



Red Hat Reference Architecture Series

Deploying OpenShift Container Platform 3 on Amazon Web Services

Ryan Cook, Scott Collier

Version 1.3, 2016-09-30

Table of Contents

- Comments and Feedback 2
- 1. Executive Summary 3
- 2. Components and Configuration 4
 - 2.1. Elastic Compute Cloud Instance Details 5
 - 2.2. Elastic Load Balancers Details 5
 - 2.3. Software Version Details 6
 - 2.4. Required Channels 6
 - 2.5. Tooling Prerequisites 7
 - 2.5.1. Ansible Setup 7
 - 2.5.2. Git Repository 7
 - 2.5.3. AWS Region Requirements 8
 - 2.5.4. Permissions for Amazon Web Services 8
 - 2.6. Virtual Private Cloud (VPC) 9
 - 2.7. NAT Gateway 10
 - 2.8. Security Groups 10
 - 2.8.1. Master ELB Security Group 11
 - 2.8.2. Internal Master ELB Security Group 12
 - 2.8.3. Bastion Security Group 13
 - 2.8.4. Master Security Group 14
 - 2.8.5. ETCD Security Group 15
 - 2.8.6. Router ELB Security Group 16
 - 2.8.7. Infrastructure Nodes Security Group 17
 - 2.8.8. Nodes Security Group 18
 - 2.9. Route53 18
 - 2.9.1. Public Zone 19
 - 2.9.2. Hosted Zone Setup 19
 - 2.9.3. Amazon Machine Images 19
 - 2.9.4. Identity and Access Management 20
 - 2.10. Bastion 20
 - 2.11. Dynamic Inventory 21
 - 2.12. Nodes 21
 - 2.12.1. Master nodes 21
 - 2.12.2. Infrastructure nodes 21
 - 2.12.3. Application nodes 22
 - 2.12.4. Node labels 22
 - 2.13. OpenShift Pods 22

- 2.14. Router 22
- 2.15. Registry 23
- 2.16. Authentication 23
- 3. Provisioning the Infrastructure 24
 - 3.1. Provisioning the Infrastructure with Ansible 24
 - 3.1.1. Authentication Prerequisite 24
 - 3.1.2. SSH Prerequisite 29
 - 3.1.3. AWS Authentication Prerequisite 30
 - 3.1.4. Red Hat Subscription Prerequisite 31
 - 3.1.5. Deploying the Environment 31
 - 3.2. Post Provisioning Results 37
- 4. Operational Management 39
 - 4.1. Validate the Deployment 39
 - 4.2. Gathering hostnames 40
 - 4.3. Running Diagnostics 40
 - 4.4. Checking the Health of ETCD 49
 - 4.5. Default Node Selector 50
 - 4.6. Management of Maximum Pod Size 50
 - 4.7. Yum Repositories 52
 - 4.8. Console Access 52
 - 4.8.1. Log into GUI console and deploy an application 52
 - 4.8.2. Log into CLI and Deploy an Application 53
 - 4.9. Explore the Environment 55
 - 4.9.1. List Nodes and Set Permissions 55
 - 4.9.2. List Router and Registry 56
 - 4.9.3. Explore the Docker Registry 57
 - 4.9.4. Explore Docker Storage 58
 - 4.9.5. Explore Security Groups 61
 - 4.9.6. Explore the AWS Elastic Load Balancers 61
 - 4.9.7. Explore the AWS VPC 62
 - 4.10. Persistent Volumes 62
 - 4.10.1. Node Labels for Persistent Volumes 63
 - 4.10.2. Creating a Persistent Volumes 63
 - 4.10.3. Creating a Persistent Volumes Claim 64
 - 4.11. Testing Failure 66
 - 4.11.1. Generate a Master Outage 66
 - 4.11.2. Observe the Behavior of ETCD with a Failed Master Node 66
 - 4.11.3. Generate an Infrastructure Node outage 67

- 5. Conclusion 71
- Appendix A: Revision History 72
- Appendix B: Contributors..... 73
- 6. Installation Failure 74
 - 6.1. Inventory 75
 - 6.2. Running the Uninstall Playbook 76
 - 6.3. Manually Launching the Installation of OpenShift 76

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

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

© 2015 by Red Hat, Inc. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

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

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

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

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

Send feedback to refarch-feedback@redhat.com

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.

1. Executive Summary

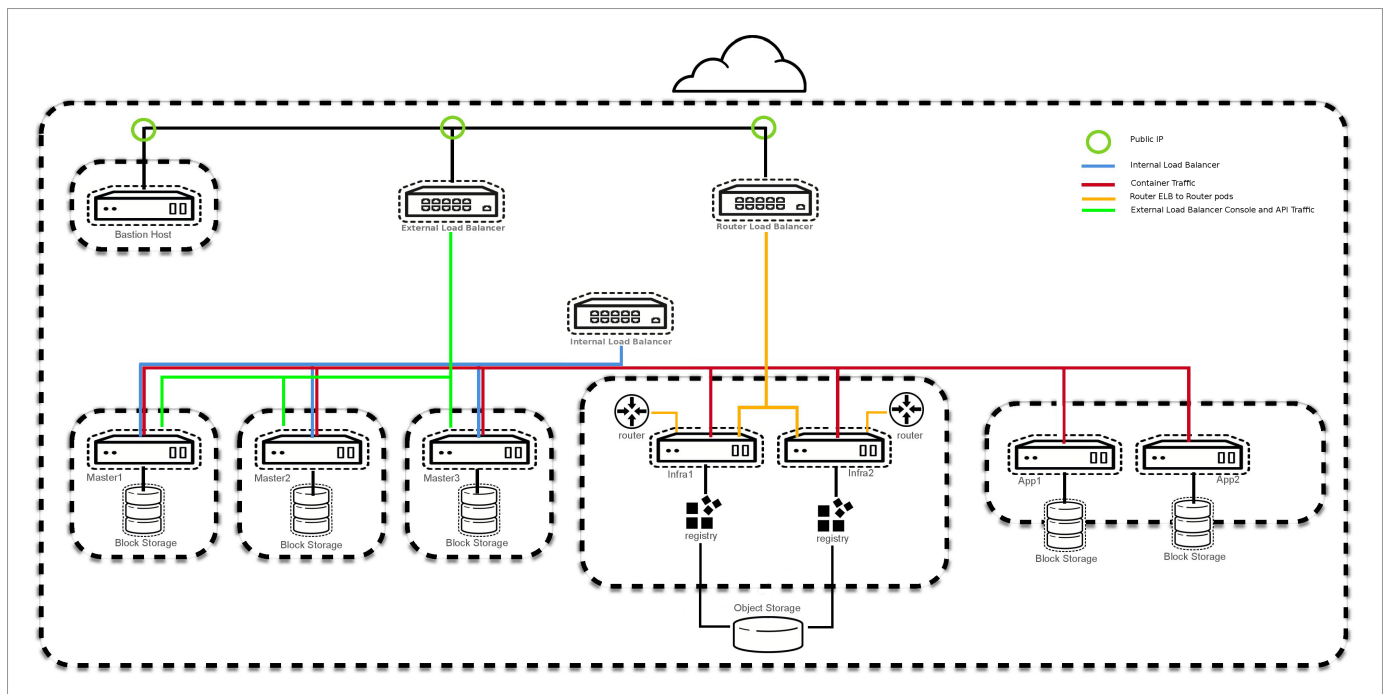
Red Hat OpenShift Container Platform 3 is built around a core of application containers powered by Docker, with orchestration and management provided by Kubernetes, on a foundation of Atomic Host and Red Hat Enterprise Linux. OpenShift Origin is the upstream community project that brings it all together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

2. Components and Configuration

This chapter describes the reference architecture environment that is deployed that enables the configuration of a highly available OpenShift Compute Platform 3 environment on Amazon Web Services (AWS).

