Reference Architectures 2018

Deploying and Managing OpenShift 3.9 on Azure

Last Updated: 2019-01-29

Reference Architectures 2018 Deploying and Managing OpenShift 3.9 on Azure

Ryan Cook
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 Microsoft Azure.

Table of Contents

COMMENTS AND FEEDBACK 4

EXECUTIVE SUMMARY 5

WHAT IS RED HAT OPENSHIFT CONTAINER PLATFORM 6

REFERENCE ARCHITECTURE SUMMARY 7

CHAPTER 1. COMPONENTS AND CONSIDERATIONS 9

1.1. AZURE INFRASTRUCTURE COMPONENTS 9
1.1.1. Resource groups 9
1.1.2. Azure DNS 9
1.1.3. Azure Virtual Networks 9
1.1.4. Network Security Groups 9
1.1.5. Availability set 9
1.1.6. Virtual Machines 10
1.1.7. Load Balancer 10
1.1.8. Storage Account 10
1.1.9. Service Principal 10
1.2. BASTION INSTANCE 10
1.3. RED HAT OPENSHIFT CONTAINER PLATFORM COMPONENTS 11
1.3.1. OpenShift Instances 11
1.3.1.1. Master Instances 11
1.3.1.2. Infrastructure Instances 12
1.3.1.3. Application Instances 12
1.3.2. etcd 13
1.3.3. Labels 13
1.3.3.1. Labels as Alternative Hierarchy 13
1.3.3.2. Labels as Node Selector 14
1.4. SOFTWARE DEFINED NETWORKING 14
1.4.1. OpenShift SDN Plugins 14
1.5. CONTAINER STORAGE 15
1.6. PERSISTENT STORAGE 15
1.6.1. Storage Classes 15
1.6.1.1. Persistent Volumes 16
1.7. REGISTRY 16
1.8. AGGREGATED LOGGING 16
1.9. AGGREGATED METRICS 18
1.10. CONTAINER-NATIVE STORAGE (OPTIONAL) 19
1.10.1. Prerequisites for Container-Native Storage 19
1.10.2. Firewall and Security Group Prerequisites 19

CHAPTER 2. RED HAT OPENSHIFT CONTAINER PLATFORM INSTANCE PREREQUISITES 21

2.1. AZURE CLI SETUP 21
2.2. SSH KEY 21
2.3. CREATING A RESOURCE GROUP 21
2.4. CREATING A RED HAT ENTERPRISE LINUX BASE IMAGE 21
2.5. CREATION OF RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKS 22
2.6. CREATING NETWORK SECURITY GROUPS 22
2.6.1. Bastion Security Group 22
2.6.2. Master Security Group 23
2.6.3. Infrastructure Node Security Group 25
2.6.4. Node Security Group 27

2.6.5. CNS Node Security Group (Optional) 28
2.7. OPENSHIFT LOAD BALANCERS 31
2.7.1. Master Load Balancer 31
2.7.2. Router Load Balancer 32
2.8. CREATING INSTANCES FOR RED HAT OPENSHIFT CONTAINER PLATFORM 33
2.8.1. Availability Sets 33
2.8.2. Master Instance Creation 34
2.8.3. Infrastructure Instance Creation 35
2.8.4. Application Instance Creation 35
2.8.5. CNS Instance Creation (Optional) 36
2.8.6. Deploying the Bastion Instance 37
2.8.7. DNS 38
2.8.8. Master Load Balancer Record 38
2.8.9. Router Load Balancer Record 38
2.8.10. Bastion Record 39
2.9. BASTION CONFIGURATION FOR RED HAT OPENSHIFT CONTAINER PLATFORM 39
2.9.1. Configure ~/.ssh/config to use Bastion as Jumphost 39
2.10. AZURE SERVICE PRINCIPAL 40
2.11. BLOB REGISTRY STORAGE 41
2.12. OPENSHIFT PREREQUISITES 41
2.12.1. Ansible Setup 41
2.12.2. OpenShift Authentication 41
2.12.3. Azure Account Key 42
2.12.4. Preparing the Inventory File 42
2.12.4.1. Cloud Provider Configuration 42
2.12.4.2. Creating the Installation Inventory 43
2.12.4.3. CNS Inventory (Optional) 45
2.12.5. Node Registration 45
2.12.6. Repository Setup 46
2.12.7. EmptyDir Storage 46
2.12.8. etcd Storage 46
2.12.9. Container Storage 47

CHAPTER 3. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM 48
3.1. STORAGE CLASS DEPLOYMENT 48
3.2. LOGGING 49
3.3. METRICS 49
3.4. SERVICE CATALOG 49
3.5. CLOUDFORMS INTEGRATION (OPTIONAL) 50
3.5.1. Requesting the Red Hat OpenShift Container Platform Management Token 50
3.5.2. Adding OpenShift as a Containtainer Provider 50
3.5.3. Adding Microsoft Azure to Cloudforms 50

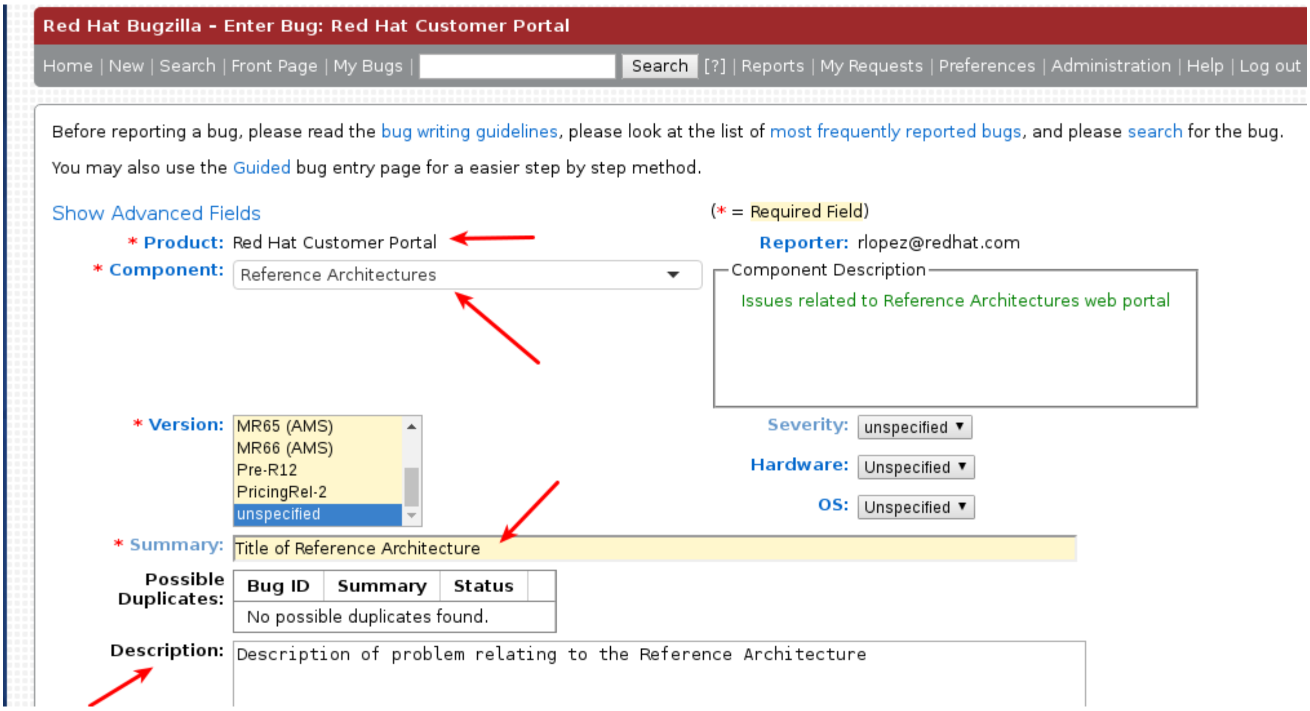
CHAPTER 4. OPERATIONAL MANAGEMENT 52
4.1. OC ADM DIAGNOSTICS 52
4.2. USING THE DIAGNOSTICS TOOL 52
4.3. RUNNING DIAGNOSTICS IN A SERVER ENVIRONMENT 55
4.4. RUNNING DIAGNOSTICS IN A CLIENT ENVIRONMENT 55
4.5. ANSIBLE-BASED HEALTH CHECKS 55
4.5.1. Running Health Checks via ansible-playbook 58
4.6. RUNNING HEALTH CHECKS VIA DOCKER CLI 58

