Reference Architectures 2018

Deploying and Managing OpenShift 3.9 on Amazon Web Services

Last Updated: 2018-10-25

Reference Architectures 2018 Deploying and Managing OpenShift 3.9 on Amazon Web Services

Chris Callegari
refarch-feedback@redhat.com

Abstract

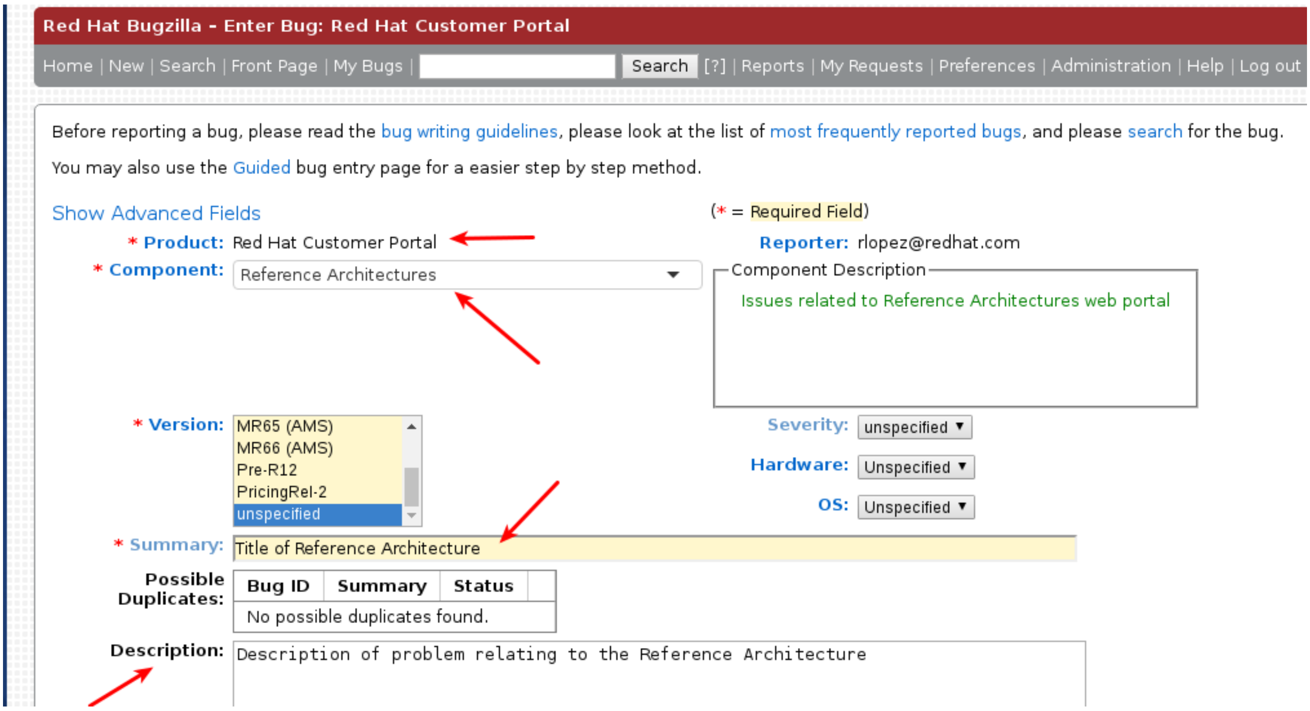
The purpose of this document is to provide guidelines and considerations for deploying and managing Red Hat OpenShift Container Platform on Amazon Web Services.

Table of Contents

COMMENTS AND FEEDBACK

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

When filing a bug report for a reference architecture, create the bug under the Red Hat Customer Portal product and select the Component labeled Reference Architectures. Within the Summary, enter the title of the reference architecture. Within the Description, provide a URL (if available) along with any feedback.



EXECUTIVE SUMMARY

Staying ahead of the needs of an increasingly connected and demanding customer base demands solutions which are not only secure and supported, but robust and scalable, where new features may be delivered in a timely manner. In order to meet these requirements, organizations must provide the capability to facilitate faster development life cycles by managing and maintaining multiple products to meet each of their business needs. Red Hat solutions — for example Red Hat OpenShift Container Platform on Amazon Web Services — simplify this process. Red Hat OpenShift Container Platform, providing a Platform as a Service (PaaS) solution, allows the development, deployment, and management of container-based applications while standing on top of a privately owned cloud by leveraging Amazon Web Services as an Infrastructure as a Service (IaaS).

This reference architecture provides a methodology to deploy a highly available Red Hat OpenShift Container Platform on Amazon Web Services environment by including a step-by-step solution along with best practices on customizing Red Hat OpenShift Container Platform.

This reference architecture is suited for system administrators, Red Hat OpenShift Container Platform administrators, and IT architects building Red Hat OpenShift Container Platform on Amazon Web Services environments.

WHAT IS RED HAT OPENSHIFT CONTAINER PLATFORM

Red Hat OpenShift Container Platform is a Platform as a Service (PaaS) that provides developers and IT organizations with a cloud application platform for deploying new applications on secure, scalable resources with minimal configuration and management overhead. It allows developers to create and deploy applications by delivering a consistent environment for both development and during the runtime life cycle that requires no server management.

NOTE

For more information regarding about Red Hat OpenShift Container Platform visit: Red Hat OpenShift Container Platform Overview

REFERENCE ARCHITECTURE SUMMARY

The deployment of Red Hat OpenShift Container Platform varies among several factors that impact the installation process. Key considerations include:

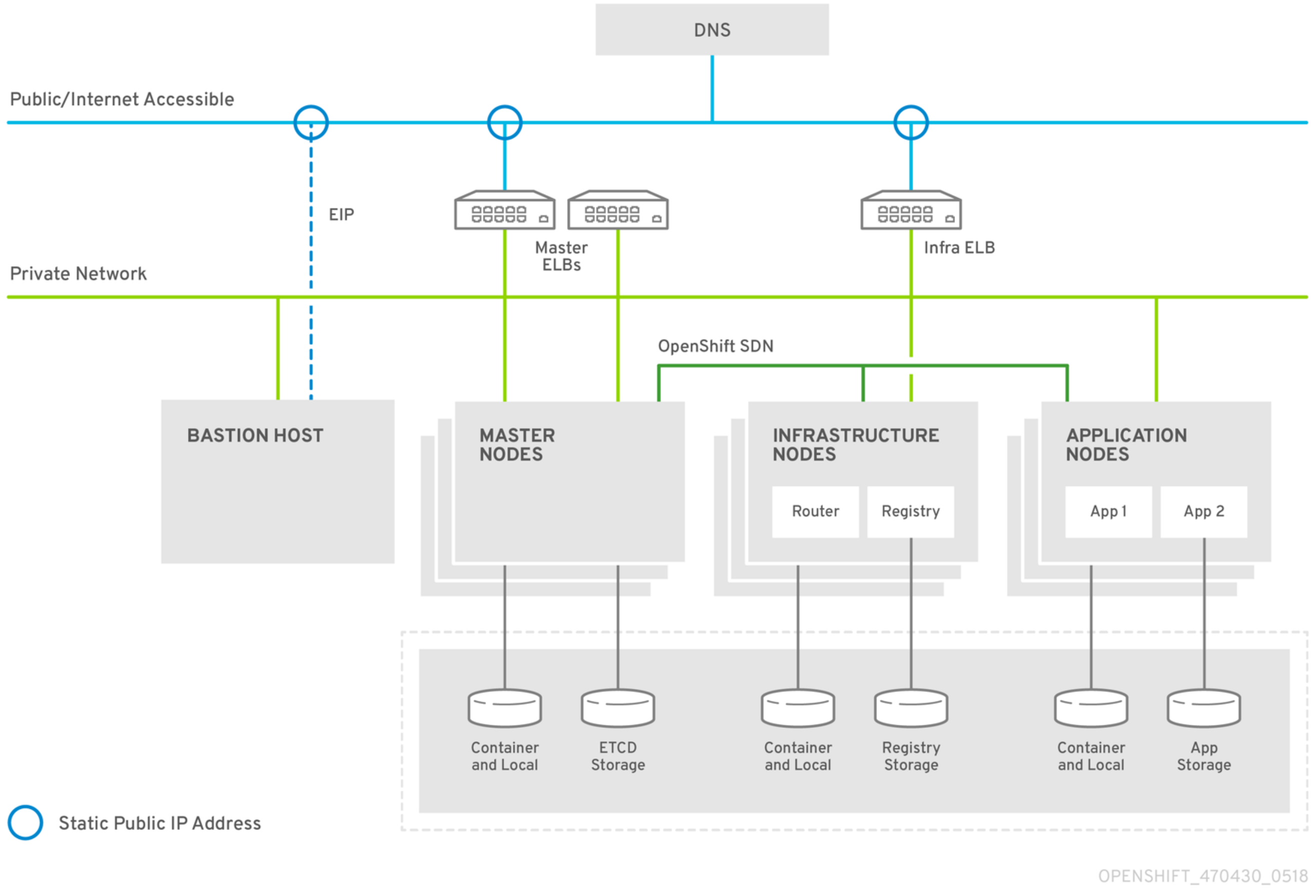
- *Which installation method do you want to use?*
- *How many instances do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other Red Hat technologies?*

For more information regarding the different options in installing an Red Hat OpenShift Container Platform cluster visit: Red Hat OpenShift Container Platform Chapter 2. Installing a Cluster

The initial planning process for this reference architecture answers these questions for this environment as follows:

- *Which installation method do you want to use?* Advanced Installation
- *How many instances do you require in the cluster?* 10
- *Is high availability required?* Yes
- *Which installation type do you want to use: RPM or containerized?* RPM
- *Is my installation supported if integrating with other Red Hat technologies?* Yes

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



The Red Hat OpenShift Container Platform Architecture diagram shows the different components in the reference architecture.

The Red Hat OpenShift Container Platform instances:

- Bastion instance
- Three master instances
- Three infrastructure instances
- Three application instances

CHAPTER 1. COMPONENTS AND CONSIDERATIONS

1.1. AWS INFRASTRUCTURE COMPONENTS

The successful installation of an Red Hat OpenShift Container Platform environment requires the following components to create a highly-available and full featured environment.

1.1.1. Virtual Private Cloud (VPC)

The VPC provides a hosted and virtually isolated network, compute and storage environment. A customer provides a CIDR range at creation time. All components are then built within the global CIDR range.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed in a single VPC with 172.16.0.0/16 as a CIDR range to provide substantial IPv4 space.

NOTE

VPC official documentation

A VPC includes the following subcomponents:

1.1.1.1. Regions and Availability Zones (AZs)

AWS manages many physical data centers within a region of a country. A VPC is an isolated namespace within the many physical datacenters. The decision to use a particular region depends upon customer requirements, regulations, and cost.

This reference environment uses region US-East-1.

Availability zones provide physically separated virtual environments for network, compute, storage environments. AZs exist within a VPC. AWS randomizes the AZs that are presented to an accounts region.

This reference environment uses the first three AZs available to the accounts region.

Deploying Red Hat OpenShift Container Platform in AWS using different zones can be beneficial to avoid single-point-of-failures but there are caveats regarding storage. AWS disks are created within a zone therefore if a Red Hat OpenShift Container Platform node goes down in zone "A" and the pods should be moved to zone "B", the persistent storage cannot be attached to those pods as the disks are in a different availability zone.

It is advised to deploy 1+ application nodes in the same AZ if running an application that requires high availbility. Archtecting applications that can cluster or run on different availabilty zones is also an option to get around this limitation.

NOTE

VPC / Regions and AZs official documentation

1.1.1.2. DHCP

A DHCP server is provided with each VPC to track IP assignment.

NOTE

VPC / DHCP official documentation

1.1.1.3. Internet Gateways (IGW)

An IGW enables internal network traffic to be routed to the Internet.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with a single IGW and attached to the VPC.

NOTE

VPC Networking Components / IGW official documentation

1.1.1.4. NAT Gateways (NatGWs)

NatGW’s provide an AZ-specific highly available network router service to enable VPC internal components in a private subnet to connect to the Internet but prevent the Internet from initiating a connection with those instances.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with three NatGW’s in each AZ providing access to the Internet for infrastructure within the AZ. No cross-AZ traffic is allowed.

NOTE

VPC Networking Components / NatGW official documentation

1.1.1.5. Subnets

VPC subnets are a logical subdivision of the global VPC CIDR range. They exist in a single AZ only. A virtual default gateway is provided for each subnet.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with six subnets spread equally across the three AZs. The first set of three are directly Internet routable. The second set of three allows VPC internal components to route to the Internet via the NatGW. All subnets are deployed as CIDR /20 ranges. This gives each subnet plenty of IPv4 capacity.

NOTE

VPC / Subnet official documentation

1.1.1.6. Network Access Control Lists (ACLs)

ACL’s are IP and CIDR range lists that can be applied to each subnet to provide inbound and outbound network security.

For simplicity, this reference architecture uses a single network ACL allowing all IP’s and all ports applied to all subnets within the VPC.

> **NOTE**
>
> VPC Security / ACL official documentation

1.1.1.7. Route Tables

Each subnet can have a route table applied. This enables cross subnet, Internet, and VPN network traffic routing.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with four RouteTables.

- 1 providing direct Internet access and assigned to the Internet routable subnets
- 3 providing a route to the Internet via the NatGW for each AZ and assigned to the private subnet of that AZ

> **NOTE**
>
> VPC Networking Components / Route Tables official documentation

1.1.1.8. Security Groups (SGs)

SG's are stateful firewalls that can be applied to any network interface within the VPC. IP's, CIDR ranges and other SGs can be added to a list to provide inbound and outbound network security.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with four SG's.

Table 1.1. Security Group association

Instance type	Security groups associated
Bastion	`bastion`
Masters	`mastersg`, `nodesg`
Infra nodes	`infrasg`, `nodesg`
App nodes	`nodesg`
CNS nodes(Optional)	`cnssg`, `nodesg`

1.1.1.8.1. Bastion Security Group ports

The *bastion* SG only needs to allow inbound **`ssh`** and **`ping`** for network connectivity testing.

Table 1.2. Bastion Security Group Ports

Port/Protocol	Service	Remote source	Purpose

ICMP	echo-request	Any	Network connectivity testing
22/TCP	SSH	Any	Secure shell login

1.1.1.8.2. Master Security Group

The *master* SG requires the most complex network access controls. In addition to the ports used by the API and master console, these nodes contain the **`etcd`** servers that form the cluster.

Table 1.3. Master Security Group Ports

Port/Protocol	Service	Remote source	Purpose
53/TCP	DNS	Masters and nodes	Internal name services (3.2+)
53/UDP	DNS	Masters and nodes	Internal name services (3.2+)
2379/TCP	etcd	Masters	Client → Server connections
2380/TCP	etcd	Masters	Server → Server cluster communications
8053/TCP	DNS	Masters and nodes	Internal name services (3.2+)
8053/UDP	DNS	Masters and nodes	Internal name services (3.2+)
443/TCP	HTTPS	Any	Master WebUI and API

1.1.1.8.3. Infrastructure Node SG

The infrastructure nodes run the Red Hat OpenShift Container Platform router and the registry. The *infra* security group must accept inbound connections on the web ports to be forwarded to their destinations.

