Reference Architectures 2018

Deploying and Managing OpenShift 3.9 on Red Hat Virtualization 4

Last Updated: 2018-07-27

Reference Architectures 2018 Deploying and Managing OpenShift 3.9 on Red Hat Virtualization 4

Chandler Wilkerson
refarch-feedback@redhat.com

Legal Notice

Abstract

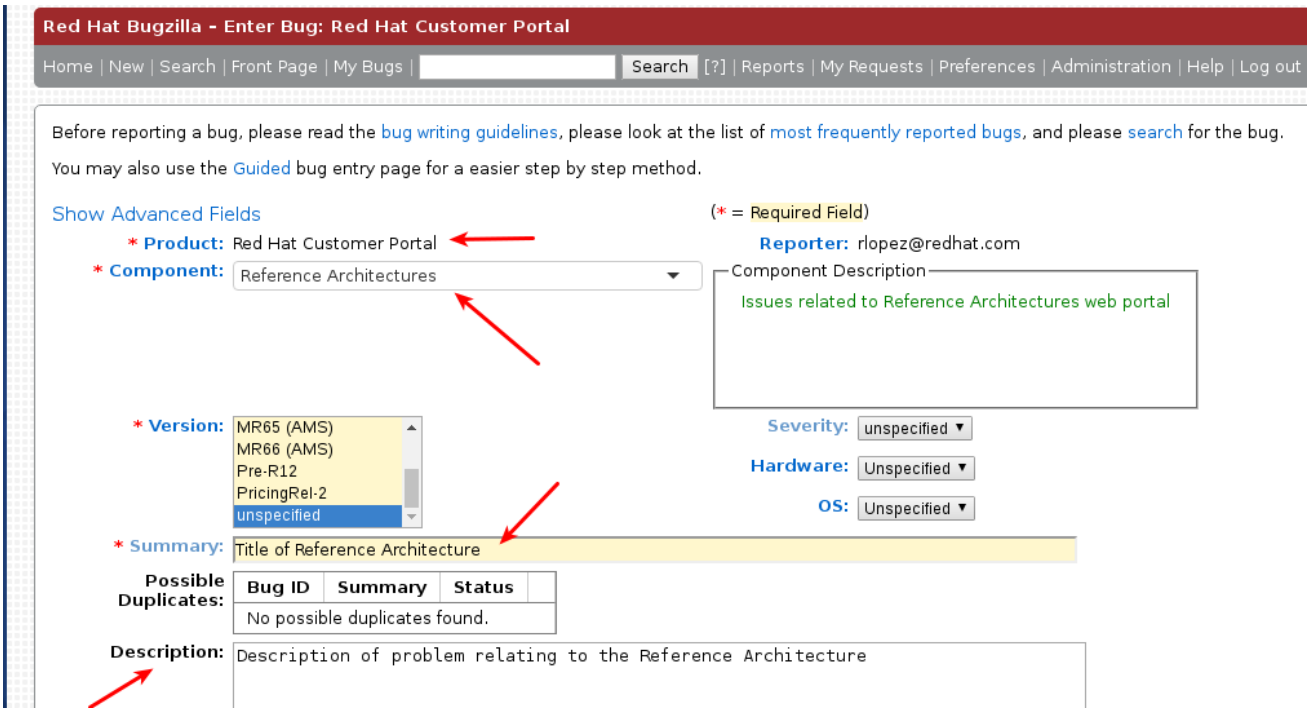
The purpose of this document is to provide guidelines and considerations for deploying and managing Red Hat OpenShift Container Platform on Red Hat Virtualization.

Table of Contents

COMMENTS AND FEEDBACK

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

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



EXECUTIVE SUMMARY

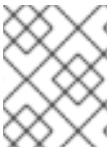
Staying ahead of the needs of an increasingly connected and demanding customer base demands solutions which are not only secure and supported, but robust and scalable, where new features may be delivered in a timely manner. In order to meet these requirements, organizations must provide the capability to facilitate faster development life cycles by managing and maintaining multiple products to meet each of their business needs. Red Hat solutions — for example Red Hat OpenShift Container Platform on Red Hat Virtualization — 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 Virtualization 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 Virtualization 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 Virtualization 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:

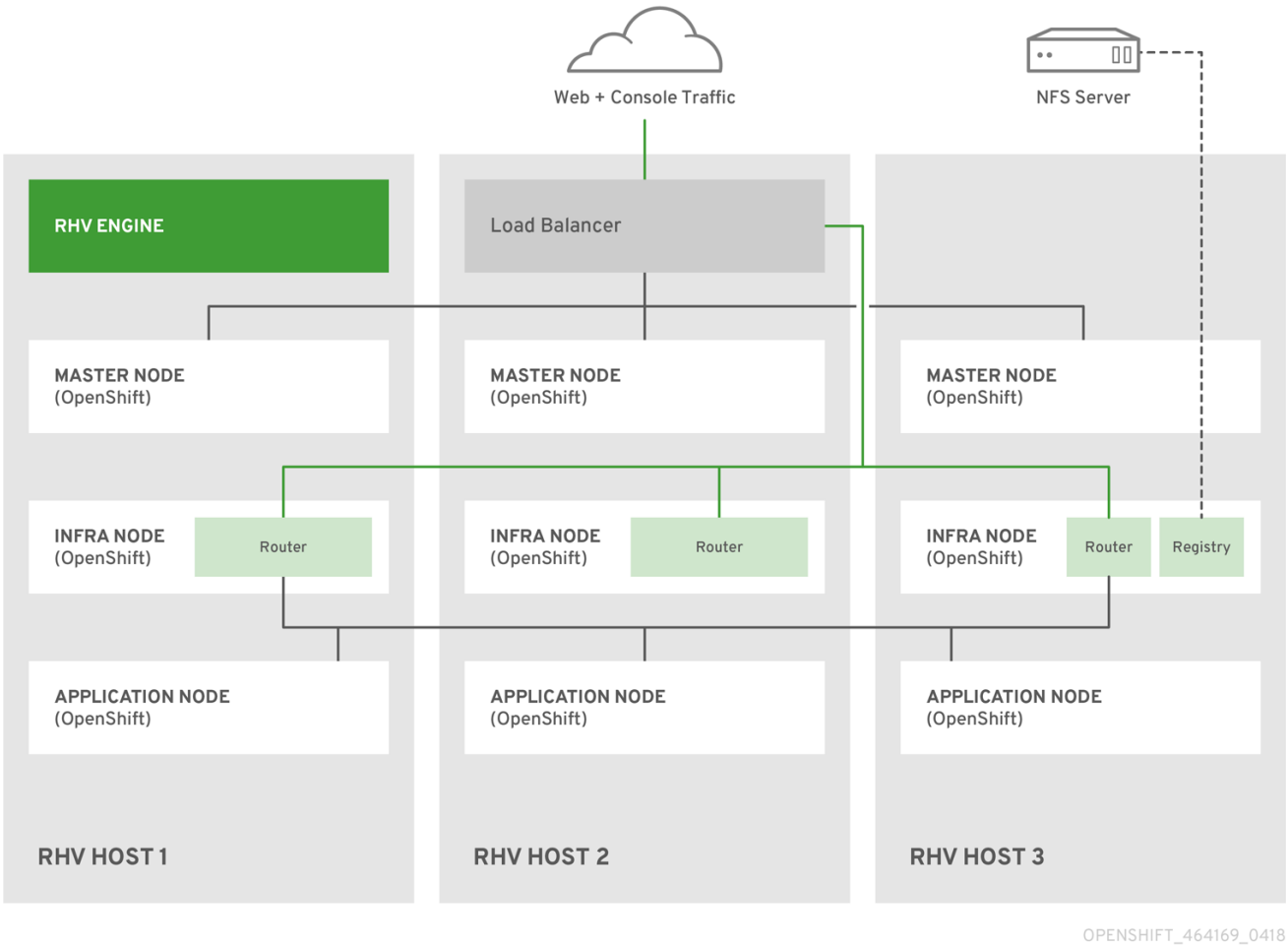
- *Which installation method do you want to use?*
- *How many instances do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other Red Hat technologies?*