CHAPTER 5. CONCLUSION 60

COMMENTS AND FEEDBACK

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

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


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

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

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

(* = Required Field)

* Product: Red Hat Customer Portal

Reporter: rlopez@redhat.com

* Component: Reference Architectures

Component Description: Issues related to Reference Architectures web portal

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

Severity: unspecified

Hardware: Unspecified

OS: Unspecified

* Summary: Title of Reference Architecture

Possible Duplicates:

Bug ID	Summary	Status
No possible duplicates found.		

Description: Description of problem relating to the Reference Architecture

EXECUTIVE SUMMARY

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

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

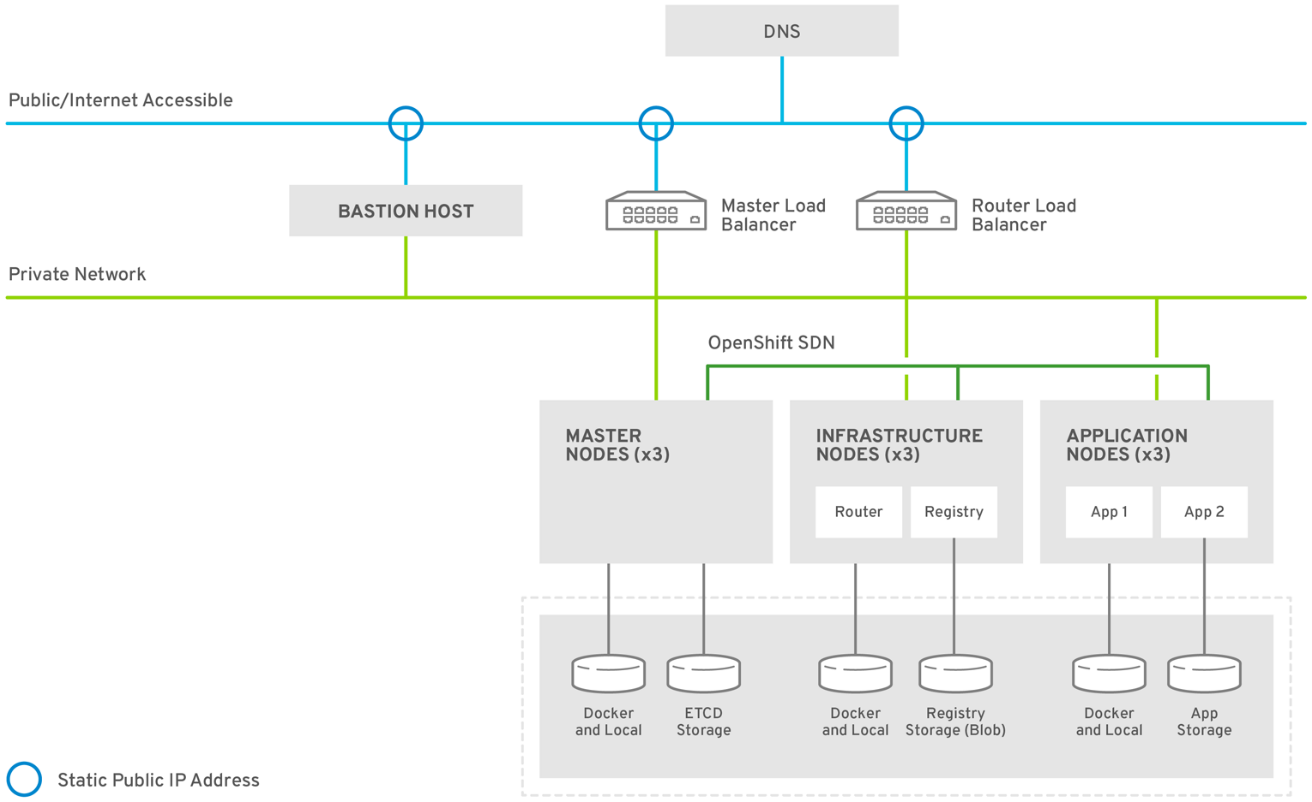
- *Which installation method do you want to use?*
- *How many instances do you require in the cluster?*
- *Is high availability required?*
- *Which installation type do you want to use: RPM or containerized?*
- *Is my installation supported if integrating with other Red Hat technologies?*

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.



OPENSHIFT_469951_0418

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

The Red Hat OpenShift Container Platform instances:

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

CHAPTER 1. COMPONENTS AND CONSIDERATIONS

1.1. AZURE INFRASTRUCTURE COMPONENTS

The successful installation of an Red Hat OpenShift Container Platform environment requires the following components to create a highly-available and full featured environment that is capable of using Microsoft Azure load balancers and Microsoft Azure storage.

1.1.1. Resource groups

Resource groups contain all of the Azure components from networking, load balancers,virtual machines, and DNS for a given deployment. Quotas and permissions can be applied to **`Resources Groups`** to control and manage resources deployed in the Microsoft Azure.

