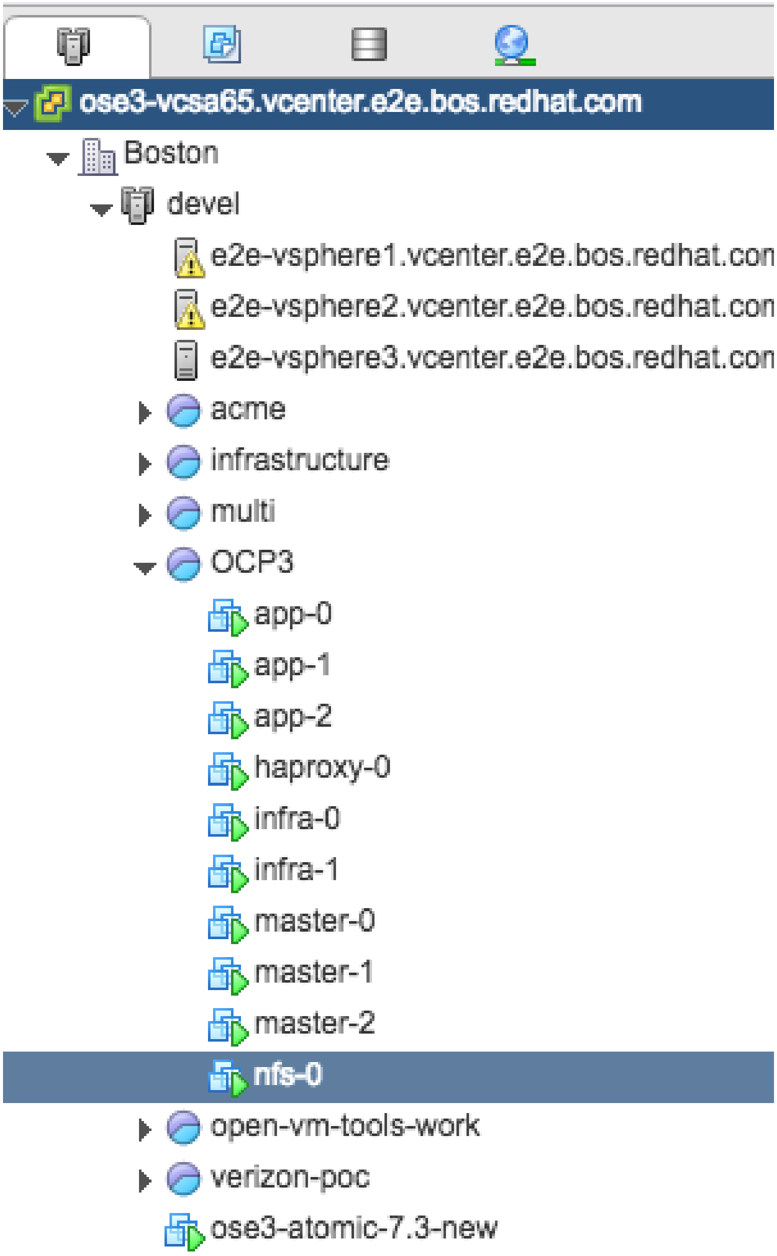
3.4. POST PROVISIONING RESULTS

At this point the infrastructure and Red Hat OpenShift Container Platform have been deployed.

Log into the VMware vCenter client and check the resources.

You should see the following:

Figure 3.3. VMware Completed Provisioning



- 3 Master nodes
- 2 Infrastructure nodes
- 3 Application nodes

Provided you built them:

- NFS Server
- HAproxy Server

At this point, the OpenShift URL will be available using the CNAME load balancer entry.

For example, https://ocp3-haproxy-0.example.com:8443

NOTE

When installing using this method the browser certificate must be accepted three times. The certificate must be accepted three times due to the number of masters in the cluster.

CHAPTER 4. OPERATIONAL MANAGEMENT

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

4.1. VALIDATING THE DEPLOYMENT

Now that OpenShift has been successfully deployed the deployment must be validated to ensure proper operation and functionality. An Ansible script in the git repository will allow for an application to be deployed which will test the functionality of the master, nodes, registry, and router. The playbook will test the deployment and clean up any projects and pods created during the validation run.

The playbook will perform the following steps:

Environment Validation

- Validate the public OpenShift load balancer address from the installation system
- Validate the public OpenShift load balancer address from the master nodes
- Validate the local master address
- Validate the health of the ETCD cluster to ensure all ETCD nodes are healthy
- Create a project in OpenShift called validate
- Create an OpenShift application
- Add a route for the application
- Validate the URL returns a status code of 200 or healthy
- Delete the validation project

NOTE

Ensure the URLs below and the tag variables match the variables used during deployment.

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ ocp-on-vmware.py --tag ocp-demo
```

4.2. GATHERING HOSTNAMES

With all of the steps that occur during the installation of Red Hat OpenShift, it is possible to lose track of the names of the instances in the recently deployed environment.

One option to get these hostnames is to run the Section 2.10, “Dynamic Inventory” script manually and look at our infra, app and master groups.

To help facilitate the Operational Management Chapter the following hostnames will be used.

- ocp3-master-0.example.com

- ocp3-master-1.example.com
- ocp3-master-2.example.com
- ocp3-infra-0.example.com
- ocp3-infra-1.example.com
- ocp3-infra-2.example.com
- ocp3-app-0.example.com
- ocp3-app-1.example.com
- ocp3-app-2.example.com

4.3. RUNNING DIAGNOSTICS

Perform the following steps from the first master node.

To run diagnostics, SSH into the first master node (ocp3-master-0.example.com). Direct access is provided to the first master node because of the configuration of the local "~/.ssh/config" file.

```
$ ssh root@ocp3-master-0.example.com
```

Connectivity to the first master node (ocp3-master-0.example.com) as the root user should have been established. Run the diagnostics that are included as part of the install.

```
# oc adm 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/haproxy-0-example-
com:8443/system:admin'
[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/haproxy-0-example-
com:8443/system:admin]
       Description: Validate client config context is complete and has
connectivity

Info:  The current client config context is 'default/haproxy-0-example-
com:8443/system:admin':
       The server URL is 'https://haproxy-0.example.com:8443'
       The user authentication is 'system:admin/haproxy-0-example-
com:8443'
       The current project is 'default'
       Successfully requested project list; has access to project(s):
         [default kube-system logging management-infra openshift
openshift-infra validate]

... output abbreviated ...

Info:  Checking journalctl logs for 'atomic-openshift-node' service
Info:  Checking journalctl logs for 'docker' service
```

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

WARN:  [DH0005 from diagnostic
MasterConfigCheck@openshift/origin/pkg/diagnostics/host/check_master_confi
g.go:52]
       Validation of master config file '/etc/origin/master/master-
config.yaml' warned:
       assetConfig.metricsPublicURL: Invalid value: "": required to view
cluster metrics in the console
       oauthConfig.identityProvider[0].provider.insecure: Invalid value:
true: validating passwords over an insecure connection could allow them to
be intercepted

[Note] Running diagnostic: NodeConfigCheck
       Description: Check the node config file

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

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

[Note] Summary of diagnostics execution (version v3.6.173.0.21):
[Note] Warnings seen: 5
```

NOTE

The warnings will not cause issues in the environment

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

4.4. CHECKING THE HEALTH OF ETCD

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.

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

```
$ ssh root@ocp3-master-0.example.com
```

```
# hostname
ocp3-master-0.example.com
# etcdctl -C https://$(hostname):2379 --ca-
file=/etc/origin/master/master.etcd-ca.crt \
--cert-file=/etc/origin/master/master.etcd-client.crt \
--key-file=/etc/origin/master/master.etcd-client.key cluster-health
member d3525253178d331c is healthy: got healthy result from
https://10.19.114.225:2379
member edf71ee725ea87b6 is healthy: got healthy result from
```

```
https://10.19.114.226:2379
member f2e1170c11b5cea8 is healthy: got healthy result from
https://10.19.114.227:2379
```

NOTE

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

4.5. EXAMINING DEFAULT NODE SELECTOR

As explained in a previous section, node labels are an important part of the Red Hat OpenShift environment. In the reference architecture installation, the default node selector is set to "role=apps" in "/etc/origin/master/master-config.yaml" on all of the master nodes. This configuration parameter is set by the Ansible role openshift-default-selector on all masters. The master API service must be restarted when making any changes to the master configuration.

SSH into the first master node (ocp-master-0.example.com) to verify the defaultNodeSelector is defined.

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

NOTE

Any changes to the master configuration require the restart of the master API service across all master nodes.

4.6. MANAGING OF MAXIMUM POD SIZE

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

Red Hat OpenShift Volume Quota

At launch time, user-data creates an XFS partition on the /dev/sdc 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.

SSH into the first infrastructure node (ocp-infra-0.example.com) to verify the entry exists within fstab.

```
$ cat /etc/fstab
/dev/sdc /var/lib/origin/openshift.local.volumes xfs gquota 0 0
```

Docker Storage Setup

The docker-storage-setup file is created at launch time by user-data. This file tells the Docker service to use /dev/sdb and create the volume group of docker-vol. Docker storage setup is performed on all master, infrastructure, and application nodes. Notice that the storage Driver is listing overlay2. This is a new driver option that has been backported to RHEL 7.4

SSH into the first infrastructure node (ocp-infra-0.example.com) to verify "/etc/sysconfig/docker-storage-setup" matches the information below.

```
$ cat /etc/sysconfig/docker-storage-setup
DEVS="/dev/sdb"
VG="docker-vol"
DATA_SIZE="95%VG"
STORAGE_DRIVER=overlay2
CONTAINER_ROOT_LV_NAME=dockerlv
CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
CONTAINER_ROOT_LV_SIZE=100%FREE
```

Red Hat OpenShift EmptyDir Quota

The perFSGroup setting restricts the ephemeral EmptyDir volume from growing larger than 512Mi. This EmptyDir quota is done on the infrastructure and application nodes. The configuration is not done on the masters due to the master nodes being unschedulable.

SSH into the first infrastructure node (ocp-infra-0.example.com) to verify "/etc/origin/node/node-config.yml" matches the information below.

```
$ vi /etc/origin/node/node-config.yml
...omitted...
volumeConfig:
  localQuota:
     perFSGroup: 512Mi
```

4.7. CHECKING THE YUM REPOSITORIES

In the Section 2.6, “Required Channels” the specific repositories for a successful OpenShift installation were defined. All systems should have the same subscriptions. 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.

Perform the following to verify the subscriptions match those defined in the Section 2.6, “Required Channels” section:

```
# yum repolist | awk '{print $2 }'
Loaded plugins: product-id, search-disabled-repos, subscription-manager
repo id
repo name
status
!rhel-7-fast-datapath-rpms/7Server/x86_64
Red Hat Enterprise Linux Fast Datapath (RHEL 7 Server) (RPMs)
38
!rhel-7-server-extras-rpms/x86_64
Red Hat Enterprise Linux 7 Server - Extras (RPMs)
619+20
```

```
!rhel-7-server-ose-3.6-rpms/x86_64
Red Hat OpenShift Container Platform 3.6 (RPMs)
503+20
!rhel-7-server-rpms/7Server/x86_64
Red Hat Enterprise Linux 7 Server (RPMs)
17,202
repolist: 18,362
```

NOTE

All other repositories are disabled and only those repositories defined in the Ansible role **rhsm** are enabled. If you use Red Hat Satellite Server, you will have the additional Satellite tools repo.

4.8. CONSOLE ACCESS

This section will cover logging into the Red Hat 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.

4.9. LOGGING INTO GUI (GRAPHICAL USER INTERFACE) CONSOLE AND DEPLOY AN APPLICATION

Perform the following steps from the local workstation.

Use the CNAME of the load balancer to access GUI console login.

Open a browser and access https://haproxy-0.example.com/console, where haproxy-0 is the CNAME of the load balancer to access GUI console login.

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. Wait for the application to finish deployment.

4.10. LOGGING INTO THE CLI (COMMAND-LINE INTERFACE) AND DEPLOY AN APPLICATION

Perform the following steps from your local workstation.

Install the oc client by visiting the public URL of the OpenShift deployment. For example, https://haproxy-0.example.com/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.

A token is required to login using GitHub OAuth and OpenShift. The token is presented on the https://haproxy-0.example.com/console/command-line page. Click the click to show token hyperlink and perform the following on the workstation in which the oc client was installed.

```
$ oc login https://haproxy-0.example.com --
token=fEAjn7LnZE6v5SOocCSRVmUWGBNIIEKbjD9h-Fv7p09
```

After the oc client is configured, 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://haproxy-0.example.com

http://test-app.apps.example.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.

4.11. EXPLORING THE ENVIRONMENT

4.11.1. Listing Nodes and Set Permissions

If you try to run the following command, it 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
```

Once the username has been established, log back into a master node and enable the appropriate permissions for your user. Perform the following step from the first master (ocp3-master-0.example.com).

```
# oc adm policy add-cluster-role-to-user cluster-admin
CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com
```

Attempt to list the nodes again and show the labels.

```
# oc get nodes --show-labels
NAME                         STATUS                      AGE
VERSION              LABELS
ocp3-app-0.example.com       Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-0.example.com,role=app
ocp3-app-1.example.com       Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-1.example.com,role=app
ocp3-app-2.example.com       Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-2.example.com,role=app
ocp3-infra-0.example.com     Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-infra-0.example.com,role=infra
ocp3-infra-1.example.com     Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-infra-1.example.com,role=infra
ocp3-infra-2.example.com     Ready                       2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-infra-2.example.com,role=infra
ocp3-master-0.example.com    Ready,SchedulingDisabled    2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
```

```
stname=ocp3-master-0.example.com,role=master
ocp3-master-1.example.com   Ready,SchedulingDisabled   2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-master-1.example.com,role=master
ocp3-master-2.example.com   Ready,SchedulingDisabled   2d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-master-2.example.com,role=master
```

4.11.2. Listing Router and Registry

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

NOTE

Perform the following steps from your the workstation.

```
# oc project default
# oc get all
# oc status
In project default on server https://haproxy-0.example.com:8443

https://docker-registry-default.apps.vcenter.e2e.bos.redhat.com
(passthrough) (svc/docker-registry)
  dc/docker-registry deploys docker.io/openshift3/ose-docker-
registry:v3.6.173.0.21
    deployment #1 deployed 23 hours ago - 1 pod

svc/kubernetes - 172.30.0.1 ports 443->8443, 53->8053, 53->8053

https://registry-console-default.apps.vcenter.e2e.bos.redhat.com
(passthrough) (svc/registry-console)
  dc/registry-console deploys
registry.access.redhat.com/openshift3/registry-console:v3.6
    deployment #1 deployed 23 hours ago - 1 pod

svc/router - 172.30.253.65 ports 80, 443, 1936
  dc/router deploys docker.io/openshift3/ose-haproxy-router:v3.6.173.0.21
    deployment #1 deployed 23 hours ago - 1 pod

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.