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 Google 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 Compute 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 provisioning of OpenShift Container Platform which is done via the Ansible playbooks installed by the [openshift-ansible-playbooks](#) rpm package. The playbooks in [openshift-ansible-contrib](#) utilize the playbooks defined by the [openshift-ansible-playbooks](#) 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.



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.large](#) and contain two extra disks used for Docker storage and [ETCD](#). The node instances are [t2.medium](#) 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 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-nodes01-2	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. RHEL OSEv3 Details

Software	Version
Red Hat Enterprise Linux 7.2 x86_64	kernel-3.10.0-327
Atomic-OpenShift{master/clients/node/sdn-ovs/utils}	3.3.x.x
Docker	1.10.x
Ansible	2.2.0-0.5.prerelease.el7.noarch

2.4. Required Channels

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

Table 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 Enterprise 3.3 (RPMs)	rhel-7-server-ose-3.3-rpms
Red Hat Enterprise Linux 7 Server - Extras (RPMs)	rhel-7-server-extras-rpms

2.5. 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.

2.5.1. Ansible Setup

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

```
$ rpm -q python-2.7
$ subscription-manager repos --enable rhel-7-server-optional-rpms
$ subscription-manager repos --enable rhel-7-server-ose-3.2-rpms
$ subscription-manager repos --enable rhel-7-server-ose-3.3-rpms
$ rpm -Uvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ yum -y install atomic-openshift-utils \
    python2-boto \
    git \
    ansible-2.2.0-0.5.prerelease.el7.noarch |
    python-netaddr \
    python-httplib2
```



The Extra Packages for Enterprise Linux (EPEL) repository is being used for the installation of the `python2-boto` packages specific for this installation. While EPEL is not explicitly supported by Red Hat, it is how Ansible interacts with **AWS**.

2.5.2. Git Repository

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.