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-overview

1.1.2. Azure DNS

Azure offers a managed DNS service that provides internal and internet accessible hostname and load balancer resolution. The reference environment uses a DNS zone to host three DNS A records to allow for mapping of public IPs to OpenShift resources and a bastion.

https://docs.microsoft.com/en-us/azure/dns/dns-overview

1.1.3. Azure Virtual Networks

`Azure Virtual Networks` are used to isolate Azure cloud networks from one another. The virtual network is used by instances and load balancers to allow for communication with each other and to/from the internet. The virtual network allows for the creation of one to many subnets to be used by components within a resource group. Virtual Networks can also be connected to various VPN services to allow for communication with on-premise services.

https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview

1.1.4. Network Security Groups

`Network Security Groups` (NSGs) provide a list of rules to either or allow or deny traffic to resources deployed within an **`Azure Virtual Network`**. **`NSGs`** use numeric priority values and rules to define what items are allowed to communicate with each other. Restrictions can be defined to allow for restrictions to be placed on where communication is allowed to occur such as only within the **`VirtualNetwork`**, from load balancers, or from everywhere.

Priority values allow for administrators to grant very granular values on the order in which port communication is allowed or not allowed to occur.

https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-nsg

1.1.5. Availability set

Availability sets ensure that the VMs deployed are distributed across multiple isolated hardware nodes in a cluster. The distribution helps to ensure that when maintenance on the cloud provider hardware occurs, instances will not all be all running on one specific node.

https://docs.microsoft.com/en-us/azure/virtual-machines/linux/manage-availability

1.1.6. Virtual Machines

Virtual Machines are the instances that run the operating system of choice for a given deployment on Azure cloud.

https://docs.microsoft.com/en-us/azure/virtual-machines/linux/

1.1.7. Load Balancer

Load Balancers are used to allow network connectivity for scaling and high availability of services running on virtual machines within the Azure environment.

https://docs.microsoft.com/en-us/azure/load-balancer/load-balancer-overview

1.1.8. Storage Account

Storage Accounts are used to allow for resources, such as virtual machines, to access the different type of storage components offered by Microsoft Azure.

https://docs.microsoft.com/en-us/azure/storage/common/storage-quickstart-create-account

1.1.9. Service Principal

Azure offers the ability to create service accounts which can be used to access, manage, or create components within Azure. The service account grants API access to specific services within Microsoft Azure. For example, within Kubernetes and OpenShift persistent storage and load balancers can be requested by the instances using a Service Principal. Service Principals allow for granular access to be given to instances or users for specific Microsoft Azure functions.

https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-group-create-service-principal-portal

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 Microsoft Azure 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.

1.3.1.1. Master Instances

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

NOTE

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

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

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

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

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

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

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

NOTE

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

1.3.1.2. 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 OpenShift documentation.

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

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

NOTE

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

1.3.1.3. 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.

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

1.6. 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 Microsoft Azure 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 OpenShift documentation for more information.

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

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

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

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

NOTE

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

Basic concepts for aggregated logging

- Cluster: Set of Elasticsearch nodes distributing the workload
- Node: Container running an instance of Elasticsearch, part of the cluster.
- Index: Collection of documents (container logs)
- Shards and Replicas: Indices can be split into sets of data containing the primary copy of the documents stored (primary shards) or backups of that primary copies (replica shards). Sharding allows the application to horizontally scaled the information and distributed/paralellized operations. Replication instead provides high availability and also better search throughput as searches are also executed on replicas.

WARNING

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

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

NOTE

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

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

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

1.9. 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 1.1. 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 INSTANCE PREREQUISITES

A successful deployment of Red Hat OpenShift Container Platform requires many prerequisites. The prerequisites include the deployment of components in Microsoft Azure 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 Microsoft Azure environment are discussed in detail.

2.1. AZURE CLI SETUP

The Microsoft Azure cli can be used to deploy all of the components associated with this reference environment. This is one of many options for deploying instances and load balancers, creating network security groups, and any required accounts to sucessfully deploy a full functional Red Hat OpenShift Container Platform. In order to install the Azure CLI perform the following steps https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-yum?view=azure-cli-latest.

Once the Azure CLI has been installed, follow the directions to authenticate to Microsoft Azure. https://docs.microsoft.com/en-us/cli/azure/authenticate-azure-cli?view=azure-cli-latest

2.2. SSH KEY

If the user performing the deployment does not currently have a public and private **SSH** key perform the following.

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

2.3. CREATING A RESOURCE GROUP

Microsoft Azure resource groups contains all of the components deployed for an environment. The resource group defines where these items are deployed geographically.

For this reference environment deployment, the resource group name is **openshift** deployed in the location of "East US".

> **WARNING**
>
> Do not attempt to deploy instances in a different geographic location than the resource group resides.

```
# az group create \
    --name openshift \
    --location "East US"
```

2.4. CREATING A RED HAT ENTERPRISE LINUX BASE IMAGE

The Red Hat Enterprise Linux image from Microsoft Azure can be used for the deployment of OpenShift but the instances deployed using this image are charged more as the image carries it's own Red Hat Enterprise Linux subscription. To save cost the following link 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.

To create an image follow the steps below. https://docs.microsoft.com/en-us/azure/virtual-machines/linux/redhat-create-upload-vhd

2.5. CREATION OF RED HAT OPENSHIFT CONTAINER PLATFORM NETWORKS

A virtual network and subnet are created to allow for virtual machines to be launched. The addresses below can be modified to suit any requirements for an organization.

```
# az network vnet create \
    --name openshiftvnet \
    --resource-group openshift \
    --subnet-name ocp \
    --address-prefix 10.0.0.0/16 \
    --subnet-prefix 10.0.0.0/24
```

2.6. CREATING NETWORK SECURITY GROUPS

Microsoft Azure network security groups (**NSGs**) allows the user to define inbound and outbound traffic filters that can be applied to each instance on a network. This allows the user to limit network traffic to each instance based on the function of the instance services and not depend on host based filtering.

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

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

Table 2.1. Security Group association

Instance type	Security groups associated
Bastion	**bastion-nsg**
Masters	**master-nsg**
Infra nodes	**infra-node-nsg**
App nodes	**node-nsg**
CNS nodes(Optional)	**cns-nsg**

2.6.1. Bastion Security Group

The *bastion* instance only needs to allow inbound **ssh**. This instance exists to serve as the jump host between the private subnet and public internet.

Table 2.2. Bastion Security Group TCP ports

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

Creation of the above security group is as follows:

```
# az network nsg create \
    --resource-group openshift \
    --name bastion-nsg \
    --tags bastion_nsg

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name bastion-nsg \
    --name bastion-nsg-ssh  \
    --priority 500 \
    --destination-port-ranges 22 \
    --access Allow --protocol Tcp \
    --description "SSH access from Internet"
```

2.6.2. Master Security Group

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

Table 2.3. Master Host Security Group Ports

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

NOTE

As masters nodes are in fact nodes, the same rules used for OpenShift nodes are used.