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 VIRTUALIZATION ENVIRONMENT CONSIDERATIONS

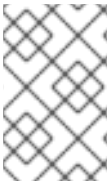
Employing Red Hat Virtualization in a self-hosted configuration, this reference architecture provides a complete high-availability environment in three physical hosts. In production environments, best practices recommend separating the Red Hat Virtualization cluster running Red Hat OpenShift Container Platform from the cluster running the management engine. Proper set up of the Red Hat Virtualization environment is beyond the scope of this document. A guide to best practices in setting up Red Hat Virtualization can be found in the Appendix C, *Links* section of the Appendix at the end of this document.

1.1.1. Hardware Requirements

The Red Hat Virtualization cluster should allow for the following resources available to virtual machines created by this reference architecture:

Table 1.1. Hypervisor System Requirements

CPU Cores per node	Memory per node	Disk (total)
8	40GiB	1125GiB



NOTE

The numbers above are based on node count and recommended minimums. For a proof of concept, they may be reduced by disabling the relevant set of (OpenShift Pre-Install Checks

1.1.2. oVirt Ansible

Virtual machines in this reference architecture are instantiated, deployed, and configured within Red Hat Virtualization using Ansible roles provided by the oVirt project. These roles supplement the oVirt modules included with Ansible by managing the orchestration of complex tasks such as uploading a QCOW2 disk image, creating a template, and instantiating a number of virtual machines with specific hardware. Included in the oVirt modules and roles is the ability to pass Cloud-Init parameters, which this reference architecture uses to ensure the virtual machines are registered to Red Hat Network and have Docker local volumes formatted and mounted at first boot.

Instructions for installing the oVirt Ansible roles (and their required prerequisite packages) are provided at The oVirt-Ansible GitHub Repository

1.1.3. Template OS Image

To save time and disk space provisioning the virtual machines for this reference architecture, a pre-installed virtual machine disk image in QCOW2 format is used to create a template within Red Hat Virtualization. The virtual machines for the Red Hat OpenShift Container Platform installation are then created from this template, and node labels direct the Red Hat OpenShift Container Platform installer in which roles run on which nodes.

The **`oVirt.image-template`** role handles the introduction of the QCOW2 image into Red Hat Virtualization in three steps:

- The variable **`qcow_url`** is downloaded to a file named in **`image_path`**, by default

`/tmp/ovirt_image_data`. If the file already exists, the download is skipped.

- The downloaded image is checked for proper QCOW2 format.
- The image is uploaded to Red Hat Virtualization.

To download a suitable Red Hat Enterprise Linux image:

- Log in at https://access.redhat.com/.
- Navigate to **Downloads**, then **Red Hat Enterprise Linux**.
- Select the latest release (7.5 at the time of this writing).
- Copy the URL for **KVM Guest Image**.

NOTE

While it is possible to provide as the `qcow_url` the download link provided by the Red Hat Access Portal, these links expire after a while. It is recommended to download the file to the instance from which the Ansible scripts will run (see Section 1.5, “Bastion Instance” for details), and provide a random URL. An example of using a template named file from `~/Downloads` is provided in the examples.

1.1.4. Storage Domain

Storage in this reference architecture makes use of a Red Hat Virtualization storage domain for virtual machine local disk images, and a network file system (NFS) server for the OpenShift Registry.

NOTE

Use of NFS for the OpenShift Registry in this reference architecture is sufficient for a proof of concept cluster. The OpenShift Installation Documentation warns against using NFS to back the OpenShift Registry at scale and recommends use of the `no_wdelay` server side parameter to ensure read-after-write consistency.

1.1.5. Network

Networking in the Red Hat Virtualization environment is simplified for the reference architecture, employing the same `ovirtmgmt` network used by RHV for its engine.

1.1.6. Affinity Groups

Affinity groups map virtual machines to hosts by certain logical rules. These rules may be grouped into *positive* and *negative* affinities.

Positive affinity, where a particular virtual machine always runs on a certain host or on the same host as like-grouped virtual machines, may be used to ensure resources required by a virtual machine are kept available.

This reference architecture employs *negative affinity* to enforce high-availability by constraining virtual machines with similar roles from running on the same host. This helps ensure a single host outage does not bring down all the master nodes or infrastructure nodes for the cluster.

1.2. 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.

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.