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

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
```

2.5.3. 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.

2.5.4. 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.

2.6. 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/git/openshift-ansible-contrib/reference-architecture/aws-ansible/playbooks/vars/main.yaml`.

Table 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
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/>

2.7. 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.

2.8. 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.

2.8.1. 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 [AWS Master ELB Security Group Details - Inbound](#) diagram and [AWS Master ELB Security Group Details - Inbound](#) table below.

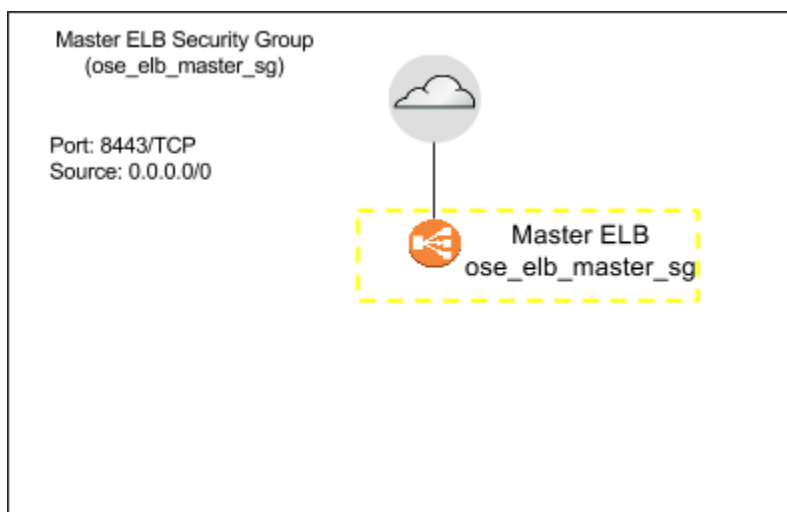


Figure 1. AWS Master ELB Security Group Details - Inbound

Table 5. AWS Master ELB Security Group Details - Inbound

Inbound	From
443 / TCP	Anywhere

Table 6. AWS Master ELB Security Group Details - Outbound

Outbound	To
443	ose_master_sg

2.8.2. 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.

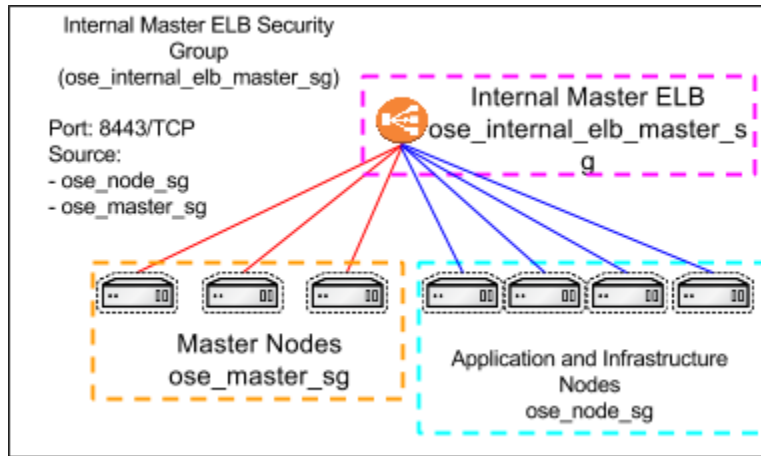


Figure 2. AWS Internal Master ELB Security Group Details - Inbound

Inbound	From
443 / TCP	ose_node_sg
443 / TCP	ose_master_sg

Table 7. AWS Internal Master ELB Security Group Details - Outbound

Outbound	To
443	ose_master_sg

2.8.3. 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.

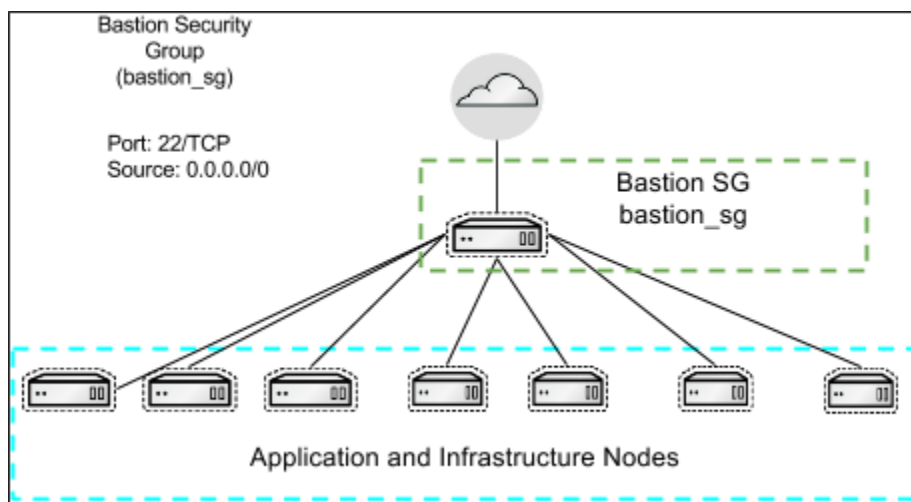


Table 8. AWS Bastion Security Group Details - Inbound

Inbound	From
22 / TCP	Anywhere

Table 9. AWS Bastion Security Group Details - Outbound

Outbound	To
All	All

2.8.4. 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 SkyDNS.

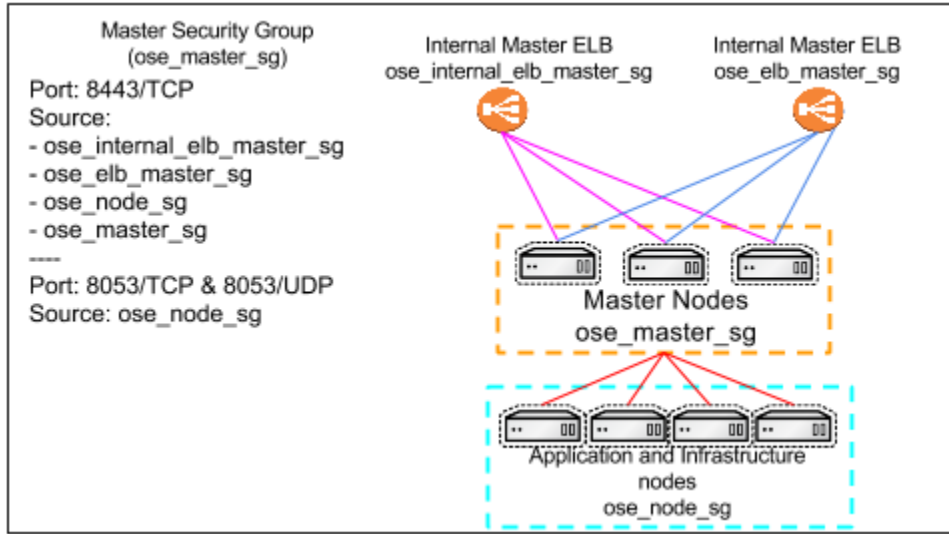


Figure 3. AWS Master Security Group Details - Inbound

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 10. AWS Master Security Group Details - Outbound

Outbound	To
All	All

2.8.5. 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.

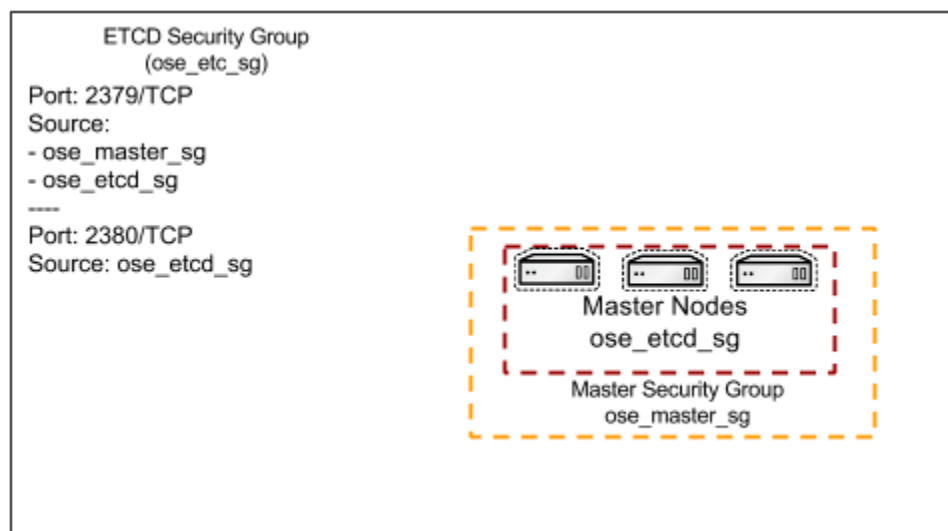


Table 11. ETCD Security Group Details - Inbound

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

Table 12. ETCD Security Group Details - Outbound

Outbound	To
All	All

2.8.6. 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.

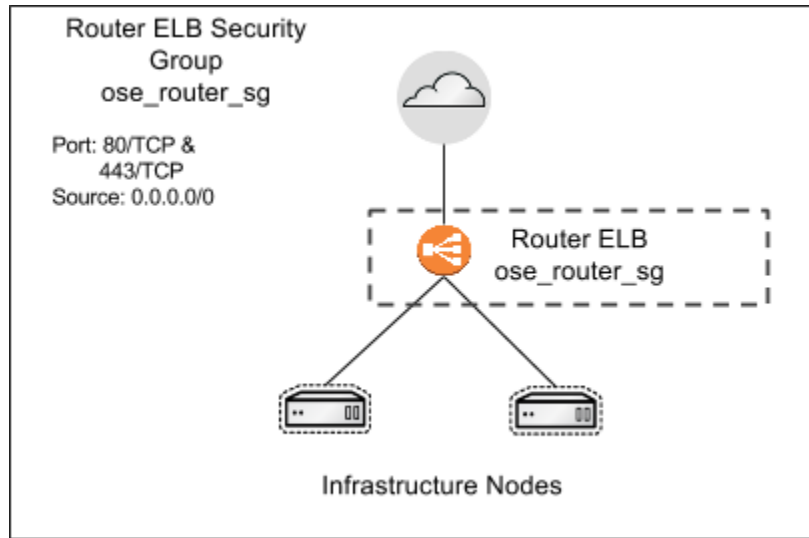


Figure 4. AWS Router ELB Security Group Details - Inbound

Inbound	From
443 / TCP	Anywhere
80 / TCP	Anywhere

Table 13. AWS Router ELB Security Group Details - Outbound

Outbound	To
80	ose_infra_node_sg
443	ose_infra_node_sg

2.8.7. Infrastructure Nodes Security Group

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

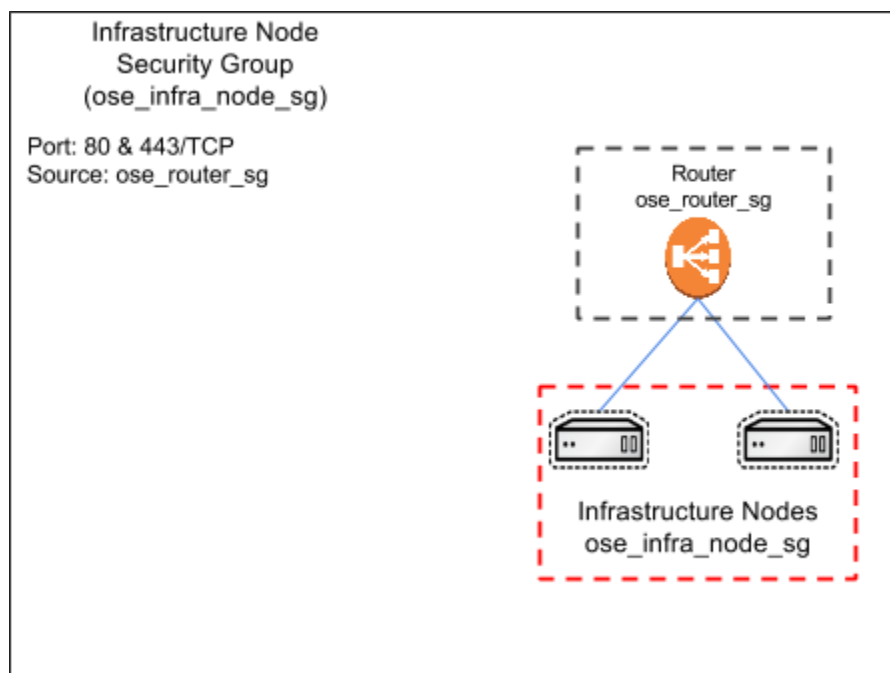


Figure 5. AWS Infrastructure Nodes Security Group Details - Inbound

Inbound	From
80 / TCP	ose_router_sg
443 / TCP	ose_router_sg

Table 14. AWS Infrastructure Nodes Security Group Details - Outbound

Outbound	To
All	All

2.8.8. Nodes Security Group

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

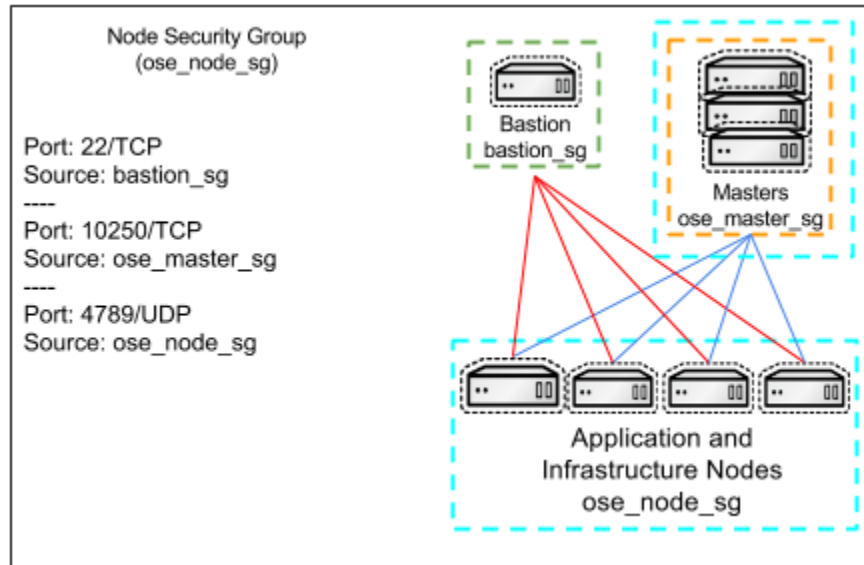


Figure 6. AWS Nodes Security Group Details - Inbound

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

Table 15. AWS Application Nodes Security Group Details - Outbound

Outbound	To
All	All

2.9. 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 [OpenShift Container Platform DNS](#). In this reference architecture Route53 will manage DNS records for the OpenShift Container Platform environment.

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

2.9.1. Public Zone

The Public [Route53](#) zone requires a domain name either purchased through [AWS](#) or another external provider such as Google Domains or GoDaddy. Once the zone is created in [Route53](#), the name servers provided by Amazon will need to be added to the registrar.

2.9.2. Hosted Zone Setup

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

Once the Public 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 if applicable.



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

2.9.3. 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](#).

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.

2.9.4. 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 be created to manage a S3 bucket used for the Docker 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/>

2.10. Bastion

As shown in the 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.

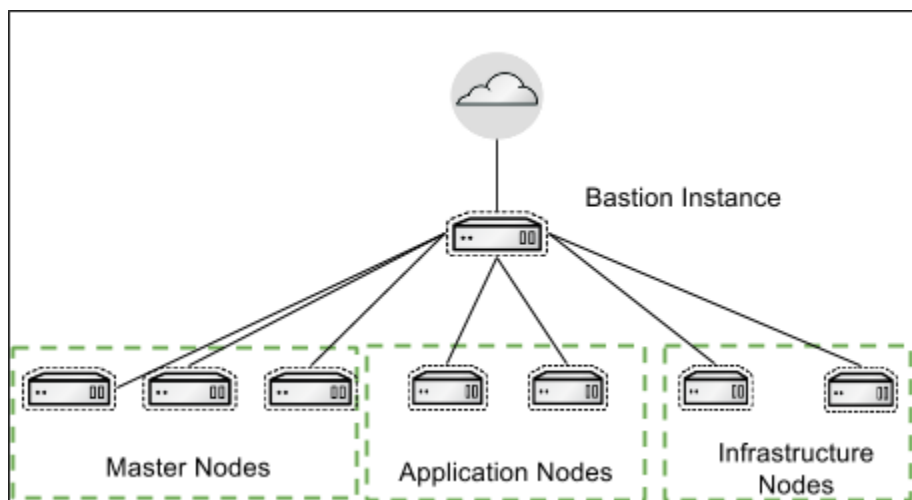


Figure 7. Bastion Diagram

2.11. 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 scanned automatically using a dynamic inventory script which generates an Ansible host file is generated in memory. 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 http://docs.ansible.com/ansible/intro_dynamic_inventory.html

2.12. 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.

2.12.1. 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.

2.12.2. 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.

2.12.3. 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.

2.12.4. 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.

2.13. OpenShift Pods

OpenShift leverages 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.

2.14. 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.

2.15. 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](#).

2.16. 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](#).

3. Provisioning the Infrastructure

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

3.1. Provisioning the Infrastructure with Ansible

The script and playbooks provided within the git repository deploys infrastructure, installs and configures OpenShift, and performs post installation tasks such as scaling the router and registry. The playbooks create specific roles, policies, and users required for cloud provider configuration in OpenShift and management of a newly created [S3](#) bucket to manage container images.

3.1.1. Authentication Prerequisite




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.

Create an Organization

An existing organization can be used when using GitHub authentication. If an organization does not exist then it is highly advised to create one. If an organization is not created anyone on GitHub can use the installation of OpenShift.


An example is provided below.

Create an organization

 Completed Set up a personal account	 Step 2: Set up the organization	 Step 3: Invite members
---	---	--

Set up the organization

Organization name



The organization will live at <https://github.com/sysdeseng-openshift>




Billing email

Receipts will be sent here

Plan

- Unlimited members and public repositories for free.
- Unlimited private repositories** at \$25/month for your first 5 users. \$9/month for each additional user.

Organizations

-  Repository management
-  Fine-grained permissions
-  Focused dashboard




The credit card and plan you choose on this screen will be billed to the organization — not your user account (**cooktheryan**).

[Learn more](#)

Figure 8. GitHub New Organization

- Insert an Organization name
- Insert a Billing email
- Select a Plan (The Unlimited members and public repositories for free is an acceptable option)
- Click Create organization

Invite organization members

 Completed Set up a personal account	 Step 2: Set up the organization	 Step 3: Invite members
---	---	--

Search by username, full name or email address


scollier Invited ×
 Scott Collier

Finish

Organization members

- ✓ See all repositories ?
- ✓ Create repositories
- ✓ Organize into teams
- ✓ Review code
- ✓ Communicate via @mentions

As an organization **owner**, you'll have complete access to **all of the organization's repositories** and have control of what members have access using fine-grained permissions.

You'll also be able to change billing info and [cancel](#) organization plans.

[Learn more](#)

Members will receive their invitation via email. They can also visit <https://github.com/sysdeseng-openshift> to accept the invitation right away.

Figure 9. GitHub Invite organization members

OPTIONAL

- Add additional GitHub accounts to the organization
- Click Finish

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.

Register a new OAuth application

Application name

Something users will recognize and trust

Homepage URL

The full URL to your application homepage

Application description

This is displayed to all potential users of your application


Authorization callback URL

Your application's callback URL. Read our [OAuth documentation](#) for more information.

Figure 10. 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

OpenShift-3.2-Reference-Arch

 cooktheryan owns this application. [Transfer ownership.](#)

0 users

Client ID

c3507b02427f9487900a

Client Secret

f6f225503a9a45a7b95557b37d30fdbfce083447

[Revoke all user tokens](#)

[Reset client secret](#)

Figure 11. 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.

OAuth Variable

Modify the file `openshift-setup.yaml` changing the `clientId` and `clientSecret` values using the information presented after registering OpenShift as a OAuth application.