Table 1.4. Infrastructure Security Group Ports

Port/Protocol	Services	Remote source	Purpose
80/TCP	HTTP	Any	Cleartext application web traffic
443/TCP	HTTPS	Any	Encrypted application web traffic

Port/Protocol	Services	Remote source	Purpose
9200/TCP	ElasticSearch	Any	ElasticSearch API
9300/TCP	ElasticSearch	Any	Internal cluster use

1.1.1.8.4. Node Security Group

The *node* SG is assigned to all instances of Red Hat OpenShift Container Platform. The rules defined only allow for **`ssh`** traffic from the *bastion* host or other nodes, pod to pod communication via SDN traffic, and kubelet communication via Kubernetes.

Table 1.5. Node Security Group Ports

Port/Protocol	Services	Remote source	Purpose
ICMP	echo-request	Any	Network connectivity testing
22/TCP	SSH	SSH SG	Secure shell login
4789/UDP	SDN	Nodes	Pod to pod communications
10250/TCP	kubernetes	Nodes	Kubelet communications

1.1.1.8.5. CNS Node Security Group (Optional)

The CNS nodes require the same ports as the nodes but also require specific ports of the **`glusterfs`** pods. The rules defined only allow for **`ssh`** traffic from the *bastion* host or other nodes, glusterfs communication, pod to pod communication via SDN traffic, and kubelet communication via Kubernetes.

Table 1.6. CNS Security Group Ports

Port/Protocol	Services	Remote source	Purpose
2222/TCP	Gluster	Gluser Nodes	CNS communication
24007/TCP	Gluster	Gluster Nodes	Gluster Daemon
24008/TCP	Gluster	Gluster Nodes	Gluster Management
49152-49664/TCP	Gluster	Gluster Nodes	Gluster Client Ports

NOTE

VPC Security / SG official documentation

1.1.1.9. Elastic IP (EIP)

EIP's are static IPv4 IP's that are Internet addressable. An EIP can be assigned to AWS infrastructure components as needed.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed with a single EIP.

NOTE

VPC Networking Components / EIP official documentation

1.1.1.10. Elastic Network Interfaces (ENI)

ENI's are virtual network interfaces that are assigned to network and compute components within the VPC. ENI's are highly dynamic and scalable.

NOTE

VPC Networking Components / ENI official documentation

1.1.1.11. Elastic Network Adapters (ENA)

ENA's are high performance virtual nic's that give a virtual machine direct access to underlaying hardware via SR-IOV and 10 or 25 Gigabit Ethernet physical networks.

NOTE

Enhanced networking / ENA official documentation

1.1.2. Elastic Loadbalancer (ELB)

ELB's are network loadbalancers that distribute network traffic to EC2's. ELB ENI's scale out based on load. It is essential that adequate IP space is available within the subnet to allow this behavior.

This reference environment consists of the following ELB instances:

- One for the master API that is Internet addressable
- One for the master API that is VPC internal only
- One for infrastructure router service that is used to route a Red Hat OpenShift Container Platform service to it's pods

NOTE

ELB official documentation

1.1.3. Elastic Compute Cloud (EC2)

EC2's are simply virtual machines within the AWS Cloud. They can run with various cpu, memory sizes as well as disk images containing RHEL, Windows and, other operating systems.

This reference environment consists of the following EC2 instances:

- A single t2.medium EC2 to be used as a bastion and ssh proxy
- A set of three m5.2xlarge EC's for master API service spread across three AZs
- A set of three m5.2xlarge EC's the infrastructure router service
- A set of three m5.2xlarge EC's to be for node service and pod hosting
- Optionally a set of three m5.2xlarge EC's can be created to provide CNS storage

This reference environment uses m5 type EC2's which make available 8 vCPU's, 32 Gb Mem, and high performance ENA's. Price and general performance are well balanced with this EC2 type.

NOTE

EC2 official documentation

1.1.4. Elastic Block Storage (EBS)

EBS's are storage volumes that are accessed from EC2's using block protocol. They are AZ-specific and can be dynamically resized and relocated to another EC2s. Snapshots may be taken of an EBS at any time to be used as data archiving and restoration.

This reference environment consists of the following EBS instances:

- Master EC2s each consume 4 100Gb EBS volumes each for root volume, etcd, EmptyDir, and Container storage
- Infra EC2s each consume 2 100Gb EBS volumes each for root volume and Container storage
- Node EC2s each consume 3 100Gb EBS volumes each for root volume, EmptyDir, and Container storage
- Optionally CNS EC2s consume 4 EBS volumes each for root volume, EmptyDir, Container storage, and CNS storage

This reference environment uses gp2 type EBS's. Price and general performance are well balanced with this EBS type.

NOTE

EBS official documentation

1.1.5. Simple Storage Service (S3)

S3 is a highly available object store target that is accessed using https protocol.

This reference environment consists of a single S3 bucket to be used for backend storage for the Red Hat OpenShift Container Platform managed registry.

NOTE

S3 official documentation

1.1.6. Identity and Access Management (IAM)

IAM is a service that provides user and policy management to secure access to AWS API's.

This reference environment consists of the following IAM users:

- IAM user clusterid-admin with a policy to allow Red Hat OpenShift Container Platform to interact with the AWS API
- IAM user clusterid-registry with a policy to allow the Red Hat OpenShift Container Platform managed registry service to interact with the S3 bucket

NOTE

It is possible to use IAM roles and policies to secure Red Hat OpenShift Container Platform master service and node kubelet access to AWS API. The roles are attached directly to the master and nodes EC2 instances. This adds complexity to the infrastructure deployment. There are also security concerns as all users on the instances would have equal access to the AWS API. This refence architecture will use IAM user access keys to avoid complexity and security topic.

NOTE

IAM official documentation

1.2. AWS INFRASTRUCTURE CHARGE MODEL

Operating within the AWS environment is not free of charge. AWS billing is complicated and changes often. This table contains links to AWS component charging pages so that customers can plan their operating expenditures.

AWS Service	Quantity	Charge Model
Route53	1 hosted zone (public) + 1 hosted zone (private) + Many RR records	Charge model
VPC	1	Charge model
VPC AZ	3	Charge model
VPC Subnet	6	None
VPC Route Table	4	None
VPC DHCP Option Set	1	None
VPC NACL	1	None
VPC SG	4	None
VPC NatGW	3	Charge model

AWS Service	Quantity	Charge Model
EIP	1	Charge model
ELB	2 (RHOCP master internal & internet-facing) + 1 (RHOCP infra internet-facing)	Charge model
EC2	3 x m5.xlarge (RHOCP master instances) + 3 x m5.xlarge (RHOCP infra instances) + 3 x m5.xlarge (RHOCP app instances) + 3 x m5.xlarge (RHOCP CNS instances, optional)	Charge model
EBS	1 x 25Gb gp2 (Bastion root vol) + 9 x 100Gb gp2 (root vols) + 3 x 100Gb (master instance etcd vol) + 15 x 100Gb (container storage vol) + 3 x 100Gb (RHOCP instances cns vol, optional)	Charge model
S3	1 bucket for RHOCP	Charge model
IAM	1 user & policy (cloudprovider_kind) + 1 user & policy (registry s3 storage)	None

TIP

AWS Cost Calculator provides fine tune estimates on AWS infrastructure billing.

1.2.1. Route53

Route53 is a managed DNS service that provides internal and Internet accessible hostname resolution.

This reference environment consists of the following Route53 DNS zones:

- refarch.example.com - Internet addressable
- refarch.example.com - VPC internal only

Customers must assign domain delegation to these zones within their DNS infrastructure.

This reference environment consists of the following Rout53 DNS records within each external and internal zone:

- API access - master.refarch.example.com
- app access - *.refarch.example.com
- ELK logging - logging.refarch.example.com

- registry - registry.refarch.example.com

> **NOTE**
>
> Route53 official documentation

1.3. BASTION INSTANCE

Best practices recommend minimizing attack vectors into a system by exposing only those services required by consumers of the system. In the event of failure or a need for manual configuration, systems administrators require further access to internal components in the form of secure administrative back-doors.

In the case of Red Hat OpenShift Container Platform running in a cloud provider context, the entry points to the Red Hat OpenShift Container Platform infrastructure such as the API, Web Console and routers are the only services exposed to the outside. The systems administrators' access from the public network space to the private network is possible with the use of a bastion instance.

A bastion instance is a non-OpenShift instance accessible from outside of the Red Hat OpenShift Container Platform environment, configured to allow remote access via secure shell (**`ssh`**). To remotely access an instance, the systems administrator first accesses the bastion instance, then "jumps" via another **`ssh`** connection to the intended OpenShift instance. The bastion instance may be referred to as a "jump host".

> **NOTE**
>
> As the bastion instance can access all internal instances, it is recommended to take extra measures to harden this instance's security. For more information on hardening the bastion instance, see the official Guide to Securing Red Hat Enterprise Linux 7

Depending on the environment, the bastion instance may be an ideal candidate for running administrative tasks such as the Red Hat OpenShift Container Platform installation playbooks. This reference environment uses the bastion instance for the installation of the Red Hat OpenShift Container Platform.

1.4. RED HAT OPENSHIFT CONTAINER PLATFORM COMPONENTS

Red Hat OpenShift Container Platform comprises of multiple instances running on Amazon Web Services that allow for scheduled and configured OpenShift services and supplementary containers. These containers can have persistent storage, if required, by the application and integrate with optional OpenShift services such as logging and metrics.

1.4.1. OpenShift Instances

Instances running the Red Hat OpenShift Container Platform environment run the **`atomic-openshift-node`** service that allows for the container orchestration of scheduling pods. The following sections describe the different instance and their roles to develop a Red Hat OpenShift Container Platform solution.

1.4.1.1. Master Instances

Master instances run the OpenShift master components, including the API server, controller manager server, and optionally **etcd**. The master manages nodes in its Kubernetes cluster and schedules pods to run on nodes.

NOTE

The master instances are considered nodes as well and run the **atomic-openshift-node** service.

For optimal performance, the **etcd** service should run on the masters instances. When collocating **etcd** with master nodes, at least three instances are required. In order to have a single entry-point for the API, the master nodes should be deployed behind a load balancer.

In order to create master instances with labels, set the following in the inventory file as:

```
... [OUTPUT ABBREVIATED] ...
[etcd]
master1.example.com
master2.example.com
master3.example.com

[masters]
master1.example.com
master2.example.com
master3.example.com

[nodes]
master1.example.com openshift_node_labels="{'region': 'master',
'masterlabel2': 'value2'}"
master2.example.com openshift_node_labels="{'region': 'master',
'masterlabel2': 'value2'}"
master3.example.com openshift_node_labels="{'region': 'master',
'masterlabel2': 'value2'}"
```

Ensure the **openshift_web_console_nodeselector** ansible variable value matches with a master node label in the inventory file. By default, the web_console is deployed to the masters.

NOTE

See the official OpenShift documentation for a detailed explanation on master nodes.

1.4.1.2. Infrastructure Instances

The infrastructure instances run the **atomic-openshift-node** service and host the Red Hat OpenShift Container Platform components such as Registry, Prometheus and Hawkular metrics. The infrastructure instances also run the Elastic Search, Fluentd, and Kibana(**EFK**) containers for aggregate logging. Persistent storage should be available to the services running on these nodes.

Depending on environment requirements at least three infrastructure nodes are required to provide a sharded/highly available aggregated logging service and to ensure that service interruptions do not occur during a reboot.

NOTE

For more infrastructure considerations, visit the official OpenShift documentation.

When creating infrastructure instances with labels, set the following in the inventory file as:

```
... [OUTPUT ABBREVIATED] ...
[nodes]
infra1.example.com openshift_node_labels="{'region': 'infra',
'infralabel1': 'value1'}"
infra2.example.com openshift_node_labels="{'region': 'infra',
'infralabel1': 'value1'}"
infra3.example.com openshift_node_labels="{'region': 'infra',
'infralabel1': 'value1'}"
```

NOTE

The router and registry pods automatically are scheduled on nodes with the label of 'region': 'infra'.

1.4.1.3. Application Instances

The Application (app) instances run the **`atomic-openshift-node`** service. These nodes should be used to run containers created by the end users of the OpenShift service.

When creating node instances with labels, set the following in the inventory file as:

```
... [OUTPUT ABBREVIATED] ...

[nodes]
node1.example.com openshift_node_labels="{'region': 'primary',
'nodelabel2': 'value2'}"
node2.example.com openshift_node_labels="{'region': 'primary',
'nodelabel2': 'value2'}"
node3.example.com openshift_node_labels="{'region': 'primary',
'nodelabel2': 'value2'}"
```

1.4.2. etcd

`etcd` is a consistent and highly-available key value store used as Red Hat OpenShift Container Platform's backing store for all cluster data. **`etcd`** stores the persistent master state while other components watch **`etcd`** for changes to bring themselves into the desired state.

Since values stored in **`etcd`** are critical to the function of Red Hat OpenShift Container Platform, firewalls should be implemented to limit the communication with **`etcd`** nodes. Inter-cluster and client-cluster communication is secured by utilizing x509 Public Key Infrastructure (PKI) key and certificate pairs.

`etcd` uses the RAFT algorithm to gracefully handle leader elections during network partitions and the loss of the current leader. For a highly available Red Hat OpenShift Container Platform deployment, an odd number (starting with three) of **`etcd`** instances are required.

1.4.3. Labels

Labels are key/value pairs attached to objects such as pods. They are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users but do not directly imply semantics to the core system. Labels can also be used to organize and select subsets of objects. Each object can have a set of labels defined at creation time or subsequently added and modified at any time.

NOTE

Each key must be unique for a given object.

```
"labels": {
  "key1" : "value1",
  "key2" : "value2"
}
```

Index and reverse-index labels are used for efficient queries, watches, sorting and grouping in UIs and CLIs, etc. Labels should not be polluted with non-identifying, large and/or structured data. Non-identifying information should instead be recorded using annotations.

1.4.3.1. Labels as Alternative Hierarchy

Service deployments and batch processing pipelines are often multi-dimensional entities (e.g., multiple partitions or deployments, multiple release tracks, multiple tiers, multiple micro-services per tier). Management of these deployments often requires cutting across the encapsulation of strictly hierarchical representations—especially those rigid hierarchies determined by the infrastructure rather than by users. Labels enable users to map their own organizational structures onto system objects in a loosely coupled fashion, without requiring clients to store these mappings.

Example labels:

```
{"release" : "stable", "release" : "canary"}
{"environment" : "dev", "environment" : "qa", "environment" : "production"}
{"tier" : "frontend", "tier" : "backend", "tier" : "cache"}
{"partition" : "customerA", "partition" : "customerB"}
{"track" : "daily", "track" : "weekly"}
```

These are just examples of commonly used labels; the ability exists to develop specific conventions that best suit the deployed environment.

1.4.3.2. Labels as Node Selector

Node labels can be used as node selector where different nodes can be labeled to different use cases. The typical use case is to have nodes running **Red Hat OpenShift Container Platform** infrastructure components like the **Red Hat OpenShift Container Platform** registry, routers, metrics or logging components named "infrastructure nodes" to differentiate them from nodes dedicated to run user applications. Following this use case, the admin can label the "infrastructure nodes" with the label "region=infra" and the application nodes as "region=app". Other uses can be having different hardware in the nodes and have classifications like "type=gold", "type=silver" or "type=bronze".