1.2.1. 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 1.2. Load Balancer DNS records

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

1.3. ROUTER

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

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

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

- HTTP
- HTTPS (with SNI)
- WebSockets
- TLS with SNI

The router utilizes the wildcard zone specified during the installation and configuration of OpenShift. This wildcard zone is used by the router to create routes for a service running within the OpenShift environment to a publicly accessible URL. The wildcard zone itself is a wildcard entry in **Route53** which is linked using a CNAME to an **ELB** which performs a health check and forwards traffic to router pods on port 80 and 443.

1.4. 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.

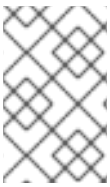
The destination for the master and application traffic must be set in the load balancer configuration after each instance is created, the floating IP address is assigned and before the installation. A single **haproxy** load balancer can forward both sets of traffic to different destinations.

1.5. 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.

1.6. RED HAT OPENSHIFT CONTAINER PLATFORM COMPONENTS

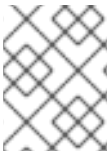
Red Hat OpenShift Container Platform comprises of multiple instances running on Red Hat Virtualization that allow for scheduled and configured OpenShift services and supplementary containers. These containers can have persistent storage, if required, by the application and integrate with optional OpenShift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

1.6.1. 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.

1.6.1.1. 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.

1.6.1.2. 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'.

1.6.1.3. 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'}"
```

1.6.2. etcd

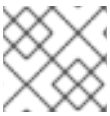
etcd is a consistent and highly-available key value store used as Red Hat OpenShift Container Platform's backing store for all cluster data. **etcd** stores the persistent master state while other components watch **etcd** for changes to bring themselves into the desired state.

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.

1.6.3. 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.

1.6.3.1. 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.

1.6.3.2. 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.

1.7. SOFTWARE DEFINED NETWORKING

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

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

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

1.7.1. OpenShift SDN Plugins

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

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

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

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

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

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

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

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

1.8. 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.

1.9. 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.

1.9.1. 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 Virtualization or from **glusterfs** if deployed.

1.9.1.1. 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**.

1.10. 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.

1.11. 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
```

1.12. AGGREGATED METRICS

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

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

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

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

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

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

1.13. CONTAINER-NATIVE STORAGE (OPTIONAL)

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

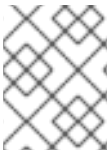
1.13.1. Prerequisites for Container-Native Storage

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

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

1.13.2. Firewall and Security Group Prerequisites

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



NOTE

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

Table 1.3. CNS - Inbound

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

CHAPTER 2. PREPARING INSTANCES FOR RED HAT OPENSHIFT CONTAINER PLATFORM

A successful deployment of Red Hat OpenShift Container Platform requires a number of prerequisites. These prerequisites include the deployment of instances in Red Hat Virtualization and the required configuration steps prior to the actual installation of Red Hat OpenShift Container Platform using Ansible. The subsequent sections detail the prerequisites and configuration changes required for an Red Hat OpenShift Container Platform on Red Hat Virtualization environment.

2.1. BASTION INSTANCE

The *bastion* instance itself may have a relatively minimal installation of Red Hat Enterprise Linux 7. It should meet recommended requirements for the operating system itself with enough spare disk space to hold downloaded QCOW2 images (around 700 MiB).

The Ansible commands to set up the RHV instances and OpenShift may be run as a normal user. Several of the following prerequisite tasks, however, require root access, preferably through sudo.

2.1.1. Generate the SSH Key Pair

Ansible works by communicating with target servers via the **`Secure Shell (SSH)`** protocol. **`SSH`** keys are used in place of passwords in the Red Hat OpenShift Container Platform installation process. The public key will be installed on Red Hat Virtualization instances by Cloud-Init.

Employ an **`ssh-agent`** on the *bastion* instance to avoid repeatedly being asked for the pass phrase.

SSH Key Generation

If **SSH** keys do not currently exist then it is required to create them. Generate an RSA key pair with a blank pass phrase by typing the following at a shell prompt:

```
$ ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa
```

A message similar to the following prints to indicate the keys are created successfully.

```
Your identification has been saved in ~/.ssh/id_rsa.
Your public key has been saved in ~/.ssh/id_rsa.pub.
The key fingerprint is:
e7:97:c7:e2:0e:f9:0e:fc:c4:d7:cb:e5:31:11:92:14 USER@bastion.example.com
The key's randomart image is:
+--[ RSA 2048]----+
|             E.  |
|            . .  |
|             o . |
|              . .|
|        S .    . |
|         + o o ..|
|          * * +oo|
|           O +..=|
|           o*  o.|
+-----------------+
```

2.1.2. Subscribe the Bastion

If the bastion instance is not already registered, do so, then list available subscription pools:

```
$ sudo subscription-manager register
$ sudo subscription-manager list --available
```

Note the *Pool IDs* of available subscriptions for both Red Hat Virtualization and Red Hat OpenShift Container Platform, then attach those pools:

```
$ sudo subscription-manager attach --pool <RHV Pool ID>
$ sudo subscription-manager attach --pool <RHOCP Pool ID>
```

Ensure the following repositories are enabled, and no others.

```
$ sudo subscription-manager repos --disable=*
$ 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-server-rhv-4-mgmt-agent-rpms
```

2.1.3. Install Packages

Install the following packages to enable the installation and deployment of Red Hat OpenShift Container Platform:

```
$ sudo yum install -y \
    ansible \
    atomic-openshift-utils \
    git \
    python-ovirt-engine-sdk4 \
    ovirt-ansible-roles
```

2.1.4. (Optional) Clone OpenShift Ansible Contrib GitHub Repository

Example variable and inventory files relevant to this reference architecture may be found in the OpenShift Ansible Contrib GitHub Repository at https://github.com/openshift/openshift-ansible-contrib/tree/master/reference-architecture/rhv-ansible

NOTE

Code and examples in the OpenShift Ansible Contrib GitHub Repository are unsupported by Red Hat.

