Reference Architectures 2018

Deploying and Managing OpenShift 3.9 on Red Hat OpenStack Platform 10

Last Updated: 2019-02-05

Reference Architectures 2018 Deploying and Managing OpenShift 3.9 on Red Hat OpenStack Platform 10

Roger Lopez
refarch-feedback@redhat.com

Abstract

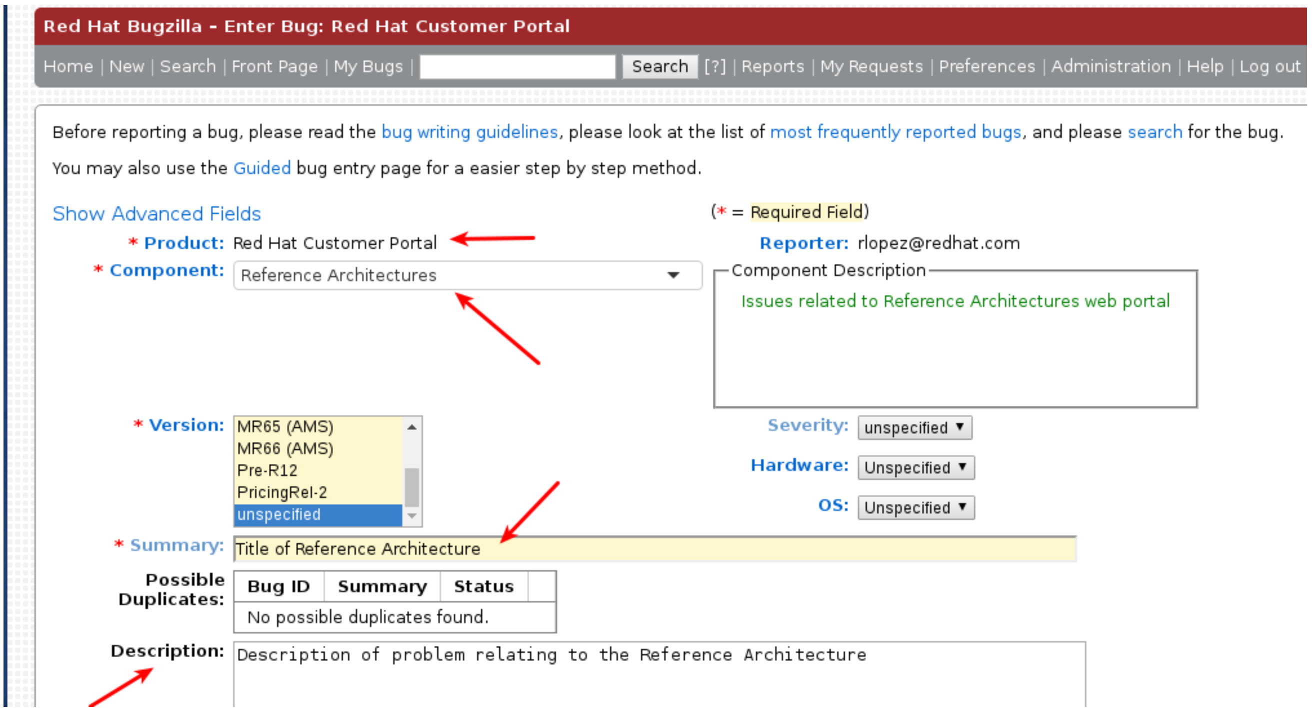
The purpose of this document is to provide guidelines and considerations for deploying and managing Red Hat OpenShift Container Platform on Red Hat OpenStack Platform.

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.


Red Hat Bugzilla – Enter Bug: Red Hat Customer Portal
Home | New | Search | Front Page | My Bugs | Search [?] | Reports | My Requests | Preferences | Administration | Help | Log out

Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

You may also use the Guided bug entry page for a easier step by step method.

Show Advanced Fields

(* = Required Field)

* Product: Red Hat Customer Portal

Reporter: rlopez@redhat.com

* Component: Reference Architectures

Component Description
Issues related to Reference Architectures web portal

* Version: MR65 (AMS), MR66 (AMS), Pre-R12, PricingRel-2, unspecified

Severity: unspecified

Hardware: Unspecified

OS: Unspecified

* Summary: Title of Reference Architecture

Possible Duplicates:

Bug ID	Summary	Status
No possible duplicates found.		

Description: Description of problem relating to the Reference Architecture

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 Red Hat OpenStack Platform — 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 Red Hat OpenStack Platform as an Infrastructure as a Service (IaaS).

This reference architecture provides a methodology to deploy a highly available Red Hat OpenShift Container Platform on Red Hat OpenStack Platform 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 Red Hat OpenStack Platform 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:

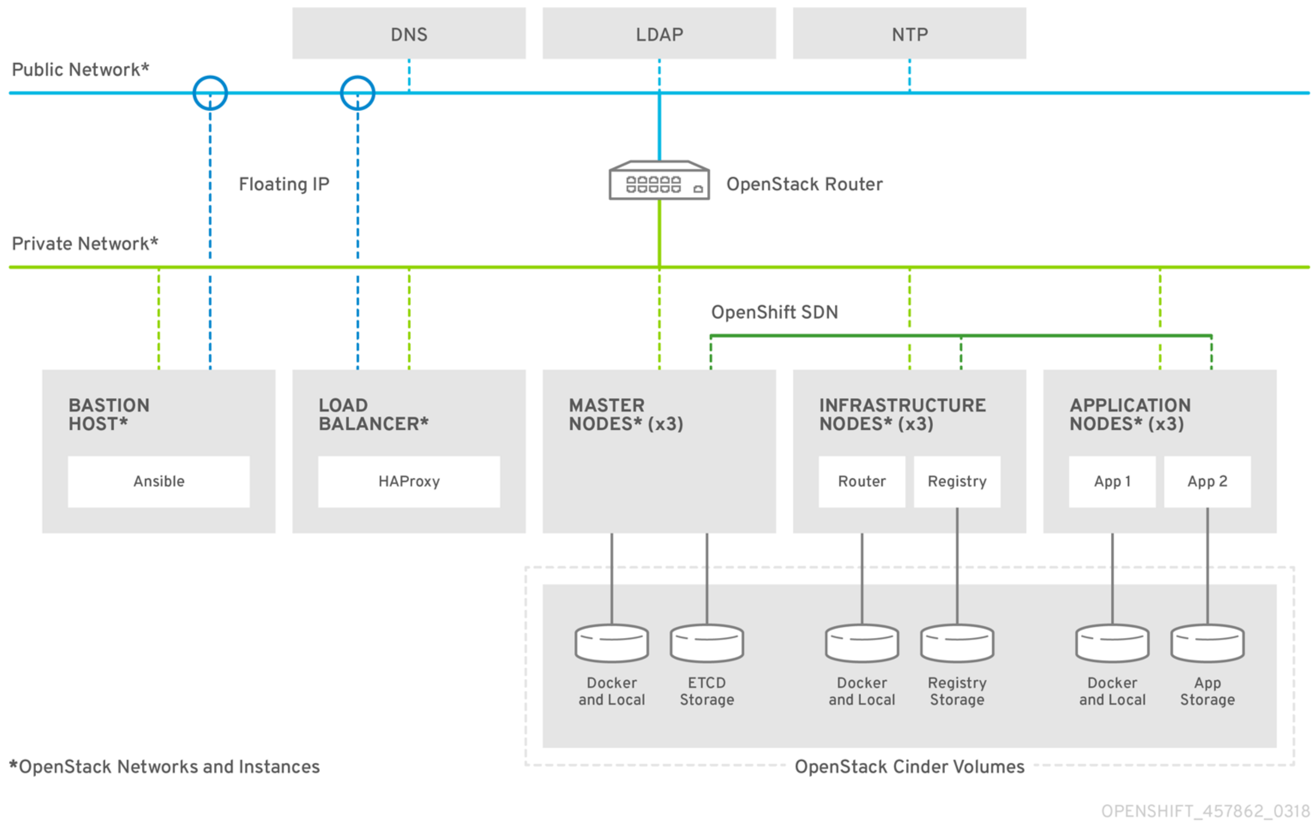
- *Which installation method do you want to use?*
- *How many instances do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other Red Hat technologies?*

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. RED HAT OPENSTACK PLATFORM INSTANCES

The reference environment in this document consists of the following OpenStack instances:

- a single *bastion* instance
- three *master* instance
- six *node* instance (3 *infrastructure* nodes and 3 *application* nodes)

The *bastion* host serves as the installer of the Ansible playbooks that deploy Red Hat OpenShift Container Platform as well as an entry point for management tasks.

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

The nodes provide the runtime environments for the containers. Each node in a cluster has the required services to be managed by the master nodes. Nodes have the required services to run pods.

For more information about the differences between master and nodes see OpenShift Documentation - Kubernetes Infrastructure.

1.2. MASTER INSTANCES AND COMPONENTS

Table 1.1. Master Components[1]

Component	Description
API Server	The Kubernetes API server validates and configures the data for pods, services, and replication controllers. It also assigns pods to nodes and synchronizes pod information with service configuration.
`etcd`	**`etcd`** stores the persistent master state while other components watch **`etcd`** for changes to bring themselves into the desired state. **`etcd`** can be optionally configured for high availability, typically deployed with 2n+1 peer services.
Controller Manager Server	The controller manager server watches **`etcd`** for changes to replication controller objects and then uses the API to enforce the desired state. Can be run as a standalone process. Several such processes create a cluster with one active leader at a time.

When using the **`native`** high availability method with HAProxy, master components have the following availability.

Table 1.2. Availability Matrix with HAProxy[1]

Role	Style	Notes
API Server	Active-Active	Managed by HAProxy
Controller Manager Server	Active-Passive	One instance is elected as a cluster lead at a time

1: OpenShift Documentation - Kubernetes Infrastructure

1.3. NODE INSTANCES AND COMPONENTS

The *node* instances run containers on behalf of its users and are separated into two functional classes: *infrastructure* and *application* nodes. The infrastructure (*infra*) nodes run the *OpenShift router* , *OpenShift logging* , *OpenShift metrics* , and the *OpenShift registry* while the application (*app*) nodes host the user container processes.