4.11.3. Exploring 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 NFS mount for storage. Using oc describe should help explain how HA works in this environment.

NOTE

Perform the following steps from your workstation:

```
$ oc describe svc/docker-registry
Name:   docker-registry
Namespace:  default
Labels:   docker-registry=default
Selector:  docker-registry=default
Type:   ClusterIP
IP:   172.30.252.119
Port:   5000-tcp 5000/TCP
Endpoints:  172.16.5.3:5000,172.16.8.2:5000
Session Affinity: ClientIP
No events.
```

Notice that the registry has two endpoints listed. Each of those endpoints represents a Docker container. The ClusterIP listed is the actual ingress point for the registries.

NOTE

Perform the following steps from the infrastructure node:

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

```
# oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-2-8b7c6   1/1       Running   0          2h
docker-registry-2-drhgz   1/1       Running   0          2h
k8s_registry.90479e7d_docker-registry-2-jueep_default_d5882b1f-5595-11e6-
a247-0eaf3ad438f1_ffc47696
```

```
# oc exec docker-registry-2-8b7c6 cat /config.yml
  version: 0.1
  log:
    level: debug
  http:
    addr: :5000
  storage:
    cache:
      blobdescriptor: inmemory
    filesystem:
      rootdirectory: /registry
    delete:
      enabled: true
  auth:
    openshift:
      realm: openshift

      # tokenrealm is a base URL to use for the token-granting registry
endpoint.
      # If unspecified, the scheme and host for the token redirect are
determined from the incoming request.
```

```
      # If specified, a scheme and host must be chosen that all registry
clients can resolve and access:
      #
      # tokenrealm: https://example.com:5000
  middleware:
    registry:
      - name: openshift
    repository:
      - name: openshift
        options:
          acceptschema2: false
          pullthrough: true
          enforcequota: false
          projectcachettl: 1m
          blobrepositorycachettl: 10m
    storage:
      - name: openshift
```

Additionally, registry information can be garnered via the oc command as well.

```
oc volume dc/docker-registry
deploymentconfigs/docker-registry
  pvc/registry-claim (allocated 20GiB) as registry-storage
    mounted at /registry
  secret/registry-certificates as registry-certificates
    mounted at /etc/secrets
```

4.11.4. Exploring 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(ocp-infra-0.example.com) is used.

The output below verifies Docker storage is not using a loop back device.

```
# docker info
...omitted...

 Server Version: 1.12.6
 Storage Driver: overlay2
  Backing Filesystem: xfs
 Logging Driver: journald
 Cgroup Driver: systemd
 Plugins:
  Volume: local
  Network: bridge host null overlay
  Authorization: rhel-push-plugin
 Swarm: inactive
 Runtimes: runc docker-runc
 Default Runtime: docker-runc
 Security Options: seccomp selinux
 Kernel Version: 3.10.0-693.el7.x86_64
 Operating System: OpenShift Enterprise
 OSType: linux
```

```
 Architecture: x86_64
 Number of Docker Hooks: 3
 CPUs: 2
 Total Memory: 15.5 GiB
 Name: master-0
 ID: DTNQ:U22B:R6GN:2ZES:AATU:SYVE:SYQU:6FKS:GATT:I4TI:CV3H:6MJ2
 Docker Root Dir: /var/lib/docker
...omitted...
Registries: registry.access.redhat.com (secure), docker.io (secure)
```

Verify three disks are attached to the instance. The disk /dev/sda is used for the OS, /dev/sdb is used for Docker storage, and /dev/sdc is used for EmptyDir storage for containers that do not use a persistent volume.

```
# fdisk -l

Disk /dev/sdb: 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: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048    83886079    41942016   8e  Linux LVM

Disk /dev/sdc: 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 /dev/sda: 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
```

4.12. TESTING FAILURE

In this section, reactions to failure are explored. After a successful installation and completion of the smoke tests nodes above, failure testing is executed.

4.12.1. Generating a Master Outage

NOTE

Perform the following steps from the VMware vCenter client.

Log into the VMware vCenter client. Under VMs and Templates, locate your running ocp3-master-2.example.com VM, select it, right click and change the state to Power off.

Ensure the console can still be accessed by opening a browser and accessing haproxy-0.example.com. At this point, the cluster is in a degraded state because only two out of three master nodes are running.

4.12.2. Observing the Behavior of etcd with a Failed Master Node

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

```
$ ssh root@ocp3-master-0.example.com
```

```
# hostname
ip-10-20-1-106.ec2.internal
# etcdctl -C https://$(hostname):2379 --ca-
file=/etc/origin/master/master.etcd-ca.crt --cert-
file=/etc/origin/master/master.etcd-client.crt --key-
file=/etc/origin/master/master.etcd-client.key cluster-health
member d3525253178d331c is healthy: got healthy result from
https://10.19.114.225:2379
failed to check the health of member edf71ee725ea87b6 on
https://10.19.114.226:2379: Get https://10.19.114.226:2379/health: dial
tcp 10.19.114.226:2379: i/o timeout
member edf71ee725ea87b6 is unreachable: [https://10.19.114.226:2379] are
all unreachable
member f2e1170c11b5cea8 is healthy: got healthy result from
https://10.19.114.227:2379
```

Notice how one member of the etcd cluster is now unreachable.

Restart ocp3-master-2.example.com by following the same steps in the VMWare console as noted above.

4.12.3. Generating an Infrastruture Node outage

This section shows what to expect when an infrastructure node fails or is brought down intentionally.

4.12.3.1. Confirming Application Accessibility

NOTE

Perform the following steps from the browser on a local workstation.

Before bringing down an infrastructure node, check behavior and ensure things are working as expected. The goal of testing an infrastructure node outage is to see how the OpenShift routers and registries behave. Confirm the simple application deployed from before is still functional. If it is not, deploy a new version. Access the application to confirm connectivity. As a reminder, to find the required information and ensure the application is still running, list the projects, change to the project that the application is deployed in, get the status of the application which includes the URL and access the application via that URL.

```
$ oc get projects
NAME                DISPLAY NAME   STATUS
openshift                          Active
```

```
openshift-infra                    Active
ttester                            Active
test-app1                          Active
default                            Active
management-infra                   Active

$ oc project test-app1
Now using project "test-app1" on server "https://haproxy-0.example.com".

$ oc status
In project test-app1 on server https://haproxy-0.example.com

http://php-test-app1.apps.sysdeseng.com to pod port 8080-tcp (svc/php-
prod)
  dc/php-prod deploys istag/php-prod:latest <-
    bc/php-prod builds https://github.com/openshift/cakephp-ex.git with
openshift/php:5.6
    deployment #1 deployed 27 minutes ago - 1 pod
```

Open a browser and ensure the application is still accessible.

4.12.3.2. Confirming Registry Functionality

This section is another step to take before initiating the outage of the infrastructure node to ensure that the registry is functioning properly. The goal is to push to the Red Hat OpenShift registry.

NOTE

Perform the following steps from a CLI on a local workstation and ensure that the oc client has been configured.

A token is needed so that the Docker registry can be logged into.

```
# oc whoami -t
feAeAgL139uFFF_72bcJlboTv7gi_bo373kf1byaAT8
```

Pull a new Docker image for the purposes of test pushing.

```
# docker pull fedora/apache
# docker images
```

Capture the registry endpoint. The svc/docker-registry shows the endpoint.

```
# oc get svc
NAME               CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
docker-registry    172.30.195.135   <none>        5000/TCP
23h
kubernetes         172.30.0.1       <none>        443/TCP,53/UDP,53/TCP
23h
registry-console   172.30.203.227   <none>        9000/TCP
23h
router             172.30.253.65    <none>        80/TCP,443/TCP,1936/TCP
23h
```

Tag the Docker image with the endpoint from the previous step.

```
# docker tag docker.io/fedora/apache
172.30.252.119:5000/openshift/prodapache
```

Check the images and ensure the newly tagged image is available.

```
# docker images
```

Issue a Docker login.

```
# docker login -u CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com
172.30.195.135:5000
```

```
# oc adm policy add-role-to-user admin
CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com -n openshift
# oc adm policy add-role-to-user system:registry
CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com
# oc adm policy add-role-to-user system:image-builder
CN=openshift,CN=Users,DC=e2e,DC=bos,DC=redhat,DC=com
```

Push the image to the OpenShift registry now.

```
# docker push 172.30.195.135:5000/openshift/prodapache
The push refers to a repository [172.30.195.135:5000/openshift/prodapache]
389eb3601e55: Layer already exists
c56d9d429ea9: Layer already exists
2a6c028a91ff: Layer already exists
11284f349477: Layer already exists
6c992a0e818a: Layer already exists
latest: digest:
sha256:ca66f8321243cce9c5dbab48dc79b7c31cf0e1d7e94984de61d37dfdac4e381f
size: 6186
```

4.12.3.3. Get Location of Router and Registry.

NOTE

Perform the following steps from the CLI of a local workstation.

Change to the default OpenShift project and check the router and registry pod locations.

```
# oc project default
Now using project "default" on server "https://haproxy-0.example.com".

# oc get pods
NAME                         READY     STATUS    RESTARTS   AGE
docker-registry-1-065t9      1/1       Running   0          23h
registry-console-1-h7vsc     1/1       Running   0          23h
router-1-ksxk1               1/1       Running   0          23h

# oc describe pod docker-registry-1-065t9 | grep -i node
```

```
Node:   infra-0/10.19.114.228
Node-Selectors: role=infra
# oc describe pod router-1-ksxk1 | grep -i node
Node:   infra-0/10.19.114.228
Node-Selectors: role=infra
```

4.12.3.4. Initiate the Failure and Confirm Functionality

> **NOTE**
>
> Perform the following steps from the VMware vCenter console

Log into the VMware console. Under VMs and Templates, locate your running infra-0.example.com VM, select it, right click and change the state to Power off. Wait a minute or two for the registry and pod to migrate over to infra-0. Check the registry locations and confirm that they are on the same node.

```
# oc describe pod docker-registry-1-065t9 | grep -i node
Node:   infra-1/10.19.114.229
Node-Selectors: role=infra
# oc describe pod router-1-ksxk1 | grep -i node
Node:   infra-1/10.19.114.229
Node-Selectors: role=infra
```

Follow the procedures above to ensure a Docker image can still be pushed to the registry now that infra-0 is down.

4.13. UPDATING THE OPENSHIFT DEPLOYMENT

Playbooks are provided to upgrade the OpenShift deployment when minor releases occur.

4.13.1. Performing the Upgrade

From the workstation that was used to clone the openshift-ansible-contrib repo run the following to ensure that the newest **`openshift-ansible`** playbooks and roles are available and to perform the minor upgrade against the deployed environment.

```
# yum update atomic-openshift-utils ansible
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ ocp-on-vmware.py --tag ocp-update*
```

4.13.2. Upgrading and Restarting the OpenShift Environment (Optional)

The **`openshift-minor-update.yaml`** playbook will not restart the instances after updating occurs. Restarting the nodes including the masters can be completed by adding the following line to the **`minor-update.yaml`** playbook.

```
$ cd ~/git/openshift-ansible-contrib/playbooks
$ vi minor-update.yaml
    openshift_rolling_restart_mode: system
```

4.13.3. Specifying the OpenShift Version when Upgrading

The deployed OpenShift environment may not be the latest major version of OpenShift. The **`minor-update.yaml`** allows for a variable to be passed to perform an upgrade on previous versions. Below is an example of performing the upgrade on a 3.6 environment.

```
# yum update atomic-openshift-utils ansible
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ vi ocp-on-vmware.ini
# OpenShift Version
openshift_vers=v3_6
$ ocp-on-vmware.py --create_ocp_vars
$ ocp-on-vmware.py --tag ocp-update
```

CHAPTER 5. PERSISTENT STORAGE CONCEPTS

Container storage by default is ephemeral. For example, if a new container build occurs then data is lost because the storage is non-persistent. If a container terminates then all of the changes to its local filesystem are lost. OpenShift offers many different types of persistent storage. Persistent storage ensures that data that should persist between builds and container migrations is available. The different storage options can be found here.

5.1. STORAGE CLASSES

The **`StorageClass`** resource object describes and classifies different types of storage that can be requested, as well as provides a means for passing parameters to the backend for dynamically provisioned storage on demand. **`StorageClass`** objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. **`Cluster Administrators (cluster-admin)`** or **`Storage Administrators (storage-admin)`** define and create the **`StorageClass`** objects that users can use without needing any intimate knowledge about the underlying storage volume sources. Because of this the naming of the **`storage class`** defined in the **`StorageClass`** object should be useful in understanding the type of storage it maps to (ie., **`HDD`** vs **`SDD`**).

5.2. PERSISTENT VOLUMES

Container storage is defined by the concept of **`persistent volumes`** (pv) which are OpenShift Container Platform objects that allow for storage to be defined and then used by pods to allow for data persistence. Requesting of **`persistent volumes`** (pv) is done by using a **`persistent volume claim`** (pvc). This claim, when successfully fulfilled by the system will also mount the persistent storage to a specific directory within a pod or multiple pods. This directory is referred to as the **`mountPath`** and facilitated using a concept known as **`bind-mount`**.

CHAPTER 6. PERSISTENT STORAGE OPTIONS

There are many possible storage options to be used within projects. When choosing a persistent storage backend ensure that the backend supports the scaling, speed, and redundancy that the project requires. This reference architecture will focus on VMware provider storage, NFS, Container-Native Storage (CNS), and Container-Ready Storage(CRS).

6.1. VSPHERE CLOUD PROVIDER CONFIGURATION

The most basic **`StorageClass`** that can be created uses the existing VMWare environment to create storage for containers. This requires that specific configuration is in place to allow OpenShift to create and attach volumes.

Starting with OpenShift 3.6 a **`StorageClass`** can be created that utilizes the DataStore within the VMware environment. The **`ocp-on-vmware`** script configures a default **`StorageClass`** using the same **`vcenter_datastore`** where the VMs are deployed. The vSphere cloud provider configuration is creating automatically via the ansible playbooks.

