Reference Architectures 2018

Deploying and Managing OpenShift 3.9 on Google Cloud Platform

Last Updated: 2019-01-16

Reference Architectures 2018 Deploying and Managing OpenShift 3.9 on Google Cloud Platform

Eduardo Minguez
refarch-feedback@redhat.com

Legal Notice

Abstract

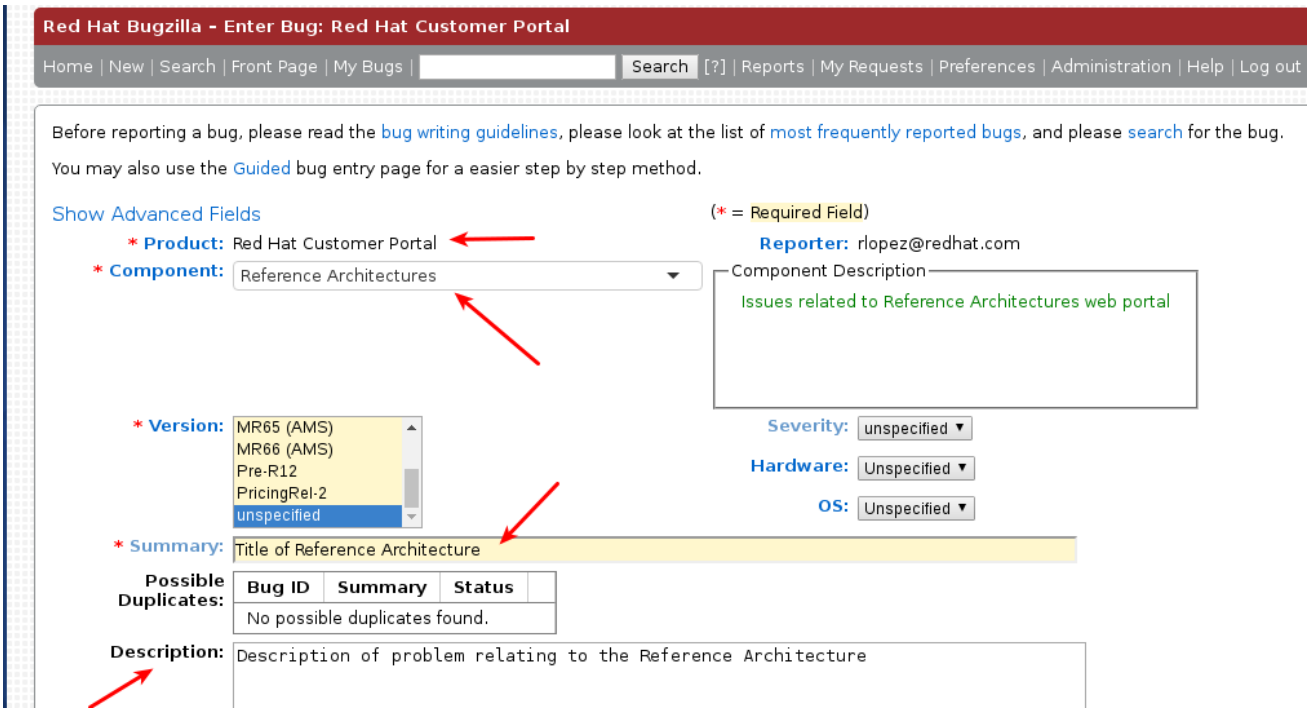
The purpose of this document is to provide guidelines and considerations for deploying and managing Red Hat OpenShift Container Platform on Google Cloud Platform.

Table of Contents

COMMENTS AND FEEDBACK

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

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



EXECUTIVE SUMMARY

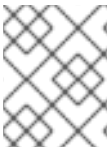
Staying ahead of the needs of an increasingly connected and demanding customer base demands solutions which are not only secure and supported, but robust and scalable, where new features may be delivered in a timely manner. In order to meet these requirements, organizations must provide the capability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

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

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

WHAT IS RED HAT OPENSHIFT CONTAINER PLATFORM

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



NOTE

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

REFERENCE ARCHITECTURE SUMMARY

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

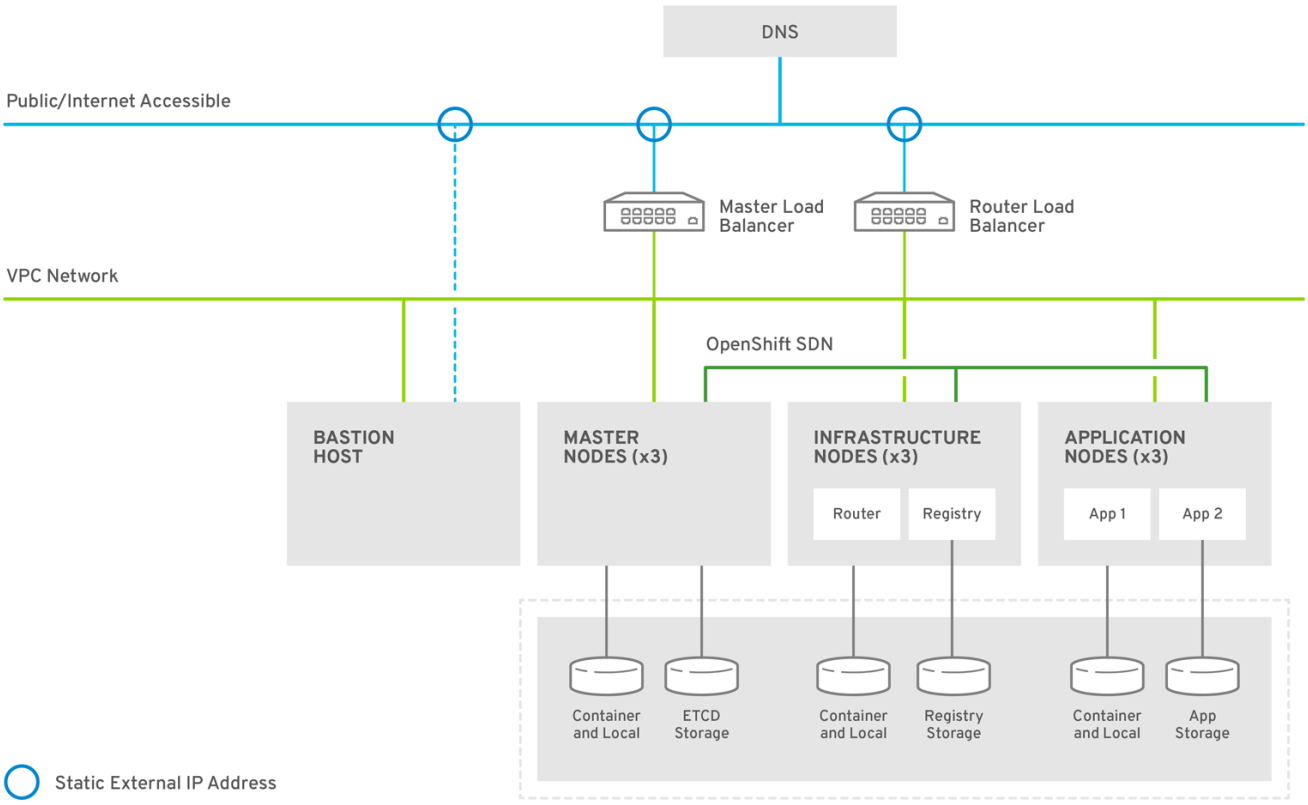
- *Which installation method do you want to use?*
- *How many instances do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other Red Hat technologies?*

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

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

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

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



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

- The Red Hat OpenShift Container Platform instances:
- Bastion instance
- Three master instances
- Three infrastructure instances
- Three application instances
- A dedicated **VPC** Network.
- Masters load balancer to balance API requests and the Red Hat OpenShift Container Platform web console.
- Applications load balancer to balance incoming connections to applications running in Red Hat OpenShift Container Platform.
- Red Hat OpenShift Container Platform routers and registry running in the infrastructure nodes.
- Dedicated storage for the Red Hat OpenShift Container Platform registry.
- Dedicated storage for container images on all nodes.
- Dedicated storage for pods' local storage on all nodes.
- Dedicated storage for **`etcd`** data on all masters.

- Dynamic storage for applications.
- External IP addresses for bastion, masters load balancer and applications load balancer.
- Red Hat OpenShift Container Platform SDN for pod to pod communication.

NOTE

Older versions of this reference architecture contained glue code to deploy Red Hat OpenShift Container Platform instances, load balancers, etc. using the OpenShift-Ansible-Contrib repository not officially supported by Red Hat. This new reference architecture uses Google Cloud Platform command line tools as an example on how to create all the required infrastructure elements in Google Cloud Platform providing more flexibility and customization to deploy the infrastructure required but in a more manual way. Red Hat efforts are focused on having an automated and supported procedure to deploy a proper Red Hat OpenShift Container Platform on different providers.

CHAPTER 1. COMPONENTS AND CONSIDERATIONS

1.1. GOOGLE CLOUD PLATFORM COMPONENTS

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

1.1.1. Project

Google Cloud Platform project is the base level organizing entity. A project is required to use Google Cloud Platform, and forms the basis for creating, enabling and using all Google Cloud Platform services, managing APIs, enabling billing, adding and removing collaborators, and managing permissions.

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

NOTE

For more information, see Google Cloud Platform documentation on **`Project`** https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy#projects

1.1.2. Cloud Identity and Access Management

Access control to the different infrastructure and services resources and fine-grained roles are available in Google Cloud Platform using the **`IAM`** service.

Service accounts with specific permissions can be created and used to deploy infrastructure components instead regular users as well as roles can be created to limit access to different users or service accounts.

Google Cloud Platform instances use service accounts to allow applications running on top to call Google Cloud Platform APIs, for instance, Red Hat OpenShift Container Platform nodes can call Google Cloud Platform disk API to provide a persistent volume to an application.

The deployment of Red Hat OpenShift Container Platform requires a user with proper permissions. The user must be able to create service accounts, cloud storage, instances, images, templates, Cloud DNS entries, and deploy load balancers and health checks. It is helpful to have delete permissions in order to be able to redeploy the environment while testing.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed using a Google Cloud Platform service account.

NOTE

For more information, see Google Cloud Platform documentation on **`IAM`** https://cloud.google.com/compute/docs/access/

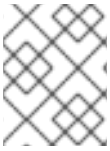
1.1.3. Google Cloud Platform Regions and Zones

Google Cloud Platform has a global infrastructure that covers regions and availability zones. Per Google; "Certain Compute Engine resources live in regions or zones. A region is a specific geographical location

where you can run your resources. Each region has one or more zones. For example, the us-central1 region denotes a region in the Central United States that has zones us-central1-a, us-central1-b, us-central1-c and us-central1-f."

Deploying Red Hat OpenShift Container Platform in Google Cloud Platform on different zones can be beneficial to avoid single-point-of-failures but there are some caveats regarding storage. Google Cloud Platform disks are created within a zone therefore if a Red Hat OpenShift Container Platform node goes down in zone "A" and the pods should be moved to zone "B", the persistent storage cannot be attached to those pods as the disks are in a different zone. See multiple zone limitations kubernetes documentation for more information.

This reference architecture can be deployed in a single region and single zone or multizone, depending on the use case required. Details in the implementation section of this document.



NOTE

For more information, see Google Cloud Platform documentation on regions and zones https://cloud.google.com/compute/docs/zones

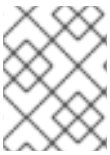
1.1.4. Google Cloud Platform Networking

Google Cloud Platform Networking encapsulates custom virtual networking components including subnets, IP addresses and firewalls.

1.1.4.1. VPC Networks

`VPC network` connects instances and other resources in a project. Projects can have multiple VPC networks and resources within a **`VPC network`** can communicate with other resources in the same **`VPC network`** using private IPv4 addresses (subject to firewall rules).

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed in a single **`VPC network`**.



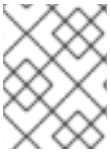
NOTE

For more information, see Google Cloud Platform documentation on **`VPC network`** https://cloud.google.com/vpc/docs/vpc

1.1.4.1.1. Subnets

Google Cloud Platform **`subnets`** are **`VPC Network`** partitions. **`VPC Network`** are global objects where **`subnets`** are regional.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform environment is deployed in a custom **`subnet`**.



NOTE

For more information, see Google Cloud Platform documentation on **`subnets`** https://cloud.google.com/vpc/docs/vpc#vpc_networks_and_subnets

1.1.4.1.2. Internal DNS

Google Cloud Platform **VPC** networks have an internal DNS service that automatically resolves internal hostnames.

The internal fully qualified domain name (FQDN) for an instance follows the **[HOST_NAME].c.[PROJECT_ID].internal** format.

NOTE

For more information, see Google Cloud Platform documentation on **Internal DNS** https://cloud.google.com/compute/docs/internal-dns

1.1.4.1.3. Firewall Rules

Google Cloud Platform **firewall rules** allow/deny ingress/egress traffic and are applied at virtual networking level. This means they are not attached to instances but the filtering is based on tags or service accounts.

Using the installation methods described in this document, different **firewall rules** are created to ensure only required ports are exposed to required hosts.

NOTE

For more information, see Google Cloud Platform documentation on **firewall rules** https://cloud.google.com/vpc/docs/firewalls

1.1.4.1.4. External IP address

For Google Cloud Platform instances to communicate with Internet an **external IP address** is required to be attached to the instance (another method is to use an outbound proxy). Also to communicate with instances deployed in Google Cloud Platform from outside the **VPC Network**, an **external IP address** is required.

WARNING

Requiring an **External IP address** for internet access is a limitation of the provider. This reference architecture configures firewall rules to avoid incoming external traffic in instances if not needed.

NOTE

For more information, see Google Cloud Platform documentation on **external IP address** https://cloud.google.com/compute/docs/ip-addresses/

1.1.4.2. Cloud DNS

Google Cloud Platform **Cloud DNS** is a DNS service to publish domain names to the global DNS using Google Cloud Platform DNS servers. The Public **Cloud DNS** zone requires a domain name either purchased through Google's "Domains" service or through one of many 3rd party providers. Once the zone is created in **Cloud DNS**, the name servers provided by Google need to be added to the registrar.

Following this implementation, a **Cloud DNS** subdomain is used to configure 3 public dns names:

- Bastion
- Red Hat OpenShift Container Platform API
- Applications wildcard

NOTE

For more information, see Google Cloud Platform documentation on **Cloud DNS** https://cloud.google.com/dns/overview

1.1.4.3. Load balancing

Google Cloud Platform **Load Balancing** service enables distribution of traffic across multiple instances in the Google Cloud Platform cloud.

There are five kinds of Load Balancing with some pros/cons:

- Internal - The forwarding rules implementation for the internal load balancer doesn't fit the way Red Hat OpenShift Container Platform master service works. Red Hat OpenShift Container Platform master services queries the master service IP (hence, the load balancer IP) and if the master service is down in that node the internal load balancer doesn't forward to any other master but itself. It is a "forwarding rule" from an external point of view but a "forward to yourself" from the backend instance point of view.
- Network load balancing - Relies on legacy HTTP Health checks for determining instance health.
- HTTP(S) load balancing - It requires using a custom certificate in the load balancer.
- SSL Proxy load balancing - Not recommended for HTTPS traffic. Limited port support.
- TCP Proxy load balancing - Requires 443/tcp to be used (doesn't work with 8443)

To be able to use https health checks for master nodes (request **/healthz** status), https and tcp proxy load balancing are the only options available.