The Red Hat OpenShift Container Platform SDN requires Master instances to be considered nodes, therefore, all nodes run the **`atomic-openshift-node`** service.

The routers are an important component of Red Hat OpenShift Container Platform as they are the entry point to applications running in Red Hat OpenShift Container Platform. Developers can expose their applications running in Red Hat OpenShift Container Platform to be available from outside the Red Hat OpenShift Container Platform SDN creating routes that are published in the routers. Once the traffic reaches the router, the router forwards traffic to the containers on the *app* nodes using the Red Hat OpenShift Container Platform SDN.

In this reference architecture a set of three routers are deployed on the *infra* nodes for high availability purposes. In order to have a single entry point for applications, a load balancer is created as part of the installation in the same Red Hat OpenStack Platform tenant where it can reach the infrastructure nodes for load balancing the incoming traffic to the routers running in the *infra* nodes.

NOTE

If using an external load balancer, the *infra* nodes require a public IP address in order to communicate with the load balancer host.

Table 1.3. Node Instance types

Type	Node role
Master	Host web console and service catalog items.
Infrastructure node	Host Red Hat OpenShift Container Platform infrastructure pods (registry, router, logging, metrics, …)
Application node	Host applications deployed on Red Hat OpenShift Container Platform

1.4. INSTANCE STORAGE

`Cinder` volumes attach to instances to provide additional disk space. Red Hat OpenShift Container Platform requires at least 40 GB of space available on the master nodes and at least 15GB on the infrastructure and application nodes for the **`/var`** partition. This reference architecture breaks out **`/var/lib/etcd`**, **`docker`** storage, and ephemeral pod storage to provide individual storage for those specific directories.

Each *master* node Red Hat OpenStack Platform instance creates and mounts **`cinder`** volumes using the steps provided in this reference architecture. One volume contains a volume for **`etcd`** storage. The *master* nodes while not schedulable by default, contain **`cinder`** volumes for both **`docker`** storage and Red Hat OpenShift Container Platform pod storage.

Each *infra* and *app* node Red Hat OpenStack Platform instance creates and mounts **`cinder`** volumes using the steps provided in this reference architecture. One volume contains the **`docker`** runtime disk overhead to store the docker container images. The other **`cinder`** volume is created for pod storage. Each node offers **`cinder`** access to the containers it hosts through Red Hat OpenShift Container Platform.

Table 1.4. Instances storage

Instance type	Role
Masters only	**`etcd`** data
Masters, infra and app nodes	Store docker images
Masters, infra and app nodes	Pod local storage

1.5. LOAD BALANCERS

This guide uses an external load balancer running **`haproxy`** to offer a single entry point for the many Red Hat OpenShift Container Platform components. Organizations can provide their own currently deployed load balancers in the event that the service already exists.

The Red Hat OpenShift Container Platform console, provided by the Red Hat OpenShift Container Platform *master* nodes, can be spread across multiple instances to provide both load balancing and high availability properties.

Application traffic passes through the Red Hat OpenShift Container Platform Router on its way to the container processes. The Red Hat OpenShift Container Platform Router is a reverse proxy service container that multiplexes the traffic to multiple containers making up a scaled application running inside Red Hat OpenShift Container Platform. The load balancer used by *infra* nodes acts as the public view for the Red Hat OpenShift Container Platform applications.

The destination for the master and application traffic must be set in the load balancer configuration after each instance is created, the floating IP address is assigned and before the installation. A single **`haproxy`** load balancer can forward both sets of traffic to different destinations.

1.6. DNS

DNS service is an important component in the Red Hat OpenShift Container Platform environment. Regardless of the provider of DNS, an organization is required to have certain records in place to serve the various Red Hat OpenShift Container Platform components.

NOTE

Red Hat OpenStack Platform 10 includes a DNS-as-a-Service (DNSaaS) component, also known as Designate, but it is currently in Technology Preview phase

Since the load balancer values for the Red Hat OpenShift Container Platform master service and infrastructure nodes running router pods are known beforehand, entries should be configured into the DNS prior to starting the deployment procedure.

1.6.1. Application DNS

Applications served by OpenShift are accessible by the router on ports 80/TCP and 443/TCP. The router uses a *wildcard* record to map all host names under a specific sub domain to the same IP address without requiring a separate record for each name.

This allows Red Hat OpenShift Container Platform to add applications with arbitrary names as long as they are under that sub domain.

For example, a wildcard record for **`*.apps.example.com`** causes DNS name lookups for **`tax.apps.example.com`** and **`home-goods.apps.example.com`** to both return the same IP address: **`10.19.x.y`**. All traffic is forwarded to the OpenShift Routers. The Routers examine the HTTP headers of the queries and forward them to the correct destination.

With a load-balancer host address of 10.19.x.y, the wildcard DNS record can be added as follows:

Table 1.5. Load Balancer DNS records

IP Address	Hostname	Purpose
10.19.x.y	**`*.apps.example.com`**	User access to application web services

1.7. 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.

1.8. RED HAT OPENSHIFT CONTAINER PLATFORM COMPONENTS

Red Hat OpenShift Container Platform comprises of multiple instances running on Red Hat OpenStack Platform 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.

1.8.1. 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.

1.8.1.1. 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.

1.8.1.2. 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'.

1.8.1.3. 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'}"
```

1.8.2. 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.

1.8.3. 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.

1.8.3.1. 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.

1.8.3.2. 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.

1.9. 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), a third-party SDN, or **`flannel`**.

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 and Flannel”, discusses pros and cons when deciding between the four popular options for the internal networks - **`ovs-multitenant`**, **`ovs-networkpolicy`**, **`ovs-subnet`**, and **`flannel`**.

1.9.1. OpenShift SDN Plugins and Flannel

This section focuses on two methods for pod communication within Red Hat OpenShift Container Platform, OpenShift SDN and **`flannel`**. Regarding OpenShift SDN, there are 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[2].

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

When comparing the three different OpenShift SDN options, one common drawback that each possesses is the fact that they use an overlay network that uses OVS. The reason it is a drawback is specific to when deploying Red Hat OpenShift Container Platform on Red Hat OpenStack Platform as Red Hat OpenShift Container Platform internode traffic is carried over a Red Hat OpenStack Platform **`neutron`** network. **`neutron`** is a SDN that uses OVS and provides network access to all the Red Hat OpenShift Container Platform instances. This in turn causes double encapsulation and may degrade performance of the pod network.

What about `flannel`?

`flannel` is a virtual networking layer designed for containers. Instead of having two SDN to route traffic, **`flannel`** runs in a mode labeled **`host-gw`** that routes packets by each node in the Red Hat OpenShift Container Platform environment running a daemon labeled **`flanneld`** that sends information to a centralized location (**`etcd`** store) thus allowing other nodes running the **`flanneld`** daemon to route packets to other containers as long as they are all on the flannel network. The main benefit to **`flannel`** is that it provides performance benefits over OVS double encapsulation since it does not create its own SDN on top of the existing RHOSP neutron network.

Why not default to `flannel`?

`flannel` uses a single IP network space for all of the pods allocating a contiguous subset of the space to each instance. Consequently, nothing prevents a pod from attempting to contact any IP address in the same network space. This hinders multi-tenancy because the network cannot be used to isolate pods in one application from another.

The key to making the right decision rests between mutli-tenancy isolation vs performance. Depending on the priority of each of these, should determine the appropriate choice when deciding between **`OpenShift SDN`** and **`flannel`** internal networks.

The steps to setup **`flannel`** instead of **`OpenShift SDN`** can be found within the Appendix B, *Using Flannel*.

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. If isolation is not important, **`flannel`** is a better choice since it would not have the double encapsulation drawback.

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**.

1.10. 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.

1.11. 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.

1.11.1. 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 Red Hat OpenStack Platform or from **glusterfs** if deployed.

1.11.1.1. 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**.

1.12. 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.

1.13. 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
```

1.14. 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"}
```

CHAPTER 2. RED HAT OPENSHIFT CONTAINER PLATFORM PREREQUISITES

A successful deployment of Red Hat OpenShift Container Platform requires many prerequisites. This consists of a set of infrastructure and host configuration steps prior to the actual installation of Red Hat OpenShift Container Platform using Ansible. In the following subsequent sections, details regarding the prerequisites and configuration changes required for an Red Hat OpenShift Container Platform on a Red Hat OpenStack Platform environment are discussed in detail.

NOTE

All the Red Hat OpenStack Platform CLI commands in this reference environment are executed using the CLI **`openstack`** commands within the Red Hat OpenStack Platform director node. If using a workstation or laptop to execute these commands instead of the Red Hat OpenStack Platform director node, please ensure to install the **`python-openstackclient`** RPM found within the **`rhel-7-server-openstack-10-rpms`** repository.

Example:

Install **`python-openstackclient`** package.

