Reference Architectures 2017 Deploying OpenShift Container Platform 3.5 on Amazon Web Services

Ryan Cook Scott Collier

Reference Architectures 2017 Deploying OpenShift Container Platform 3.5 on Amazon Web Services

Ryan Cook

Scott Collier
refarch-feedback@redhat.com

Legal Notice

Abstract

The purpose of this document is to provide guidelines and considerations for installing and configuring Red Hat OpenShift Container Platform on Amazon Web Services.

Table of Contents

COMMENTS AND FEEDBACK

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

CHAPTER 1. EXECUTIVE SUMMARY

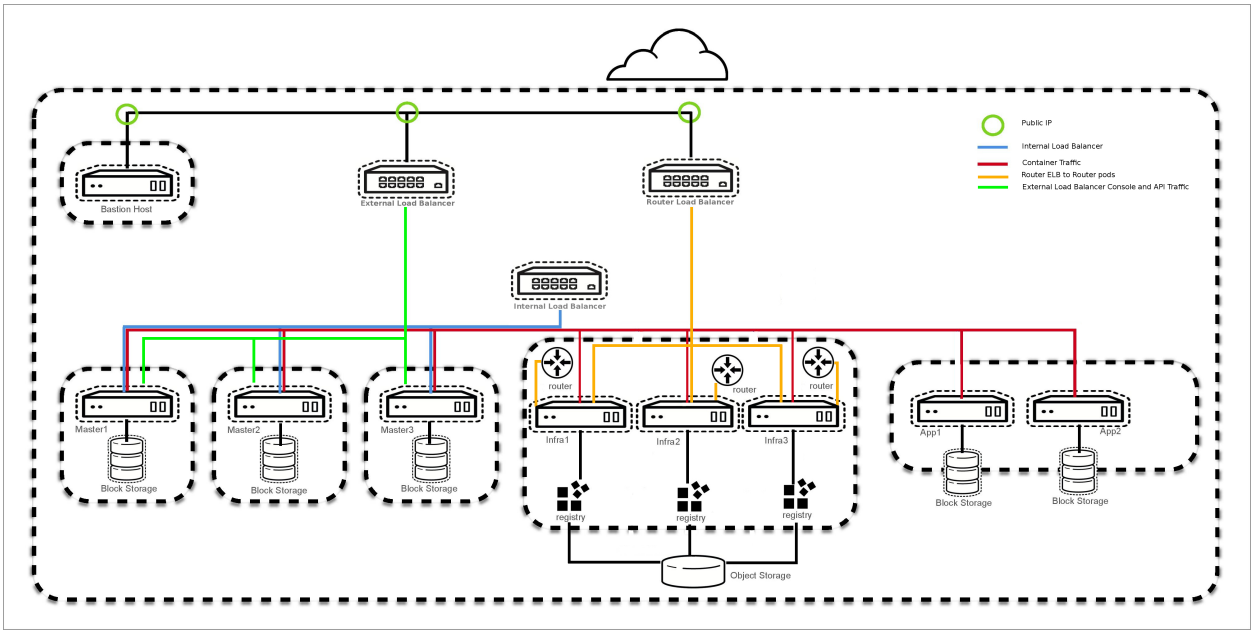
Red Hat® OpenShift Container Platform 3 is built around a core of application containers, with orchestration and management provided by Kubernetes, on a foundation of Atomic Host and Red Hat Enterprise Linux. OpenShift Origin is the upstream community project that brings it all together along with extensions, 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 Amazon Web Services in order to create a highly available OpenShift Container Platform 3 environment. The configuration consists of three OpenShift Container Platform masters, three OpenShift Container Platform infrastructure nodes, two OpenShift Container Platform application nodes, and native Amazon Web Services integration. In addition to the configuration, operational management tasks are shown to demonstrate functionality.

CHAPTER 2. COMPONENTS AND CONFIGURATION

This chapter describes the highly available OpenShift Container Platform 3 reference architecture environment on Amazon Web Services (AWS) that is deployed.

The image below provides a high-level representation of the components within this reference architecture. Using Amazon Web Services (AWS), resources are highly available using a combination of multiple **availability zones**, Elastic Load Balancers(ELB), and an **S3** bucket. Instances deployed are given specific roles to support OpenShift. The Bastion host limits the external access to internal servers by ensuring that all **SSH** traffic passes through the Bastion host. 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. Authentication is managed by GitHub OAuth. OpenShift on **AWS** has two cloud native storage options; Elastic Block Storage is used for the filesystem of instances but can also be used for persistent storage in containers. The other storage option is **S3** which is object based storage. **S3** is used for the persistent storage of the OpenShift registry. The network is configured to leverage three **AWS ELBs** for access to the OpenShift API, OpenShift console, and the OpenShift routers. The first **ELB** is for the OpenShift API and console access originating from outside of the cluster. The second **ELB** is for API access within the cluster. The third ELB is for accessing services deployed in the cluster that have been exposed through routes. Finally, the image shows that **DNS** is handled by **Route53**. In this case the systems engineering team is managing all **DNS** entries through **Route53**.



This reference architecture breaks down the deployment into separate phases.

- Phase 1: Provision the infrastructure on **AWS**
- Phase 2: Provision OpenShift Container Platform on **AWS**
- 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 installation of OpenShift Container Platform which is done via Ansible playbooks. These playbooks are installed by the **openshift-ansible-playbooks** rpm package. The playbooks in **openshift-ansible-contrib** utilize the playbooks provided by the **openshift-ansible-playbooks** RPM package to perform the

installation of OpenShift and also to configure **AWS** specific parameters. During Phase 2 the router and registry are deployed. The last phase, Phase 3, concludes the deployment by confirming the environment was deployed properly. This is done by running tools like **oadm diagnostics** and the systems engineering teams **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. ELASTIC COMPUTE CLOUD INSTANCE DETAILS

Within this reference environment, the instances are deployed in multiple **availability zones** in the **us-east-1** region by default. Although the default region can be changed, the reference architecture deployment can only be used in Regions with three or more **availability zones**. The master instances for the OpenShift environment are **m4.xlarge** and contain three extra disks used for Docker storage, OpenShift ephemeral volumes, and **ETCD**. The node instances are **t2.large** and contain two extra disks used for Docker storage and OpenShift ephemeral volumes. The bastion host is a **t2.micro**. Instance sizing can be changed in the variable files for each installer which is covered in later chapters.

2.2. ELASTIC LOAD BALANCERS DETAILS

Three load balancers are used in the reference environment. The table below describes the load balancer **DNS** name, the instances in which the **ELB** is attached, and the port monitored by the load balancer to state whether an instance is in or out of service.

Table 2.1. Elastic Load Balancers

ELB	Assigned Instances	Port
openshift-master.sysdeseng.com	master01-3	443
internal-openshift-master.sysdeseng.com	master01-3	443
*.apps.sysdeseng.com	infra-nodes01-3	80 and 443

Both the **internal-openshift-master**, and the **openshift-master ELB** utilize the OpenShift Master API port for communication. The **internal-openshift-master ELB** uses the private subnets for internal cluster communication with the API in order to be more secure. The **openshift-master ELB** is used for externally accessing the OpenShift environment through the API or the web interface. The **openshift-master ELB** uses the public subnets to allow communication from anywhere over port 443. The *.apps **ELB** 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.

2.3. SOFTWARE VERSION DETAILS

The following tables provide the installed software versions for the different servers that make up the Red Hat OpenShift highly available reference environment.

Table 2.2. RHEL OSEv3 Details

Software	Version
Red Hat Enterprise Linux 7.3 x86_64	kernel-3.10.0.x
Atomic-OpenShift{master/clients/node/sdn-ovs/utils}	3.5.x.x
Docker	1.12.x
Ansible	2.2.1.x

2.4. REQUIRED CHANNELS

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

Table 2.3. Required Channels - OSEv3 Master and Node Instances

Channel	Repository Name
Red Hat Enterprise Linux 7 Server (RPMs)	rhel-7-server-rpms
Red Hat OpenShift Container Platform 3.5 (RPMs)	rhel-7-server-ose-3.5-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms

Channel	Repository Name
Red Hat Enterprise Linux Fast Datapath (RHEL 7 Server) (RPMs)	rhel-7-fast-datapath-rpms

2.5. AWS REGION REQUIREMENTS

The reference architecture environment must be deployed in a Region containing at least 3 **availability zones** and have 2 free elastic IPs. The environment requires 3 public and 3 private subnets. The usage of 3 public and 3 private subnets allows for the OpenShift deployment to be highly-available and only exposes the required components externally. The subnets can be created during the installation of the reference architecture environment deployment.

2.6. PERMISSIONS FOR AMAZON WEB SERVICES

The deployment of OpenShift requires a user that has the proper permissions by the **AWS IAM** administrator. The user must be able to create accounts, **S3** buckets, roles, policies, **Route53** entries, and deploy **ELBs** and **EC2** instances. It is helpful to have delete permissions in order to be able to redeploy the environment while testing.

2.7. VIRTUAL PRIVATE CLOUD (VPC)

An **AWS VPC** provides the ability to set up custom virtual networking which includes subnets, IP address ranges, route tables and gateways. In this reference implementation guide, a dedicated **VPC** is created with all its accompanying services to provide a stable network for the OpenShift v3 deployment.

A **VPC** is created as a logical representation of a networking environment in the **AWS** cloud. The following subnets and CIDR listed below are used. Substitute the values to ensure no conflict with an existing CIDR or subnet in the environment. The values are defined in **/home/<user>/git/openshift-ansible-contrib/reference-architecture/aws-ansible/playbooks/vars/main.yaml**.

Table 2.4. VPC Networking

CIDR / Subnet	Values
CIDR	10.20.0.0/16
Private Subnet 1	10.20.1.0/24
Private Subnet 2	10.20.2.0/24

CIDR / Subnet	Values
Private Subnet 3	10.20.3.0/24
Public Subnet 1	10.20.4.0/24
Public Subnet 2	10.20.5.0/24
Public Subnet 3	10.20.6.0/24

The **VPC** is created and a human readable tag is assigned. Six subnets are created and tagged in the **VPC**. Three subnets are considered public and three subnets are private. The design of one public and one private subnet per Availability Zone allows for high availability(HA). The public subnets are used for the bastion instance and the two external **ELBs**. The bastion instance is part of the public subnet due to its role as the **SSH** jumpbox. The two external **ELBs** allow access to the OpenShift master and the routing of application traffic. In the public route table, an internet gateway and routes are defined and attached to the **VPC**. The route table has a destination internet gateway associated so that traffic can exit the **VPC**. The private subnets use the NAT Gateway to communicate to the internet for packages, container images, and Github repositories. The private subnets are assigned their own route table with the NAT Gateway defined. The master, infrastructure, and application nodes are in the private network, as well as the internal-openshift-master, which ensures the nodes cannot be accessed externally.

For more information see https://aws.amazon.com/vpc/

2.8. NAT GATEWAY

The reference architecture deployment utilizes the **AWS** NAT Gateway Service to ensure that instances in the private subnets have the ability to download packages, container images, and Github repositories. The NAT Gateway Service funnels all external traffic from private subnets to the outside world. This allows for a smaller external footprint and does not use unneeded public IP and public **DNS** entries.

2.9. SECURITY GROUPS

In this reference architecture, eight groups are created. The purpose of the security groups is to restrict traffic from outside of the **VPC** to servers inside of the **VPC** . The security groups also are used to restrict server to server communications inside the **VPC**. Security groups provide an extra layer of security similar to a firewall. In the event a port is opened on an instance, the security group will not allow the communication to the port unless explicitly stated in a security group. See the tables below for details on each security group.

2.9.1. Master ELB Security Group

The Master **ELB** security group allows inbound access on port 443 from the internet to the **ELB**. The traffic is then allowed to be forwarded to the master instances. See Figure 2.1, “AWS Master ELB

Security Group Details - Inbound" diagram and Table 2.5, "AWS Master ELB Security Group Details - Inbound" table below.

Figure 2.1. AWS Master ELB Security Group Details - Inbound

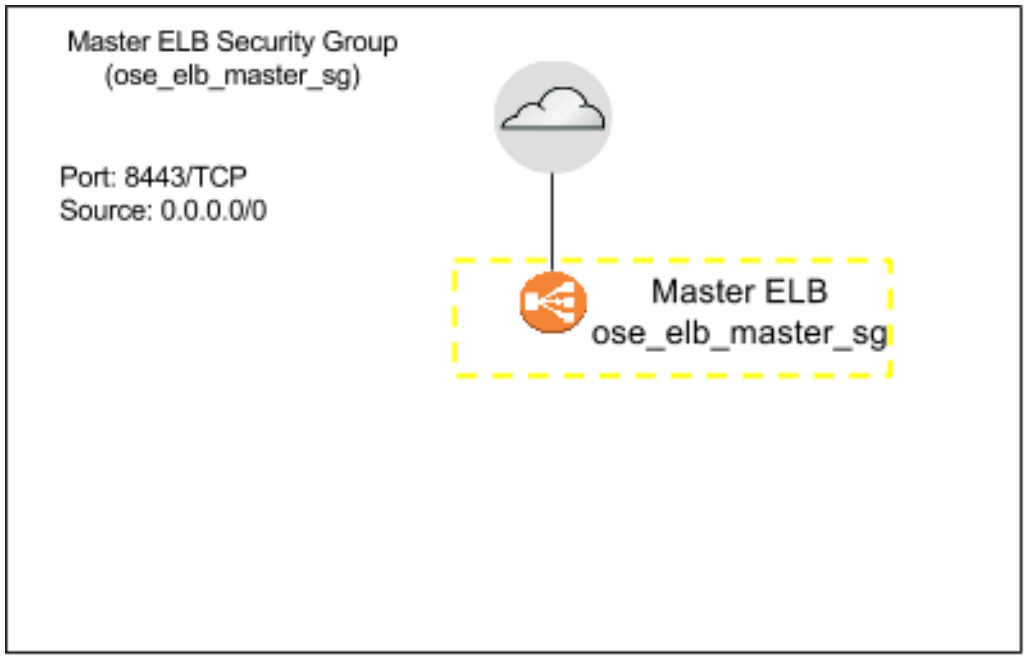


Table 2.5. AWS Master ELB Security Group Details - Inbound

Inbound	From
443 / TCP	Anywhere

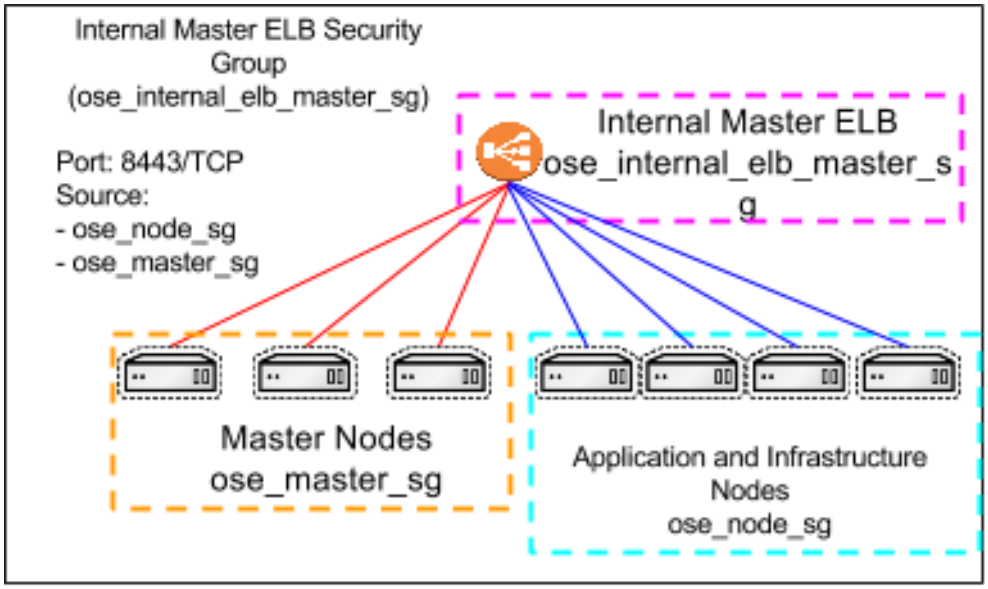
Table 2.6. AWS Master ELB Security Group Details - Outbound

Outbound	To
443	ose_master_sg

2.9.2. Internal Master ELB Security Group

The Internal Master **ELB** is in the private subnet and utilizes the NAT Gateway. Traffic external from the **VPC** cannot access the Internal Master **ELB**.

Figure 2.2. AWS Internal Master ELB Security Group Details - Inbound



Inbound	From
443 / TCP	ose_node_sg
443 / TCP	ose_master_sg

Table 2.7. AWS Internal Master ELB Security Group Details - Outbound

Outbound	To
443	ose_master_sg

2.9.3. Bastion Security Group

The bastion security group allows inbound port **SSH** traffic from outside the **VPC**. Any connectivity via **SSH** to the master, application or infrastructure nodes must go through the bastion host. Ensure the bastion host is secured per your companies security requirements.

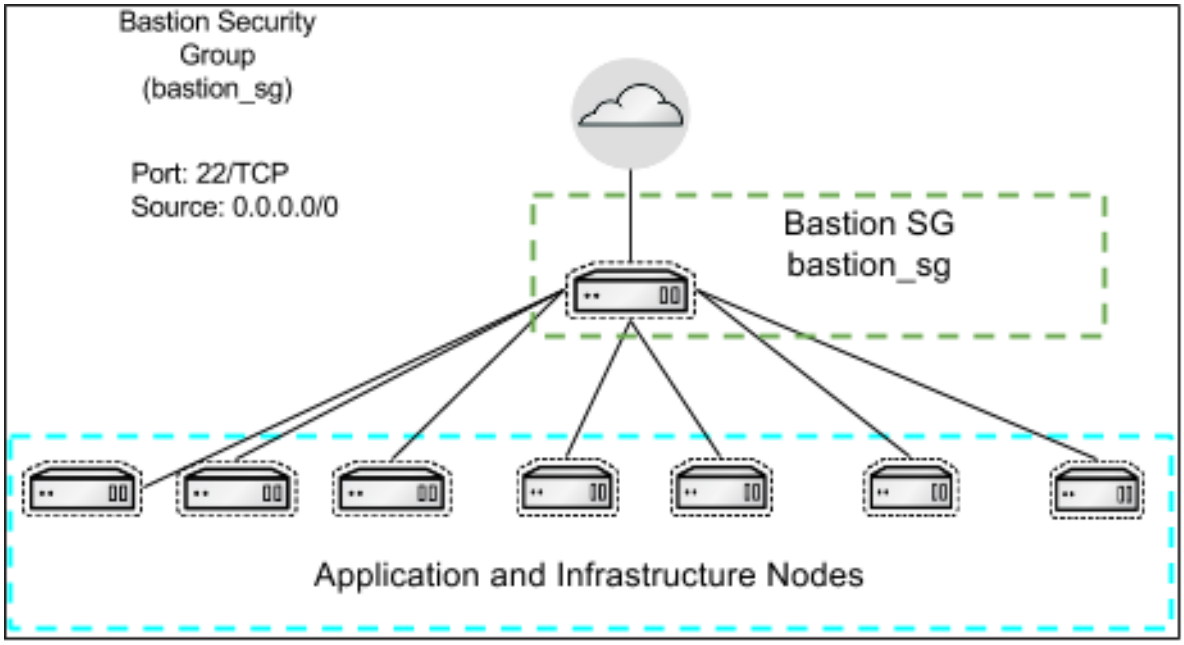


Table 2.8. AWS Bastion Security Group Details - Inbound

Inbound	From
22 / TCP	Anywhere

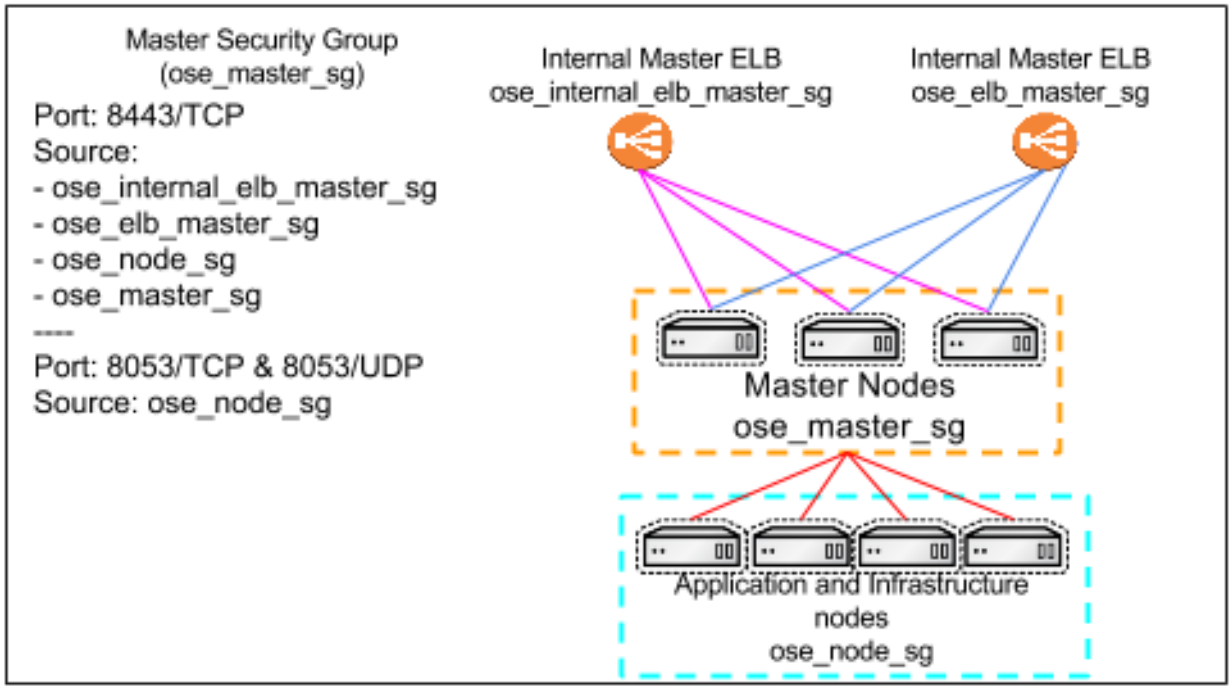
Table 2.9. AWS Bastion Security Group Details - Outbound

Outbound	To
All	All

2.9.4. Master Security Group

The master security group allows traffic to the master instances from the two **ELBs** and nodes to contact the **OpenShift API** and **DNS**.