As the https load balancing requires a custom certificate, this reference architecture uses TCP Proxy load balancing to simplify the process.

WARNING

TCP Proxy load balancing only allows port 443 for https, therefore the Red Hat OpenShift Container Platform API is exposed in the 443 port instead 8443.

Checking routers health properly (request **/healthz** status) is incompatible with HTTP(S) load balancing, SSL Proxy load balancing & TCP Proxy load balancing, so network load balancing is used for load balancing routers.

NOTE

For more information, see Google Cloud Platform documentation on **`Load Balancing`** https://cloud.google.com/compute/docs/load-balancing-and-autoscaling

1.1.5. Google Cloud Platform Compute Engine

Instances, images and disks are the building blocks of the cloud computing IaaS model. Google Cloud Platform encapsulates these concepts in the Google Cloud Platform Compute Engine service.

1.1.5.1. Images

Google Cloud Platform images are preconfigured operating system images to create boot disks for the instances. Google Cloud Platform provides public images maintained by Google or third party vendors and custom images can be build and uploaded to Google Cloud Platform with customization.

Red Hat offers two possibilities to run Red Hat Enterprise Linux on certified cloud providers such as Google. There is an "hourly" option where the fee for the instance includes the Red Hat subscription, first and second level support for Red Hat Enterprise Linux is usually provided by the cloud provider. Those images are assembled by Google and need an additional agreement with Google for full technical support. Red Hat provides a third level support and the usual benefits of accessing erratas and documentation. Using such images gives the possibility to burst out into the cloud without the need of managing the fluctuating demand of subscriptions.

The second option is to use existing subscriptions in the Cloud Access program, paying only the bare system to the cloud provider. Specific for Google the image has to be assembled by the customer and uploaded to the Google cloud. Support is directly available by Red Hat under the conditions defined by the subscription. Subscriptions not visible in the Cloud Access enrollment form are not eligible for this program.

Following this implementation, the Cloud Access program is used by creating a Red Hat Enterprise Linux image and uploading it as a Google Cloud Platform custom image.

NOTE

For more information, see Google Cloud Platform documentation on **`Images`** https://cloud.google.com/compute/docs/images and Operating System Details

1.1.5.2. Metadata

Google Cloud Platform **`metadata`** (data or information about other data) is key:value pairs of information used in the Google Cloud Platform instances.

`Metadata` can be assigned at both the project and instance level. Project level **`metadata`** propagates to all virtual machine instances within the project, while instance level **`metadata`** only impacts that instance.

Google Cloud Platform **`metadata`** is used also to store the **`ssh`** keys that are injected at boot time in the instances to allow **`ssh`** access.

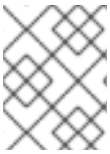
NOTE

For more information, see Google Cloud Platform documentation on **`Metadata`** https://cloud.google.com/compute/docs/storing-retrieving-metadata

1.1.5.3. Instances

`Instances` are virtual machines created from Google Cloud Platform **`images`** running in Google Cloud datacenters in different regions or zones.

Following this implementation, Red Hat Enterprise Linux instances are created to host Red Hat OpenShift Container Platform components.



NOTE

For more information, see Google Cloud Platform documentation on **`Instances`** https://cloud.google.com/compute/docs/instances/

1.1.5.4. Instances sizes

A successful Red Hat OpenShift Container Platform environment requires some minimum hardware requirements. The next table shows the default sizes used in this implementation:

Table 1.1. Instances sizes

Role	Size
Bastion	**`g1-small`**
Master	**`n1-standard-8`**
Infrastructure node	**`n1-standard-8`**
Application node	**`n1-standard-2`**

NOTE

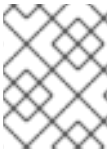
For more information about instance sizes, see https://cloud.google.com/compute/docs/machine-types and Red Hat OpenShift Container Platform Minimum Hardware Requirements

1.1.5.5. Storage Options

By default, each **`instance`** has a small root persistent disk that contains the operating system. When applications running on the instance require more storage space, additional storage options can be added to the instance:

- Standard Persistent Disks
- SSD Persistent Disks
- Local SSDs
- Cloud Storage Buckets

Following this implementation and for performance reasons, "SSD Persistent Disks" are used.



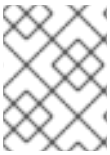
NOTE

For more information, see Google Cloud Platform documentation on **`Storage Options`** https://cloud.google.com/compute/docs/disks/

1.1.6. Cloud Storage

Google Cloud Platform provides object cloud storage used by Red Hat OpenShift Container Platform to store container images using the Red Hat OpenShift Container Platform container registry.

Using the installation methods described in this document, the Red Hat OpenShift Container Platform container registry is deployed using a **`Google Cloud Storage`** (**`GCS`**) bucket.



NOTE

For more information, see Google Cloud Platform documentation on **`Cloud Storage`**https://cloud.google.com/storage/docs/

1.2. 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.

1.3. RED HAT OPENSHIFT CONTAINER PLATFORM COMPONENTS

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

1.3.1. 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.

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

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

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

In this reference architecture a set of three routers are deployed on the *infra* nodes for high availability purposes. In order to have a single entry point for applications, a Google Cloud Platform load balancer is created for load balancing the incoming traffic to the routers running in the *infra* nodes.

Table 1.2. Node Instance types

Type	Node role
Master	Host some components such as the web interface
Infrastructure node	Host Red Hat OpenShift Container Platform infrastructure pods (registry, router, logging & metrics)
Application node	Host applications deployed on Red Hat OpenShift Container Platform

1.3.1.1. Master Instances

The Master instances contains basic Red Hat OpenShift Container Platform components:

Table 1.3. Master Components[1]

Component	Description
API Server	The Kubernetes API server validates and configures the data for pods, services, and replication controllers. It also assigns pods to nodes and synchronizes pod information with service configuration.

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

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

Table 1.4. Availability Matrix[1]

Role	Style	Notes
API Server	Active-Active	Managed by Google Cloud Platform load balancer
Controller Manager Server	Active-Passive	One instance is elected as a cluster lead at a time

1: OpenShift Documentation - Kubernetes Infrastructure

NOTE

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

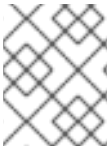
For optimal performance, the **etcd** service should run on the masters instances. When collocating **etcd** with master nodes, at least three instances are required. In order to have a single entry-point for the API, the master nodes should be deployed behind a load balancer.

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 Red Hat OpenShift Container Platform documentation for a detailed explanation on master nodes.

1.3.1.2. 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.

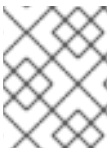
Depending on environment requirements at least three infrastructure nodes are required to provide a sharded/highly available aggregated logging service and to ensure that service interruptions do not occur during a reboot.

NOTE

For more infrastructure considerations, visit the official Red Hat OpenShift Container Platform 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'.

1.3.1.3. 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'}"
```

1.3.2. 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.

1.3.3. 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.

1.3.3.1. 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.

1.3.3.2. 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.

1.4. 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`**.

1.4.1. 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`**.

1.5. 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.

Using this reference architecture, the container storage is configured at instance creation by creating a dedicated **`xfs`** filesystem to a dedicated disk attached to the instance. The prerequisites playbook executes **`container-storage-setup`** to configure the **`overlay2`** storage driver on top of the dedicated **`xfs`** filesystem. endinf::[]

1.6. 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.

1.6.1. 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 Google Cloud Platform or from **glusterfs** if deployed.

1.6.1.1. 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**.

1.7. 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.

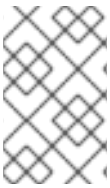
The registry stores images and metadata. For production environment, persistent storage should be used for the registry, otherwise any images that were built or pushed into the registry would disappear if the pod were to restart.

1.8. 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 Red Hat OpenShift Container Platform documentation for more information.

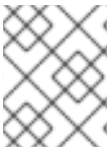
Once deployed in the cluster, Fluentd (deployed as a **DaemonSet** on any node with the right labels) gathers logs from all nodes and containers, enriches the log document with useful metadata (e.g. namespace, container_name, node) and forwards them into Elasticsearch, where Kibana provides a web

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
```