```
$ sudo subscription-manager repos --enable rhel-7-server-openstack-10-rpms
$ sudo yum install -y python-openstackclient
```

Verify the package version for **`python-openstackclient`** is at least version **`3.2.1-1`**

```
$ rpm -q python-openstackclient
python-openstackclient-3.2.1-1.el7ost.noarch
```

To use bash completion feature, where the openstack commands are completed by pressing <tab> keystroke, use the following command in the host where the python-openstackclient has been installed:

```
$ openstack complete | sudo tee /etc/bash_completion.d/osc.bash_completion
> /dev/null
```

Logout and login again and the openstack commands can be autocompleted by pressing the <tab> keystroke as for **`openstack network list`**:

```
$ openstack netw<tab> li<tab>
```

2.1. CREATING OPENSTACK USER ACCOUNTS, PROJECTS AND ROLES

Before installing Red Hat OpenShift Container Platform, the Red Hat OpenStack Platform environment requires a project (tenant) that stores the Red Hat OpenStack Platform instances that are to install Red Hat OpenShift Container Platform. This project requires ownership by a user and role of that user to be set to *member*.

The following steps show how to accomplish the above.

As the Red Hat OpenStack Platform overcloud administrator,

1. Create a project (tenant) that is to store the RHOSP instances

   ```
   $ openstack project create <project>
   ```

2. Create a Red Hat OpenStack Platform user that has ownership of the previously created project

   ```
   $ openstack user create --password <password> <username>
   ```

3. Set the role of the user

   ```
   $ openstack role add --user <username> --project <project> member
   ```

Once the above is complete, an OpenStack administrator can create an RC file with all the required information to the user(s) implementing the Red Hat OpenShift Container Platform environment.

An example RC file:

```
$ cat path/to/examplerc
export OS_USERNAME=<username>
export OS_TENANT_NAME=<project>
export OS_PASSWORD=<password>
export OS_CLOUDNAME=<overcloud>
export OS_AUTH_URL=http://<ip>:5000/v2.0
```

As the user(s) implementing the Red Hat OpenShift Container Platform environment, within the Red Hat OpenStack Platform director node or workstation, ensure to **`source`** the credentials as follows:

```
$ source path/to/examplerc
```

2.2. CREATING A RED HAT ENTERPRISE LINUX BASE IMAGE

After the user and project are created, a cloud image is required for all the Red Hat OpenStack Platform instances that are to install Red Hat OpenShift Container Platform. For this particular reference environment the image used is Red Hat Enterprise Linux 7.5.

NOTE

Red Hat Enterprise Linux 7.5 includes support for overlay2 file system used in this reference architecture for container storage purposes. If using prior Red Hat Enterprise Linux versions, do not use overlay2 file system.

The Red Hat Enterprise Linux 7.5 cloud image is located at Red Hat Enterprise Linux 7.5 KVM Guest Image Copy the image to the director or workstation node and store the image within **`glance`**. Glance image services provide the ability to discover, register and retrieve virtual machine (VM) images.

NOTE

Depending on the *glance* storage backend, it can be required to convert the image format from *qcow2* to a different format. For *Ceph* storage backends, it is required the image to converted to *raw* image format.

The following steps show the process of incorporating the Red Hat Enterprise Linux 7.5 image.

```
$ source /path/to/examplerc
$ mkdir ~/images
$ sudo cp /path/to/downloaded/rhel-server-7.5-x86_64-kvm.qcow2 ~/images
$ cd ~/images
$ qemu-img convert \
    -f qcow2 -O raw \
    rhel-server-7.5-x86_64-kvm.qcow2 ./<img-name>.raw
$ openstack image create <img-name> \
    --disk-format=raw \
    --container-format=bare < <img-name>.raw
```

An Red Hat OpenStack Platform administrator may allow this **`glance`** Red Hat Enterprise Linux image to be readily available to all projects within the Red Hat OpenStack Platform environment. This may be done by **`source`** an RC file with OpenStack administrator privileges and including the **`--public`** option prior to creating the **`raw`** image.

Confirm the creation of the image via **`openstack image list`**. An example output of the reference environment is shown.

```
$ openstack image list
+--------------------------------------+-------+--------+
| ID                                   | Name  | Status |
+--------------------------------------+-------+--------+
| 7ce292d7-0e7f-495d-b2a6-aea6c7455e8c | rhel7 | active |
+--------------------------------------+-------+--------+
```

2.3. CREATE AN OPENSTACK FLAVOR

Within OpenStack, flavors define the size of a virtual server by defining the compute, memory, and storage capacity of **nova** computing instances. Since the base image within this reference architecture is Red Hat Enterprise Linux 7.5, a **`m1.node`** and **`m1.master`** sized flavor is created with the following specifications as shown in Table 2.1, "Minimum System Requirements for OpenShift".

Table 2.1. Minimum System Requirements for OpenShift

Node Type	CPU	RAM	Root Disk	Flavor
Masters	2	16 GB	45 GB	**`m1.master`**
Nodes	1	8 GB	20 GB	**`m1.node`**

As an OpenStack administrator,

```
$ openstack flavor create <flavor_name> \
    --id auto \
    --ram <ram_in_MB> \
    --disk <disk_in_GB> \
    --vcpus <num_vcpus>
```

An example below showing the creation of flavors within this reference environment.

```
$ openstack flavor create m1.master \
    --id auto \
    --ram 16384 \
    --disk 45 \
    --vcpus 2
$ openstack flavor create m1.node \
    --id auto \
    --ram 8192 \
    --disk 20 \
    --vcpus 1
```

NOTE

If access to OpenStack administrator privileges to create new flavors is unavailable, use existing flavors within the OpenStack environment that meet the requirements in Table 2.1, “Minimum System Requirements for OpenShift”

2.4. CREATING AN OPENSTACK KEYPAIR

Red Hat OpenStack Platform uses **`cloud-init`** to place an **`ssh`** public key on each instance as it is created to allow **`ssh`** access to the instance. Red Hat OpenStack Platform expects the user to hold the private key.

In order to generate a keypair use the following command

```
$ openstack keypair create <keypair-name> > /path/to/<keypair-name>.pem
```

Once the keypair is created, set the permissions to 600 thus only allowing the owner of the file to read and write to that file.

```
$ chmod 600 /path/to/<keypair-name>.pem
```

2.5. CREATION OF RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKS VIA OPENSTACK

When deploying Red Hat OpenShift Container Platform on Red Hat OpenStack Platform as described in this reference environment, the requirements are two networks — *public* and *internal* network.

2.5.1. Public Network

The *public* network is a network that contains external access and can be reached by the outside world. The *public* network creation can be only done by an OpenStack administrator.

The following commands provide an example of creating an OpenStack provider network for *public* network access.

As an OpenStack administrator,

```
$ source /path/to/examplerc
$ neutron net-create public_network \
    --router:external \
    --provider:network_type flat \
    --provider:physical_network datacentre
$ neutron subnet-create \
    --name public_network \
    --gateway <ip> \
    --allocation-pool start=<float_start>,end=<float_end> \
    public_network <CIDR>
```

NOTE

`<float_start>` and **`<float_end>`** are the associated floating IP pool provided to the network labeled *public* network. The Classless Inter-Domain Routing (CIDR) uses the format <ip>/<routing_prefix>, i.e. 10.5.2.1/24

2.5.2. Internal Network

The *internal* network is connected to the *public* network via a router during the network setup. This allows each Red Hat OpenStack Platform instance attached to the *internal* network the ability to request a floating IP from the *public* network for public access.

The following commands create the *internal* network.

WARNING

If there currently is not a DNS service within the organization proceed to Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance* before continuing as this is required. The *internal* network is required for the DNS instance thus it can be created without specifying the *dns-nameserver* setting then add it as **`openstack subnet set --dns-nameserver <dns-ip> <subnet>`** once the DNS server is properly installed.

```
$ source /path/to/examplerc
$ openstack router create <router-name>
$ openstack network create <private-net-name>
$ openstack subnet create --net <private-net-name> \
    --dns-nameserver <dns-ip> \
    --subnet-range <cidr> \
    <private-net-name>
$ openstack subnet list
$ openstack router add subnet <router-name> <private-subnet-uuid>
$ neutron router-gateway-set <router-name> <public-subnet-uuid>
```

2.6. SETTING UP DNS FOR RED HAT OPENSHIFT CONTAINER PLATFORM

The installation process for Red Hat OpenShift Container Platform depends on a reliable name service that contains an address record for each of the target instances. If a DNS is currently not set, please refer to the appendix section on Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance*.

NOTE

Using **`/etc/hosts`** is not valid, a proper DNS service must exist.

2.7. CREATING RED HAT OPENSTACK PLATFORM SECURITY GROUPS

Red Hat OpenStack Platform networking allows the user to define inbound and outbound traffic filters that can be applied to each instance on a network. This allows the user to limit network traffic to each instance based on the function of the instance services and not depend on host based filtering.

This section describes the ports and services required for each type of host and how to create the security groups in Red Hat OpenStack Platform.

The following table shows the security group association to every instance type:

Table 2.2. Security Group association

Instance type	Security groups associated
Bastion	**`<bastion-sg-name>`**
Masters	**`<master-sg-name>`** **`<node-sg-name>`**
Infra nodes	**`<infra-sg-name>`** **`<node-sg-name>`**
App nodes	**`<node-sg-name>`**

As the <node-sg-name> security group is applied to all the instance types, the common rules are applied to it.

```
$ source /path/to/examplerc
$ for SG in <bastion-sg-name> <master-sg-name> <infra-sg-name> <node-sg-
name>
do
  openstack security group create $SG
done
```

The following tables and commands describe the security group network access controls for the *bastion* host, *master*, *infrastructure* and *app* instances.

2.7.1. Bastion Security Group

The *bastion* instance only needs to allow inbound **`ssh`**. This instance exists to give operators a stable base to deploy, monitor and manage the Red Hat OpenShift Container Platform environment.

Table 2.3. Bastion Security Group TCP ports

Port/Protocol	Service	Remote source	Purpose

Port/Protocol	Service	Remote source	Purpose
ICMP	ICMP	Any	Allow ping, traceroute, etc.
22/TCP	SSH	Any	Secure shell login

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <bastion-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    <bastion-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group show <bastion-sg-name>
```

2.7.2. Master Security Group

The Red Hat OpenShift Container Platform master security group 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 2.4. Master Host Security Group Ports

Port/Protocol	Service	Remote source	Purpose
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+)
8443/TCP	HTTPS	Any	Master WebUI and API

NOTE

As masters nodes are in fact nodes, the node security group is applied, as well as, the master security group.