```
$ vim /home/<user>/git/openshift-ansible-contrib/reference-architecture/aws-
ansible/playbooks/openshift-setup.yaml
... omitted ...
  openshift_master_identity_providers:
  - name: github
    kind: GitHubIdentityProvider
    login: true
    challenge: false
    mapping_method: claim
    clientId: 3a90715d36470ad14a9c
    clientSecret: 47a0c61f7095b351839675ed78aecfb7876925f9
    organizations:
    - openshift
... omitted ...
```


3.1.2. SSH Prerequisite

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.



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 16. 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
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

3.1.3. AWS Authentication Prerequisite

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.html>

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>
```

3.1.4. Red Hat Subscription Prerequisite

The installation of OCP requires a valid Red Hat subscription. For the installation of OCP on AWS the following items are required:

Red Hat Subscription Manager User: rhsm-se

Red Hat Subscription Manager Password: SecretPass

Subscription Name or Pool ID: 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 Pool ID and Subscription Name as the values will be used below during the deployment.



Subscription Name or Pool ID can be used. An example Pool ID value would be 8b85a9813e313e4a013e47f6cbe16ee0.

3.1.5. Deploying 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.

Introduction to `ose-on-aws.py`

The `ose-on-aws.py` script contains many different configuration options such as changing 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. To see all of the potential options the `--help` trigger is available.