1.9. 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"}
```

1.10. 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.

1.10.1. 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.

1.10.2. 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 1.5. 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

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

CHAPTER 2. RED HAT OPENSHIFT CONTAINER PLATFORM PREREQUISITES

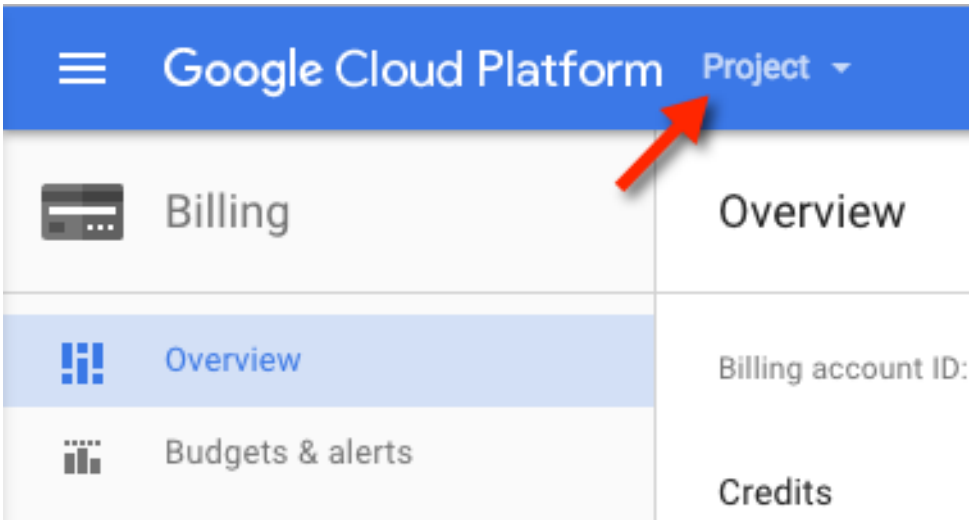
A successful deployment of Red Hat OpenShift Container Platform requires many prerequisites. The prerequisites include the deployment of components in Google Cloud Platform and the required configuration steps prior to the actual installation of Red Hat OpenShift Container Platform using Ansible. In the subsequent sections, details regarding the prerequisites and configuration changes required for an Red Hat OpenShift Container Platform on a Google Cloud Platform environment are discussed in detail.

2.1. GOOGLE CLOUD PLATFORM SETUP

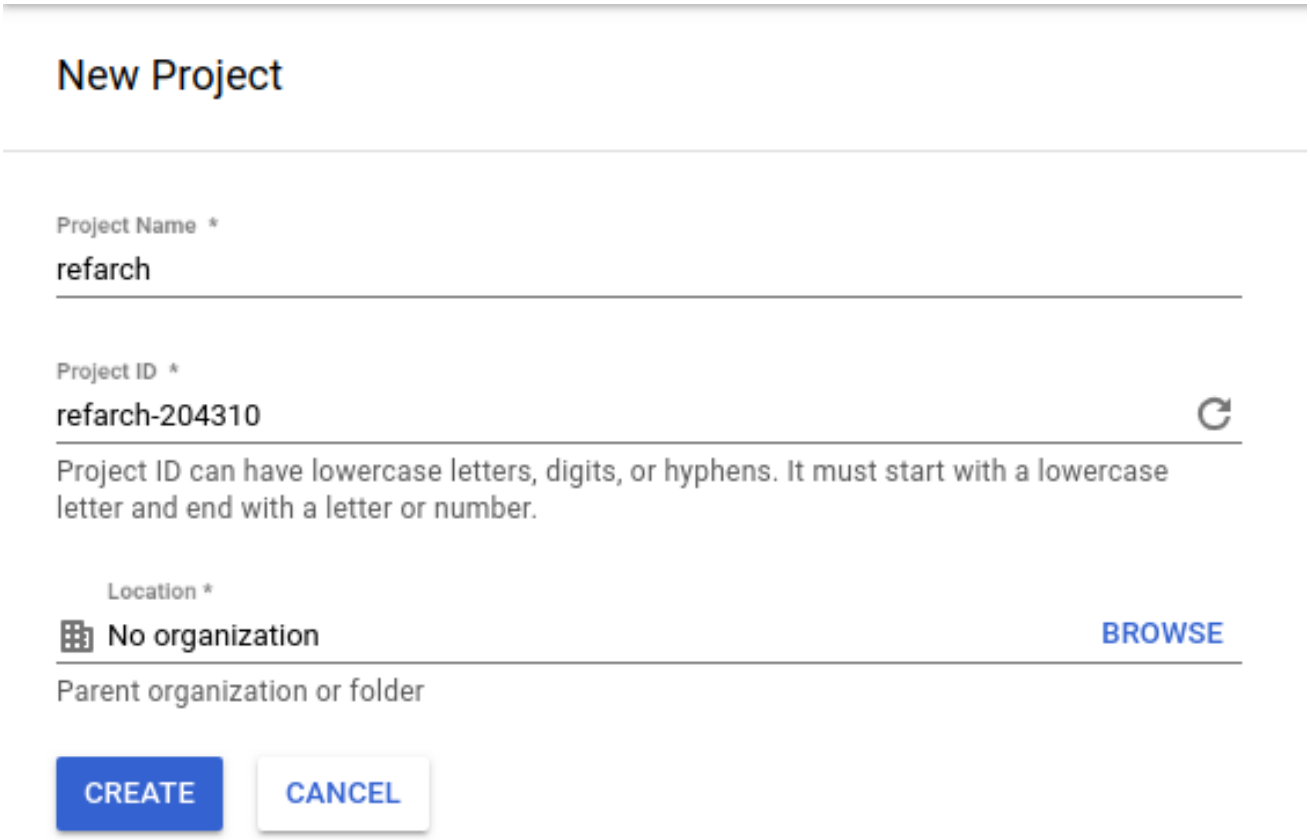
Documentation for the Google Cloud Platform setup can be found on the following link: https://cloud.google.com/compute/docs. To access the Google Cloud Platform Console, a Google account is needed.

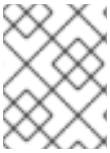
2.1.1. Google Cloud Platform Project

This reference architecture assumes that a "Greenfield" deployment is being done in a newly created Google Cloud Platform project. To set that up, log into the Google Cloud Platform console home and select the Project button:



The name **`refarch`** is used for this reference architecture:



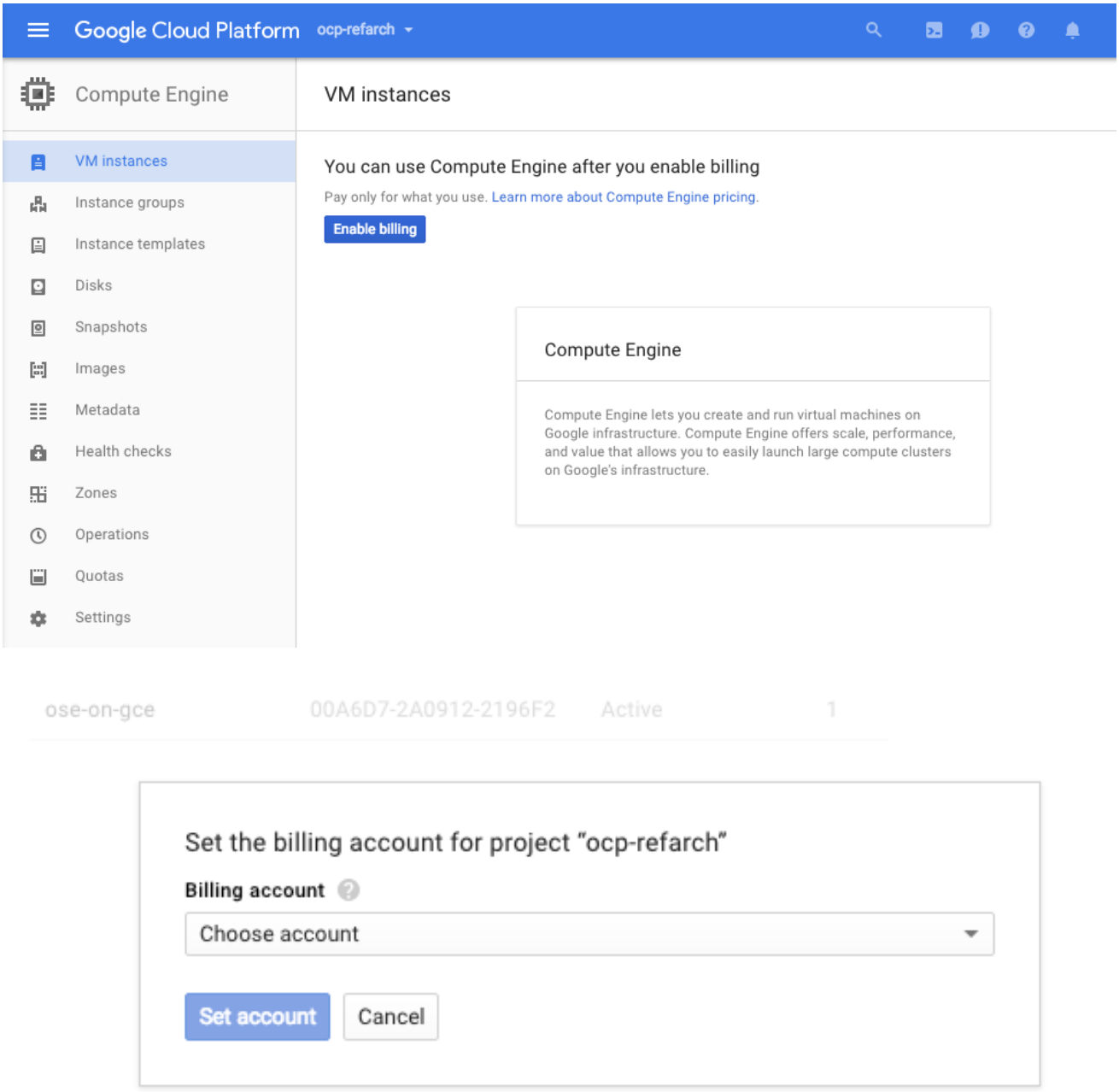


NOTE

It is required the project id to be unique in all the Google Cloud Platform. In this implementation **`refarch-204310`** is used.

2.1.2. Google Cloud Platform Billing

The next step is to set up billing for Google Cloud Platform so new resources can be created. Select "Enable Billing" in the "Billing tab" in the Google Cloud Platform. The new project can be linked to an existing project or financial information can be entered:



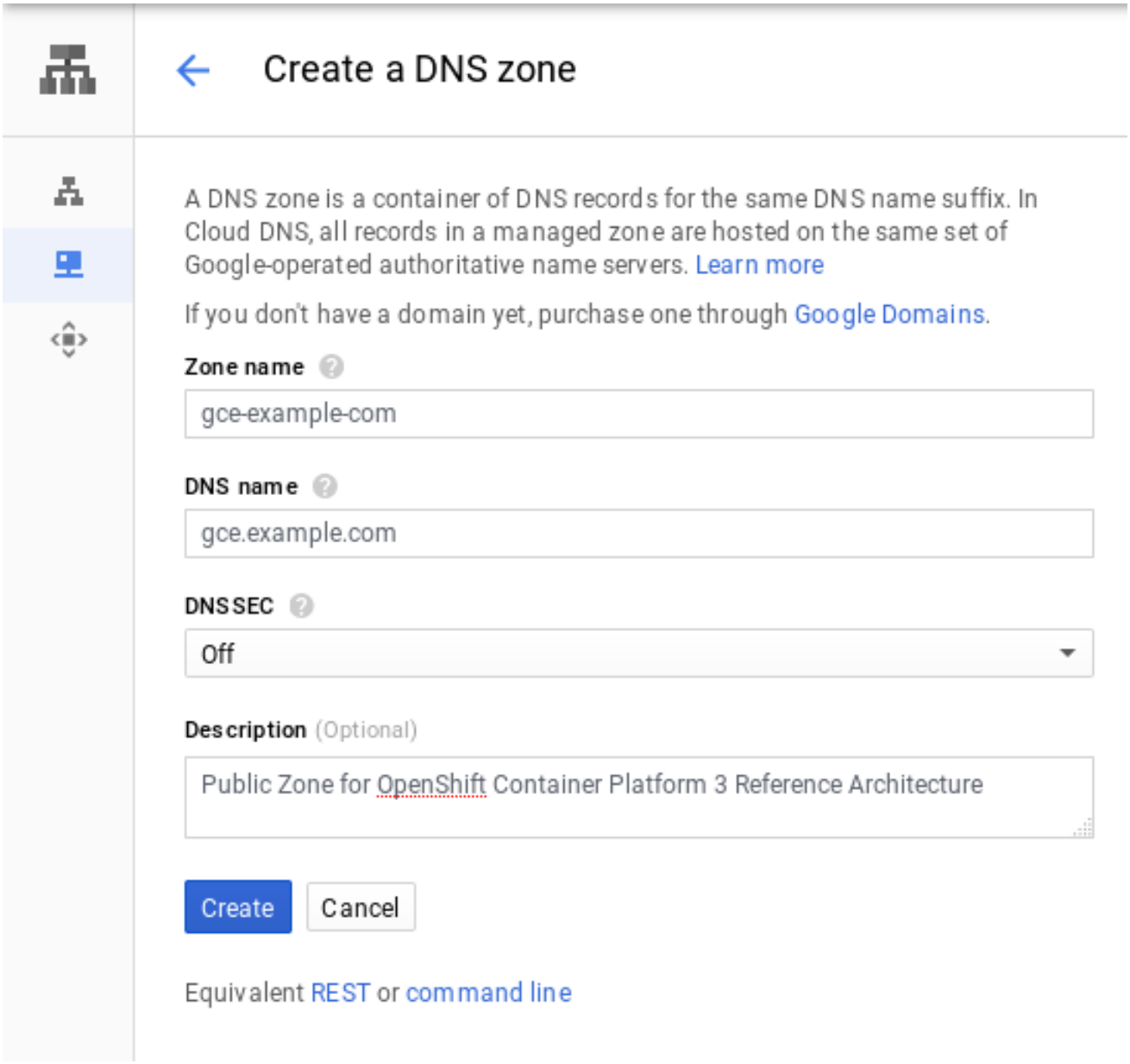
2.1.3. Google Cloud Platform Cloud DNS Zone

In this reference implementation guide, the domain **`example.com`** was purchased through an external provider and the subdomain **`gce.example.com`** is configured to be managed by **`Cloud DNS`**. For the example below, the domain **`gce.example.com`** represents the Public Hosted Domain Name used for the installation of Red Hat OpenShift Container Platform. The following instructions shows how to add the Publicly Hosted Domain Name to Cloud DNS.

- Select the "Network services" → "Cloud DNS" section within the Google Cloud Platform console.
- Select "Create Zone".
- Enter a name logical represent the Publicly Hosted Domain Name as the Cloud DNS Zone in the "Zone Name" box.
    - Zone name must be separated by dashes (-)

- It is recommended to just replace any dots (.) with dashes (-) for easier recognition of the zone (**`gce-example-com`**)

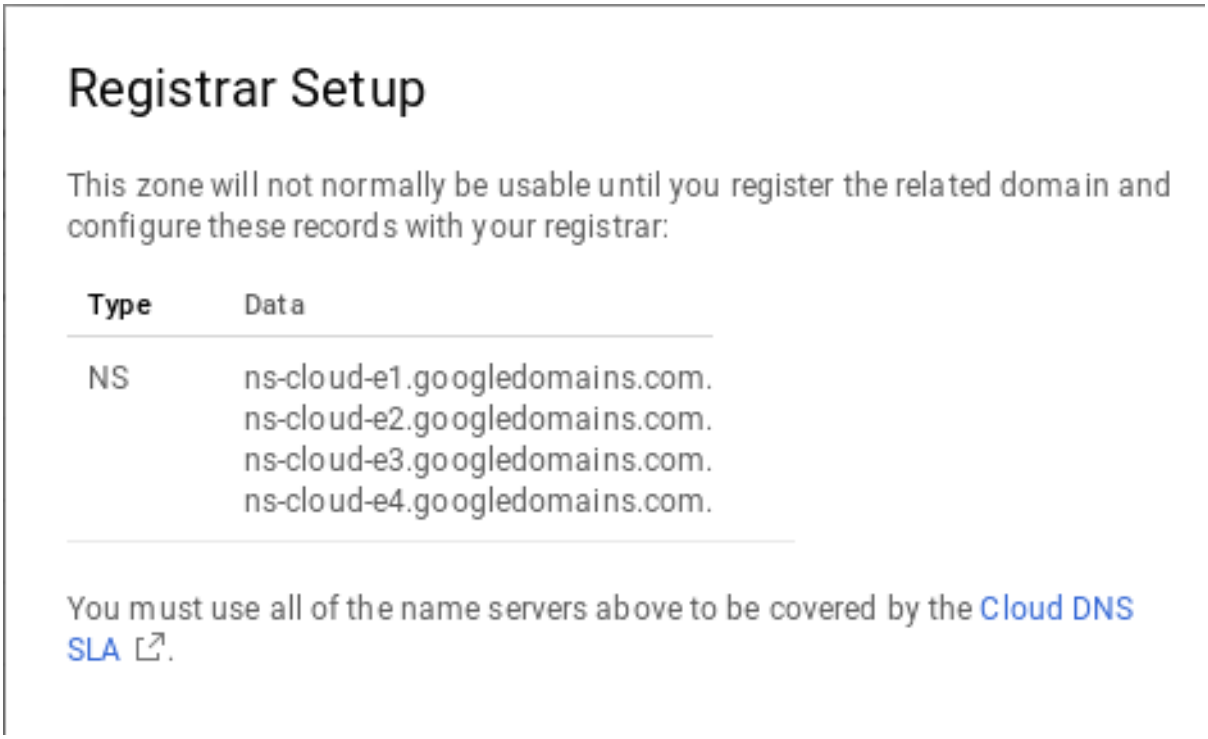
- Enter the Publicly Hosted Domain Name (**`gce.example.com`**) in the "DNS name" box.
- Enter a Description (Public Zone for OpenShift Container Platform 3 Reference Architecture)
- Select "Create"



Once the Pubic Zone is created, delegation to **`gce.example.com`** should be setup to use the Google Name Servers.

NOTE

Refer to the domain registrar provider documentation for instructions on how to make the name server change.



2.1.4. Google Cloud Platform service account

A service account is created to avoid using personal users when deploying Google Cloud Platform objects. This is an optional step and the infrastructure objects can be deployed using a personal account.

NOTE

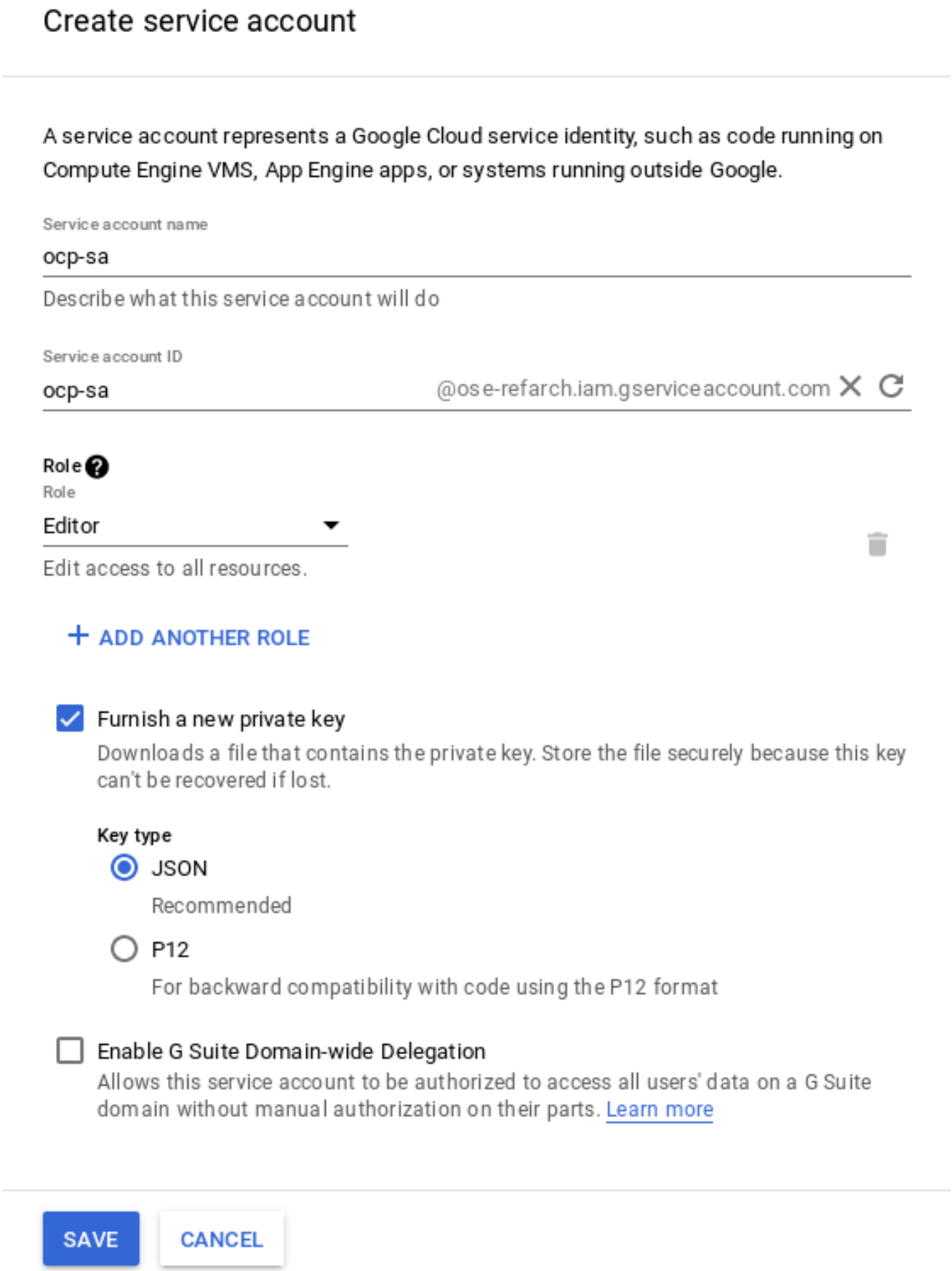
For simplicity, the service account has project editor privileges meaning it can create/delete any object within the project. Use it with caution.

The following instructions shows how to create a service account in the Google Cloud Platform project:

- Select the "IAM" → "Service accounts" section within the Google Cloud Platform console.
- Select "Create Service account".
- Select "Project" → "Editor" as service account Role.
- Select "Furnish a new private key".
- Select "Save"

WARNING

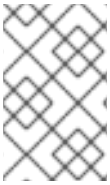
This process generates a json file with a private key that is used to access Google Cloud Platform. It should be store securely to prevent Google Cloud Platform access.



2.1.5. Google Cloud Platform SSH Keys (optional)

Google Cloud Platform injects ssh public keys as authorized keys to be able to login using ssh in the instances created. The list of ssh keys injected in the instance can be configured per instance basis or per project.

The following instructions shows how to create and add a ssh key in the Google Cloud Platform project. The creation step is optional if the **ssh** key exists, but the import should be done.



NOTE

This procedure adds the ssh key to the whole project but this can be overwritten on every instance. See https://cloud.google.com/compute/docs/instances/adding-removing-ssh-keys#block-project-keys for more information.

- From the local workstation, create a ssh keypair:

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

This process generates both private and public key:

```
Your identification has been saved in ~/.ssh/gcp_key.
Your public key has been saved in ~/.ssh/gcp_key.pub.
The key fingerprint is:
SHA256:5gmNKPww1KkUhX8NwPM2OfO+R5ipS1vLJgZacm0FIys myuser@example.com
The key's randomart image is:
+---[RSA 2048]----+
|  .+o.           |
|  .o+.+          |
|  o.o= *         |
| +E.o.Ooo        |
|  *..+o*S+       |
|  .=+ o+=..      |
|   =.o.o+.       |
|  .  .++o..      |
|     .o++o       |
+----[SHA256]-----+
```

- View and copy the public key:

```
ssh-rsa <KEY_VALUE> <user>@<hostname>
```

NOTE

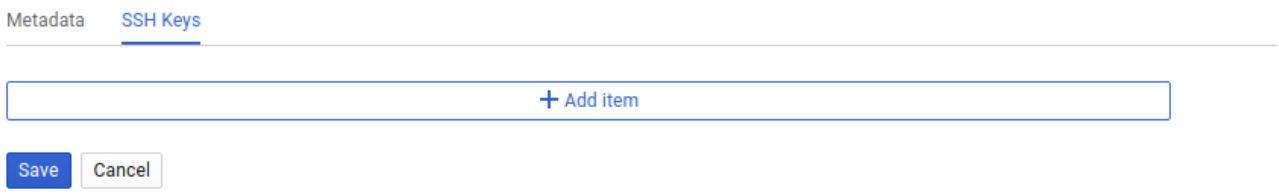
Where the <user>@<hostname> is usually just a comment, when using Google Cloud Platform tools to inject the ssh key, the user is created and it is required to fit the environment.

As an example, the following public key has been tweaked to remove the <hostname>:

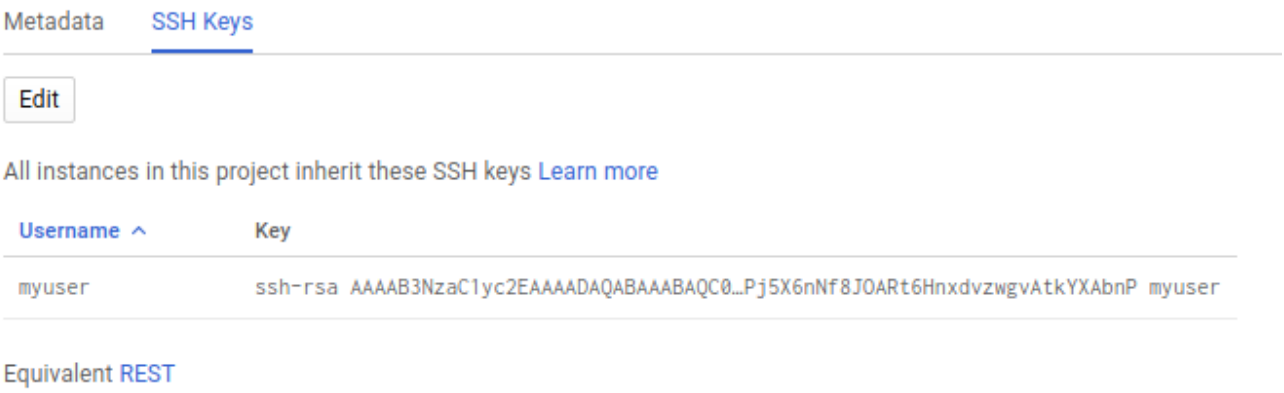
```
$ cat ~/.ssh/gcp_key.pub
ssh-rsa AAAAB3NzaC1y...[OUTPUT OMITTED]...vzwgvAtkYXAbnP myuser
```

- Select the "Compute Engine" → "Metadata" section within the Google Cloud Platform console.

- Select "SSH Keys" tab and select "+ Add item"



- Add the public key content to the box and "Save" it.

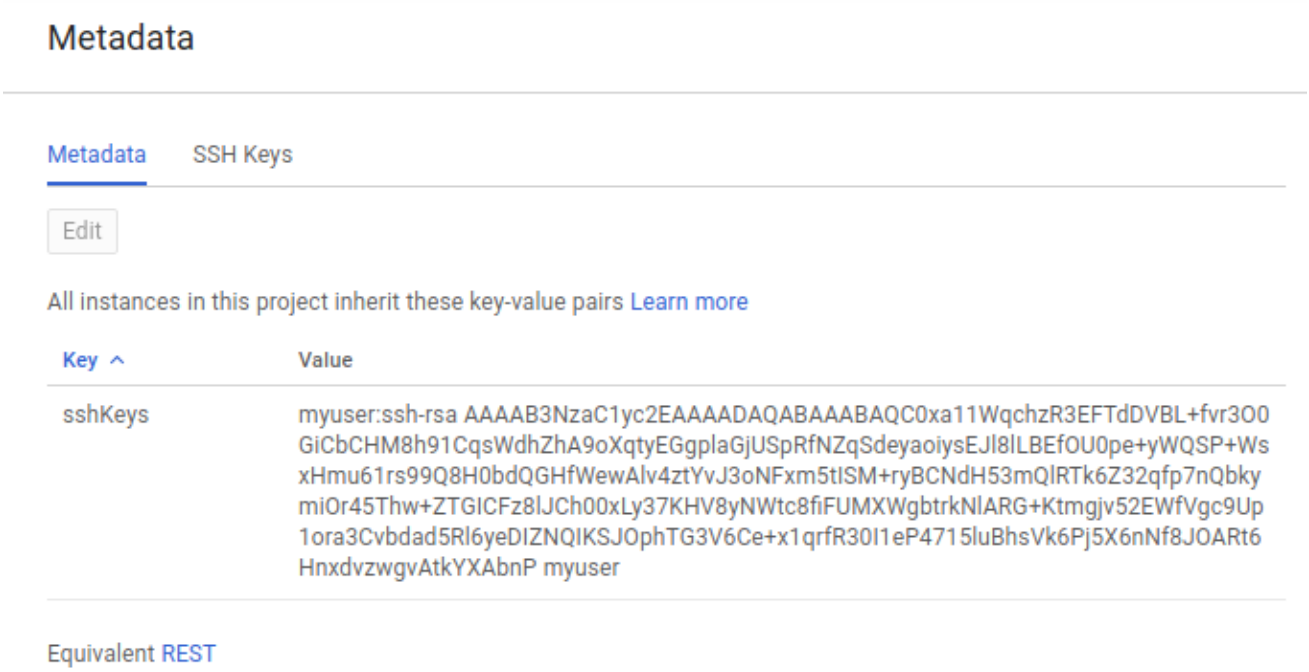


- Select the "Compute Engine" → "Metadata" → "Metadata" section within the Google Cloud Platform console.
- Edit the 'sshKeys' key and append the ssh public key with the following format:

```
<user>:ssh-rsa <KEY_VALUE> <user>
```

As an example:

```
myuser:ssh-rsa AAAAB3NzaC1y...[OUTPUT OMITTED]...vzwgvAtkYXAbnP myuser
```



2.2. GOOGLE CLOUD PLATFORM TOOLING PREPARATION

To manage the Google Cloud Platform resources using the command line, it is required to download and install the Google Cloud SDK with the **gcloud** command-line tool.

Google provides repository which can be used to install Google Cloud SDK. The following instructions shows how to install the Google Cloud SDK on Red Hat Enterprise Linux 7 or Fedora based OS:

- Configure a new repository with the Google Cloud SDK information:

```
$ sudo tee -a /etc/yum.repos.d/google-cloud-sdk.repo << EOF
[google-cloud-sdk]
name=Google Cloud SDK
baseurl=https://packages.cloud.google.com/yum/repos/cloud-sdk-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
       https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

- Install the **google-cloud-sdk** package and dependencies

```
$ sudo yum install google-cloud-sdk
```

2.2.1. Google Cloud Platform command line configuration

The **gcloud** command requires some configuration to access Google Cloud Platform project and resources. As a regular user execute **gcloud init** to configure it using the user/password used to access the Google Cloud Platform console:

```
$ gcloud init
```

Optionally disable usage reporting:

```
$ gcloud config set disable_usage_reporting true
```

To verify it is working:

```
$ gcloud info
Google Cloud SDK [200.0.0]
...[OUTPUT OMITTED]...
Account: [XXX]
Project: [refarch-204310]

Current Properties:
  [core]
    project: [refarch-204310]
    account: [XXX]
    disable_usage_reporting: [True]
  [compute]
    region: [us-west1]
    zone: [us-west1-a]
...[OUTPUT OMITTED]...
```

2.2.2. Google Cloud Platform command line service account configuration

The service account previously created should be used to deploy the objects. The json file downloaded as part of the service account creation is used.

- Verify the content:

```
$ cat ~/Downloads/my-key-file.json
{
  "type": "service_account",
  "project_id": "refarch-204310",
  "private_key_id": "xxxx",
  "private_key": "....",
  ...[OUTPUT OMITTED]...
```

- Use the key file to activate the service account:

```
$ gcloud auth activate-service-account --key-file=~/Downloads/my-key-
file.json
```

- Verify it is used with **`gcloud`** cli:

```
$ gcloud auth list
                 Credentialed Accounts
ACTIVE  ACCOUNT
        my-regular-user@example.com
*       ocp-sa@refarch-204310.iam.gserviceaccount.com
```

2.2.2.1. Google Cloud Platform gsutil command line service account configuration

gsutil command is used to handle object storage. By default **gsutil** should use the service account access configured to **gcloud**.

If the command fails due to bad permissions, verify the service account has proper permissions to handle storage ('storage.bucket.*').

As a test, a sample bucket can be created/deleted:

```
$ BUCKETNAME=$(cat /dev/urandom | tr -dc 'a-z' | fold -w 32 | head -n 1)
$ gsutil mb gs://${BUCKETNAME}
$ gsutil rb gs://${BUCKETNAME}
```

If after verifying it has the proper permissions **gsutil** still complains about permissions, configure **gsutil** to use the HMAC access key and secret.

- Verify the content of the key file:

```
$ cat ~/Downloads/my-key-file.json
{
  "type": "service_account",
  "project_id": "refarch-204310",
  "private_key_id": "xxxx",
  "private_key": "....",
  ...[OUTPUT OMITTED]...
```

- Configure gsutil to use the **private_key_id** and **private_key**:

```
$ gsutil config -a
  ...[OUTPUT OMITTED]...
  What is your google access key ID? xxxx
  What is your google secret access key? "-----BEGIN PRIVATE KEY-----\n...
[OUTPUT OMITTED]...\n-----END PRIVATE KEY-----\n"
```

2.3. ENVIRONMENT CONFIGURATION

To simplify the Google Cloud Platform components creation, the following variables are exported to environment variables to be reused during the commands execution. These variables are examples that need to be modified to fit the particular environment.

Environment variables

```
# Google Project ID
export PROJECTID="refarch-204310"
# Google Region
export REGION="us-west1"
export DEFAULTZONE="us-west1-a"
# For multizone deployments
ZONES=("us-west1-a" "us-west1-b" "us-west1-c")
# For single zone deployments
# ZONES=("us-west1-a")
export MYZONES_LIST="$(declare -p ZONES)"
# OpenShift Cluster ID
export CLUSTERID="refarch"
# Network and subnet configuration
```

```
export CLUSTERID_NETWORK="${CLUSTERID}-net"
export CLUSTERID_SUBNET="${CLUSTERID}-subnet"
# Subnet CIDR, modify if needed
export CLUSTERID_SUBNET_CIDR="10.240.0.0/24"
# DNS
export DNSZONE="example-com"
export DOMAIN="gce.example.com."
export TTL=3600
# RHEL image to be used
export RHELIMAGE="${CLUSTERID}-rhel-image"
export IMAGEPROJECT="${PROJECTID}"
# Bastion settings
export BASTIONDISKSIZE="20GB"
export BASTIONSIZE="g1-small"
# Master nodes settings
export MASTER_NODE_COUNT=3
export MASTERDISKSIZE="40GB"
export MASTERSIZE="n1-standard-8"
export ETCDSIZE="50GB"
export MASTERCONTAINERSSIZE="20GB"
export MASTERLOCALSIZE="30GB"
# Infra nodes settings
export INFRA_NODE_COUNT=3
export INFRADISKSIZE="40GB"
# By default, 8Gi RAM is required to run elasticsearch pods
# as part of the aggregated logging component
export INFRASIZE="n1-standard-8"
export INFRACONTAINERSSIZE="20GB"
export INFRALOCALSIZE="30GB"
# App nodes settings
export APP_NODE_COUNT=3
export APPDISKSIZE="40GB"
export APPSIZE="n1-standard-2"
export APPCONTAINERSSIZE="20GB"
export APPLOCALSIZE="30GB"
# CNS nodes settings
export CNS_NODE_COUNT=3
export CNSDISKSIZE="40GB"
# By default, 8Gi RAM is required to run CNS nodes
export CNSSIZE="n1-standard-8"
export CNSDISKSIZE="40GB"
export CNSCONTAINERSSIZE="20GB"
export CNSGLUSTERSIZE="100GB"
```

Instead exporting every variable and to simplify the deployment, a file can be created with the previous content, then use **`source`** to export them in a single command:

Environment variables source script

```
$ cat ~/myvars
# Google Project ID
export PROJECTID="refarch-204310"
# Google Region
export REGION="us-west1"
export DEFAULTZONE="us-west1-a"
# For multizone deployments
```

```
ZONES=("us-west1-a" "us-west1-b" "us-west1-c")
# For single zone deployments
# ZONES=("us-west1-a")
export MYZONES_LIST="$(declare -p ZONES)"
# OpenShift Cluster ID
export CLUSTERID="refarch"
# Network and subnet configuration
export CLUSTERID_NETWORK="${CLUSTERID}-net"
export CLUSTERID_SUBNET="${CLUSTERID}-subnet"
# Subnet CIDR, modify if needed
export CLUSTERID_SUBNET_CIDR="10.240.0.0/24"
# DNS
export DNSZONE="example-com"
export DOMAIN="gce.example.com."
export TTL=3600
# RHEL image to be used
export RHELIMAGE="${CLUSTERID}-rhel-image"
export IMAGEPROJECT="${PROJECTID}"
# Bastion settings
export BASTIONDISKSIZE="20GB"
export BASTIONSIZE="g1-small"
# Master nodes settings
export MASTER_NODE_COUNT=3
export MASTERDISKSIZE="40GB"
export MASTERSIZE="n1-standard-8"
export ETCDSIZE="50GB"
export MASTERCONTAINERSSIZE="20GB"
export MASTERLOCALSIZE="30GB"
# Infra nodes settings
export INFRA_NODE_COUNT=3
export INFRADISKSIZE="40GB"
# By default, 8Gi RAM is required to run elasticsearch pods
# as part of the aggregated logging component
export INFRASIZE="n1-standard-8"
export INFRACONTAINERSSIZE="20GB"
export INFRALOCALSIZE="30GB"
# App nodes settings
export APP_NODE_COUNT=3
export APPDISKSIZE="40GB"
export APPSIZE="n1-standard-2"
export APPCONTAINERSSIZE="20GB"
export APPLOCALSIZE="30GB"
# CNS nodes settings
export CNS_NODE_COUNT=3
export CNSDISKSIZE="40GB"
# By default, 8Gi RAM is required to run CNS nodes
export CNSSIZE="n1-standard-8"
export CNSDISKSIZE="40GB"
export CNSCONTAINERSSIZE="20GB"
export CNSGLUSTERSIZE="100GB"
$ source ~/myvars
```

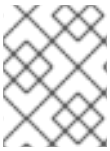
Configure the default project, region and zone.