The scheduler can be configured to use node labels to assign pods to nodes depending on the **`node-selector`**. At times it makes sense to have different types of nodes to run certain pods, the**`node-selector`** can be set to select which labels are used to assign pods to nodes.

1.5. SOFTWARE DEFINED NETWORKING

Red Hat OpenShift Container Platform offers the ability to specify how pods communicate with each other. This could be through the use of Red Hat provided Software-defined networks (SDN) or a third-party SDN.

Deciding on the appropriate internal network for an Red Hat OpenShift Container Platform step is a crucial step. Unfortunately, there is no right answer regarding the appropriate pod network to chose, as this varies based upon the specific scenario requirements on how a Red Hat OpenShift Container Platform environment is to be used.

For the purposes of this reference environment, the Red Hat OpenShift Container Platform **`ovs-networkpolicy`** SDN plug-in is chosen due to its ability to provide pod isolation using Kubernetes **`NetworkPolicy`**. The following section, “OpenShift SDN Plugins”, discusses important details when deciding between the three popular options for the internal networks - **`ovs-multitenant`**, **`ovs-networkpolicy`** and **`ovs-subnet`**.

1.5.1. OpenShift SDN Plugins

This section focuses on multiple plugins for pod communication within Red Hat OpenShift Container Platform using OpenShift SDN. The three plugin options are listed below.

- **`ovs-subnet`** - the original plugin that provides an overlay network created to allow pod-to-pod communication and services. This pod network is created using Open vSwitch (OVS).
- **`ovs-multitenant`** - a plugin that provides an overlay network that is configured using OVS, similar to the **`ovs-subnet`** plugin, however, unlike the **`ovs-subnet`** it provides Red Hat OpenShift Container Platform project level isolation for pods and services.
- **`ovs-networkpolicy`** - a plugin that provides an overlay network that is configured using OVS that provides the ability for Red Hat OpenShift Container Platform administrators to configure specific isolation policies using NetworkPolicy objects[1].

1: https://docs.openshift.com/container-platform/3.9/admin_guide/managing_networking.html#admin-guide-networking-networkpolicy

Network isolation is important, which `OpenShift SDN` to choose?

With the above, this leaves two **`OpenShift SDN`** options: **`ovs-multitenant`** and **`ovs-networkpolicy`**. The reason **`ovs-subnet`** is ruled out is due to it not having network isolation.

While both **`ovs-multitenant`** and **`ovs-networkpolicy`** provide network isolation, the optimal choice comes down to what type of isolation is required. The **`ovs-multitenant`** plugin provides project-level isolation for pods and services. This means that pods and services from different projects cannot communicate with each other.

On the other hand, **`ovs-networkpolicy`** solves network isolation by providing project administrators the flexibility to create their own network policies using Kubernetes **`NetworkPolicy`** objects. This means that by default all pods in a project are accessible from other pods and network endpoints until **`NetworkPolicy`** objects are created. This in turn may allow pods from separate projects to communicate with each other assuming the appropriate **`NetworkPolicy`** is in place.

Depending on the level of isolation required, should determine the appropriate choice when deciding between **`ovs-multitenant`** and **`ovs-networkpolicy`**.

1.6. CONTAINER STORAGE

Container images are stored locally on the nodes running Red Hat OpenShift Container Platform pods. The **container-storage-setup** script uses the **/etc/sysconfig/docker-storage-setup** file to specify the storage configuration.

The **/etc/sysconfig/docker-storage-setup** file should be created before starting the **docker** service, otherwise the storage would be configured using a loopback device. The container storage setup is performed on all hosts running containers, therefore masters, infrastructure, and application nodes.

1.7. PERSISTENT STORAGE

Containers by default offer ephemeral storage but some applications require the storage to persist between different container deployments or upon container migration. **Persistent Volume Claims** (PVC) are used to store the application data. These claims can either be added into the environment by hand or provisioned dynamically using a **StorageClass** object.

1.7.1. Storage Classes

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

1.7.1.1. Persistent Volumes

Persistent volumes (PV) provide pods with non-ephemeral storage by configuring and encapsulating underlying storage sources. A **persistent volume claim** (PVC) abstracts an underlying PV to provide provider agnostic storage to OpenShift resources. A PVC, when successfully fulfilled by the system, mounts the persistent storage to a specific directory (**mountPath**) within one or more pods. From the container point of view, the mountPath is connected to the underlying storage mount points by a **bind-mount**.

1.8. REGISTRY

OpenShift can build containerimages from source code, deploy them, and manage their lifecycle. To enable this, OpenShift provides an internal, integrated registry that can be deployed in the OpenShift environment to manage images.

The registry stores images and metadata. For production environment, persistent storage should be used for the registry, otherwise any images that were built or pushed into the registry would disappear if the pod were to restart.

1.9. AGGREGATED LOGGING

One of the Red Hat OpenShift Container Platform optional components named Red Hat OpenShift Container Platform aggregated logging collects and aggregates logs from the pods running in the Red Hat OpenShift Container Platform cluster as well as **/var/log/messages** on nodes enabling Red Hat

OpenShift Container Platform users to view the logs of projects which they have view access using a web interface.

Red Hat OpenShift Container Platform aggregated logging component it is a modified version of the **ELK** stack composed by a few pods running on the Red Hat OpenShift Container Platform environment:

- **Elasticsearch**: An object store where all logs are stored.
- **Kibana**: A web UI for Elasticsearch.
- **Curator**: Elasticsearch maintenance operations performed automatically on a per-project basis.
- **Fluentd**: Gathers logs from nodes and containers and feeds them to Elasticsearch.

> **NOTE**
>
> Fluentd can be configured to send a copy of the logs to a different log aggregator and/or to a different Elasticsearch cluster, see OpenShift documentation for more information.

Once deployed in the cluster, Fluentd (deployed as a **DaemonSet** on any node with the right labels) gathers logs from all nodes and containers, enriches the log document with useful metadata (e.g. namespace, container_name, node) and forwards them into Elasticsearch, where Kibana provides a web interface to users to be able to view any logs. Cluster administrators can view all logs, but application developers can only view logs for projects they have permission to view. To avoid users to see logs from pods in other projects, the Search Guard plugin for Elasticsearch is used.

A separate Elasticsearch cluster, a separate Kibana, and a separate Curator components can be deployed to form the **OPS cluster** where Fluentd send logs from the **default**, **openshift**, and **openshift-infra** projects as well as **/var/log/messages** on nodes into this different cluster. If the **OPS cluster** is not deployed those logs are hosted in the regular aggregated logging cluster.

Red Hat OpenShift Container Platform aggregated logging components can be customized for longer data persistence, pods limits, replicas of individual components, custom certificates, etc. The customization is provided by the **Ansible** variables as part of the deployment process.

The OPS cluster can be customized as well using the same variables using the suffix ***ops*** as in **openshift_logging_es_ops_pvc_size**.

> **NOTE**
>
> For more information about different customization parameters, see Aggregating Container Logs documentation.

Basic concepts for aggregated logging

- Cluster: Set of Elasticsearch nodes distributing the workload
- Node: Container running an instance of Elasticsearch, part of the cluster.
- Index: Collection of documents (container logs)
- Shards and Replicas: Indices can be split into sets of data containing the primary copy of the documents stored (primary shards) or backups of that primary copies (replica shards). Sharding allows the application to horizontally scaled the information and distributed/paralellized

operations. Replication instead provides high availability and also better search throughput as searches are also executed on replicas.

> **WARNING**
>
> Using NFS storage as a volume or a persistent volume (or via NAS such as Gluster) is not supported for Elasticsearch storage, as Lucene relies on file system behavior that NFS does not supply. Data corruption and other problems can occur.

By default every Elasticsearch pod of the **Red Hat OpenShift Container Platform** aggregated logging components has the role of Elasticsearch master and Elasticsearch data node. If only 2 Elasticsearch pods are deployed and one of the pods fails, all logging stops until the second master returns, so there is no availability advantage to deploy 2 Elasticsearch pods.

> **NOTE**
>
> Elasticsearch shards require their own storage, but Red Hat OpenShift Container Platform **`deploymentconfig`** shares storage volumes between all its pods, therefore every Elasticsearch pod is deployed using a different **`deploymentconfig`** so it cannot be scaled using **`oc scale`**. In order to scale the aggregated logging Elasticsearch replicas after the first deployment, it is required to modify the **`openshift_logging_es_cluser_size`** in the inventory file and re-run the **`openshift-logging.yml`** playbook.

Below is an example of some of the best practices when deploying Red Hat OpenShift Container Platform aggregated logging. **`Elasticsearch`**, **`Kibana`**, and **`Curator`** are deployed on nodes with the label of "region=infra". Specifying the node label ensures that the **`Elasticsearch`** and **`Kibana`** components are not competing with applications for resources. A highly-available environment for Elasticsearch is deployed to avoid data loss, therefore, at least 3 Elasticsearch replicas are deployed and **`openshift_logging_es_number_of_replicas`** parameter is configured to be **`1`** at least. The settings below would be defined in a variable file or static inventory.

```
openshift_logging_install_logging=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=100Gi
openshift_logging_es_cluster_size=3
openshift_logging_es_nodeselector={"region":"infra"}
openshift_logging_kibana_nodeselector={"region":"infra"}
openshift_logging_curator_nodeselector={"region":"infra"}
openshift_logging_es_number_of_replicas=1
```

1.10. AGGREGATED METRICS

Red Hat OpenShift Container Platform has the ability to gather metrics from kubelet and store the values in **`Heapster`**. Red Hat OpenShift Container Platform Metrics provide the ability to view CPU, memory, and network-based metrics and display the values in the user interface. These metrics can allow for the

horizontal autoscaling of pods based on parameters provided by an Red Hat OpenShift Container Platform user. It is important to understand capacity planning when deploying metrics into an Red Hat OpenShift Container Platform environment.

Red Hat OpenShift Container Platform metrics is composed by a few pods running on the Red Hat OpenShift Container Platform environment:

- **Heapster**: Heapster scrapes the metrics for CPU, memory and network usage on every pod, then exports them into Hawkular Metrics.
- **Hawkular Metrics**: A metrics engine that stores the data persistently in a Cassandra database.
- **Cassandra**: Database where the metrics data is stored.

Red Hat OpenShift Container Platform metrics components can be customized for longer data persistence, pods limits, replicas of individual components, custom certificates, etc. The customization is provided by the **`Ansible`** variables as part of the deployment process.

As best practices when metrics are deployed, persistent storage should be used to allow for metrics to be preserved. Node selectors should be used to specify where the Metrics components should run. In the reference architecture environment, the components are deployed on nodes with the label of "region=infra".

```
openshift_metrics_install_metrics=True
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}
```

1.11. INTEGRATION WITH AMAZON WEB SERVICES

When **`cloudprovider_kind = aws`** is set Red Hat OpenShift Container Platform integration with AWS API's will be enabled. This integration will provide the following functions:

- Create EBS volumes
- Attach/Reattach EBS volumes to EC2s (AZ dependent!)
- Create persistent volumes from EC2 attached EBS
- Create persistent volume claims from EBS backed persistent volumes
- Create persistent volumes from EFS
- Create ELB
- Register app EC2s to ELB

When **`cloudprovider_kind = aws`** is set and Service Catalog and Service Broker are installed deeper integration with AWS will be enabled. This integration will provide the following functions:

- Create Simple Queue Service instances (SQS)
- Create Relational Database Service instances (RDS)

- Create Route 53 resources
- Create Simple Storage Services buckets (S3)
- Create Simple Notification Service instances (SNS)
- Create ElastiCache instances
- Create Redshift instances
- Create DynamoDB instances
- Create Elastic MapReduce instances (EMR)

1.12. CONTAINER-NATIVE STORAGE (OPTIONAL)

Container-Native Storage (CNS) provides dynamically provisioned storage for containers on Red Hat OpenShift Container Platform across cloud providers, virtual and bare-metal deployments. **`CNS`** relies on block devices available on the OpenShift nodes and uses software-defined storage provided by Red Hat Gluster Storage. **`CNS`** runs Red Hat Gluster Storage containerized, allowing OpenShift storage pods to spread across the cluster and across different data centers if latency is low between them. **`CNS`** enables the requesting and mounting of **`Gluster`** storage across one or many containers with access modes of either **`ReadWriteMany(RWX)`**, **`ReadOnlyMany(ROX)`** or **`ReadWriteOnce(RWO)`**. **`CNS`** can also be used to host the OpenShift registry.

1.12.1. Prerequisites for Container-Native Storage

Deployment of Container-Native Storage (CNS) on OpenShift Container Platform (OCP) requires at least three OpenShift nodes with at least one 100GB unused block storage device attached on each of the nodes. Dedicating three OpenShift nodes to **`CNS`** allows for the configuration of one **`StorageClass`** object to be used for applications.

If the **`CNS`** instances serve dual roles such as hosting application pods and **`glusterfs`** pods, ensure the instances have enough resources to support both operations. **`CNS`** hardware requirements state that there must be 32GB of RAM per instance.

1.12.2. Firewall and Security Group Prerequisites

The following ports must be open to properly install and maintain **`CNS`**.

NOTE

The nodes used for **`CNS`** also need all of the standard ports an OpenShift node would need.

Table 1.7. CNS - Inbound

Port/Protocol	Services	Remote source	Purpose
111/TCP	Gluster	Gluser Nodes	Portmap
111/UDP	Gluster	Gluser Nodes	Portmap

Port/Protocol	Services	Remote source	Purpose
2222/TCP	Gluster	Gluser Nodes	CNS communication
3260/TCP	Gluster	Gluser Nodes	Gluster Block
24007/TCP	Gluster	Gluster Nodes	Gluster Daemon
24008/TCP	Gluster	Gluster Nodes	Gluster Management
24010/TCP	Gluster	Gluster Nodes	Gluster Block
49152-49664/TCP	Gluster	Gluster Nodes	Gluster Client Ports

CHAPTER 2. RED HAT OPENSHIFT CONTAINER PLATFORM PREREQUISITES

2.1. RED HAT CLOUD ACCESS

AWS provides a RHEL based machine image to use as root volumes for EC2 virtual machines. AWS adds an additional per hour charge to EC2s running the image to cover the price of the license. This enables customers to pay one fee to AWS and not deal with multiple vendors. To other customers this may be a security risk as AWS has the duty of building the machine image and maintaining its lifecycle.

Red Hat provides a bring your own license program for Red Hat Enterprise Linux in AWS. Machine images are presented to participating customer for use as EC2 root volumes. Once signed up Red Hat machine images are then made available in the AMI Private Images inventory.

Red Hat Cloud Access program: https://www.redhat.com/en/technologies/cloud-computing/cloud-access

NOTE

This reference architecture expects customers to be Cloud Access participants

2.2. EXECUTION ENVIRONMENT

RHEL 7 is the only operating system supported by Red Hat OpenShift Container Platform installer therefore provider infrastructure deployment and installer must be run from one of the following locations running RHEL 7:

- Local workstation / server / virtual machine / Bastion
- Jenkins continuous delivery engine

This reference architecture focus's on deploying and installing Red Hat OpenShift Container Platform from local workstation/server/virtual machine and bastion. Registering this host to Red Hat subscription manager will be a requirement to get access to Red Hat OpenShift Container Platform installer and related rpms. Jenkins is out of scope.