The **`vsphere.conf`** file contains specific information that relates to the VMWare environment in which OpenShift is deployed. This file is stored on all nodes of the cluster. The example below shows the values of an example **`vsphere.conf`**. The **`vsphere.conf`** file works in conjunction with the **`master-config.yaml`** and **`node-config.yaml`** to identify and attach volumes to hosts to be used for persistent storage.

```
$ cat /etc/vsphere/vsphere.conf
[Global]
user = "administrator@vsphere.local"
password = "*******"
server = "10.*.*.25"
port = 443
insecure-flag = 1
datacenter = Boston
datastore = ose3-vmware-prod
working-dir = /Boston/vm/ocp36/
[Disk]
scsicontrollertype = pvscsi
```

6.1.1. vSphere Storage Class

As stated in the previous section a **`StorageClass`** is created and configured when using the **`ocp-on-vmware`** script. To view the configuration of the **`StorageClass`** log into the first master node and view the file **`/root/cloud-provider-storage-class.yaml`**.

```
$ cat cloud-provider-storage-class.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: "ose3-vmware-prod"
provisioner: kubernetes.io/vsphere-volume
parameters:
    diskformat: zeroedthick
    datastore: "ose3-vmware-prod"

Since the StorageClass object is created at default. The oc command can be
```

```
used
to verify the StorageClass exists.

$ oc get sc
NAME                TYPE
ose3-vmware-prod    kubernetes.io/vsphere-volume

$ oc describe sc ose3-vmware-prod
Name:  ose3-vmware-prod
IsDefaultClass: No
Annotations: <none>
Provisioner: kubernetes.io/vsphere-volume
Parameters: datastore=ose3-vmware-prod,diskformat=zeroedthick
Events:  <none>
```

6.1.2. VMDK Dynamic provisioning

With the `StorageClass` object created OpenShift can now dynamically provision VMDKs for persistent storage for containers within the OpenShift environment.

NOTE

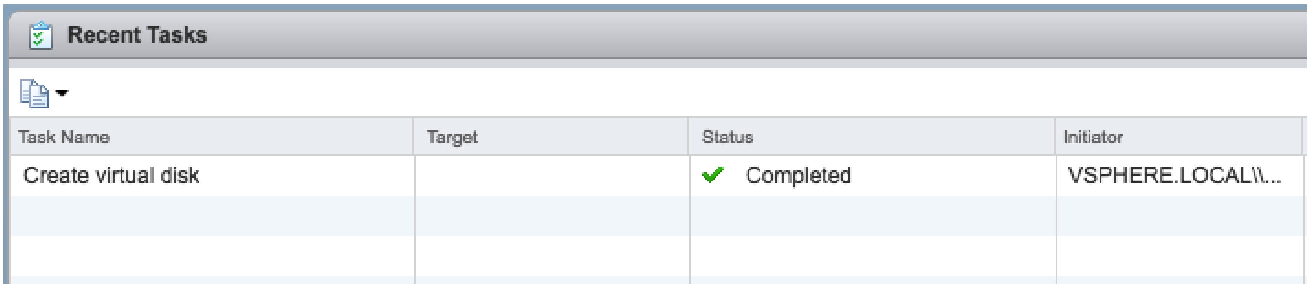
This **pvc** will need to be initiated on the newly created OpenShift cluster.

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

kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: ose3-vmware-prod
  annotations:
    volume.beta.kubernetes.io/storage-class: ose3-vmware-prod
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi

$ oc create -f storage-class-vmware-claim.yaml
$ oc describe pvc ose3-vmware-prod

Name:  ose3-vmware-prod
Namespace: default
StorageClass: ose3-vmware-prod
Status:  Bound
Volume:  pvc-cc8a9970-7c76-11e7-ae86-005056a571ee
Labels:  <none>
Annotations: pv.kubernetes.io/bind-completed=yes
  pv.kubernetes.io/bound-by-controller=yes
  volume.beta.kubernetes.io/storage-class=vmware-datastore-ssd
  volume.beta.kubernetes.io/storage-provisioner=kubernetes.io/vsphere-
volume
Capacity: 2Gi
```

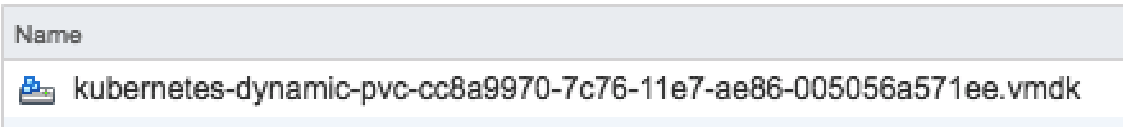
```
Access Modes: RWO
Events:
  FirstSeen LastSeen Count From    SubObjectPath Type  Reason    Message
  --------- -------- ----- ----    ------------- -------- ------   ----
---
  19s  19s  1 persistentvolume-controller   Normal  ProvisioningSucceeded
Successfully provisioned volume pvc-cc8a9970-7c76-11e7-ae86-005056a571ee
using kubernetes.io/vsphere-volume
```

Now, in vCenter, a couple of changes are initiated:

Here the new disk is created.



Secondly, the disk is ready to be consumed by a VM to be attached to a POD.



While datastores are generally accessible via shared storage to all the nodes of your cluster, the VMDKs are tied to a specific machine. This explains the **`ReadWriteOnce`** limitation of the persistent storage.

6.2. CREATING AN NFS PERSISTENT VOLUME

Login to the first Red Hat OCP master to define the persistent volume. Creating persistent volumes requires privileges that a default user account does not have. For this example, the system:admin account will be used due to the account having cluster-admin privileges.

We will use a different NFS share on the same NFS server. Remember, different tiers of storage should be assigned as needed by different workloads. In this example, we are just providing an outline for any future PVCs you choose to create.

For more information regarding persistent volume claims on NFS take a look at the documentation.

```
$ vi nfs-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv001
spec:
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteOnce
  nfs:
```

```
    path: /pv1
    server: nfs-0.example.com
  persistentVolumeReclaimPolicy: Recycle
```

The cluster-admin or storage-admin can then create the **PV** object using the yaml file.

```
$ oc create -f nfs-pv.yaml
```

6.2.1. Creating an NFS Persistent Volumes Claim

The persistent volume claim (**PVC**) will change the pod from using EmptyDir non-persistent storage to storage backed by an NFS volume or **PV** as created above. The **PVC** will be created as long as the storage size is equal or greater to the **PV** and the **accessModes** are the same (i.e., ReadWriteOnce).

```
$ vi nfs-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db
spec:
  accessModes:
  - ReadWriteOnce
  resources:
     requests:
       storage: 5Gi


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

6.2.2. Deleting a Persistent Volumes Claim

There may become a point in which a **PVC** is no longer necessary for a project. The following can be done to remove the **PVC**.

```
$ oc delete pvc db
persistentvolumeclaim "db" deleted
$ oc get pvc db
No resources found.
Error from server: persistentvolumeclaims "db" not found
```

6.3. CONTAINER-NATIVE STORAGE OVERVIEW

Container-Native Storage (CNS) provides dynamically provisioned persistent storage for containers on OpenShift Container Platform with common semantics across cloud virtual, cloud providers and bare-metal deployments. **CNS** relies on VMware volumes **VMDK** or **Raw Device Mapping(RDM)** mounted on the OCP nodes and uses software-defined storage provided by Red Hat Gluster Storage. **CNS** runs Red Hat Gluster Storage containerized allowing OCP storage pods to spread across the cluster and across VMware servers. **CNS** enables the requesting and mounting of **Gluster** storage across one or many containers with access modes of either **ReadWriteMany(RWX)**, **ReadOnlyMany(ROX)** or **ReadWriteOnce(RWO)**. **CNS** can also be used to host the OCP registry.

6.3.1. Prerequisites for Container-Native Storage

Deployment of Container-Native Storage (CNS) on OpenShift Container Platform (OCP) requires at least three OpenShift nodes with at least one unused block storage device attached on each of the nodes. Dedicating three OpenShift nodes to **CNS** will allow for the configuration of one **StorageClass** object to be used for applications. If two types of StorageClass objects are required (e.g. HDD and SSD types) then a minimum of six CNS nodes must be deployed and configured. This is because only a single **CNS** container per OpenShift node is supported.

If the **CNS** instances will serve dual roles such as hosting application pods and **glusterfs** pods ensure the instances have enough resources to support both operations. **CNS** hardware requirements state that there must be 32GB of RAM per node or virtual machine. There is a current limit of 300 volumes or **PVs** per 3 node **CNS** cluster.

NOTE

If there is a need to use the **CNS** instances for application or infrastructure pods the label role=app can be applied to the nodes. In the adoption phase it is expected that the platform will run less than 300 **PVs** and the remaining memory on the 32GB instance is enough to serve the application pods.

6.3.2. Deployment of CNS Infrastructure

A python script named **add-node.py** is provided in the **openshift-ansible-contrib** git repository which will deploy three nodes or virtual machines, add the virtual machines to the OpenShift environment with specific OCP labels and add a **VMDK** volume to each node as an available block device to be used for **CNS**.

NOTE

Section 7.2, "Introduction to add-node.py" provides an introduction and overview of **add-node.py**

Do the following from the workstation performing the deployment of the OCP Reference Architecture. The **ocp-on-vmware.ini** file used to create the OpenShift deployment must be in the directory where **add-node.py** is ran from. There are two entries in the **ocp-on-vmware.ini** file that must be added or modified. They are the **vcenter_datastore** and **container_storage**. The VMware datastore entered is where all of the new OCP **CNS** nodes or virtual machines will be stored so verify that this datastore has at least 1TB of available storage.

NOTE

The initial deployment should be on a **vcenter_datastore** on the first host in the cluster. After successful deployment, two of the three new OCP **CNS** virtual machines should be migrated to unique VMware hypervisors in the cluster & datastores from **vcenter_datastore**.

```
$ cd /root/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/
$ cat ocp-on-vmware.ini
….omitted….
# folder/cluster/resource pool in vCenter to organize VMs
vcenter_folder=ocp3
```

```
vcenter_datastore=DPLHP380G9-10-SS200-2
vcenter_cluster=OCP3
vcenter_resource_pool=OCP3
vcenter_datacenter=vDPL
….omitted….
# persistent container storage: none, crs, cns
container_storage=cns
```

```
$ ./add-node.py --node_type=storage
Configured inventory values:
  console_port:  8443
  deployment_type:  openshift-enterprise
  openshift_vers:  v3_6
  vcenter_host:  172.0.10.246
  vcenter_username:  administrator@vsphere.local
  vcenter_password: *******
  vcenter_template_name:  ocp-server-template-2.0.2-new
  vcenter_folder:  ocp3
  vcenter_datastore:  DPLHP380G9-10-SS200-2
  vcenter_cluster:  OCP3
  vcenter_resource_pool:  OCP3
  vcenter_datacenter:  vDPL
  public_hosted_zone:  dpl.local
  app_dns_prefix:  apps
  vm_dns:  172.0.10.241
  vm_gw:  172.0.10.2
  vm_netmask:  255.255.255.0
  vm_network:  "Private"
  rhel_subscription_user: rhn_user
  rhel_subscription_pass: *******
  rhel_subscription_server:
  rhel_subscription_pool:  Red Hat OpenShift Container Platform, Premium*
  byo_lb:  False
  lb_host:  haproxy-0
  byo_nfs:  False
  nfs_host:  nfs-0
  nfs_registry_mountpoint:  /exports
  master_nodes:  3
  infra_nodes:  3
  app_nodes:  6
  storage_nodes:  0
  vm_ipaddr_start:  172.0.10.201
  ocp_hostname_prefix:  ocp3-
  auth_type:  ldap
  ldap_user:  openshift
  ldap_user_password: *******
  ldap_fqdn:  dpl.local
  openshift_hosted_metrics_deploy:  false
  openshift_sdn:  redhat/openshift-ovs-subnet
  containerized:  false
  container_storage:  cns
  tag:  None
  node_number:  1
  ini_path:  ./ocp-on-vmware.ini
  node_type:  storage
```

```
Continue creating the inventory file with these values? [y/N]: y
Gluster topology file created using /dev/sdd: topology.json
Inventory file created: add-node.json
host_inventory:
  ocp3-app-cns-0:
    guestname: ocp3-app-cns-0
    ip4addr: 172.0.10.211
    tag: storage
  ocp3-app-cns-1:
    guestname: ocp3-app-cns-1
    ip4addr: 172.0.10.212
    tag: storage
  ocp3-app-cns-2:
    guestname: ocp3-app-cns-2
    ip4addr: 172.0.10.213
    tag: storage

Continue adding nodes with these values? [y/N]:
```

NOTE

The script above is optional. Instances can be deployed without using this script as long as the new instances are added to the OCP cluster using the OCP add node playbooks or using the **add-node.py**.

6.3.3. Firewall Prerequisites

The correct firewall ports are automatically applied on the nodes deployed using the **add-node.py** with option **--node_type=storage** script. If the script has not been used to create the new OCP **CNS** nodes then the ports will need to be configured manually. On each of the OCP nodes that will host the Red Hat Gluster Storage container, add the following rules to /etc/sysconfig/iptables and reload the iptables:

```
$ cat /etc/sysconfig/iptables
….omitted….
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 24007 -j
ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 24008 -j
ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 2222 -j
ACCEPT
-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m multiport --dports
49152:49664 -j ACCEPT
….omitted….

# systemctl reload iptables
```

6.4. CNS INSTALLATION OVERVIEW

The process for creating a **CNS** deployment on OpenShift Container Platform starts with creating an OCP project that will host the **glusterfs** pods and the **CNS** service/pod/route resources. The Red Hat utility **cns-deploy** will automate the creation of these resources. After the creation of the **CNS** components then a **StorageClass** can be defined for creating Persistent Volume Claims (PVCs) against the Container-Native Storage Service. **CNS** uses services from **heketi** to create a **gluster**

Trusted Storage Pool.

Container-Native Storage service are Red Hat Gluster Storage container pods running on OCP Nodes managed by a Heketi Service. A single **heketi** service can manage multiple **CNS** Trusted Storage Pools. This is implemented using a **DaemonSet**, a specific way to deploy containers to ensure nodes participating in that **DaemonSet** always run exactly one instance of the **glusterfs** image as a pod. **DaemonSets** are required by **CNS** because the **glusterfs** pods must use the host's networking resources. The default configuration ensures that no more than one **glusterfs** pod can run on one OCP node.

6.4.1. Creating CNS Project

These activities should be done on the master due to the requirement of setting the **node selector**. The account performing the **CNS** activities must be a cluster-admin. Example shown below for the openshift user.

```
$ oc adm policy add-cluster-role-to-user cluster-admin
cn=openshift,cn=users,dc=example,dc=com
```

The project name used for this example will be **storage** but the project name can be whatever value an administrator chooses.

If the **CNS** nodes will only be used for **CNS** then a **node-selector** should be supplied.

```
$ oc adm new-project storage --node-selector='role=storage'
```

If the **CNS** nodes will serve the role of being used for both **CNS** and application pods then a **node-selector** does not need to supplied.

```
$ oc adm new-project storage
```

An **oc adm** policy must be set to enable the deployment of the privileged containers as Red Hat Gluster Storage containers can only run in the privileged mode.

```
$ oc project storage
$ oc adm policy add-scc-to-user privileged -z default
```

6.4.2. Gluster Deployment Prerequisites

Perform the following steps from CLI on a local or deployment workstation and ensure that the oc client has been installed and configured. An entitlement for **Red Hat Gluster Storage** is required to install the **Gluster** services.

```
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-server-rpms
# subscription-manager repos --enable=rhel-7-server-rpms
# yum install -y cns-deploy heketi-client
```

6.4.3. Deploying Container-Native Storage

The Container-Native Storage **glusterfs** and **heketi** pods, services, and **heketi** route are created using the **cns-deploy** tool which was installed during the prerequisite step.