Default project, region and zone

```
$ gcloud config set project ${PROJECTID}
$ gcloud config set compute/region ${REGION}
$ gcloud config set compute/zone ${DEFAULTZONE}
```

2.4. CREATING A RED HAT ENTERPRISE LINUX BASE IMAGE

The Red Hat Enterprise Linux image from Google Cloud Platform can be used for the deployment of Red Hat OpenShift Container Platform but the instances deployed using this image are charged more as the image carries it's own Red Hat Enterprise Linux subscription. To avoid double paying, the following process can be used to upload an image of Red Hat Enterprise Linux. For this particular reference environment the image used is Red Hat Enterprise Linux 7.5.



NOTE

For more information on how to create custom Google Cloud Platform images, see https://cloud.google.com/compute/docs/images#custom_images

The following instructions shows how to create and add a Red Hat Enterprise Linux image in a Google Cloud Platform project with the proper Google Cloud Platform tools included:

- Visit the Red Hat Cloud Access site https://www.redhat.com/en/technologies/cloud-computing/cloud-access
- Review the terms and enroll in the program
- After following the steps a certificate is generated with the number of subscriptions that can be used in the cloud provider

Image Import was successfully created.

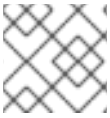
Image Registration Confirmation

You have successfully registered your image for import.
You may now move your image to your selected cloud provider.
Please access the provider's website for instructions on using their import tools.

Redhat Login	
Email Address	
Name	
Company Name	
Cloud Provider	Google Compute Engine
Cloud Provider Account Number/s	
Product	
Quantity	

- Download the latest Red Hat Enterprise Linux 7.5 KVM image from the Red Hat Customer Portal https://access.redhat.com/downloads/content/69/ver=/rhel---7/latest/x86_64/product-software in the Red Hat Enterprise Linux or Fedora workstation where the **gcloud** tool is configured.
- Install **qemu-img** package and convert the qcow2 format image to raw (as required per Google Cloud Platform)

```
$ sudo yum install qemu-img

$ qemu-img convert -p -S 4096 -f qcow2 -O raw rhel-server-7.5-x86_64-
kvm.qcow2 disk.raw
```



NOTE

The raw disk name should be **disk.raw** and the raw file size can be up to 10GB.

- Create a **tar.gz** file with the raw disk

```
$ tar -Szcf rhel-7.5.tar.gz disk.raw
```

- Create a bucket to host the temporary image with the proper labels

```
$ gsutil mb -l ${REGION} gs://${CLUSTERID}-rhel-image-temp

$ cat <<EOF > labels.json
{
  "ocp-cluster": "${CLUSTERID}"
}
EOF

$ gsutil label set labels.json gs://${CLUSTERID}-rhel-image-temp

$ rm -f labels.json
```

NOTE

Google Cloud Platform buckets are global objects. This means that every bucket name must be unique. If the bucket creation process fails because the name is used, select a different bucket name.

- Upload the file

```
$ gsutil -o GSUtil:parallel_composite_upload_threshold=150M \
    cp rhel-7.5.tar.gz gs://${CLUSTERID}-rhel-image-temp
```

- Create a Google Cloud Platform image with the temporary image

```
$ gcloud compute images create ${CLUSTERID}-rhel-temp-image \
  --family=rhel \
  --source-uri=https://storage.googleapis.com/${CLUSTERID}-rhel-image-
temp/rhel-7.5.tar.gz
```

- Create a temporary firewall rule to allow ssh access to the temporary instance:

```
$ gcloud compute firewall-rules create ${CLUSTERID}-ssh-temp \
  --direction=INGRESS --priority=1000 --network=default \
  --action=ALLOW --rules=tcp:22,icmp \
  --source-ranges=0.0.0.0/0 --target-tags=${CLUSTERID}-temp
```

- Create a instance with the temporary image to be customized for Google Cloud Platform integration

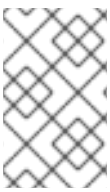
```
$ gcloud compute instances create ${CLUSTERID}-temp \
  --zone=${DEFAULTZONE} \
  --machine-type=g1-small \
  --network=default --async \
  --maintenance-policy=MIGRATE \
  --
scopes=https://www.googleapis.com/auth/devstorage.read_only,https://www.go
ogleapis.com/auth/logging.write,https://www.googleapis.com/auth/monitoring
.write,https://www.googleapis.com/auth/servicecontrol,https://www.googleap
is.com/auth/service.management.readonly,https://www.googleapis.com/auth/tr
ace.append \
  --min-cpu-platform=Automatic \
  --image=${CLUSTERID}-rhel-temp-image \
  --tags=${CLUSTERID}-temp \
  --boot-disk-size=10GB --boot-disk-type=pd-standard \
  --boot-disk-device-name=${CLUSTERID}-temp
```

- Verify the instance is working properly

```
$ gcloud compute instances get-serial-port-output ${CLUSTERID}-temp

$ gcloud compute instances list
```

- Connect to the instance using ssh and the key previously created. The IP address is shown in the previous **`gcloud compute instances list`** command.

```
$ ssh -i ~/.ssh/gcp_key cloud-user@instance-ip
```



NOTE

As the Google Cloud Platform tools are not installed (but the ssh public key is injected using **`cloud-init`**), the user is not created and it uses **`cloud-user`** by default even if the key was created with a different username.

- Subscribe the instance temporary, attach it to the proper pool and configure basic repositories:

```
$ sudo subscription-manager register --username=myuser --password=mypass

$ sudo subscription-manager list --available

$ sudo subscription-manager attach --pool=mypool
```

```
$ sudo subscription-manager repos --disable="*" \
    --enable=rhel-7-server-rpms
```

- Add the Google Cloud Platform repo:

```
$ sudo tee /etc/yum.repos.d/google-cloud.repo << EOF
[google-cloud-compute]
name=Google Cloud Compute
baseurl=https://packages.cloud.google.com/yum/repos/google-cloud-compute-
el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
       https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

- Remove unneeded packages and install the required ones

```
$ sudo yum remove irqbalance cloud-init rhevm-guest-agent-common \
    kexec-tools microcode_ctl rpcbind

$ sudo yum install google-compute-engine python-google-compute-engine \
    rng-tools acpid firewalld
```

NOTE

`google-compute-engine` tools include some tools that provide similar functionality as **`cloud-init`** and they are recommended in Google Cloud Platform instances. For more information see https://github.com/GoogleCloudPlatform/compute-image-packages

- Optionally, update all the packages to the latest versions

```
$ sudo yum update
```

- Enable required serivces

```
$ sudo systemctl enable rngd \
    google-accounts-daemon \
    google-clock-skew-daemon \
    google-shutdown-scripts \
    google-network-daemon
```

- Clean up the image:

```
$ sudo tee /etc/sysconfig/network-scripts/ifcfg-eth0 << EOF
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
MTU=1460
EOF
```

```
$ sudo yum clean all

$ sudo rm -rf /var/cache/yum

$ sudo logrotate -f /etc/logrotate.conf

$ sudo rm -f /var/log/*-???????? /var/log/*.gz

$ sudo rm -f /var/log/dmesg.old /var/log/anaconda

$ cat /dev/null | sudo tee /var/log/audit/audit.log

$ cat /dev/null | sudo tee /var/log/wtmp

$ cat /dev/null | sudo tee /var/log/lastlog

$ cat /dev/null | sudo tee /var/log/grubby

$ sudo rm -f /etc/udev/rules.d/70-persistent-net.rules \
    /etc/udev/rules.d/75-persistent-net-generator.rules \
    /etc/ssh/ssh_host_* /home/cloud-user/.ssh/*

$ sudo subscription-manager remove --all

$ sudo subscription-manager unregister

$ sudo subscription-manager clean

$ export HISTSIZE=0

$ sudo poweroff
```

NOTE

For more information on the steps to make a clean VM for use as a template or clone, see https://access.redhat.com/solutions/198693

- Once the instance has been powered off, the proper image can be created:

```
$ gcloud compute images create ${CLUSTERID}-rhel-image \
  --family=rhel --source-disk=${CLUSTERID}-temp \
  --source-disk-zone=${DEFAULTZONE}
```

After that last step, the image is ready to be used as instances for Red Hat OpenShift Container Platform.

The next steps clean up the temporary image, buckets and firewall rules:

```
$ gcloud compute instances delete ${CLUSTERID}-temp \
  --delete-disks all --zone=${DEFAULTZONE}

$ gcloud compute images delete ${CLUSTERID}-rhel-temp-image

$ gsutil -m rm -r gs://${CLUSTERID}-rhel-image-temp
```

```
$ gcloud compute firewall-rules delete ${CLUSTERID}-ssh-temp
```

Last step is to clean the workstation:

```
$ rm -f disk.raw \
      rhel-server-7.5-x86_64-kvm.qcow2 \
      rhel-7.5.tar.gz
```

2.5. CREATING RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKS

A VPC Network and subnet are created to allow for virtual machines to be launched.

Network and subnet creation

```
# Network
$ gcloud compute networks create ${CLUSTERID_NETWORK} --subnet-mode custom

# Subnet
$ gcloud compute networks subnets create ${CLUSTERID_SUBNET} \
  --network ${CLUSTERID_NETWORK} \
  --range ${CLUSTERID_SUBNET_CIDR}
```

2.6. CREATING FIREWALL RULES

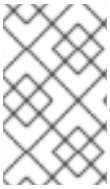
Google Cloud Platform firewall rules allows the user to define inbound and outbound traffic filters that can be applied to tags on a network. This allows the user to limit network traffic to each instance based on the function of the instance services and not depend on host based filtering.

This section describes the ports and services required for each type of host and how to create the security groups on Google Cloud Platform.

The following table shows the tags associated to every instance type:

Table 2.1. Firewall Rules association

Instance type	Tags associated
Bastion	**${CLUSTERID}-bastion**
App nodes	**${CLUSTERID}-node**
Masters	**${CLUSTERID}-master** and **${CLUSTERID}-node**
Infra nodes	**${CLUSTERID}-infra** & **${CLUSTERID}-node**
CNS nodes	**${CLUSTERID}-cns** & **${CLUSTERID}-node**



NOTE

There is an extra tag **`${CLUSTERID}ocp`** used by default when creating "LoadBalancer" services that needs to be applied to application nodes. A bugzilla is being investigated to allow customization of that tag.

2.6.1. Bastion Firewall rules

The *bastion* instance only needs to allow inbound **`ssh`** from the internet and it should be allowed to reach any host in the network as the installation is performed in the bastion instace. Also this instance exists to serve as the jump host between the private subnet and public internet.

Table 2.2. Bastion Allowed firewall rules

Port/Protocol	Service	Remote source	Purpose
22/TCP	SSH	Any	Secure shell login

Creation of the above firewall rules is as follows:

Bastion Firewall rules

```
# External to bastion
$ gcloud compute firewall-rules create ${CLUSTERID}-external-to-bastion \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:22,icmp \
  --source-ranges=0.0.0.0/0 --target-tags=${CLUSTERID}-bastion

# Bastion to all hosts
$ gcloud compute firewall-rules create ${CLUSTERID}-bastion-to-any \
    --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
    --action=ALLOW --rules=all \
    --source-tags=${CLUSTERID}-bastion --target-tags=${CLUSTERID}-node
```

2.6.2. Master Firewall Rules

The Red Hat OpenShift Container Platform master firewall rules requires the most complex network access controls. In addition to the ports used by the API and master console, these nodes contain the **`etcd`** servers that form the cluster.

Table 2.3. Master Host Allowed firewall rules

Port/Protocol	Service	Remote source	Purpose
2379/TCP	etcd	Masters	Client → Server connections
2380/TCP	etcd	Masters	Server → Server cluster communications

Port/Protocol	Service	Remote source	Purpose
8053/TCP	DNS	Masters and nodes	Internal name services (3.2+)
8053/UDP	DNS	Masters and nodes	Internal name services (3.2+)
443/TCP	HTTPS	Any	Master WebUI and API
10250/TCP	kubelet	Nodes	Kubelet communications

NOTE

As masters nodes are in fact nodes, the same rules used for Red Hat OpenShift Container Platform nodes are used.

Creation of the above security group is as follows:

Master Firewall rules

```
# Nodes to master
$ gcloud compute firewall-rules create ${CLUSTERID}-node-to-master \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=udp:8053,tcp:8053 \
  --source-tags=${CLUSTERID}-node --target-tags=${CLUSTERID}-master

# Master to node
$ gcloud compute firewall-rules create ${CLUSTERID}-master-to-node \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:10250 \
  --source-tags=${CLUSTERID}-master --target-tags=${CLUSTERID}-node

# Master to master
$ gcloud compute firewall-rules create ${CLUSTERID}-master-to-master \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:2379,tcp:2380 \
  --source-tags=${CLUSTERID}-master --target-tags=${CLUSTERID}-master

# Any to master
$ gcloud compute firewall-rules create ${CLUSTERID}-any-to-masters \
  --direction=INGRESS --priority=1000  --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:443 \
  --source-ranges=${CLUSTERID_SUBNET_CIDR} --target-tags=${CLUSTERID}-
master
```

2.6.3. Infrastructure Node Firewall Rules

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

Table 2.4. Infrastructure Node Allowed firewall rules

Port/Protocol	Services	Remote source	Purpose
80/TCP	HTTP	Any	Cleartext application web traffic
443/TCP	HTTPS	Any	Encrypted application web traffic
9200/TCP	ElasticSearch	Any	ElasticSearch API
9300/TCP	ElasticSearch	Any	Internal cluster use

Creation of the above security group is as follows:

Infrastructure node Firewall rules

```
# Infra node to infra node
$ gcloud compute firewall-rules create ${CLUSTERID}-infra-to-infra \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:9200,tcp:9300 \
  --source-tags=${CLUSTERID}-infra --target-tags=${CLUSTERID}-infra