2.2. CONFIGURE ANSIBLE

Before running playbooks, modify **`ansible.cfg`** to reflect the deployed environment:

```
$ sudo vi /etc/ansible/ansible.cfg
```

The code block below implements the following changes:

- **forks** is increased to twenty from the default five. This allows tasks on the ten instances in this reference architecture to run simultaneously.
- **host_key_checking** is set to false to allow Ansible commands to run on instances without first manually logging in to add them to the **~/.ssh/known_hosts** file. An alternative is to use **ssh-keyscan** to generate the known-hosts entries.
- **remote_user** is changed to **root** from the default of whichever user calls Ansible.
- **gathering** is changed from the default **implicit** to **smart** to enable caching of host facts.
- Logging is activated and retry files are disabled to provide more helpful debugging of issues during Ansible runs.
- A **vault_password_file** is specified for protected values like passwords. See the following section for more information on setting up an Ansible Vault file.

```
[defaults]
forks = 20
host_key_checking = False
remote_user = root
gathering = smart

log_path=./ansible.log
retry_files_enabled=False

vault_password_file=~/.test_vault_pw
```

2.3. VARIABLES AND STATIC INVENTORY

According to the Red Hat OpenShift Container Platform Advanced Installation Guide, cluster variables should be configured in the system inventory (**/etc/ansible/hosts**) within the [OSEv3:vars] section.

The oVirt Ansible roles may be configured in a separate variable file and included during **ansible-playbook** runs using the **-e** flag, however it is also possible to include these in the same inventory file under a group created just for **localhost**.

The following snippets of the inventory for this reference architecture define **localhost** as part of a group called **workstation**:

```
[workstation]
localhost ansible_connection=local
```

Variables assigned to **localhost** include:

Credentials for the RHV Engine

In the following snippet, the credentials for the RHV engine are assigned to special variables which are then encrypted by Ansible Vault because they contain sensitive information like administrative passwords. Creation of this Ansible Vault file is explained in the next section.

```
[workstation:vars]
# RHV Engine
engine_url="{{ vault_engine_url }}"
```

```
engine_user="{{ vault_engine_user }}"
engine_password="{{ vault_engine_password }}"
# CA file copied from engine:/etc/pki/ovirt-engine/ca.pem
# path is relative to playbook directory
engine_cafile=../ca.pem
```

Guest Image URL and Path

```
# QCOW2 KVM Guest Image
#qcow_url=https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-
GenericCloud.qcow2c
qcow_url=https://access.cdn.redhat.com//content/origin/files/XXXX/rhel-
server-7.5-x86_64-kvm.qcow2?_auth_=XXXX
template_name=rhel75
image_path="{{ lookup('env', 'HOME') }}/Downloads/{{ template_name
}}.qcow2"
```

RHV Cluster information and SSH key

```
# RHV VM Cluster Info
rhv_cluster=Default
rhv_data_storage=vmstore
root_ssh_key="{{ lookup('file', '~/.ssh/id_rsa.pub') }}"
```

Some variables pertain to all hosts, including **`localhost`** and the OpenShift instances. They belong in a special group called **`all`**:

```
[all:vars]
app_dns_prefix=apps
public_hosted_zone=example.com
load_balancer_hostname=lb.{{public_hosted_zone}}
openshift_master_cluster_hostname="{{ load_balancer_hostname }}"
openshift_master_cluster_public_hostname=openshift-master.{{
public_hosted_zone }}
openshift_master_default_subdomain="{{ app_dns_prefix }}.{{
public_hosted_zone }}"
openshift_public_hostname="{{openshift_master_cluster_public_hostname}}"
```

OpenShift specific variables belong in the previously mentioned **`[OSEv3:vars]`** section:

```
[OSEv3:vars]
# General variables
ansible_ssh_user=root
debug_level=2
deployment_type=openshift-enterprise
openshift_debug_level="{{ debug_level }}"
openshift_deployment_type="{{ deployment_type }}"
openshift_master_cluster_method=native
openshift_node_debug_level="{{ node_debug_level | default(debug_level,
true) }}"
openshift_release=3.9
```

As Red Hat Virtualization does not yet provide dynamic storage allocation capabilities to Red Hat OpenShift Container Platform, the Service Catalog must be disabled at install time:

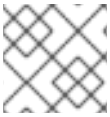
```
openshift_enable_service_catalog=False
```

For easier readability, the inventory is broken down further into logical sections:

```
# Docker
container_runtime_docker_storage_setup_device=/dev/vdb
container_runtime_docker_storage_type=overlay2
openshift_docker_use_system_container=False
openshift_node_local_quota_per_fsgroup=512Mi
openshift_use_system_containers=False
```

```
# Pod Networking
os_sdn_network_plugin_name=redhat/openshift-ovs-networkpolicy
```

Here, an external NFS server is used as the backing store for the OpenShift Registry.



NOTE

A value is required for **`openshift_hosted_registry_storage_host`**.

```
# Registry
openshift_hosted_registry_replicas=1
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_selector='region=infra'
openshift_hosted_registry_storage_host=
openshift_hosted_registry_storage_nfs_directory=/var/lib/exports
openshift_hosted_registry_storage_volume_name=registryvol
openshift_hosted_registry_storage_volume_size=20Gi
```

```
# Red Hat Subscription Management
rhsub_pool=Red Hat OpenShift Container Platform*
rhsub_user="{{ vault_rhsub_user }}"
rhsub_password="{{ vault_rhsub_password }}"
```

The embedded json here was converted from yaml. The yaml version of the inventory is also in the OpenShift Ansible Contrib GitHub Repo.

```
# Load Balancer Config
openshift_loadbalancer_additional_frontends=[{"name":"apps-
http","option":"tcplog","binds":["*:80"],"default_backend":"apps-http"},
{"name":"apps-https","option":"tcplog","binds":
["*:443"],"default_backend":"apps-http"}]
openshift_loadbalancer_additional_backends=[{"name":"apps-
http","balance":"source","servers":[{"name":"infra0","address":"{{
groups['infras'].0 }}:80","opts":"check"},{"name":"infra1","address":"{{
groups['infras'].1 }}:80","opts":"check"},{"name":"infra2","address":"{{
groups['infras'].2 }}:80","opts":"check"}]},{"name":"apps-
https","balance":"source","servers":[{"name":"infra0","address":"{{
groups['infras'].0 }}:443","opts":"check"},{"name":"infra1","address":"{{
groups['infras'].1 }}:443","opts":"check"},{"name":"infra2","address":"{{
groups['infras'].2 }}:443","opts":"check"}]}]
```

Finally, the host names for the OpenShift instances are grouped and provided below:

```
[OSEv3:children]
nodes
masters
etcd
lb
```