Figure 2.3. AWS Master Security Group Details - Inbound



Inbound	From
8053 / TCP	ose_node_sg
8053 / UDP	ose_node_sg
443 / TCP	ose_internal_elb_master_sg
443 / TCP	ose_elb_master_sg
443 / TCP	ose_node_sg
443 / TCP	ose_master_sg

Table 2.10. AWS Master Security Group Details - Outbound

Outbound	To
All	All

2.9.5. ETCD Security Group

The **ETCD** security group allows for the **ETCD** service running on the master instances to reach a quorum. The security group allows for the **ose-master-sg** to communication with the **ETCD** for the OpenShift master services.

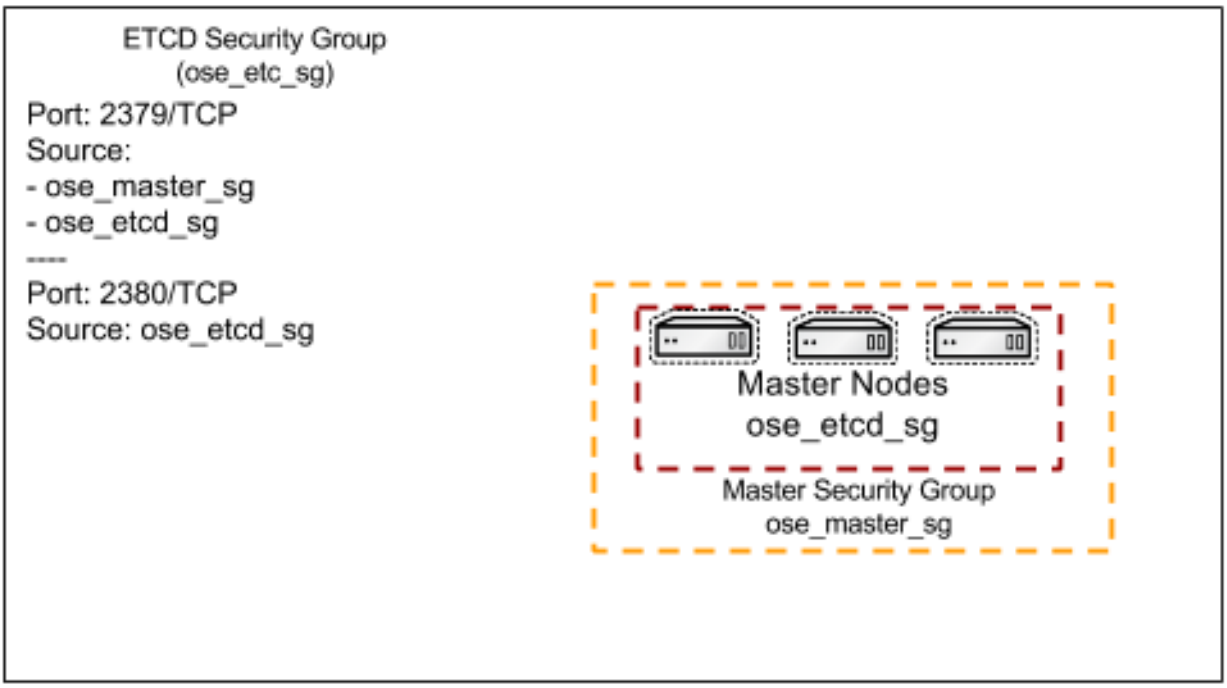


Table 2.11. ETCD Security Group Details - Inbound

Inbound	From
2379 / TCP	ose-etcd-sg
2379 / TCP	ose-master-sg
2380 / TCP	ose-etcd-sg

Table 2.12. ETCD Security Group Details - Outbound

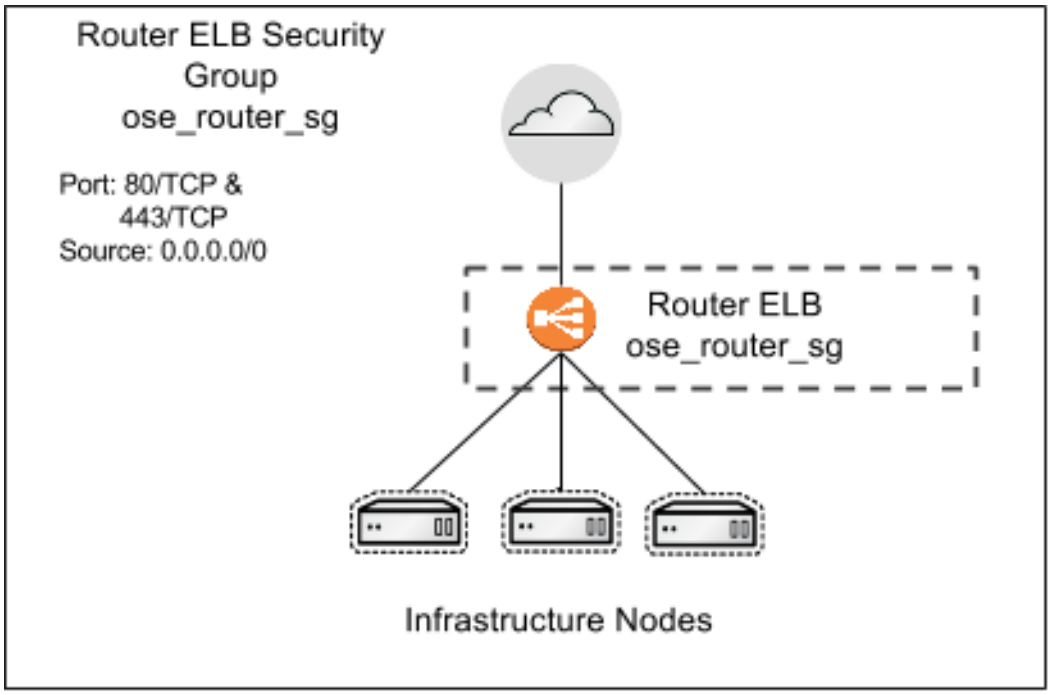
Outbound	To
All	All

2.9.6. Router ELB Security Group

The Router **ELB** security group allows inbound access on port 80 and 443. If the applications

running on the OpenShift cluster are using different ports this can be adjusted as needed.

Figure 2.4. AWS Router ELB Security Group Details - Inbound



Inbound	From
443 / TCP	Anywhere
80 / TCP	Anywhere

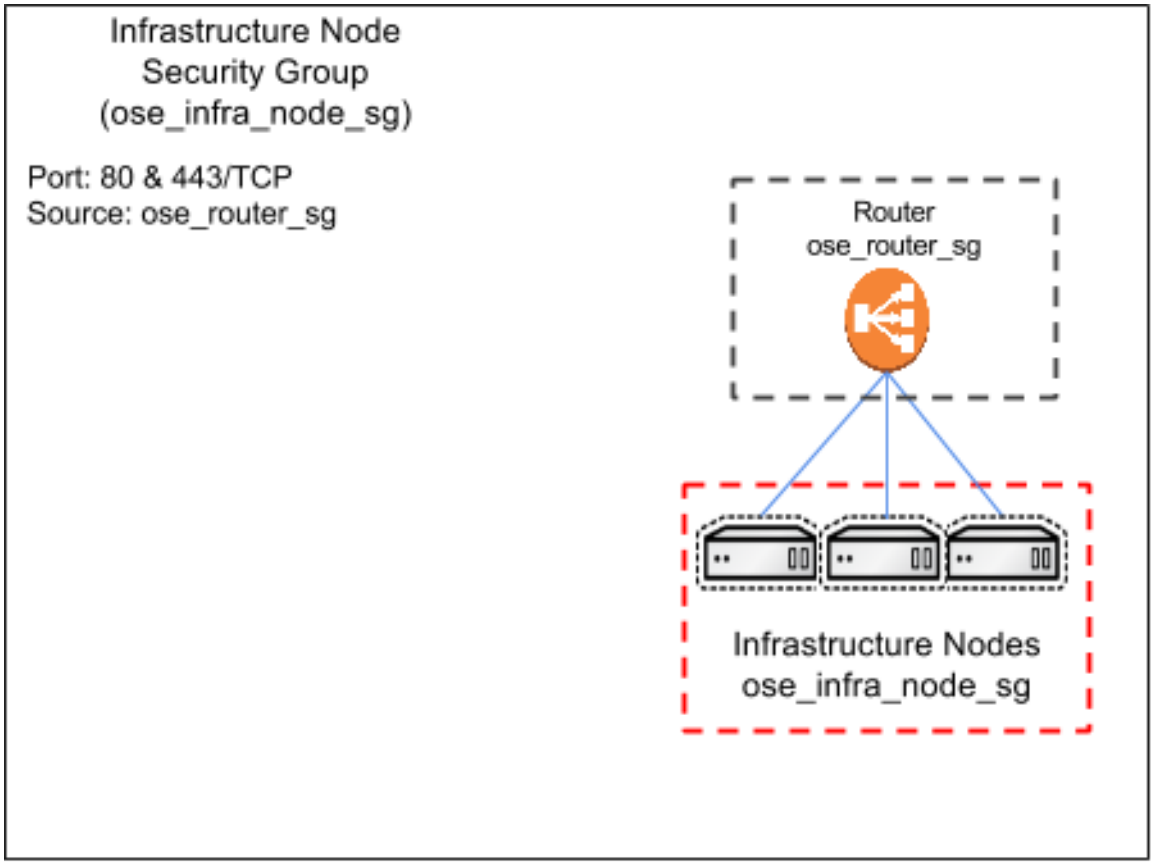
Table 2.13. AWS Router ELB Security Group Details - Outbound

Outbound	To
80	ose_infra_node_sg
443	ose_infra_node_sg

2.9.7. Infrastructure Nodes Security Group

The infrastructure nodes security group allows traffic from the router security group.

Figure 2.5. AWS Infrastructure Nodes Security Group Details - Inbound



Inbound	From
80 / TCP	ose_router_sg
443 / TCP	ose_router_sg

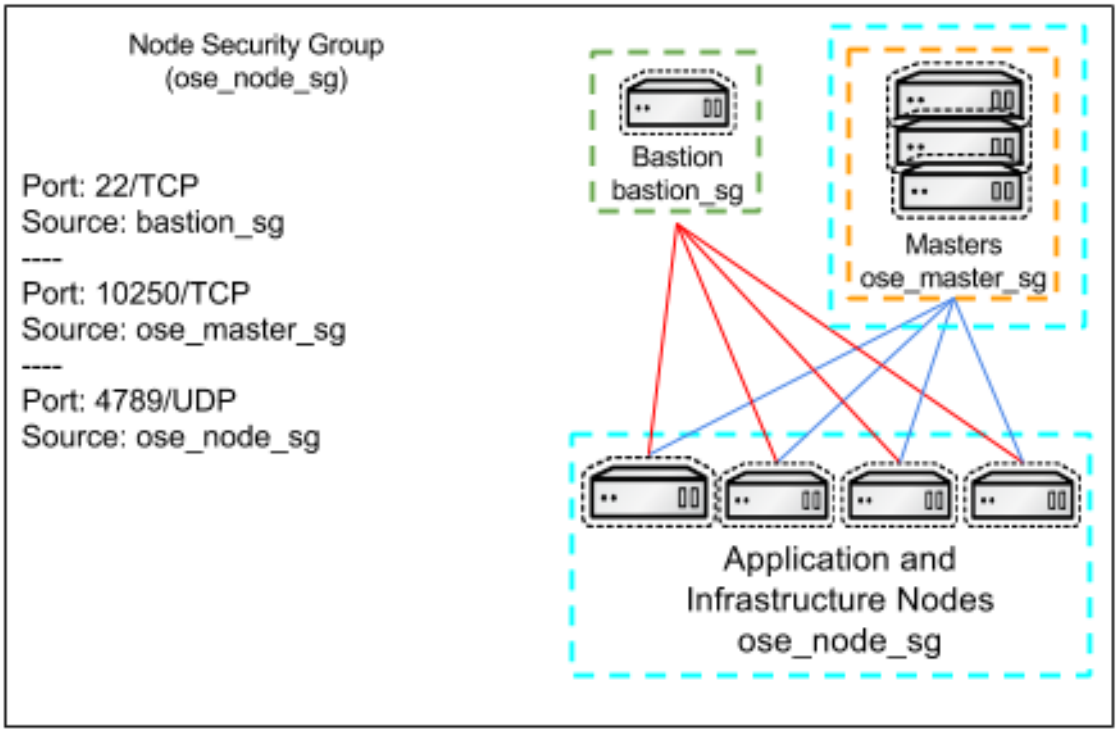
Table 2.14. AWS Infrastructure Nodes Security Group Details - Outbound

Outbound	To
All	All

2.9.8. Nodes Security Group

The node security group only allows traffic from the bastion and traffic relating to OpenShift node services.

Figure 2.6. AWS Nodes Security Group Details - Inbound



Inbound	From
22 / TCP	bastion_sg
10250 / TCP	ose_master_sg
10250 / TCP	ose_node_sg
4789 / UDP	ose_node_sg

Table 2.15. AWS Application Nodes Security Group Details - Outbound

Outbound	To
All	All

2.10. ROUTE53

DNS is an integral part of a successful OpenShift Compute Platform deployment/environment. **AWS** has a **DNS** web service, per Amazon; "Amazon Route 53 is a highly available and scalable cloud**DNS** web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to internet applications by translating names like

www.example.com into numeric IP addresses like 192.0.2.1 that computers use to connect to each other."

OpenShift Compute 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 Configuring A DNS Wildcard. In this reference architecture **Route53** will manage **DNS** records for the OpenShift Container Platform environment.

For more information see https://aws.amazon.com/route53/

2.10.1. Public Zone

The reference architecture environment automatically adds entries into **Route53**. A Hosted Zone is required for OpenShift. A domain name can either purchased through **AWS** or another external provider such as Google Domains or GoDaddy. If using a zone from an external provider ensure the NS records point to a **Route53** hosted zone. Steps will be detailed in Chapter 3 of this document.

2.11. AMAZON MACHINE IMAGES

Amazon Machine Images (AMIs) provide the required information to launch an instance. In this guide, the gold image provided by Red Hat is used. The **AMI** is shared to a specific **AWS** account which is priced less than the Red Hat Enterprise Linux image provided by **AWS**.

For more information see AWS Documentation.

2.11.1. Red Hat Gold Image

The Red Hat Cloud Access provided gold image allows Instances to be run at a cheaper cost than using the Amazon provided RHEL image. Since a subscription is required to install OpenShift then it is not necessary to use the Amazon provided image which has a built in charge back for the RHEL subscription.

To register for the Red Hat Cloud Access Gold Image please see Red Hat's Website and select the tab for Red Hat Gold Image.

2.12. IDENTITY AND ACCESS MANAGEMENT

AWS provides **IAM** to securely control access to **AWS** services and resources for users. **IAM** can allow or deny access to certain resources for user accounts and for roles within the **AWS** environment. For this reference architecture, an **IAM** account will need access to create roles, instances, **Route53** entries, **ELBs**, and many more components. The predefined policy **AdministratorAccess** has been proven to provide all of the access required to create the environment defined in the this document.

During the installation of OpenShift Container Platform, one account is automatically created to manage a **S3** bucket used for the registry. A role and policy are also created to allow for attaching and detaching of EBS volumes for persistent storage within the environment.

For more information see https://aws.amazon.com/iam/

2.13. DYNAMIC INVENTORY

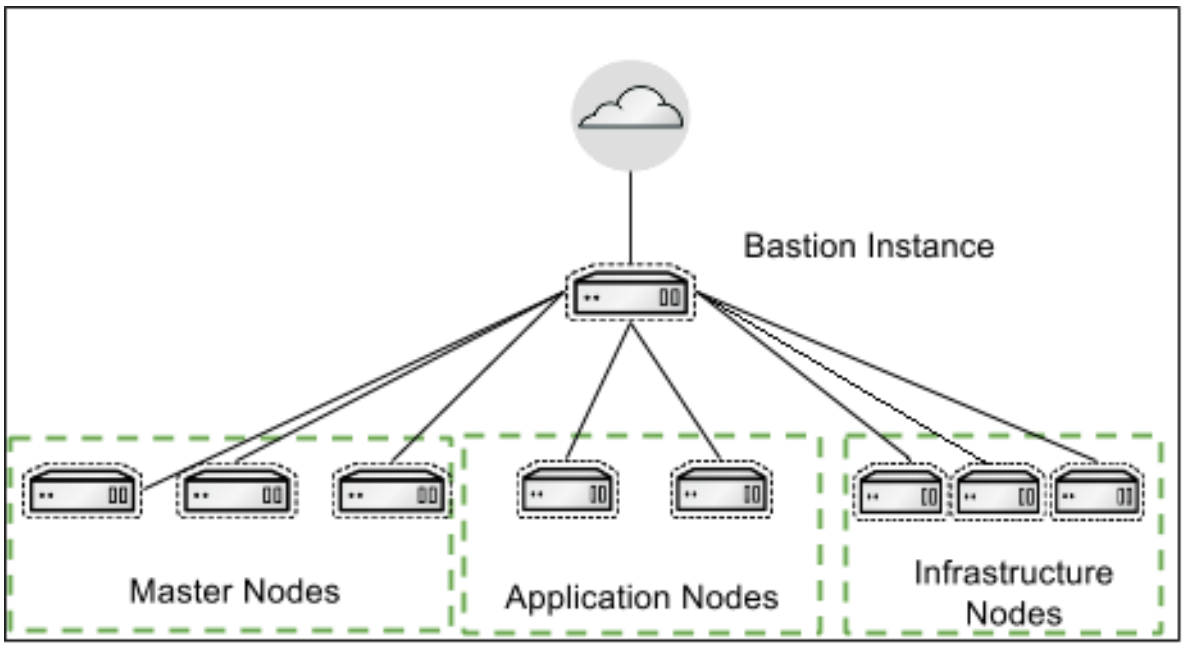
Ansible relies on inventory files and variables to perform playbook runs. As part of the reference architecture provided Ansible playbooks, the inventory is created automatically using a dynamic inventory script. The dynamic inventory script provided queries the Amazon API to display information about **EC2** instances. The dynamic inventory script is also referred to as an Ansible Inventory script and the **AWS** specific script is written in python. The script can manually be executed to provide information about the environment but for this reference architecture, it is automatically called to generate the Ansible Inventory. For the 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. The reason the instances can be grouped is because during Phase 1 when the infrastructure was provisioned **AWS EC2** tags were applied to each instance. The masters were assigned the **master** tag, the infrastructure nodes were assigned the **infra** tag, and the application nodes were assigned the **app** tag.