# Routers
$ gcloud compute firewall-rules create ${CLUSTERID}-any-to-routers \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --source-ranges 0.0.0.0/0 \
  --target-tags ${CLUSTERID}-infra \
  --allow tcp:443,tcp:80
```

2.6.4. Node Firewall rules

The node firewall rules allows pod to pod communication via SDN traffic.

Table 2.5. Node Allowed firewall rules

Port/Protocol	Services	Remote source	Purpose
4789/UDP	SDN	Nodes	Pod to pod communications
10250/TCP	Kubelet	Infra nodes	Heapster to gather metrics from nodes

Creation of the above security group is as follows:

Node Firewall rules

```
# Node to node SDN
$ gcloud compute firewall-rules create ${CLUSTERID}-node-to-node \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
```

```
  --action=ALLOW --rules=udp:4789 \
  --source-tags=${CLUSTERID}-node --target-tags=${CLUSTERID}-node

# Infra to node kubelet
$ gcloud compute firewall-rules create ${CLUSTERID}-infra-to-node \
    --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
    --action=ALLOW --rules=tcp:10250 \
    --source-tags=${CLUSTERID}-infra --target-tags=${CLUSTERID}-node
```

NOTE

As masters and infrastructure nodes are tagged as nodes the same rules apply to them.

2.6.5. CNS Node Firewall Rules (Optional)

The CNS nodes require specific ports of the **`glusterfs`** pods. The rules defined only allow glusterfs communication.

Table 2.6. CNS Allowed firewall rules

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

Creation of the above security group is as follows:

CNS Node Firewall rules

```
# CNS to CNS node
$ gcloud compute firewall-rules create ${CLUSTERID}-cns-to-cns \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --action=ALLOW --rules=tcp:2222 \
  --source-tags=${CLUSTERID}-cns --target-tags=${CLUSTERID}-cns

# Node to CNS node (client)
$ gcloud compute firewall-rules create ${CLUSTERID}-node-to-cns \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
```

```
  --action=ALLOW \
  --rules=tcp:111,udp:111,tcp:3260,tcp:24007-24010,tcp:49152-49664 \
  --source-tags=${CLUSTERID}-node --target-tags=${CLUSTERID}-cns
```

2.7. CREATING EXTERNAL IP ADDRESSES

Even if all the instances have an external IP to be able to access internet, those IPs are ephemeral. It is required to create some named external IPs to be able to register them with DNS:

- Masters load balancer IP
- Applications load balancer IP
- Bastion host

Creation of the above external IPs is as follows:

External IPs

```
# Masters load balancer
$ gcloud compute addresses create ${CLUSTERID}-master-lb \
    --ip-version=IPV4 \
    --global

# Applications load balancer
$ gcloud compute addresses create ${CLUSTERID}-apps-lb \
    --region ${REGION}

# Bastion host
$ gcloud compute addresses create ${CLUSTERID}-bastion \
  --region ${REGION}
```

2.8. CREATING EXTERNAL DNS RECORDS

`Cloud DNS` service is used to register 3 external entries:

Table 2.7. DNS Records

DNS Record	IP	Example
`${CLUSTERID}-ocp.${DOMAIN}`	Masters load balancer IP	**`refarch-ocp.gce.example.com`**
`*.${CLUSTERID}-apps.${DOMAIN}`	Applications load balancer IP	**`*.refarch-apps.gce.example.com`**
`${CLUSTERID}-bastion.${DOMAIN}`	Bastion host	**`refarch-bastion.gce.example.com`**

DNS Records

```
# Masters load balancer entry
```

```
$ export LBIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-master-lb" --format="value(address)")

$ gcloud dns record-sets transaction start --zone=${DNSZONE}

$ gcloud dns record-sets transaction add \
  ${LBIP} --name=${CLUSTERID}-ocp.${DOMAIN} --ttl=${TTL} --type=A \
  --zone=${DNSZONE}
$ gcloud dns record-sets transaction execute --zone=${DNSZONE}

# Applications load balancer entry
$ export APPSLBIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-apps-lb" --format="value(address)")

$ gcloud dns record-sets transaction start --zone=${DNSZONE}

$ gcloud dns record-sets transaction add \
  ${APPSLBIP} --name=\*.${CLUSTERID}-apps.${DOMAIN} --ttl=${TTL} --type=A
\
  --zone=${DNSZONE}

$ gcloud dns record-sets transaction execute --zone=${DNSZONE}

# Bastion host
$ export BASTIONIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-bastion" --format="value(address)")

$ gcloud dns record-sets transaction start --zone=${DNSZONE}

$ gcloud dns record-sets transaction add \
  ${BASTIONIP} --name=${CLUSTERID}-bastion.${DOMAIN} --ttl=${TTL} --type=A
\
  --zone=${DNSZONE}

$ gcloud dns record-sets transaction execute --zone=${DNSZONE}
```

2.9. CREATING GOOGLE CLOUD PLATFORM INSTANCES

The reference environment in this document consists of the following Google Cloud Platform instances:

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

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

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

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

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

NOTE

Always check resource quotas before the deployment of resources. Support requests can be submitted to request the ability to deploy more types of an instance or to increase the total amount of CPUs.

2.9.1. Instance Storage

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

Each *master* node mounts 3 dedicated disks using the steps provided in this reference architecture:

- A dedicated disk to host **etcd** storage.
- A dedicated disk to store the container container images.
- A dedicated disk to store ephemeral pod storage.

Each *infra* and *app* node Google Cloud Platform instance creates and mounts 2 dedicated disks using the steps provided in this reference architecture:

- A dedicated disk to store the container container images.
- A dedicated disk to store ephemeral pod storage.

Table 2.8. Instances storage

Disk	Role	Mountpoint	Instance type
${CLUSTERID}-master-${i}-etcd	**etcd** storage	**/var/lib/etcd**	Master only
${CLUSTERID}-master-${i}-containers	Container storage	**/var/lib/docker**	Master and nodes
${CLUSTERID}-master-${i}-local	Ephemeral pod storage	**/var/lib/origin/openshift.local.volumes**	Master and nodes

2.9.2. Bastion instance

The *bastion* host serves as the installer of the Ansible playbooks that deploy Red Hat OpenShift Container Platform as well as an entry point for management tasks. The bastion instance is created and attached to the external IP previously created:

Bastion instance

```
$ export BASTIONIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-bastion" --format="value(address)")

$ gcloud compute instances create ${CLUSTERID}-bastion \
  --async --machine-type=${BASTIONSIZE} \
  --subnet=${CLUSTERID_SUBNET} \
  --address=${BASTIONIP} \
  --maintenance-policy=MIGRATE \
  --
scopes=https://www.googleapis.com/auth/cloud.useraccounts.readonly,https:/
/www.googleapis.com/auth/compute,https://www.googleapis.com/auth/devstorag
e.read_write,https://www.googleapis.com/auth/logging.write,https://www.goo
gleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/service.
management.readonly,https://www.googleapis.com/auth/servicecontrol \
  --tags=${CLUSTERID}-bastion \
  --metadata "ocp-cluster=${CLUSTERID},${CLUSTERID}-type=bastion" \
  --image=${RHELIMAGE} --image-project=${IMAGEPROJECT} \
  --boot-disk-size=${BASTIONDISKSIZE} --boot-disk-type=pd-ssd \
  --boot-disk-device-name=${CLUSTERID}-bastion \
  --zone=${DEFAULTZONE}
```

2.9.3. Master Instances and Components

The following script needs to be created in the local workstation to be used in the instance for customization at boot time:

Master customization

```
$ vi master.sh
#!/bin/bash
LOCALVOLDEVICE=$(readlink -f /dev/disk/by-id/google-*local*)
ETCDDEVICE=$(readlink -f /dev/disk/by-id/google-*etcd*)
CONTAINERSDEVICE=$(readlink -f /dev/disk/by-id/google-*containers*)
LOCALDIR="/var/lib/origin/openshift.local.volumes"
ETCDDIR="/var/lib/etcd"
CONTAINERSDIR="/var/lib/docker"

for device in ${LOCALVOLDEVICE} ${ETCDDEVICE} ${CONTAINERSDEVICE}
do
  mkfs.xfs ${device}
done

for dir in ${LOCALDIR} ${ETCDDIR} ${CONTAINERSDIR}
do
  mkdir -p ${dir}
  restorecon -R ${dir}
done

echo UUID=$(blkid -s UUID -o value ${LOCALVOLDEVICE}) ${LOCALDIR} xfs
defaults,discard,gquota 0 2 >> /etc/fstab
echo UUID=$(blkid -s UUID -o value ${ETCDDEVICE}) ${ETCDDIR} xfs
defaults,discard 0 2 >> /etc/fstab
echo UUID=$(blkid -s UUID -o value ${CONTAINERSDEVICE}) ${CONTAINERSDIR}
```

```
xfs defaults,discard 0 2 >> /etc/fstab

mount -a
```

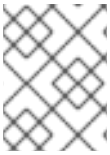
NOTE

The previous customization script should be named **`master.sh`**

The master instances are created using the following bash loop:

Master instances

```
# Disks multizone and single zone support
$ eval "$MYZONES_LIST"

$ for i in $(seq 0 $((${MASTER_NODE_COUNT}-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute disks create ${CLUSTERID}-master-${i}-etcd \
    --type=pd-ssd --size=${ETCDSIZE} --zone=${zone[$i]}
  gcloud compute disks create ${CLUSTERID}-master-${i}-containers \
    --type=pd-ssd --size=${MASTERCONTAINERSSIZE} --zone=${zone[$i]}
  gcloud compute disks create ${CLUSTERID}-master-${i}-local \
    --type=pd-ssd --size=${MASTERLOCALSIZE} --zone=${zone[$i]}
done

# Master instances multizone and single zone support
$ for i in $(seq 0 $((${MASTER_NODE_COUNT}-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances create ${CLUSTERID}-master-${i} \
    --async --machine-type=${MASTERSIZE} \
    --subnet=${CLUSTERID_SUBNET} \
    --address="" --no-public-ptr \
    --maintenance-policy=MIGRATE \
    --
scopes=https://www.googleapis.com/auth/cloud.useraccounts.readonly,https:/
/www.googleapis.com/auth/compute,https://www.googleapis.com/auth/devstorag
e.read_only,https://www.googleapis.com/auth/logging.write,https://www.goog
leapis.com/auth/monitoring.write,https://www.googleapis.com/auth/service.m
anagement.readonly,https://www.googleapis.com/auth/servicecontrol \
    --tags=${CLUSTERID}-master,${CLUSTERID}-node \
    --metadata "ocp-cluster=${CLUSTERID},${CLUSTERID}-type=master" \
    --image=${RHELIMAGE}  --image-project=${IMAGEPROJECT} \
    --boot-disk-size=${MASTERDISKSIZE} --boot-disk-type=pd-ssd \
    --boot-disk-device-name=${CLUSTERID}-master-${i} \
    --disk=name=${CLUSTERID}-master-${i}-etcd,device-name=${CLUSTERID}-
master-${i}-etcd,mode=rw,boot=no \
    --disk=name=${CLUSTERID}-master-${i}-containers,device-
name=${CLUSTERID}-master-${i}-containers,mode=rw,boot=no \
    --disk=name=${CLUSTERID}-master-${i}-local,device-name=${CLUSTERID}-
master-${i}-local,mode=rw,boot=no \
    --metadata-from-file startup-script=./master.sh \
    --zone=${zone[$i]}
done
```



NOTE

The customization script is executed at boot time as it specified with the **`--metadata-from-file startup-script=./master.sh`** flag.

2.9.4. Infrastructure and application nodes

The following script needs to be created in the local workstation to be used in the instance for customization at boot time and it is valid for both infrastructure and application nodes:

Node customization

```
$ vi node.sh
#!/bin/bash
LOCALVOLDEVICE=$(readlink -f /dev/disk/by-id/google-*local*)
CONTAINERSDEVICE=$(readlink -f /dev/disk/by-id/google-*containers*)
LOCALDIR="/var/lib/origin/openshift.local.volumes"
CONTAINERSDIR="/var/lib/docker"

for device in ${LOCALVOLDEVICE} ${CONTAINERSDEVICE}
do
  mkfs.xfs ${device}
done

for dir in ${LOCALDIR} ${CONTAINERSDIR}
do
  mkdir -p ${dir}
  restorecon -R ${dir}
done

echo UUID=$(blkid -s UUID -o value ${LOCALVOLDEVICE}) ${LOCALDIR} xfs
defaults,discard,gquota 0 2 >> /etc/fstab
echo UUID=$(blkid -s UUID -o value ${CONTAINERSDEVICE}) ${CONTAINERSDIR}
xfs defaults,discard 0 2 >> /etc/fstab

mount -a
```

NOTE

The previous customization script should be named **node.sh**

The infrastructure node instances are created using the following bash loop:

Infrastructure instances

```
# Disks multizone and single zone support
$ eval "$MYZONES_LIST"

$ for i in $(seq 0 $(($INFRA_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute disks create ${CLUSTERID}-infra-${i}-containers \
  --type=pd-ssd --size=${INFRACONTAINERSSIZE} --zone=${zone[$i]}
  gcloud compute disks create ${CLUSTERID}-infra-${i}-local \
  --type=pd-ssd --size=${INFRALOCALSIZE} --zone=${zone[$i]}
```

```
done

# Infrastructure instances multizone and single zone support
$ for i in $(seq 0 $(($INFRA_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances create ${CLUSTERID}-infra-${i} \
    --async --machine-type=${INFRASIZE} \
    --subnet=${CLUSTERID_SUBNET} \
    --address="" --no-public-ptr \
    --maintenance-policy=MIGRATE \
    --
scopes=https://www.googleapis.com/auth/cloud.useraccounts.readonly,https:/
/www.googleapis.com/auth/compute,https://www.googleapis.com/auth/devstorag
e.read_write,https://www.googleapis.com/auth/logging.write,https://www.goo
gleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/service.
management.readonly,https://www.googleapis.com/auth/servicecontrol \
    --tags=${CLUSTERID}-infra,${CLUSTERID}-node,${CLUSTERID}ocp \
    --metadata "ocp-cluster=${CLUSTERID},${CLUSTERID}-type=infra" \
    --image=${RHELIMAGE}  --image-project=${IMAGEPROJECT} \
    --boot-disk-size=${INFRADISKSIZE} --boot-disk-type=pd-ssd \
    --boot-disk-device-name=${CLUSTERID}-infra-${i} \
    --disk=name=${CLUSTERID}-infra-${i}-containers,device-
name=${CLUSTERID}-infra-${i}-containers,mode=rw,boot=no \
    --disk=name=${CLUSTERID}-infra-${i}-local,device-name=${CLUSTERID}-
infra-${i}-local,mode=rw,boot=no \
    --metadata-from-file startup-script=./node.sh \
    --zone=${zone[$i]}
done
```

NOTE

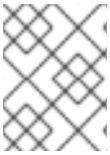
The customization script is executed at boot time as it specified with the **`--metadata-from-file startup-script=./node.sh`** flag.

The application node instances are created using the following bash loop:

Application instances

```
# Disks multizone and single zone support
$ eval "$MYZONES_LIST"