Usage: ose-on-aws.py [OPTIONS]

Options:

```
--console-port INTEGER RANGE  OpenShift web console port [default: 443]
--deployment-type TEXT        OpenShift deployment type [default:
                                openshift-enterprise]
--region TEXT                  ec2 region [default: us-east-1]
--ami TEXT                     ec2 ami [default: ami-10251c7a]
--master-instance-type TEXT    ec2 instance type [default: m4.large]
--node-instance-type TEXT     ec2 instance type [default: t2.medium]
--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]
--no-confirm                   Skip confirmation prompt
-h, --help                     Show this message and exit.
-v, --verbose
```

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 [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:SpfGaSv23aDasVsIRPftNsXa0AbfiuSJ1Pj+e5tN52Y user@goku.rdu.redhat.com
The key's randomart image is:
+---[RSA 2048]-----+
| . . . . . |
| . . . . . |
| . . ooo+* |
| . o BoX.o |
|o = @ B S. |
| = 0 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 --create-key=yes --rsm-user=rsm-user --rsm-password=rsm-password \
--public-hosted-zone=sysdeseng.com --key-path=/home/<user>/.ssh/id_rsa.pub \
--keypair=OSE-key --rsm-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 --rhsm-user=rhsm-user --rhsm-password=rhsm-password \  
--public-hosted-zone=sysdeseng.com --keypair=OSE-key \  
--rhsm-pool="Red Hat OpenShift Container Platform, Standard, 2-Core"
```

Example of Greenfield Deployment values

```
Configured values:  
ami: ami-10251c7a  
region: us-east-1  
master_instance_type: t2.medium  
node_instance_type: t2.medium  
keypair: OSE-key  
create_key: no  
key_path: /dev/null  
create_vpc: yes  
vpc_id: None  
subnet_id1: None  
subnet_id2: None  
subnet_id3: None  
subnet_id4: None  
subnet_id5: None  
subnet_id6: None  
byo_bastion: no  
console port: 443  
deployment_type: openshift-enterprise  
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
```

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

Brownfield Deployment

The `ose-on-aws.py` script allows for deployments into an existing environment in which a VPC already exists and subnets are already created. The script expects three public and three private subnets are created. The private subnets must be able to connect externally. 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 --create-vpc=no --rasm-user=rasm-user --rasm-password=rasm-password \
--public-hosted-zone=sysdeseng.com --keypair=OSE-key \ --rasm-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 within `ose-on-aws.py` exists to not deploy the bastion instance.



If the bastion instance is already deployed supply the security group id of the bastion security group.

```
$ ./ose-on-aws.py --create-vpc=no --rasm-user=rasm-user --rasm-password=rasm-password \
--public-hosted-zone=sysdeseng.com --keypair=OSE-key --byo-bastion=yes \
--bastion-sg=sg-a34ff3af --rasm-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 trigger 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 \  
--bastion-sg=sg-a34ff3af --ami=ami-2051294a --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

Example of Brownfield Deployment values

Configured values:

```
ami: ami-2051294a
region: us-east-1
master_instance_type: t2.medium
node_instance_type: t2.medium
keypair: OSE-key
create_key: no
key_path: /dev/null
create_vpc: no
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: yes
console port: 443
deployment_type: openshift-enterprise
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
```

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

Post Ansible Deployment

Once the playbooks have successfully completed the next steps will be to perform the steps defined in [Operational Management](#). In the event that OpenShift failed to install, follow the steps in [Appendix C: Installation Failure](#) to restart the installation of OpenShift.

3.2. 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
- 2 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
- 1 IAM role
- 1 IAM Policy
- 1 S3 Bucket
- 1 IAM user
- 2 Zones in Route53

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>.



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.

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



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' 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-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

[Note] Determining if client configuration exists for client/cluster diagnostics

Info: Successfully read a client config file at '/root/.kube/config'

Info: Using context for cluster-admin access: 'default/internal-openshift-master-sysdeseng-com:443/system:admin'

[Note] Performing systemd discovery

[Note] Running diagnostic: ConfigContexts[default/internal-openshift-master-sysdeseng-com:443/system:admin]

Description: Validate client config context is complete and has connectivity

Info: The current client config context is 'default/internal-openshift-master-sysdeseng-com:443/system:admin':

The server URL is 'https://internal-openshift-master.sysdeseng.com'

The user authentication is 'system:admin/internal-openshift-master-sysdeseng-com:443'

The current project is 'default'

Successfully requested project list; has access to project(s):

[management-infra openshift openshift-infra test default kube-system logging]

[Note] Running diagnostic: ConfigContexts[default/openshift-master-sysdeseng-com:443/system:admin]

Description: Validate client config context is complete and has connectivity

Info: For client config context 'default/openshift-master-sysdeseng-com:443/system:admin':

The server URL is 'https://openshift-master.sysdeseng.com'

The user authentication is 'system:admin/internal-openshift-master-sysdeseng-com:443'

The current project is 'default'

Successfully requested project list; has access to project(s):

[test default kube-system logging management-infra openshift openshift-infra]

[Note] Running diagnostic: DiagnosticPod

Description: Create a pod to run diagnostics from the application standpoint

Info: Output from the diagnostic pod (image openshift3/ose-deployer:v3.3.0.32):

[Note] Running diagnostic: PodCheckAuth

Description: Check that service account credentials authenticate as expected

Info: Service account token successfully authenticated to master

Info: Service account token was authenticated by the integrated registry.

[Note] Running diagnostic: PodCheckDns
Description: Check that DNS within a pod works as expected

[Note] Summary of diagnostics execution (version v3.3.0.32):
[Note] Completed with no errors or warnings seen.

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

Info: The "docker-registry" service has multiple associated pods each mounted with ephemeral storage, but also has a custom config /etc/registryconfig/config.yml mounted; assuming storage config is as desired.