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

2.14. BASTION

As shown in the Figure 2.7, “Bastion Diagram” the **bastion** server in this reference architecture provides a secure way to limit **SSH** access to the **AWS** environment. The master and node security groups only allow for **SSH** connectivity between nodes inside of the Security Group while the bastion allows **SSH** access from everywhere. The bastion host is the only ingress point for **SSH** in the cluster from external entities. When connecting to the OpenShift Container Platform infrastructure, the bastion forwards the request to the appropriate server. Connecting through the bastion server requires specific **SSH** configuration. The **.ssh/config** is outlined in the deployment section of the reference architecture guide.

Figure 2.7. Bastion Diagram



2.15. NODES

Nodes are **AWS** instances that serve a specific purpose for OpenShift. OpenShift masters are also considered nodes. Nodes deployed on **AWS** can be vertically scaled before or after the OpenShift installation using the **AWS EC2** console. All OpenShift specific nodes are assigned an **IAM** role

which allows for cloud specific tasks to occur against the environment such as adding persistent volumes or removing a node from the OpenShift Container Platform cluster automatically. There are three types of nodes as described below.

2.15.1. Master nodes

The master nodes contain the master components, including the API server, controller manager server and **ETCD**. The master maintains the clusters configuration, manages nodes in its OpenShift cluster. The master assigns pods to nodes and synchronizes pod information with service configuration. The master is used to define routes, services, and volume claims for pods deployed within the OpenShift environment.

2.15.2. 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 **S3** which allows for multiple pods to use the same storage. **AWS S3** storage is used because it is synchronized between the availability zones, providing data redundancy.

2.15.3. Application nodes

The Application nodes are the instances where non-infrastructure based containers run. Depending on the application, **AWS** specific storage can be applied such as a **Elastic Block Storage** 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 OpenShift Container Platform user containers will be placed on the application nodes by default.

2.15.4. Node labels

All 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 projects automatically are deployed on Application nodes.

2.16. OPENSHIFT PODS

OpenShift uses the Kubernetes concept of a pod, which is one or more containers deployed together on one host, and 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 has an ephemeral database. 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.

2.17. ROUTER

Pods inside of an 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
- TLS with 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 publically accessible URL. The wildcard zone itself is a wildcard entry in **Route53** which is linked using a CNAME to an **ELB** which performs a health check and forwards traffic to router pods on port 80 and 443.

2.18. REGISTRY

OpenShift can build Docker images from your source code, deploy them, and manage their lifecycle. 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 a **S3** bucket. The **S3** bucket allows for multiple pods to be deployed at once for HA but also use the same persistent backend storage. **S3** is object based storage which does not get assigned to nodes in the same way that EBS volumes are attached and assigned to a node. The bucket does not mount as block based storage to the node so commands like fdisk or lsblk will not show information in regards to the **S3** bucket. The configuration for the **S3** bucket and credentials to login to the bucket are stored as OpenShift secrets and applied to the pod. The registry can be scaled to many pods and even have multiple instances of the registry running on the same host due to the use of **S3**.

2.19. AUTHENTICATION

There are several options when it comes to authentication of users in 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 identification providers occurs on the OpenShift master instances. OpenShift allows for multiple identity providers to be specified. The reference architecture document uses GitHub 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 OpenShift documentation.

CHAPTER 3. DEPLOYING OPENSHIFT

This chapter focuses on Phase 1 and 2 of the process. The prerequisites defined below are required for a successful deployment of Infrastructure and the installation of OpenShift.

3.1. PREREQUISITES FOR PROVISIONING

The script and playbooks provided within the git repository deploys infrastructure, installs and configures OpenShift, and scales 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 **S3** bucket to manage container images.

3.1.1. Tooling Prerequisites

This section describes how the environment should be configured to use Ansible to provision the infrastructure, install OpenShift, and perform post installation tasks.

Note

The following tasks should be performed on the workstation that the Ansible playbooks will be launched from. The workstation can be an virtual machine, workstation, vagrant vm, or instance running in the cloud. The most important requirement is that the system is running RHEL 7.

3.1.1.1. Ansible Setup

Install the following packages on the system performing the provisioning of **AWS** infrastructure and installation of OpenShift.

Note

The ordering of the installation of **Ansible** before the installation of the **epelRPM** is important. If the **Ansible** package is intalled from **epel** then the version may be incompatible with OpenShift.

```
$ rpm -q python-2.7
$ subscription-manager repos --enable rhel-7-server-optional-rpms
$ subscription-manager repos --enable rhel-7-server-ose-3.5-rpms
$ subscription-manager repos --enable rhel-7-fast-datapath-rpms
$ yum -y install ansible atomic-openshift-utils
$ rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-
7.noarch.rpm
$ yum -y install python2-boto \
                 python2-boto3 \
                 pyOpenSSL \
                 git \
                 python-netaddr \
                 python-click \
                 python-httplib2
```

3.1.1.2. Git Repository

3.1.1.3. GitHub Repositories

The code in the **`openshift-ansible-contrib`** repository referenced below handles the installation of OpenShift and the accompanying infrastructure. The **`openshift-ansible-contrib`** repository is not explicitly supported by Red Hat but the Reference Architecture team performs testing to ensure the code operates as defined and is secure.

3.1.1.4. Directory Setup

```
$ cd /home/<user>/git
$ git clone https://github.com/openshift/openshift-ansible-contrib.git
```

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

```
$ yum -y install tree
$ tree /home/<user>/git/

... content abbreviated ...

|-- openshift-ansible-contrib
```

3.1.2. Authentication

As mentioned in the previous section, Authentication for the reference architecture deployment is handled by GitHub OAuth. The steps below describe both the process for creating an organization and performing the configuration steps required for GitHub authentication.

3.1.2.1. Create an Organization

An existing organization can be used when using GitHub authentication. If an organization does not exist then one must be created. The **`ose-on-aws.py`** script forces an organization to be defined. The script forces an organization due to the fact that if no organizations are provided all of GitHub can login to the OpenShift environment. GitHub users will need to be added to the organization and be at least a member but they also could be an owner.

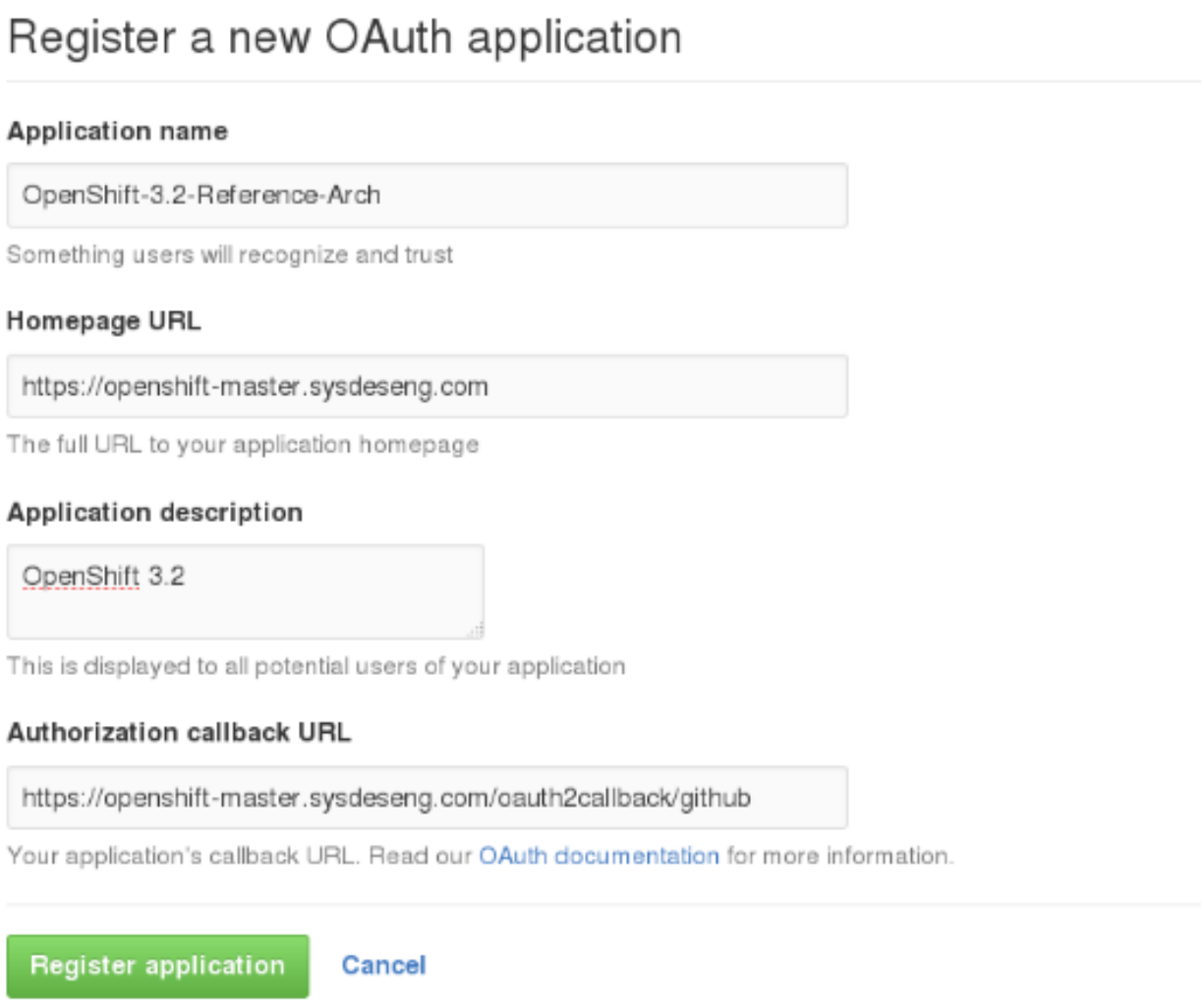
Follow the directions in the link provided to create an organization.
https://help.github.com/articles/creating-a-new-organization-from-scratch/

3.1.2.2. Configuring OAuth

Browse to https://github.com/settings/applications/new and login to GitHub

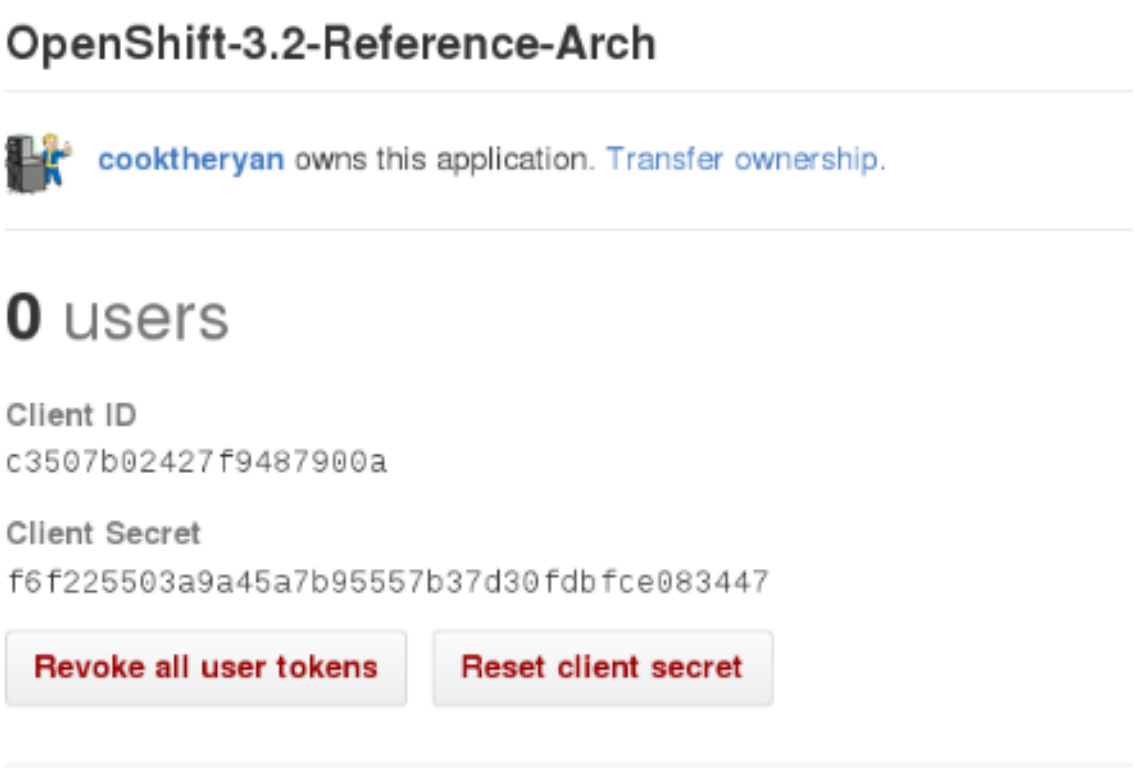
The image below will provide an example configuration. Insert values that will be used during the OpenShift deployment.

Figure 3.1. GitHub OAuth Application



- Insert an Application name
- Insert a Homepage **URL** (This will be the **URL** used when accessing OpenShift)
- Insert an Application description (Optional)
- Insert an Authorization callback URL (The entry will be the Homepage **URL** + /oauth2callback/github
- Click Register application

Figure 3.2. GitHub OAuth Client ID



A Client ID and Client Secret will be presented. These values will be used as variables during the installation of Openshift.

3.1.3. DNS

In this reference implementation guide a domain called **sysdeseng.com** domain was purchased through **AWS** and managed by **Route53**. In the example below, the domain **sysdeseng.com** will be the hosted zone used for the installation of OpenShift. Follow the below instructions to add the main hosted zone.

- From the main **AWS** dashboard, in the Networking section click **Route53**
  - Click Hosted Zones
    - Click Create Hosted Zone
      - Input a Domain Name: **sysdeseng.com**
      - Input a Comment: Public Zone for RH Reference Architecture
      - Type: Public Hosted Zone
        - Click Create

A subdomain can also be used. The same steps listed above are applicable when using a subdomain. Once the Pubic Zone is created select the radio button for the Domain and copy the Name Servers from the right and add those to the external registrar or top level domain in **Route53**. Ensure that the Name Servers(NS) records are copied to the root domain or name resolution will not work for the subdomain.

3.1.4. SSH

3.1.4.1. SSH Configuration

Before beginning the deployment of the **AWS** Infrastructure and the deployment of OpenShift, a specific **SSH** configuration must be in place to ensure that **SSH** traffic passes through the bastion instance. If this configuration is not in place the deployment of the infrastructure will be successful but the deployment of OpenShift will fail. Use the domain or subdomain configured during Section 2.10, “Route53” to fill in the values below. For example, the domain sysdeseng.com was used so the bastion will be bastion.sysdeseng.com and the wildcard will be *.sysdeseng.com.

Note

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

```
$ cat /home/<user>/.ssh/config

Host bastion
     HostName                bastion.sysdeseng.com
     User                      ec2-user
     StrictHostKeyChecking     no
     ProxyCommand              none
     CheckHostIP               no
     ForwardAgent              yes
     IdentityFile              /home/<user>/.ssh/id_rsa

Host *.sysdeseng.com
     ProxyCommand              ssh ec2-user@bastion -W %h:%p
     user                      ec2-user
     IdentityFile              /home/<user>/.ssh/id_rsa
```

Table 3.1. SSH Configuration

Option	Purpose
Host Bastion	Configuration Alias
Hostname	Hostname of the bastion instance
user	Remote user to access the bastion instance
StrictHostKeyChecking	Automatically add new host keys to known host file
ProxyCommand	Not required for the bastion

Option	Purpose
CheckHostIP	Key checking is against hostname rather than IP
ForwardAgent	Used to forward the SSH connection
IdentityFile	Key used to access bastion instance
Host *.sysdeseng.com	Wildcard for all *.sysdeseng instances
ProxyCommand	SSH command used to jump from the bastion host to another host in the environment
IdentityFile	Key used for all *.sysdeseng instances

Note

In the event an environment needs to be redeployed the entries in .ssh/known_hosts will need to be removed or the installation will not occur due to ssh failing because of "WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!".

3.1.5. AWS Authentication

3.1.5.1. AWS Configuration

The **`AWS Access Key ID`** and **`Secret Access Key`** must be exported on the workstation executing the Ansible playbooks. This account must have the ability to create **`IAM`** users, **`IAM`** Policies, and **`S3`** buckets.

If the **`ACCESS KEY ID`** and **`SECRET ACCESS KEY`** were not already created follow the steps provided by **`AWS`**.

http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSGettingStartedGuide/AWSCredentials

To export the Access Key ID and Secret perform the following on the workstation performing the deployment of **`AWS`** and OpenShift:

```
$ export AWS_ACCESS_KEY_ID=<key_id>
$ export AWS_SECRET_ACCESS_KEY=<access_key>
```

3.1.6. Red Hat Subscription

The installation of OpenShift Container Platform (OCP) requires a valid Red Hat subscription. During the installation of OpenShift the **Ansible** redhat_subscription module will attempt to register the instances. The script will fail if the OpenShift entitlements are exhausted. For the installation of **OCP** on **AWS** the following items are required:

Red Hat Subscription Manager User: < Red Hat Username >

Red Hat Subscription Manager Password: < Red Hat Password >

Subscription Name: Red Hat OpenShift Container Platform, Standard, 2-Core

The items above are examples and should reflect subscriptions relevant to the account performing the installation. There are a few different variants of the OpenShift Subscription Name. It is advised to visit https://access.redhat.com/management/subscriptions to find the specific Subscription Name as the values will be used below during the deployment.

3.2. PROVISIONING THE ENVIRONMENT

Within the **openshift-ansible-contrib** git repository is a python script called **ose-on-aws.py** that launches **AWS** resources and installs OpenShift on the new resources. Intelligence is built into the playbooks to allow for certain variables to be set using options provided by the **ose-on-aws.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 **AWS** deployment to the OpenShift deployment and post installation tasks.

Note

The ose-on-aws.py script does not validate EC2 instance limits. Using a web browser login to an **AWS** account that has access to deploy EC2 instances and Select EC2. Next, select Limits to view the current instance size limits based on the **AWS** acount.

3.2.1. The ose-on-aws.py Script

The **ose-on-aws.py** script contains many different configuration options such as the ability to change the AMI, instance size, and the ability to use a currently deployed bastion host. The region can be changed but keep in mind the AMI may need to be changed if the Red Hat Cloud Access gold image AMI ID is different. The **Cloudformation** stack name is also configurable. Setting **--stack-name** to a unique value ensures that one **Cloudformation** stack does not overwrite another **Cloudformation** stack. The script creates both an auto-generated **S3 bucket name** and an **IAM** user account to be used for the registry. These values can be changed before launching. To see all of the potential options the **--help** trigger is available.

Note

The **ose-on-aws.py** script requires the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY exported as an environment variable.

```
$ ./ose-on-aws.py --help
Options:
  --stack-name TEXT               Cloudformation stack name. Must be
```

```
unique
                                [default: openshift-infra]
  --console-port INTEGER RANGE    OpenShift web console port  [default:
443]
  --deployment-type [origin|openshift-enterprise]
                                OpenShift deployment type
[default:
                                openshift-enterprise]
  --openshift-sdn [openshift-ovs-subnet|openshift-ovs-multitenant]
                                OpenShift SDN  [default: openshift-
ovs-
                                subnet]
  --region TEXT                 ec2 region  [default: us-east-1]
  --ami TEXT                    ec2 ami  [default: ami-a33668b4]
  --master-instance-type TEXT     ec2 instance type  [default:
m4.xlarge]
  --node-instance-type TEXT       ec2 instance type  [default:
t2.large]
  --app-instance-type TEXT        ec2 instance type  [default:
t2.large]
  --bastion-instance-type TEXT    ec2 instance type  [default:
t2.micro]
  --keypair TEXT                ec2 keypair name
  --create-key TEXT             Create SSH keypair  [default: no]
  --key-path TEXT               Path to SSH public key. Default is
/dev/null
                                which will skip the step  [default:
                                /dev/null]
  --create-vpc TEXT             Create VPC  [default: yes]
  --vpc-id TEXT                 Specify an already existing VPC
  --private-subnet-id1 TEXT       Specify a Private subnet within the
existing
                                VPC
  --private-subnet-id2 TEXT       Specify a Private subnet within the
existing
                                VPC
  --private-subnet-id3 TEXT       Specify a Private subnet within the
existing
                                VPC
  --public-subnet-id1 TEXT        Specify a Public subnet within the
existing
                                VPC
  --public-subnet-id2 TEXT        Specify a Public subnet within the
existing
                                VPC
  --public-subnet-id3 TEXT        Specify a Public subnet within the
existing
                                VPC
  --public-hosted-zone TEXT       hosted zone for accessing the
environment
  --app-dns-prefix TEXT           application dns prefix  [default:
apps]
  --rhsm-user TEXT              Red Hat Subscription Management User
  --rhsm-password TEXT          Red Hat Subscription Management
Password
  --rhsm-pool TEXT              Red Hat Subscription Management Pool
```

```
ID or
                                    Subscription Name
  --byo-bastion TEXT                skip bastion install when one exists
within
                                    the cloud provider  [default: no]
  --bastion-sg TEXT                 Specify Bastion Security group used
with
                                    byo-bastion  [default: /dev/null]
  --containerized TEXT              Containerized installation of
OpenShift
                                    [default: False]
  --s3-bucket-name TEXT             Bucket name for S3 for registry
  --github-client-id TEXT           GitHub OAuth ClientID
  --github-client-secret TEXT       GitHub OAuth Client Secret
  --github-organization TEXT        GitHub Organization
  --s3-username TEXT                S3 user for registry access
  --no-confirm                      Skip confirmation prompt
  -h, --help                        Show this message and exit.
  -v, --verbose
```

The -v trigger is available as well. This will allow for **Ansible** to run more verbose allowing for a more in-depth output of the steps occurring while running **ose-on-aws.py**.