$ for i in $(seq 0 $(($APP_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute disks create ${CLUSTERID}-app-${i}-containers \
  --type=pd-ssd --size=${APPCONTAINERSSIZE} --zone=${zone[$i]}
  gcloud compute disks create ${CLUSTERID}-app-${i}-local \
  --type=pd-ssd --size=${APPLOCALSIZE} --zone=${zone[$i]}
done

# Application instances multizone and single zone support
$ for i in $(seq 0 $(($APP_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances create ${CLUSTERID}-app-${i} \
    --async --machine-type=${APPSIZE} \
    --subnet=${CLUSTERID_SUBNET} \
```

```
    --address="" --no-public-ptr \
    --maintenance-policy=MIGRATE \
    --
scopes=https://www.googleapis.com/auth/cloud.useraccounts.readonly,https:/
/www.googleapis.com/auth/compute,https://www.googleapis.com/auth/devstorag
e.read_only,https://www.googleapis.com/auth/logging.write,https://www.goog
leapis.com/auth/monitoring.write,https://www.googleapis.com/auth/service.m
anagement.readonly,https://www.googleapis.com/auth/servicecontrol \
    --tags=${CLUSTERID}-node,${CLUSTERID}ocp \
    --metadata "ocp-cluster=${CLUSTERID},${CLUSTERID}-type=app" \
    --image=${RHELIMAGE}  --image-project=${IMAGEPROJECT} \
    --boot-disk-size=${APPDISKSIZE} --boot-disk-type=pd-ssd \
    --boot-disk-device-name=${CLUSTERID}-app-${i} \
    --disk=name=${CLUSTERID}-app-${i}-containers,device-name=${CLUSTERID}-
app-${i}-containers,mode=rw,boot=no \
    --disk=name=${CLUSTERID}-app-${i}-local,device-name=${CLUSTERID}-app-
${i}-local,mode=rw,boot=no \
    --metadata-from-file startup-script=./node.sh \
    --zone=${zone[$i]}
done
```



NOTE

The customization script is executed at boot time as it specified with the **`--metadata-from-file startup-script=./node.sh`** flag.

NOTE

Notice the differences on the tags as different firewall rules need to be applied to different instances.

2.10. CREATING LOAD BALANCERS

Load balancers are used to provide highly-availability to the Red Hat OpenShift Container Platform master and router services.

2.10.1. Master Load Balancer

The master load balancer requires a static public IP address which is used when specifying the OpenShift public hostname(*refarch-ocp.example.com*). The load balancer uses probes to validate that instances in the backend pools are available.

As explained in the Section 1.1.4.3, “Load balancing” section, there are different kinds of Google Cloud Platform load balancers. TCP global load balancing is used for masters load balancing.

The load balancer uses probes to validate that instances in the backend pools are available. The probe request the "/healthz" endpoint of the Red Hat OpenShift Container Platform API at 443/TCP port.

Master load balancer health check

```
# Health check
$ gcloud compute health-checks create https ${CLUSTERID}-master-lb-
healthcheck \
```

```
    --port 443 --request-path "/healthz" --check-interval=10s --
timeout=10s \
    --healthy-threshold=3 --unhealthy-threshold=3
```

A backend is created using the health check and using client IP affinity rule to ensure client connections are redirected to the same backend server as the initial connection occurred to avoid websocket communication timeouts.

Master load balancer backend

```
# Create backend and set client ip affinity to avoid websocket timeout
$ gcloud compute backend-services create ${CLUSTERID}-master-lb-backend \
    --global \
    --protocol TCP \
    --session-affinity CLIENT_IP \
    --health-checks ${CLUSTERID}-master-lb-healthcheck \
    --port-name ocp-api
```

An **unmanaged** instance group is created in every zone to host the master instances and then are added to the backend service.

Master load balancer backend addition

```
$ eval "$MYZONES_LIST"

# Multizone and single zone support for instance groups
$ for i in $(seq 0 $((${#ZONES[@]}-1))); do
  ZONE=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instance-groups unmanaged create ${CLUSTERID}-masters-
${ZONE} \
    --zone=${ZONE}
  gcloud compute instance-groups unmanaged set-named-ports \
    ${CLUSTERID}-masters-${ZONE} --named-ports=ocp-api:443 --zone=${ZONE}
  gcloud compute instance-groups unmanaged add-instances \
    ${CLUSTERID}-masters-${ZONE} --instances=${CLUSTERID}-master-${i} \
    --zone=${ZONE}
  # Instances are added to the backend service
  gcloud compute backend-services add-backend ${CLUSTERID}-master-lb-
backend \
    --global \
    --instance-group ${CLUSTERID}-masters-${ZONE} \
    --instance-group-zone ${ZONE}
done
```

The TCP proxy rule is created with no proxy header to be transparent and using the backend service previously created.

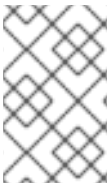
Master load balancer tcp proxy

```
# Do not set any proxy header to be transparent
$ gcloud compute target-tcp-proxies create ${CLUSTERID}-master-lb-target-
proxy \
    --backend-service ${CLUSTERID}-master-lb-backend \
    --proxy-header NONE
```

Finally, create the forwarding rules and allow health checks to be performed from Google's health check IPs.

Master load balancer forwarding rules

```
$ export LBIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-master-lb" --format="value(address)")

# Forward only 443/tcp port
$ gcloud compute forwarding-rules create \
    ${CLUSTERID}-master-lb-forwarding-rule \
    --global \
    --target-tcp-proxy ${CLUSTERID}-master-lb-target-proxy \
    --address ${LBIP} \
    --ports 443

# Allow health checks from Google health check IPs
$ gcloud compute firewall-rules create ${CLUSTERID}-healthcheck-to-lb \
  --direction=INGRESS --priority=1000 --network=${CLUSTERID_NETWORK} \
  --source-ranges 130.211.0.0/22,35.191.0.0/16 \
  --target-tags ${CLUSTERID}-master \
  --allow tcp:443
```



NOTE

See https://cloud.google.com/compute/docs/load-balancing/health-checks#health_check_source_ips_and_firewall_rules for more information on health checks and Google IP addresses performing the health checks.

2.10.2. Applications Load Balancer

The applications load balancer requires a static public IP address which is used when specifying the applications wildcard public hostname (**.myocp-apps.example.com*). The load balancer uses probes to validate that instances in the backend pools are available.

As explained in the Section 1.1.4.3, "Load balancing" section, there are different kinds of Google Cloud Platform load balancers. Network load balancing is used for applications load balancing.

The load balancer uses probes to validate that instances in the backend pools are available. The probe request the "/healthz" endpoint of the Red Hat OpenShift Container Platform routers at 1936/TCP port (https probe)

Applications load balancer health check

```
# Health check
$ gcloud compute http-health-checks create ${CLUSTERID}-infra-lb-
healthcheck \
    --port 1936 --request-path "/healthz" --check-interval=10s --
timeout=10s \
    --healthy-threshold=3 --unhealthy-threshold=3
```

A target pool is created using the health check and infrastructure nodes are added.

Applications load balancer target pool

```
# Target Pool
$ gcloud compute target-pools create ${CLUSTERID}-infra \
    --http-health-check ${CLUSTERID}-infra-lb-healthcheck

$ for i in $(seq 0 $(($INFRA_NODE_COUNT-1))); do
  gcloud compute target-pools add-instances ${CLUSTERID}-infra \
  --instances=${CLUSTERID}-infra-${i}
done
```

Finally, create the forwarding rules to 80/TCP and 443/TCP.

Applications load balancer forwarding rules

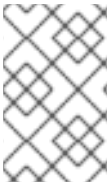
```
# Forwarding rules and firewall rules
$ export APPSLBIP=$(gcloud compute addresses list \
  --filter="name:${CLUSTERID}-apps-lb" --format="value(address)")

$ gcloud compute forwarding-rules create ${CLUSTERID}-infra-http \
    --ports 80 \
    --address ${APPSLBIP} \
    --region ${REGION} \
    --target-pool ${CLUSTERID}-infra

$ gcloud compute forwarding-rules create ${CLUSTERID}-infra-https \
    --ports 443 \
    --address ${APPSLBIP} \
    --region ${REGION} \
    --target-pool ${CLUSTERID}-infra
```

2.11. CREATING RED HAT OPENSHIFT CONTAINER PLATFORM REGISTRY STORAGE

To leverage Google Cloud Platform cloud storage, the Red Hat OpenShift Container Platform uses a dedicated bucket to store images. Using Google Cloud Platform Cloud Storage allow for the registry to grow dynamically in size without the need for intervention from an Administrator.

Registry Storage

```
# Bucket to host registry
$ gsutil mb -l ${REGION} gs://${CLUSTERID}-registry

$ cat <<EOF > labels.json
{
  "ocp-cluster": "${CLUSTERID}"
}
EOF

$ gsutil label set labels.json gs://${CLUSTERID}-registry

$ rm -f labels.json
```



NOTE

Google Cloud Platform buckets are global objects. This means that every bucket name must be unique. If the bucket creation process fails because the name is used, select a different bucket name.

2.12. CREATING CNS INSTANCES (OPTIONAL)

The following script needs to be created in the local workstation to be used in the CNS instance for customization at boot time:

CNS customization

```
$ vi cns.sh
#!/bin/bash
CONTAINERSDEVICE=$(readlink -f /dev/disk/by-id/google-*containers*)
CONTAINERSDIR="/var/lib/docker"

mkfs.xfs ${CONTAINERSDEVICE}
mkdir -p ${CONTAINERSDIR}
restorecon -R ${CONTAINERSDIR}

echo UUID=$(blkid -s UUID -o value ${CONTAINERSDEVICE}) ${CONTAINERSDIR}
xfs defaults,discard 0 2 >> /etc/fstab

mount -a
```

NOTE

The previous customization script should be named **cns.sh**

The CNS node instances are created using the following bash loop:

CNS instances

```
# Disks multizone and single zone support
$ eval "$MYZONES_LIST"

$ for i in $(seq 0 $(($CNS_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute disks create ${CLUSTERID}-cns-${i}-containers \
  --type=pd-ssd --size=${CNSCONTAINERSSIZE} --zone=${zone[$i]}
  gcloud compute disks create ${CLUSTERID}-cns-${i}-gluster \
  --type=pd-ssd --size=${CNSGLUSTERSIZE} --zone=${zone[$i]}
done

# CNS instances multizone and single zone support
$ for i in $(seq 0 $(($CNS_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances create ${CLUSTERID}-cns-${i} \
    --async --machine-type=${CNSSIZE} \
    --subnet=${CLUSTERID_SUBNET} \
    --address="" --no-public-ptr \
    --maintenance-policy=MIGRATE \
```

```
    --
scopes=https://www.googleapis.com/auth/cloud.useraccounts.readonly,https:/
/www.googleapis.com/auth/compute,https://www.googleapis.com/auth/devstorag
e.read_write,https://www.googleapis.com/auth/logging.write,https://www.goo
gleapis.com/auth/monitoring.write,https://www.googleapis.com/auth/service.
management.readonly,https://www.googleapis.com/auth/servicecontrol\
    --tags=${CLUSTERID}-cns,${CLUSTERID}-node,${CLUSTERID}ocp \
    --metadata "ocp-cluster=${CLUSTERID},${CLUSTERID}-type=cns" \
    --image=${RHELIMAGE} --image-project=${IMAGEPROJECT} \
    --boot-disk-size=${CNSDISKSIZE} --boot-disk-type=pd-ssd \
    --boot-disk-device-name=${CLUSTERID}-cns-${i} \
    --disk=name=${CLUSTERID}-cns-${i}-containers,device-name=${CLUSTERID}-
cns-${i}-containers,mode=rw,boot=no \
    --disk=name=${CLUSTERID}-cns-${i}-gluster,device-name=${CLUSTERID}-
cns-${i}-gluster,mode=rw,boot=no \
    --metadata-from-file startup-script=./cns.sh \
    --zone=${zone[$i]}
done
```

CNS instances should have a minimum of 4vCPUs and 32GB of RAM. These instances by default only schedule the **`glusterfs`** pods, therefore the extra storage disks are to host container images and for **`glusterfs`** storage.

2.13. REMOVING STARTUP SCRIPTS

To avoid rerunning the startup scripts and wipe the whole content of the attached disks, the startup scripts are disabled after the first boot:

Disable startup scripts

```
$ eval "$MYZONES_LIST"

# Masters
$ for i in $(seq 0 $((${MASTER_NODE_COUNT}-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances remove-metadata \
    --keys startup-script ${CLUSTERID}-master-${i} --zone=${zone[$i]}
done

# Application nodes
$ for i in $(seq 0 $(($APP_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances remove-metadata \
    --keys startup-script ${CLUSTERID}-app-${i} --zone=${zone[$i]}
done

# Infrastructure nodes
$ for i in $(seq 0 $(($INFRA_NODE_COUNT-1))); do
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances remove-metadata \
    --keys startup-script ${CLUSTERID}-infra-${i} --zone=${zone[$i]}
done

# CNS nodes
$ for i in $(seq 0 $(($CNS_NODE_COUNT-1))); do
```

```
  zone[$i]=${ZONES[$i % ${#ZONES[@]}]}
  gcloud compute instances remove-metadata \
    --keys startup-script ${CLUSTERID}-cns-${i} --zone=${zone[$i]}
done
```

2.14. CONFIGURING BASTION FOR RED HAT OPENSHIFT CONTAINER PLATFORM

The following subsections describe all the steps needed to use the bastion instance as a jump host to access the instances in the private subnet.

2.14.1. Configuring ~/.ssh/config to use Bastion as Jumphost

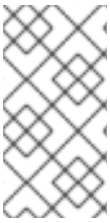
To easily connect to the Red Hat OpenShift Container Platform environment, follow the steps below.

On the local workstation with private key previously created:

```
$ eval $(ssh-agent -s)
$ ssh-add /path/to/<keypair-name>
Identity added: /path/to/<keypair-name> (/path/to/<keypair-name>)
```

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

```
$ ssh -A <myuser>@bastion
```



NOTE

As the instances are using the Linux Guest Environment for Google Compute Engine instead cloud-init, the user is created at boot time and the ssh key injected. This means in order to be able to access the instance, the keypair user should be used instead the default 'cloud-user' that comes with Red Hat Enterprise Linux.

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

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

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

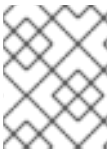
NOTE

No password should be prompted if working properly.