Creation of the above security group is as follows:

```
# az network nsg create \
    --resource-group openshift \
    --name master-nsg \
    --tags master_security_group

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-ssh  \
    --priority 500 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 22 \
    --access Allow --protocol Tcp \
    --description "SSH from the bastion"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-etcd  \
    --priority 525 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 2379 2380 \
    --access Allow --protocol Tcp \
    --description "ETCD service ports"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-api \
    --priority 550 \
    --destination-port-ranges 443 \
    --access Allow --protocol Tcp \
    --description "API port"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-api-lb \
    --source-address-prefixes VirtualNetwork \
    --priority 575 \
    --destination-port-ranges 443 \
    --access Allow --protocol Tcp \
    --description "API port"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-ocp-tcp  \
    --priority 600 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 8053 \
    --access Allow --protocol Tcp \
    --description "TCP DNS and fluentd"
```

```
# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name master-ocp-udp  \
    --priority 625 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 8053 \
    --access Allow --protocol Udp \
    --description "UDP DNS and fluentd"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name node-kubelet  \
    --priority 650 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 10250 \
    --access Allow --protocol Tcp \
    --description "kubelet"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name master-nsg \
    --name node-sdn  \
    --priority 675 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 4789 \
    --access Allow --protocol Udp \
    --description "OpenShift sdn"
```

2.6.3. Infrastructure Node Security Group

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 Security Group Ports

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

Creation of the above security group is as follows:

```
# az network nsg create \
    --resource-group openshift \
    --name infra-node-nsg \
    --tags infra_security_group

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name infra-ssh  \
    --priority 500 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 22  \
    --access Allow --protocol Tcp \
    --description "SSH from the bastion"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name router-ports  \
    --priority 525 \
    --source-address-prefixes AzureLoadBalancer \
    --destination-port-ranges 80 443  \
    --access Allow --protocol Tcp \
    --description "OpenShift router"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name infra-ports  \
    --priority 550 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 9200 9300 \
    --access Allow --protocol Tcp \
    --description "ElasticSearch"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name node-kubelet  \
    --priority 575 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 10250 \
    --access Allow --protocol Tcp \
    --description "kubelet"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name node-sdn  \
    --priority 600 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 4789 \
    --access Allow --protocol Udp \
    --description "OpenShift sdn"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name infra-node-nsg \
    --name router-ports \
    --priority 625 \
    --destination-port-ranges 80 443 \
    --access Allow --protocol Tcp \
    --description "OpenShift router"
```

2.6.4. Node Security Group

The node security group is assigned to application instances. The rules defined only allow for **ssh** traffic from the *bastion* host or other nodes, pod to pod communication via SDN traffic and kubelet communication via Kubernetes.

Table 2.5. Node Security Group Ports

Port/Protocol	Services	Remote source	Purpose
22/TCP	SSH	Bastion	Secure shell login
4789/UDP	SDN	Nodes	Pod to pod communications
10250/TCP	kubernetes	Nodes	Kubelet communications
10256/TCP	Health Check	Nodes	External LB health check

Creation of the above security group is as follows:

```
# az network nsg create \
    --resource-group openshift \
    --name node-nsg \
    --tags node_security_group

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name node-nsg \
    --name node-ssh  \
    --priority 500 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 22 \
    --access Allow --protocol Tcp \
    --description "SSH from the bastion"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name node-nsg \
    --name node-kubelet \
    --priority 525 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 10250 \
    --access Allow --protocol Tcp \
    --description "kubelet"
```

```
# az network nsg rule create \
    --resource-group openshift \
    --nsg-name node-nsg \
    --name node-sdn  \
    --priority 550 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 4789 --access Allow \
    --protocol Udp \
    --description "ElasticSearch and ocp apps"
```

```
# az network nsg rule create \
    --resource-group openshift \
    --nsg-name node-nsg \
    --name node-sdn  \
    --priority 575 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 10256 --access Allow \
    --protocol Tcp \
    --description "Load Balancer health check"
```

2.6.5. CNS Node Security Group (Optional)

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

Table 2.6. CNS Security Group Ports

Port/Protocol	Services	Remote source	Purpose
22/TCP	SSH	Bastion	Secure shell login
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
4789/UDP	SDN	Nodes	Pod to pod communications
10250/TCP	kubernetes	Nodes	Kubelet communications
24007/TCP	Gluster	Gluster Nodes	Gluster Daemon
24008/TCP	Gluster	Gluster Nodes	Gluster Management
24010/TCP	Gluster	Gluster Nodes	Gluster Block

Port/Protocol	Services	Remote source	Purpose
49152-49664/TCP	Gluster	Gluster Nodes	Gluster Client Ports

Creation of the above security group is as follows:

```
# az network nsg create \
    --resource-group openshift \
    --name cns-nsg \
    --tags node_security_group

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name cns-nsg \
    --name node-ssh  \
    --priority 500 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 22 \
    --access Allow --protocol Tcp \
    --description "SSH from the bastion"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name cns-nsg \
    --name node-kubelet \
    --priority 525 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 10250 \
    --access Allow --protocol Tcp \
    --description "kubelet"

# az network nsg rule create \
    --resource-group openshift \
    --nsg-name cns-nsg \
    --name node-sdn  \
    --priority 550 \
    --source-address-prefixes VirtualNetwork \
    --destination-port-ranges 4789 --access Allow \
    --protocol Udp \
    --description "ElasticSearch and ocp apps"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-ssh  \
      --priority 575 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges 2222 \
      --access Allow --protocol Tcp \
      --description "Gluster SSH"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-daemon  \
      --priority 600 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges 24008 \
      --access Allow --protocol Tcp \
      --description "Gluster Daemon"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-mgmt  \
      --priority 625 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges 24009 \
      --access Allow --protocol Tcp \
      --description "Gluster Management"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-client  \
      --priority 650 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges  49152-49664 \
      --access Allow --protocol Tcp \
      --description "Gluster Clients"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name portmap-tcp  \
      --priority 675 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges  111 \
      --access Allow --protocol Tcp \
      --description "Portmap tcp"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name portmap-udp  \
      --priority 700 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges  111 \
      --access Allow --protocol Udp \
      --description "Portmap udp"

# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-iscsi  \
      --priority 725 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges  3260 \
      --access Allow --protocol Tcp \
      --description "Gluster Blocks"
```

```
# az network nsg rule create \
      --resource-group openshift \
      --nsg-name cns-nsg \
      --name gluster-block  \
      --priority 750 \
      --source-address-prefixes VirtualNetwork \
      --destination-port-ranges  24010 \
      --access Allow --protocol Tcp \
      --description "Gluster Block"
```

2.7. OPENSHIFT LOAD BALANCERS

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

2.7.1. Master Load Balancer

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

The first step when working with load balancers on Microsoft Azure is to request a static IP address to be used for the load balancer.

```
# az network public-ip create \
    --resource-group openshift \
    --name masterExternalLB \
    --allocation-method Static
```

Using the static IP address create a load balancer to be used by the master instances.

```
# az network lb create \
    --resource-group openshift \
    --name OcpMasterLB \
    --public-ip-address masterExternalLB \
    --frontend-ip-name masterApiFrontend \
    --backend-pool-name masterAPIBackend
```

The load balancer uses probes to validate that instances in the backend pools are available. The probe will attempt to connect to TCP port 443 which is used by the OpenShift API.

```
# az network lb probe create \
    --resource-group openshift \
    --lb-name OcpMasterLB \
    --name masterHealthProbe \
    --protocol tcp \
    --port 443
```

Lastly, create the load balancing rule. This rule will allow the load balanacer to accept port 443 and forward the requests to the Red Hat OpenShift Container Platform Master API and web console containers. The **`load-distribution`** variable ensures that client connections will redirect to the same backend server as the initial connection occured.

```
# az network lb rule create \
    --resource-group openshift \
    --lb-name OcpMasterLB \
    --name ocpApiHealth \
    --protocol tcp --frontend-port 443 \
    --backend-port 443 \
    --frontend-ip-name masterApiFrontend \
    --backend-pool-name masterAPIBackend \
    --probe-name masterHealthProbe \
    --load-distribution SourceIPProtocol
```

2.7.2. Router Load Balancer

The Router load balancer is used by the infrastructure instances hosting the router pods. This load balancer requires the use of a static public IP address. The static ip address is used for the OpenShift subdomain (*.apps.example.com) for the routing to application containers.