3.2.2. Containerized Deployment

The OCP installation playbooks allow for OpenShift to be installed in containers. These containers can run on either Atomic Host or RHEL. If the containerized installation of OpenShift is preferred specify **--containerized=true** while running **ose-on-aws.py**. If using Atomic Host the configuration trigger --containerized=true must be specified or the installation will fail. Also, when using Atomic Host ensure the **AMI** being used has Docker 1.10 installed.

3.2.3. SDN Selection

Two software-defined networks are provided as choices when deploying OpenShift on **AWS** openshift-ovs-subnet and openshift-ovs-multitenant. By default the openshift-ovs-subnet is configured as the **SDN**.

- 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.

3.2.4. Greenfield Deployment

For deploying OpenShift into a new environment, **ose-on-aws.py** creates instances, load balancers, **Route53** entries, and **IAM** users an ssh key can be entered to be uploaded and used with the new instances. Once the values have been entered into the **ose-on-aws.py** script all values will be presented and the script will prompt to continue with the values or exit. By default, the Red Hat gold image AMI Section 2.11, “Amazon Machine Images” is used when provisioning

instances but can be changed when executing the **ose-on-aws.py**. The keypair in the example below **OSE-key** is the keypair name as it appears within the **AWS EC2** dashboard. If a keypair has not been created and uploaded to **AWS** perform the steps below to create, upload, and name the **SSH** keypair.

Create a Public/Private key

If a user does not currently have a public and private **SSH** key perform the following.

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Created directory '/home/user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:SpfGaSv23aDasVsIRPfTNsXaOAbfiuSJ1Pj+e5tN52Y
user@goku.rdu.redhat.com
The key's randomart image is:
+---[RSA 2048]----+
| . .    ..       |
|. . ... ..       |
| .  ooo+*        |
|.  o BoX.o       |
|o = @ B S.       |
| = O X o. .      |
|  = = .  o .     |
|   o .  o.+      |
|  .   .ooE..     |
+----[SHA256]-----+
```

Create a Public/Private key

To deploy the environment using the newly created private/public **SSH** key which currently does not exist within **AWS** perform the following.

```
$ export AWS_ACCESS_KEY_ID=<key_id>
$ export AWS_SECRET_ACCESS_KEY=<access_key>
$ ./ose-on-aws.py --stack-name=dev --create-key=yes --rhsm-user=rhsm-
user \ --rhsm-password=rhsm-password \ --public-hosted-
zone=sysdeseng.com \ --key-path=/home/<user>/.ssh/id_rsa.pub --
keypair=OSE-key \ --rhsm-pool="Red Hat OpenShift Container Platform,
Standard, 2-Core"
```

If an SSH key has already been uploaded to **AWS** specify the name of the keypair as it appears within the **AWS EC2** dashboard.

```
$ ./ose-on-aws.py --stack-name= dev --rhsm-user=rhsm-user --rhsm-
password=rhsm-password \ --public-hosted-zone=sysdeseng.com --
keypair=OSE-key \ --github-client-
secret=47a0c41f0295b451834675ed78aecfb7876905f9 \ --github-
organization=openshift \ --github-organization=RHSyseng --github-
client-id=3a30415d84720ad14abc \ --rhsm-pool="Red Hat OpenShift
Container Platform, Standard, 2-Core"
```

Example of Greenfield Deployment values

```
Configured values:
    stack_name: dev
    ami: ami-a33668b4
    region: us-east-1
    master_instance_type: m4.xlarge
    node_instance_type: t2.large
    app_instance_type: t2.large
    bastion_instance_type: t2.micro
    keypair: OSE-key
    create_key: no
    key_path: /dev/null
    create_vpc: yes
    vpc_id: None
    private_subnet_id1: None
    private_subnet_id2: None
    private_subnet_id3: None
    public_subnet_id1: None
    public_subnet_id2: None
    public_subnet_id3: None
    byo_bastion: no
    bastion_sg: /dev/null
    console port: 443
    deployment_type: openshift-enterprise
    openshift_sdn: openshift-ovs-subnet
    public_hosted_zone: sysdeseng.com
    app_dns_prefix: apps
    apps_dns: apps.sysdeseng.com
    rhsm_user: rhsm_user
    rhsm_password: *******
    rhsm_pool: Red Hat OpenShift Container Platform, Standard, 2-Core
    containerized: False
    s3_bucket_name: dev-ocp-registry-sysdeseng
    s3_username: dev-s3-openshift-user
    github_client_id: *******
    github_client_secret: *******
    github_organization: openshift,RHSyseng


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

3.2.5. Brownfield Deployment

The **`ose-on-aws.py`** script allows for deployments into an existing environment in which a **VPC** already exists and at least six subnets, three public and three private, are already created. The script expects three public and three private subnets are created. The private subnets must be able to connect externally which requires a NAT gateway to be deployed. Before running the brownfield deployment ensure that a NAT gateway is deployed and proper **`Route Table`** entries are created (private subnets, 0.0.0.0/0 → nat-xxxx; public subnets, 0.0.0.0/0 → igw-xxxx). By default, the Red Hat gold image AMI is used when provisioning instances but can be changed when executing the **`ose-on-aws.py`**.

Running the following will prompt for subnets and the **VPC** to deploy the instances and OpenShift.

```
$ ./ose-on-aws.py --stack-name=dev --rhsm-user=rhsm-user --rhsm-
password=rhsm-password \ --public-hosted-zone=sysdeseng.com --
keypair=OSE-key --create-vpc=no \ --github-client-
secret=47a0c41f0295b451834675ed78aecfb7876905f9 \ --github-
organization=openshift \ --github-organization=RHSyseng --github-
client-id=3a30415d84720ad14abc \ --rhsm-pool="Red Hat OpenShift
Container Platform, Standard, 2-Core"

Specify the VPC ID: vpc-11d06976
Specify the first Private subnet within the existing VPC: subnet-
3e406466
Specify the second Private subnet within the existing VPC: subnet-
66ae905b
Specify the third Private subnet within the existing VPC: subnet-
4edfd438
Specify the first Public subnet within the existing VPC: subnet-
1f416547
Specify the second Public subnet within the existing VPC: subnet-
c2ae90ff
Specify the third Public subnet within the existing VPC: subnet-
1ddfd46b
```

In the case that a bastion instance has already been deployed, an option **`--byo-bastion=yes`** exists within **`ose-on-aws.py`** exists to not deploy the bastion instance.

Note

If the bastion instance is already deployed supply the security group id of the bastion security group. The existing bastion host must be in the same AWS region as the deployment. The bastion host must have the hostname of bastion either through an A record or CNAME.

```
$ ./ose-on-aws.py --stack-name=dev --rhsm-user=rhsm-user --rhsm-
password=rhsm-password \ --public-hosted-zone=sysdeseng.com --
keypair=OSE-key --byo-bastion=yes --create-vpc=no \ --github-client-
secret=47a0c41f0295b451834675ed78aecfb7876905f9 \ --github-
organization=openshift \ --github-organization=RHSyseng --github-
client-id=3a30415d84720ad14abc \ --bastion-sg=sg-a34ff3af \ --rhsm-
pool="Red Hat OpenShift Container Platform, Standard, 2-Core"

Specify the VPC ID: vpc-11d06976
Specify the first Private subnet within the existing VPC: subnet-
3e406466
Specify the second Private subnet within the existing VPC: subnet-
66ae905b
Specify the third Private subnet within the existing VPC: subnet-
4edfd438
Specify the first Public subnet within the existing VPC: subnet-
1f416547
Specify the second Public subnet within the existing VPC: subnet-
c2ae90ff
Specify the third Public subnet within the existing VPC: subnet-
1ddfd46b
Specify the the Bastion Security group(example: sg-4afdd24): sg-
```

```
a34ff3af
```

As stated in the Greenfield deployment the option exists to not use the Red Hat Cloud Access provided gold image AMI. Using the same command from above the **--ami=** option allows the default value to be changed.

```
$ ./ose-on-aws.py --create-vpc=no --rhsm-user=rhsm-user --rhsm-
password=rhsm-password \ --public-hosted-zone=sysdeseng.com --
keypair=OSE-key --byo-bastion=yes \ --github-client-
secret=47a0c41f0295b451834675ed78aecfb7876905f9 \ --github-
organization=openshift \ --github-organization=RHSyseng --github-
client-id=3a30415d84720ad14abc \ --bastion-sg=sg-a34ff3af --ami=ami-
2051294a --rhsm-pool="Red Hat OpenShift Container Platform, Standard,
2-Core" --stack-name=dev

Specify the VPC ID: vpc-11d06976
Specify the first Private subnet within the existing VPC: subnet-
3e406466
Specify the second Private subnet within the existing VPC: subnet-
66ae905b
Specify the third Private subnet within the existing VPC: subnet-
4edfd438
Specify the first Public subnet within the existing VPC: subnet-
1f416547
Specify the second Public subnet within the existing VPC: subnet-
c2ae90ff
Specify the third Public subnet within the existing VPC: subnet-
1ddfd46b
Specify the the Bastion Security group(example: sg-4afdd24): sg-
a34ff3af
```

Example of Brownfield Deployment values

```
    stack_name: dev
    ami: ami-a33668b4
    region: us-east-1
    master_instance_type: m4.xlarge
    node_instance_type: t2.large
    app_instance_type: t2.large
    bastion_instance_type: t2.micro
    keypair: OSE-key
    create_key: no
    key_path: /dev/null
    create_vpc: yes
    vpc_id: vpc-11d06976
    private_subnet_id1: subnet-3e406466
    private_subnet_id2: subnet-66ae905b
    private_subnet_id3: subnet-4edfd438
    public_subnet_id1: subnet-1f416547
    public_subnet_id2: subnet-c2ae90ff
    public_subnet_id3: subnet-1ddfd46b
    byo_bastion: no
    bastion_sg: /dev/null
    console port: 443
    deployment_type: openshift-enterprise
    openshift_sdn: openshift-ovs-subnet
```

```
    public_hosted_zone: sysdeseng.com
    app_dns_prefix: apps
    apps_dns: apps.sysdeseng.com
    rhsm_user: rhsm_user
    rhsm_password: *******
    rhsm_pool: Red Hat OpenShift Container Platform, Standard, 2-Core
    containerized: False
    s3_bucket_name: dev-ocp-registry-sysdeseng
    s3_username: dev-s3-openshift-user
    github_client_id: *******
    github_client_secret: *******
    github_organization: openshift,RHSyseng

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

3.3. POST ANSIBLE DEPLOYMENT

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 the Appendix to restart the installation of OpenShift.

3.4. POST PROVISIONING RESULTS

At this point the infrastructure and Red Hat OpenShift Container Platform have been deployed. Log into the **AWS** console and check the resources. In the **AWS** console, check for the following resources:

- 3 Master nodes
- 3 Infrastructure nodes
- 2 Application nodes
- 1 Unique **VPC** with the required components
- 8 Security groups
- 2 Elastic IPs
- 1 NAT Gateway
- 1 Key pair
- 3 **ELBs**
- 2 **IAM** roles
- 2 **IAM** Policies
- 1 **S3** Bucket
- 1 **IAM** user
- 1 Zones in **Route53**

Information is also available in the **CloudFormation** output. This information describes some of the currently deployed items like the subnets, security groups, and etc. These outputs can be used

by the **`add-node.py`**, **`add-cns-storage.py`**, and **`add-crs-storage.py`** scripts to auto-populate variables like security groups and other required values.

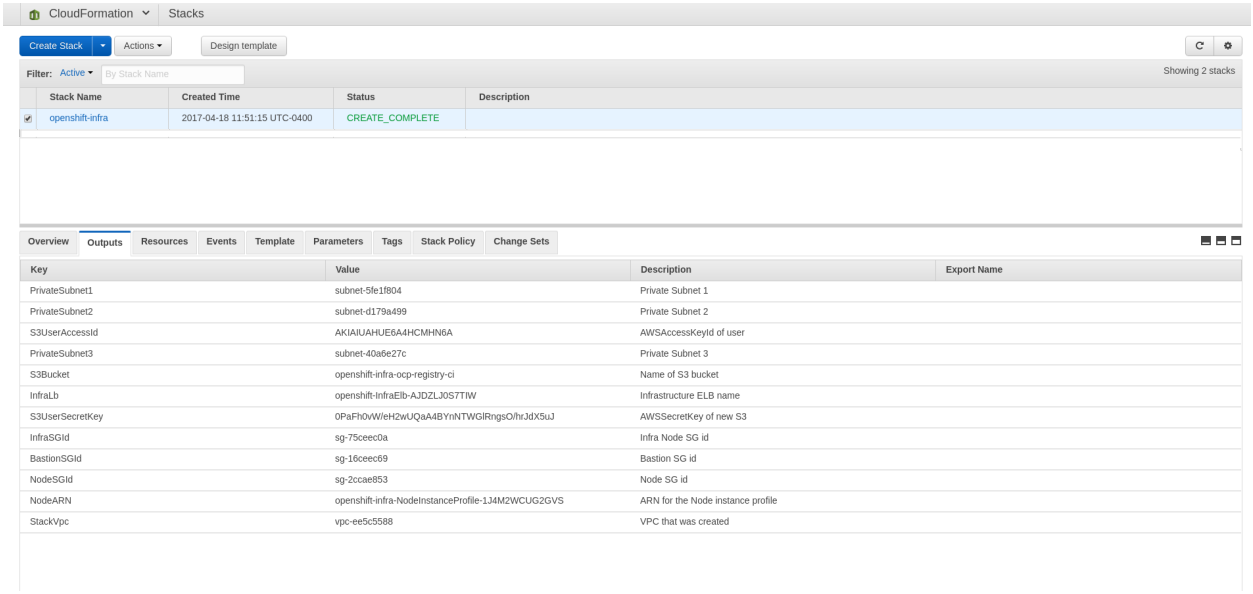


Table 3.2. Cloudformation Outputs

Key	Value	Description
PrivateSubnet1	subnet-ea3f24b1	Private Subnet 1
PrivateSubnet2	subnet-1a02dd52	Private Subnet 2
S3UserAccessId	AKIAJVDMBDLYBJGNHXSA	AWSAccessKeyId of user
PrivateSubnet3	subnet-22adea1e	Private Subnet 3
S3Bucket	openshift-infra-ocp-registry-sysdeseng	Name of S3 bucket
InfraLb	openshift-infra-InfraElb-1X0YK42S95B28	Infrastructure ELB name
S3UserSecretKey	bbiabom7XPbNyGUJf1Dy8EDB8cSyo4z9y5PiZCY+	AWSSecretKey of new S3

InfraSGId	sg-9f456de0	Infra Node SG id
BastionSGId	sg-53456d2c	Bastion SG id
NodeSGId	sg-85466efa	Node SG id
NodeARN	openshift-infra-NodeInstanceProfile-7WF689M7WLT1	ARN for the Node instance profile
StackVpc	vpc-9c0b00fa VPC that was createdPrivateSubnet1 subnet-ea3f24b1	Private Subnet 1
PrivateSubnet2	subnet-1a02dd52	Private Subnet 2
S3UserAccessId	AKIAJVDMBDLYBJGNHXSA	AWSAccessKeyId of user
PrivateSubnet3	subnet-22adea1e	Private Subnet 3
S3Bucket	openshift-infra-ocp-registry-sysdeseng	Name of S3 bucket
InfraLb	openshift-infra-InfraElb-1X0YK42S95B28	Infrastructure ELB name
S3UserSecretKey	bbiabom7XPbNyGUJf1Dy8EDB8cSyo4z9y5PiZCY+	AWSSecretKey of new S3
InfraSGId	sg-9f456de0	Infra Node SG id
BastionSGId	sg-53456d2c	Bastion SG id
NodeSGId	sg-85466efa	Node SG id

NodeARN	openshift-infra-NodeInstanceProfile-7WF689M7WLT1	ARN for the Node instance profile
StackVpc	vpc-9c0b00fa	VPC that was created

At this point, the OpenShift public URL will be available using the public hosted zone URL provided while running the ose-on-aws.py. For example, https://openshift-master.sysdeseng.com.

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 the Red Hat OpenShift Container Platform.

4.1. VALIDATE THE DEPLOYMENT

With the successful deployment of OpenShift, the following section demonstrates how to confirm proper functionality of the OpenShift environment. 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 **ELB** address from the installation system
- Validate the public OpenShift **ELB** address from the master nodes
- Validate the internal OpenShift **ELB** address from the master nodes
- Validate the master 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 /home/<user>/git/openshift-ansible-contrib/reference-
architecture/aws-ansible
$ ansible-playbook -i inventory/aws/hosts/ -e
'public_hosted_zone=sysdeseng.com wildcard_zone=apps.sysdeseng.com
console_port=443 stack_name=dev' playbooks/validation.yaml
```

4.2. GATHERING HOSTNAMES

With all of the steps that occur during the installation of 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 browse to the **AWS EC2** dashboard and select **Running Instances** under Resources. Selecting

Running Resources shows all instances currently running within **EC2**. To view only instances specific to the reference architecture deployment filters can be used. Under Instances → Instances within **EC2** click beside the magnifying glass. Select a **Tag Key** such as **openshift-role** and click **All values**. The filter shows all instances relating to the reference architecture deployment.

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

- ose-master01.sysdeseng.com
- ose-master02.sysdeseng.com
- ose-master03.sysdeseng.com
- ose-infra-node01.sysdeseng.com
- ose-infra-node02.sysdeseng.com
- ose-infra-node03.sysdeseng.com
- ose-app-node01.sysdeseng.com
- ose-app-node02.sysdeseng.com

4.3. RUNNING DIAGNOSTICS

Perform the following steps from the first master node.

To run diagnostics, **SSH** into the first master node (ose-master01.sysdeseng.com). Direct access is provided to the first master node because of the configuration of the local **~/.ssh/config** file.

```
$ ssh ec2-user@ose-master01.sysdeseng.com
$ sudo -i
```

Connectivity to the first master node (ose-master01.sysdeseng.com) as the **root** user should have been established. Run the diagnostics that are included as part of the install.

```
# oadm diagnostics
... ommitted ...
[Note] Summary of diagnostics execution (version v3.5.5.5):
[Note] Warnings seen: 8
```

Note

The warnings will not cause issues in the environment

Based on the results of the diagnostics, actions can be taken to alleviate 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 (ose-master01.sysdeseng.com). Using the output of the command **hostname** issue the **etcdctl** command to confirm that the cluster is healthy.

```
$ ssh ec2-user@ose-master01.sysdeseng.com
$ sudo -i
```

```
# hostname
ip-10-20-1-106.ec2.internal
# etcdctl -C https://ip-10-20-1-106.ec2.internal:2379 --ca-file
/etc/etcd/ca.crt --cert-file=/etc/origin/master/master.etcd-client.crt
--key-file=/etc/origin/master/master.etcd-client.key cluster-health
member 82c895b7b0de4330 is healthy: got healthy result from
https://10.20.1.106:2379
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.20.3.74:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.20.2.157:2379
```

Note

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

4.5. DEFAULT NODE SELECTOR

As explained in section 2.12.4 node labels are an important part of the OpenShift environment. By default of 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 and the master API service is restarted that is required when making any changes to the master configuration.