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

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

[infras]
infra0.example.com
infra1.example.com
infra2.example.com

[lb]
lb.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': 'primary'}"
openshift_hostname=app0.example.com
app1.example.com openshift_node_labels="{'region': 'primary'}"
openshift_hostname=app1.example.com
app2.example.com openshift_node_labels="{'region': 'primary'}"
openshift_hostname=app2.example.com
```

For the complete inventory, see https://github.com/openshift/openshift-ansible-contrib/blob/master/reference-architecture/rhv-ansible/example/inventory

2.4. ANSIBLE VAULT

To avoid leaving secret passwords for critical infrastructure in clear-text on the file system, create an encrypted *Ansible Vault* variable file for inclusion in the Ansible commands in the next chapter.

To create the encrypted file, for example in **`~/vault.yaml`**, run the following command:

```
$ ansible-vault create ~/vault.yaml
```

To edit the newly created vault file, run the following command:

```
$ ansible-vault edit ~/vault.yaml
```

This reference architecture employs the following process:

- Identify a variable which requires protection, e.g. **engine_password**
- In the vault file, create an entry prefixed by **vault_** for the variable, e.g. **vault_engine_password**
- In the variables file, assign the vault variable to the regular variable:

```
engine_password: '{{ vault_engine_password }}'
```

If using a vault file to protect some values, it must be included alongside the variables file that references it, e.g. **-e @~/vault.yaml**

2.5. RED HAT VIRTUALIZATION ENGINE CREDENTIALS AND CA

The **oVirt Ansible** roles responsible for setting up instances in Red Hat Virtualization require credentials in order to connect to the engine and perform operations.

```
engine_url: https://engine.example.com/ovirt-engine/api
engine_user: admin@internal
engine_password: '{{ vault_engine_password }}'
```

In order to establish a trusted SSL connection with the engine, the **oVirt** API requires a copy of the engine's certificate authority in the form of a self-signed certificate *PEM*.

To download the certificate from the engine itself, run:

```
$ curl --output ca.pem 'http://engine.example.com/ovirt-
engine/services/pki-resource?resource=ca-certificate&format=X509-PEM-CA'
```

Provide the CA file to **oVirt Ansible** with the following variable:

```
engine_cafile: ../ca.pem
```

NOTE

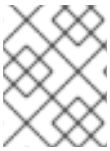
The path of the CA file is relative to the playbook where the **oVirt** roles are run. Typically this is in a subdirectory of the path where the CA file is downloaded, thus the **../**. If everything is in the same directory, just use **ca.pem** instead.

2.6. OPENSHIFT AUTHENTICATION

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

When configuring LDAP as the authentication provider, the following parameters can be added to the Ansible inventory:

```
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=ocp3,dc=openshift,dc=com',
'bindPassword': 'ldapadmin', 'ca': '/etc/origin/master/ca.crt',
'insecure': 'false', 'url':
'ldap://ldap.ocp3.example.com/cn=users,cn=accounts,dc=ocp3,dc=openshift,dc
=com?uid?sub?(memberOf=cn=ose-
user,cn=groups,cn=accounts,dc=ocp3,dc=openshift,dc=com)'}]
```



NOTE

If using LDAPS, all master nodes must have the relevant *ca.crt* file in place prior to the installation, otherwise the installation fails.

For more information about the required LDAP parameters, see LDAP Authentication

2.7. DEPLOY RED HAT VIRTUALIZATION INSTANCES

Creation of OpenShift instances in Red Hat Virtualization takes place in two *oVirt Ansible* roles: **`oVirt.image-template`** and **`oVirt.vm-infra`**. Documentation and examples of these roles can be found on their respective GitHub project pages.

The following is a playbook which calls the *Image Template* and *VM Infrastructure* roles from *oVirt Ansible*:

ovirt-vm-infra.yaml

```
---
- name: oVirt 39 all-in-one playbook
  hosts: localhost
  connection: local
  gather_facts: false
  roles:
    - oVirt.image-template
    - oVirt.vm-infra
  vars:
    # Common
    compatibility_version: 4.2
    data_center_name: Default
    debug_vm_create: true
    wait_for_ip: true
    vm_infra_wait_for_ip_retries: 10
    vm_infra_wait_for_ip_delay: 40

    # oVirt Image Template vars
```

```
    template_cluster: "{{ rhv_cluster }}"
    template_memory: 8GiB
    template_cpu: 1
    template_disk_storage: "{{ rhv_data_storage }}"
    template_disk_size: 60GiB
    template_nics:
      - name: nic1
        profile_name: ovirtmgmt
        interface: virtio

    # Profiles
    master_vm:
      cluster: "{{ rhv_cluster }}"
      template: "{{ template_name }}"
      memory: 16GiB
      cores: 2
      high_availability: true
      disks:
        - size: 100GiB
          storage_domain: "{{ rhv_data_storage }}"
          name: docker_disk
          interface: virtio
        - size: 50GiB
          storage_domain: "{{ rhv_data_storage }}"
          name: localvol_disk
          interface: virtio
      state: running

    node_vm:
      cluster: "{{ rhv_cluster }}"
      template: "{{ template_name }}"
      memory: 8GiB
      cores: 2
      disks:
        - size: 100GiB
          storage_domain: "{{ rhv_data_storage }}"
          name: docker_disk
          interface: virtio
        - size: 50GiB
          storage_domain: "{{ rhv_data_storage }}"
          name: localvol_disk
          interface: virtio
      state: running

    # Cloud Init Script
    cloud_init_script: |
      runcmd:
        - mkdir -p '/var/lib/origin/openshift.local.volumes'
        - /usr/sbin/mkfs.xfs -L localvol /dev/vdc
        - sleep "$(($RANDOM % 60))"
        - sync
        - reboot
      mounts:
        - [ '/dev/vdc', '/var/lib/origin/openshift.local.volumes', 'xfs',
'defaults,gquota' ]
      rh_subscription:
```

```
        username: {{vault_rhsub_user}}
        password: {{vault_rhsub_password}}
        add-pool: [{{vault_rhsub_pool}}]
        server-hostname: {{vault_rhsub_server}}
        enable-repo: ['rhel-7-server-rpms', 'rhel-7-server-extras-rpms',
'rhel-7-fast-datapath-rpms', 'rhel-7-server-ose-3.9-rpms']
        disable-repo: []

    vms:
      # Master VMs
      - name: "master0.{{ public_hosted_zone }}"
        profile: "{{ master_vm }}"
        tag: openshift_master
        cloud_init:
          host_name: "master0.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"
      - name: "master1.{{ public_hosted_zone }}"
        tag: openshift_master
        profile: "{{ master_vm }}"
        cloud_init:
          host_name: "master1.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"
      - name: "master2.{{ public_hosted_zone }}"
        tag: openshift_master
        profile: "{{ master_vm }}"
        cloud_init:
          host_name: "master2.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"