The first step when working with load balancers on Microsoft Azure is to request a static IP address to be used for the load balancer.

```
# az network public-ip create \
    --resource-group openshift \
    --name routerExternalLB \
    --allocation-method Static
```

Using the static IP address create a load balancer to be used by the infrastructure instances.

```
# az network lb create \
    --resource-group openshift \
    --name OcpRouterLB \
    --public-ip-address routerExternalLB \
    --frontend-ip-name routerFrontEnd \
    --backend-pool-name routerBackEnd
```

The load balancer uses probes to validate that instances in the backend pools are available. The probe will attempt to connect to TCP port 80.

```
# az network lb probe create \
    --resource-group openshift \
    --lb-name OcpRouterLB \
    --name routerHealthProbe \
    --protocol tcp \
    --port 80
```

The final step for configuring the load balancer is to create rules for routing. The rules created allow the load balancer to listen on ports 80 and 443 and pass traffic to the infrastructure instances hosting the HAProxy containers for routing. For simplicity, the probe **`routerHealthProbe`** is used for both rule definitions because the HAProxy container serves both 80 and 443 so if the service is not available for port 80 then the service is not available for 443 as well.

```
# az network lb rule create \
    --resource-group openshift \
    --lb-name OcpRouterLB \
    --name routerRule \
    --protocol tcp \
    --frontend-port 80 \
    --backend-port 80 \
    --frontend-ip-name routerFrontEnd \
    --backend-pool-name routerBackEnd \
    --probe-name routerHealthProbe \
    --load-distribution SourceIPProtocol

# az network lb rule create \
    --resource-group openshift \
    --lb-name OcpRouterLB \
    --name httpsRouterRule \
    --protocol tcp \
    --frontend-port 443 \
    --backend-port 443 \
    --frontend-ip-name routerFrontEnd \
    --backend-pool-name routerBackEnd \
    --probe-name routerHealthProbe \
    --load-distribution SourceIPProtocol
```

2.8. CREATING INSTANCES FOR RED HAT OPENSHIFT CONTAINER PLATFORM

This reference environment consists of the following instances:

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

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

NOTE

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.

2.8.1. Availability Sets

Availability sets are created for masters, infrastructure, and application instances. The availability sets are used in Red Hat OpenShift Container Platform to define which nodes should be part of load balancer OpenShift services.

```
# az vm availability-set create \
    --resource-group openshift \
    --name ocp-app-instances
```

```
# az vm availability-set create \
    --resource-group openshift \
    --name ocp-infra-instances

# az vm availability-set create \
    --resource-group openshift \
    --name ocp-master-instances
```

2.8.2. Master Instance Creation

The bash command lines runs a for loop that allow for all of the master instances to be created with their respective specific mounts, instance size, and network security groups configured at launch time.

IMPORTANT

Master instances should have a minimum of 4vCPUs and 16GB of ram. This reference environment uses Standard Disks and specific sizes of S4 managed disks for more information visit https://docs.microsoft.com/en-us/azure/virtual-machines/windows/about-disks-and-vhds.

```
# for i in 1 2 3; do \
    az network nic create \
    --resource-group openshift \
    --name ocp-master-${i}VMNic \
    --vnet-name openshiftvnet \
    --subnet ocp \
    --network-security-group master-nsg \
    --lb-name OcpMasterLB \
    --lb-address-pools masterAPIBackend \
    --internal-dns-name ocp-master-${i} \
    --public-ip-address ""; \
  done

# for i in 1 2 3; do \
    az vm create \
    --resource-group openshift \
    --name ocp-master-$i \
    --availability-set ocp-master-instances \
    --size Standard_D4s_v3 \
    --image RedHat:RHEL:7-RAW:latest \
    --admin-user cloud-user \
    --ssh-key /root/.ssh/id_rsa.pub \
    --data-disk-sizes-gb 32 \
    --nics ocp-master-${i}VMNic; \
  done

# for i in 1 2 3; do \
    az vm disk \
    attach --resource-group openshift \
    --vm-name ocp-master-$i \
    --disk ocp-master-container-$i \
    --new --size-gb 32; \
  done

# for i in 1 2 3; do \
    az vm disk \
    attach --resource-group openshift \
    --vm-name ocp-master-$i \
    --disk ocp-master-etcd-$i \
    --new --size-gb 32; \
  done
```

2.8.3. Infrastructure Instance Creation

The bash command lines runs a for loop that allow for all of the infrastructure instances to be created with their respective specific mounts, instance size, and security groups configured at launch time.

Infrastructure instances should have a minimum of 1vCPUs and 8GB of RAM, but if logging and metrics are deployed, a larger instances should be created. Below the instance size of 4vCPUs and 16GB of RAM are used to ensure resource requirements are met to host the **EFK** pods. This reference environment uses Standard Disks and specific sizes of S4 and S6 managed disks for more information visit https://docs.microsoft.com/en-us/azure/virtual-machines/windows/about-disks-and-vhds.

```
# for i in 1 2 3; do \
    az network nic create \
    --resource-group openshift \
    --name ocp-infra-${i}VMNic \
    --vnet-name openshiftvnet \
    --subnet ocp \
    --network-security-group infra-node-nsg \
    --lb-name ocpRouterLB \
    --lb-address-pools routerBackend \
    --internal-dns-name ocp-infra-$i \
    --public-ip-address ""; \
  done

# for i in 1 2 3; do \
    az vm create \
    --resource-group openshift \
    --name ocp-infra-$i \
    --availability-set ocp-infra-instances \
    --size Standard_D4s_v3 \
    --image RedHat:RHEL:7-RAW:latest \
    --admin-user cloud-user \
    --ssh-key /root/.ssh/id_rsa.pub \
    --data-disk-sizes-gb 32 \
    --nics ocp-infra-${i}VMNic; \
  done

# for i in 1 2 3; do \
    az vm disk attach \
    --resource-group openshift \
    --vm-name ocp-infra-$i \
    --disk ocp-infra-container-$i \
    --new --size-gb 64; \
  done
```

2.8.4. Application Instance Creation

The bash command lines runs a for loop that allow for all of the application instances to be created with their respective mounts, instance size, and security groups configured at launch time.