SSH into the first master node (ose-master01.sysdeseng.com) to verify the **defaultNodeSelector** is defined.

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

Note

If making any changes to the master configuration then the master API service must be restarted or the configuration change will not take place. Any changes and the subsequent restart must be done on all masters.

4.6. MANAGEMENT OF MAXIMUM POD SIZE

Quotas are set on ephemeral volumes within pods to prohibit a pod from becoming to 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 using a combination of user-data and Ansible.

Openshift Volume Quota

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

SSH into the first infrastructure node (ose-infra-node01.sysdeseng.com) to verify the entry exists within fstab.

```
# vi /etc/fstab
/dev/xvdc /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/xvdb** and create the volume group of **docker-vol**. The extra Docker storage options ensures that a container can grow no larger than 3G. Docker storage setup is performed on all master, infrastructure, and application nodes.

SSH into the first infrastructure node (ose-infra-node01.sysdeseng.com) to verify **/etc/sysconfig/docker-storage-setup** matches the information below.

```
# vi /etc/sysconfig/docker-storage-setup
DEVS=/dev/xvdb
VG=docker-vol
DATA_SIZE=95%VG
EXTRA_DOCKER_STORAGE_OPTIONS="--storage-opt dm.basesize=3G"
```

OpenShift Emptydir Quota

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

SSH into the first infrastructure node (ose-infra-node01.sysdeseng.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. YUM REPOSITORIES

In section 2.3 Required Channels the specific repositories for a successful OpenShift installation were defined. All systems except for the bastion host should have the same subscriptions. To verify subscriptions match those defined in Required Channels perform the following. The repositories

below are enabled during the rhsm-repos playbook during the installation. The installation will be unsuccessful if the repositories are missing from the system.

```
# *yum repolist*
Loaded plugins: amazon-id, rhui-lb, search-disabled-repos,
subscription-manager
repo id                                                 repo name
status
rhel-7-server-extras-rpms/x86_64                        Red Hat
Enterprise Linux 7 Server - Extras (RPMs)                   249
rhel-7-fast-datapath-rpms/7Server/x86_64                Red Hat
Enterprise Linux Fast Datapath (RHEL 7 Server) (RPMs)        27
rhel-7-server-ose-3.5-rpms/x86_64                       Red Hat
OpenShift Container Platform 3.5 (RPMs)                  404+10
rhel-7-server-rpms/7Server/x86_64                       Red Hat
Enterprise Linux 7 Server (RPMs)                         11,088
!rhui-REGION-client-config-server-7/x86_64              Red Hat Update
Infrastructure 2.0 Client Configuration Server 7       6
!rhui-REGION-rhel-server-releases/7Server/x86_6         Red Hat
Enterprise Linux Server 7 (RPMs)                         11,088
!rhui-REGION-rhel-server-rh-common/7Server/x86_         Red Hat
Enterprise Linux Server 7 RH Common (RPMs)                   196
repolist: 23,196
```

Note

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

4.8. CONSOLE ACCESS

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

4.8.1. Log into GUI console and deploy an application

Perform the following steps from the local workstation.

Open a browser and access https://openshift-master.sysdeseng.com/console. When logging into the OpenShift web interface the first time the page will redirect and prompt for GitHub credentials. Log into GitHub using an account that is a member of the Organization specified during the install. Next, GitHub will prompt to grant access to authorize the login. If GitHub access is not granted the account will not be able to login to the OpenShift web console.

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

4.8.2. Log into CLI 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://openshift-master.sysdeseng.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://openshift-master.sysdeseng.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://openshift-master.sysdeseng.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://openshift-master.sysdeseng.com
```

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

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

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

4.9. EXPLORE THE ENVIRONMENT

4.9.1. List Nodes and Set Permissions

If you try to run the following command, it should fail.

```
# oc get nodes --show-labels
Error from server: User "sysdes-admin" 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 (ose-master01.sysdeseng.com).

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

Attempt to list the nodes again and show the labels.

```
# oc get nodes --show-labels
NAME                          STATUS                     AGE
ip-10-30-1-164.ec2.internal   Ready                      1d
ip-10-30-1-231.ec2.internal   Ready                      1d
ip-10-30-1-251.ec2.internal   Ready,SchedulingDisabled   1d
ip-10-30-2-142.ec2.internal   Ready                      1d
ip-10-30-2-157.ec2.internal   Ready,SchedulingDisabled   1d
ip-10-30-2-97.ec2.internal    Ready                      1d
ip-10-30-3-74.ec2.internal    Ready,SchedulingDisabled   1d
```

4.9.2. List Router and Registry

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

Note

If the OpenShift account configured on the workstation has cluster-admin privileges perform the following. If the account does not have this privilege ssh to one of the OpenShift masters and perform the steps.

```
# oc project default
# oc get all
NAME                          REVISION        DESIRED       CURRENT
TRIGGERED BY
dc/docker-registry            1               3             3
config
dc/router                     1               3             3
config
NAME                          DESIRED         CURRENT       AGE
rc/docker-registry-1          3               3             10m
rc/router-1                   3               3             10m
NAME                          CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
svc/docker-registry           172.30.243.63   <none>        5000/TCP
10m
svc/kubernetes                172.30.0.1      <none>
443/TCP,53/UDP,53/TCP     20m
svc/router                    172.30.224.41   <none>
80/TCP,443/TCP,1936/TCP   10m
NAME                          READY           STATUS        RESTARTS
AGE
po/docker-registry-1-2a1ho    1/1             Running       0
8m
po/docker-registry-1-krpix    1/1             Running       0
8m
po/router-1-1g84e             1/1             Running       0
8m
po/router-1-t84cy             1/1             Running       0
8m
```

Observe the output of **`oc get all`**

4.9.3. Explore the Registry

The OpenShift Ansible playbooks configure three infrastructure nodes that have three registries running. In order to understand the configuration and mapping process of the registry pods, the command 'oc describe' is used. **`oc describe`** details how registries are configured and mapped to the Amazon **S3** buckets for storage. Using **`oc describe`** should help explain how HA works in this environment.

Note

If the OpenShift account configured on the workstation has cluster-admin privileges perform the following. If the account does not have this privilege ssh to one of the OpenShift masters and perform the steps.

```
$ oc describe svc/docker-registry
Name:   docker-registry
Namespace:  default
Labels:   docker-registry=default
Selector:  docker-registry=default
Type:   ClusterIP
IP:   172.30.110.31
Port:   5000-tcp 5000/TCP
Endpoints:  172.16.4.2:5000,172.16.4.3:5000
Session Affinity: ClientIP
No events.
```

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

The **oc** client allows similar functionality to the **docker** command. To find out more information about the registry storage perform the following.

```
# 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
docker-registry-2-2s2ca   1/1       Running   0          2h
```

```
# oc exec docker-registry-2-8b7c6 cat /etc/registry/config.yml
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    layerinfo: inmemory
  s3:
    accesskey: "AKIAJZO3LDPPKZFORUQQ"
    secretkey: "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"
    region: us-east-1
    bucket: "1476274760-openshift-docker-registry"
    encrypt: true
    secure: true
    v4auth: true
    rootdirectory: /registry
auth:
  openshift:
    realm: openshift
middleware:
  repository:
    - name: openshift
```

Observe the **S3** stanza. Confirm the bucket name is listed, and access the**AWS** console. Click on the **S3 AWS** and locate the bucket. The bucket should contain content. Confirm that the same bucket is mounted to the other registry via the same steps.

4.9.4. Explore Docker Storage

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

The example below can be performed on any node but for this example the infrastructure node(ose-infra-node01.sysdeseng.com) is used.

The output below describing the **`Storage Driver: docker— vol-docker— pool`** states that docker storage is not using a loop back device.

```
$ docker info
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 4
Server Version: 1.10.3
Storage Driver: devicemapper
 Pool Name: docker--vol-docker--pool
 Pool Blocksize: 524.3 kB
 Base Device Size: 3.221 GB
 Backing Filesystem: xfs
 Data file:
 Metadata file:
 Data Space Used: 1.221 GB
 Data Space Total: 25.5 GB
 Data Space Available: 24.28 GB
 Metadata Space Used: 307.2 kB
 Metadata Space Total: 29.36 MB
 Metadata Space Available: 29.05 MB
 Udev Sync Supported: true
 Deferred Removal Enabled: true
 Deferred Deletion Enabled: true
 Deferred Deleted Device Count: 0
 Library Version: 1.02.107-RHEL7 (2016-06-09)
Execution Driver: native-0.2
Logging Driver: json-file
Plugins:
 Volume: local
 Network: bridge null host
 Authorization: rhel-push-plugin
Kernel Version: 3.10.0-327.10.1.el7.x86_64
Operating System: Employee SKU
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 2
CPUs: 2
Total Memory: 7.389 GiB
Name: ip-10-20-3-46.ec2.internal
ID: XDCD:7NAA:N2S5:AMYW:EF33:P2WM:NF5M:XOLN:JHAD:SIHC:IZXP:MOT3
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Registries: registry.access.redhat.com (secure), docker.io (secure)
```

Verify 3 disks are attached to the instance. The disk **`/dev/xvda`** is used for the OS, **`/dev/xvdb`** is used for docker storage, and **`/dev/xvdc`** is used for emptyDir storage for containers that do not use a persistent volume.

```
$ fdisk -l
WARNING: fdisk GPT support is currently new, and therefore in an
experimental phase. Use at your own discretion.

Disk /dev/xvda: 26.8 GB, 26843545600 bytes, 52428800 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: gpt


#         Start          End    Size  Type            Name
 1         2048         4095      1M  BIOS boot parti
 2         4096     52428766     25G  Microsoft basic

Disk /dev/xvdc: 53.7 GB, 53687091200 bytes, 104857600 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/xvdb: 26.8 GB, 26843545600 bytes, 52428800 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/xvdb1            2048    52428799    26213376   8e  Linux LVM

Disk /dev/mapper/docker--vol-docker--pool_tmeta: 29 MB, 29360128 bytes,
57344 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/docker--vol-docker--pool_tdata: 25.5 GB, 25497174016
bytes, 49799168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes


Disk /dev/mapper/docker--vol-docker--pool: 25.5 GB, 25497174016 bytes,
49799168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 131072 bytes / 524288 bytes


Disk /dev/mapper/docker-202:2-75507787-
4a813770697f04b1a4e8f5cdaf29ff52073ea66b72a2fbe2546c469b479da9b5: 3221
MB, 3221225472 bytes, 6291456 sectors
Units = sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 131072 bytes / 524288 bytes


Disk /dev/mapper/docker-202:2-75507787-
260bda602f4e740451c428af19bfec870a47270f446ddf7cb427eee52caafdf6: 3221
MB, 3221225472 bytes, 6291456 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 131072 bytes / 524288 bytes
```

4.9.5. Explore Security Groups

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

Note

Perform the following steps from the **AWS** web console.

On the main **AWS** console, click on **EC2**. Next on the left hand navigation panel select the **Security Groups**. Click through each group and check out both the **Inbound** and **Outbound** rules that were created as part of the infrastructure provisioning. For example, notice how the **Bastion** security group only allows **SSH** traffic inbound. That can be further restricted to a specific network or host if required. Next take a look at the **Master** security group and explore all the **Inbound** and **Outbound** TCP and UDP rules and the networks from which traffic is allowed.

4.9.6. Explore the AWS Elastic Load Balancers

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

Note

Perform the following steps from the **AWS** web console.

On the main **AWS** console, click on **EC2**. Next on the left hand navigation panel select the **Load Balancers**. Select the **ose-master** load balancer and on the **Description** page note the **Port Configuration** and how it is configured for port 443. That is for the OpenShift web console traffic. On the same tab, check the **Availability Zones**, note how those are **Public** subnets. Move to the **Instances** tab. There should be three master instances running with a **Status** of **InService**. Next check the **Health Check** tab and the options that were configured. Further details of the configuration can be viewed by exploring the Ansible playbooks to see exactly what was configured. Finally, change to the **ose-internal-master** and compare the subnets. The subnets for the **ose-internal-master** are all private. They are private because that **ELB** is reserved for traffic coming from the OpenShift infrastructure to the master servers. This results in reduced charges from Amazon because the packets do not have to be processed by the public facing **ELB**.

4.9.7. Explore the AWS VPC

As mentioned earlier in the document a Virtual Private Cloud was created. The purpose of this section is to encourage exploration of the **VPC** that was created.

Note

Perform the following steps from the **AWS** web console.

On the main Amazon Web Services console, click on **VPC**. Next on the left hand navigation panel select the **Your VPCs**. Select the **VPC** recently created and explore the **Summary** and **Tags** tabs. Next, on the left hand navigation panel, explore the **Subnets**, **Route Tables**, **Internet Gateways**, **DHCP Options Sets**, **NAT Gateways**, **Security Groups** and **Network ACLs**. More detail can be looked at with the configuration by exploring the Ansible playbooks to see exactly what was configured.

4.10. TESTING FAILURE

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

4.10.1. Generate a Master Outage

Note

Perform the following steps from the **AWS** web console and the OpenShift public URL.

Log into the **AWS** console. On the dashboard, click on the **EC2** web service and then click **Instances**. Locate your running ose-master02.sysdeseng.com instance, select it, right click and change the state to **stopped**.

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

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

SSH into the first master node (ose-master01.sysdeseng.com). Using the output of the command **hostname** issue the **etcdctl** command to confirm that the cluster is healthy.

```
$ ssh ec2-user@ose-master01.sysdeseng.com
$ sudo -i
```

```
# hostname
ip-10-20-1-106.ec2.internal
# etcdctl -C https://ip-10-20-1-106.ec2.internal:2379 --ca-file
/etc/etcd/ca.crt --cert-file=/etc/origin/master/master.etcd-client.crt
--key-file=/etc/origin/master/master.etcd-client.key cluster-health
failed to check the health of member 82c895b7b0de4330 on
https://10.20.2.251:2379: Get https://10.20.1.251:2379/health: dial tcp
10.20.1.251:2379: i/o timeout
```

```
member 82c895b7b0de4330 is unreachable: [https://10.20.1.251:2379] are
all unreachable
member c8e7ac98bb93fe8c is healthy: got healthy result from
https://10.20.3.74:2379
member f7bbfc4285f239ba is healthy: got healthy result from
https://10.20.1.106:2379
cluster is healthy
```

Notice how one member of the **ETCD** cluster is now unreachable. Restart ose-master02.sysdeseng.com by following the same steps in the **AWS** web console as noted above.

4.10.3. Generate an Infrastructure Node outage

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

4.10.3.1. Confirm 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 the 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 including 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://openshift-
master.sysdeseng.com".

$ oc status
In project test-app1 on server https://openshift-master.sysdeseng.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

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

-

Open a browser and ensure the application is still accessible.

4.10.3.2. Confirm 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 OpenShift registry.

Note

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

A token is needed so that the 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 status
In project default on server https://internal-openshift-
master.sysdeseng.com:443

https://docker-registry-default.apps.sysdeseng.com (passthrough)
(svc/docker-registry)
  dc/docker-registry deploys docker.io/openshift3/ose-docker-
registry:v3.5.5.5
    deployment #1 deployed 44 minutes ago - 3 pods

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

https://registry-console-default.apps.sysdeseng.com (passthrough)
(svc/registry-console)
  dc/registry-console deploys
registry.access.redhat.com/openshift3/registry-console:3.5
    deployment #1 deployed 43 minutes ago - 1 pod

svc/router - 172.30.41.42 ports 80, 443, 1936
  dc/router deploys docker.io/openshift3/ose-haproxy-router:v3.5.5.5
    deployment #1 deployed 45 minutes ago - 3 pods

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

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

```
# docker tag docker.io/fedora/apache
172.30.110.31:5000/openshift/prodapache
```

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

```
# docker images
```

Issue a Docker login.

```
# docker login -u sysdesadmin -e sysdesadmin -p
_7yJcnXfeRtAbJVEaQwPwXreEhlV56TkgDwZ6UEUDWw 172.30.110.31:5000
```

```
# oadm policy add-role-to-user admin sysdesadmin -n openshift
# oadm policy add-role-to-user system:registry sysdesadmin
# oadm policy add-role-to-user system:image-builder sysdesadmin
```

Push the image to the OpenShift registry now.

```
# docker push 172.30.110.222:5000/openshift/prodapache
The push refers to a repository
[172.30.110.222:5000/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
```

4.10.3.3. 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://openshift-
master.sysdeseng.com".

$ oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-2-gmvdr    1/1       Running   1          21h
docker-registry-2-jueep    1/1       Running   0          7h
router-1-6y5td             1/1       Running   1          21h
router-1-rlcwj             1/1       Running   1          21h

$ oc describe pod docker-registry-2-jueep | grep -i node
Node:  ip-10-30-1-17.ec2.internal/10.30.1.17
$ oc describe pod docker-registry-2-gmvdr | grep -i node
```

```
Node:  ip-10-30-2-208.ec2.internal/10.30.2.208
$ oc describe pod router-1-6y5td | grep -i node
Node:  ip-10-30-1-17.ec2.internal/10.30.1.17
$ oc describe pod router-1-rlcwj | grep -i node
Node:  ip-10-30-2-208.ec2.internal/10.30.2.208
```

4.10.3.4. Initiate the Failure and Confirm Functionality

Note

Perform the following steps from the **AWS** web console and a browser.

Log into the **AWS** console. On the dashboard, click on the **EC2** web service. Locate your running infra01 instance, select it, right click and change the state to **stopped**. Wait a minute or two for the registry and pod to migrate over to infra01. Check the registry locations and confirm that they are on the same node.

```
$ oc describe pod docker-registry-2-fw1et | grep -i node
Node:  ip-10-30-2-208.ec2.internal/10.30.2.208
$ oc describe pod docker-registry-2-gmvdr | grep -i node
Node:  ip-10-30-2-208.ec2.internal/10.30.2.208
```

Follow the procedures above to ensure an image can still be pushed to the registry now that infra01 is down.

4.11. UPDATING THE OPENSHIFT DEPLOYMENT

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

4.11.1. Performing the Upgrade

From the workstation that was used to perform the installation of **OpenShift on AWS** 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.

Note

Ensure the variables below are relevant to the deployed OpenShift environment. The variables that should be customerized for the deployed OpenShift environment are stack_name, public_hosted_zone, console_port, region, and containerized.

4.11.1.1. Non-Containerized Upgrade

Use the following lines below to perform the upgrade in a non-containerized environment.

```
$ yum update atomic-openshift-utils ansible
$ cd ~/git/openshift-ansible-contrib/reference-architecture/aws-ansible
$ ansible-playbook -i inventory/aws/hosts -e 'stack_name=openshift-
infra public_hosted_zone=sysdeseng.com console_port=443 region=us-east-
```

```
1' playbooks/openshift-minor-upgrade.yaml
```

4.11.1.2. Containerized Upgrade

Use the following lines below to perform the upgrade in a containerized environment.

```
$ yum update atomic-openshift-utils ansible
$ cd ~/git/openshift-ansible-contrib/reference-architecture/aws-ansible
$ ansible-playbook -i inventory/aws/hosts -e 'stack_name=openshift-
infra public_hosted_zone=sysdeseng.com console_port=443 region=us-east-
1 containerized=true' playbooks/openshift-minor-upgrade.yaml
```

4.11.2. Upgrading and Restarting the OpenShift Environment (Opitonal)

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
```

4.11.3. 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 3.3 non-containerized environment.

```
$ yum update atomic-openshift-utils ansible
$ cd ~/git/openshift-ansible-contrib/reference-architecture/aws-ansible
$ ansible-playbook -i inventory/aws/hosts -e 'stack_name=openshift-
infra public_hosted_zone=sysdeseng.com console_port=443 region=us-east-
1 openshift_vers=v3_4' playbooks/openshift-minor-upgrade.yaml
```

CHAPTER 5. PERSISTENT STORAGE

Container storage by default is not persistent. For example, if a new container build occurs then data is lost because the storage is non-persistent. If a container terminates then of the all changes to it's 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 at https://docs.openshift.com/container-platform/3.5/architecture/additional_concepts/storage.html#types-of-persistent-volumes. 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 cloud provider specific storage, Container-Native Storage (CNS), and Container-Ready Storage(CRS).

5.1. PERSISTENT VOLUMES

Container storage is defined by the concept of **`persistent volumes`** (pv) which are OpenShift objects that allow for storage to be defined and then used by pods to allow for data persistence. Requesting of **`persistent volumes`** 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`**.

5.2. 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`** or **`st1`** vs **`gp2`**).

5.3. CLOUD PROVIDER SPECIFIC STORAGE

Cloud provider specific storage is storage that is provided from **`AWS`**. This type of storage is presented as an **`EBS`** volume and can be mounted by one pod at a time. The **`EBS`** volume must exist in the same availability zone as the pod that requires the storage. This is because **`EBS`** volumes cannot be mounted by an **`EC2`** instance outside of the availability zone that it was created. For **`AWS`** there are 4 types of persistent disks that can be utilized http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSVolumeTypes.html [io1, gp2, sc1, and st1] for cloud provider specific storage. Cloud provider storage can be created manually and assigned as a persistent volume or a persistent volume can be created dynamically using a **`StorageClass`** object. Note that **`EBS`** storage can only use the access mode of Read-Write-Once (RWO).

5.3.1. Creating a Storage Class

When requesting cloud provider specific storage the name, zone, and type are configurable items. Ensure that the zone configuration parameter specified when creating the **`StorageClass`** object is an **`AZ`** that currently hosts application node instances. This will ensure that **`PVCs`** will be created in

the correct **AZ** in which containers are running. The **cluster-admin** or **storage-admin** can perform the following commands which will allow for dynamically provisioned gp2 storage on demand, in the us-east-1a AZ, and using the name "standard" as the name of the **StorageClass** object.