WARN: [DClu1012 from diagnostic
ClusterRegistry@openshift/origin/pkg/diagnostics/cluster/registry.go:300]
The pod logs for the "docker-registry-2-whjg6" pod belonging to the "docker-registry" service indicated unknown errors. This could result in problems with builds or deployments. Please examine the log entries to determine if there might be any related problems:

```
time="2016-09-30T11:17:19-04:00" level=error msg="obsolete configuration detected, please add openshift registry middleware into registry config file"
time="2016-09-30T11:17:19-04:00" level=error msg="obsolete configuration detected, please add openshift storage middleware into registry config file"
time="2016-09-30T11:26:26.480496017-04:00" level=error msg="error authorizing context: authorization header required" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=0eca492c-849a-4e0d-954d-2b137d581e90 http.request.method=GET http.request.remoteaddr="172.16.6.1:53710" http.request.uri="/v2/" http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64" instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
time="2016-09-30T11:26:26.620574914-04:00" level=error msg="response completed with error" err.code="blob unknown"
err.detail=sha256:e95563b12733327ccee6c9f46d31d3d785fbc255e479afdbbaee017228f29f err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=62978d37-d1ef-4457-ae92-06786806b828 http.request.method=HEAD http.request.remoteaddr="172.16.6.1:53713" http.request.uri="/v2/test/ruby/blobs/sha256:e95563b12733327ccee6c9f46d31d3d785fbc255e479afdbbaee017228f29f" http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64" http.response.contentType="application/json; charset=utf-8" http.response.duration=122.872562ms http.response.status=404 http.response.written=157 instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
vars.digest="sha256:e95563b12733327ccee6c9f46d31d3d785fbc255e479afdbbaee017228f29f" vars.name="test/ruby"
time="2016-09-30T11:26:26.628583093-04:00" level=error msg="response completed with error" err.code="blob unknown"
```

```
err.detail=sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a87814f991f2119d4c862
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=25a1b861-ec5e-4aef-bd6e-
b3eb55d927dd http.request.method=HEAD http.request.remoteaddr="172.16.6.1:53714"
http.request.uri="/v2/test/ruby/blobs/sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a
87814f991f2119d4c862" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=125.700888ms http.response.status=404 http.response.written=157
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
vars.digest="sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a87814f991f2119d4c862"
vars.name="test/ruby"
    time="2016-09-30T11:26:26.632076376-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b05c1b6fd0b2797b0f9ff9
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=09eb7bc4-4f85-4ec8-83f1-
d78935ee259b http.request.method=HEAD http.request.remoteaddr="172.16.6.1:53716"
http.request.uri="/v2/test/ruby/blobs/sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b0
5c1b6fd0b2797b0f9ff9" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=134.340197ms http.response.status=404 http.response.written=157
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
vars.digest="sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b05c1b6fd0b2797b0f9ff9"
vars.name="test/ruby"
    time="2016-09-30T11:26:26.707144372-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0d602c575e32e9804baed
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=00e147a6-78b3-4e68-911d-
70d34137dc0f http.request.method=HEAD http.request.remoteaddr="172.16.6.1:53715"
http.request.uri="/v2/test/ruby/blobs/sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0
d602c575e32e9804baed" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=201.651374ms http.response.status=404 http.response.written=157
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
vars.digest="sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0d602c575e32e9804baed"
vars.name="test/ruby"
    time="2016-09-30T11:27:07.424507327-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:12d39f69f188b6010c2d6dc8e45d86cd4f36b6cdded4942620b3cae2802a35253
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=bb29115e-a270-41fa-8238-
6f9c344c2d78 http.request.method=HEAD http.request.remoteaddr="172.16.6.1:53753"
http.request.uri="/v2/test/ruby/blobs/sha256:12d39f69f188b6010c2d6dc8e45d86cd4f36b6cdded49
42620b3cae2802a35253" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
```

```
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=32.413514ms http.response.status=404 http.response.written=157
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
vars.digest="sha256:12d39f69f188b6010c2d6dc8e45d86cd4f36b6cded4942620b3cae2802a35253"
vars.name="test/ruby"
    time="2016-09-30T11:27:08.471372091-04:00" level=error msg="response completed
with error" err.code=unknown err.detail="manifest invalid: manifest invalid"
err.message="unknown error" go.version=go1.6.2
http.request.contentType="application/vnd.docker.distribution.manifest.v2+json"
http.request.host="172.30.146.134:5000" http.request.id=4a168355-b2cc-44e3-a7d2-
9417ab47c024 http.request.method=PUT http.request.remoteaddr="172.16.6.1:53759"
http.request.uri="/v2/test/ruby/manifests/latest" http.request.useragent="docker/1.10.3
go/go1.6.2 git-commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux
arch/amd64" http.response.contentType="application/json; charset=utf-8"
http.response.duration=9.195356ms http.response.status=500 http.response.written=136
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9 vars.name="test/ruby"
vars.reference=latest
    time="2016-09-30T11:27:17.053093202-04:00" level=error msg="error authorizing
context: authorization header required" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=bf905049-e31f-4581-8865-
cc734f01cd5f http.request.method=GET http.request.remoteaddr="172.16.6.1:53765"
http.request.uri="/v2/" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
instance.id=a9e2b8db-aff3-4ba0-baba-5611e953bca9
```

WARN: [DCLu1012 from diagnostic

ClusterRegistry@openshift/origin/pkg/diagnostics/cluster/registry.go:300]

The pod logs for the "docker-registry-2-yzq07" pod belonging to
the "docker-registry" service indicated unknown errors.
This could result in problems with builds or deployments.
Please examine the log entries to determine if there might be
any related problems:

```
    time="2016-09-30T11:17:10-04:00" level=error msg="obsolete configuration detected,
please add openshift registry middleware into registry config file"
    time="2016-09-30T11:17:10-04:00" level=error msg="obsolete configuration detected,
please add openshift storage middleware into registry config file"
    time="2016-09-30T11:22:42.268428221-04:00" level=error msg="error authorizing
context: authorization header required" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=22ce3dfe-3070-4e58-8d32-
df8ef3bc8a12 http.request.method=GET http.request.remoteaddr="172.16.1.1:46819"
http.request.uri="/v2/" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
    time="2016-09-30T11:22:42.417097654-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b05c1b6fd0b2797b0f9ff9
```



```
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=c37e0eba-1723-49b0-a6e8-
cbc53a6b70e1 http.request.method=HEAD http.request.remoteaddr="172.16.1.1:46825"
http.request.uri="/v2/validate/cakephp-
example/blobs/sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b05c1b6fd0b2797b0f9ff9"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=121.549543ms http.response.status=404 http.response.written=157
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
vars.digest="sha256:30cf2e26a24f2a8426cbe8444f8af2ecb7023bd468b05c1b6fd0b2797b0f9ff9"
vars.name="validate/cakephp-example"
    time="2016-09-30T11:22:42.417806241-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a87814f991f2119d4c862
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=5708d5e9-df4e-4b5a-a1fc-
b43912dee898 http.request.method=HEAD http.request.remoteaddr="172.16.1.1:46824"
http.request.uri="/v2/validate/cakephp-
example/blobs/sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a87814f991f2119d4c862"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=124.092336ms http.response.status=404 http.response.written=157
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
vars.digest="sha256:2772ae0d9360d210b6349b96f9e340ec6cb6dafb813a87814f991f2119d4c862"
vars.name="validate/cakephp-example"
    time="2016-09-30T11:22:42.420944307-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:f6db1d2870e85d05aa08cb2d769e18847e5dc321cda780c6d5952f8f52c922f9
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=812c064a-c6ba-4c61-a6ad-
8cf771c23633 http.request.method=HEAD http.request.remoteaddr="172.16.1.1:46822"
http.request.uri="/v2/validate/cakephp-
example/blobs/sha256:f6db1d2870e85d05aa08cb2d769e18847e5dc321cda780c6d5952f8f52c922f9"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=132.329049ms http.response.status=404 http.response.written=157
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
vars.digest="sha256:f6db1d2870e85d05aa08cb2d769e18847e5dc321cda780c6d5952f8f52c922f9"
vars.name="validate/cakephp-example"
    time="2016-09-30T11:22:42.421132455-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0d602c575e32e9804baed
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=9c3840e8-d085-4968-b870-
fd358f36e394 http.request.method=HEAD http.request.remoteaddr="172.16.1.1:46823"
```

```
http.request.uri="/v2/validate/cakephp-
example/blobs/sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0d602c575e32e9804baed"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=130.629462ms http.response.status=404 http.response.written=157
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
vars.digest="sha256:99dd41655d8a45c2fb74f9eeb73e327b3ad4796f0ff0d602c575e32e9804baed"
vars.name="validate/cakephp-example"
    time="2016-09-30T11:23:25.459368181-04:00" level=error msg="response completed
with error" err.code="blob unknown"
err.detail=sha256:a8b77515dce134a9ac9ce832085d214fd7a45439ae14935c7fa2abd788f3fdbb
err.message="blob unknown to registry" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=7b5307e9-4d9b-4dab-97fb-
adffc926c521 http.request.method=HEAD http.request.remoteaddr="172.16.1.1:46861"
http.request.uri="/v2/validate/cakephp-
example/blobs/sha256:a8b77515dce134a9ac9ce832085d214fd7a45439ae14935c7fa2abd788f3fdbb"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=604.990389ms http.response.status=404 http.response.written=157
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
vars.digest="sha256:a8b77515dce134a9ac9ce832085d214fd7a45439ae14935c7fa2abd788f3fdbb"
vars.name="validate/cakephp-example"
    time="2016-09-30T11:23:27.414434085-04:00" level=error msg="response completed
with error" err.code=unknown err.detail="manifest invalid: manifest invalid"
err.message="unknown error" go.version=go1.6.2
http.request.contentType="application/vnd.docker.distribution.manifest.v2+json"
http.request.host="172.30.146.134:5000" http.request.id=438ead9d-5c2a-4401-9bd7-
768d22cd6572 http.request.method=PUT http.request.remoteaddr="172.16.1.1:46870"
http.request.uri="/v2/validate/cakephp-example/manifests/latest"
http.request.useragent="docker/1.10.3 go/go1.6.2 git-commit/5206701-unsupported
kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
http.response.contentType="application/json; charset=utf-8"
http.response.duration=11.024697ms http.response.status=500 http.response.written=136
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a vars.name="validate/cakephp-example"
vars.reference=latest
    time="2016-09-30T11:24:14.322762876-04:00" level=error msg="error authorizing
context: authorization header required" go.version=go1.6.2
http.request.host="172.30.146.134:5000" http.request.id=3aef73ab-5138-42c2-ba3f-
c4388a8a0a13 http.request.method=GET http.request.remoteaddr="172.16.1.1:46883"
http.request.uri="/v2/" http.request.useragent="docker/1.10.3 go/go1.6.2 git-
commit/5206701-unsupported kernel/3.10.0-327.10.1.el7.x86_64 os/linux arch/amd64"
instance.id=e2aa8dd4-89f4-4b51-93bd-8c681e842e6a
```