      # Infra VMs
      - name: "infra0.{{ public_hosted_zone }}"
        tag: openshift_infra
        profile: "{{ node_vm }}"
        cloud_init:
          host_name: "infra0.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"
      - name: "infra1.{{ public_hosted_zone }}"
        tag: openshift_infra
        profile: "{{ node_vm }}"
        cloud_init:
          host_name: "infra1.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"
      - name: "infra2.{{ public_hosted_zone }}"
        tag: openshift_infra
        profile: "{{ node_vm }}"
        cloud_init:
          host_name: "infra2.{{ public_hosted_zone }}"
          authorized_ssh_keys: "{{ root_ssh_key }}"
          custom_script: "{{ cloud_init_script }}"

      # Node VMs
```

```
    - name: "app0.{{ public_hosted_zone }}"
      tag: openshift_node
      profile: "{{ node_vm }}"
      cloud_init:
        host_name: "app0.{{ public_hosted_zone }}"
        authorized_ssh_keys: "{{ root_ssh_key }}"
        custom_script: "{{ cloud_init_script }}"
    - name: "app1.{{ public_hosted_zone }}"
      tag: openshift_node
      profile: "{{ node_vm }}"
      cloud_init:
        host_name: "app1.{{ public_hosted_zone }}"
        authorized_ssh_keys: "{{ root_ssh_key }}"
        custom_script: "{{ cloud_init_script }}"
    - name: "app2.{{ public_hosted_zone }}"
      tag: openshift_node
      profile: "{{ node_vm }}"
      cloud_init:
        host_name: "app2.{{ public_hosted_zone }}"
        authorized_ssh_keys: "{{ root_ssh_key }}"
        custom_script: "{{ cloud_init_script }}"

    # Load balancer
    - name: "lb.{{ public_hosted_zone }}"
      tag: openshift_lb
      profile: "{{ node_vm }}"
      cloud_init:
        host_name: "lb.{{ public_hosted_zone }}"
        authorized_ssh_keys: "{{ root_ssh_key }}"
        custom_script: "{{ cloud_init_script }}"

  affinity_groups:
    - name: masters_ag
      cluster: "{{ rhv_cluster }}"
      vm_enforcing: false
      vm_rule: negative
      vms:
        - "master0.{{ public_hosted_zone }}"
        - "master1.{{ public_hosted_zone }}"
        - "master2.{{ public_hosted_zone }}"
      wait: true
    - name: infra_ag
      cluster: "{{ rhv_cluster }}"
      vm_enforcing: false
      vm_rule: negative
      vms:
        - "infra0.{{ public_hosted_zone }}"
        - "infra1.{{ public_hosted_zone }}"
        - "infra2.{{ public_hosted_zone }}"
      wait: true

    - name: app_ag
      cluster: "{{ rhv_cluster }}"
      vm_enforcing: false
      vm_rule: negative
      vms:
```

```
          - "app0.{{ public_hosted_zone }}"
          - "app1.{{ public_hosted_zone }}"
          - "app2.{{ public_hosted_zone }}"
  pre_tasks:
    - name: Log in to oVirt
      ovirt_auth:
        url: "{{ engine_url }}"
        username: "{{ engine_user }}"
        password: "{{ engine_password }}"
        ca_file: "{{ engine_cafile | default(omit) }}"
        insecure: "{{ engine_insecure | default(true) }}"
      tags:
        - always
  post_tasks:
    - name: Logout from oVirt
      ovirt_auth:
        state: absent
        ovirt_auth: "{{ ovirt_auth }}"
      tags:
        - always
...
```

Run this playbook from the *bastion* host as follows:

```
$ ansible-playbook -e@~vault.yaml ovirt-vm-infra.yaml
```

Upon successful completion of this playbook, ten new virtual machines will be running in Red Hat Virtualization, ready for deployment.

2.8. ADD INSTANCES TO DNS

After deploying the instances, ensure they are properly updated in DNS.

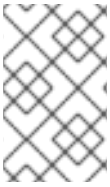
CHAPTER 3. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM

With the prerequisites met, the focus shifts to installation of Red Hat OpenShift Container Platform. A series of Ansible playbooks and roles provided by the **atomic-openshift** packages ensure the remainder of the OpenShift prerequisites and set up the cluster.

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

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

The playbooks run through the complete process of installing Red Hat OpenShift Container Platform. Any errors during the playbook run result in helpful messages specifying what failed and how to retry just that section of the install.



NOTE

Some errors are not recoverable, and will require a complete reinstall of the Red Hat Virtualization instances. See Appendix B, *Reinstall Red Hat OpenShift Container Platform* for a process to streamline re-installation.

3.1. ADDING OPENSHIFT LOGGING (OPTIONAL)

Red Hat OpenShift Container Platform allows the option to deploy aggregate logging for containers running in the OpenShift environment. OpenShift uses the **Elasticsearch**, **Fluentd**, and **Kibana** (EFK) stack to collect logs from applications and present them to OpenShift users. Cluster administrators can view all logs but application developers can only view logs for projects they have permission to view. The EFK stack consists of the following components:

- **Elasticsearch** - Object store for logs with search capability
- **Fluentd** - Unified logging layer to gather logs from OpenShift
- **Kibana** - Web interface to visualize data in Elasticsearch
- **Curator** - Coordinates and schedules Elasticsearch maintenance operations

Below is an example of some of the best practices when deploying OpenShift 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. The **Elasticsearch** cluster size (**openshift_logging_es_cluster_size**) of three is the minimum number of nodes to ensure data durability and redundancy.

NOTE

Fluentd runs on all OpenShift nodes regardless of the node label.

The logging project created during installation must be modified to bypass the default node selector. If this is not done then the **Elasticsearch**, **Kibana**, and **Curator** components cannot be started. There are two options in making the appropriate edit.

Option 1:

ssh into the first master instance (*master1.example.com*) or the *bastion* instance and modify the logging project.

Add the following line **openshift.io/node-selector: ""** under annotations but do not remove any lines.