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 2379:2380 \
    --src-group <master-sg-name> \
    <master-sg-name>

for PROTO in tcp udp;
  do
    openstack security group rule create \
      --ingress \
      --protocol $PROTO \
      --dst-port 8053 \
      --src-group <node-sg-name> \
      <master-sg-name>;
done
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 8443 \
    <master-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group show <master-sg-name>
```

2.7.3. Infrastructure Node Security Group

The infrastructure nodes run the Red Hat OpenShift Container Platform router and the local registry. It must accept inbound connections on the web ports that are forwarded to their destinations.

Table 2.5. Infrastructure Node 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
9200/TCP	ElasticSearch	Any	ElasticSearch API
9300/TCP	ElasticSearch	Any	Internal cluster use

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ for PORT in 80 443 9200 9300;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port $PORT \
    <infra-sg-name>;
done
```

Verification of the security group is as follows:

```
$ openstack security group show <infra-sg-name>
```

2.7.4. Node Security Group

All instances run OpenShift node service. All instances should only accept ICMP from any source, **ssh** traffic from the *bastion* host or other nodes, pod to pod communication via SDN traffic and kubelet communication via Kubernetes.

Table 2.6. Node Security Group Ports

Port/Protocol	Services	Remote source	Purpose
ICMP	Ping et al.	Any	Debug connectivity issues
22/TCP	SSH	Bastion	Secure shell login
4789/UDP	SDN	Nodes	Pod to pod communications
10250/TCP	kubernetes	Nodes	Kubelet communications

Creation of the above security group is as follows:

```
$ source /path/to/examplerc
$ openstack security group rule create \
    --ingress \
    --protocol icmp \
    <node-sg-name>
$ for SRC in <bastion-sg-name> <node-sg-name>;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 22 \
    --src-group $SRC \
    <node-sg-name>;
done
```

```
$ openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port 10250 \
    --src-group <node-sg-name> \
    <node-sg-name>
$ openstack security group rule create \
    --ingress \
    --protocol udp \
    --dst-port 4789 \
    --src-group <node-sg-name> \
    <node-sg-name>
```

Verification of the security group is as follows:

```
$ openstack security group show <node-sg-name>
```

2.8. CREATING RED HAT OPENSTACK PLATFORM CINDER VOLUMES

The master instances contain volumes to store **`docker`** images, OpenShift local volumes, and **`etcd`** storage. Each volume has a role in keeping the **`/var`** filesystem partition from corruption and provides the ability for the use of **`cinder`** volume snapshots.

The node instances contain two volumes - a **`docker`** and OpenShift local storage volume. These volumes do not require snapshot capability, but serve the purpose of ensuring that a large image or container does not compromise node performance or abilities.

Table 2.7. Cinder Volumes

Instance type	Volumes	Role	Minimum recommended size
Masters only	**`<instance_name>-etcd`**	**`etcd`** data	25 GB
Masters, infra and app nodes	**`<instance_name>-docker`**	Store docker images	15 GB
Masters, infra and app nodes	**`<instance_name>-openshift-local`**	Pod local storage	30 GB

NOTE

Those volumes are created at boot time, there is not need to precreate them prior creating the instance.

2.8.1. Docker Volume

During the installation of Red Hat OpenShift Container Platform, the Red Hat OpenStack Platform instances created for RHOCP should include various **`cinder`** volumes to ensure that various OpenShift directories do not fill up the disk or cause disk contention in the **`/var`** partition.

NOTE

The value of the docker volume size should be at least 15 GB.

2.8.2. etcd Volume

A **`cinder`** volume is created on the *master* instances for the storage of **`/var/lib/etcd`**. Storing **`etcd`** allows the similar benefit of protecting **`/var`** but more importantly provides the ability to perform snapshots of the volume when performing **`etcd`** maintenance. The **`cinder etcd`** volume is created only on the *master* instances.

NOTE

The value of the etcd volume size should be at least 25 GB.

2.8.3. OpenShift Local Volume

A **`cinder`** volume is created for the directory of **`/var/lib/origin/openshift.local.volumes`** that is used with the **`perFSGroup`** setting at installation and with the mount option of **`gquota`**. These settings and volumes set a quota to ensure that containers cannot grow to an unreasonable size.

NOTE

The value of OpenShift local volume size should be at least 30 GB.

2.8.4. Registry volume

The OpenShift image registry requires a cinder volume to ensure that images are saved in the event that the registry needs to migrate to another node.

```
$ source /path/to/examplerc
$ openstack volume list
```

```
$ openstack volume create --size <volume-size-in-GB> openshift-registry
```

NOTE

The registry volume size should be at least 30GB.

2.9. CREATING RHOSP INSTANCES FOR RHOCP

This reference environment consists of the following instances:

- one *bastion* instance
- three *master* instances
- three *infrastructure* instances
- three *application* instances

`etcd` requires that an odd number of cluster members exist. Three masters were chosen to support high availability and **`etcd`** clustering. Three infrastructure instances allow for minimal to zero downtime for applications running in the OpenShift environment. Applications instance can be one to many instances depending on the requirements of the organization.

NOTE

infra and *app* node instances can easily be added after the initial install.

The creation of each instance within Red Hat OpenStack Platform varies based upon the following:

- Hostname
- Attached **`cinder`** storage volumes
- Assigned security group based upon instance type

Red Hat OpenShift Container Platform uses the hostnames of all the nodes to identify, control and monitor the services on them. It is critical that the instance hostnames, the DNS hostnames and IP addresses are mapped correctly and consistently.

Without any inputs, Red Hat OpenStack Platform uses the **`nova`** instance name as the hostname and the domain as *novalocal*. The *bastion* host's FQDN would result in **`bastion.novalocal`**. This would suffice if Red Hat OpenStack Platform populated a DNS service with these names thus allowing each instance to find the IP addresses by name.

However, using the *novalocal* domain requires creating a zone in the external DNS service named *novalocal*. Since the RHOSP instance names are unique only within a project, this risks name collisions with other projects. To remedy this issue, creation of a subdomain for the *internal* network is implemented under the project domain, i.e. *example.com*

Table 2.8. Subdomain for RHOCP Internal Network

Domain Name	Description
`example.com`	All interfaces on the internal only network

The **`nova`** instance name and the instance hostname are the same. Both are in the **`internal`** subdomain. The floating IPs are assigned to the top level domain **`example.com`**.

Table 2.9. Sample FQDNs

Fully Qualified Name	Description
`master0.example.com`	Name of the *internal* network interface on the **`master0`** instance
`infra0.example.com`	Name of the *internal* network interface on the **`infra0`** instance
`app0.example.com`	Name of the *internal* network interface on the **`app0`** instance

Fully Qualified Name	Description
openshift.example.com	Name of the Red Hat OpenShift Container Platform console using the address of the **haproxy** instance on the *public* network

2.9.1. Cloud-Init

Red Hat OpenStack Platform provides a way for users to pass in information to be applied when an instance boots. The **--user-data** switch to **nova boot** command makes the contents of the provided file available to the instance through **cloud-init**. **cloud-init** is a set of init scripts for cloud instances. It is available via the **rhel-7-server-rh-common-rpms** repository and queries a standard URL for the **user-data** file and processes the contents to initialize the OS of the deployed Red Hat OpenStack Platform instance.

The **user-data** is placed in files named *<hostname>.yaml* where *<hostname>* is the name of the instance.

An example of the *<hostname>.yaml* file is shown below. This is required for every Red Hat OpenStack Platform instance that is to be used for the Red Hat OpenShift Container Platform deployment.

Create a **user-data** directory to store the *<hostname>.yaml* files.

```
$ mkdir /path/to/user-data
```

NOTE

nova boot command is used in conjunction with the **--block-device** option to ensure that at boot time the proper volume is attached to the device that is specified. Currently, this cannot be done via **openstack server create** as it may attach different volume (i.e. attach vdb as vdc) then specified. More information: https://bugzilla.redhat.com/show_bug.cgi?id=1489001

2.9.1.1. Master Cloud Init

The master instances require docker storage, partitions, and certain mounts be available for a successful deployment using the cinder volumes created in the previous steps.

```
#cloud-config
cloud_config_modules:
- disk_setup
- mounts

hostname: <hostname>
fqdn: <hostname>.<domain>

write_files:
  - path: "/etc/sysconfig/docker-storage-setup"
    permissions: "0644"
    owner: "root"
    content: |
      DEVS='/dev/vdb'
      VG=docker_vol
      DATA_SIZE=95%VG
      STORAGE_DRIVER=overlay2
      CONTAINER_ROOT_LV_NAME=dockerlv
      CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
      CONTAINER_ROOT_LV_SIZE=100%FREE

fs_setup:
- label: emptydir
  filesystem: xfs
  device: /dev/vdc
  partition: auto
- label: etcd_storage
  filesystem: xfs
  device: /dev/vdd
  partition: auto

runcmd:
- mkdir -p /var/lib/origin/openshift.local.volumes
- mkdir -p /var/lib/etcd

mounts:
- [ /dev/vdc, /var/lib/origin/openshift.local.volumes, xfs,
"defaults,gquota" ]
- [ /dev/vdd, /var/lib/etcd, xfs, "defaults" ]
```

2.9.1.2. Node Cloud Init

The node instances, regardless if the instance is an *application* or *infrastructure* node require docker storage, partitions, and certain mounts be available for a successful deployment using the cinder volumes created in the previous steps.

```
#cloud-config
cloud_config_modules:
- disk_setup
- mounts

hostname: <hostname>
fqdn: <hostname>.<domain>

write_files:
  - path: "/etc/sysconfig/docker-storage-setup"
    permissions: "0644"
    owner: "root"
    content: |
      DEVS='/dev/vdb'
      VG=docker_vol
      DATA_SIZE=95%VG
      STORAGE_DRIVER=overlay2
      CONTAINER_ROOT_LV_NAME=dockerlv
      CONTAINER_ROOT_LV_MOUNT_PATH=/var/lib/docker
      CONTAINER_ROOT_LV_SIZE=100%FREE

fs_setup:
- label: emptydir
```

```
  filesystem: xfs
  device: /dev/vdc
  partition: auto

runcmd:
- mkdir -p /var/lib/origin/openshift.local.volumes

mounts:
- [ /dev/vdc, /var/lib/origin/openshift.local.volumes, xfs,
"defaults,gquota" ]
```

Once the yaml files are created for each Red Hat OpenStack Platform instance, create the instances using the **`openstack`** command.