[Note] Running diagnostic: ClusterRoleBindings

Description: Check that the default ClusterRoleBindings are present and contain the expected subjects

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

Use the `oadm policy reconcile-cluster-role-bindings` command to update the role binding to remove extra subjects.

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

[Note] Running diagnostic: ClusterRoles

Description: Check that the default ClusterRoles are present and contain the expected permissions

[Note] Running diagnostic: ClusterRouterName

Description: Check there is a working router

[Note] Running diagnostic: MasterNode

Description: Check if master is also running node (for Open vSwitch)

WARN: [DClu3004 from diagnostic

MasterNode@openshift/origin/pkg/diagnostics/cluster/master_node.go:175]

Unable to find a node matching the cluster server IP.

This may indicate the master is not also running a node, and is unable to proxy to pods over the Open vSwitch SDN.

[Note] Skipping diagnostic: MetricsApiProxy

Description: Check the integrated heapster metrics can be reached via the API proxy

Because: The heapster service does not exist in the openshift-infra project at this time,
so it is not available for the Horizontal Pod Autoscaler to use as a source of metrics.

[Note] Running diagnostic: NodeDefinitions

Description: Check node records on master

WARN: [DClu0003 from diagnostic

NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definitions.go:112]

Node ip-10-20-4-244.ec2.internal is ready but is marked Unschedulable.

This is usually set manually for administrative reasons.

An administrator can mark the node schedulable with:

```
oadm manage-node ip-10-20-4-244.ec2.internal --schedulable=true
```

While in this state, pods should not be scheduled to deploy on the node.

Existing pods will continue to run until completed or evacuated (see other options for 'oadm manage-node').

WARN: [DClu0003 from diagnostic

```
NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definitions.go:112]
```

```
Node ip-10-20-5-57.ec2.internal is ready but is marked Unschedulable.
```

```
This is usually set manually for administrative reasons.
```

```
An administrator can mark the node schedulable with:
```

```
oadm manage-node ip-10-20-5-57.ec2.internal --schedulable=true
```

```
While in this state, pods should not be scheduled to deploy on the node.
```

```
Existing pods will continue to run until completed or evacuated (see  
other options for 'oadm manage-node').
```

```
WARN: [DC1u0003 from diagnostic
```

```
NodeDefinition@openshift/origin/pkg/diagnostics/cluster/node_definitions.go:112]
```

```
Node ip-10-20-6-152.ec2.internal is ready but is marked Unschedulable.
```

```
This is usually set manually for administrative reasons.
```

```
An administrator can mark the node schedulable with:
```

```
oadm manage-node ip-10-20-6-152.ec2.internal --schedulable=true
```

```
While in this state, pods should not be scheduled to deploy on the node.
```

```
Existing pods will continue to run until completed or evacuated (see  
other options for 'oadm manage-node').
```

```
[Note] Running diagnostic: ServiceExternalIPs
```

```
Description: Check for existing services with ExternalIPs that are disallowed by  
master config
```

```
[Note] Running diagnostic: AnalyzeLogs
```

```
Description: Check for recent problems in systemd service logs
```

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

```
WARN: [DS2005 from diagnostic
```

```
AnalyzeLogs@openshift/origin/pkg/diagnostics/systemd/analyze_logs.go:120]
```

```
Found 'atomic-openshift-node' journald log message:
```

```
W0930 11:12:51.277122 5787 subnets.go:236] Could not find an allocated subnet  
for node: ip-10-20-4-244.ec2.internal, Waiting...
```

```
This warning occurs when the node is trying to request the  
SDN subnet it should be configured with according to the master,  
but either can't connect to it or has not yet been assigned a subnet.
```

```
This can occur before the master becomes fully available and defines a  
record for the node to use; the node will wait until that occurs,  
so the presence of this message in the node log isn't necessarily a  
problem as long as the SDN is actually working, but this message may  
help indicate the problem if it is not working.
```

```
If the master is available and this log message persists, then it may  
be a sign of a different misconfiguration. Check the master's URL in
```

the node kubeconfig.

- * Is the protocol http? It should be https.

- * Can you reach the address and port from the node using `curl -k?`

Info: Checking journalctl logs for 'docker' service

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

WARN: [DH0005 from diagnostic
MasterConfigCheck@openshift/origin/pkg/diagnostics/host/check_master_config.go:52]
Validation of master config file '/etc/origin/master/master-config.yaml' warned:
assetConfig.loggingPublicURL: Invalid value: "": required to view aggregated
container logs in the console
assetConfig.metricsPublicURL: Invalid value: "": required to view cluster metrics
in the console

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

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

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

[Note] Summary of diagnostics execution (version v3.3.0.32):

[Note] Warnings seen: 8



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
```



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...
```



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

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. The configuration is not done on the masters due to the master nodes being unschedulable.

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 role `openshift-emptydir-quota` sets a parameter within the node configuration. The `perFSGroup` setting restricts the ephemeral `emptyDir` volume from growing larger than 512Mi. This empty dir quota is done on the infrastructure and application nodes. The configuration is not done on the masters due to the master nodes being unschedulable.

SSH into the first infrastructure node (`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-server-ose-3.3-rpms/x86_64     Red Hat OpenShift Enterprise 3.3
(RPMs)                                  569
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
```



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=fEAjn7LnZE6v5S0ocCSRvmUWGBNIIIEKbjD9h-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 "user@redhat.com" cannot list all nodes in the cluster
```

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

```
$ oc whoami
```

Once the username has been established, log back into a master node and enable the appropriate permissions for your user. Perform the following step from the first master (ose-master01.sysdeseng.com).

```
# oadm policy add-cluster-role-to-user cluster-admin user@redhat.com
```

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.