2.3. AWS IAM USER

An IAM user with an admin policy and access key is required to interface with AWS API and deploy infrastructure using either AWS CLI or Boto AWS SDK.

Follow this procedure to create an IAM user
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_create.html#id_users_create_console

Follow this procedure to create an access key for an IAM user
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html

2.4. EPEL AND SUPPORTING RPMS

`epel` repo contains supporting rpms to assist with tooling and Red Hat OpenShift Container Platform infrastructure deployment and installation.

Use the following command to enable the `epel` repo:

```
$ yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-
latest-7.noarch.rpm
```

python-pip is useful to install awscli, boto3/boto AWS python bindings.

```
$ yum install -y python-pip
```

jq simplifies data extraction from json output or files.

```
$ yum install -y jq
```

epel is no longer needed and can be removed.

```
$ yum erase epel-release
```

2.5. AWS COMMAND LINE INTERFACE (AWSCLI)

The **awscli** can be used to deploy all of the components associated with this Reference Architecture.

Detailed installation procedure is here: https://docs.aws.amazon.com/cli/latest/userguide/installing.html

This reference architecture supports the following minimum version of awscli:

```
$ aws --version
aws-cli/1.15.16 Python/2.7.5 Linux/3.10.0-862.el7.x86_64 botocore/1.8.50
```

2.6. BOTO3 AWS SDK FOR PYTHON

Boto3 is the AWS SDK for Python, which allows Python developers to write software that makes use of AWS services. Boto provides an easy to use, object-oriented API, and low-level direct service access.

Detailed Boto3 AWS python bindings installation procedure is here:
http://boto3.readthedocs.io/en/latest/guide/quickstart.html#installation

Detailed legacy Boto AWS python bindings installation procedure is here:
http://boto.cloudhackers.com/en/latest/getting_started.html

NOTE

If using Ansible to deploy AWS infrastructure installing boto3 AND legacy boto python bindings is mandatory as some Ansible modules still use the legacy boto AWS python bindings.

Ansible AWS tasks can experience random errors due the speed of execution and AWS API rate limiting. Follow this procedure to ensure Ansible tasks complete sucessfully:

```
$ cat << EOF > ~/.boto
[Boto]
debug = 0
num_retries = 10
EOF
```

This reference architecture supports the following minimum version of boto3:

```
$ pip freeze | grep boto3
boto3==1.5.24
```

This reference architecture supports the following minimum version of boto:

```
$ pip freeze | grep boto
boto==2.48.0
```

2.7. SET UP CLIENT CREDENTIALS

Once the AWS CLI and Boto AWS SDK are installed, the credential needs to be set up.

Detailed IAM user authorization and authentication credentials procedure is here:
https://docs.aws.amazon.com/cli/latest/userguide/cli-config-files.html

Shell environment variables can be an alternative to a static credential file.

```
$ export AWS_ACCESS_KEY_ID = <your aws_access_key_id here>

$ export AWS_SECRET_ACCESS_KEY = <your aws_secret_access_key here>
```

2.8. CLIENT TESTING

A successful **awscli** and credentials test will look similar to the following:

```
$ aws sts get-caller-identity

output:
{
    "Account": "123123123123",
    "UserId": "TH75ISMYR3F4RCHUS3R1D",
    "Arn": "arn:aws:iam::123123123123:user/refarchuser"
}
```

The following error indicates an issue with the AWS IAM user’s access key and local credentials:

```
$ aws sts get-caller-identity

output:
An error occurred (InvalidClientTokenId) when calling the
GetCallerIdentity operation: The security token included in the request is
invalid.
```

A successful boto and credentials will look similar to the following:

```
$ cat << EOF | python
import boto3
print(boto3.client('sts').get_caller_identity()['Arn'])
EOF
```

```
output:
arn:aws:iam::123123123123:user/refarch
```

The following error indicates an issue with the AWS IAM user’s access key and local credentials:

```
output:
....
botocore.exceptions.ClientError: An error occurred (InvalidClientTokenId)
when calling the GetCallerIdentity operation: The security token included
in the request is invalid.
```

2.9. GIT (OPTIONAL)

Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

Install **`git`** package

```
$ yum install -y git
```

NOTE

When using Ansible to deploy AWS infrastructure this step is mandatory as git is used to clone a source code repository.

2.10. SUBSCRIPTION MANAGER

Register instance with Red Hat Subscription Manager and activate yum repos

```
$ subscription-manager register

$ subscription-manager attach \
    --pool NUMBERIC_POOLID

$ subscription-manager repos \
    --disable="*" \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms \
    --enable=rhel-7-server-ansible-2.4-rpms \
    --enable=rhel-7-server-ose-3.9-rpms

$ yum -y install atomic-openshift-utils
```

2.11. CREATE AWS INFRASTRUCTURE FOR RED HAT OPENSHIFT CONTAINER PLATFORM

Infrastructure creation can be executed anywhere provided the AWS API is network accessible and tooling requirements have been met. It is common for local workstations to be used for this step.

NOTE

When using a bastion as the installer execution environment be sure that all of the previous requirements get met. In some cases such as when an AWS EC2 is being used as a bastion there can be a chicken / egg issue where AWS network components and bastion instance to be used as the installer execution environment is not yet available.

2.11.1. CLI

The following **awscli** commands are provided to expose the complexity behind automation tools such Ansible.

Review the following environment variables now and ensure values fit requirements. When values are satisfactory execute the commands in terminal.

```
$ export clusterid="rhocp"
$ export dns_domain="example.com"
$ export region="us-east-1"
$ export cidrvpc="172.16.0.0/16"
$ export cidrsubnets_public=("172.16.0.0/24" "172.16.1.0/24"
"172.16.2.0/24")
$ export cidrsubnets_private=("172.16.16.0/20" "172.16.32.0/20"
"172.16.48.0/20")
$ export ec2_type_bastion="t2.medium"
$ export ec2_type_master="m5.2xlarge"
$ export ec2_type_infra="m5.2xlarge"
$ export ec2_type_node="m5.2xlarge"
$ export rhel_release="rhel-7.5"
```

Create a public private ssh keypair to be used with ssh-agent and ssh authentication on AWS EC2s.

```
$ if [ ! -f ${HOME}/.ssh/${clusterid}.${dns_domain} ]; then
  echo 'Enter ssh key password'
  read -r passphrase
  ssh-keygen -P ${passphrase} -o -t rsa -f
~/.ssh/${clusterid}.${dns_domain}
fi

$ export sshkey=($(cat ~/.ssh/${clusterid}.${dns_domain}.pub))
```

Gather available Availability Zones. The first 3 are used.

```
$ IFS=$' '; export az=($(aws ec2 describe-availability-zones \
    --filters "Name=region-name,Values=${region}" | \
    jq -r '.[][].ZoneName' | \
    head -3 | \
    tr '\n' ' ' | \
    sed -e "s/ $//g"));

$ unset IFS
```

Retrive Red Hat Cloud Access AMI

```
$ export ec2ami=($(aws ec2 describe-images \
```

```
    --region ${region} --owners 309956199498 | \
    jq -r '.Images[] | [.Name,.ImageId] | @csv' | \
    sed -e 's/,/ /g' | \
    sed -e 's/"//g' | \
    grep HVM_GA | \
    grep Access2-GP2 | \
    grep -i ${rhel_release} | \
    sort | \
    tail -1))
```

Deploy IAM users and sleep for 15 sec so that AWS can instantiate the users

```
$ export iamuser=$(aws iam create-user --user-name
${clusterid}.${dns_domain}-admin)

$ export s3user=$(aws iam create-user --user-name
${clusterid}.${dns_domain}-registry)

$ sleep 15
```

Create access key for IAM users

```
$ export iamuser_accesskey=$(aws iam create-access-key --user-name
${clusterid}.${dns_domain}-admin)

$ export s3user_accesskey=$(aws iam create-access-key --user-name
${clusterid}.${dns_domain}-registry)
```

Create and attach policies to IAM users

```
$ cat << EOF > ~/.iamuser_policy_cpk
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:DescribeVolume*",
                "ec2:CreateVolume",
                "ec2:CreateTags",
                "ec2:DescribeInstances",
                "ec2:AttachVolume",
                "ec2:DetachVolume",
                "ec2:DeleteVolume",
                "ec2:DescribeSubnets",
                "ec2:CreateSecurityGroup",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeRouteTables",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:RevokeSecurityGroupIngress",
                "elasticloadbalancing:DescribeTags",
                "elasticloadbalancing:CreateLoadBalancerListeners",
                "elasticloadbalancing:ConfigureHealthCheck",
                "elasticloadbalancing:DeleteLoadBalancerListeners",
                "elasticloadbalancing:RegisterInstancesWithLoadBalancer",
                "elasticloadbalancing:DescribeLoadBalancers",
```

```
                "elasticloadbalancing:CreateLoadBalancer",
                "elasticloadbalancing:DeleteLoadBalancer",
                "elasticloadbalancing:ModifyLoadBalancerAttributes",
                "elasticloadbalancing:DescribeLoadBalancerAttributes"
            ],
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "1"
        }
    ]
}
EOF

$ aws iam put-user-policy \
    --user-name ${clusterid}.${dns_domain}-admin \
    --policy-name Admin \
    --policy-document file://~/.iamuser_policy_cpk

$ cat << EOF > ~/.iamuser_policy_s3
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "arn:aws:s3:::${clusterid}.${dns_domain}-registry",
                "arn:aws:s3:::${clusterid}.${dns_domain}-registry/*"
            ],
            "Effect": "Allow",
            "Sid": "1"
        }
    ]
}
EOF

$ aws iam put-user-policy \
    --user-name ${clusterid}.${dns_domain}-registry \
    --policy-name S3 \
    --policy-document file://~/.iamuser_policy_s3

$ rm -rf ~/.iamuser_policy*
```

Deploy EC2 keypair

```
$ export keypair=$(aws ec2 import-key-pair \
    --key-name ${clusterid}.${dns_domain} \
    --public-key-material file://~/.ssh/${clusterid}.${dns_domain}.pub \
    )
```

Deploy S3 bucket and policy

WARN: If region is a region other than us-east-1 then **`aws s3api create-bucket`** will require **`--create-bucket-configuration LocationConstraint=${region}`**

```
$ export aws_s3bucket=$(aws s3api create-bucket \
    --bucket $(echo ${clusterid}.${dns_domain}-registry) \
    --region ${region} \
    )

$ aws s3api put-bucket-tagging \
    --bucket $(echo ${clusterid}.${dns_domain}-registry) \
    --tagging "TagSet=[{Key=Clusterid,Value=${clusterid}}]"

$ aws s3api put-bucket-policy \
    --bucket $(echo ${clusterid}.${dns_domain}-registry) \
    --policy "\
{ \
  \"Version\": \"2012-10-17\",
  \"Statement\": [ \
    { \
      \"Action\": \"s3:*\", \
      \"Effect\": \"Allow\", \
      \"Principal\": { \
        \"AWS\": \"$(echo ${s3user} | jq -r '.User.Arn')\" \
      }, \
      \"Resource\": \"arn:aws:s3:::$(echo ${aws_s3bucket} | jq -r
'.Location' | sed -e 's/^\///g')\", \
      \"Sid\": \"1\" \
    } \
  ] \
}"
```

Deploy VPC and DHCP server

```
$ export vpc=$(aws ec2 create-vpc --cidr-block ${cidrvpc} | jq -r '.')

$ if [ $region == "us-east-1" ]; then
    export vpcdhcpopts_dnsdomain="ec2.internal"
else
    export vpcdhcpopts_dnsdomain="${region}.compute.internal"
fi

$ export vpcdhcpopts=$(aws ec2 create-dhcp-options \
    --dhcp-configuration " \
    [ \
      { \"Key\": \"domain-name\", \"Values\": [
\"${vpcdhcpopts_dnsdomain}\" ] }, \
      { \"Key\": \"domain-name-servers\", \"Values\": [
\"AmazonProvidedDNS\" ] } \
    ]")

$ aws ec2 modify-vpc-attribute \
    --enable-dns-hostnames \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId')

$ aws ec2 modify-vpc-attribute \
    --enable-dns-support \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId')

$ aws ec2 associate-dhcp-options \
```

```
    --dhcp-options-id $(echo ${vpcdhcpopts} | jq -r
'.DhcpOptions.DhcpOptionsId') \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId')
```

Deploy IGW and attach to VPC

```
$ export igw=$(aws ec2 create-internet-gateway)

$ aws ec2 attach-internet-gateway \
  --internet-gateway-id $(echo ${igw} | jq -r
'.InternetGateway.InternetGatewayId') \
  --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId')
```

Deploy subnets

```
$ for i in $(seq 1 3); do
    export subnet${i}_public="$(aws ec2 create-subnet \
        --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId') \
        --cidr-block ${cidrsubnets_public[$(expr $i - 1 )]} \
        --availability-zone ${az[$(expr $i - 1)]}
    )"
done

$ for i in $(seq 1 3); do
    export subnet${i}_private="$(aws ec2 create-subnet \
        --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId') \
        --cidr-block ${cidrsubnets_private[$(expr $i - 1 )]} \
        --availability-zone ${az[$(expr $i - 1)]}
    )"
done
```

Deploy EIPs

```
$ for i in $(seq 0 3); do
    export eip${i}="$(aws ec2 allocate-address --domain vpc)"
done
```

Deploy NatGW's

```
$ for i in $(seq 1 3); do
    j="eip${i}"
    k="subnet${i}_public"
    export natgw${i}="$(aws ec2 create-nat-gateway \
        --subnet-id $(echo ${!k} | jq -r '.Subnet.SubnetId') \
        --allocation-id $(echo ${!j} | jq -r '.AllocationId') \
    )"
done
```

Deploy RouteTables and routes

```
$ export routetable0=$(aws ec2 create-route-table \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId')
    )
```

```
$ aws ec2 create-route \
    --route-table-id $(echo ${routetable0} | jq -r
'.RouteTable.RouteTableId') \
    --destination-cidr-block 0.0.0.0/0 \
    --nat-gateway-id $(echo ${igw} | jq -r
'.InternetGateway.InternetGatewayId') \
    > /dev/null 2>&1

$ for i in $(seq 1 3); do
    j="subnet${i}_public"
    aws ec2 associate-route-table \
        --route-table-id $(echo ${routetable0} | jq -r
'.RouteTable.RouteTableId') \
        --subnet-id $(echo ${!j} | jq -r '.Subnet.SubnetId')
done

$ for i in $(seq 1 3); do
    export routetable${i}="$(aws ec2 create-route-table \
        --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId') \
    )"
done

$ for i in $(seq 1 3); do
    j="routetable${i}"
    k="natgw${i}"
    aws ec2 create-route \
        --route-table-id $(echo ${!j} | jq -r '.RouteTable.RouteTableId')
\
        --destination-cidr-block 0.0.0.0/0 \
        --nat-gateway-id $(echo ${!k} | jq -r '.NatGateway.NatGatewayId')
\
        > /dev/null 2>&1
done

$ for i in $(seq 1 3); do
    j="routetable${i}"
    k="subnet${i}_private"
    aws ec2 associate-route-table \
        --route-table-id $(echo ${!j} | jq -r '.RouteTable.RouteTableId')
\
        --subnet-id $(echo ${!k} | jq -r '.Subnet.SubnetId')
done
```