```
$ vi standard-storage-class.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  zone: us-east-1a
```

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

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

Multiple **StorageClassess** objects can be defined depending on the storage needs of the pods within OpenShift.

5.3.2. Creating and using a Persistent Volumes Claim

The example below shows a dynamically provisioned volume being requested from the **StorageClass** named standard.

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

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

5.4. CONTAINER-NATIVE STORAGE OVERVIEW

Container-Native Storage (CNS) provides dynamically provisioned storage for containers on OpenShift across cloud providers, virtual and bare-metal deployments. **CNS** relies on **EBS** volumes mounted on the OpenShift nodes and uses software-defined storage provided by Red Hat Gluster Storage. **CNS** runs Red Hat Gluster Storage containerized allowing OpenShift storage pods to

spread across the cluster and across **Availabilty Zones**. **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 OpenShift registry.

5.4.1. 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 CNS storage are required 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 **EC2** instance. There is a current limit of 300 volumes or **PVs** per 3 node **CNS** cluster. The **CNS EC2** instance type may need to be increased to support 300 volumes.

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. For nodes which carry both the app and the storage label the **EC2** instance type of **m4.2xlarge** is a conscious choice that provides balance between enough memory requirements in the adoption phase and immediate **EC2** instance cost. 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. Over time the amount of apps and **PVs** will grow and the EC2 instance type choice must be re-evaluated, or more app-only nodes need to be added.

5.4.2. Deployment of CNS Infrastructure

A python script named **add-cns-storage.py** is provided in the **openshift-ansible-contrib** git repository which will deploy three nodes, add the nodes to the OpenShift environment with specific OpenShift labels and attach an **EBS** volume to each node as an available block device to be used for **CNS**. Do the following from the workstation performing the deployment of the OpenShift Reference Architecture.

Note

On deployments of the Reference Architecture environment post OpenShift 3.5 --use-cloudformation-facts is available to auto-populate values. An example of these values can be viewed in the **Post Provisioning Results** section of this Reference Architecture. If the deployment occurred before version 3.5 then it is required to fill in the values manually. To view the possible configuration triggers run **add-cns-storage.py -h**. The **add-cns-storage.py** script requires the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY exported as an environment variable.

If the Reference Architecture deployment version is >= 3.5. Use the deployment option **--use-cloudformation-facts** to auto-populate some values based on the existing **Cloudformations**

stack.

```
$ cd /home/<user>/git/openshift-ansible-contrib/reference-
architecture/aws-ansible/
./add-cns-storage.py --rhsm-user=username --rhsm-password=password --
public-hosted-zone=sysdeseng.com --region=us-east-1 --gluster-stack=cns
\ --rhsm-pool="Red Hat OpenShift Container Platform, Premium, 2-Core" -
-keypair=OSE-key --existing-stack=openshift-infra --use-cloudformation-
facts
```

If the Reference Architecture deployment was performed before OpenShift version 3.5. Fill in the values to represent the existing **`Cloudformations`** stack.

```
$ cd /home/<user>/git/openshift-ansible-contrib/reference-
architecture/aws-ansible/
./add-cns-storage.py --rhsm-user=username --rhsm-password=PASSWORD --
public-hosted-zone=sysdeseng.com --gluster-stack=cns \ --rhsm-pool="Red
Hat OpenShift Container Platform, Premium, 2-Core" --keypair=OSE-key --
existing-stack=openshift-infra \ --private-subnet-id1=subnet-ad2b23f6 -
-private-subnet-id2=subnet-7cd61a34 --region=us-east-1 \ --private-
subnet-id3=subnet-77e89a4b --node-sg=sg-0c7e0f73 \ --iam-role=backup-
NodeInstanceProfile-AX9H0AOAINY3
```

Note

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

5.4.3. Firewall and Security Group Prerequisites

The following ports must be open on the **`CNS`** nodes and on the node security group in **`AWS`**. Ensure the following ports defined in the table below are open. This configuration is automatically applied on the nodes deployed using the **`add-cns-storage.py`** script.

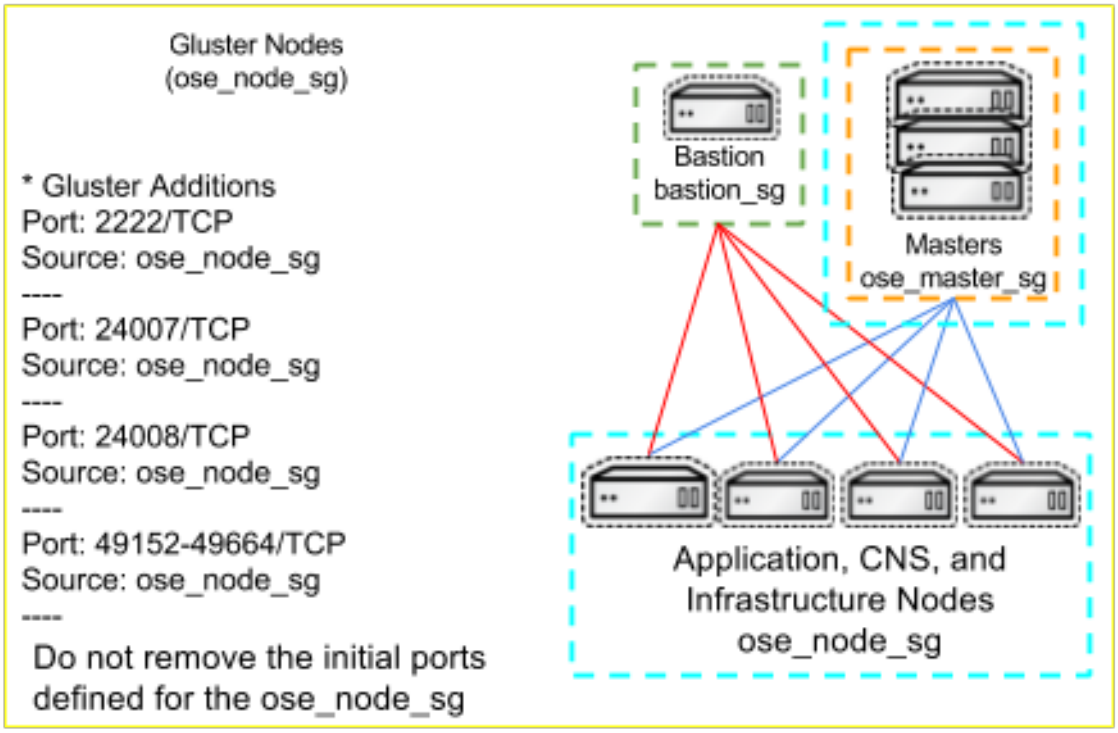


Table 5.1. AWS Nodes Security Group Details - Inbound

Inbound	From
22 / TCP	bastion_sg
2222 / TCP	ose_node_sg
4789 / UDP	ose_node_sg
10250 / TCP	ose_master_sg
10250 / TCP	ose_node_sg
24007 / TCP	ose_node_sg
24008 / TCP	ose_node_sg
49152-49664 / TCP	ose_node_sg

5.5. CNS INSTALLATION OVERVIEW

The process for creating a **CNS** deployment on OpenShift starts with creating an OpenShift 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 consists of a Red Hat Gluster Storage single container pods running on OpenShift 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 an OpenShift node.

5.5.1. 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.

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.

```
# oadm 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.

```
# oadm new-project storage
```

An **oadm** 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
# oadm policy add-scc-to-user privileged -z default
```

5.5.2. 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
```

5.5.3. 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. Below a table shows the different instances running in different **AZs** to distinguish failure domains defined as a zone in **heketi**. This information will be used to make intelligent decisions about how to structure a volume that is, to create Gluster volume layouts that have no more than one brick or storage device from a single failure domain. This information will also be used when healing degraded volumes in the event of a loss of device or an entire node.

Table 5.2. CNS Topology file

Instance	AZ	Zone	Device
ip-10-20-4-163.ec2.internal	us-east-1a	1	/dev/xvdd
ip-10-20-5-247.ec2.internal	us-east-1c	2	/dev/xvdd
ip-10-20-6-191.ec2.internal	us-east-1d	3	/dev/xvdd

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 in the **us-east-1** region with **/dev/xvdd** as the **EBS** volume or device used for **CNS**. Edit the values of node.hostnames.manage, node.hostnames.storage, and devices in the **topology.json** file based on the the OpenShift nodes that have been deployed in the previous step.

```
# vi gluster-topology.json
{
    "clusters": [
        {
            "nodes": [
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ip-10-20-4-163.ec2.internal"
```

```
                            ],
                            "storage": [
                                "10.20.4.163"
                            ]
                        },
                        "zone": 1
                    },
                    "devices": [
                        "/dev/xvdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ip-10-20-5-247.ec2.internal"
                            ],
                            "storage": [
                                "10.20.5.247"
                            ]
                        },
                        "zone": 2
                    },
                    "devices": [
                        "/dev/xvdd"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ip-10-20-6-191.ec2.internal"
                            ],
                            "storage": [
                                "10.20.6.191"
                            ]
                        },
                        "zone": 3
                    },
                    "devices": [
                        "/dev/xvdd"
                    ]
                }
            ]
        }
    ]
}
```

Ensure that the storage project is the current project.

```
# oc project storage
Already on project "storage" on server "https://openshift-
master.sysdeseng.com"
```

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 OpenShift. 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 gluster-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    9m
Using namespace "storage".
template "deploy-heketi" created
serviceaccount "heketi-service-account" created
template "heketi" created
template "glusterfs" created
node "ip-10-20-4-163.ec2.internal" labeled
node "ip-10-20-5-247.ec2.internal" labeled
node "ip-10-20-6-191.ec2.internal" 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 ... OK
  % Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
                                 Dload  Upload   Total   Spent
Left  Speed
100    17  100    17    0     0      3      0  0:00:05  0:00:05 --:--
:--     4
Creating cluster ... ID: 372cf750e0b256fbc8565bb7e4afb434
 Creating node ip-10-20-4-163.ec2.internal ... ID:
9683e22a0f98f8c40ed5c3508b2b4a38
  Adding device /dev/xvdd ... OK
 Creating node ip-10-20-5-247.ec2.internal ... ID:
b9bb8fc7be62de3152b9164a7cb3a231
  Adding device /dev/xvdd ... OK
 Creating node ip-10-20-6-191.ec2.internal ... ID:
790bff20ac0115584b5cd8225565b868
  Adding device /dev/xvdd ... OK
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
secret "heketi-storage-secret" deleted
service "heketi" created
route "heketi" created
deploymentconfig "heketi" created
Waiting for heketi pod to start ... OK
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
NAME              READY     STATUS    RESTARTS   AGE
glusterfs-7gr8y   1/1       Running   0          4m
glusterfs-fhy3e   1/1       Running   0          4m
glusterfs-ouay0   1/1       Running   0          4m
heketi-1-twis5    1/1       Running   0          1m
```

5.5.4. Exploring Heketi

A 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
heketi    heketi-storage.apps.sysdeseng.com   heketi     <all>
# export HEKETI_CLI_SERVER=http://heketi-storage.apps.sysdeseng.com
# 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:
372cf750e0b256fbc8565bb7e4afb434
```

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

```
# heketi-cli cluster info 372cf750e0b256fbc8565bb7e4afb434
```

5.5.5. Store the Heketi Secret

OpenShift 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. OpenShift 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
```

5.5.6. Creating a Storage Class

The **StorageClass** object created using the **CNS** components is a more robust solution than using cloud provider specific storage due to the fact that the storage is not dependant on **AZs**. 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 EBS volumes attached to achieve the desired properties of a tier (e.g. io1 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
heketi    heketi-storage.apps.sysdeseng.com              heketi
<all>
# export HEKETI_CLI_SERVER=http://heketi-storage.apps.sysdeseng.com
# 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:
 eb08054c3d42c88f0924fc6a57811610
```

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-st1.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: http://heketi-storage.apps.sysdeseng.com
```

```
  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-st1.yaml
```

OpenShift 3.5

```
# vi glusterfs-storageclass-st1.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-slow
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: http://heketi-storage.apps.sysdeseng.com
  clusterid: eb08054c3d42c88f0924fc6a57811610
  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-st1.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=e73da525319cbf784ed27df3e8715ea8,restauthenabled=true,resturl
=http://heketi-
storage.apps.sysdeseng.com,restuser=admin,secretName=heketi-
secret,secretNamespace=default
No events.
```

5.5.7. 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 test
```

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

$ vi db-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 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" created
```

5.6. ADDITIONAL CNS STORAGE DEPLOYMENTS (OPTIONAL)

An OpenShift 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 objects name.

Note

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

5.6.1. Deployment of a second Gluster Storage Pool

To deploy an additional **glusterfs** pool OpenShift 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-cns-storage.py** script or by manually deploying three instances and installing and configuring those nodes for OpenShift.

Note

If running the **add-cns-storage.py** nodes a second time provide a unique value for configuration parameter of **--gluster-stack**. If the value of **--gluster-stack** is the same for the old environment and the new then the existing **CNS** deployment will be replaced.

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
NAME                           STATUS                     AGE
...omitted...
ip-10-20-4-189.ec2.internal   Ready                      5m
ip-10-20-5-204.ec2.internal   Ready                      5m
ip-10-20-6-39.ec2.internal    Ready                      5m
...omitted...

# oc label node ip-10-20-4-189.ec2.internal storagenode=glusterfs
# oc label node ip-10-20-5-204.ec2.internal storagenode=glusterfs
# oc label node ip-10-20-6-39.ec2.internal 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.

```
[subs=+quotes]
# *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.

5.6.2. 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 call ...
        }
      ]
    },
```

```
    {
      "nodes": [
        {
          "node": {
            "hostnames": {
              "manage": [
                "ip-10-20-4-189.ec2.internal"
              ],
              "storage": [
                "10.20.4.189"
              ]
            },
            "zone": 1
          },
          "devices": [
            "/dev/xvdd"
          ]
        },
        {
          "node": {
            "hostnames": {
              "manage": [
                "ip-10-20-5-204.ec2.internal"
              ],
              "storage": [
                "10.20.5.204"
              ]
            },
            "zone": 2
          },
          "devices": [
            "/dev/xvdd"
          ]
        },
        {
          "node": {
            "hostnames": {
              "manage": [
                "ip-10-20-6-39.ec2.internal"
              ],
              "storage": [
                "10.20.6.39"
              ]
            },
            "zone": 3
          },
          "devices": [
            "/dev/xvdd"
          ]
        }
      ]
    }
  ]
}
```

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.sysdeseng.com
# export HEKETI_CLI_USER=admin
# export HEKETI_CLI_KEY=myS3cr3tpassw0rd
```

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

```
# heketi-cli topology load --json=gluster-topology.json
 Found node ip-10-20-4-163.ec2.internal on cluster
372cf750e0b256fbc8565bb7e4afb434
  Found device /dev/xvdd
 Found node ip-10-20-5-247.ec2.internal on cluster
372cf750e0b256fbc8565bb7e4afb434
  Found device /dev/xvdd
 Found node ip-10-20-6-191.ec2.internal on cluster
372cf750e0b256fbc8565bb7e4afb434
  Found device /dev/xvdd
Creating cluster ... ID: 269bb26142a15ee10fa8b1cdeb0a37b7
 Creating node ip-10-20-4-189.ec2.internal ... ID:
0bd56937ef5e5689e003f68a7fde7c69
  Adding device /dev/xvdd ... OK
 Creating node ip-10-20-5-204.ec2.internal ... ID:
f79524a4de9b799524c87a4feb41545a
  Adding device /dev/xvdd ... OK
 Creating node ip-10-20-6-39.ec2.internal ... ID:
4ee40aee71b60b0627fb57be3dd2c66e
  Adding device /dev/xvdd ... 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 **EC2** nodes are in the new cluster.

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

5.6.3. 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-gp2.yaml
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: gluster-cns-fast
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: http://heketi-storage.apps.sysdeseng.com
  clusterid: 269bb26142a15ee10fa8b1cdeb0a37b7
```

```
  restauthenabled: "true"
  restuser: "admin"
  secretNamespace: "default"
  secretName: "heketi-secret"
```

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

```
# oc create -f glusterfs-storageclass-gp2.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
 annotations:
   volume.beta.kubernetes.io/storage-class: gluster-cns-fast
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi
```

5.7. CONTAINER-READY STORAGE OVERVIEW

Container-Ready Storage (CRS) like **CNS**, uses Red Hat Gluster Storage to provide dynamically provisioned storage. Unlike **CNS** where OpenShift deploys **glusterfs** and **heketi** specific pods to be used for OpenShift storage **CRS** requires an Administrator to install packages and enable the storage services on EC2 instances. 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**.

5.7.1. Prerequisites for Container-Ready Storage

Deployment of Container-Ready Storage (CRS) requires at least 3 **AWS** instances with at least one unused block storage device or **EBS** volume on each node. The instances should have at least 4 CPUs, 32GB RAM, and an unused volume 100GB or larger per node. An entitlement for **Red Hat Gluster Storage** is also required to install the **Gluster** services.

5.7.2. Deployment of CRS Infrastructure

A python script named **add-crs-storage.py** is provided in the **openshift-ansible-contrib** git repository which will deploy three **AWS** instances, register the instances, and install the prerequisites for **CRS** for Gluster on each instance. Perform the following from the workstation where the deployment of the OpenShift Reference Architecture was initiated.

Note

On deployments of the Reference Architecture environment post OpenShift 3.5 **`--use-cloudformation-facts`** is available to auto-populate values. An example of these values can be viewed in the Post Provisioning Results section of this Reference Architecture. If the deployment occurred before 3.5 then it is required to fill in the values manually. To view the possible configuration triggers run **`add-crs-storage.py -h`**. The **`add-crs-storage.py`** script requires the **`AWS_ACCESS_KEY_ID`** and **`AWS_SECRET_ACCESS_KEY`** exported as an environment variable.

If the Reference Architecture deployment is >= OpenShift 3.5

```
$ cd /home/<user>/git/openshift-ansible-contrib/reference-
architecture/aws-ansible/
./add-crs-storage.py --rhsm-user=username --rhsm-password=password --
region=us-east-1 --gluster-stack=crs --public-hosted-zone=sysdeseng.com
\ --rhsm-pool="Red Hat Gluster Storage , Standard" --keypair=OSE-key --
existing-stack=openshift-infra --use-cloudformation-facts
```

If the Reference Architecture deployment was performed before 3.5.

```
$ cd /home/<user>/git/openshift-ansible-contrib/reference-
architecture/aws-ansible/
./add-crs-storage.py --rhsm-user=username --rhsm-password=PASSWORD --
public-hosted-zone=sysdeseng.com \ --rhsm-pool="Red Hat Gluster Storage
, Standard" --keypair=OSE-key --existing-stack=openshift-infra \ --
private-subnet-id1=subnet-ad2b23f6 --private-subnet-id2=subnet-7cd61a34
--region=us-east-1 \ --private-subnet-id3=subnet-77e89a4b --node-sg=sg-
0c7e0f73 -- bastion-sg=sg-1a2b5a23 --gluster-stack=crs
```

Note

Using the script **`add-crs-storage.py`** is optional. Nodes can be deployed without using this script as long as the 3 new instances have 4 CPUs, 32GB RAM, and an unused storage device or **EBS** volume.

5.7.3. 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.

If the **`add-crs-storage.py`** script was not used perform the following on the 3 **CRS** instances to enable the required repository. Ensure the pool that is specified matches a pool available to the **RHSM** credentials provided (example pool ID shown below).