Perform the following steps from your the workstation.

```
# oc project default
# oc get all
NAME                                     REVISION           DESIRED           CURRENT           TRIGGERED BY
dc/docker-registry                      1                   2                 2                 config
dc/router                                1                   2                 2                 config
NAME                                     DESIRED           CURRENT           AGE
rc/docker-registry-1                    2                 2                 10m
rc/router-1                              2                 2                 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 Docker Registry

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



Perform the following steps from your the workstation.

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

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



Perform the following steps from the infrastructure node.

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

```
# docker ps | grep ose-docker-registry
073d869f0d5f      openshift3/ose-docker-registry:v3.3.0.32   "/bin/sh -c 'DOCKER_R"   6
hours ago        Up 6 hours                                k8s_registry.90479e7d_docker-
registry-2-jueep_default_d5882b1f-5595-11e6-a247-0eaf3ad438f1_ffc47696
```

```
# docker exec -it d40901ea1240 cat /etc/registryconfig/config.yml
version: 0.1
log:
  level: debug
http:
  addr: :5000
storage:
  cache:
    layerinfo: inmemory
  s3:
    accesskey: "AKIAIP6SRGJHSX3AS2IQ"
    secretkey: "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"
    region: us-east-1
    bucket: "1469667928-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 verifies 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

```
$ 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.



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.



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.



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. Persistent Volumes

`Persistent volumes` (pv) are OpenShift objects that allow for storage to be defined and then claimed by pods to allow for data persistence. The most common `persistent volume` source on `AWS` is `EBS`. `EBS` volumes can only be mounted or claimed by one pod at a time. Mounting of `persistent volumes` is done by using a `persistent volume claim` (pvc). This claim will mount the persistent storage to a specific directory within a pod. This directory is referred to as the `mountPath`.

4.10.1. Node Labels for Persistent Volumes

One important item to remember when creating **persitent volumes** within **AWS** is that the **EBS** volume must be in the same **AZ** as the OpenShift application node that will run the pod. To ensure that a pod is mounted on an application node in the same **AZ** the **node selector** can be modified at the time of project creation. The **node selector** can also be modified on a previously defined project by using `oc edit namespace $project`.

The example below performed on the first OpenShift master will create a project and set the **node selector**. The **AZ** will be used as part of the node label to ensure the pod is created within that **AZ**.

```
$ oc get nodes --show-labels | grep us-east-1c | grep app
ip-10-20-5-31.ec2.internal    Ready          2h
beta.kubernetes.io/instance-type=t2.medium,failure-domain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone=us-east-1c,kubernetes.io/hostname=ip-10-20-5-31.ec2.internal,role=app
$ oadm new-project persistent --node-selector='failure-domain.beta.kubernetes.io/zone=us-east-1c,role=app'
Created project persistent
$ oc project persistent
Now using project "persistent" on server "https://internal-openshift-master.sysdeseng.com".
$ oc describe project persistent
Name:                persistent
Created:             About a minute ago
Labels:              <none>
Annotations:         openshift.io/description=
                    openshift.io/display-name=
                    openshift.io/node-selector=role=app,zone=us-east-1c
                    openshift.io/sa.scc.mcs=s0:c8,c2
                    openshift.io/sa.scc.supplemental-groups=1000060000/10000
                    openshift.io/sa.scc.uid-range=1000060000/10000
Display Name:        <none>
Description:         <none>
Status:              Active
Node Selector:       failure-domain.beta.kubernetes.io/zone=us-east-1c,kubernetes.io/hostname=ip-10-20-5-31.ec2.internal,role=app
Quota:               <none>
Resource limits:     <none>
```

4.10.2. Creating a Persistent Volumes

Log into the **AWS** console. On the dashboard, click on the **EC2** web service and then click **Volumes** under **Elastic Block Store**. Click **Create Volume** and modify the size and the **Availability Zone**. Using the information above the **Availability Zone** of **us-east-1c** will be used. Due to this being an example, the size of the volume will be set to 10Gib. Once the size and **Availability Zone** are defined click **Create**.

Record the **Volume ID** as it will be used when creating the OpenShift **persistent volume**.

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

```
$ oc project persistent
$ vi pv.yaml
apiVersion: "v1"
kind: "PersistentVolume"
metadata:
  name: "persistent"
spec:
  capacity:
    storage: "10Gi"
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    fsType: "ext4"
    volumeID: "vol-02c1cbd0"
$ oc create -f pv.yaml
persistentvolume "persistent" created
$ oc get pv
NAME          CAPACITY  ACCESSMODES  STATUS    CLAIM    REASON    AGE
persistent    10Gi      RWO           Available                47s
```

4.10.3. Creating a Persistent Volumes Claim

The **persistent volume claim** will change the pod from using `EmptyDir` non-persistent storage to storage backed by an **EBS** volume. To claim space from the **persistent volume** a database server will be used to demonstrate a **persistent volume claim**.

```
$ oc new-app --docker-image registry.access.redhat.com/openshift3/mysql-55-rhel7
--name=db -e 'MYSQL_USER=rcook,MYSQL_PASSWORD=d0nth0x,MYSQL_DATABASE=persistent'
```

```
... omitted ...
```

```
$ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
db-1-dwa7o	1/1	Running	0	5m

```
$ oc describe pod db-1-dwa7o
```

```
... omitted ...
```

```
Volumes:
```

```
db-volume-1:
```

```
  Type:  EmptyDir (a temporary directory that shares a pod's lifetime)
```

```
  Medium:
```

```
... omitted ...
```

```
$ oc volume dc/db --add --overwrite --name=db-volume-1 --type=persistentVolumeClaim
--claim-size=10Gi
```

```
persistentvolumeclaims/pvc-ic0mu
```

```
deploymentconfigs/db
```

```
$ oc get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESSMODES	AGE
pvc-ic0mu	Bound	persistent	10Gi	RWO	4s

```
$ oc get pods
```

NAME	READY	STATUS	RESTARTS	AGE
db-2-0srls	1/1	Running	0	23s

```
$ oc describe pod db-2-0srls
```

```
.... omitted ....
```

```
Volumes:
```

```
db-volume-1:
```

```
  Type:  PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same
namespace)
```

```
  ClaimName:  pvc-ic0mu
```

```
  ReadOnly:  false
```

```
.... omitted ....
```

The above has created a database pod with a `persistent volume claim` named `database` and has attached the claim to the previously `EmptyDir` volume.

4.11. 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.

4.11.1. Generate a Master Outage



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.

4.11.2. 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.

4.11.3. Generate an Infrastructure Node outage

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

Confirm Application Accessibility



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.

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.



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

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

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

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

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

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

```
# oc status  
In project default on server https://openshift-master.sysdeseng.com  
  
svc/docker-registry - 172.30.237.147:5000  
  dc/docker-registry deploys docker.io/openshift3/ose-docker-registry:v3.3.0.32  
    deployment #2 deployed 51 minutes ago - 2 pods  
    deployment #1 deployed 53 minutes ago  
  
svc/kubernetes - 172.30.0.1 ports 443, 53->8053, 53->8053  
  
svc/router - 172.30.144.227 ports 80, 443, 1936  
  dc/router deploys docker.io/openshift3/ose-haproxy-router:v3.3.0.32  
    deployment #1 deployed 55 minutes ago - 2 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 prod@redhat.com -e prod@redhat.com -p  
_7yJcnXfeRtAbJVEaQwPwXreEh1V56TkgDwZ6UEUDWw 172.30.110.31:5000
```

```
# oadm policy add-role-to-user admin prod@redhat.com -n openshift  
# oadm policy add-role-to-user system:registry prod@redhat.com  
# oadm policy add-role-to-user system:image-builder prod@redhat.com
```

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
```

Get Location of Router and Registry.



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-gmvdv             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-gmvdv | 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
```

Initiate the Failure and Confirm Functionality



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-gmvdv | grep -i node
Node:          ip-10-30-2-208.ec2.internal/10.30.2.208
```

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

5. 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
<i>PDF generated by Asciidoctor PDF</i>		
<i>Reference Architecture Theme version 1.2</i>		

Appendix B: Contributors

Jason DeTiberus, content provider

Erik Jacobs, content reviewer

Matt Woodson, content reviewer

Rayford Johnson, content reviewer

Roger Lopez, content reviewer

6. 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.

6.1. Inventory

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

```

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

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

[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-app-node01.sysdeseng.com openshift_node_labels="{ 'role': 'app' }"
ose-app-node02.sysdeseng.com openshift_node_labels="{ 'role': 'app' }"

```

6.2. 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
```

6.3. 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 password, 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 '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 region=us-east-1 s3_username=openshift-  
s3-docker-registry byo_bastion=no --keypair=OSE-key rhsm_pool=Red Hat OpenShift Container  
Platform, Standard, 2-Core' playbooks/openshift-install.yaml
```