Application instances should have a minimum of 1vCPUs and 8GB of RAM. Instances can be resized but resizing requires a reboot. Application instances can be added after the installation. The instances deployed below have 2vCPUs and 8GB of RAM. This reference environment uses Standard Disks and specific sizes of S4 and S6 managed disks for more information visit https://docs.microsoft.com/en-us/azure/virtual-machines/windows/about-disks-and-vhds.

NOTE

If there is a desire to deploy more than three nodes modify the for loop below to reflect the desired number of instances.

```
# for i in 1 2 3; do \
    az network nic create \
    --resource-group openshift \
    --name ocp-app-${i}VMNic \
    --vnet-name openshiftvnet \
    --subnet ocp \
    --network-security-group node-nsg \
    --internal-dns-name ocp-app-$i \
    --public-ip-address ""; \
  done

# for i in 1 2 3; do \
    az vm create \
    --resource-group openshift \
    --name ocp-app-$i \
    --availability-set ocp-app-instances \
    --size Standard_D2S_v3 \
    --image RedHat:RHEL:7-RAW:latest \
    --admin-user cloud-user \
    --ssh-key /root/.ssh/id_rsa.pub \
    --data-disk-sizes-gb 64 \
    --nics ocp-app-${i}VMNic; \
  done

# for i in 1 2 3; do \
    az vm disk attach \
    --resource-group openshift \
    --vm-name ocp-app-$i \
    --disk ocp-app-container-$i \
    --new --size-gb 64; \
  done
```

2.8.5. CNS Instance Creation (Optional)

The bash command lines runs a for loop that allows for all of the CNS instances to be created with their respective mounts, instance size, and network security groups configured at launch time.

CNS instances should have a minimum of 4vCPUs and 32GB of RAM. These instances by default only schedule the **glusterfs** pods. This reference environment uses Standard Disks and specific sizes of S4, S6, and S20 managed disks for more information visit https://docs.microsoft.com/en-us/azure/virtual-machines/windows/about-disks-and-vhds.

First, create the availability set to be used for the CNS instances.

```
# az vm availability-set create \
    --resource-group openshift \
    --name ocp-cns-instances
```

Create the instances using the **`cns-nsg`** and the **`ocp-cns-instances`** availability set.

```
# for i in 1 2 3; do \
    az network nic create
    --resource-group openshift \
    --name ocp-cns-${i}VMNic \
    --vnet-name openshiftvnet \
    --subnet ocp \
    --network-security-group cns-nsg \
    --internal-dns-name ocp-cns-$i \
    --public-ip-address ""; \
  done

# for i in 1 2 3; do \
  az vm create \
    --resource-group openshift \
    --name ocp-cns-$i \
    --availability-set ocp-cns-instances \
    --size Standard_D8s_v3 \
    --image RedHat:RHEL:7-RAW:latest \
    --admin-user cloud-user \
    --ssh-key /root/.ssh/id_rsa.pub \
    --data-disk-sizes-gb 32 \
    --nics ocp-cns-${i}VMNic; \
  done

# for i in 1 2 3; do \
    az vm disk attach \
    --resource-group openshift \
    --vm-name ocp-cns-$i \
    --disk ocp-cns-container-$i \
    --new --size-gb 64; \
  done

# for i in 1 2 3; do \
    az vm disk attach \
    --resource-group openshift \
    --vm-name ocp-cns-$i \
    --disk ocp-cns-volume-$i \
    --sku Premium_LRS \
    --new --size-gb 512; \
  done
```

2.8.6. Deploying the Bastion Instance

The bastion instance allows for limited access **`ssh`** access from one network into another. This requires the bastion instance to have a static IP address.

The bastion instance is not resource intensive. An instance size with a small amount of vCPUs and ram may be used and recommended.

```
# az network public-ip create \
    --name bastion-static \
    --resource-group openshift \
    --allocation-method Static
```

```
# az network nic create \
    --name bastion-VMNic \
    --resource-group openshift \
    --subnet ocp \
    --vnet-name openshiftvnet \
    --network-security-group bastion-nsg \
    --public-ip-address bastion-static
```

Finally, launch the instance using the newly provisioned network interface.

```
# az vm create \
    --resource-group openshift \
    --name bastion --size Standard_D1 \
    --image RedHat:RHEL:7-RAW:latest \
    --admin-user cloud-user \
    --ssh-key /root/.ssh/id_rsa.pub \
    --nics bastion-VMNic
```

2.8.7. DNS

Three DNS records are set for the environment described in this document. The document assumes that a DNS zone has been created in Azure already and the **`nameservers`** are already configured to use this zone.

2.8.8. Master Load Balancer Record

The static IP address that was created for the master load balancer requires an *A record set* to allow for mapping to the OpenShift master console.

```
# az network public-ip show \
    --resource-group openshift \
    --name masterExternalLB \
    --query "{address: ipAddress}" \
    --output tsv

40.70.56.154

# az network dns record-set \
    a add-record \
    --resource-group openshift \
    --zone-name example.com \
    --record-set-name openshift-master \
    --ipv4-address 40.70.56.154
```

2.8.9. Router Load Balancer Record

The static IP address that was created for the Router load balancer requires an *A record*. This DNS entry is used to route to applications containers in the OpenShift environment through the HAProxy containers running on the infrastucture instances.

```
# az network public-ip show \
    --resource-group openshift \
    --name RouterExternalLB \
    --query "{address: ipAddress}" \
    --output tsv

52.167.228.197

# az network dns record-set \
    a add-record \
    --resource-group openshift \
    --zone-name example.com \
    --record-set-name *.apps \
    --ipv4-address 52.167.228.197
```

2.8.10. Bastion Record

The final DNS record that needs to be created is for the bastion instance. This DNS record is used to map the static IP address to *bastion.example.com*

```
# az network public-ip show \
    --resource-group openshift \
    --name bastion-static \
  --query "{address: ipAddress}" \
  --output tsv

52.179.166.233


# az network dns record-set \
  a add-record \
  --resource-group openshift \
  --zone-name example.com \
  --record-set-name bastion \
  --ipv4-address 13.82.214.23
```

2.9. BASTION CONFIGURATION 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.

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

To easily connect to the Red Hat OpenShift Container Platform environment, the following snippet can be added to the **`~/.ssh/config`** file in the host used to administer the Red Hat OpenShift Container Platform platform:

```
Host bastion
```

```
    HostName        bastion.example.com
    User            <cloud-user>
    IdentityFile    /path/to/<keypair-name>.pem

Host ocp*
    ProxyCommand    ssh cloud-user@bastion -W %h:%p
    IdentityFile    /path/to/<keypair-name>.pem
    User            <cloud-user>
```

As an example, to access one of the master instances, this can be used:

```
# ssh master1
```

2.10. AZURE SERVICE PRINCIPAL

A service principal is used to allow for the creation and management of Kubernetes service load balancers and disks for persistent storage. The service principal values are then added to **/etc/origin/cloudprovider/azure.conf** to be used for the OpenShift masters and nodes.