```
$ oc edit project logging
... [OUTPUT ABBREVIATED] ...
  annotations:
    openshift.io/node-selector: ""
... [OUTPUT ABBREVIATED] ...
project "logging" edited
```

Option 2:

Using **oc patch** command

```
oc patch namespace logging \
    -p "{\"metadata\":{\"annotations\":{\"openshift.io/node-
selector\":\"\"}}}"
```

Log out of the *master1* instance and on the *bastion* instance add the following lines to the **/etc/ansible/hosts** file below the registry and above the [masters] entry.

```
openshift_hosted_logging_deploy=true
openshift_logging_es_pvc_dynamic=true
openshift_logging_es_pvc_size=10Gi
openshift_logging_es_cluster_size=3
openshift_logging_es_nodeselector={"region":"infra"}
openshift_logging_kibana_nodeselector={"region":"infra"}
openshift_logging_curator_nodeselector={"region":"infra"}
```

NOTE

A **StorageClass** must be defined when requesting dynamic storage.

Run the deployment playbooks to deploy logging using the parameters defined in the inventory file.

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-cluster/openshift-logging.yml
```

An example of a successful playbook run is shown below.

```
PLAY RECAP
*********************************************************************
localhost                  : ok=11   changed=0    unreachable=0
failed=0
```

```
master1.example.com : ok=237  changed=94   unreachable=0    failed=0
master2.example.com : ok=16   changed=4    unreachable=0    failed=0
master3.example.com : ok=16   changed=4    unreachable=0    failed=0
Wednesday 28 March 2018  12:34:47 -0400 (0:00:00.415)       0:03:58.679
******
=============================================================================
=====
openshift_facts : Ensure various deps are installed -------------------
46.27s
openshift_facts : Ensure various deps are installed -------------------
38.13s
openshift_logging : Run JKS generation script --------------------------
8.98s
openshift_logging : restart master api ---------------------------------
- 3.98s
openshift_logging : Gather OpenShift Logging Facts ---------------------
3.69s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.60s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.13s
openshift_logging : restart master api ---------------------------------
- 2.90s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.74s
openshift_logging : include_role ---------------------------------------
- 1.60s
openshift_logging_elasticsearch : Set logging-elasticsearch-view-role role
--- 1.51s
openshift_logging_kibana : Set logging-kibana service ------------------
1.46s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.45s
openshift_logging_elasticsearch : Set logging-es service ---------------
1.43s
openshift_logging_elasticsearch : Set logging-es service ---------------
1.41s
openshift_logging_elasticsearch : Set logging-elasticsearch-view-role role
--- 1.40s
openshift_logging_elasticsearch : Create rolebinding-reader role --------
1.40s
openshift_logging_elasticsearch : Set logging-es service ---------------
1.39s
openshift_logging_elasticsearch : Set logging-es-cluster service --------
1.37s
openshift_logging_elasticsearch : Create rolebinding-reader role --------
1.36s
```

Once the playbook finishes **ssh** into the first master instance (*master1.example.com*) or the *bastion* host, and view the pods in the logging project.

```
$ oc get pods -n logging
NAME                                      READY     STATUS    RESTARTS
AGE
logging-curator-1-tzrsx                   1/1       Running   2
4m
```

```
logging-es-data-master-9uq6xi6z-1-dw6vq    1/1       Running   0
5m
logging-es-data-master-mcanh3m7-1-deploy   1/1       Running   0
5m
logging-es-data-master-mcanh3m7-1-vfkt2    1/1       Running   0
4m
logging-es-data-master-qbwcw4j6-1-227gj    1/1       Running   0
4m
logging-es-data-master-qbwcw4j6-1-deploy   1/1       Running   0
4m
logging-fluentd-49hfs                      1/1       Running   0
4m
logging-fluentd-4jwd0                      1/1       Running   0
4m
logging-fluentd-8wtph                      1/1       Running   0
4m
logging-fluentd-bnld6                      1/1       Running   0
4m
logging-fluentd-dp89n                      1/1       Running   0
4m
logging-fluentd-hsgl8                      1/1       Running   0
4m
logging-fluentd-htgx0                      1/1       Running   0
4m
logging-fluentd-s9jp0                      1/1       Running   0
4m
logging-kibana-1-76wxp                     2/2       Running   0
4m
```

NOTE

The curator pod restarts are normal as it tries to connect to the Elasticsearch cluster before it is created.

3.2. ADDING OPENSHIFT METRICS (OPTIONAL)

Red Hat OpenShift Container Platform has the ability to gather metrics from running pods by querying **`kubelet`** and storing the values in **`Heapster`**. Cluster metrics add CPU, memory, and network-based metrics to the OpenShift administrative interface. Additionally, having cluster metrics configured in the cluster enables the creation of Horizontal Pod Autoscalers by the OpenShift user. It is important to understand capacity planning when deploying metrics into an OpenShift environment.

Persistent storage should be employed to prevent the loss of metrics in the event of a pod restart. 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".

Add the following lines to the **`/etc/ansible/hosts`** file in the **`[OSEv3:vars]`** section.

```
openshift_hosted_metrics_deploy=true
openshift_hosted_metrics_storage_kind=dynamic
openshift_hosted_metrics_storage_volume_size=10Gi
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}
```

NOTE

A **`StorageClass`** must be defined when requesting dynamic storage.