```
$ ssh ${CLUSTERID}-master-2
```

After verifying the previous steps work and in order to simplify the process, the following snippet can be added to the **~/.ssh/config** file:

```
Host bastion
    HostName        ${CLUSTERID}-bastion.${DOMAIN}
    User            <myuser>
    IdentityFile    /path/to/<keypair-name>
    ForwardAgent     yes

Host ${CLUSTERID}-*
    ProxyCommand    ssh <myuser>@bastion -W %h:%p
    IdentityFile    /path/to/<keypair-name>
    User            <myuser>
```



NOTE

keypair-name needs to be replaced with the proper keypair (**~/.ssh/gcp_key** in previous examples) as well as the user.

Then from the local workstation ssh can 'jump' to any instance as ssh uses the bastion host as a jump host:

```
$ ssh ${CLUSTERID}-master-2
```

2.15. RED HAT OPENSHIFT CONTAINER PLATFORM PREREQUISITES

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

2.15.1. OpenShift Authentication

Red Hat OpenShift Container Platform provides the ability to use many different authentication platforms. For this reference architecture, Google's OpenID Connect Integration is used. A listing of other authentication options are available at Configuring Authentication and User Agent.

When configuring the authentication, the following parameters must be added to the ansible inventory. An example is shown below.

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

NOTE

Multiple authentication providers can be supported depending on the needs of the end users.

NOTE

The following tasks should be performed on the bastion host.

2.15.2. Ansible Setup

Register the bastion host and install the required packages.

Bastion preparation

```
$ sudo subscription-manager register --username <rhuser> --password <pass>
# or if using activation key
# sudo subscription-manager register --activationkey=<ak> --org=<orgid>
$ sudo subscription-manager attach --pool=<poolid>
$ sudo subscription-manager repos --disable="*" \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.9-rpms" \
    --enable="rhel-7-fast-datapath-rpms" \
    --enable="rhel-7-server-ansible-2.4-rpms"

$ sudo yum install atomic-openshift-utils
# Optionally, update to the latest packages and reboot the host
$ sudo yum update
$ sudo reboot
```

It is recommended to configure ansible.cfg as follows:

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

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

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

NOTE

See Ansible Install Optimization for more information.

2.15.3. Inventory File

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

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

WARNING

The following inventory needs to be adjusted to the environment about to be deployed and substitute bash variables with the real values.

```
$ vi inventory
[OSEv3:children]
masters
etcd
nodes

[OSEv3:vars]
ansible_become=true
openshift_release=v3.9
os_firewall_use_firewalld=True
openshift_clock_enabled=true

openshift_cloudprovider_kind=gce
openshift_gcp_project=${PROJECTID}
openshift_gcp_prefix=${CLUSTERID}
# If deploying single zone cluster set to "False"
openshift_gcp_multizone="True"
openshift_gcp_network_name=${CLUSTERID}-net

openshift_master_api_port=443
openshift_master_console_port=443

openshift_node_local_quota_per_fsgroup=512Mi

openshift_hosted_registry_replicas=1
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_gcs_bucket=${CLUSTERID}-registry

openshift_master_cluster_method=native
openshift_master_cluster_hostname=${CLUSTERID}-ocp.${DOMAIN}
openshift_master_cluster_public_hostname=${CLUSTERID}-ocp.${DOMAIN}
openshift_master_default_subdomain=${CLUSTERID}-apps.${DOMAIN}

os_sdn_network_plugin_name=redhat/openshift-ovs-networkpolicy

deployment_type=openshift-enterprise
```

```
# Required per https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-
server-rhel7
openshift_storage_glusterfs_block_image=registry.access.redhat.com/rhgs3/r
hgs-gluster-block-prov-rhel7
openshift_storage_glusterfs_s3_image=registry.access.redhat.com/rhgs3/rhgs
-s3-server-rhel7
openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/
rhgs-volmanager-rhel7

# Service catalog
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

# Metrics
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}

# Aggregated logging
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

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

[masters]
${CLUSTERID}-master-0
${CLUSTERID}-master-1
${CLUSTERID}-master-2

[etcd]
${CLUSTERID}-master-0
${CLUSTERID}-master-1
${CLUSTERID}-master-2

[nodes]
${CLUSTERID}-master-0 openshift_node_labels="{'region': 'master'}"
${CLUSTERID}-master-1 openshift_node_labels="{'region': 'master'}"
```

```
${CLUSTERID}-master-2 openshift_node_labels="{'region': 'master'}"
${CLUSTERID}-infra-0 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
${CLUSTERID}-infra-1 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
${CLUSTERID}-infra-2 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
${CLUSTERID}-app-0 openshift_node_labels="{'region': 'apps'}"
${CLUSTERID}-app-1 openshift_node_labels="{'region': 'apps'}"
${CLUSTERID}-app-2 openshift_node_labels="{'region': 'apps'}"
```

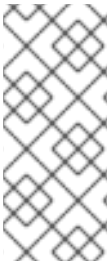
NOTE

The **`openshift_gcp_*`** values are required for Red Hat OpenShift Container Platform to be able to create Google Cloud Platform resources such as disks for persistent volumes or **`LoadBalancer`** type services.

2.15.3.1. CNS Inventory (Optional)

If CNS is used in the Red Hat OpenShift Container Platform installation specific variables must be set in the inventory.

```
$ vi inventory
[OSEv3:children]
masters
etcd
nodes
glusterfs

....omitted...

[nodes]
....omitted...
${CLUSTERID}-cns-0 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"
${CLUSTERID}-cns-1 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"
${CLUSTERID}-cns-2 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"

[glusterfs]
${CLUSTERID}-cns-0 glusterfs_devices='[ "/dev/disk/by-
id/google-${CLUSTERID}-cns-0-gluster" ]'
openshift_node_local_quota_per_fsgroup=""
${CLUSTERID}-cns-1 glusterfs_devices='[ "/dev/disk/by-
id/google-${CLUSTERID}-cns-1-gluster" ]'
openshift_node_local_quota_per_fsgroup=""
${CLUSTERID}-cns-2 glusterfs_devices='[ "/dev/disk/by-
id/google-${CLUSTERID}-cns-2-gluster" ]'
openshift_node_local_quota_per_fsgroup=""
```



NOTE

As the CNS nodes are not intended to run pods with local storage and they don't have a dedicated disk for **`/var/lib/origin/openshift.local.volumes`**, the **`openshift_node_local_quota_per_fsgroup`** requires to be disabled on those nodes otherwise the installation complains about the filesystem not mounted with the **`gquota`** option.

The following is an example of a full inventory including CNS nodes:

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

[OSEv3:vars]
ansible_become=true
openshift_release=v3.9
os_firewall_use_firewalld=True
openshift_clock_enabled=true

openshift_cloudprovider_kind=gce
openshift_gcp_project=refarch-204310
openshift_gcp_prefix=refarch
# If deploying single zone cluster set to "False"
openshift_gcp_multizone="True"
openshift_gcp_network_name=refarch-net

openshift_master_api_port=443
openshift_master_console_port=443

openshift_node_local_quota_per_fsgroup=512Mi

openshift_hosted_registry_replicas=1
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=gcs
openshift_hosted_registry_storage_gcs_bucket=refarch-registry

openshift_master_cluster_method=native
openshift_master_cluster_hostname=refarch.gce.example.com
openshift_master_cluster_public_hostname=refarch.gce.example.com
openshift_master_default_subdomain=refarch-apps.gce.example.com

os_sdn_network_plugin_name=redhat/openshift-ovs-networkpolicy

deployment_type=openshift-enterprise

# Required per https://access.redhat.com/solutions/3480921
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-
server-rhel7
openshift_storage_glusterfs_block_image=registry.access.redhat.com/rhgs3/r
hgs-gluster-block-prov-rhel7
```

```
openshift_storage_glusterfs_s3_image=registry.access.redhat.com/rhgs3/rhgs
-s3-server-rhel7
openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/
rhgs-volmanager-rhel7

# Service catalog
openshift_hosted_etcd_storage_kind=dynamic
openshift_hosted_etcd_storage_volume_name=etcd-vol
openshift_hosted_etcd_storage_access_modes=["ReadWriteOnce"]
openshift_hosted_etcd_storage_volume_size=1G
openshift_hosted_etcd_storage_labels={'storage': 'etcd'}

# Metrics
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_storage_volume_size=20Gi
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}

# Aggregated logging
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

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

[masters]
refarch-master-0
refarch-master-1
refarch-master-2

[etcd]
refarch-master-0
refarch-master-1
refarch-master-2

[nodes]
refarch-master-0 openshift_node_labels="{'region': 'master'}"
refarch-master-1 openshift_node_labels="{'region': 'master'}"
refarch-master-2 openshift_node_labels="{'region': 'master'}"
refarch-infra-0 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
refarch-infra-1 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
refarch-infra-2 openshift_node_labels="{'region': 'infra', 'node-
role.kubernetes.io/infra': 'true'}"
```

```
refarch-app-0 openshift_node_labels="{'region': 'apps'}"
refarch-app-1 openshift_node_labels="{'region': 'apps'}"
refarch-app-2 openshift_node_labels="{'region': 'apps'}"
refarch-cns-0 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"
refarch-cns-1 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"
refarch-cns-2 openshift_node_labels="{'region': 'cns', 'node-
role.kubernetes.io/cns': 'true'}"

[glusterfs]
refarch-cns-0 glusterfs_devices='[ "/dev/disk/by-id/google-refarch*-cns-0-
gluster" ]' openshift_node_local_quota_per_fsgroup=""
refarch-cns-1 glusterfs_devices='[ "/dev/disk/by-id/google-refarch*-cns-1-
gluster" ]' openshift_node_local_quota_per_fsgroup=""
refarch-cns-2 glusterfs_devices='[ "/dev/disk/by-id/google-refarch*-cns-2-
gluster" ]' openshift_node_local_quota_per_fsgroup=""
```

2.15.4. Node Registration

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

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

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

OR

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

2.15.5. Repository Setup

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

```
$ ansible nodes -i inventory -b -m shell -a \
    'subscription-manager repos --disable="*" \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.9-rpms" \
    --enable="rhel-7-fast-datapath-rpms" \
    --enable="rhel-7-server-ansible-2.4-rpms"'
```

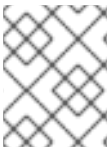
2.15.6. Preflight checks and other configurations

By default, the health check port in the infrastructure nodes is blocked by iptables rules.

```
$ ansible *-infra-* -i inventory -b -m firewalld -a \
    "port=1936/tcp permanent=true state=enabled"
```

If using **`LoadBalancer`** type services, port 10256/tcp should be opened for health checks:

```
$ ansible nodes -i inventory -b -m firewalld -a \
    "port=10256/tcp permanent=true state=enabled"
```



NOTE

Red Hat is investigating this requirement in this Bugzilla to be fixed in future Red Hat OpenShift Container Platform releases.

It is recommended to update all packages to the latest version and reboot the nodes before the installation to ensure all services are using the updated bits:

```
$ ansible all -i inventory -b -m yum -a "name=* state=latest"
$ ansible all -i inventory -b -m command -a "reboot"
```

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

```
$ ssh <instance>
$ lsblk
$ mount
$ free -m
$ cat /proc/cpuinfo
$ sudo journalctl
$ sudo yum repolist
```

2.15.7. Red Hat OpenShift Container Platform Prerequisites Playbook

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

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

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

CHAPTER 3. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM

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

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

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

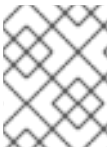
The playbooks runs through the complete process of installing Red Hat OpenShift Container Platform and reports a playbook recap showing the number of changes and errors (if any).

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

INSTALLER STATUS
*********************************************************************
*********
Initialization             : Complete (0:00:38)
Health Check               : Complete (0:01:02)
etcd Install               : Complete (0:01:03)
Master Install             : Complete (0:03:31)
Master Additional Install  : Complete (0:00:45)
Node Install               : Complete (0:04:39)
Hosted Install             : Complete (0:02:06)
Web Console Install        : Complete (0:00:40)
Metrics Install            : Complete (0:01:23)
Logging Install            : Complete (0:03:39)
Service Catalog Install    : Complete (0:01:26)


Tuesday 29 August 2018  10:34:49 -0400 (0:00:01.002)       0:29:54.775
********
===============================================================================
=====
openshift_hosted : Ensure OpenShift router correctly rolls out (best-
effort today) -- 92.44s
openshift_hosted : Ensure OpenShift registry correctly rolls out (best-
effort today) -- 61.93s
openshift_health_check ---------------------------------------------
```

```
- 53.92s
openshift_common : Install the base package for versioning ------------
42.15s
openshift_common : Install the base package for versioning ------------
36.36s
openshift_hosted : Sanity-check that the OpenShift registry rolled out
correctly -- 31.43s
cockpit : Install cockpit-ws ------------------------------------------
27.65s
openshift_version : Get available atomic-openshift version ------------
25.27s
etcd_server_certificates : Install etcd -------------------------------
15.53s
openshift_master : Wait for master controller service to start on first
master -- 15.21s
openshift_master : pause ----------------------------------------------
- 15.20s
openshift_node : Configure Node settings ------------------------------
13.56s
openshift_excluder : Install openshift excluder -----------------------
13.54s
openshift_node : Install sdn-ovs package ------------------------------
13.45s
openshift_master : Create master config -------------------------------
11.92s
openshift_master : Create the scheduler config ------------------------
10.92s
openshift_master : Create the policy file if it does not already exist --
10.65s
openshift_node : Install Node service file ----------------------------
10.43s
openshift_node : Install Node package ---------------------------------
10.39s
openshift_node : Start and enable node --------------------------------
- 8.96s
```

3.1. CLOUDFORMS INTEGRATION (OPTIONAL)

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



NOTE

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

3.1.1. Requesting the Red Hat OpenShift Container Platform Management Token

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

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

```
oc sa get-token -n management-infra management-admin
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY
2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJtYW5hZ2V
tZW50LWluZnJhIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6I
m1hbmFnZW1lbnQtYWRtaW4tdG9rZW4tdHM0cTIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2N
vdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoibWFuYWdlbWVudC1hZG1pbiIsImt1YmVybmV0Z
XMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6ImY0ZDlmMGMxLTEyY2Y
tMTFlOC1iNTgzLWZhMTYzZTEwNjNlYSIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDptY
W5hZ2VtZW50LWluZnJhOm1hbmFnZW1lbnQtYWRtaW4ifQ.LwNm0652paGcJu7m63PxBhs4mjXw
YcqMS5KD-
0aWkEMCPo64WwNEawyyYH31SvuEPaE6qFxZwDdJHwdNsfq1CjUL4BtZHv1I2QZxpVl6gMBQowN
f6fWSeGe1FDZ4lkLjzAoMOCFUWA0Z7lZM1FAlyjfz2LkPNKaFW0ffelSJ2SteuXB_4FNup-
T5bKEPQf2pyrwvs2DadClyEEKpIrdZxuekJ9ZfIubcSc3pp1dZRu8wgmSQSLJ1N75raaUU5obu
9cHjcbB9jpDhTW347oJOoL_Bj4bf0yyuxjuUCp3f4fs1qhyjHb5N5LKKBPgIKzoQJrS7j9Sqzo
9TDMF9YQ5JLQ
```

3.1.2. Adding OpenShift as a Containtainer Provider

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

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

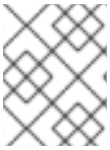
3.1.3. Adding Google Cloud Platform to Cloudforms

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

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

CHAPTER 4. OPERATIONAL MANAGEMENT

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



NOTE

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

4.1. OC ADM DIAGNOSTICS

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

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

4.2. USING THE DIAGNOSTICS TOOL

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

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

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

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

```
$ sudo oc adm diagnostics
```

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

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

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

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

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

The standard locations are listed below:

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

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

Available diagnostics include:

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

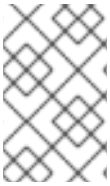
Due to potential changes the health check playbooks could make to the environment, the playbooks should only be run against clusters that have been deployed using Ansible with the same inventory file used during deployment. The changes consist of installing dependencies in order to gather required information. 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 Google Cloud Platform environments with the following tasks:

- Deciding between different internal network technologies
- Provisioning instances within Google Cloud Platform 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.