The first step is to obtain the subscription id.

```
# az account list
[
{
  "cloudName": "AzureCloud",
  "id": "8227d1d9-c10c-4366-86cc-e3ddbbcbba1d",
  "isDefault": false,
  "name": "Pay-As-You-Go",
  "state": "Enabled",
  "tenantId": "82d3ef91-24fd-4deb-ade5-e96dfd00535e",
  "user": {
    "name": "admin@example.com",
    "type": "user"
  }
]
```

Once the subscription id is obtained, create the service principal with the role of contributor in the openshift Microsoft Azure resource group. Record the output of these values to be used in future steps when defining the cloud provider values.

```
# az ad sp create-for-rbac --name openshiftcloudprovider \
     --password Cl0udpr0vid2rs3cr3t --role contributor \
     --scopes /subscriptions/8227d1d9-c10c-4366-86cc-
e3ddbbcbba1d/resourceGroups/openshift

Retrying role assignment creation: 1/36
Retrying role assignment creation: 2/36
{
  "appId": "17b0c26b-41ff-4649-befd-a38f3eec2768",
  "displayName": "ocpcloudprovider",
  "name": "http://ocpcloudprovider",
  "password": "$3r3tR3gistry",
  "tenant": "82d3ef91-24fd-4deb-ade5-e96dfd00535e"
}
```

2.11. BLOB REGISTRY STORAGE

Microsoft Azure blob storage is used to allow for the storing of container images. The Red Hat OpenShift Container Platform registry uses blob storage to allow for the registry to grow dynamically in size without the need for intervention from an Administrator. An account must be created to allowing for the storage to be used.

```
# az storage account create \
    --name openshiftregistry \
    --resource-group openshift \
    --location eastus \
    --sku Standard_LRS
```

2.12. OPENSHIFT 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 OpenShift.

NOTE

The following tasks should be performed on the workstation that was used to provision the infrastructure. The workstation can be an virtual machine, workstation, vagrant vm, or instance running in the cloud. The most important requirement is that the system is running RHEL 7.

2.12.1. Ansible Setup

Install the following packages on the system performing the installation of OpenShift.

```
$ subscription-manager repos \
    --enable="rhel-7-server-rpms" \
    --enable="rhel-7-server-extras-rpms" \
    --enable="rhel-7-server-ose-3.9-rpms" \
    --enable="rhel-7-fast-datapath-rpms" \
    --enable="rhel-7-server-ansible-2.4-rpms"

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

2.12.2. 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.

2.12.3. Azure Account Key

Using the blob storage account created in the earlier steps, query Microsoft Azure to get the key for the **`openshiftregistry`** account. This key is used in the inventory to setup the image registry.

```
# az storage account keys list \
    --account-name openshiftregistry \
    --resource-group openshift \
    --output table

KeyName    Permissions    Value
key1       Full
QgmccEiitO7F1ZSYGDdpwe4HtzCrGCKvMi1vyCqFjxBcJnOliKYiVsez1jol1qUh74P75KInnX
x78gFDuz6obQ==
key2       Full
liApA26Y3GVRllvrVyFx51xF3MEuFKlsSGDxBXk4JERNsxu6juvcPWO/fNNiG2O2Z++ATlxhJ+
nSPXtU4Zs5xA==
```

Record the value of the first key to be used in the inventory file created in the next section for the variable **`openshift_hosted_registry_storage_azure_blob_accountkey`**.

2.12.4. Preparing the Inventory File

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

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

2.12.4.1. Cloud Provider Configuration

Based off of the values defined in the above steps. The installation requires certain values to be defined to support the Azure cloud provider.

Table 2.7. Cloud Provider Values

Key	Value
openshift_cloudprovider_azure_tenant_id	The value of tenant from the output of the service principal creation
openshift_cloudprovider_azure_subscription_id	{provder} subscription
openshift_cloudprovider_azure_client_id	The alphanumeric value from the output of the service principal creation(appID)
openshift_cloudprovider_azure_client_secret	Password provided at service principal creation(password)

Key	Value
openshift_cloudprovider_azure_resource_group	The resource group containing the Microsoft Azure components
cloud	AzurePublicCloud AzureUSGovernmentCloud AzureChinaCloud AzureGermanCloud
openshift_cloudprovider_azure_location	Microsoft Azure geographic location

These entries allow for the creation of load balaners but these values are optional to the installation.

Table 2.8. Optional Cloud Provider values

openshift_cloudprovider_azure_security_group_name	Security group name used by nodes
openshift_cloudprovider_azure_vnet_name	Virtual network where the VMs run
openshift_cloudprovider_azure_availability_set_name	The availability set in which the nodes reside

2.12.4.2. Creating the Installation Inventory

The file **`/etc/ansible/hosts`** will be used to define all of the configuration items in which the OpenShift installer will configure to allow for the OpenShift services to run. Below is an example inventory based on the values defined in previous steps.

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

[OSEv3:vars]
ansible_ssh_user=cloud-user
ansible_become=true
openshift_cloudprovider_kind=azure
osm_controller_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/origin/cloudprovider/azure.conf']}
osm_api_server_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/origin/cloudprovider/azure.conf']}
openshift_node_kubelet_args={'cloud-provider': ['azure'], 'cloud-config':
['/etc/origin/cloudprovider/azure.conf'], 'enable-controller-attach-
detach': ['true']}
openshift_master_api_port=443
openshift_master_console_port=443
openshift_hosted_router_replicas=3
openshift_hosted_registry_replicas=1
openshift_master_cluster_method=native
openshift_master_cluster_hostname=openshift-master.example.com
openshift_master_cluster_public_hostname=openshift-master.example.com
```

```
openshift_master_default_subdomain=apps.example.com
deployment_type=openshift-enterprise
#cloudprovider
openshift_cloudprovider_kind=azure
openshift_cloudprovider_azure_client_id=17b0c26b-41ff-4649-befd-
a38f3eec2768
openshift_cloudprovider_azure_client_secret=Cl0udpr0vid2rs3cr3t
openshift_cloudprovider_azure_tenant_id=422r3f91-21fe-4esb-vad5-
d96dfeooee5d
openshift_cloudprovider_azure_subscription_id=8227d1d9-c10c-4366-86cc-
e3ddbbcbba1d
openshift_cloudprovider_azure_resource_group=openshift
openshift_cloudprovider_azure_location=eastus
#endcloudprovider
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'}]
openshift_node_local_quota_per_fsgroup=512Mi
networkPluginName=redhat/ovs-networkpolicy
oreg_url_master=registry.access.redhat.com/openshift3/ose-
${component}:${version}
oreg_url_node=registry.access.redhat.com/openshift3/ose-
${component}:${version}
openshift_examples_modify_imagestreams=true
oreg_url=registry.access.redhat.com/openshift3/ose-${component}:${version}
openshift_storage_glusterfs_image=registry.access.redhat.com/rhgs3/rhgs-
server-rhel7
openshift_storage_glusterfs_heketi_image=registry.access.redhat.com/rhgs3/
rhgs-volmanager-rhel7

# Do not uninstall service catalog until post installation. Needs storage
class object
openshift_enable_service_catalog=false

# Setup azure blob registry storage
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=azure_blob
openshift_hosted_registry_storage_azure_blob_accountname=openshiftregistry
openshift_hosted_registry_storage_azure_blob_accountkey=QgmccEiitO7F1ZSYGD
dpwe4HtzCrGCKvMi1vyCqFjxBcJnOliKYiVsez1jol1qUh74P75KInnXx78gFDuz6obQ==
openshift_hosted_registry_storage_azure_blob_container=registry
openshift_hosted_registry_storage_azure_blob_realm=core.windows.net


[masters]
ocp-master-1
ocp-master-2
ocp-master-3

[etcd]
ocp-master-1
ocp-master-2
ocp-master-3
```

```
[nodes]
ocp-master-1 openshift_node_labels="{'region': 'master'}"
openshift_hostname=ocp-master-1
ocp-master-2 openshift_node_labels="{'region': 'master'}"
openshift_hostname=ocp-master-2
ocp-master-3 openshift_node_labels="{'region': 'master'}"
openshift_hostname=ocp-master-3
ocp-infra-1 openshift_node_labels="{'region': 'infra'}"
openshift_hostname=ocp-infra-1
ocp-infra-2 openshift_node_labels="{'region': 'infra'}"
openshift_hostname=ocp-infra-2
ocp-infra-3 openshift_node_labels="{'region': 'infra'}"
openshift_hostname=ocp-infra-3
ocp-app-1 openshift_node_labels="{'region': 'apps'}"
openshift_hostname=ocp-app-1
ocp-app-2 openshift_node_labels="{'region': 'apps'}"
openshift_hostname=ocp-app-2
ocp-app-3 openshift_node_labels="{'region': 'apps'}"
openshift_hostname=ocp-app-3
```