A **heketi** topology file is used to create the Trusted Storage Pool. The topology describes the OpenShift nodes that will host Red Hat Gluster Storage services and their attached storage devices. A sample topology file **topology-sample.json** is installed with the **heketi-client** package in the **/usr/share/heketi/** directory.

> **NOTE**
>
> These activities should be done on the workstation where **cns-deploy** and **heketi-client** were installed. Ensure that the OpenShift client has the cluster-admin privilege before proceeding.

Below is an example of 3 node **topology.json** file with **/dev/sdd** as the VMware volume or device used for **CNS**. This file, **topology.json** is created and placed in the directory where **add-node.py --node_type=storage** is issued from.

Edit the values of node.hostnames.manage, node.hostnames.storage, and devices in the **topology.json** file based on the the OCP nodes that have been deployed in the previous step if the add-node.py script was not used.

```
$ vi topology.json
{
    "clusters": [
        {
            "nodes": [
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-0.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.211"
                            ]
                        },
                        "zone": 1
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-1.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.212"
                            ]
                        },
                        "zone": 2
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
```

```
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-2.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.213"
                            ]
                        },
                        "zone": 3
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                }
            ]
        }
    ]
}
```

Ensure that the storage project is the current project.

```
$ oc project storage
Already on project "storage" on server "https://ocp3-haproxy-
0.dpl.local:8443".
```

To launch the deployment of **CNS** the script cns-deploy will be used. It is advised to specify an **`admin-key`** and **`user-key`** for security reasons when launching the topology. Both **`admin-key`** and **`user-key`** are user defined values, they do not exist before this step. The **`heketi`** admin key (password) will later be used to create a **`heketi-secret`** in OCP. Be sure to note these values as they will be needed in future operations. The cns-deploy script will prompt the user before proceeding.

```
$ cns-deploy -n storage -g topology.json --admin-key 'myS3cr3tpassw0rd' --
user-key 'mys3rs3cr3tpassw0rd'
Welcome to the deployment tool for GlusterFS on Kubernetes and OpenShift.

Before getting started, this script has some requirements of the execution
environment and of the container platform that you should verify.

The client machine that will run this script must have:
 * Administrative access to an existing Kubernetes or OpenShift cluster
 * Access to a python interpreter 'python'
 * Access to the heketi client 'heketi-cli'

Each of the nodes that will host GlusterFS must also have appropriate
firewall
rules for the required GlusterFS ports:
 * 2222  - sshd (if running GlusterFS in a pod)
 * 24007 - GlusterFS Daemon
 * 24008 - GlusterFS Management
 * 49152 to 49251 - Each brick for every volume on the host requires its
own
   port. For every new brick, one new port will be used starting at 49152.
```

```
We
   recommend a default range of 49152-49251 on each host, though you can
adjust
   this to fit your needs.

In addition, for an OpenShift deployment you must:
 * Have 'cluster_admin' role on the administrative account doing the
deployment
 * Add the 'default' and 'router' Service Accounts to the 'privileged' SCC
 * Have a router deployed that is configured to allow apps to access
services
   running in the cluster

Do you wish to proceed with deployment?

[Y]es, [N]o? [Default: Y]: Y
Using OpenShift CLI.
NAME           STATUS    AGE
storage   Active    32m
Using namespace "storage".
Checking that heketi pod is not running ... OK
template "deploy-heketi" created
serviceaccount "heketi-service-account" created
template "heketi" created
template "glusterfs" created
role "edit" added: "system:serviceaccount:storage:heketi-service-account"
node "ocp3-app-cns-0.dpl.local" labeled
node "ocp3-app-cns-1.dpl.local" labeled
node "ocp3-app-cns-2.dpl.local" labeled
daemonset "glusterfs" created
Waiting for GlusterFS pods to start ... OK
service "deploy-heketi" created
route "deploy-heketi" created
deploymentconfig "deploy-heketi" created
Waiting for deploy-heketi pod to start ... ^[[AOK
Creating cluster ... ID: 8578ad5529c354e9d21898cba4c4a1c9
Creating node ocp3-app-cns-0.dpl.local ... ID:
8424caa3b7bd9e9098ef200fe8033edf
Adding device /dev/sdd ... OK
Creating node ocp3-app-cns-1.dpl.local ... ID:
ca9344f2390798304b1a7877ecc0bb85
Adding device /dev/sdd ... OK
Creating node ocp3-app-cns-2.dpl.local ... ID:
ca2a1703a3ed68979eef307abe2f1770
Adding device /dev/sdd ... OK
heketi topology loaded.
Saving heketi-storage.json
secret "heketi-storage-secret" created
endpoints "heketi-storage-endpoints" created
service "heketi-storage-endpoints" created
job "heketi-storage-copy-job" created
deploymentconfig "deploy-heketi" deleted
route "deploy-heketi" deleted
service "deploy-heketi" deleted
job "heketi-storage-copy-job" deleted
pod "deploy-heketi-1-cjt16" deleted
```

```
secret "heketi-storage-secret" deleted
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ... OK
heketi is now running.
Ready to create and provide GlusterFS volumes.
```

After successful deploy validate that there are now 3 **glusterfs** pods and 1 **heketi** pod in the storage project.

```
$ oc get pods -o=wide
NAME                     READY     STATUS    RESTARTS   AGE     IP
NODE
glusterfs-496n0          1/1       Running   0          14m
172.16.2.4     ocp3-app-cns-0
glusterfs-hx6z1          1/1       Running   0          14m
172.16.3.4     ocp3-app-cns-1
glusterfs-mtr2t          1/1       Running   0          14m
172.16.4.4     ocp3-app-cns-2
heketi-1-xllk8           1/1       Running   0          9m
172.16.6.2     ocp3-app-cns-0
```

6.4.4. Exploring Heketi

A new route will be created for the **heketi** service that was deployed during the run of the **cns-deploy** script. The **heketi** route URL is used by the **heketi-client**. The same route URL will be used to create **StorageClass** objects.

The first step is to find the endpoint for the **heketi** service and then set the environment variables for the route of the **heketi** server, the **heketi cli user**, and the **heketi cli key**.

```
$ oc get routes heketi
NAME     HOST/PORT     PATH      SERVICES   PORT      TERMINATION
WILDCARD
heketi   heketi-storage.apps.dpl.local            heketi     <all>
None

$ export HEKETI_CLI_SERVER=http://heketi-storage.apps.dpl.local
$ export HEKETI_CLI_USER=admin
$ export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

To validate that **heketi** loaded the topology and has the cluster created execute the following commands:

```
$ heketi-cli topology info
... ommitted ...
$ heketi-cli cluster list
Clusters:
8578ad5529c354e9d21898cba4c4a1c9
```

Use the output of the cluster list to view the nodes and volumes within the cluster.

```
$ heketi-cli cluster info 8578ad5529c354e9d21898cba4c4a1c9
```

```
Cluster id: 8578ad5529c354e9d21898cba4c4a1c9
Nodes:
8424caa3b7bd9e9098ef200fe8033edf
ca2a1703a3ed68979eef307abe2f1770
ca9344f2390798304b1a7877ecc0bb85
Volumes:
6e59417c9c3c5dec057607e5450dc9ed
```

6.4.5. Store the Heketi Secret

OpenShift Container Platform allows for the use of secrets so that items do not need to be stored in clear text. The admin password for **heketi**, specified during installation with cns-deploy, should be stored in base64-encoding. OCP can refer to this secret instead of specifying the password in clear text.

To generate the base64-encoded equivalent of the admin password supplied to the cns-deploy command perform the following.

```
$ echo -n myS3cr3tpassw0rd | base64
bXlzZWNyZXRwYXNzdzByZA==
```

On the master or workstation with the OpenShift client installed and a user with cluster-admin privileges use the base64 password string in the following YAML to define the secret in OpenShift's default project or namespace.

```
$ vi heketi-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlzZWNyZXRwYXNzdzByZA==
type: kubernetes.io/glusterfs
```

Create the secret by using the following command.

```
$ oc create -f heketi-secret.yaml
secret "heketi-secret" created
```

6.4.6. Creating a Storage Class

The cluster-admin or storage-admin can perform the following which will allow for dynamically provisioned **CNS** storage on demand. The key benefit of this storage is that the persistent storage created can be configured with access modes of ReadWriteOnce(RWO), ReadOnlyMany (ROX), or ReadWriteMany (RWX) adding much more flexibility than cloud provider specific storage.

If Multiple types of CNS storage are desired, additional StorageClass objects can be created to realize multiple tiers of storage defining different types of storage behind a single **heketi** instance. This will involve deploying more **glusterfs** pods on additional storage nodes (one gluster pod per OpenShift node) with different type and quality of volumes attached to achieve the desired properties of a tier (e.g. SSDs for “fast” storage, magnetic for “slow” storage). For the examples below we will assume that only one type of storage is required.

Perform the following steps from CLI on a workstation or master node where the OpenShift client has been configured.

```
$ oc project storage
$ oc get routes heketi
NAME      HOST/PORT     PATH      SERVICES   PORT      TERMINATION
WILDCARD
heketi    heketi-storage.apps.dpl.local            heketi     <all>
None

$ export HEKETI_CLI_SERVER=http://heketi-storage.apps.dpl.local
$ export HEKETI_CLI_USER=admin
$ export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

Record the cluster id of the **`glusterfs`** pods in **`heketi`**.

```
$ heketi-cli cluster list
Clusters:
8578ad5529#354e9d21898cba4c4a1c9
```

The **`StorageClass`** object requires both the cluster id and the **`heketi`** route to be defined to successfully created. Use the information from the output of **`heketi-cli cluster list`** and **`oc get routes heketi`** to fill in the resturl and clusterid. For OpenShift 3.4, the value of **`clusterid`** is not supported for the **`StorageClass`** object. If a value is provided the **`StorageClass`** object will fail to create for OpenShift version 3.4. The failure occurs because OpenShift 3.4 can only have a single **TSP** or **CNS** cluster.

OpenShift 3.4

```
$ vi glusterfs-storageclass-slow.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: http://heketi-storage.apps.dpl.local
  restauthenabled: "true"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
```

The **`StorageClass`** object can now be created using this yaml file.

```
$ oc create -f glusterfs-storageclass-slow.yaml
```

OpenShift 3.6

```
$ vi glusterfs-storageclass-slow.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-slow
provisioner: kubernetes.io/glusterfs
```

```
parameters:
  resturl: http://heketi-storage.apps.dpl.local
  clusterid: 8578ad5529c354e9d21898cba4c4a1c9
  restauthenabled: "true"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
```

The **StorageClass** object can now be created using this yaml file.

```
$ oc create -f glusterfs-storageclass-slow.yaml
```

To validate the **StorageClass** object was created perform the following.

```
$ oc get storageclass gluster-cns-slow
NAME             TYPE
gluster-cns-dd   kubernetes.io/glusterfs
$ oc describe storageclass gluster-cns-slow
Name:  gluster-cns-slow
IsDefaultClass: No
Annotations: <none>
Provisioner: kubernetes.io/glusterfs
Parameters:
clusterid=8578ad5529c354e9d21898cba4c4a1c9,restauthenabled=true,resturl=ht
tp://heketi-storage.apps.dpl.local,restuser=admin,secretName=heketi-
secret,secretNamespace=default
No events.
```

6.5. CREATING A PERSISTENT VOLUME CLAIM

The **StorageClass** object created in the previous section allows for storage to be dynamically provisioned using the **CNS** resources. The example below shows a dynamically provisioned volume being requested from the **gluster-cns-slow StorageClass** object. A sample persistent volume claim is provided below:

```
$ oc new-project persistent
$ oc get storageclass
NAME       TYPE
gluster-cns-slow   kubernetes.io/glusterfs

$ vi db-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db-slow
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-cns-slow
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
```

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

6.6. ADDITIONAL CNS STORAGE DEPLOYMENTS (OPTIONAL)

An OCP administrator may wish to offer multiple storage tiers to developers and users of the OpenShift Container Platform. Typically these tiers refer to certain performance characteristics, e.g. a storage tier called “fast” might be backed by **SSDs** whereas a storage tier called “slow” is backed by magnetic drives or HDDs. With **CNS** an administrator can realize this by deploying additional storage nodes running **glusterfs** pods. The additional nodes allow for the creation of additional **Storage Classes**. A developer then consumes different storage tiers by select the appropriate **StorageClass** object by the object’s name.

NOTE

Creating additional **CNS** storage deployments is not possible if using OCP 3.4. Only one **CNS** and subsequent **StorageClass** object can be created.

6.6.1. Deployment of a second Gluster Storage Pool

To deploy an additional **glusterfs** pool OCP requires additional nodes to be available that currently are not running **glusterfs** pods yet. This will require that another three OpenShift nodes are available in the environment using either the **add-node** with option **--node_type=storage** script or by manually deploying three instances and installing and configuring those nodes for OpenShift.

Once the new nodes are available, the next step is to get **glusterfs** pods up and running on the additional nodes. This is achieved by extending the members of the DaemonSet defined in the first **CNS** deployment. The **storagenode=glusterfs** label must be applied to the nodes to allow for the scheduling of the **glusterfs** pods.

First identify the three nodes that will be added to the **CNS** cluster and then apply the label.

```
$ oc get nodes --show-labels
NAME                          STATUS                          AGE
VERSION              LABELS
ocp3-app-cns-0.dpl.local      Ready                           8d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-0.dpl.local,storagenode=glusterfs
ocp3-app-cns-1.dpl.local      Ready                           8d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-1.dpl.local,storagenode=glusterfs
ocp3-app-cns-2.dpl.local      Ready                           8d
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-2.dpl.local,storagenode=glusterfs
ocp3-app-cns-3.dpl.local      Ready                           1h
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-3.dpl.local
ocp3-app-cns-4.dpl.local      Ready                           1h
v1.6.1+5115d708d7
```

```
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-4.dpl.local
ocp3-app-cns-5.dpl.local      Ready                      1h
v1.6.1+5115d708d7
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/ho
stname=ocp3-app-cns-5.dpl.local
...omitted...

$ oc label node ocp3-app-cns-3.dpl.local storagenode=glusterfs
$ oc label node ocp3-app-cns-4.dpl.local storagenode=glusterfs
$ oc label node ocp3-app-cns-5.dpl.local storagenode=glusterfs
```

Once the label has been applied then the **glusterfs** pods will scale from 3 pods to 6. The **glusterfs** pods will be running on both the newly labeled nodes and the existing nodes.

```
$ oc get pods
NAME               READY     STATUS    RESTARTS   AGE
glusterfs-2lcnb    1/1       Running   0          26m
glusterfs-356cf    1/1       Running   0          26m
glusterfs-fh4gm    1/1       Running   0          26m
glusterfs-hg4tk    1/1       Running   0          2m
glusterfs-v759z    1/1       Running   0          1m
glusterfs-x038d    1/1       Running   0          2m
heketi-1-cqjzm     1/1       Running   0          22m
```

Wait until all of the **glusterfs** pods are in READY 1/1 state before continuing. The new pods are not yet configured as a **CNS** cluster. The new **glusterfs** pods will be a new CNS cluster after the **topology.json** file is updated to define the new nodes they reside on and the **heketi-cli** is executed with this new **topology.json** file as input.