Run the deployment playbooks to deploy metrics using the parameters defined in the inventory file.

```
$ ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/byo/openshift-cluster/openshift-metrics.yml
```

An example of a successful playbook run is shown below.

```
PLAY RECAP
*********************************************************************
localhost                  : ok=11   changed=0    unreachable=0
failed=0
master1.example.com : ok=177  changed=47   unreachable=0    failed=0
master2.example.com : ok=16   changed=3    unreachable=0    failed=0
master3.example.com : ok=16   changed=3    unreachable=0    failed=0

Wednesday 30 August 2017  12:49:12 -0400 (0:00:00.296)       0:01:54.693
******
=========================================================================
=====
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.87s
openshift_metrics : restart master api -------------------------------
- 3.33s
openshift_facts : Gather Cluster facts and set is_containerized if needed
--- 3.19s
openshift_facts : Ensure various deps are installed --------------------
3.09s
openshift_facts : Ensure various deps are installed --------------------
2.74s
openshift_metrics : slurp ---------------------------------------------
- 2.56s
openshift_metrics : Set serviceaccounts for hawkular metrics/cassandra ---
2.38s
openshift_metrics : restart master api -------------------------------
- 2.25s
openshift_metrics : Stop Heapster -------------------------------------
- 1.89s
openshift_metrics : Stop Hawkular Metrics -----------------------------
1.60s
openshift_metrics : Start Hawkular Metrics ----------------------------
1.57s
openshift_metrics : Start Hawkular Cassandra --------------------------
1.56s
openshift_metrics : command -------------------------------------------
- 1.53s
openshift_metrics : Start Heapster -------------------------------------
- 1.51s
openshift_metrics : read files for the hawkular-metrics secret ---------
1.46s
openshift_metrics : Generate services for cassandra --------------------
1.24s
```

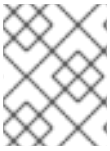
```
openshift_metrics : Generating serviceaccounts for hawkular
metrics/cassandra --- 1.19s
openshift_metrics : Set hawkular cluster roles -------------------------
1.14s
openshift_metrics : generate hawkular-cassandra keys -------------------
1.02s
Gathering Facts --------------------------------------------------------
-- 0.83s
```

Once the playbook finishes **ssh** into the first master instance (*master1.example.com*) or the *bastion* and view the pods in the **openshift-infra** project.

```
$ oc get pods -n openshift-infra
NAME                         READY     STATUS    RESTARTS   AGE
hawkular-cassandra-1-4q46f   1/1       Running   0          3m
hawkular-metrics-0w46z       1/1       Running   0          3m
heapster-gnhsh               1/1       Running   0          3m
```

3.3. 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.

3.3.1. 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
```

3.3.2. 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

3.3.3. Adding Red Hat Virtualization 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 Virtualization. The link below contains the steps for adding Red Hat Virtualization to Cloudforms.

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

CHAPTER 4. OPERATIONAL MANAGEMENT

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

NOTE

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

4.1. OC ADM DIAGNOSTICS

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

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

4.2. USING THE DIAGNOSTICS TOOL

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

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

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

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

```
$ sudo oc adm diagnostics
```

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

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

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

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

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

The standard locations are listed below:

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

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

Available diagnostics include:

Diagnostic Name	Purpose
`AggregatedLogging`	Check the aggregated logging integration for proper configuration and operation.
`AnalyzeLogs`	Check systemd service logs for problems. Does not require a configuration file to check against.
`ClusterRegistry`	Check that the cluster has a working Docker registry for builds and image streams.
`ClusterRoleBindings`	Check that the default cluster role bindings are present and contain the expected subjects according to base policy.
`ClusterRoles`	Check that cluster roles are present and contain the expected permissions according to base policy.
`ClusterRouter`	Check for a working default router in the cluster.
`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 **sudo oc adm diagnostics** works without any flags. In the event, the standard location of the configuration files is not used, options flags as those listed in the example below may be used.

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

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

4.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT

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

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

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

4.5. ANSIBLE-BASED HEALTH CHECKS

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

These checks can be run either using the **ansible-playbook** command (the same method used during Advanced Installation) or as a containerized version of **openshift-ansible**. For the **ansible-playbook** method, the checks are provided by the **atomic-openshift-utils** RPM package.

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

Example usage for each method are provided in subsequent sections.

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

WARNING

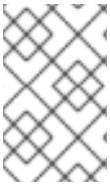
Due to potential changes the health check playbooks could make to the environment, the playbooks should only be run against clusters that have been deployed using Ansible with the same inventory file used during deployment. The changes consist of installing dependencies in order to gather required information.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

Table 4.1. Diagnostic Health Checks

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

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

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



NOTE

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

4.5.1. Running Health Checks via ansible-playbook

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

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

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

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

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

```
openshift_disable_check=etcd_traffic,etcd_volume
```

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

4.6. RUNNING HEALTH CHECKS VIA DOCKER CLI

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

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

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

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

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

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

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

CHAPTER 5. CONCLUSION

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

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

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

For any questions or concerns, please email refarch-feedback@redhat.com and ensure to visit the Red Hat Reference Architecture page to find about all of our Red Hat solution offerings.

APPENDIX A. CONTRIBUTORS

Chris Callegari, content provider

Ryan Cook, content provider

Roger Lopez, content provider

Eduardo Minguez, content provider

Davis Phillips, content provider

Chandler Wilkerson, content provider

APPENDIX B. REINSTALL RED HAT OPENSHIFT CONTAINER PLATFORM

In case of a misconfiguration that puts the OpenShift installation into an unrecoverable state, it is easiest to delete all environment instances and start over. Before deleting the instances, it is wise to first recover the subscriptions by unregistering the nodes. This may be performed with an Ansible ad-hoc command:

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

To aid in VM deletion, copy the list of virtual machines **vms:** from the infrastructure deployment playbook, **ovirt-vm-infra.yaml**. An alternative playbook may be created from the **vms** list using the following syntax:

```
$ cat ovirt-vm-uninstall.yaml
---
- name Remove VMs
  hosts: localhost
  vars:
    vms:
... [ VM list here ] ...
  tasks:
    - name: Set VMs absent
      ovirt_vms:
        auth: "{{ ovirt_auth }}"
        name: "{{ item }}"
        state: absent
      with_items:
        - "{{ vms }}"
  pre_tasks:
    - name: Log in to oVirt
      ovirt_auth:
        url: "{{ engine_url }}"
        username: "{{ engine_user }}"
        password: "{{ engine_password }}"
        ca_file: "{{ engine_cafile | default(omit) }}"
        insecure: "{{ engine_insecure | default(true) }}"
      tags:
        - always
  post_tasks:
    - name: Logout from oVirt
      ovirt_auth:
        state: absent
        ovirt_auth: "{{ ovirt_auth }}"
      tags:
        - always
```

Call this playbook in the same manner as the infrastructure creation playbook:

```
$ ansible-playbook -e@~vault.yaml playbooks/ovirt-vm-uninstall.yaml
```

APPENDIX C. LINKS

- Product Documentation for Red Hat Virtualization 4.2
- Product Documentation for OpenShift Container Platform 3.9
- Best Practices for Red Hat Virtualization 4