Deploy SecurityGroups and rules

```
$ for i in Bastion infra master node; do
    export awssg_$(echo ${i} | tr A-Z a-z)="$(aws ec2 create-security-
group \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId') \
    --group-name ${i} \
    --description ${i})"
done

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_bastion} | jq -r '.GroupId') \
    --ip-permissions '[
```

```
                      {"IpProtocol": "icmp", "FromPort": 8, "ToPort": -1,
"IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
                      {"IpProtocol": "tcp", "FromPort": 22, "ToPort": 22,
"IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                      ]'

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_infra} | jq -r '.GroupId') \
    --ip-permissions '[
                      {"IpProtocol": "tcp", "FromPort": 80, "ToPort": 80,
"IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
                      {"IpProtocol": "tcp", "FromPort": 443, "ToPort":
443, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
                      {"IpProtocol": "tcp", "FromPort": 9200, "ToPort":
9200, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]},
                      {"IpProtocol": "tcp", "FromPort": 9300, "ToPort":
9300, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                      ]'

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_master} | jq -r '.GroupId') \
    --ip-permissions '[
                      {"IpProtocol": "tcp", "FromPort": 443, "ToPort":
443, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                      ]'

$ for i in 2379-2380; do
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_master} | jq -r '.GroupId') \
    --protocol tcp \
    --port $i \
    --source-group $(echo ${awssg_master} | jq -r '.GroupId')
done

$ for i in 2379-2380; do
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_master} | jq -r '.GroupId') \
    --protocol tcp \
    --port $i \
    --source-group $(echo ${awssg_node} | jq -r '.GroupId')
done

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_node} | jq -r '.GroupId') \
    --ip-permissions '[
                      {"IpProtocol": "icmp", "FromPort": 8, "ToPort": -1,
"IpRanges": [{"CidrIp": "0.0.0.0/0"}]}
                      ]'

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_node} | jq -r '.GroupId') \
    --protocol tcp \
    --port 22 \
    --source-group $(echo ${awssg_bastion} | jq -r '.GroupId')

$ for i in 53 2049 8053 10250; do
```

```
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_node} | jq -r '.GroupId') \
    --protocol tcp \
    --port $i \
    --source-group $(echo ${awssg_node} | jq -r '.GroupId')
done

$ for i in 53 4789 8053; do
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_node} | jq -r '.GroupId') \
    --protocol udp \
    --port $i \
    --source-group $(echo ${awssg_node} | jq -r '.GroupId')
done
```

Deploy ELB's

```
$ export elb_masterext=$(aws elb create-load-balancer \
    --load-balancer-name ${clusterid}-public-master \
    --subnets \
      $(echo ${subnet1_public} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet2_public} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet3_public} | jq -r '.Subnet.SubnetId') \
    --listener
Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort=443 \
    --security-groups $(echo ${awssg_master} | jq -r '.GroupId') \
    --scheme internet-facing \
    --tags Key=name,Value=${clusterid}-public-master
Key=Clusterid,Value=${clusterid})

$ aws elb modify-load-balancer-attributes \
    --load-balancer-name ${clusterid}-public-master \
    --load-balancer-attributes "{
        \"CrossZoneLoadBalancing\":{\"Enabled\":true},
        \"ConnectionDraining\":{\"Enabled\":false}
    }"

$ aws elb configure-health-check \
    --load-balancer-name ${clusterid}-public-master \
    --health-check
Target=HTTPS:443/api,HealthyThreshold=3,Interval=5,Timeout=2,UnhealthyThre
shold=2

$ export elb_masterint=$(aws elb create-load-balancer \
    --load-balancer-name ${clusterid}-private-master \
    --subnets \
      $(echo ${subnet1_private} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet2_private} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet3_private} | jq -r '.Subnet.SubnetId') \
    --listener
Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort=443 \
    --security-groups $(echo ${awssg_master} | jq -r '.GroupId') \
    --scheme internal \
    --tags Key=name,Value=${clusterid}-private-master
Key=Clusterid,Value=${clusterid})
```

```
$ aws elb modify-load-balancer-attributes \
    --load-balancer-name ${clusterid}-private-master \
    --load-balancer-attributes "{
        \"CrossZoneLoadBalancing\":{\"Enabled\":true},
        \"ConnectionDraining\":{\"Enabled\":false}
    }"

$ aws elb configure-health-check \
    --load-balancer-name ${clusterid}-private-master \
    --health-check
Target=HTTPS:443/api,HealthyThreshold=3,Interval=5,Timeout=2,UnhealthyThre
shold=2

$ export elb_infraext=$(aws elb create-load-balancer \
    --load-balancer-name ${clusterid}-public-infra \
    --subnets \
      $(echo ${subnet1_public} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet2_public} | jq -r '.Subnet.SubnetId') \
      $(echo ${subnet3_public} | jq -r '.Subnet.SubnetId') \
    --listener \

Protocol=TCP,LoadBalancerPort=80,InstanceProtocol=TCP,InstancePort=80 \

Protocol=TCP,LoadBalancerPort=443,InstanceProtocol=TCP,InstancePort=443 \
    --security-groups $(echo ${awssg_infra} | jq -r '.GroupId') \
    --scheme internet-facing \
    --tags Key=name,Value=${clusterid}-public-infra
Key=Clusterid,Value=${clusterid})

$ aws elb modify-load-balancer-attributes \
    --load-balancer-name ${clusterid}-public-infra \
    --load-balancer-attributes "{
        \"CrossZoneLoadBalancing\":{\"Enabled\":true},
        \"ConnectionDraining\":{\"Enabled\":false}
    }"

$ aws elb configure-health-check \
    --load-balancer-name ${clusterid}-public-infra \
    --health-check
Target=TCP:443,HealthyThreshold=2,Interval=5,Timeout=2,UnhealthyThreshold=
2
```

Deploy Route53 zones and resources

```
$ export route53_extzone=$(aws route53 create-hosted-zone \
  --caller-reference $(date +%s) \
  --name ${clusterid}.${dns_domain} \
  --hosted-zone-config "PrivateZone=False")

$ export route53_intzone=$(aws route53 create-hosted-zone \
  --caller-reference $(date +%s) \
  --name ${clusterid}.${dns_domain} \
  --vpc "VPCRegion=${region},VPCId=$(echo ${vpc} | jq -r '.Vpc.VpcId')" \
  --hosted-zone-config "PrivateZone=True")

$ aws route53 change-resource-record-sets \
```

```
    --hosted-zone-id $(echo ${route53_extzone} | jq -r '.HostedZone.Id' |
sed 's/\/hostedzone\///g') \
    --change-batch "\
{ \
  \"Changes\": [ \
    { \
      \"Action\": \"CREATE\", \
      \"ResourceRecordSet\": { \
        \"Name\": \"master.${clusterid}.${dns_domain}\", \
        \"Type\": \"CNAME\", \
        \"TTL\": 300, \
        \"ResourceRecords\": [ \
          { \"Value\": \"$(echo ${elb_masterext} | jq -r '.DNSName')\" } \
        ] \
      } \
    } \
  ] \
}"

$ aws route53 change-resource-record-sets \
    --hosted-zone-id $(echo ${route53_intzone} | jq -r '.HostedZone.Id' |
sed 's/\/hostedzone\///g') \
    --change-batch "\
{ \
  \"Changes\": [ \
    { \
      \"Action\": \"CREATE\", \
      \"ResourceRecordSet\": { \
        \"Name\": \"master.${clusterid}.${dns_domain}\", \
        \"Type\": \"CNAME\", \
        \"TTL\": 300, \
        \"ResourceRecords\": [ \
          { \"Value\": \"$(echo ${elb_masterint} | jq -r '.DNSName')\" } \
        ] \
      } \
    } \
  ] \
}"

$ aws route53 change-resource-record-sets \
    --hosted-zone-id $(echo ${route53_extzone} | jq -r '.HostedZone.Id' |
sed 's/\/hostedzone\///g') \
    --change-batch "\
{ \
  \"Changes\": [ \
    { \
      \"Action\": \"CREATE\", \
      \"ResourceRecordSet\": { \
        \"Name\": \"*.apps.${clusterid}.${dns_domain}\", \
        \"Type\": \"CNAME\", \
        \"TTL\": 300, \
        \"ResourceRecords\": [ \
          { \"Value\": \"$(echo ${elb_infraext} | jq -r '.DNSName')\" } \
        ] \
      } \
    } \
```

```
  ] \
}"

$ aws route53 change-resource-record-sets \
    --hosted-zone-id $(echo ${route53_intzone} | jq -r '.HostedZone.Id' |
sed 's/\/hostedzone\///g') \
    --change-batch "\
{ \
  \"Changes\": [ \
    { \
      \"Action\": \"CREATE\", \
      \"ResourceRecordSet\": { \
        \"Name\": \"\*.apps.${clusterid}.${dns_domain}\", \
        \"Type\": \"CNAME\", \
        \"TTL\": 300, \
        \"ResourceRecords\": [ \
          { \"Value\": \"$(echo ${elb_infraext} | jq -r '.DNSName')\" } \
        ] \
      } \
    } \
  ] \
}"
```

Create EC2 user-data script

```
$ cat << EOF > /tmp/ec2_userdata.sh
#!/bin/bash
if [ "\$#" -ne 2 ]; then exit 2; fi

printf '%s\n' "#cloud-config"

printf '%s' "cloud_config_modules:"
if [ "\$1" == 'bastion' ]; then
  printf '\n%s\n\n' "- package-update-upgrade-install"
else
  printf '\n%s\n\n' "- package-update-upgrade-install
- disk_setup
- mounts
- cc_write_files"
fi

printf '%s' "packages:"
if [ "\$1" == 'bastion' ]; then
  printf '\n%s\n' "- nmap-ncat"
else
  printf '\n%s\n' "- lvm2"
fi

if [ "\$1" != 'bastion' ]; then
  printf '\n%s' 'write_files:
- content: |
    STORAGE_DRIVER=overlay2
    DEVS=/dev/'
  if [[ "\$2" =~ (c5|c5d|i3.metal|m5) ]]; then
    printf '%s' 'nvme1n1'
  else
```

```
    printf '%s' 'xvdb'
  fi
  printf '\n%s\n' '    VG=dockervg
    CONTAINER_ROOT_LV_NAME=dockerlv
    CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
    CONTAINER_ROOT_LV_SIZE=100%FREE
  path: "/etc/sysconfig/docker-storage-setup"
  permissions: "0644"
  owner: "root"'

  printf '\n%s' 'fs_setup:'
  printf '\n%s' '- label: ocp_emptydir
  filesystem: xfs
  device: /dev/'
  if [[ "\$2" =~ (c5|c5d|i3.metal|m5) ]]; then
    printf '%s\n' 'nvme2n1'
  else
    printf '%s\n' 'xvdc'
  fi
  printf '%s' '  partition: auto'
  if [ "\$1" == 'master' ]; then
    printf '\n%s' '- label: etcd
  filesystem: xfs
  device: /dev/'
    if [[ "\$2" =~ (c5|c5d|i3.metal|m5) ]]; then
      printf '%s\n' 'nvme3n1'
    else
      printf '%s\n' 'xvdd'
    fi
    printf '%s' '  partition: auto'
  fi
  printf '\n'

  printf '\n%s' 'mounts:'
  printf '\n%s' '- [ "LABEL=ocp_emptydir",
"/var/lib/origin/openshift.local.volumes", xfs, "defaults,gquota" ]'
  if [ "\$1" == 'master' ]; then
    printf '\n%s' '- [ "LABEL=etcd", "/var/lib/etcd", xfs,
"defaults,gquota" ]'
  fi
  printf '\n'
fi
EOF

$ chmod u+x /tmp/ec2_userdata.sh
```

Deploy the bastion EC2, sleep for 15 sec while AWS instantiates the EC2, and associate an EIP with the bastion for direct Internet access.

```
$ export ec2_bastion=$(aws ec2 run-instances \
    --image-id ${ec2ami[1]} \
    --count 1 \
    --instance-type ${ec2_type_bastion} \
    --key-name ${clusterid}.${dns_domain} \
    --security-group-ids $(echo ${awssg_bastion} | jq -r '.GroupId') \
    --subnet-id $(echo ${subnet1_public} | jq -r '.Subnet.SubnetId') \
```

```
    --associate-public-ip-address \
    --block-device-mappings "DeviceName=/dev/sda1,Ebs=
{DeleteOnTermination=False,VolumeSize=25}" \
    --user-data "$(/tmp/ec2_userdata.sh bastion ${ec2_type_bastion})" \
    --tag-specifications "ResourceType=instance,Tags=
[{Key=Name,Value=bastion},{Key=Clusterid,Value=${clusterid}}]" \
  )

$ sleep 15

$ aws ec2 associate-address \
    --allocation-id $(echo ${eip0} | jq -r '.AllocationId') \
    --instance-id $(echo ${ec2_bastion} | jq -r '.Instances[].InstanceId')
```

Create the master, infra and node EC2s along with EBS volumes

```
$ for i in $(seq 1 3); do
    j="subnet${i}_private"
    export ec2_master${i}="$(aws ec2 run-instances \
        --image-id ${ec2ami[1]} \
        --count 1 \
        --instance-type ${ec2_type_master} \
        --key-name ${clusterid}.${dns_domain} \
        --security-group-ids $(echo ${awssg_master} | jq -r '.GroupId')
$(echo ${awssg_node} | jq -r '.GroupId') \
        --subnet-id $(echo ${!j} | jq -r '.Subnet.SubnetId') \
        --block-device-mappings \
            "DeviceName=/dev/sda1,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdb,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdc,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdd,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
        --user-data "$(/tmp/ec2_userdata.sh master ${ec2_type_master})" \
        --tag-specifications "ResourceType=instance,Tags=[ \
            {Key=Name,Value=master${i}}, \
            {Key=Clusterid,Value=${clusterid}}, \
            {Key=ami,Value=${ec2ami}}, \
            {Key=kubernetes.io/cluster/${clusterid},Value=${clusterid}}]"
        )"
done

$ for i in $(seq 1 3); do
    j="subnet${i}_private"
    export ec2_infra${i}="$(aws ec2 run-instances \
        --image-id ${ec2ami[1]} \
        --count 1 \
        --instance-type ${ec2_type_infra} \
        --key-name ${clusterid}.${dns_domain} \
        --security-group-ids $(echo ${awssg_infra} | jq -r '.GroupId')
$(echo ${awssg_node} | jq -r '.GroupId') \
        --subnet-id $(echo ${!j} | jq -r '.Subnet.SubnetId') \
        --block-device-mappings \
            "DeviceName=/dev/sda1,Ebs=
```

```
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdb,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdc,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdd,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
        --user-data "$(/tmp/ec2_userdata.sh infra ${ec2_type_infra})" \
        --tag-specifications "ResourceType=instance,Tags=[ \
            {Key=Name,Value=infra${i}}, \
            {Key=Clusterid,Value=${clusterid}}, \
            {Key=ami,Value=${ec2ami}}, \
            {Key=kubernetes.io/cluster/${clusterid},Value=${clusterid}}]"
        )"
done

$ for i in $(seq 1 3); do
    j="subnet${i}_private"
    export ec2_node${i}="$(aws ec2 run-instances \
        --image-id ${ec2ami[1]} \
        --count 1 \
        --instance-type ${ec2_type_node} \
        --key-name ${clusterid}.${dns_domain} \
        --security-group-ids $(echo ${awssg_node} | jq -r '.GroupId') \
        --subnet-id $(echo ${!j} | jq -r '.Subnet.SubnetId') \
        --block-device-mappings \
            "DeviceName=/dev/sda1,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdb,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdc,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
        --user-data "$(/tmp/ec2_userdata.sh node ${ec2_type_node})" \
        --tag-specifications "ResourceType=instance,Tags=[ \
            {Key=Name,Value=node${i}}, \
            {Key=Clusterid,Value=${clusterid}}, \
            {Key=ami,Value=${ec2ami}}, \
            {Key=kubernetes.io/cluster/${clusterid},Value=${clusterid}}]"
        )"
done

$ rm -rf /tmp/ec2_userdata*
```