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

5.7.4. Firewall and Security Group Prerequisites

The following ports must be opened on the **CRS** nodes and in the **AWS** **`gluster-crs-sg`** security group. Ensure the following ports defined in the table below are opened. Iptables or firewalld can be used depending on the preference of the Administrator. These steps are done as part of the automated provisioning of the instances with the **`add-crs-storage.py`**.

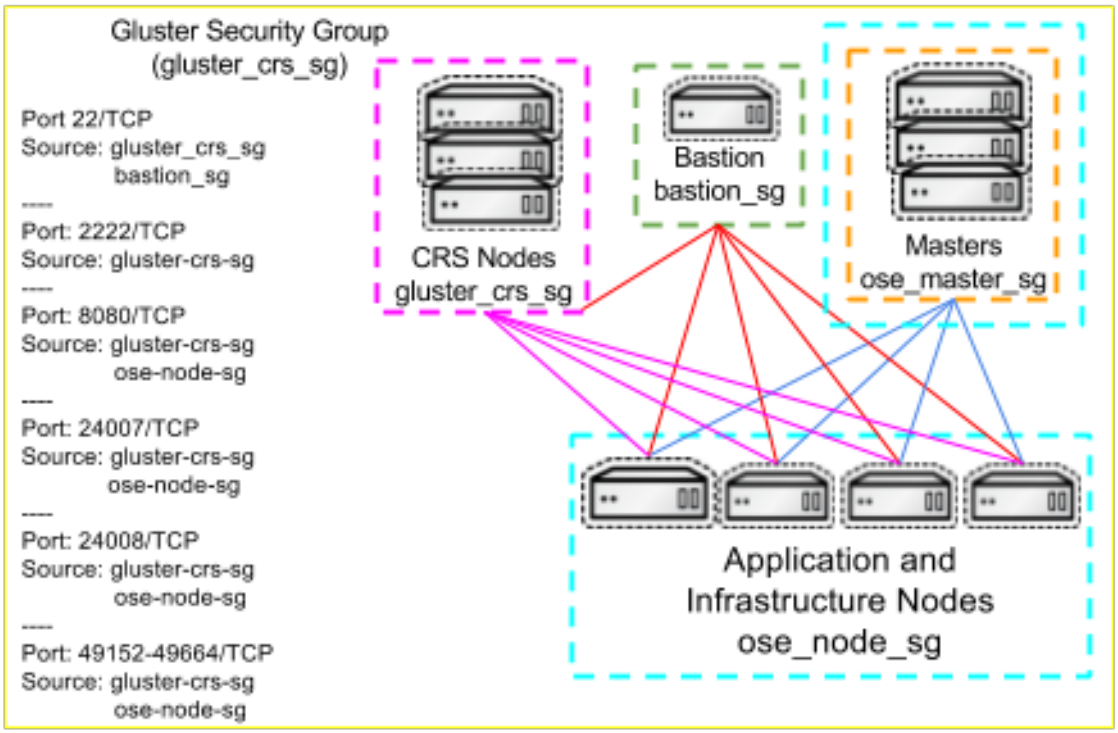


Table 5.3. AWS Nodes Security Group Details - Inbound

Inbound	From
22 / TCP	bastion_sg
22 / TCP	gluster-crs-sg
2222 / TCP	gluster-crs-sg
8080 / TCP	gluster-crs-sg
8080 / TCP	ose-node-sg
24007 / TCP	gluster-crs-sg

Inbound	From
24007 / TCP	ose-node-sg
24008 / TCP	gluster-crs-sg
24008 / TCP	ose-node-sg
49152-49664 / TCP	gluster-crs-sg
49152-49664 / TCP	ose-node-sg

The **add-crs-gluster.py** uses iptables and creates the rules shown in the table above on each of the **CRS** nodes. The following commands can be ran on the 3 new instances if the instances were built without using the **add-crs-gluster.py** 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
```

5.7.5. 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. If the **add-crs-storage.py** script was not used perform the following on the each of the three **CRS** instances.

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

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

```
# systemctl enable glusterd
# systemctl start glusterd
```

5.7.6. 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**. **Heketi** can be installed on one of the **CRS** instances or even within OpenShift if desired. Regardless of whether

the **add-crs-storage.py** script was used or not the following must be performed on the **CRS** instance chosen to run **Heketi**. For the steps below the first **CRS Gluster** instance will be used.

```
# 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 **/home/ec2-user/.ssh/authorized_keys** file. This must be done on all 3 instances including the **CRS** node where the **heketi** services are running. Also, on each of the 3 instances **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 OpenShift. 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": "ec2-user",
      "sudo": true,
      "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 **OpenShift** master or from any of the **CRS** instances.

```
# curl http://ip-10-20-4-40.ec2.internal:8080/hello
Hello from Heketi
```

5.7.7. 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 in 3 zones. Both **CRS** and **CNS** use the same format for the **topology.json** file.

```
# vi topology.json
{
    "clusters": [
        {
            "nodes": [
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ip-10-20-4-40.ec2.internal"
                            ],
                            "storage": [
                                "10.20.4.40"
                            ]
                        },
                        "zone": 1
                    },
                    "devices": [
                        "/dev/xvdb"
                    ]
                },
                {
                    "node": {
```

```
                        "hostnames": {
                            "manage": [
                                "ip-10-20-5-104.ec2.internal"
                            ],
                            "storage": [
                                "10.20.5.104"
                            ]
                        },
                        "zone": 2
                    },
                    "devices": [
                        "/dev/xvdb"
                    ]
                },
                {
                    "node": {
                        "hostnames": {
                            "manage": [
                                "ip-10-20-6-79.ec2.internal"
                            ],
                            "storage": [
                                "10.20.6.79"
                            ]
                        },
                        "zone": 3
                    },
                    "devices": [
                        "/dev/xvdb"
                    ]
                }
            ]
        }
    ]
}
```

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 **AWS** instance 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://ip-10-20-4-40.ec2.internal: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
 Found node ip-10-20-4-40.ec2.internal on cluster
c21779dd2a6fb2d665f3a5b025252849
  Adding device /dev/xvdb ... OK
 Creating node ip-10-20-5-104.ec2.internal ... ID:
53f9e1af44cd5471dd40f3349b00b1ed
```

```
  Adding device /dev/xvdb ... OK
 Creating node ip-10-20-6-79.ec2.internal ... ID:
328dfe7fab00a989909f6f46303f561c
  Adding device /dev/xvdb ... OK
```

5.7.8. Validating Gluster Installation(Optional)

From the instance 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_4950679f18b9fad6f118b2b20b0c727e
Size: 50
Volume Id: 4950679f18b9fad6f118b2b20b0c727e
Cluster Id: b708294a5a2b9fed2430af9640e7cae7
Mount: 10.20.4.119:vol_4950679f18b9fad6f118b2b20b0c727e
Mount Options: backup-volfile-servers=10.20.5.14,10.20.6.132
Durability Type: replicate
Distributed+Replica: 3
```

The command **gluster volume info** can provide further information on the newly created **Gluster** volume.

```
# gluster volume info

Volume Name: vol_f2eec68b2dea1e6c6725d1ca3f9847a4
Type: Replicate
Volume ID: f001d6e9-fee4-4e28-9908-359bbd28b8f5
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 3 = 3
Transport-type: tcp
Bricks:
Brick1:
10.20.5.104:/var/lib/heketi/mounts/vg_9de785372f550942e33d0f3abd8cd9ab/
brick_03cfb63f8293238affe791032ec779c2/brick
Brick2:
10.20.4.40:/var/lib/heketi/mounts/vg_ac91b30f6491c571d91022d24185690f/b
rick_2b60bc032bee1be7341a2f1b5441a37f/brick
Brick3:
10.20.6.79:/var/lib/heketi/mounts/vg_34faf7faaf6fc469298a1c15a0b2fd2f/b
rick_d7181eb86f3ca37e4116378181e28855/brick
Options Reconfigured:
transport.address-family: inet
performance.readdir-ahead: on
nfs.disable: on
```

5.8. CRS FOR OPENSHIFT

5.8.1. Store the heketi secret

OpenShift 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. OpenShift 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 with **`cluster-admin`** privileges use the base64 password string in the following YAML to define the secret in OpenShift's default 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
```

5.8.2. 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)`** adding more flexibility than cloud provider specific storage.

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`**.

```
# vi storage-crs.json
apiVersion: storage.k8s.io/v1beta1
kind: StorageClass
metadata:
  name: crs-slow-st1
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://ip-10-20-4-40.ec2.internal: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 storage-crs.json
```

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

```
# oc get storageclass
NAME             TYPE
crs-slow-st1 kubernetes.io/glusterfs

# oc describe storageclass crs-slow-st1
Name:  crs-slow-st1
IsDefaultClass: No
Annotations: storageclass.beta.kubernetes.io/is-default-class=true
Provisioner: kubernetes.io/glusterfs
Parameters: restauthenabled=true,resturl=http://ip-10-20-4-
40.ec2.internal:8080,
restuser=admin,secretName=heketi-secret,secretNamespace=default
No events.
```

5.8.3. 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-slow-st1 StorageClass** object. A sample persistent volume claim is provided below:

```
$ oc new-project test

$ vi db-claim.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: db
 annotations:
   volume.beta.kubernetes.io/storage-class: crs-slow-st1
spec:
 accessModes:
  - ReadWriteOnce
 resources:
   requests:
     storage: 10Gi

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

5.9. 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 OpenShift standard template and one of the **StorageClass** objects defined

above.

Create an OpenShift 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
```

5.10. 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
```

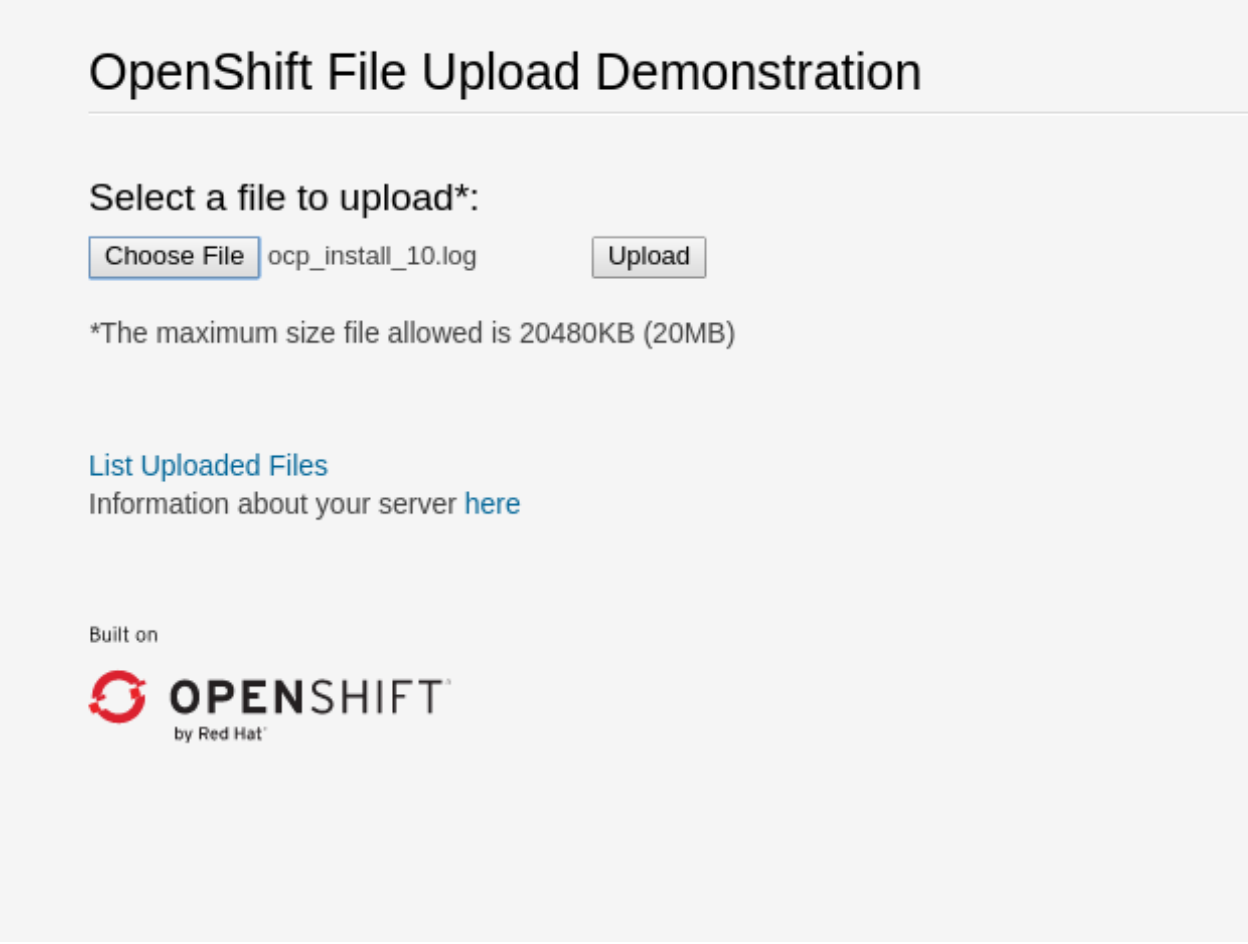
Expose the service as a public route by using the **oc expose** command.

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

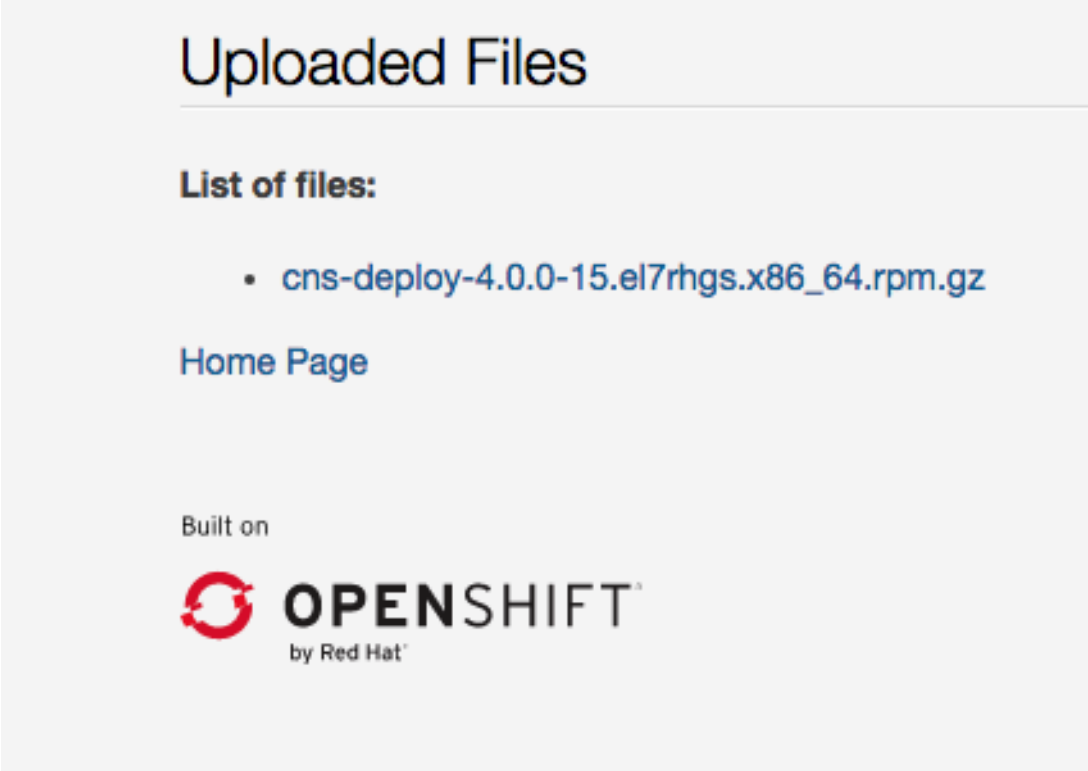
OpenShift will create a route based on the application name, project, and wildcard zone. This will be the **URL** that can be accessed by browser.

```
# oc get route
NAME      HOST/PORT                                   PATH
SERVICES   PORT       TERMINATION   WILDCARD
demo      demo-manual.apps.rcook-aws.sysdeseng.com              demo
8080-tcp                  None
```

Using a web browser validate the application (example http://demo-manual.apps.sysdeseng.com/) 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
total 16K
-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 acces 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 OpenShift 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

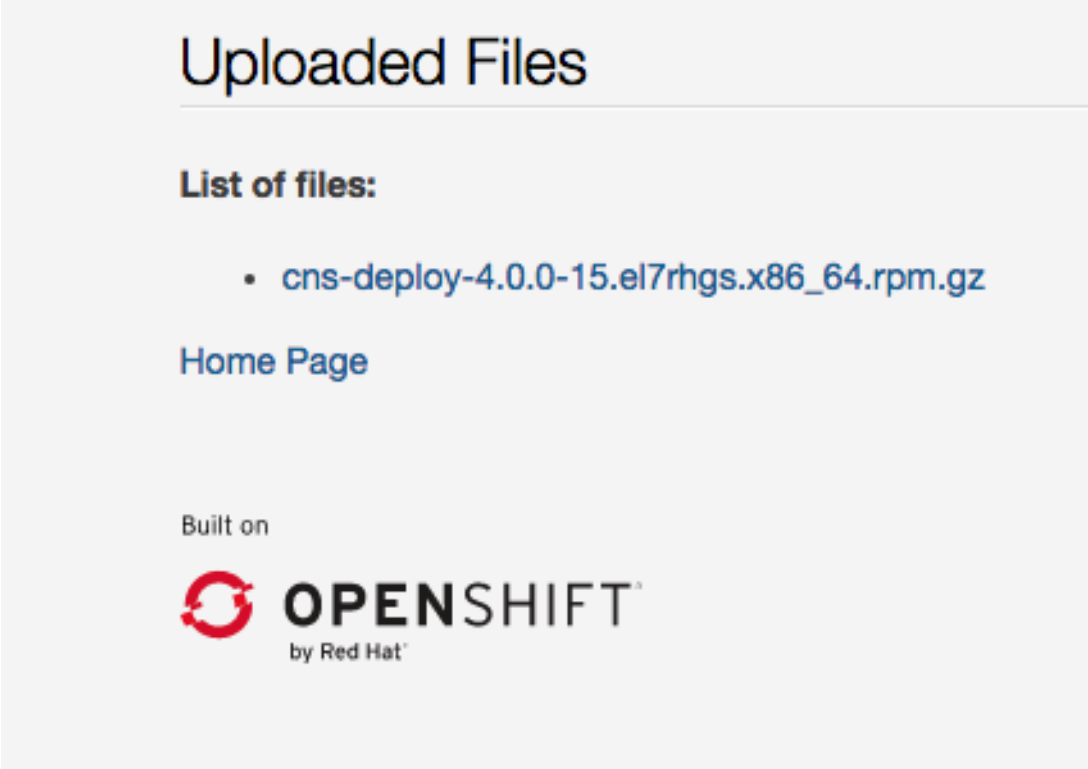
```
# oc get pods
NAME           READY     STATUS      RESTARTS   AGE
demo-1-build   0/1       Completed   0          16m
```

```
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 (example http://demo-manual.apps.sysdeseng.com/).



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….
10.20.4.115:vol_624ec880d10630989a8bdf90ae183366
10G   39M   10G   1% /opt/app-root/src/uploaded
sh-4.2$ cd /opt/app-root/src/uploaded
sh-4.2$ ls -lh
total 5.6M
-rw-r--r--. 1 1000080000 2002 5.6M Apr 26 21:51 heketi-client-4.0.0-
7.el7rhgs.x86_64.rpm.gz
```

On the second pod perform the following.

```
# oc rsh demo-2-m1mwt
sh-4.2$ df -h
Filesystem
Size  Used Avail Use% Mounted on
….omitted….
10.20.4.115:vol_624ec880d10630989a8bdf90ae183366
10G   39M   10G   1% /opt/app-root/src/uploaded
sh-4.2$ cd /opt/app-root/src/uploaded
sh-4.2$ ls -lh
total 5.6M
-rw-r--r--. 1 1000080000 2002 5.6M Apr 26 21:51 heketi-client-4.0.0-
7.el7rhgs.x86_64.rpm.gz
```

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

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

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 5.6M
-rw-r--r--. 1 1000080000 2002 5.6M Apr 26 21:51 heketi-client-4.0.0-
7.el7rhgs.x86_64.rpm.gz
```

Because of the use of a **CNS RWX** persistent volume for the mount point **/opt/app-root/src/uploaded** the file that was uploaded using the Web UI for the demo application is now available to be read or downloaded by all **demo-2** pods no matter how they are scaled up or down.

5.11. DELETING A PVC (OPTIONAL)

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
```

CHAPTER 6. EXTENDING THE CLUSTER

By default, the reference architecture playbooks are configured to deploy 3 master, 3 application, and 2 infrastructure nodes. This cluster size provides enough resources to get started with deploying a few test applications or a Continuous Integration Workflow example. However, 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 **`ose-on-aws.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 meany times as needed. The **`add-node.py`** script launches a new **`AWS Cloudformation Stack`** to provision the new resource.