6.6.2. Modifying the Topology File

Modify the **topology.json** file of the first **CNS** cluster to include a second entry in the “clusters” list containing the additional nodes. The initial nodes have been omitted from the output below but are still required.

```
$ vi gluster-topology.json
{
  "clusters": [
    {
      "nodes": [
        {
  ... nodes from initial cns-deploy ...
        }
      ]
    },
    {
            "nodes": [
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-3"
                            ],
                            "storage": [
```

```
                                "172.0.10.214"
                            ]
                        },
                        "zone": 1
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-4"
                            ],
                            "storage": [
                                "172.0.10.215"
                            ]
                        },
                        "zone": 2
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-app-cns-5"
                            ],
                            "storage": [
                                "172.0.10.216"
                            ]
                        },
                        "zone": 3
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                }
            ]
        }
    ]
}
```

Using **`heketi-cli`** load the modified **`topology.json`** file via **`heketi`** to trigger the creation of a second cluster using the steps below. The first step is to export the values of the **`heketi`** server, user, and key. The **HEKET_CLI_KEY** value should be the same as that created for the first cluster (set using **`--admin-key`** for **`cns-deploy`**).

```
$ export HEKETI_CLI_SERVER=http://heketi-storage.apps.dpl.local
$ export HEKETI_CLI_USER=admin
$ export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

With these environment variables exported the next step is to load the newly modified **topology.json**.

```
$ heketi-cli topology load --json=gluster-topology.json
 Found node ocp3-app-cns-0.dpl.local on cluster
ca8d539dbcb480655b611693b2d7b573
  Found device /dev/sdd
 Found node ocp3-app-cns-1.dpl.local on cluster
ca8d539dbcb480655b611693b2d7b573
  Found device /dev/sdd
 Found node ocp3-app-cns-2.dpl.local on cluster
ca8d539dbcb480655b611693b2d7b573
  Found device /dev/sdd
Creating cluster ... ID: 5cc7333acb8824e4a238217b8f360940
 Creating node ocp3-app-cns-3.dpl.local ... ID:
4a0b77b6fae1ee17ec8a6d72e5e3bf64
  Adding device /dev/sdd ... OK
 Creating node ocp3-app-cns-4.dpl.local ... ID:
6c83714f41913bc686177c14f818d304
  Adding device /dev/sdd ... OK
 Creating node ocp3-app-cns-5.dpl.local ... ID:
d151866310b7328c4cfe923317a5d2b1
  Adding device /dev/sdd ... OK
```

Observe the second cluster being created and verify that there is a new **clusterid** created in the console output. Verify you now have a second **clusterid** and that the correct OCP **CNS** nodes are in the new cluster.

```
$ heketi-cli cluster list
$ heketi-cli topology info
```

6.6.3. Creating an Additional Storage Class

Create a second **StorageClass** object via a YAML file similar to the first one with the same **heketi** route and **heketi** secret but using the new **clusterid** and a unique **StorageClass** object name.

```
$ vi glusterfs-storageclass-fast.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-fast
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: http://heketi-storage.apps.dpl.local
  clusterid: 5cc7333acb8824e4a238217b8f360940
  restauthenabled: "true"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
```

Using the OpenShift client create the **StorageClass** object.

```
$ oc create -f glusterfs-storageclass-fast.yaml
```

The second **StorageClass** object will now be available to make storage requests using gluster-cns-fast when creating the **PVC**.

```
$ vi claim2.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db-fast
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-cns-fast
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
```

6.7. CONTAINER-READY STORAGE OVERVIEW

Container-Ready Storage (CRS) like **CNS**, uses Red Hat Gluster Storage to provide dynamically provisioned storage. Unlike **CNS** where OpenShift Container Platform deploys **glusterfs** and **heketi** specific pods to be used for OCP storage, **CRS** requires an Administrator to install packages and enable the storage services on virtual or physical servers. Like **CNS**, **CRS** enables the requesting and mounting of Red Hat Gluster Storage across one or many containers (access modes RWX, ROX and RWO). **CRS** allows for the Red Hat Gluster Storage to be used outside of OpenShift. **CRS** can also be used to host the OpenShift registry as can **CNS**.

6.7.1. Prerequisites for Container-Ready Storage

Deployment of Container-Ready Storage (CRS) requires at least 3 virtual machines with at least one unused block storage device or or drive on each node. The virtual machines should have at least 2 CPUs, 32GB RAM, and an unused drive or volume of 100GB or larger per node. An entitlement for **Red Hat Gluster Storage** is also required to install the **Gluster** services.

6.7.2. Deployment of CRS Infrastructure

A python script named **add-node.py** is provided in the **openshift-ansible-contrib** git repository. When **add-node.py** is used with the **--node_type=storage** option the following will be done.

1. Create three VMware virtual machines with 32 GB Mem and 2 vCPU
2. Register the new machines with Red Hat
3. Install the prerequisites for **CRS** for Gluster on each machine
4. Add a **VMDK** volume to each node as an available block device to be used for **CRS**
5. Create a topology file using virtual machine hostnames and new **VMDK** device name
6. Install heketi and heketi-cli packages on one of the **CRS** nodes
7. Copy heketi public key to all **CRS** nodes

8. Modify heketi.json file with user supplied admin and user paswords and other necessary configuration for passwordless SSH to all **CRS** nodes

9. Using heketi-cli deploy the new **CRS** cluster

10. Create heketi-secret and new **StorageClass** object for **PVC** creation

Do the following from the workstation performing the deployment of the OpenShift Reference Architecture. The **ocp-on-vmware.ini** file used to create the OpenShift deployment must be in the directory where **add-node.py** is ran from. There are two entries in the **ocp-on-vmware.ini** file that must be added or modified. They are the **vcenter_datastore** and **container_storage**. The VMware datastore entered is where all of the new OpenShift **CRS** nodes or virtual machines will be stored so verify that this datastore has at least 1TB of available storage.

NOTE

The initial deployment should be on a **vcenter_datastore** on the first host in the cluster. After successful deployment, two of the three new OCP **CNS** virtual machines should be migrated to unique VMware hypervisors in the cluster & datastores from **vcenter_datastore**.

```
$ cd /root/git/openshift-ansible-contrib/reference-architecture/vmware-ansible/
$ cat ocp-on-vmware.ini
….omitted….
$ folder/cluster/resource pool in vCenter to organize VMs
vcenter_folder=ocp3
vcenter_datastore=DPLHP380G9-10-SS200-2
vcenter_cluster=OCP3
vcenter_resource_pool=OCP3
vcenter_datacenter=vDPL
….omitted….
$ persistent container storage: none, crs, cns
container_storage=crs
```

```
$ ./add-node.py --node_type=storage
Configured inventory values:
  console_port:  8443
  deployment_type:  openshift-enterprise
  openshift_vers:  v3_6
  vcenter_host:  172.0.10.246
  vcenter_username:  administrator@vsphere.local
  vcenter_password: *******
  vcenter_template_name:  ocp-server-template-2.0.2-new
  vcenter_folder:  ocp3
  vcenter_datastore:  DPLHP380G9-10-SS200-2
  vcenter_cluster:  OCP3
  vcenter_resource_pool:  OCP3
  vcenter_datacenter:  vDPL
  public_hosted_zone:  dpl.local
  app_dns_prefix:  apps
  vm_dns:  172.0.10.241
  vm_gw:  172.0.10.2
  vm_netmask:  255.255.255.0
```

```
  vm_network:  "Private"
  rhel_subscription_user: rhn_user
  rhel_subscription_pass: *******
  rhel_subscription_server:
  rhel_subscription_pool:  Red Hat OpenShift Container Platform, Premium*
  byo_lb:  False
  lb_host:  haproxy-0
  byo_nfs:  False
  nfs_host:  nfs-0
  nfs_registry_mountpoint:  /exports
  master_nodes:  3
  infra_nodes:  3
  app_nodes:  6
  storage_nodes:  0
  vm_ipaddr_start:  172.0.10.201
  ocp_hostname_prefix:  ocp3-
  auth_type:  ldap
  ldap_user:  openshift
  ldap_user_password: *******
  ldap_fqdn:  dpl.local
  openshift_hosted_metrics_deploy:  false
  openshift_sdn:  redhat/openshift-ovs-subnet
  containerized:  false
  container_storage:  crs
  tag:  None
  node_number:  1
  ini_path:  ./ocp-on-vmware.ini
  node_type:  storage

Continue creating the inventory file with these values? [y/N]: y
Gluster topology file created using /dev/sdd: topology.json
Inventory file created: add-node.json
host_inventory:
  ocp3-crs-0:
    guestname: ocp3-crs-0
    ip4addr: 172.0.10.211
    tag: storage
  ocp3-crs-1:
    guestname: ocp3-crs-1
    ip4addr: 172.0.10.212
    tag: storage
  ocp3-crs-2:
    guestname: ocp3-crs-2
    ip4addr: 172.0.10.213
    tag: storage

Continue adding nodes with these values? [y/N]:
Admin key password for heketi?:
User key password for heketi?:
```

Both admin-key and user-key are user defined values, they do not exist before this step. The heketi admin key (password) will later be used to create a heketi-secret in OCP. Be sure to note these values as they will be needed in future operations.

NOTE

Using the script **add-node.py** with option **--node_type=storage** is optional. Nodes can be deployed without using this script as long as the 3 new virtual machines have 2 CPUs, 32GB RAM, and an unused storage device (**VMDK** or **RDM** disk).

6.7.3. CRS Subscription Prerequisites

CRS requires the instances to use the **Red Hat Gluster Storage** entitlement which which allows access to the **rh-gluster-3-for-rhel-7-server-rpms** repository containing the required **RPMs** for a successful installation.

Ensure the pool that is specified matches a pool available to the **RHSM** credentials provided (example pool ID shown below).

NOTE

If the **add-node.py** with option **--node_type=storage** was used all of the subscription-manager commands below will be completed.

```
# subscription-manager register
# subscription-manager attach --pool=8a85f98156981319015699f0183a253c
# subscription-manager repos --disable=''*
# subscription-manager repos --enable=rhel-7-server-rpms
# subscription-manager repos --enable=rh-gluster-3-for-rhel-7-server-rpms
```

6.7.4. Firewall and Security Group Prerequisites

The **add-node.py** with option **--node_type=storage** uses iptables and creates the rules needed for **Gluster** and **Heketi** on each of the **CRS** nodes. The following commands can be ran on the 3 new virtual machines if the instances were built without using the script.

```
# yum -y install firewalld
# systemctl enable firewalld
# systemctl disable iptables
# systemctl stop iptables
# systemctl start firewalld
# firewall-cmd --add-port=24007/tcp --add-port=24008/tcp --add-
port=2222/tcp \ --add-port=8080/tcp --add-port=49152-49251/tcp --permanent
# firewall-cmd --reload
# firewall-cmd --list-all
```

6.7.5. CRS Package Prerequisites

The **redhat-storage-server** package and dependencies will install all of the required **RPMs** for a successful Red Hat Gluster Storage installation. Perform the following on the each of the three **CRS** nodes.

NOTE

If the **add-node.py** with option **--node_type=storage** was used the **redhat-storage-server** package will be installed and glusterfs will be enabled and started.

```
$ yum install -y redhat-storage-server
```

After successful installation enable and start the **glusterd.service**.

```
$ systemctl enable glusterd
$ systemctl start glusterd
```

6.7.6. Installing and Configuring Heketi

Heketi is used to manage the Gluster **Trusted Storage Pool(TSP)**. **Heketi** is used to perform tasks such as adding volumes, removing volumes, and creating the initial **TSP**. If the **add-node.py** with option **--node_type=storage** script was not used perform the following on the each of the **CRS** nodes. **Heketi** can be installed on one of the **CRS** instances. For the steps below the first **CRS Gluster** virtual machine will be used.

> **NOTE**
>
> If the **add-node.py** with option **--node_type=storage** was used all of the following steps up to the next section **Loading Topology File** will be automatically completed.

```
$ yum install -y heketi heketi-client
```

Create the **heketi** private key on the instance designated to run **heketi**.

```
$ ssh-keygen -f /etc/heketi/heketi_key -t rsa -N ''
$ chown heketi:heketi /etc/heketi/heketi_key.pub
$ chown heketi:heketi /etc/heketi/heketi_key
```

Copy the contents of the **/etc/heketi/heketi_key.pub** into a clipboard and login to each **CRS** node and paste the contents of the clipboard as a new line into the **/root/.ssh/authorized_keys** file. Also make sure on each CRS node to modify the /etc/ssh/sshd_config file to allow root passwordless ssh access (enable “PermitRootLogin” and “RSAAuthentication”) and restart sshd.service. This must be done on all 3 instances including the **CRS** node where the **heketi** services are running. Also, on each of the 3 virtual machines **requiretty** must be disabled or removed in **/etc/sudoers** to allow for management of those hosts using **sudo**. Ensure that the line below either does not exist in sudoers or that it is commented out.

```
$ visudo
... omitted ...
#Defaults    requiretty
... omitted ...
```

On the node where **Heketi** was installed, edit the **/etc/heketi/heketi.json** file to setup the SSH executor and the admin and user keys. The **heketi** admin key (password) will be used to create a **heketi-secret** in OCP. This secret will then be used during the creation of the **StorageClass** object.

```
$ vi /etc/heketi/heketi.json
... omitted ...
"_use_auth": "Enable JWT authorization. Please enable for deployment",
```

```
"use_auth": true,

"_jwt": "Private keys for access",
"jwt": {
  "_admin": "Admin has access to all APIs",
  "admin": {
    "key": "myS3cr3tpassw0rd"
  },
  "_user": "User only has access to /volumes endpoint",
  "user": {
    "key": "mys3rs3cr3tpassw0rd"
  }
},

 "glusterfs": {
    "_executor_comment": [
      "Execute plugin. Possible choices: mock, ssh",
      "mock: This setting is used for testing and development.",
      "      It will not send commands to any node.",
      "ssh:  This setting will notify Heketi to ssh to the nodes.",
      "      It will need the values in sshexec to be configured.",
      "kubernetes: Communicate with GlusterFS containers over",
      "            Kubernetes exec api."
    ],
    "executor": "ssh",
    "_sshexec_comment": "SSH username and private key file information",
    "sshexec": {
      "keyfile": "/etc/heketi/heketi_key",
      "user": "root",
      "port": "22",
      "fstab": "/etc/fstab"
    },
... omitted ...
```

Restart and enable **heketi** service to use the configured **/etc/heketi/heketi.json** file.