Register EC2s to ELB's

```
$ export elb_masterextreg=$(aws elb register-instances-with-load-balancer
\
    --load-balancer-name ${clusterid}-public-master \
    --instances \
      $(echo ${ec2_master1} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_master2} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_master3} | jq -r '.Instances[].InstanceId') \
  )

$ export elb_masterintreg=$(aws elb register-instances-with-load-balancer
\
```

```
    --load-balancer-name ${clusterid}-private-master \
    --instances \
      $(echo ${ec2_master1} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_master2} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_master3} | jq -r '.Instances[].InstanceId') \
  )

$ export elb_infrareg=$(aws elb register-instances-with-load-balancer \
    --load-balancer-name ${clusterid}-public-infra \
    --instances \
      $(echo ${ec2_infra1} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_infra2} | jq -r '.Instances[].InstanceId') \
      $(echo ${ec2_infra3} | jq -r '.Instances[].InstanceId') \
  )
```

Create tags on AWS components

```
 aws ec2 create-tags --resources $(echo $vpc | jq -r ".Vpc.VpcId") --tags
Key=Name,Value=${clusterid}; \
 aws ec2 create-tags --resources $(echo ${eip0} | jq -r '.AllocationId') -
-tags Key=Name,Value=bastion; \
 aws ec2 create-tags --resources $(echo ${eip1} | jq -r '.AllocationId') -
-tags Key=Name,Value=${az[0]}; \
 aws ec2 create-tags --resources $(echo ${eip2} | jq -r '.AllocationId') -
-tags Key=Name,Value=${az[1]}; \
 aws ec2 create-tags --resources $(echo ${eip3} | jq -r '.AllocationId') -
-tags Key=Name,Value=${az[2]}; \
 aws ec2 create-tags --resources $(echo ${natgw1} | jq -r
'.NatGateway.NatGatewayId') --tags Key=Name,Value=${az[0]}; \
 aws ec2 create-tags --resources $(echo ${natgw2} | jq -r
'.NatGateway.NatGatewayId') --tags Key=Name,Value=${az[1]}; \
 aws ec2 create-tags --resources $(echo ${natgw3} | jq -r
'.NatGateway.NatGatewayId') --tags Key=Name,Value=${az[2]}; \
 aws ec2 create-tags --resources $(echo ${routetable0} | jq -r
'.RouteTable.RouteTableId') --tags Key=Name,Value=routing; \
 aws ec2 create-tags --resources $(echo ${routetable1} | jq -r
'.RouteTable.RouteTableId') --tags Key=Name,Value=${az[0]}; \
 aws ec2 create-tags --resources $(echo ${routetable2} | jq -r
'.RouteTable.RouteTableId') --tags Key=Name,Value=${az[1]}; \
 aws ec2 create-tags --resources $(echo ${routetable3} | jq -r
'.RouteTable.RouteTableId') --tags Key=Name,Value=${az[2]}; \
 aws ec2 create-tags --resources $(echo ${awssg_bastion} | jq -r
'.GroupId') --tags Key=Name,Value=Bastion; \
 aws ec2 create-tags --resources $(echo ${awssg_bastion} | jq -r
'.GroupId') --tags Key=clusterid,Value=${clusterid}; \
 aws ec2 create-tags --resources $(echo ${awssg_master} | jq -r
'.GroupId') --tags Key=Name,Value=Master; \
 aws ec2 create-tags --resources $(echo ${awssg_master} | jq -r
'.GroupId') --tags Key=clusterid,Value=${clusterid}; \
 aws ec2 create-tags --resources $(echo ${awssg_infra} | jq -r '.GroupId')
--tags Key=Name,Value=Infra; \
 aws ec2 create-tags --resources $(echo ${awssg_infra} | jq -r '.GroupId')
--tags Key=clusterid,Value=${clusterid}; \
 aws ec2 create-tags --resources $(echo ${awssg_node} | jq -r '.GroupId') -
```

```
-tags Key=Name,Value=Node; \
 aws ec2 create-tags --resources $(echo ${awssg_node} | jq -r '.GroupId') -
-tags Key=clusterid,Value=${clusterid}
```

Create configuration files. These files are used to assist with installing

```
$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}
#<!-- BEGIN OUTPUT -->
Host bastion
  HostName                  $(echo ${eip0} | jq -r '.PublicIp')
  User                      ec2-user
  StrictHostKeyChecking     no
  ProxyCommand              none
  CheckHostIP               no
  ForwardAgent              yes
  ServerAliveInterval       15
  TCPKeepAlive              yes
  ControlMaster             auto
  ControlPath               ~/.ssh/mux-%r@%h:%p
  ControlPersist            15m
  ServerAliveInterval       30
  IdentityFile              ~/.ssh/${clusterid}.${dns_domain}

Host *.compute-1.amazonaws.com
  ProxyCommand              ssh -w 300 -W %h:%p bastion
  user                      ec2-user
  StrictHostKeyChecking     no
  CheckHostIP               no
  ServerAliveInterval       30
  IdentityFile              ~/.ssh/${clusterid}.${dns_domain}

Host *.ec2.internal
  ProxyCommand              ssh -w 300 -W %h:%p bastion
  user                      ec2-user
  StrictHostKeyChecking     no
  CheckHostIP               no
  ServerAliveInterval       30
  IdentityFile              ~/.ssh/${clusterid}.${dns_domain}
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-domaindelegation
#<!-- BEGIN OUTPUT -->
$(echo ${route53_extzone} | jq -r '.DelegationSet.NameServers[]')
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-cpkuser_access_key
#<!-- BEGIN OUTPUT -->
openshift_cloudprovider_aws_access_key=$(echo ${iamuser_accesskey} | jq -r
'.AccessKey.AccessKeyId')
openshift_cloudprovider_aws_secret_key=$(echo ${iamuser_accesskey} | jq -r
'.AccessKey.SecretAccessKey')
#<!-- END OUTPUT -->
EOF
```

```
$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-cpk
#<!-- BEGIN OUTPUT -->
openshift_cloudprovider_kind=aws
openshift_clusterid=${clusterid}
EOF
cat ~/.ssh/config-${clusterid}.${dns_domain}-cpkuser_access_key | \
    grep -v 'OUTPUT -->' >> \
    ~/.ssh/config-${clusterid}.${dns_domain}-cpk
cat << EOF >> ~/.ssh/config-${clusterid}.${dns_domain}-cpk
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-s3user_access_key
#<!-- BEGIN OUTPUT -->
openshift_hosted_registry_storage_s3_accesskey=$(echo ${s3user_accesskey}
| jq -r '.AccessKey.AccessKeyId')
openshift_hosted_registry_storage_s3_secretkey=$(echo ${s3user_accesskey}
| jq -r '.AccessKey.SecretAccessKey')
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-s3
#<!-- BEGIN OUTPUT -->
openshift_hosted_manage_registry=true
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
EOF
cat ~/.ssh/config-${clusterid}.${dns_domain}-s3user_access_key | \
    grep -v 'OUTPUT -->' >> \
    ~/.ssh/config-${clusterid}.${dns_domain}-s3
cat << EOF >> ~/.ssh/config-${clusterid}.${dns_domain}-s3
openshift_hosted_registry_storage_s3_bucket=${clusterid}.${dns_domain}-
registry
openshift_hosted_registry_storage_s3_region=${region}
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_registry_replicas=3
openshift_hosted_registry_selector='region=infra'
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-urls
#<!-- BEGIN OUTPUT -->
openshift_master_default_subdomain=apps.${clusterid}.${dns_domain}
openshift_master_cluster_hostname=master.${clusterid}.${dns_domain}
openshift_master_cluster_public_hostname=master.${clusterid}.${dns_domain}
#<!-- END OUTPUT -->
EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-hosts
[masters]
$(echo ${ec2_master1} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'master'}"
```

```
$(echo ${ec2_master2} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'master'}"
$(echo ${ec2_master3} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'master'}"

[etcd]

[etcd:children]
masters

[nodes]
$(echo ${ec2_node1} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'apps'}"
$(echo ${ec2_node2} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'apps'}"
$(echo ${ec2_node3} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'apps'}"
$(echo ${ec2_infra1} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'infra', 'zone': 'default'}"
$(echo ${ec2_infra2} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'infra', 'zone': 'default'}"
$(echo ${ec2_infra3} | jq -r '.Instances[].PrivateDnsName')
openshift_node_labels="{'region': 'infra', 'zone': 'default'}"

[nodes:children]
masters
EOF
```

2.11.2. Ansible

GitHub OpenShift openshift-ansible-contrib repository contains an Ansible playbook that provides a friendly and repeatable deployment experience.

Clone openshift-ansible-contrib repository then enter the reference architecture 3.9 directory.

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

$ cd openshift-ansible-contrib/reference-architecture/3.9
```

Create a simple Ansible inventory

```
$ sudo vi /etc/ansible/hosts
[local]
127.0.0.1

[local:vars]
ansible_connection=local
ansible_become=False
```

Review **playbooks/vars/main.yaml** now and ensure values fit requirements.

When values are satisfactory execute **deploy_aws.yaml** play to deploy AWS infrastructure.

```
$ ansible-playbook playbooks/deploy_aws.yaml
```

NOTE

Troubleshooting the **`deploy_aws.yaml`** play is simple. It is self repairing. If any errors are encountered simply rerun play.

2.12. CREATE AWS INFRASTRUCTURE FOR RED HAT OPENSHIFT CONTAINER PLATFORM CNS (OPTIONAL)

2.12.1. CLI

Set environment variables to be sourced by other commands. These values can be modified to fit any environment.

```
$ export ec2_type_cns="m5.2xlarge"
```

Deploy SecurityGroup and rules

```
$ export awssg_cns=$(aws ec2 create-security-group \
    --vpc-id $(echo ${vpc} | jq -r '.Vpc.VpcId') \
    --group-name cns \
    --description "cns")

$ for i in 111 2222 3260 24007-24008 24010 49152-49664; do
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_cns} | jq -r '.GroupId') \
    --protocol tcp \
    --port $i \
    --source-group $(echo ${awssg_cns} | jq -r '.GroupId')
done

$ for i in 3260 24007-24008 24010 49152-49664; do
    aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_cns} | jq -r '.GroupId') \
    --protocol tcp \
    --port $i \
    --source-group $(echo ${awssg_node} | jq -r '.GroupId')
done

$ aws ec2 authorize-security-group-ingress \
    --group-id $(echo ${awssg_cns} | jq -r '.GroupId') \
    --protocol udp \
    --port 111 \
    --source-group $(echo ${awssg_cns} | jq -r '.GroupId')
```

Create node EC2 instances along with EBS volumes

```
$ for i in 1 2 3; do
    j="subnet${i}_private"
    export ec2_cns${i}="$(aws ec2 run-instances \
        --image-id ${ec2ami[1]} \
        --count 1 \
        --instance-type ${ec2_type_cns} \
        --key-name ${clusterid}.${dns_domain} \
        --security-group-ids $(echo ${awssg_cns} | jq -r '.GroupId')
```

```
$(echo ${awssg_node} | jq -r '.GroupId') \
        --subnet-id $(echo ${!j} | jq -r '.Subnet.SubnetId') \
        --block-device-mappings \
            "DeviceName=/dev/sda1,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdb,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdc,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
            "DeviceName=/dev/xvdd,Ebs=
{DeleteOnTermination=False,VolumeSize=100}" \
        --user-data "$(/tmp/ec2_userdata.sh node ${ec2_type_node})" \
        --tag-specifications "ResourceType=instance,Tags=[ \
            {Key=Name,Value=cns${i}}, \
            {Key=Clusterid,Value=${clusterid}}, \
            {Key=ami,Value=${ec2ami}}, \
            {Key=kubernetes.io/cluster/${clusterid},Value=${clusterid}}]"
        )"
done
```

Create configuration files. These files are used to assist with installing Red Hat OpenShift Container Platform.

```
$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-hostscns
$(echo ${ec2_cns1} | jq -r '.Instances[].PrivateDnsName')
openshift_schedulable=True
$(echo ${ec2_cns2} | jq -r '.Instances[].PrivateDnsName')
openshift_schedulable=True
$(echo ${ec2_cns3} | jq -r '.Instances[].PrivateDnsName')
openshift_schedulable=True

EOF

$ cat << EOF > ~/.ssh/config-${clusterid}.${dns_domain}-hostsgfs
[glusterfs]
$(echo ${ec2_cns1} | jq -r '.Instances[].PrivateDnsName')
glusterfs_devices='[ "/dev/nvme3n1" ]'
$(echo ${ec2_cns2} | jq -r '.Instances[].PrivateDnsName')
glusterfs_devices='[ "/dev/nvme3n1" ]'
$(echo ${ec2_cns3} | jq -r '.Instances[].PrivateDnsName')
glusterfs_devices='[ "/dev/nvme3n1" ]'
EOF
```

2.12.2. Ansible

Run **deploy_aws_cns.yaml** play to deploy additional infrastructure for CNS.