2.9.2. Master Instance Creation

The bash lines below are a loop that allow for all of the master instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *master* instances by filling in the bold items relevant to the current OpenStack values.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in master-{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <master-sg-name>,<node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
local-volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdd,size=<etcd-volume-
size>,shutdown=preserve \
    $node.$domain;
done
```

NOTE

Assuming the **`user-data`** yaml files are labeled **`master-<num>.yaml`**

WARNING

The volume device order is important as cloud-init creates filesystems based on the volume order.

2.9.3. Infrastructure Instance Creation

The bash lines below are a loop that allow for all of the infrastructure instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *infrastructure* instances by filling in the bold items relevant to the current OpenStack values.

```
$ domain = <domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in infra{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <infra-sg-name>,<node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
local-volume-size>,shutdown=preserve \
    $node.$domain;
done
```

NOTE

Assuming the **`user-data`** yaml files are labeled **`infra<num>.yaml`**

WARNING

The volume device order is important as cloud-init creates filesystems based on the volume order.

2.9.4. Application Instance Creation

The bash lines below are a loop that allow for all of the application instances to be created and the specific mounts, instance size, and security groups are configured at launch time. Create the *application* instances by filling in the bold items relevant to the current OpenStack values.

```
$ domain = <domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ for node in app{0..2};
do
  nova boot \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-groups <node-sg-name> \
    --user-data=/path/to/user-data/$node.yaml \
    --block-device source=blank,dest=volume,device=vdb,size=<docker-
volume-size>,shutdown=preserve \
    --block-device source=blank,dest=volume,device=vdc,size=<openshift-
local-volume-size>,shutdown=preserve \
      $node.$domain;
done
```

NOTE

Assuming the **`user-data`** yaml files are labeled **`app<num>.yaml`**

WARNING

The volume device order is important as cloud-init creates filesystems based on the volume order.

2.9.5. Confirming Instance Deployment

The above creates 3 *master*, *infra* and *app* instances.

Verify the creation of the Red Hat OpenStack Platform instances via:

```
$ openstack server list
```

Using the values provided by the **`openstack server list`** command, update the DNS master **`zone.db`** file as shown in Section A.1, “Setting up the DNS Red Hat OpenStack Platform Instance” with the appropriate IP addresses. Do not proceed to the next section until the DNS resolution is configured.

2.9.6. Rename volumes (optional)

cinder volumes are created using the instance id making them confusing to identify.

To rename the *cinder* volumes and make them human readable, use the following snippet:

```
$ for node in master-{0..2} app{0..2} infra{0..2};
do
  dockervol=$(openstack volume list -f value -c ID -c "Attached to" | awk
"/$node/ && /vdb/ {print \$1}")
  ocplocalvol=$(openstack volume list -f value -c ID -c "Attached to" |
awk "/$node/ && /vdc/ {print \$1}")
  openstack volume set --name $node-docker $dockervol
  openstack volume set --name $node-ocplocal $ocplocalvol
done
for master in master{0..2};
do
  etcdvol=$(openstack volume list -f value -c ID -c "Attached to" | awk
"/$master/ && /vdd/ {print \$1}")
  openstack volume set --name $master-etcd $etcdvol
done
```

2.10. CREATING AND CONFIGURING AN HAPROXY RED HAT OPENSTACK PLATFORM INSTANCE

If an organization currently does not have a load balancer in place then **HAProxy** can be deployed. A load balancer such as **HAProxy** provides a single view of the Red Hat OpenShift Container Platform master services for the applications. The master services and the applications use different TCP ports so a single TCP load balancer can handle all of the inbound connections.

The load balanced DNS name that developers use must be in a DNS A record pointing to the **haproxy** server before installation. For applications, a wildcard DNS entry must point to the **haproxy** host.

The configuration of the load balancer is created after all of the Red Hat OpenShift Container Platform instances have been created and floating IP addresses assigned.

The following steps show how to create the security group and add specific rules to the security group.

```
$ source /path/to/examplerc
$ openstack security group create <haproxy-sg-name>
$ for PORT in 22 80 443 8443;
do
  openstack security group rule create \
    --ingress \
    --protocol tcp \
    --dst-port $PORT \
    <haproxy-sg-name>;
done
```

NOTE

If the *m1.small* flavor does not exist by default then create a flavor with 1vcpu and 2GB of RAM.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-group <haproxy-sg-name> \
    haproxy.$domain
```

Assign a floating ip via:

```
$ openstack floating ip create public_network
$ openstack server add floating ip haproxy.$domain <ip>
```

Using the floating IP, verify logging into the **HAProxy** server.

```
$ ssh -i /path/to/<keypair-name>.pem cloud-user@<IP>
```

The configuration of the HAProxy instance is completed within the subsequent steps as the *bastion* host configures the Red Hat subscriptions for all the instances and the Red Hat OpenShift Container Platform installer auto configures the HAProxy instance based upon the information found within the Red Hat OpenShift Container Platform inventory file.

In order to have a single entry point for applications, the load balancer either requires making changes to the existing DNS that allows for wildcards to use the Round Robin algorithm across the different infra nodes. An example can be seen on the Appendix A, *Creating the DNS Red Hat OpenStack Platform Instance* section.

IMPORTANT

If changes within the DNS server are not possible, the following entries within the HAproxy instance, specifically the */etc/haproxy/haproxy.cfg* file is required.

Below is an example of the *haproxy.cfg* file with the appropriate requirements if adding the DNS entries is not possible.

```
#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon
    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

defaults
    log                     global
    option                  httplog
    option                  dontlognull
    option                  http-server-close
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000
listen stats :9000
    stats enable
    stats realm Haproxy\ Statistics
    stats uri /haproxy_stats
    stats auth admin:password
    stats refresh 30
    mode http

frontend  main *:80
    default_backend router80
```

```
backend router80
    balance source
    mode tcp
    # INFRA_80
    server infra0.example.com <IP>:80 check
    server infra1.example.com <IP>:80 check
    server infra2.example.com <IP>:80 check

frontend  main *:443
    default_backend router443

backend router443
    balance source
    mode tcp
    # INFRA_443
    server infra0.example.com <IP>:443 check
    server infra1.example.com <IP>:443 check
    server infra2.example.com <IP>:443 check

frontend  main *:8443
    default_backend mgmt8443

backend mgmt8443
    balance source
    mode tcp
    # MASTERS 8443
    server master0.example.com <IP>:8443 check
    server master1.example.com <IP>:8443 check
    server master2.example.com <IP>:8443 check
```

2.11. CREATING AND CONFIGURING THE BASTION INSTANCE

The bastion servers two functions in this deployment of Red Hat OpenShift Container Platform. The first function is a **ssh** jump box allowing administrators to access instances within the private network. The other role of the *bastion* instance is to serve as a utility host for the deployment and management of Red Hat OpenShift Container Platform.

2.11.1. Deploying the Bastion Instance

Create the *bastion* instance via:

NOTE

If the *m1.small* flavor does not exist by default then create a flavor with 1 vCPU and 2GB of RAM.

```
$ domain=<domain>
$ netid1=$(openstack network show <internal-network-name> -f value -c id)
$ openstack server create \
    --nic net-id=$netid1 \
    --flavor <flavor> \
    --image <image> \
    --key-name <keypair> \
    --security-group <bastion-sg-name> \
    bastion.$domain
```

2.11.2. Creating and Adding Floating IPs to Instances

Once the instances are created, floating IPs must be created and then can be allocated to the instance. The following shows an example

```
$ source /path/to/examplerc
$ openstack floating ip create <public-network-name>
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-08-24T22:44:03Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 10.20.120.150                       |
| floating_network_id | 084884f9-d9d2-477a-bae7-26dbb4ff1873 |
| headers             |                                      |
| id                  | 2bc06e39-1efb-453e-8642-39f910ac8fd1 |
| port_id             | None                                 |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| project_id          | ca304dfee9a04597b16d253efd0e2332     |
| revision_number     | 1                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2017-08-24T22:44:03Z                 |
+---------------------+--------------------------------------+
```

Within the above output, the **`floating_ip_address`** field shows that the floating IP **`10.20.120.150`** is created. In order to assign this IP to one of the Red Hat OpenShift Container Platform instances, run the following command:

```
$ source /path/to/examplerc
$ openstack server add floating ip <instance-name> <ip>
```

For example, if instance **`bastion.example.com`** is to be assigned IP **`10.20.120.150`** the command would be:

```
$ source /path/to/examplerc
$ openstack server add floating ip bastion.example.com 10.20.120.150
```

NOTE

Within this reference architecture, the only Red Hat OpenShift Container Platform instances requiring a floating IP are the *bastion* host and the *haproxy* host. In case the load balancer is located in a different Red Hat OpenStack Platform tenant or somewhere else, the masters and infra nodes need to have a floating IP each as the load balancer needs to reach the masters and infra nodes from outside.

2.11.3. Bastion Configuration for Red Hat OpenShift Container Platform

The following subsections describe all the steps needed to properly configure the *bastion* instance.

2.11.3.1. Configure ~/.ssh/config to use Bastion as Jumphost

To easily connect to the Red Hat OpenShift Container Platform environment, follow the steps below.

On the Red Hat OpenStack Platform director node or local workstation with private key, <keypair-name>.pem:

```
$ exec ssh-agent bash

$ ssh-add /path/to/<keypair-name>.pem
Identity added: /path/to/<keypair-name>.pem (/path/to/<keypair-name>.pem)
```

Add to the **`~/.ssh/config`** file:

```
Host bastion
    HostName        <bastion_fqdn_hostname OR IP address>
    User            cloud-user
    IdentityFile    /path/to/<keypair-name>.pem
    ForwardAgent     yes
```

`ssh` into the bastion host with the **`-A`** option that enables forwarding of the authentication agent connection.

```
$ ssh -A cloud-user@bastion
```

Once logged into the bastion host, verify the ssh agent forwarding is working via checking for the **`SSH_AUTH_SOCK`**

```
$ echo "$SSH_AUTH_SOCK"
/tmp/ssh-NDFDQD02qB/agent.1387
```

Attempt to ssh into one of the Red Hat OpenShift Container Platform instances using the ssh agent forwarding.