6.1. PREREQUISITES FOR 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
ip-10-20-4-198.ec2.internal    Ready,SchedulingDisabled   14m
ip-10-20-4-209.ec2.internal    Ready                      14m
ip-10-20-4-232.ec2.internal    Ready                      14m
ip-10-20-5-187.ec2.internal    Ready                      14m
ip-10-20-5-22.ec2.internal     Ready                      14m
ip-10-20-5-94.ec2.internal     Ready,SchedulingDisabled   14m
ip-10-20-6-42.ec2.internal     Ready                      14m
ip-10-20-6-20.ec2.internal     Ready,SchedulingDisabled   14m
```

6.2. INTRODUCTION TO ADD-NODE.PY

The python script **`add-node.py`** is operationally similar to the **`ose-on-aws.py`** script. Parameters can optionally be passed in when calling the script. The **`existing-stack`** trigger allows the Ansible playbooks to associate the new node with the existing **`AWS`** instances. The **`existing-stack`** is the value of **`--stack-name`** when running **`ose-on-aws.py`**. Any required parameters not already set will automatically prompted for at run time. To see all allowed parameters, the --help trigger is available.

Note

On deployments of the Reference Architecture environment post 3.5 --use-cloudformation-facts is available to auto-populate values. If the deployment occurred before 3.5 then it is required to fill in the values manually. To view the possible configuration triggers run **add-node.py -h**

6.3. 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

If **`--use-cloudformation-facts`** is not used the **`--iam-role`** or **`Specify the name of the existing IAM Instance Profile`** is available by logging into the IAM Dashboard and selecting the role sub-menu. Select the node role and record the information from the **`Instance Profile ARN(s)`** line. An example Instance Profile would be **`OpenShift-Infra-NodeInstanceProfile-TNAGMYGY9W8K`**.

If the Reference Architecture deployment is >= 3.5

```
$ ./add-node.py --existing-stack=dev --rhsm-user=rhsm-user --rhsm-
password=password --public-hosted-zone=sysdeseng.com --keypair=OSE-key
--rhsm-pool="Red Hat OpenShift Container Platform, Premium, 2-Core" --
use-cloudformation-facts --shortname=ose-app-node03 --subnet-id=subnet-
0a962f4
```

If the Reference Architecture deployment was performed before 3.5.

```
$ ./add-node.py --existing-stack=dev --rhsm-user=rhsm-user --rhsm-
password=password --public-hosted-zone=sysdeseng.com --keypair=OSE-key
--rhsm-pool="Red Hat OpenShift Container Platform, Premium, 2-Core" --
node-sg=sg-309f0a4a --shortname=ose-app-node03 --iam-role=OpenShift-
Infra-NodeInstanceProfile-TNAGMYGY9W8K --subnet-id=subnet-0a962f4
```

6.4. 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 of the **`infrastructure security group`** (ose_infra_node_sg) and the name of the **`ELB`** used by the router (ose_router_elb). Follow the example steps below to add a new infrastructure node.

Note

If **`--use-cloudformation-facts`** is not used the **`--iam-role`** or **`Specify the name of the existing IAM Instance Profile:`** is available visiting the IAM Dashboard and selecting the role sub-menu. Select the node role and record the information from the **`Instance Profile ARN(s)`** line. An example Instance Profile would be **`OpenShift-Infra-NodeInstanceProfile-TNAGMYGY9W8K`**.

If the Reference Architecture deployment is >= 3.5

```
$ ./add-node.py --existing-stack=dev --rhsm-user=rhsm-user --rhsm-
password=password --public-hosted-zone=sysdeseng.com --keypair=OSE-key
--rhsm-pool="Red Hat OpenShift Container Platform, Premium, 2-Core" --
use-cloudformation-facts --shortname=ose-infra-node04 --node-type=infra
--subnet-id=subnet-0a962f4
```

If the Reference Architecture deployment was performed before 3.5.

```
$ ./add-node.py --rhsm-user=user --rhsm-password=password --public-
```

```
hosted-zone=sysdeseng.com --keypair=OSE-key --rhsm-pool="Red Hat
OpenShift Container Platform, Premium, 2-Core" --node-type=infra --iam-
role=OpenShift-Infra-NodeInstanceProfile-TNAGMYGY9W8K --node-sg=sg-
309f9a4a --infra-sg=sg-289f9a52 --shortname=ose-infra-node04 --subnet-
id=subnet-0a962f4 --infra-elb-name=OpenShift-InfraElb-1N0DZ3CFCAHLV
```

6.5. 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 **`ip-10-20-6-198.ec2.internal`** is an application node newly deployed by the **`add-node.py`** playbooks..

```
$ oc get nodes
NAME                           STATUS                     AGE
ip-10-20-4-198.ec2.internal    Ready,SchedulingDisabled   34m
ip-10-20-4-209.ec2.internal    Ready                      34m
ip-10-20-4-232.ec2.internal    Ready                      34m
ip-10-20-5-187.ec2.internal    Ready                      34m
ip-10-20-5-22.ec2.internal     Ready                      34m
ip-10-20-5-94.ec2.internal     Ready,SchedulingDisabled   34m
ip-10-20-6-198.ec2.internal    Ready                      1m
ip-10-20-6-42.ec2.internal     Ready                      14m
ip-10-20-6-20.ec2.internal     Ready,SchedulingDisabled   34m

$ oc get nodes --show-labels | grep app | wc -l
3
```

CHAPTER 7. MULTIPLE OPENSHIFT DEPLOYMENTS

7.1. PREREQUISITES

The prerequisites described in Section 3.1, “Prerequisites for Provisioning” are required when deploying another **OCP** environment into **AWS**. Below is a checklist to perform to prepare for the deployment of another **OCP** cluster.

- Create subdomain
- Map subdomain NS records to root domain
- Configure authentication

7.1.1. SSH Configuration

The **.ssh/config** will need to reflect both the existing environment and the new environment. Below is an example. The environment of dev will be the existing deployment and prod will be the new deployment.

```
Host dev
     Hostname                bastion.dev.sysdeseng.com
     user                      ec2-user
     StrictHostKeyChecking     no
     ProxyCommand              none
     CheckHostIP               no
     ForwardAgent              yes
     IdentityFile              /home/<user>/.ssh/id_rsa

Host *.dev.sysdeseng.com
     ProxyCommand              ssh ec2-user@dev -W %h:%p
     user                      ec2-user
     IdentityFile              /home/<user>/.ssh/id_rsa

Host prod
     Hostname                  bastion.prod.sysdeseng.com
     user                      ec2-user
     StrictHostKeyChecking     no
     ProxyCommand              none
     CheckHostIP               no
     ForwardAgent              yes
     IdentityFile              /home/<user>/.ssh/id_rsa

Host *.prod.sysdeseng.com
     ProxyCommand              ssh ec2-user@prod -W %h:%p
     user                      ec2-user
     IdentityFile              /home/<user>/.ssh/id_rsa
```

7.2. DEPLOYING THE ENVIRONMENT

Using the **ose-on-aws.py** script to deploy another **OCP** cluster is almost exactly the same as defined in Section 3.1, “Prerequisites for Provisioning” the important difference is **--stack-name**. In the event that **ose-on-aws.py** is launched with the same stack name as the previously

deployed environment the cloudformation facts will be changed causing the existing deployment to be broken.

Note

Verify the existing **`stack name`** by browsing to **`AWS`** and clicking the **`Cloudformation`** service before proceeding with the steps below.

```
$ export AWS_ACCESS_KEY_ID=<key_id>
$ export AWS_SECRET_ACCESS_KEY=<access_key>
$ ./ose-on-aws.py --stack-name=prod --rhsm-user=rhsm-user --rhsm-
password=rhsm-password \ --public-hosted-zone=prod.sysdeseng.com --
keypair=OSE-key \ --github-client-
secret=47a0c41f0295b451834675ed78aecfb7876905f9 \ --github-
organization=openshift \ --github-organization=RHSyseng --github-
client-id=3a30415d84720ad14abc --rhsm-pool="Red Hat OpenShift Container
Platform, Standard, 2-Core"
```

Example of Greenfield Deployment Values

The highlighted value **`stack_name: prod`** ensures that the dev deployment will not be compromised.

```
Configured values:
    stack_name: prod
    ami: ami-10251c7a
    region: us-east-1
    master_instance_type: m4.large
    node_instance_type: t2.medium
    app_instance_type: t2.medium
    bastion_instance_type: t2.micro
    keypair: OSE-key
    create_key: no
    key_path: /dev/null
    create_vpc: yes
    vpc_id: None
    private_subnet_id1: None
    private_subnet_id2: None
    private_subnet_id3: None
    public_subnet_id1: None
    public_subnet_id2: None
    public_subnet_id3: None
    byo_bastion: no
    bastion_sg: /dev/null
    console port: 443
    deployment_type: openshift-enterprise
    openshift_sdn: openshift-ovs-subnet
    public_hosted_zone: prod.sysdeseng.com
    app_dns_prefix: apps
    apps_dns: apps.prod.sysdeseng.com
    rhsm_user: rhsm-user
    rhsm_password:  rhsm_pool: Red Hat OpenShift Container Platform,
```

```
Standard, 2-Core containerized: False s3_bucket_name: prod-ocp-
registry-prod s3_username: prod-s3-openshift-user github_client_id:
    github_client_secret: *
    github_organization: openshift,RHSyseng
```

CHAPTER 8. CONCLUSION

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

This reference architecture covered the following topics:

- **A completely provisioned infrastructure in `AWS`**
- **OpenShift Masters in Multiple `Availability Zones`**
- **Infrastructure nodes in Multiple `Availability Zones` with Router and Registry pods scaled accordingly**
- **Native integration with `AWS` services like `Route53`, `EBS`, `S3`, `IAM`, `EC2`**
  - **Elastic Load Balancers for the Master instances and for the Infrastructure instances**
  - **S3 storage for persistent storage of container images**
  - **EBS storage for /var/lib/docker on each node**
  - **A role assigned to instances that will allow OpenShift to mount EBS volumes**
- **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

Revision	Release Date	Author(s)
1.0	Tuesday September 8, 2016	Scott Collier / Ryan Cook
1.1	Tuesday September 20, 2016	Scott Collier / Ryan Cook
1.2	Tuesday September 27, 2016	Scott Collier / Ryan Cook
1.3	Friday September 30, 2016	Scott Collier / Ryan Cook
1.4	Friday October 21, 2016	Scott Collier / Ryan Cook
1.5	Friday October 28, 2016	Scott Collier / Ryan Cook
1.6	Monday November 14, 2016	Scott Collier / Ryan Cook
1.7	Monday December 9, 2016	Scott Collier / Ryan Cook
1.8	Tuesday January 24, 2017	Scott Collier / Ryan Cook
1.9	Tuesday February 21, 2017	Scott Collier / Ryan Cook
1.10	Friday April 28, 2017	Scott Collier / Ryan Cook
PDF generated by Asciidoctor PDF		
Reference Architecture Theme version 1.2		

1.10 Revision Changelog

- **Moving of container storage to it’s own chapter**
- **OpenShift 3.5**

- **CNS deployment defined**
- **CRS deployment defined**
- **Safer method of installing Ansible**

1.9 Revision Changelog

- **Addition of minor upgrade playbook**
- **SDN selection**
- **Suggestions and fixes of document**

1.8 Revision Changelog

- **Addition of 1 infrastructure node to allow for 0 downtime upgrade of OpenShift**
- **Dynamic Provisioning and Storage Classes Defined in Ch4**
- **3.3 → 3.4 of OpenShift Container Platform**

APPENDIX B. CONTRIBUTORS

Jason DeTiberus, content provider

Annette Clewett, content provider

Daniel Messer, content provider

Erik Jacobs, content reviewer

Matt Woodson, content reviewer

Rayford Johnson, content reviewer

Roger Lopez, content reviewer

APPENDIX C. INSTALLATION FAILURE

In the event of an OpenShift installation failure perform the following steps. The first step is to create an inventory file and run the uninstall playbook.

C.1. INVENTORY

The static inventory is used with the uninstall playbook to identify OpenShift nodes. Modify the inventory below to match the deployed environment or use the Ansible playbook located at `openshift-ansible-contrib/reference-architecture/aws-ansible/playbooks/create-inventory-file.yaml` to create an inventory.

Automated inventory creation

```
# cd /home/USER/git/openshift-ansible-contrib/reference-
architecture/aws-ansible
# ansible-playbook -i inventory/aws/hosts -e 'region=us-east-1
stack_name=openshift-infra
github_client_secret=c3cd9271ffb9f7258e135fcf3ea3a358cffa46b1
github_organization=["openshift"] console_port=443
wildcard_zone=apps.sysdeseng.com public_hosted_zone=sysdeseng.com
playbooks/create-inventory-file.yaml
```

Manual creation

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

[OSEv3:vars]
debug_level=2
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=443
openshift_master_console_port=443
osm_cluster_network_cidr=172.16.0.0/16
openshift_registry_selector="role=infra"
openshift_router_selector="role=infra"
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=3
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi
openshift_cloudprovider_kind=aws
openshift_master_cluster_hostname=internal-openshift-
master.sysdeseng.com
openshift_master_cluster_public_hostname=openshift-master.sysdeseng.com
osm_default_subdomain=apps.sysdeseng.com
openshift_master_default_subdomain=apps.sysdeseng.com
```

```
osm_default_node_selector="role=app"
deployment_type=openshift-enterprise
os_sdn_network_plugin_name="redhat/openshift-ovs-subnet"
openshift_master_identity_providers=[{'name': 'github',
'mapping_method': 'claim', 'clientID': 's3taasdgdt34tq',
'clientSecret': 'asfgasfag34qg3q4gq43gv', 'login': 'true', 'challenge':
'false', 'kind': 'GitHubIdentityProvider', 'organizations': 'openshift'
}]
osm_use_cockpit=true
containerized=false
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_accesskey="{{
hostvars['localhost']['s3user_id'] }}"
openshift_hosted_registry_storage_s3_secretkey="{{
hostvars['localhost']['s3user_secret'] }}"
openshift_hosted_registry_storage_s3_bucket="{{ hostvars['localhost']
['s3_bucket_name'] }}"
openshift_hosted_registry_storage_s3_region="{{ hostvars['localhost']
['region'] }}"
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true

[masters]
ose-master01.sysdeseng.com openshift_node_labels="{'role': 'master'}"
ose-master02.sysdeseng.com openshift_node_labels="{'role': 'master'}"
ose-master03.sysdeseng.com openshift_node_labels="{'role': 'master'}"

[etcd]
ose-master01.sysdeseng.com
ose-master02.sysdeseng.com
ose-master03.sysdeseng.com
[nodes]
ose-master01.sysdeseng.com openshift_node_labels="{'role': 'master'}"
ose-master02.sysdeseng.com openshift_node_labels="{'role': 'master'}"
ose-master03.sysdeseng.com openshift_node_labels="{'role': 'master'}"
ose-infra-node01.sysdeseng.com openshift_node_labels="{'role':
'infra'}"
ose-infra-node02.sysdeseng.com openshift_node_labels="{'role':
'infra'}"
ose-infra-node03.sysdeseng.com openshift_node_labels="{'role':
'infra'}"
ose-app-node01.sysdeseng.com openshift_node_labels="{'role': 'app'}"
ose-app-node02.sysdeseng.com openshift_node_labels="{'role': 'app'}"
```

C.2. RUNNING THE UNINSTALL PLAYBOOK

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

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

C.3. MANUALLY LAUNCHING THE INSTALLATION OF OPENSHIFT

The playbook below is the same playbook that is ran once the deployment of AWS resources is completed. Replace the rhsm_user and rhsm_password, stack_name, set the wildcard_zone and public_hosted_zone relevant to the information in Route53 and optionally modify the AWS region in the event us-east-1 was not used..

```
ansible-playbook -i inventory/aws/hosts -e 'region=us-east-1
stack_name=openshift-infra
keypair=OSE-key public_hosted_zone=sysdeseng.com
wildcard_zone=apps.sysdeseng.com
console_port=443 deployment_type=openshift-enterprise
rhsm_user=RHSM_USER
rhsm_password=RHSM_PASSWORD rhsm_pool="Red Hat OpenShift Container
Platform, Standard, 2-Core"
containerized=False github_client_id=e76865557b0417387b35
github_client_secret=c3cd9271ffb9f7258e135fcf3ea3a358cffa46b1
github_organization=["openshift"]' playbooks/openshift-install.yaml
```

Also, the ose-on-aws.py can be executed again but this must be done with caution. If any of the variables in ose-on-aws.py are changed the cloudformation may update causing the AWS components to change.

```
./ose-on-aws.py --rhsm-user=RHSM_USER --rhsm-password=RHSM_PASSWORD --
public-hosted-zone=sysdeseng.com
--rhsm-pool="Red Hat OpenShift Container Platform, Standard, 2-Core" --
keypair=OSE-key
--master-instance-type=t2.medium --stack-name=tag --github-client-
id=e76865557b0417387b35
--github-organization=openshift --github-client-
secret=c3cd9271ffb9f7258e135fcf3ea3a358cffa46b1
```

APPENDIX D. FAILURE WHEN ADDING NODE(S)

At the time of writing there is a known bug in the OpenShift installation routine that may cause a failure when adding nodes either with the `add-cns-storage.py` or `add-node.py` playbooks. The failure occurs during the step "Generate the node client config". In this case the output of the the installation routine is similar to the following:

```
failed: [ose-ocp-cns01-node03.sysdeseng.com -> ose-
master03.sysdeseng.com] (item=ose-ocp-cns01-node03.sysdeseng.com) => {
    "changed": true,
    "cmd": [
        "oc",
        "adm",
        "create-api-client-config",
        "--certificate-authority=/etc/origin/master/ca.crt",
        "--client-dir=/etc/origin/generated-configs/node-ip-10-20-6-
56.us-west-2.compute.internal",
        "--groups=system:nodes",
        "--master=https://internal-openshift-master.sysdeseng.com",
        "--signer-cert=/etc/origin/master/ca.crt",
        "--signer-key=/etc/origin/master/ca.key",
        "--signer-serial=/etc/origin/master/ca.serial.txt",
        "--user=system:node:ip-10-20-6-56.us-west-2.compute.internal"
    ],
    "delta": "0:00:00.158849",
    "end": "2017-04-17 17:27:29.789575",
    "failed": true,
    "item": "ose-ocp-cns01-node03.sysdeseng.com",
    "rc": 1,
    "start": "2017-04-17 17:27:29.630726",
    "warnings": []
}

STDERR:

error: --signer-serial, "/etc/origin/master/ca.serial.txt" must be a
valid file
See 'oc adm create-api-client-config -h' for help and examples.
```

In this case the `Ansible` deployment routine picks the wrong OpenShift master node to create the client configuration. Only one of the 3 master nodes has the file `/etc/origin/master/ca.serial.txt` present and in the case above an OpenShift master was used, that doesn't have this file. If this error occurs, find the master node on which this file exists and copy the file to the all master nodes where this file does not exist in the `/etc/origin/master` directory. The file should only contain 2 hexadecimal digits therefore creating the file in `/etc/origin/master` path on the master nodes by copy/pasting the content is sufficient as well. The file needs to be owned by the user and group "root" and carries the permission mask 0644. When done, simply re-run the deployment script again with the same arguments and this error should not reappear. Once the node installation is successful remove the `ca.serial.txt` from the masters that did not initially have the file.

APPENDIX E. 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 OpenShift user doesn't have permission in the current project The OpenShift app nodes don't have connectivity to the Red Hat Registry to download the GlusterFS container images The firewall rules on the EC2 app nodes or the AWS security groups 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 EC2 nodes being used for CNS cluster deployments.

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

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. This must be done on all EC2 nodes referenced in the topology.json file.

```
# lvremove -y vg_xxxxxxxxxxxxxxx
# pvremove /dev/<block-device>
```