```
$ systemctl restart heketi
$ systemctl enable heketi
```

The **heketi** service should now be running. **Heketi** provides an endpoint to perform a health check. This validation can be done from either an **OCP** master or from any of the **CRS** nodes. The hostname is the **CRS** node where **Heketi** is installed, in this case ocp3-crs-0.dpl.local.

```
$ curl http://ocp3-crs-0.dpl.local:8080/hello
Hello from Heketi
```

6.7.7. Loading Topology File

The **topology.json** is used to tell **heketi** about the environment and which nodes and storage devices it will manage. There is a sample file located in **/usr/share/heketi/topology-sample.json** and an example shown below for 3 CRS nodes on 3 different VMware hypervisors. Both **CRS** and **CNS** use the same format for the **topology.json** file.

NOTE

If the **`add-node.py`** with option **`--node_type=storage`** was used the **`topology.json`** file will be in the directory where the script was launched from as well as on one of the **CRS** nodes where **`Heketi`** service was installed.

```
$ vi topology.json
{
    "clusters": [
        {
            "nodes": [
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-crs-0.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.211"
                            ]
                        },
                        "zone": 1
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-crs-1.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.212"
                            ]
                        },
                        "zone": 2
                    },
                    "devices": [
                        "/dev/sdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ocp3-crs-2.dpl.local"
                            ],
                            "storage": [
                                "172.0.10.213"
                            ]
                        },
                        "zone": 3
                    },
```

```
                        "devices": [
                            "/dev/sdd"
                        ]
                    }
                ]
            }
        ]
    }
```

The **HEKETI_CLI_SERVER**, **HEKETI_CLI_USER**, and **HEKETI_CLI_KEY** environment variables are required for **heketi-cli** commands to be ran. The **HEKETI_CLI_SERVER** is the **CRS** node name where the **heketi** services are running. The **HEKETI_CLI_KEY** is the admin key value configured in the **/etc/heketi/heketi.json** file.

```
$ export HEKETI_CLI_SERVER=http://ocp3-crs-0.dpl.local:8080
$ export HEKETI_CLI_USER=admin
$ export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

Using **heketi-cli**, run the following command to load the topology of your environment.

```
$ heketi-cli topology load --json=topology.json
      Creating cluster ... ID: bb802020a9c2c5df45f42075412c8c05
 Creating node ocp3-crs-0.dpl.local ... ID:
b45d38a349218b8a0bab7123e004264b
    Adding device /dev/sdd ... OK
 Creating node ocp3-crs-1.dpl.local ... ID:
2b3b30efdbc3855a115d7eb8fdc800fe
    Adding device /dev/sdd ... OK
 Creating node ocp3-crs-2.dpl.local ... ID:
c7d366ae7bd61f4b69de7143873e8999
    Adding device /dev/sdd ... OK
```

6.8. VALIDATING GLUSTER INSTALLATION(OPTIONAL)

From the **CRS** node where **heketi** client is installed and the **heketi** environment variables has been exported create a Gluster volume to verify **heketi**.

```
$ heketi-cli volume create --size=50
Name: vol_c81c139ef1f907a95a0b136e91eab44a
Size: 50
Volume Id: c81c139ef1f907a95a0b136e91eab44a
Cluster Id: bb802020a9c2c5df45f42075412c8c05
Mount: 172.0.10.211:vol_c81c139ef1f907a95a0b136e91eab44a
Mount Options: backup-volfile-servers=172.0.10.212,172.0.10.213
Durability Type: replicate
Distributed+Replica: 3
```

The command **gluster volume info** can provide further information on the newly created **Gluster** volume. Issue this command for one of the **CRS** nodes.

```
$ gluster volume info

Volume Name: vol_c81c139ef1f907a95a0b136e91eab44a
```

```
Type: Replicate
Volume ID: 8d9f94a4-9d64-4fa5-b376-2f5a7861ca39
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1:
172.0.10.211:/var/lib/heketi/mounts/vg_9a56433faa906302dbfa37776e321f30/br
ick_304bc8a12bc7356500d5752b2ee3eebf/brick
Brick2:
172.0.10.212:/var/lib/heketi/mounts/vg_b991283f0fc801e21ab154373e03027d/br
ick_b73fd447db6d00b3caca237ced4ebfbe/brick
Brick3:
172.0.10.213:/var/lib/heketi/mounts/vg_d15c1d4861eac16697eb311fa51b0dde/br
ick_1386645297de95f526433e254fa0349f/brick
Options Reconfigured:
transport.address-family: inet
performance.readdir-ahead: on
nfs.disable: on
```

6.9. DEPLOYING CRS FOR OPENSHIFT CONTAINER PLATFORM (OCP)

6.9.1. Store the heketi secret

OCP allows for the use of secrets so that items do not need to be stored in clear text. The admin password for **heketi**, specified during configuration of the **heketi.json** file, should be stored in base64-encoding. OCP can refer to this secret instead of specifying the password in clear text.

To generate the base64-encoded equivalent of the admin password supplied to the cns-deploy command perform the following.

```
$ echo -n myS3cr3tpassw0rd | base64
bXlTM2NyM3RwYXNzdzByZA==
```

On the master or workstation with the OCP client installed with **cluster-admin** privileges use the base64 password string in the following YAML to define the secret in OCP's default namespace.

NOTE

If the **add-node.py** with option **--node_type=storage** was used the **heketi-secret.yaml** file will be in the directory where the script was launched from. The operation to create the heketi-secret in the default project will also be completed.

```
$ vi heketi-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: heketi-secret
  namespace: default
data:
  key: bXlTM2NyM3RwYXNzdzByZA==
type: kubernetes.io/glusterfs
```

Create the secret by using the following command.

```
$ oc create -f heketi-secret.yaml
secret "heketi-secret" created
```

6.9.2. Creating a Storage Class

CRS storage has all of the same benefits that **CNS** storage has with regard to OpenShift storage. The **cluster-admin** or **storage-admin** can perform the following which will allow for dynamically provisioned **CRS** storage on demand. The key benefit of this storage is that the persistent storage can be created with access modes **ReadWriteOnce(RWO)**, **ReadOnlyMany(ROX)**, or **ReadWriteMany(RWX)**.

A **StorageClass** object requires certain parameters to be defined to successfully create the resource. Use the values of the exported environment variables from the previous steps to define the **resturl**, **restuser**, **secretNamespace**, and **secretName**.

NOTE

If the **add-node.py** with option **--node_type=storage** was used the **storageclass.yaml** file will be in the directory where the script was launched in. The operation to create the **StorageClass** crs-gluster will also be completed.

```
$ vi storageclass.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: crs-gluster
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://ocp3-crs-0.dpl.local:8080"
  restauthenabled: "true"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
```

Once the **Storage Class** json file has been created use the **oc create** command to create the object in OpenShift.

```
$ oc create -f storageclass.yaml
```

To validate the **Storage Class** was created perform the following.

```
$ oc get storageclass
NAME             TYPE
crs-gluster kubernetes.io/glusterfs

$ oc describe storageclass crs-gluster
Name:  crs-gluster
IsDefaultClass: No
Annotations: <none>
Provisioner: kubernetes.io/glusterfs
Parameters: restauthenabled=true,resturl=http://ocp3-crs-
```

```
0.dpl.local:8080,restuser=admin,secretName=heketi-
secret,secretNamespace=default
No events.
```

6.9.3. Creating a Persistent Volume Claim

The **`Storage Class`** created in the previous section allows for storage to be dynamically provisioned using the **CRS** resources. The example below shows a dynamically provisioned volume being requested from the **`crs-gluster StorageClass`** object. A sample persistent volume claim is provided below:

```
$ vi db-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 annotations:
   volume.beta.kubernetes.io/storage-class: crs-gluster
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi

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

6.10. RWO PERSISTENT STORAGE EXAMPLE (OPTIONAL)

For **`ReadWriteOnce`** storage, any of the **`StorageClass`** objects created in the above sections can be used. The persistent volume claim will be done at the time of application deployment and provisioned based on the rules in the **`StorageClass`** object. The example below uses a **`MySQL`** deployment using an OCP standard template and one of the **`StorageClass`** objects defined above.

Create an OCP project for **`MySQL`** deployment.

```
$ oc new-project rwo
```

The ‘mysql-persistent’ template will be used for deploying **`MySQL`**. The first step is to check to see if the template is available for use.

```
$ oc get templates -n openshift | grep "MySQL database service, with
persistent storage"
mysql-persistent  MySQL database service, with persistent storage.
```

Export the default mysql-persistent template content into a yaml file. The OpenShift client can provide a view of the available parameters for this template.

```
$ oc export template/mysql-persistent -n openshift -o yaml > mysql-
persistent.yaml
$ oc process -f mysql-persistent.yaml --parameters
```

View the contents of the yaml file and add the lines below to identify the **StorageClass** object the **MySQL PVC** will be created from. If these lines are not added the default StorageClass object will be used.

> **NOTE**
>
> Any of the **StorageClass** objects created in this reference architecture can be used.

```
$ vi mysql-persistent.yaml
…. omitted ….
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: ${DATABASE_SERVICE_NAME}
    annotations:
      volume.beta.kubernetes.io/storage-class: gluster-cns-fast
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: ${VOLUME_CAPACITY}
…. omitted ….
```

Create a deployment manifest from the **mysql-persistent.yaml** template file and view contents. Make sure to modify the ‘storage: ${VOLUME_CAPACITY}’ to be the desired size for the database (1Gi is default value).

```
$ oc process -f mysql-persistent.yaml -o yaml > cns-mysql-persistent.yaml
$ vi cns-mysql-persistent.yaml
…. omitted ….
- apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    annotations:
      volume.beta.kubernetes.io/storage-class: gluster-cns-fast
    labels:
      template: mysql-persistent-template
    name: mysql
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
…. omitted ….
```

Using the deployment manifest, create the the objects for the **MySQL** application.

```
$ oc create -f cns-mysql-persistent.yaml
secret "mysql" created
service "mysql" created
persistentvolumeclaim "mysql" created
deploymentconfig "mysql" created
```

Validate application is using a persistent volume claim.

```
$ oc describe dc mysql
…. omitted ….
  Volumes:
   mysql-data:
    Type: PersistentVolumeClaim (a reference to a PersistentVolumeClaim in
the same namespace)
    ClaimName: mysql
    ReadOnly: false
…. omitted ….
```

```
$ oc get pvc mysql
NAME      STATUS    VOLUME                                     CAPACITY
ACCESSMODES
mysql     Bound     pvc-fc297b76-1976-11e7-88db-067ee6f6ca67   1Gi
RWO
```

Validate that the **MySQL** pod has a PV mounted at **/var/lib/mysql/data** directory.

```
$ oc volumes dc mysql
deploymentconfigs/mysql
pvc/mysql (allocated 1GiB) as mysql-data
mounted at /var/lib/mysql/data
```

The option also exists to connect to the running pod to view the storage that is currently in use.

```
$ oc rsh mysql-1-4tb9g
sh-4.2$ df -h /var/lib/mysql/data
Filesystem                                       Size  Used Avail Use%
Mounted on
10.20.4.40:vol_e9b42baeaaab2b20d816b65cc3095558 1019M  223M  797M  22%
/var/lib/mysql/data
```

6.11. RWX PERSISTENT STORAGE (OPTIONAL)

One of the benefits of using Red Hat Gluster Storage is the ability to use access mode ReadWriteMany(RWX) for container storage. This example is for a **PHP** application which has requirements for a persistent volume mount point. The application will be scaled to show the benefits of **RWX** persistent storage.

Create a test project for the demo application.

```
$ oc new-project rwx
```

Create the application using the following github link:

```
$ oc new-app openshift/php:7.0~https://github.com/christianh814/openshift-
php-upload-demo --name=demo
--> Found image d3b9896 (2 weeks old) in image stream "openshift/php"
under tag "7.0" for "openshift/php:7.0"

    Apache 2.4 with PHP 7.0
```

```
    -----------------------
    Platform for building and running PHP 7.0 applications

    Tags: builder, php, php70, rh-php70

    * A source build using source code from
https://github.com/christianh814/openshift-php-upload-demo will be created
      * The resulting image will be pushed to image stream "demo:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "demo"
    * Port 8080/tcp will be load balanced by service "demo"
      * Other containers can access this service through the hostname
"demo"

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

Validate that the build is complete and the pods are running.

```
$ oc get pods
NAME             READY     STATUS      RESTARTS   AGE
demo-1-build     0/1       Completed   0          20s
demo-1-sch77     1/1       Running     0          7s
```

The next step is to retrieve the name of the OpenShift **svc** which will be used to create a route.

```
$ oc get svc
NAME      CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
demo      172.30.211.203   <none>        8080/TCP   1m
```