NOTE

No password should be prompted if working properly.

```
$ ssh master1
```

2.11.3.2. Subscription Manager and Enabling Red Hat OpenShift Container Platform Repositories

Via the *bastion* instance, register the instance via Red Hat Subscription Manager. This is accomplished via using credentials or an activation key.

Via credentials:

```
$ sudo subscription-manager register --username <user> --password
'<password>'
```

OR

Via activation key:

```
$ sudo subscription-manager register --org="<org_id>" --activationkey=
<keyname>
```

Once registered, enable the following repositories as follows.

```
$ sudo subscription-manager repos \
    --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" \
    --enable="rhel-7-server-ansible-2.4-rpms"
```

Install the following package: **`atomic-openshift-utils`** via:

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

2.11.3.3. Configure Ansible

`ansible` is installed on the *bastion* instance to perform the registration, installation of packages, and the deployment of the Red Hat OpenShift Container Platform environment on the master and node instances.

Before running playbooks, it is important to create a *ansible.cfg* to reflect the deployed environment:

```
$ cat ~/ansible.cfg

[defaults]
forks = 20
host_key_checking = False
remote_user = cloud-user
roles_path = roles/
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
nocows = 1
callback_whitelist = profile_tasks

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o
UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=false
control_path = %(directory)s/%%h-%%r
pipelining = True
timeout = 10

[persistent_connection]
connect_timeout = 30
connect_retries = 30
connect_interval = 1
```

NOTE

The code block above can overwrite the default values in the file. Ensure to populate <keypair-name> with the keypair that was copied to the *bastion* instance.

2.11.3.4. Simple Ansible Inventory

Create a simple Ansible inventory file in order to facilitate the enablement of the required Red Hat OpenShift Container Platform repositories for all the instances. For this reference environment, a file labeled **`hosts`** with the the instances include:

```
$ cat ~/hosts

haproxy.example.com
master0.example.com
master1.example.com
master2.example.com
infra0.example.com
infra1.example.com
infra2.example.com
app0.example.com
app1.example.com
```

Ensure connectivity to all instances via the *bastion* instances via:

```
$ ansible all -i ~/hosts -m ping
```

Once connectivity to all instances has been established, register the instances via Red Hat Subscription Manager. This is accomplished via using credentials or an activation key.

Via credentials the **`ansible`** command is as follows:

```
$ ansible all -b -i ~/hosts -m command -a "subscription-manager register -
-username <user> --password '<password>'"
```

Via activation key, the **`ansible`** command is as follows:

```
$ ansible all -b -i ~/hosts -m command -a "subscription-manager register -
-org=<org_id> --activationkey=<keyname>"
```

where the following options:

- -b - referes to privileged escalation
- -i - location of inventory file
- -m - module to use
- -a - module argument

Once all the instances have been successfully registered, enable all the required RHOCP repositories on all the instances via:

```
$ ansible all -b -i ~/hosts -m command -a "subscription-manager repos \
      --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" \
      --enable="rhel-7-server-ansible-2.4-rpms""
```

2.11.3.5. OpenShift Authentication

Red Hat OpenShift Container Platform provides the ability to use many different authentication platforms. For this reference architecture, LDAP is the preferred authentication mechanism. A listing of other authentication options are available at Configuring Authentication and User Agent.

When configuring LDAP as the authentication provider the following parameters can be added to the ansible inventory. An example is shown below.

```
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=openshift,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=openshift,dc=com?uid?sub?
(memberOf=cn=ose-user,cn=groups,cn=accounts,dc=openshift,dc=com)'}]
```

NOTE

If using LDAPS, all the masters must have the relevant *ca.crt* file for LDAP in place prior to the installation, otherwise the installation fails. The file should be placed locally on the *bastion* instance and be called within the inventory file from the variable *openshift_master_ldap_ca_file*

2.11.3.6. Preparing the Inventory File

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. In the example below, some values are **bold** and must reflect the deployed environment from the previous chapter.

To gather the **`openshift_hosted_registry_storage_openstack_volumeID`** value use:

```
$ openstack volume show openshift-registry -f value -c id
```

NOTE

The above command runs on the Red Hat OpenStack Platform director node or the workstation that can access the Red Hat OpenStack Platform environment.

The **`openshift_cloudprovider_openstack_*`** values are required for Red Hat OpenShift Container Platform to be able to create Red Hat OpenStack Platform resources such as **`cinder`** volumes for persistent volumes. It is recommended to create a dedicated OpenStack user within the tenant as shown in Section 2.1, “Creating OpenStack User Accounts, Projects and Roles”.

```
$ cat ~/inventory

[OSEv3:children]
masters
etcd
nodes
lb

[OSEv3:vars]

ansible_ssh_user=cloud-user
deployment_type=openshift-enterprise
debug_level=2
openshift_vers=v3_9
openshift_enable_service_catalog=false
ansible_become=true
openshift_master_api_port=8443
openshift_master_console_port=8443
openshift_debug_level="{{ debug_level }}"
openshift_node_debug_level="{{ node_debug_level | default(debug_level,
true) }}"
openshift_master_debug_level="{{ master_debug_level | default(debug_level,
true) }}"
openshift_master_ldap_ca_file=/home/cloud-user/mycert.crt
openshift_master_identity_providers=[{'name': 'idm', 'challenge': 'true',
'login': 'true', 'kind': 'LDAPPasswordIdentityProvider', 'attributes':
{'id': ['dn'], 'email': ['mail'], 'name': ['cn'], 'preferredUsername':
['uid']}, 'bindDN': 'uid=admin,cn=users,cn=accounts,dc=example,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub?
(memberOf=cn=ose-user,cn=groups,cn=accounts,dc=openshift,dc=com)'}]
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_node_local_quota_per_fsgroup=512Mi
openshift_cloudprovider_kind=openstack
openshift_cloudprovider_openstack_auth_url=http://10.19.114.177:5000/v2.0
openshift_cloudprovider_openstack_username=<project-user>
openshift_cloudprovider_openstack_password=<password>
openshift_cloudprovider_openstack_tenant_name=openshift-tenant
openshift_master_cluster_hostname=openshift.example.com
openshift_master_cluster_public_hostname=openshift.example.com
openshift_master_default_subdomain=apps.example.com
os_sdn_network_plugin_name='redhat/openshift-ovs-networkpolicy'
osm_use_cockpit=true
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

#registry
openshift_hosted_registry_storage_kind=openstack
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
openshift_hosted_registry_storage_openstack_filesystem=ext4
openshift_hosted_registry_storage_openstack_volumeID=d5a1dfd6-3561-4784-
89d8-61217a018787
openshift_hosted_registry_storage_volume_size=15Gi
```

```
openshift_public_hostname=openshift.example.com

[masters]
master0.example.com
master1.example.com
master2.example.com

[etcd]
master0.example.com
master1.example.com
master2.example.com

[lb]
haproxy.example.com

[nodes]
master0.example.com openshift_node_labels="{'region': 'master'}"
openshift_hostname=master0.example.com
master1.example.com openshift_node_labels="{'region': 'master'}"
openshift_hostname=master1.example.com
master2.example.com openshift_node_labels="{'region': 'master'}"
openshift_hostname=master2.example.com
infra0.example.com  openshift_node_labels="{'region': 'infra'}"
openshift_hostname=infra0.example.com
infra1.example.com  openshift_node_labels="{'region': 'infra'}"
openshift_hostname=infra1.example.com
infra2.example.com  openshift_node_labels="{'region': 'infra'}"
openshift_hostname=infra2.example.com
app0.example.com  openshift_node_labels="{'region': 'app'}"
openshift_hostname=app0.example.com
app1.example.com  openshift_node_labels="{'region': 'app'}"
openshift_hostname=app1.example.com
```

NOTE

If openshift_cloudprovider_openstack_password contains a hash symbol, please ensure to double escape the password. For example, openshift_cloudprovider_openstack_password='"password###"'. For more information, visit: https://bugzilla.redhat.com/show_bug.cgi?id=1583523

NOTE

The following two parameters are added to the inventory file due to the the following bugzilla https://bugzilla.redhat.com/show_bug.cgi?id=1588768 oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version} openshift_examples_modify_imagestreams=true

2.11.3.7. Instance Verification

It can be useful to check for potential issues or misconfigurations in the instances before continuing the installation process. Connect to every instance using the *bastion* host and verify the disks are properly created and mounted, the **`cloud-init`** process finished successfully and verify for potential errors in the log files to ensure everything is ready for the Red Hat OpenShift Container Platform installation:

```
$ ssh bastion.example.com
$ ssh <instance>
$ lsblk
$ sudo journalctl
$ sudo less /var/log/cloud-init.log
$ free -m
$ sudo yum repolist
```

where *instance* is for example *master0.example.com*

For reference, below is example output of **`lsblk`** for the master nodes.

```
$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    253:0    0  40G  0 disk
└─vda1 253:1    0  40G  0 part /
vdb    253:16   0  15G  0 disk
vdc    253:32   0  30G  0 disk /var/lib/origin/openshift.local.volumes
vdd    253:48   0  25G  0 disk /var/lib/etcd
```

For reference, below is an example of output of **`lsblk`** for the infra and app nodes.

```
$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    253:0    0  40G  0 disk
└─vda1 253:1    0  40G  0 part /
vdb    253:16   0  15G  0 disk
vdc    253:32   0  30G  0 disk /var/lib/origin/openshift.local.volumes
```

NOTE

`vdb` on all nodes stores the docker images is not set as expected as this step is completed via the prerequisites playbook in the following section.