```
$ ansible-playbook playbooks/deploy_aws_cns.yaml
```

NOTE

deploy_aws_cns.yaml playbook requires **deploy_aws.yaml** to be previously run. Troubleshooting the **deploy_aws_cns.yaml** play is simple. It is self repairing. If any errors are encountered simply rerun play.

2.13. BASTION CONFIGURATION

It is often preferrable to use a bastion host as an ssh proxy for security. Awscli and Ansible both output a ssh configuration file to enable direct ssh connections to Red Hat OpenShift Container Platform instances.

Perform the following procedure to set configuration:

```
$ mv ~/.ssh/config ~/.ssh/config-orig

$ ln -s ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN > ~/.ssh/config

$ chmod 400 ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN >
```

Perform the following procedure to ensure ssh-agent is running and local ssh key can be used to proxy connections to target EC2 instances:

```
$ if [ ! "$(env | grep SSH_AGENT_PID)" ] || [ ! "$(ps -ef | grep -v grep |
grep ${SSH_AGENT_PID})" ]; then
  rm -rf ${SSH_AUTH_SOCK} 2> /dev/null
  unset SSH_AUTH_SOCK
  unset SSH_AGENT_PID
  pkill ssh-agent
  export sshagent=$(nohup ssh-agent &)
  export sshauthsock=$(echo ${sshagent} | awk -F'; ' {'print $1'})
  export sshagentpid=$(echo ${sshagent} | awk -F'; ' {'print $3'})
  export ${sshauthsock}
  export ${sshagentpid}
  for i in sshagent sshauthsock sshagentpid; do
    unset $i
  done
fi

$ export sshkey=($(cat ~/.ssh/< CLUSTERID >.< DNS_DOMAIN >.pub))

$ IFS=$'\n'; if [ ! $(ssh-add -L | grep ${sshkey[1]}) ]; then
  ssh-add ~/.ssh/< CLUSTERID >.< DNS_DOMAIN >
fi

$ unset IFS
```

Verify < CLUSTERID >.< DNS_DOMAIN > ssh key is added to ssh-agent

```
$ ssh-add -l

$ ssh-add -L
```

2.14. OPENSHIFT PREPARATIONS

Once the instances have been deployed and the **`~/.ssh/config`** file reflects the deployment the following steps should be performed to prepare for the installation of OpenShift.

2.14.1. Public domain delegation

To access Red Hat OpenShift Container Platform on AWS from the public Internet delegation for the subdomain must be setup using the following information.

DNS subdomain	Route53 public zone NS records
< CLUSTERID >.< DNS_DOMAIN >	See file ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN >-domaindelegation for correct values

WARN: DNS service provder domain delegation is out of scope of this guide. A customer will need to follow their providers best practise in delegation setup.

2.14.2. OpenShift Authentication

Red Hat OpenShift Container Platform provides the ability to use many different authentication platforms.

Detailed listing of other authentication providers are available at Configuring Authentication and User Agent.

For this reference architecture Google's OpenID Connect Integration is used. When configuring the authentication, the following parameters must be added to the ansible inventory. An example is shown below.

```
openshift_master_identity_providers=[{'name': 'google', 'challenge':
'false', 'login': 'true', 'kind': 'GoogleIdentityProvider',
'mapping_method': 'claim', 'clientID': '246358064255-
5ic2e4b1b9ipfa7hddfkhuf8s6eq2rfj.apps.googleusercontent.com',
'clientSecret': 'Za3PWZg7gQxM26HBljgBMBBF', 'hostedDomain': 'redhat.com'}]
```

2.14.3. Openshift-ansible Installer Inventory

This section provides an example inventory file required for an advanced installation of Red Hat OpenShift Container Platform.

The inventory file contains both variables and instances used for the configuration and deployment of Red Hat OpenShift Container Platform.

```
$ sudo vi /etc/ansible/hosts
[OSEv3:children]
masters
etcd
nodes

[OSEv3:vars]
debug_level=2
ansible_user=ec2-user
ansible_become=yes
openshift_deployment_type=openshift-enterprise
openshift_release=v3.9
openshift_master_api_port=443
openshift_master_console_port=443
openshift_portal_net=172.30.0.0/16
os_sdn_network_plugin_name='redhat/openshift-ovs-networkpolicy'
openshift_master_cluster_method=native
```

```
container_runtime_docker_storage_setup_device=/dev/nvme1n1
openshift_node_local_quota_per_fsgroup=512Mi
osm_use_cockpit=true
openshift_hostname_check=false
openshift_examples_modify_imagestreams=true
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}

openshift_hosted_router_selector='region=infra'
openshift_hosted_router_replicas=3

# UPDATE TO CORRECT IDENTITY PROVIDER
openshift_master_identity_providers=[{'name': 'google', 'challenge':
'false', 'login': 'true', 'kind': 'GoogleIdentityProvider',
'mapping_method': 'claim', 'clientID': '246358064255-
5ic2e4b1b9ipfa7hddfkhuf8s6eq2rfj.apps.googleusercontent.com',
'clientSecret': 'Za3PWZg7gQxM26HBljgBMBBF', 'hostedDomain': 'redhat.com'}]

# UPDATE USING VALUES FOUND IN awscli env vars or
~/playbooks/var/main.yaml
# SEE ALSO FILE ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN >-urls
openshift_master_default_subdomain=apps.< CLUSTERID >.< DNS_DOMAIN >
openshift_master_cluster_hostname=master.< CLUSTERID >.< DNS_DOMAIN >
openshift_master_cluster_public_hostname=master.< CLUSTERID >.< DNS_DOMAIN
>

# UPDATE USING VALUES FOUND IN FILE ~/.ssh/config-< CLUSTERID >.<
DNS_DOMAIN >-cpk
openshift_cloudprovider_kind=aws
openshift_clusterid=refarch
openshift_cloudprovider_aws_access_key=UPDATEACCESSKEY
openshift_cloudprovider_aws_secret_key=UPDATESECRETKEY

# UPDATE USING VALUES FOUND IN FILE ~/.ssh/config-< CLUSTERID >.<
DNS_DOMAIN >-s3
openshift_hosted_manage_registry=true
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_accesskey=UPDATEACCESSKEY
openshift_hosted_registry_storage_s3_secretkey=UPDATESECRETKEY
openshift_hosted_registry_storage_s3_bucket=refarch-registry
openshift_hosted_registry_storage_s3_region=REGION
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_registry_replicas=3
openshift_hosted_registry_selector='region=infra'

# Aggregated logging
openshift_logging_install_logging=true
openshift_logging_storage_kind=dynamic
openshift_logging_storage_volume_size=25Gi
openshift_logging_es_cluster_size=3

# Metrics
```

```
openshift_metrics_install_metrics=true
openshift_metrics_storage_kind=dynamic
openshift_metrics_storage_volume_size=25Gi

openshift_enable_service_catalog=true

template_service_broker_install=true

# UPDATE USING HOSTS FOUND IN FILE ~/.ssh/config-< CLUSTERID >.<
DNS_DOMAIN >-hosts
[masters]
ip-172-16-30-15.ec2.internal openshift_node_labels="{'region': 'master'}"
ip-172-16-47-176.ec2.internal openshift_node_labels="{'region': 'master'}"
ip-172-16-48-251.ec2.internal openshift_node_labels="{'region': 'master'}"

[etcd]

[etcd:children]
masters

# UPDATE USING HOSTS FOUND IN FILE ~/.ssh/config-< CLUSTERID >.<
DNS_DOMAIN >-hosts
[nodes]
ip-172-16-16-239.ec2.internal openshift_node_labels="{'region': 'apps'}"
ip-172-16-37-179.ec2.internal openshift_node_labels="{'region': 'apps'}"
ip-172-16-60-134.ec2.internal openshift_node_labels="{'region': 'apps'}"
ip-172-16-17-209.ec2.internal openshift_node_labels="{'region': 'infra',
'zone': 'default'}"
ip-172-16-46-136.ec2.internal openshift_node_labels="{'region': 'infra',
'zone': 'default'}"
ip-172-16-56-149.ec2.internal openshift_node_labels="{'region': 'infra',
'zone': 'default'}"

[nodes:children]
masters
```

NOTE

Advanced installations example configurations and options are provided in **/usr/share/doc/openshift-ansible-docs-3.9.*/docs/example-inventories directory**.

2.14.4. CNS Inventory (Optional)

If CNS is used in the OpenShift installation specific variables must be set in the inventory.

```
$ sudo vi /etc/ansible/hosts
# ADD glusterfs to section [OSEv3:children]
[OSEv3:children]
masters
etcd
nodes
glusterfs

[OSEv3:vars]
```

```
....omitted...

# CNS storage cluster
openshift_storage_glusterfs_namespace=glusterfs
openshift_storage_glusterfs_storageclass=true
openshift_storage_glusterfs_storageclass_default=false
openshift_storage_glusterfs_block_deploy=false
openshift_storage_glusterfs_block_host_vol_create=false
openshift_storage_glusterfs_block_host_vol_size=80
openshift_storage_glusterfs_block_storageclass=false
openshift_storage_glusterfs_block_storageclass_default=false

....omitted...

# ADD CNS HOSTS TO SECTION [nodes]
# SEE INVENTORY IN FILE ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN >-
hostscns
[nodes]
....omitted...
ip-172-16-17-130.ec2.internal openshift_schedulable=True
ip-172-16-40-219.ec2.internal openshift_schedulable=True
ip-172-16-63-212.ec2.internal openshift_schedulable=True

# ADD SECTION [glusterfs]
# SEE INVENTORY IN FILE ~/.ssh/config-< CLUSTERID >.< DNS_DOMAIN >-
hostsgfs
[glusterfs]
ip-172-16-17-130.ec2.internal glusterfs_devices='[ "/dev/nvme3n1" ]'
ip-172-16-40-219.ec2.internal glusterfs_devices='[ "/dev/nvme3n1" ]'
ip-172-16-63-212.ec2.internal glusterfs_devices='[ "/dev/nvme3n1" ]'
```

2.14.5. EC2 instances

2.14.5.1. Test network connectivity

Ansible ping is used to test if the local environment can reach all instances. If there is a failure here then please check ssh config or network connectivity for the instance.

```
$ ansible nodes -b -m ping
```

2.14.5.2. Remove RHUI yum repos

Existing Red Hat Update Infrastructure yum repos must be disabled to avoid repository and rpm dependency conflicts with Red Hat OpenShift Container Platform repository and rpms.

```
$ ansible nodes -b -m command -a "yum-config-manager \
    --disable 'rhui-REGION-client-config-server-7' \
    --disable 'rhui-REGION-rhel-server-rh-common' \
    --disable 'rhui-REGION-rhel-server-releases'"
```

2.14.5.3. Node Registration

Now that the inventory has been created the nodes must be subscribed using **`subscription-manager`**.

The ad-hoc playbook below uses the **`redhat_subscription`** module to register the instances. The first example uses the numeric pool value for the OpenShift subscription. The second uses an activation key and organization.

```
$ ansible nodes -b -m redhat_subscription -a \
    "state=present username=USER password=PASSWORD
pool_ids=NUMBERIC_POOLID"
```

OR

```
$ ansible nodes -b -m redhat_subscription -a \
    "state=present activationkey=KEY org_id=ORGANIZATION"
```

2.14.5.4. Repository Setup

Once the instances are registered the proper repositories must be assigned to the instances to allow for packages for Red Hat OpenShift Container Platform to be installed.

```
$ ansible nodes -b -m shell -a \
    'subscription-manager repos --disable="*" \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.9-rpms" \
    --enable="rhel-7-fast-datapath-rpms"'
```

2.14.6. EmptyDir Storage

During the deployment of instances an extra volume is added to the instances for **`EmptyDir`** storage. EmptyDir provides ephemeral (as opposed to persistent) storage for containers. The volume is used to help ensure that the */var* volume does not get filled by containers using the storage.

The user-data script attached to EC2 instances automatically provisioned filesystem, added an entry to fstab, and mounted this volume.

2.14.7. `etcd` Storage

During the deployment of the master instances an extra volume was added to the instances for **`etcd`** storage. Having the separate disks specifically for **`etcd`** ensures that all of the resources are available to the **`etcd`** service such as i/o and total disk space.

The user-data script attached to EC2 instances automatically provisioned filesystem, added an entry to fstab, and mounted this volume.

2.14.8. Container Storage

The prerequisite playbook provided by the OpenShift Ansible RPMs configures container storage and installs any remaining packages for the installation.

```
$ ansible-playbook \
    /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

CHAPTER 3. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM

With the prerequisites met, the focus shifts to the installation Red Hat OpenShift Container Platform. The installation and configuration is done via a series of **`Ansible`** playbooks and roles provided by the **`atomic-openshift`** packages.

Run the installer playbook to install Red Hat OpenShift Container Platform:

```
$ ansible-playbook \
    /usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
```

The playbooks runs through the complete process of installing Red Hat OpenShift Container Platform and reports a playbook recap showing the number of changes and errors (if any). The Ansible PLAY RECAP report should look something similar to the following.

```
PLAY RECAP
ip-172-16-22-77.ec2.internal : ok=499  changed=75   unreachable=0
failed=0
ip-172-16-25-51.ec2.internal : ok=126  changed=14   unreachable=0
failed=0
ip-172-16-27-213.ec2.internal : ok=127  changed=14   unreachable=0
failed=0
ip-172-16-32-161.ec2.internal : ok=127  changed=14   unreachable=0
failed=0
ip-172-16-34-88.ec2.internal : ok=126  changed=14   unreachable=0
failed=0
ip-172-16-47-107.ec2.internal : ok=295  changed=40   unreachable=0
failed=0
ip-172-16-50-52.ec2.internal : ok=126  changed=14   unreachable=0
failed=0
ip-172-16-52-136.ec2.internal : ok=295  changed=40   unreachable=0
failed=0
ip-172-16-62-235.ec2.internal : ok=127  changed=14   unreachable=0
failed=0
localhost                  : ok=12   changed=0    unreachable=0
failed=0


INSTALLER STATUS
Initialization             : Complete (0:02:18)
Health Check               : Complete (0:05:23)
etcd Install               : Complete (0:04:37)
Master Install             : Complete (0:10:36)
Master Additional Install  : Complete (0:02:52)
Node Install               : Complete (0:17:11)
Hosted Install             : Complete (0:02:15)
Web Console Install        : Complete (0:01:41)
```

3.1. CLOUDFORMS INTEGRATION (OPTIONAL)

The steps defined below assume that Red Hat Cloudforms has been deployed and is accessible by the OpenShift environment.

NOTE

To receive the most information about the deployed environment ensure that the OpenShift metrics components are deployed.

3.1.1. Requesting the Red Hat OpenShift Container Platform Management Token

The management token is used to allow for Cloudforms to retrieve information from the recently deployed OpenShift environment.