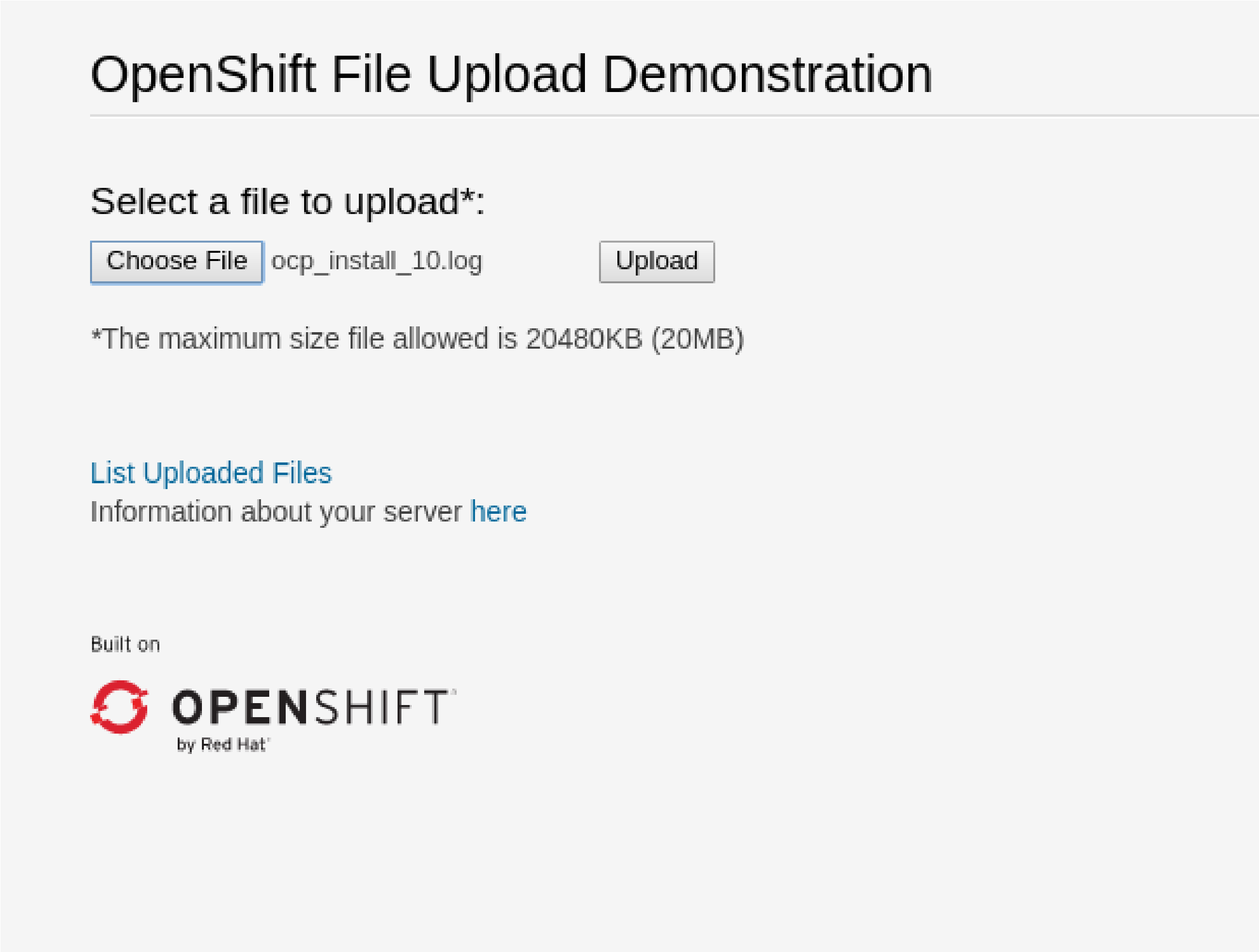
Expose the service as a public route by using the **oc expose** command.

```
$ oc expose svc/demo
route "demo" exposed
```

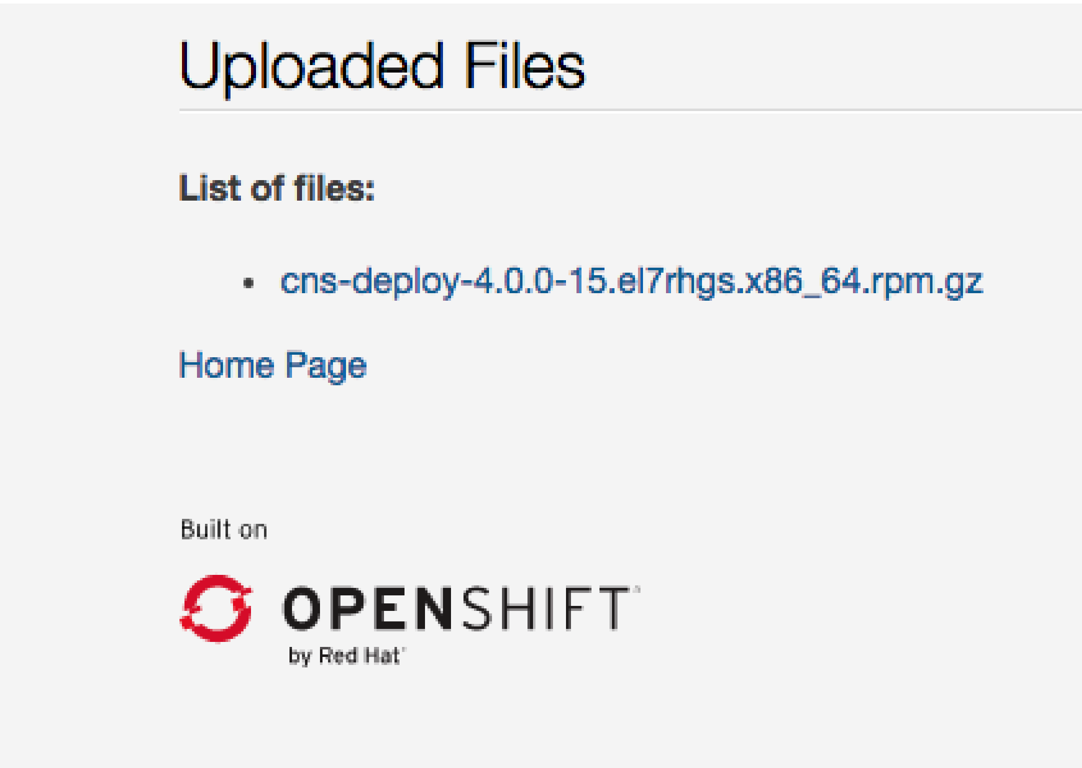
OpenShift will create a route based on the application name, project, and wildcard zone. This will be the **URL** that can be accessed by browser.

```
$ oc get route
NAME      HOST/PORT                  PATH      SERVICES   PORT
TERMINATION   WILDCARD
demo      demo-rwx.apps.dpl.local             demo       8080-tcp
None
```

Using a web browser validate the application using the route defined in the previous step.



Upload a file using the web UI.



Connect to the **`demo-1-sch77`** and verify the file exists.

```
$ oc get pods
```

```
NAME            READY     STATUS      RESTARTS   AGE
demo-1-sch77    1/1       Running     0          5m
$ oc rsh demo-1-sch77
sh-4.2$ cd uploaded
sh-4.2$ pwd
/opt/app-root/src/uploaded
sh-4.2$ ls -lh
-rw-r--r--. 1 1000080000 root 16K Apr 26 21:32 cns-deploy-4.0.0-
15.el7rhgs.x86_64.rpm.gz
```

Scale up the number of demo-1 pods from 1 to 2.

```
$ oc scale dc/demo --replicas=2
```

```
$ oc get pods
NAME            READY     STATUS              RESTARTS   AGE
demo-1-build    0/1       Completed           0          7m
demo-1-sch77    1/1       Running             0          7m
demo-1-sdz28    0/1       Running             0          3s
```

Login to the newly created pod and view the **uploaded** directory.

```
$ oc rsh demo-1-sdz28
sh-4.2$ cd uploaded
sh-4.2$ pwd
/opt/app-root/src/uploaded
sh-4.2$ ls -lh
total 0
```

The uploaded file is not available to this newly created second pod because the storage is local to the pod, demo-1-sch77. In the next steps, the storage for the pods will be changed from local or ephemeral storage to a **RWX** persistent volume claim for the mount point **/opt/app-root/src/uploaded**.

First, add a persistent volume claim to the project. The existing OCP **StorageClass** object created for a CNS cluster (gluster-cns-slow) will be used to create a **PVC** with the access mode of **RWX**.

NOTE

A **CRS StoreClass** object can be used in the steps below as well.

The first step is to create the **app-claim.yaml** file.

```
$ vi app-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: app
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-cns-slow
spec:
 accessModes:
  - ReadWriteMany
```

```
  resources:
    requests:
      storage: 10Gi
```

Using the **app-claim.yaml** file use the OCP client to create the **PVC**.

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

Verify the **PVC** was created.

```
$ oc get pvc app
NAME      STATUS    VOLUME                                     CAPACITY
ACCESSMODES   AGE
app       Bound     pvc-418330b7-2ac9-11e7-946e-067f85bdafe9   10Gi
RWX           46s
```

Now that the **PVC** exists tie the claim to the deployment configuration using the existing mount path **/opt/app-root/src/uploaded** for **demo** pods.

```
$ oc volume dc/demo --add --name=persistent-volume --
type=persistentVolumeClaim --claim-name=app --mount-path=/opt/app-
root/src/uploaded
```

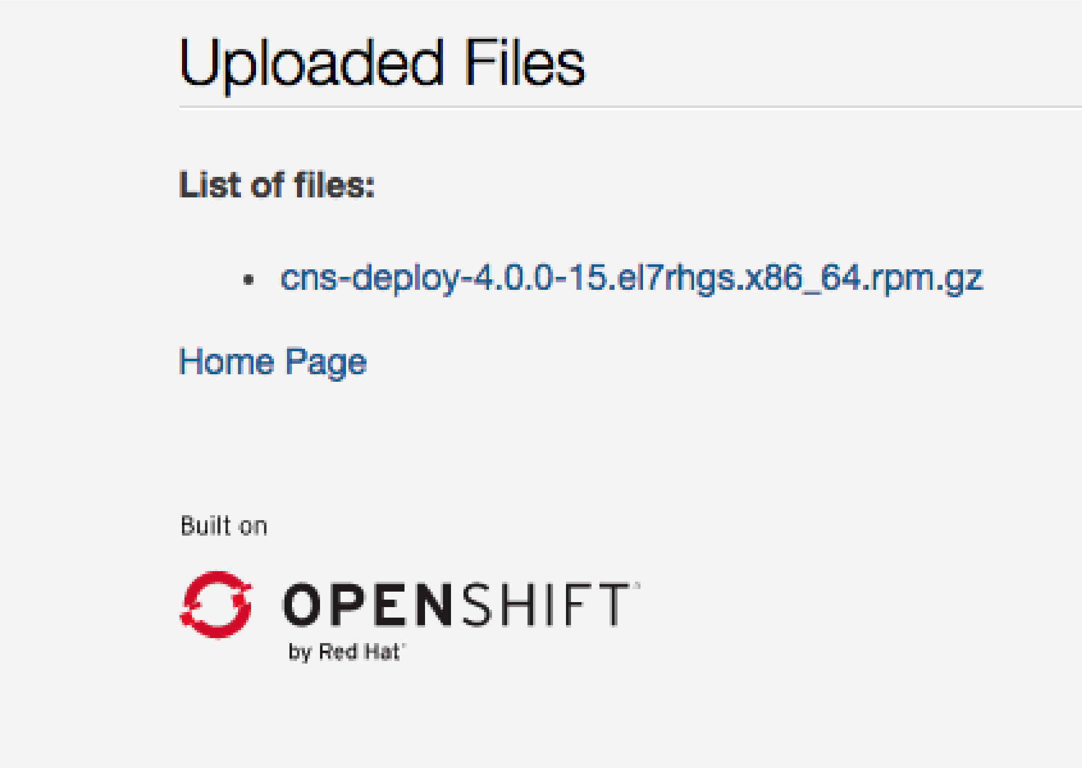
A new deployment is created using the **PVC** and there are two new **demo** pods

```
$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          16m
demo-2-9cv88   1/1       Running     0          8s
demo-2-m1mwt   1/1       Running     0          13s
```

Now there is a persistent volume allocated using the gluster-cns-slow storage class and mounted at **/opt/app-root/src/uploaded** on the **demo-2** pods.

```
$ oc volumes dc demo
deploymentconfigs/demo
  pvc/app (allocated 10GiB) as persistent-volume
    mounted at /opt/app-root/src/uploaded
```

Using the route for the **demo-2** deployment upload a new file.



Now login to both pods and validate that both pods can read the newly uploaded file.

On the first pod perform the following.

```
$ oc rsh demo-2-9cv88
sh-4.2$ df -h
Filesystem
Size  Used Avail Use% Mounted on
….omitted….
172.0.10.231:vol_a3c4d6122b7ef970b5c301fac1f18621
10G   34M   10G   1% /opt/app-root/src/uploaded
sh-4.2$ cd /opt/app-root/src/uploaded
sh-4.2$ ls -lh
total 812K
-rw-r--r--. 1 1000070000 root 809K Jul  5 21:16 putty.exe
```

On the second pod perform the following.

```
$ oc rsh demo-2-m1mwt
Filesystem
Size  Used Avail Use% Mounted on
….omitted….
172.0.10.231:vol_a3c4d6122b7ef970b5c301fac1f18621
10G   34M   10G   1% /opt/app-root/src/uploaded

sh-4.2$ cd /opt/app-root/src/uploaded
sh-4.2$ ls -lh
total 812K
-rw-r--r--. 1 1000070000 root 809K Jul  5 21:16 putty.exe
```

Scale up the number of demo-2 pods from two to three.

```
$ oc scale dc/demo --replicas=3
```

■

Verify the third pod has a STATUS of Running.

```
$ oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          43m
demo-2-9cv88   1/1       Running     0          26m
demo-2-kcc16   1/1       Running     0          5s
demo-2-m1mwt   1/1       Running     0          27m
```

Login to the third pod and validate the uploaded file exists.

```
$ oc rsh demo-2-kcc16
sh-4.2$ cd uploaded
sh-4.2$ ls -lh
total 809K
-rw-r--r--. 1 1000070000 2000 809K Jul  5 22:38 putty.exe
```

CHAPTER 7. EXTENDING THE CLUSTER

By default, the reference architecture playbooks are configured to deploy 3 master, 3 application, and 3 infrastructure nodes. As the cluster begins to be utilized by more teams and projects, it will be become necessary to provision more application or infrastructure nodes to support the expanding environment. To facilitate easily growing the cluster, the **add-node.py** python script (similar to **ocp-on-vmware.py**) is provided in the **openshift-ansible-contrib** repository. It will allow for provisioning either an Application or Infrastructure node per run and can be ran as many times as needed.