2.11.3.8. Red Hat OpenShift Container Platform Prerequisites Playbook

The Red Hat OpenShift Container Platform Ansible installation provides a playbook to ensure all prerequisites are met prior to the installation of Red Hat OpenShift Container Platform. This includes steps such as registering all the nodes with Red Hat Subscription Manager and setting up the docker on the docker volumes.

Via the **`ansible-playbook`** command on the *bastion* instance, ensure all the prerequisites are met using **`prerequisites.yml`** playbook:

```
$ ansible-playbook -i hosts /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 OpenShift RPM 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 installling Red Hat OpenShift Container Platform and reports a playbook recap showing the number of changes and errors (if any).

```
PLAY RECAP
*************************************************************************
*********
app1.example.com : ok=233  changed=40   unreachable=0    failed=0
app2.example.com : ok=233  changed=40   unreachable=0    failed=0
app3.example.com : ok=233  changed=40   unreachable=0    failed=0
infra1.example.com : ok=233  changed=40   unreachable=0    failed=0
infra2.example.com : ok=233  changed=40   unreachable=0    failed=0
infra3.example.com : ok=233  changed=40   unreachable=0    failed=0
localhost                  : ok=12   changed=0    unreachable=0
failed=0
master1.example.com : ok=674  changed=161  unreachable=0    failed=0
master2.example.com : ok=442  changed=103  unreachable=0    failed=0
master3.example.com : ok=442  changed=103  unreachable=0    failed=0

Tuesday 29 August 2018  10:34:49 -0400 (0:00:01.002)       0:29:54.775
********
=========================================================================
=====
openshift_hosted : Ensure OpenShift router correctly rolls out (best-
effort today) -- 92.44s
openshift_hosted : Ensure OpenShift registry correctly rolls out (best-
effort today) -- 61.93s
openshift_health_check ----------------------------------------------------
- 53.92s
openshift_common : Install the base package for versioning -------------
42.15s
openshift_common : Install the base package for versioning -------------
36.36s
openshift_hosted : Sanity-check that the OpenShift registry rolled out
correctly -- 31.43s
cockpit : Install cockpit-ws -------------------------------------------
27.65s
openshift_version : Get available atomic-openshift version -------------
25.27s
etcd_server_certificates : Install etcd --------------------------------
15.53s
openshift_master : Wait for master controller service to start on first
master -- 15.21s
openshift_master : pause -----------------------------------------------
```

```
- 15.20s
openshift_node : Configure Node settings --------------------------------
13.56s
openshift_excluder : Install openshift excluder ------------------------
13.54s
openshift_node : Install sdn-ovs package --------------------------------
13.45s
openshift_master : Create master config ---------------------------------
11.92s
openshift_master : Create the scheduler config -------------------------
10.92s
openshift_master : Create the policy file if it does not already exist --
10.65s
openshift_node : Install Node service file ------------------------------
10.43s
openshift_node : Install Node package -----------------------------------
10.39s
openshift_node : Start and enable node ---------------------------------
- 8.96s
```

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 Red Hat OpenStack Platform to Cloudforms

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

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

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 the following:

```
$ 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:

```
$ 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.
`ConfigContexts`	Check that each context in the client configuration is complete and has connectivity to its API server.

Diagnostic Name	Purpose
`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 **`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.

```
$ 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 **`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.
`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.

Check Name	Purpose
`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.
`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.

APPENDIX A. CREATING THE DNS RED HAT OPENSTACK PLATFORM INSTANCE

The installation process for Red Hat OpenShift Container Platform depends on a reliable name service that contains an address record for each of the target instances. When installing Red Hat OpenShift Container Platform in a cloud environment, the IP address of each instance is not known at the beginning. The address record for each instance must be created dynamically by the orchestration system as the instances themselves are created.

Red Hat OpenStack Platform does not have an integrated DNS service, but it is possible to create new records in a DNS service using dynamic updates as defined in RFC2137. This method uses a symmetric key to authenticate and authorize updates to a specific zone.

Installation of Red Hat OpenShift Container Platform on Red Hat OpenStack Platform requires a DNS service capable of accepting DNS update records for the new instances. This section describes a method to create a suitable DNS service within Red Hat OpenStack Platform. The DNS is capable of accepting DNS queries and update requests for the Red Hat OpenStack Platform service. It is suitable for delegation to make the Red Hat OpenShift Container Platform service accessible to users.

The following steps create a DNS Red Hat OpenStack Platform instance and configure it.

1. Source the RC file for the project

   ```
   $ source /path/to/examplerc
   ```

2. Create a security group to allow incoming connections to the DNS service (ports 53 UDP and TCP), **ssh** to manage the instance and ICMP for debug purposes.

   ```
   $ openstack security group create <dns-sg-name>
   $ openstack security group rule create \
       --ingress \
       --protocol icmp \
       <dns-sg-name>
   $ openstack security group rule create \
       --ingress \
       --protocol tcp \
       --dst-port 22 \
       <dns-sg-name>
   $ for PROTO in udp tcp;
   do
     openstack security group rule create \
       --ingress \
       --protocol $PROTO \
       --dst-port 53 \
       <dns-sg-name>;
   done
   ```

3. Create the Red Hat OpenStack Platform instance

   ```
   $ netid1=$(openstack network show <internal-network-name> -f value -
   c id)
   $ openstack server create \
       --nic net-id=$netid1 \
       --security-group=<dns-sg-name> \
       --flavor <flavor-name> \
   ```

```
    --image <image-name> \
    --key-name <keypair-name> <instance-name>
```

4. Assign and attach a floating IP to the instance

```
$ openstack floating ip create <public-network-name>
$ openstack server add floating ip <instance-name> <ip>
```

5. Log into the Red Hat OpenStack Platform instance that shall become the the DNS server.

```
$ ssh -i /path/to/<keypair-name>.pem cloud-user@<IP-of-dns-instance>
```

A.1. SETTING UP THE DNS RED HAT OPENSTACK PLATFORM INSTANCE

The following steps configure a DNS server once logged into the instance.

1. Ensure that *PEERDNS=no* and *DNS1=<forwarder_dns_ip>* within the */etc/sysconfig/network_scripts/ifcfg-eth0*

```
$ cat /etc/sysconfig/network_scripts/ifcfg-eth0
# Created by cloud-init on instance boot automatically, do not edit.
#
BOOTPROTO=dhcp
DEVICE=eth0
HWADDR=fa:16:3e:d0:2f:d8
ONBOOT=yes
TYPE=Ethernet
USERCTL=no
PEERDNS=no
DNS1=<forwarder_dns_ip>
```

2. Subscribe to the following repositories using **`subscription-manager`**

```
$ sudo subscription-manager register
$ sudo subscription-manager attach --pool <pool_id>
$ sudo subscription-manager repos --disable=*
$ sudo subscription-manager repos \
    --enable=rhel-7-server-rpms \
    --enable=rhel-7-server-extras-rpms
```

3. Subscribe to the EPEL repository

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

4. Install the following packages:

   - **`firewalld`**
   - **`python-firewall`**
   - **`bind-utils`**

- **bind**

```
$ sudo yum install -y firewalld python-firewall bind-utils bind
```

5. Enable the **firewalld** service and start it

```
$ sudo systemctl enable firewalld && sudo systemctl start firewalld
```

6. Add the DNS service to the **firewalld** public zone

```
$ sudo firewall-cmd --zone public --add-service dns
$ sudo firewall-cmd --zone public --add-service dns --permanent
```

7. Copy the **named.conf** prior to modification

```
$ sudo cp /etc/named.conf{,.orig}
```

The following **diff** command shows the difference between the original *named.conf* file and newest *named.conf* file. The **-** means the line was removed, while the **+** means the line has been added to the newest *named.conf*. When modifying the *named.conf* remove all lines with a **-** and include all lines with **+**.

```
$ diff -u /etc/named.conf.orig /etc/named.conf
--- /etc/named.conf.orig        2016-03-22 14:15:45.000000000 +0000
+++ /etc/named.conf     2017-05-03 16:21:00.579000000 +0000
@@ -6,17 +6,15 @@
 //
 // See /usr/share/doc/bind*/sample/ for example named configuration
files.
 //
-// See the BIND Administrator's Reference Manual (ARM) for details about
the
-// configuration located in /usr/share/doc/bind-{version}/Bv9ARM.html

 options {
-   listen-on port 53 { 127.0.0.1; };
-   listen-on-v6 port 53 { ::1; };
+   listen-on port 53 { any ; };
+   // listen-on-v6 port 53 { ::1; };
    directory   "/var/named";
    dump-file   "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
-   allow-query     { localhost; };
+   allow-query     { any ; };

    /*
     - If you are building an AUTHORITATIVE DNS server, do NOT enable
recursion.
@@ -28,18 +26,27 @@
       attacks. Implementing BCP38 within your network would greatly
       reduce such attack surface
    */
-   recursion yes;
+   // recursion yes;
+  forward only;
+  forwarders { <IP of DNS1 forwarder> ; <IP of DNS2
forwarder_if_more_than_1> ; } ;

-   dnssec-enable yes;
-   dnssec-validation yes;
+   // dnssec-enable yes;
+   // dnssec-validation yes;

    /* Path to ISC DLV key */
-   bindkeys-file "/etc/named.iscdlv.key";
+   /* In case you want to use ISC DLV, please uncomment the following
line. */
+   //bindkeys-file "/etc/named.iscdlv.key";

    managed-keys-directory "/var/named/dynamic";

    pid-file "/run/named/named.pid";
    session-keyfile "/run/named/session.key";
+
+   /* https://fedoraproject.org/wiki/Changes/CryptoPolicy */
+  //include "/etc/crypto-policies/back-ends/bind.config";
+
+        /* tickle slaves to pull updates */
+        notify yes ;
 };

 logging {
@@ -54,6 +61,10 @@
    file "named.ca";
 };

+
 include "/etc/named.rfc1912.zones";
 include "/etc/named.root.key";

+include "/etc/named/zones.conf" ;
```

Once the changes have been made, save the the *named.conf* file.