To request this token run the following command from a system with the oc client installed and from an account that has privileges to request the token from the `management-infra` namespace.

```
oc sa get-token -n management-infra management-admin
```


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY
2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJtYW5hZ2V
tZW50LWluZnJhIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6I
m1hbmFnZW1lbnQtYWRtaW4tdG9rZW4tdHM0cTIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2N
vdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoibWFuYWdlbWVudC1hZG1pbiIsImt1YmVybmV0Z
XMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImY0ZDlmMGMxLTEyY2Y
tMTFlOC1iNTgzLWZhMTYzZTEwNjNlYSIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDptY
W5hZ2VtZW50LWluZnJhOm1hbmFnZW1lbnQtYWRtaW4ifQ.LwNm0652paGcJu7m63PxBhs4mjXw
YcqMS5KD-
0aWkEMCPo64WwNEawyyYH31SvuEPaE6qFxZwDdJHwdNsfq1CjUL4BtZHv1I2QZxpVl6gMBQowN
f6fWSeGe1FDZ4lkLjzAoMOCFUWA0Z7lZM1FAlyjfz2LkPNKaFW0ffelSJ2SteuXB_4FNup-
T5bKEPQf2pyrwvs2DadClyEEKpIrdZxuekJ9ZfIubcSc3pp1dZRu8wgmSQSLJ1N75raaUU5obu
9cHjcbB9jpDhTW347oJOoL_Bj4bf0yyuxjuUCp3f4fs1qhyjHb5N5LKKBPgIKzoQJrS7j9Sqzo
9TDMF9YQ5JLQ
```


3.1.2. Adding OpenShift as a Containtainer Provider

Now that the token has been acquired, perform the following steps in the link below to add Red Hat OpenShift Container Platform to Red Hat Cloudforms.

https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.6/html/integration_with_openshift_container_platform/integration

3.1.3. Adding Amazon Web Services to Cloudforms

Red Hat Cloudforms also allows for not only the management of Red Hat OpenShift Container Platform but also for the management of Amazon Web Services. The link below contains the steps for adding Amazon Web Services to Cloudforms.

https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.6/html/managing_providers/cloud_providers#amazon_ec2_providers

CHAPTER 4. OPERATIONAL MANAGEMENT

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

NOTE

The following subsections are from OpenShift Documentation - Diagnostics Tool site. For the latest version of this section, reference the link directly.

4.1. OC ADM DIAGNOSTICS

The `oc adm diagnostics` command runs a series of checks for error conditions in the host or cluster. Specifically, it:

- Verifies that the default registry and router are running and correctly configured.
- Checks `ClusterRoleBindings` and `ClusterRoles` for consistency with base policy.
- Checks that all of the client configuration contexts are valid and can be connected to.
- Checks that SkyDNS is working properly and the pods have SDN connectivity.
- Validates master and node configuration on the host.
- Checks that nodes are running and available.
- Analyzes host logs for known errors.
- Checks that systemd units are configured as expected for the host.

4.2. USING THE DIAGNOSTICS TOOL

Red Hat OpenShift Container Platform may be deployed in numerous scenarios including:

- built from source
- included within a VM image
- as a container image
- via enterprise RPMs

Each method implies a different configuration and environment. The diagnostics were included within `openshift` binary to minimize environment assumptions and provide the ability to run the diagnostics tool within an Red Hat OpenShift Container Platform server or client.

To use the diagnostics tool, preferably on a master host and as cluster administrator, run a sudo user:

```
$ sudo oc adm diagnostics
```

The above command runs all available diagnostis skipping any that do not apply to the environment.

The diagnostics tool has the ability to run one or multiple specific diagnostics via name or as an enabler to address issues within the Red Hat OpenShift Container Platform environment. For example:

```
$ sudo oc adm diagnostics <name1> <name2>
```

The options provided by the diagnostics tool require working configuration files. For example, the NodeConfigCheck does not run unless a node configuration is readily available.

Diagnostics verifies that the configuration files reside in their standard locations unless specified with flags (respectively, `--config`, `--master-config`, and `--node-config`)

The standard locations are listed below:

- **Client:**
  - **As indicated by the `$KUBECONFIG` environment variable variable**
  - ***~/.kube/config file***
- **Master:**
  - ***/etc/origin/master/master-config.yaml***
- **Node:**
  - ***/etc/origin/node/node-config.yaml***

If a configuration file is not found or specified, related diagnostics are skipped.

Available diagnostics include:

Diagnostic Name	Purpose
`AggregatedLogging`	Check the aggregated logging integration for proper configuration and operation.
`AnalyzeLogs`	Check systemd service logs for problems. Does not require a configuration file to check against.
`ClusterRegistry`	Check that the cluster has a working Docker registry for builds and image streams.
`ClusterRoleBindings`	Check that the default cluster role bindings are present and contain the expected subjects according to base policy.
`ClusterRoles`	Check that cluster roles are present and contain the expected permissions according to base policy.
`ClusterRouter`	Check for a working default router in the cluster.

Diagnostic Name	Purpose
`ConfigContexts`	Check that each context in the client configuration is complete and has connectivity to its API server.
`DiagnosticPod`	Creates a pod that runs diagnostics from an application standpoint, which checks that DNS within the pod is working as expected and the credentials for the default service account authenticate correctly to the master API.
`EtcdWriteVolume`	Check the volume of writes against etcd for a time period and classify them by operation and key. This diagnostic only runs if specifically requested, because it does not run as quickly as other diagnostics and can increase load on etcd.
`MasterConfigCheck`	Check this particular hosts master configuration file for problems.
`MasterNode`	Check that the master node running on this host is running a node to verify that it is a member of the cluster SDN.
`MetricsApiProxy`	Check that the integrated Heapster metrics can be reached via the cluster API proxy.
`NetworkCheck`	Create diagnostic pods on multiple nodes to diagnose common network issues from an application standpoint. For example, this checks that pods can connect to services, other pods, and the external network. If there are any errors, this diagnostic stores results and retrieved files in a local directory (***/tmp/openshift/***, by default) for further analysis. The directory can be specified with the **`--network-logdir`** flag.
`NodeConfigCheck`	Checks this particular hosts node configuration file for problems.
`NodeDefinitions`	Check that the nodes defined in the master API are ready and can schedule pods.
`RouteCertificateValidation`	Check all route certificates for those that might be rejected by extended validation.
`ServiceExternalIPs`	Check for existing services that specify external IPs, which are disallowed according to master configuration.

Diagnostic Name	Purpose
`UnitStatus`	Check systemd status for units on this host related to Red Hat OpenShift Container Platform. Does not require a configuration file to check against.

4.3. RUNNING DIAGNOSTICS IN A SERVER ENVIRONMENT

An Ansible-deployed cluster provides additional diagnostic benefits for nodes within Red Hat OpenShift Container Platform cluster due to:

- Standard location for both master and node configuration files
- Systemd units are created and configured for managing the nodes in a cluster
- All components log to journald.

Standard location of the configuration files placed by an Ansible-deployed cluster ensures that running `sudo oc adm diagnostics` works without any flags. In the event, the standard location of the configuration files is not used, options flags as those listed in the example below may be used.

```
$ sudo oc adm diagnostics --master-config=<file_path> --node-config=
<file_path>
```

For proper usage of the log diagnostic, systemd units and log entries within `journald` are required. If log entries are not using the above method, log diagnostics won't work as expected and are intentionally skipped.

4.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT

The diagnostics runs using as much access as the existing user running the diagnostic has available. The diagnostic may run as an ordinary user, a cluster-admin user or cluster-admin user.

A client with ordinary access should be able to diagnose its connection to the master and run a diagnostic pod. If multiple users or masters are configured, connections are tested for all, but the diagnostic pod only runs against the current user, server, or project.

A client with cluster-admin access available (for any user, but only the current master) should be able to diagnose the status of the infrastructure such as nodes, registry, and router. In each case, running `sudo oc adm diagnostics` searches for the standard client configuration file location and uses it if available.

4.5. ANSIBLE-BASED HEALTH CHECKS

Additional diagnostic health checks are available through the Ansible-based tooling used to install and manage Red Hat OpenShift Container Platform clusters. The reports provide common deployment problems for the current Red Hat OpenShift Container Platform installation.

These checks can be run either using the `ansible-playbook` command (the same method used during Advanced Installation) or as a containerized version of openshift-ansible. For the

`ansible-playbook` method, the checks are provided by the atomic-openshift-utils RPM package.

For the containerized method, the openshift3/ose-ansible container image is distributed via the Red Hat Container Registry.

Example usage for each method are provided in subsequent sections.

The following health checks are a set of diagnostic tasks that are meant to be run against the Ansible inventory file for a deployed Red Hat OpenShift Container Platform cluster using the provided *health.yml* playbook.

> **WARNING**
>
> Due to potential changes the health check playbooks could make to the environment, the playbooks should only be run against clusters that have been deployed using Ansible with the same inventory file used during deployment. The changes consist of installing dependencies in order to gather required information. In some circumstances, additional system components (i.e. `docker` or networking configurations) may be altered if their current state differs from the configuration in the inventory file. These health checks should only be run if the administrator does not expect the inventory file to make any changes to the existing cluster configuration.

Table 4.1. Diagnostic Health Checks

Check Name	Purpose
`etcd_imagedata_size`	This check measures the total size of Red Hat OpenShift Container Platform image data in an etcd cluster. The check fails if the calculated size exceeds a user-defined limit. If no limit is specified, this check fails if the size of image data amounts to 50% or more of the currently used space in the etcd cluster. A failure from this check indicates that a significant amount of space in etcd is being taken up by Red Hat OpenShift Container Platform image data, which can eventually result in etcd cluster crashing. A user-defined limit may be set by passing the `etcd_max_image_data_size_bytes` variable. For example, setting `etcd_max_image_data_size_bytes=40000000000` causes the check to fail if the total size of image data stored in etcd exceeds 40 GB.

Check Name	Purpose
`etcd_traffic`	This check detects higher-than-normal traffic on an etcd host. The check fails if a **`journalctl`** log entry with an etcd sync duration warning is found. For further information on improving etcd performance, see Recommended Practices for Red Hat OpenShift Container Platform etcd Hosts and the Red Hat Knowledgebase.
`etcd_volume`	This check ensures that the volume usage for an etcd cluster is below a maximum user-specified threshold. If no maximum threshold value is specified, it is defaulted to **`90%`** of the total volume size. A user-defined limit may be set by passing the **`etcd_device_usage_threshold_percent`** variable.
`docker_storage`	Only runs on hosts that depend on the **docker** daemon (nodes and containerized installations). Checks that **docker**'s total usage does not exceed a user-defined limit. If no user-defined limit is set, **docker**'s maximum usage threshold defaults to 90% of the total size available. The threshold limit for total percent usage can be set with a variable in the inventory file, for example **`max_thinpool_data_usage_percent=90`**. This also checks that **docker**'s storage is using a supported configuration.
`curator`, **`elasticsearch`**, **`fluentd`**, **`kibana`**	This set of checks verifies that Curator, Kibana, Elasticsearch, and Fluentd pods have been deployed and are in a **`running`** state, and that a connection can be established between the control host and the exposed Kibana URL. These checks run only if the **`openshift_logging_install_logging`** inventory variable is set to **`true`**. Ensure that they are executed in a deployment where cluster logging has been enabled.

Check Name	Purpose
`logging_index_time`	This check detects higher than normal time delays between log creation and log aggregation by Elasticsearch in a logging stack deployment. It fails if a new log entry cannot be queried through Elasticsearch within a timeout (by default, 30 seconds). The check only runs if logging is enabled. A user-defined timeout may be set by passing the **`openshift_check_logging_index_timeout_seconds`** variable. For example, setting **`openshift_check_logging_index_timeout_seconds=45`** causes the check to fail if a newly-created log entry is not able to be queried via Elasticsearch after 45 seconds.

NOTE

A similar set of checks meant to run as part of the installation process can be found in Configuring Cluster Pre-install Checks. Another set of checks for checking certificate expiration can be found in Redeploying Certificates.

4.5.1. Running Health Checks via ansible-playbook

The openshift-ansible health checks are executed using the `ansible-playbook` command and requires specifying the cluster's inventory file and the *health.yml* playbook:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
checks/health.yml
```

In order to set variables in the command line, include the `-e` flag with any desired variables in key=value format. For example:

```
$ ansible-playbook -i <inventory_file> \
    /usr/share/ansible/openshift-ansible/playbooks/openshift-
checks/health.yml
    -e openshift_check_logging_index_timeout_seconds=45
    -e etcd_max_image_data_size_bytes=40000000000
```

To disable specific checks, include the variable `openshift_disable_check` with a comma-delimited list of check names in the inventory file prior to running the playbook. For example:

```
openshift_disable_check=etcd_traffic,etcd_volume
```

Alternatively, set any checks to disable as variables with `-e openshift_disable_check=<check1>,<check2>` when running the `ansible-playbook` command.

4.6. RUNNING HEALTH CHECKS VIA DOCKER CLI

The openshift-ansible playbooks may run in a Docker container avoiding the requirement for installing and configuring Ansible, on any host that can run the ose-ansible image via the Docker CLI.

This is accomplished by specifying the cluster's inventory file and the *health.yml* playbook when running the following `docker run` command as a non-root user that has privileges to run containers:

```
$ docker run -u `id -u` \ 1
    -v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \ 2
    -v /etc/ansible/hosts:/tmp/inventory:ro \ 3
    -e INVENTORY_FILE=/tmp/inventory \
    -e PLAYBOOK_FILE=playbooks/openshift-checks/health.yml \ 4
    -e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 -e
etcd_max_image_data_size_bytes=40000000000" \ 5
    openshift3/ose-ansible
```

1. These options make the container run with the same UID as the current user, which is required for permissions so that the SSH key can be read inside the container (SSH private keys are expected to be readable only by their owner).
2. Mount SSH keys as a volume under */opt/app-root/src/.ssh* under normal usage when running the container as a non-root user.
3. Change */etc/ansible/hosts* to the location of the cluster's inventory file, if different. This file is bind-mounted to */tmp/inventory*, which is used according to the `INVENTORY_FILE` environment variable in the container.
4. The `PLAYBOOK_FILE` environment variable is set to the location of the *health.yml* playbook relative to */usr/share/ansible/openshift-ansible* inside the container.
5. Set any variables desired for a single run with the `-e key=value` format.

In the above command, the SSH key is mounted with the `:Z` flag so that the container can read the SSH key from its restricted SELinux context. This ensures the original SSH key file is relabeled similarly to `system_u:object_r:container_file_t:s0:c113,c247`. For more details about `:Z`, see the `docker-run(1)` man page.

It is important to note these volume mount specifications because it could have unexpected consequences. For example, if one mounts (and therefore relabels) the *$HOME/.ssh* directory, sshd becomes unable to access the public keys to allow remote login. To avoid altering the original file labels, mounting a copy of the SSH key (or directory) is recommended.

It is plausible to want to mount an entire *.ssh* directory for various reasons. For example, this enables the ability to use an SSH configuration to match keys with hosts or modify other connection parameters. It could also allow a user to provide a *known_hosts* file and have SSH validate host keys, which is disabled by the default configuration and can be re-enabled with an environment variable by adding `-e ANSIBLE_HOST_KEY_CHECKING=True` to the `docker` command line.

CHAPTER 5. CONCLUSION

Red Hat solutions involving the Red Hat OpenShift Container Platform provide an excellent foundation for building a production ready environment which simplifies the deployment process, provides the latest best practices, and ensures stability by running applications in a highly available environment.

The steps and procedures described in this reference architecture provide system, storage, and Red Hat OpenShift Container Platform administrators the blueprints required to create solutions to meet business needs. Administrators may reference this document to simplify and optimize their Red Hat OpenShift Container Platform on Amazon Web Services environments with the following tasks:

- Deciding between different internal network technologies
- Provisioning instances within Amazon Web Services for Red Hat OpenShift Container Platform readiness
- Deploying Red Hat OpenShift Container Platform 3.9
- Using dynamic provisioned storage
- Verifying a successful installation
- Troubleshooting common pitfalls

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. CONTRIBUTORS

Chris Callegari, content provider

Ryan Cook, content provider

Roger Lopez, content provider

Eduardo Minguez, content provider

Davis Phillips, content provider

Chandler Wilkerson, content provider

APPENDIX B. UNDEPLOY AWS INFRASTRUCTURE

GitHub OpenShift openshift-ansible-contrib repository contains an Ansible playbook that provides a friendly and repeatable undeployment experience.

It is usefule to first recover the subscriptions by unregistering the nodes. This may be performed with an Ansible ad-hoc command:

```
$ ansible nodes -a 'subscription-manager unregister'
```

Execute undeploy_aws.yaml play to deploy AWS infrastructure.

```
$ ansible-playbook playbooks/undeploy_aws.yaml
```

WARNING

This play will destroy Red Hat OpenShift Container Platform and all AWS infrastructure resources as well as supporting files.

APPENDIX C. LINKS

- AWS Getting Started Resource Center
- Getting Started with AWS Documentation
- Product Documentation for OpenShift Container Platform 3.9