7.1. BEFORE ADDING A NODE

Verify the quantity and type of the nodes in the cluster by using the **oc get nodes** command. The output below is an example of a complete OpenShift environment after the deployment of the reference architecture environment.

```
$ oc get nodes
NAME                   STATUS                                         AGE
VERSION
master-0.example.com   Ready,SchedulingDisabled                       14m
v1.6.1+5115d708d7
master-1.example.com   Ready,SchedulingDisabled                       14m
v1.6.1+5115d708d7
master-2.example.com   Ready,SchedulingDisabled                       14m
v1.6.1+5115d708d7
infra-0.example.com    Ready                                          14m
v1.6.1+5115d708d7
infra-1.example.com    Ready                                          14m
v1.6.1+5115d708d7
infra-2.example.com    Ready                                          14m
v1.6.1+5115d708d7
app-0.example.com      Ready                                          14m
v1.6.1+5115d708d7
app-1.example.com      Ready                                          14m
v1.6.1+5115d708d7
app-2.example.com      Ready                                          14m
v1.6.1+5115d708d7
```

7.2. INTRODUCTION TO ADD-NODE.PY

The python script **add-node.py** is operationally similar to the **ocp-on-vmware.py** script. Parameters can optionally be passed in when calling the script and values are read from **ocp-on-vmware.ini**. Any required parameters not already set will automatically prompted for at run time. To see all allowed parameters, the --help trigger is available.

```
$ ./add-node.py --help
usage: add-node.py [-h] [--node_type NODE_TYPE] [--node_number
NODE_NUMBER]
                   [--create_inventory] [--no_confirm NO_CONFIRM] [--tag
TAG]
                   [--verbose]

Add new nodes to an existing OCP deployment
```

```
optional arguments:
  -h, --help            show this help message and exit
  --node_type NODE_TYPE
                        Specify the node label: app, infra, storage
  --node_number NODE_NUMBER
                        Specify the number of nodes to add
  --create_inventory    Helper script to create json inventory file and
exit
  --no_confirm NO_CONFIRM
                        Skip confirmation prompt
  --tag TAG             Skip to various parts of install valid tags
include:
                          - vms (create vms for adding nodes to cluster
or CNS/CRS)
                          - node-setup (install the proper packages on
the CNS/CRS nodes)
                          - heketi-setup (install heketi and config on
the crs master/CRS ONLY)
                          - heketi-ocp (install the heketi secret and
storage class on OCP/CRS ONLY)
                          - clean (remove vms and unregister them from
RHN also remove storage classes or secrets
  --verbose             Verbosely display commands
```

7.3. ADDING AN APPLICATION NODE

To add an application node, run the **add-node.py** script following the example below. Once the instance is launched, the installation of OpenShift will automatically begin.

NOTE

The **storage** node_type is available to add persistent storage to the OCP cluster using container native storage **CNS** or container ready storage **CRS**. Please see the chapters involving persistent storage for more information about this options.

```
$ ./add-node.py --node_type=app
Configured inventory values:
     cluster_id:  3e06olsus6x2rbgcxw4t
  console_port:  8443
  deployment_type:  openshift-enterprise
  openshift_vers:  v3_6
...omitted...
  node_number:  1
  ini_path:  ./ocp-on-vmware.ini
  node_type:  app


Continue creating the inventory file with these values? [y/N]: y

Inventory file created: add-node.json
host_inventory:
  app-1:
    guestname: app-1
```

```
    ip4addr: 10.x.x.230
    tag: app

Continue adding nodes with these values? [y/N]:
```

7.4. ADDING AN INFRASTRUCTURE NODE

The process for adding an Infrastructure Node is nearly identical to adding an Application Node. The only differences in adding an Infrastructure node is the requirement updating the HAproxy load balancer entry used by the router. Follow the example steps below to add a new infrastructure node.

```
$ *./add-node.py --node_type=infra

Configured inventory values:
  cluster_id: 3e06olsus6x2rbgcxw4t
     console_port:  8443
  deployment_type:  openshift-enterprise
  openshift_vers:  v3_6
...omitted...
node_number:  1
ini_path:  ./ocp-on-vmware.ini
node_type:  infra

Continue creating the inventory file with these values? [y/N]: y

   Inventory file created: add-node.json
   host_inventory:
     infra-1:
       guestname: infra-1
       ip4addr: 10.x.x.230
       tag: infra

   Continue adding nodes with these values? [y/N]:
```

7.5. VALIDATING A NEWLY PROVISIONED NODE

To verify a newly provisioned node that has been added to the existing environment, use the **oc get nodes** command. In this example, node **app-3.example.com** is an application node newly deployed by the **add-node.py** playbooks..

```
$ oc get nodes
NAME                                      STATUS                      AGE
master-0.example.com   Ready,SchedulingDisabled                  14m
v1.6.1+5115d708d7
master-1.example.com   Ready,SchedulingDisabled                  14m
v1.6.1+5115d708d7
master-2.example.com   Ready,SchedulingDisabled                  14m
v1.6.1+5115d708d7
infra-0.example.com    Ready                                     14m
v1.6.1+5115d708d7
infra-1.example.com    Ready                                     14m
v1.6.1+5115d708d7
infra-2.example.com    Ready                                     14m
v1.6.1+5115d708d7
```

```
app-0.example.com      Ready                                        14m
v1.6.1+5115d708d7
app-1.example.com      Ready                                        14m
v1.6.1+5115d708d7
app-2.example.com      Ready                                        14m
v1.6.1+5115d708d7
app-3.example.com      Ready                                        2m
v1.6.1+5115d708d7

$ oc get nodes --show-labels | grep app | wc -l
4
```

CHAPTER 8. CONCLUSION

Red Hat solutions involving Red Hat 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 Red Hat OpenShift Container Platform infrastructure in VMware
- OpenShift masters spread across multiple VMware cluster nodes utilizing anti-affinity pinning rules
- Infrastructure nodes spread across multiple VMware cluster nodes with router and registry pods scaled accordingly
- Native integration with existing services and the capacity to create those services if need be
    - HAproxy load balancer for the master instances and for the infrastructure instances
    - NFS storage for persistent storage of container images
- Native integration with VMware services like VMDK disks, HA, SIOC
    - VMDK thin provisioned storage for persistent /var/lib/docker on each node
    - VMware Storage IO Control SIOC to address latency workloads on nodes
    - VMware native HA for high availability on nodes
- Creation of applications
- Validating the environment
- Testing failover

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

APPENDIX B. CONTRIBUTORS

Christoph Goern, content provider

Chandler Wilkerson, content reviewer

Eduardo Minguez, content reviewer

Ryan Cook, content reviewer

Ken Bell, content reviewer

Erik Jacobs, content reviewer

Scott Collier, content reviewer

Annette Clewett, content provider

Red Hat would also like to thank Western Digital for use of their Data Propulsion Lab during the creation of this reference architecture. Please visit wdc.com (link) for more information on Western Digital enterprise HDD products, and sandisk.com/business (link) for more information on Western Digital enterprise SSD products.

APPENDIX C. QUICK STEPS: HOW TO INSTALL RED HAT OPENSHIFT CONTAINER PLATFORM

- Make sure the template and SSH keys are copied over into the appropriate places.
- Clone the git repo and run the script in setup.

```
$ cd ~ && git clone -b vmw-3.6 https://github.com/openshift/openshift-
ansible-contrib
$ sh ~/openshift-ansible-contrib/reference-architecture/vmware-
ansible/scripts/setup_ansible.sh
```

- Fill out the variables in the ocp-on-vmware.ini file.

```
$ vim ~/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ocp-on-vmware.ini
```

- Run ocp-on-vmware.py with --create_ocp_vars.

```
$ cd ~/openshift-ansible-contrib/reference-architecture/vmware-ansible/ &&
./ocp-on-vmware.py --create_ocp_vars
```

- Run ocp-on-vmware.py with --create_inventory.

```
$ cd ~/openshift-ansible-contrib/reference-architecture/vmware-ansible/ &&
./ocp-on-vmware.py --create_inventory
```

- Run ocp-on-vmware.py by itself.

```
$ cd ~/openshift-ansible-contrib/reference-architecture/vmware-ansible/ &&
./ocp-on-vmware.py
```

- Test the install by running ocp-on-vmware.py --tag ocp-demo

```
$ cd ~/openshift-ansible-contrib/reference-architecture/vmware-ansible/ &&
./ocp-on-vmware.py --tag ocp-demo
```

If installation fails during the ./ocp-on-vmware.py run by itself, it can be re-run safely.

APPENDIX D. TROUBLESHOOTING ANSIBLE BY RED HAT

In the event of a deployment failure, there are a couple of options to use to troubleshoot Ansible.

- ocp-on-vmware.py -vvvvvv : the very verbose option gives obfuscated additional information about the Ansible run.
    - This can be helpful in determining connection issues or run book play errors.

```
TASK [rhn-subscription : Is the host already registered?]
**********************
task path: /opt/ansible/roles/rhn-subscription/tasks/main.yaml:16
Using module file /usr/lib/python2.7/site-
packages/ansible/modules/core/commands/command.py
<10.19.114.224> ESTABLISH SSH CONNECTION FOR USER: root
<10.19.114.224> SSH: ansible.cfg set ssh_args: (-C)(-o)
(ControlMaster=auto)(-o)(ControlPersist=900s)(-o)(GSSAPIAuthentication=no)
(-o)(PreferredAuthentications=publickey)
<10.19.114.224> SSH: ANSIBLE_HOST_KEY_CHECKING/host_key_checking disabled:
(-o)(StrictHostKeyChecking=no)
<10.19.114.224> SSH:
ANSIBLE_PRIVATE_KEY_FILE/private_key_file/ansible_ssh_private_key_file
set: (-o)(IdentityFile="ssh_key/ocp3-installer")
<10.19.114.224> SSH: ansible_password/ansible_ssh_pass not set: (-o)
(KbdInteractiveAuthentication=no)(-o)(PreferredAuthentications=gssapi-
with-mic,gssapi-keyex,hostbased,publickey)(-o)(PasswordAuthentication=no)
<10.19.114.224> SSH: ANSIBLE_REMOTE_USER/remote_user/ansible_user/user/-u
set: (-o)(User=root)
<10.19.114.224> SSH: ANSIBLE_TIMEOUT/timeout set: (-o)(ConnectTimeout=10)
<10.19.114.224> SSH: PlayContext set ssh_common_args: ()
<10.19.114.224> SSH: PlayContext set ssh_extra_args: ()
<10.19.114.224> SSH: found only ControlPersist; added ControlPath: (-o)
(ControlPath=/var/run/%h-%r)
<10.19.114.224> SSH: EXEC ssh -vvv -C -o ControlMaster=auto -o
ControlPersist=900s -o GSSAPIAuthentication=no -o
PreferredAuthentications=publickey -o StrictHostKeyChecking=no -o
'IdentityFile="ssh_key/ocp3-installer"' -o KbdInteractiveAuthentication=no
-o PreferredAuthentications=gssapi-with-mic,gssapi-
keyex,hostbased,publickey -o PasswordAuthentication=no -o User=root -o
ConnectTimeout=10 -o ControlPath=/var/run/%h-%r 10.19.114.224 '/bin/sh -c
'"'"'/usr/bin/python && sleep 0'"'"''
```

- If there is a failure during the playbook, occasionally the playbooks may be rerun.

APPENDIX E. INSTALLATION FAILURE

In the event of an OpenShift installation failure, create an inventory file and run the uninstall playbook.

E.1. INVENTORY

The manual inventory is used with the uninstall playbook to identify OpenShift nodes.

```
vi ~/inventory
[OSEv3:children]
masters
etcd
nodes

[OSEv3:vars]
openshift_master_cluster_hostname="internal-openshift-master.{{
public_hosted_zone }}"
openshift_master_cluster_public_hostname="openshift-master.{{
public_hosted_zone }}"
osm_default_subdomain="{{ wildcard_zone }}"
deployment_type=openshift-enterprise
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level | default(debug_level,
true) }}"
openshift_master_debug_level="{{ master_debug_level | default(debug_level,
true) }}"
openshift_master_access_token_max_seconds=2419200
openshift_master_api_port="{{ console_port }}"
openshift_master_console_port="{{ console_port }}"
osm_cluster_network_cidr=172.16.0.0/16
osm_use_cockpit=false
openshift_registry_selector="role=infra"
openshift_router_selector="role=infra"
openshift_master_cluster_method=native
openshift_cloudprovider_kind=vmware

[masters]
master-0.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
'master'}"
master-1.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
'master'}"
master-2.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
'master'}"

[etcd]
master-0.vcenter.e2e.bos.redhat.com
master-1.vcenter.e2e.bos.redhat.com
master-2.vcenter.e2e.bos.redhat.com

[nodes]
master-0.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
'master'}"
master-1.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
'master'}"
master-2.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role':
```

```
'master'}"
infra-0.vcenter.e2e.bos.redhat.com  openshift_node_labels="{'role':
'infra'}"
infra-1.vcenter.e2e.bos.redhat.com  openshift_node_labels="{'role':
'infra'}"
app-0.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role': 'app'}"
app-1.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role': 'app'}"
app-2.vcenter.e2e.bos.redhat.com openshift_node_labels="{'role': 'app'}"
```

E.2. RUNNING THE UNINSTALL PLAYBOOK

The uninstall playbook removes Red Hat OpenShift related packages, etcd, and removes any certificates that were created during the failed install.

```
ansible-playbook -i ~/inventory /usr/share/ansible/openshift-
ansible/playbooks/adhoc/uninstall.yml
```

E.3. MANUALLY LAUNCHING THE INSTALLATION OF RED HAT OPENSHIFT

The playbook below is the same playbook that is ran once the deployment of VMware resources is completed. Replace the rhsm user and password, set the wildcard_zone and public_hosted_zone relevant to the environment.

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ && ./ocp-on-vmware.py --tag ocp-install
```

E.4. STARTING COMPLETELY OVER

The following options can be completed with the ocp-on-vmware.py script to remove all the VMs from vCenter. This utilizes the infrastructure.json created during Section 3.3, “ocp-on-vmware.py - inventory” inventory:

```
$ cd ~/git/openshift-ansible-contrib/reference-architecture/vmware-
ansible/ && ./ocp-on-vmware.py --clean
```

E.5. TROUBLESHOOTING CNS DEPLOYMENT FAILURES

If the CNS deployment process fails, it is possible to use the following command to clean up all the resource that were created in the current installation:

```
$ cns-deploy -n <project_name> -g topology.json --abort
```

There are a couple of recurring reasons why the deployment might fail: The current OCP user doesn’t have permission in the current project. The OCP app nodes don’t have connectivity to the Red Hat Registry to download the GlusterFS container images. The firewall rules on the app nodes is blocking traffic on one or more ports. The initialization of the block devices referenced in the topology fails because there are some unexpected partitioning structures. Use the following command to completely wipe the disk of VMware virtual machines being used for `CNS` cluster deployments.

```
$ sgdisk --zap-all /dev/<block-device>
```

An alternative reason for failure could be the device specified is already part of a LVM volume group (potentially due to a previous failed run of the **`cns-deploy`** installer), remove it with the following commands on the OCP node that the device(s) is connected to (i.e. ocp3-app-cns-3). This must be done on all nodes referenced in the **`topology.json`** file.

```
$ lvremove -y vg_xxxxxxxxxxxxxxxx
$ pvremove /dev/<block-device>
```

APPENDIX F. REVISION HISTORY

Revision 3.6-0	**September 13, 2017**	**Davis Phillips**

- **OpenShift Release 3.6**

Revision 3.5.2-0	**July 26, 2017**	**Annette Clewett, Davis Phillips**

- **Addition of CNS and storage chapter**

Revision 3.5.1-0	**May 24, 2017**	**Ryan Cook**

- **Addition of OpenShift Cluster Metrics**
- **Inventory Issue fix**

Revision 3.5.0-0	**May 11, 2017**	**Ryan Cook**

- **OpenShift Release 3.5**

Revision 3.4.0-0	**Jan 24, 2017**	**Davis Phillips**

- **OpenShift Release 3.4**