Next, create a *zones.conf* file that provides the zone that is to be used for the RHOCP installation. An example of a *zones.conf* is shown below.

/etc/named/zones.conf

```
include "/etc/named/update.key" ;

zone example.com {
  type master ;
  file "/var/named/dynamic/zone.db" ;
  allow-update { key update-key ; } ;
};
```

In order to update the DNS, the DNS requires a special key string that can be generated by **`rndc-confgen`** which is part of the **`bind`** RPM. The *update.key* file should exist within the */etc/named* directory.

To create the *update.key* file run the following command:

```
$ sudo rndc-confgen -a -c /etc/named/update.key -k update-key -r
/dev/urandom
$ sudo chown root.named /etc/named/update.key
$ sudo chmod 640 /etc/named/update.key
```

Example output:

```
$ sudo cat /etc/named/update.key
key "update-key" {
    algorithm hmac-md5;
    secret "key string"; # don't use this
};
```

WARNING

The significant part of the key output is the secret field. This should be kept secret as someone may make edits to the DNS entries with the key.

Once the *zones.conf* file has been created, one must update using nsupdate to contain all the Red Hat OpenShift Container Platform hosts and their IP addresses.

NOTE

Due to not having any Red Hat OpenStack Platform instances created yet, thus no IP addresses, return to this section for completing the input of IPs required within the *zones.db* file once Section 2.9, “Creating RHOSP Instances for RHOCP” section is completed.

Once the instances have been created, and IP addresses are obtained, update the DNS entries for zone *example.com* and zone *apps.example.com* as follows:

```
nsupdate -k /etc/named/update.key <<EOF
server <dns-server-ip>
zone example.com
update add bastion.example.com 3600 A <internal-net-ip>
update add app0.example.com 3600 A <internal-net-ip>
update add app1.example.com 3600 A <internal-net-ip>
update add master0.example.com 3600 A <internal-net-ip>
update add master1.example.com 3600 A <internal-net-ip>
update add master2.example.com 3600 A <internal-net-ip>
update add infra0.example.com 3600 A <internal-net-ip>
update add infra1.example.com 3600 A <internal-net-ip>
update add infra2.example.com 3600 A <internal-net-ip>
update add dns.example.com 3600 A <public-ip>
update add haproxy.example.com 3600 A <public-ip>
update add openshift.example.com 3600 A <haproxy-public-ip>
send
quit
EOF
```

```
nsupdate -k /etc/named/update.key <<EOF
server <dns-server-ip>
zone apps.example.com
update add * 3600 A <infra0_public_IP>
update add * 3600 A <infra1_public_IP>
update add * 3600 A <infra2_public_IP>
send
quit
EOF
```

Example of the */var/named/dynamic/zone.db* file:

/var/named/dynamic/zone.db

```
$ORIGIN .
$TTL 300        ; 5 minutes
example.com             IN SOA  dns.example.com. admin.example.com. (
                                16         ; serial
                                28800      ; refresh (8 hours)
                                3600       ; retry (1 hour)
                                604800     ; expire (1 week)
                                86400      ; minimum (1 day)
                                )
                        NS      dns.example.com.
$ORIGIN apps.example.com.
*                       A       <infra0_public_IP>
                        A       <infra1_public_IP>
                        A       <infra2_public_IP>
$ORIGIN example.com.
app0              A       <internal-net-IP>
app1              A       <internal-net-IP>
bastion                 A       <internal-net-IP>
dns                     A       <public_IP>
haproxy                 A       <public_IP>
infra0              A       <internal-net-IP>
infra1              A       <internal-net-IP>
infra2              A       <internal-net-IP>
master0                 A       <internal-net-IP>
master1                 A       <internal-net-IP>
master2                 A       <internal-net-IP>
openshift               A       <haproxy-public_IP>
```

WARNING

This file should not be updated manually, please use *nsupdate* for changes.

Once everything is set, configure *named* service to start at boot and start it

```
$ sudo systemctl enable named && sudo systemctl start named
```

Verification of the entire zone can be done using the **`dig`** command. The example below shows the records of the DNS **`dns.example.com`**

```
$ dig @<DNS_IP> example.com axfr
;; Connection to ::1#53(::1) for example.com failed: connection refused.

; <<>> DiG 9.9.4-RedHat-9.9.4-50.el7 <<>> @localhost example.com axfr
; (2 servers found)
;; global options: +cmd
example.com.            300     IN      SOA     dns.example.com.
admin.example.com. 16 28800 3600 604800 86400
example.com.            300     IN      NS      dns.example.com.
app0.example.com. 300     IN      A       <internal-net-IP>
app1.example.com. 300     IN      A       <internal-net-IP>
*.apps.example.com.     300     IN      A       <infra0-public-IP>
*.apps.example.com.     300     IN      A       <infra1-public-IP>
*.apps.example.com.     300     IN      A       <infra2-public-IP>
bastion.example.com.    300     IN      A       <internal-net-IP>
dns.example.com.        300     IN      A       <public-IP>
haproxy.example.com.    300     IN      A       <public-IP>
infra0.example.com. 300   IN      A       <internal-net-IP>
infra1.example.com. 300   IN      A       <internal-net-IP>
infra2.example.com. 300   IN      A       <internal-net-IP>
master0.example.com.   300     IN      A       <internal-net-IP>
master1.example.com.   300     IN      A       <internal-net-IP>
master2.example.com.   300     IN      A       <internal-net-IP>
openshift.example.com.  300     IN      A       <haproxy-public-IP>
example.com.            300     IN      SOA     dns.example.com.
admin.example.com. 16 28800 3600 604800 86400
;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Wed Apr 04 17:44:54 EDT 2018
;; XFR size: 19 records (messages 1, bytes 513)
```

APPENDIX B. USING FLANNEL

As an alternate to the default SDN, Red Hat OpenShift Container Platform provides Ansible playbooks for installing **flannel**-based networking. This is useful when running Red Hat OpenShift Container Platform within a cloud provider platform that relies on SDN, such as Red Hat OpenShift Container Platform, and want to avoid encapsulating packets twice through both platforms.

NOTE

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

Flannel uses a single IP network space for all of the containers allocating a contiguous subset of the space to each instance. Consequently, nothing prevents a container from attempting to contact any IP address in the same network space. This hinders multi-tenancy because the network cannot be used to isolate containers in one application from another.

Depending on preference, isolation or performance, determine the appropriate choice when deciding between the different OpenShift SDN plugins and flannel options for internal networks.

IMPORTANT

The current version of **`neutron`** enforces port security on ports by default. This prevents the port from sending or receiving packets with a MAC address different from that on the port itself. Flannel creates virtual MACs and IP addresses and must send and receive packets on the port, thus port security must be disabled on the ports that carry flannel traffic.

To enable flannel within an Red Hat OpenShift Container Platform cluster:

1. **`neutron`** port security controls must be configured to be compatible with Flannel. The default configuration of Red Hat OpenShift Container Platform disables user control of **`port_security`**. Configure **`netruon`** to allow users to control the **`port_security`** setting on individual ports.

   a. On the **`neutron`** servers, add the following to the ***/etc/neutron/plugins/ml2/ml2_conf.ini*** file:

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

   b. Then, restart the **`neutron`** services:

   ```
   service neutron-dhcp-agent restart
   service neutron-ovs-cleanup restart
   service neutron-metadata-agentrestart
   service neutron-l3-agent restart
   service neutron-plugin-openvswitch-agent restart
   service neutron-vpn-agent restart
   service neutron-server  restart
   ```

2. When creating the Red Hat OpenShift Container Platform instances on Red Hat OpenStack Platform, disable both port security and security groups in the ports where the container network Flannel interface resides:

```
neutron port-update $port --no-security-groups --port-security-
enabled=False
```

NOTE

Flannel gather information from etcd to configure and assign the subnets in the nodes. Therefore, the security group attached to the etcd hosts should allow access from nodes to port 2379/tcp, and nodes security group should allow egress communication to that port on the etcd hosts.

a. Set the following variables in the Ansible inventory file before running the installation:

```
openshift_use_openshift_sdn=false 1
openshift_use_flannel=true 2
flannel_interface=eth0
```

1 Set **`openshift_use_openshift_sdn`** to **`false`** to disable the default SDN.

2 Set **`openshift_use_flannel`** to **`true`** to enable **flannel** in place.

b. Optionally, specify the interface to use for inter-host communication using the **`flannel_interface`** variable. Without this variable, the Red Hat OpenShift Container Platform installation uses the default interface.

NOTE

Custom networking CIDR for pods and services using flannel are to be supported in a future release. **BZ#1473858**

3. After the Red Hat OpenShift Container Platform installation, add a set of iptables rules on every Red Hat OpenShift Container Platform node:

```
iptables -A DOCKER -p all -j ACCEPT
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

To persist those changes in the ***/etc/sysconfig/iptables*** use the following command on every node:

```
cp /etc/sysconfig/iptables{,.orig}
sh -c "tac /etc/sysconfig/iptables.orig | sed -e '0,/:DOCKER -/
s/:DOCKER -/:DOCKER ACCEPT/' | awk '"\!"p && /POSTROUTING/{print \"-
A POSTROUTING -o eth1 -j MASQUERADE\"; p=1} 1' | tac >
/etc/sysconfig/iptables"
```

NOTE

The **`iptables-save`** command saves all the current *in memory* iptables rules. However, because Docker, Kubernetes and Red Hat OpenShift Container Platform create a high number of iptables rules (services, etc.) not designed to be persisted, saving these rules can become problematic.

To isolate container traffic from the rest of the Red Hat OpenShift Container Platform traffic, Red Hat recommends creating an isolated tenant network and attaching all the nodes to it. If using a different network interface (eth1), ensure to configure the interface to start at boot time through the ***/etc/sysconfig/network-scripts/ifcfg-eth1*** file:

```
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
ONBOOT=yes
DEFTROUTE=no
PEERDNS=no
```