2.12.4.3. CNS Inventory (Optional)

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

```
# vi /etc/ansible/hosts
[OSEv3:children]
masters
etcd
nodes
glusterfs

....omitted...

[nodes]
....omitted...
ocp-cns-1 openshift_schedulable=True openshift_hostname=ocp-cns-1
ocp-cns-2 openshift_schedulable=True openshift_hostname=ocp-cns-2
ocp-cns-3 openshift_schedulable=True openshift_hostname=ocp-cns-3

[glusterfs]
ocp-cns-1 glusterfs_devices='[ "/dev/sde" ]'
ocp-cns-2 glusterfs_devices='[ "/dev/sde" ]'
ocp-cns-3 glusterfs_devices='[ "/dev/sde" ]'
```

2.12.5. Node Registration

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

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

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

OR

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

2.12.6. Repository Setup

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

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

2.12.7. EmptyDir Storage

During the deployment of instances, an extra volume was added to the instances for **`EmptyDir`** storage. **`EmptyDir`** storage is storage that is used by containers that are not persistent. The volume is used to help ensure that the */var* volume does not get filled by containers using the storage.

The ad-hoc plays creates a filesystem on the disk, add an entry to fstab, and then mounts the volume.

```
# ansible nodes -b -m filesystem -a "fstype=xfs dev=/dev/sdc"

# ansible nodes -b -m file -a \
    "path=/var/lib/origin/openshift.local.volumes \
    state=directory mode=0755"

# ansible nodes -b -m mount -a \
    "path=/var/lib/origin/openshift.local.volumes \
    src=/dev/sdc state=present \
    fstype=xfs opts=gquota"

# ansible nodes -b -m shell -a \
    "restorecon -R \
    /var/lib/origin/openshift.local.volumes"

# ansible nodes -b -m mount -a \
    "path=/var/lib/origin/openshift.local.volumes \
    src=/dev/sdc state=mounted fstype=xfs opts=gquota"
```

2.12.8. `etcd` Storage

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

The ad-hoc plays creates a filesystem on the disk, add an entry to fstab, and then mounts the volume.

```
# ansible masters -b -m filesystem -a "fstype=xfs dev=/dev/sde"

# ansible masters -b -m file -a \
    "path=/var/lib/etcd \
    state=directory mode=0755"

# ansible masters -b -m mount -a \
    "path=/var/lib/etcd \
    src=/dev/sde state=present \
    fstype=xfs"

# ansible masters -b -m shell -a \
    "restorecon -R \
    /var/lib/etcd"

# ansible masters -b -m mount -a \
    "path=/var/lib/etcd \
    src=/dev/sde state=mounted fstype=xfs"
```

2.12.9. Container Storage

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

```
# ansible-playbook -e \
    'container_runtime_docker_storage_setup_device=/dev/sdd' \
    -e 'container_runtime_docker_storage_type=overlay2' \
    /usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
```

NOTE

/dev/sdd reflects the disk created for container storage.

CHAPTER 3. DEPLOYING RED HAT OPENSHIFT CONTAINER PLATFORM

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

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

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

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

```
PLAY RECAP
**************************************************************************
**************************************************************************
****************************************************
ocp-app-1 : ok=233  changed=40   unreachable=0    failed=0
ocp-app-2 : ok=233  changed=40   unreachable=0    failed=0
ocp-app-3 : ok=233  changed=40   unreachable=0    failed=0
ocp-infra-1 : ok=233  changed=40   unreachable=0    failed=0
ocp-infra-2 : ok=233  changed=40   unreachable=0    failed=0
ocp-infra-3 : ok=233  changed=40   unreachable=0    failed=0
localhost   : ok=12   changed=0    unreachable=0    failed=0
ocp-master-1 : ok=674  changed=161  unreachable=0    failed=0
ocp-master-2 : ok=442  changed=103  unreachable=0    failed=0
ocp-master-3 : ok=442  changed=103  unreachable=0    failed=0
```

3.1. STORAGE CLASS DEPLOYMENT

A Red Hat OpenShift Container Platform **storageclass** object is required for persistent storage for applications and logging if CNS is not deployed. A Red Hat OpenShift Container Platform deployment can have many **storageclass** objects but only one can be the default.

Create the **yaml** file defining the Microsoft Azure storage to be used.

```
# vi storageclass.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: managed
```

Log into one of the master instances and use the **oc** client to create the default **storageclass** object.

```
# oc create -f storageclass.yaml
```

3.2. LOGGING

Return to the host that performed the deployment of OpenShift. Now that a persistent storage exists Red Hat OpenShift Container Platform logging can be deployed to store application container logs.

```
# vi /etc/ansible/hosts
...omitted...
openshift_logging_install_logging=true
openshift_logging_es_pvc_size=100Gi
openshift_logging_es_pvc_dynamic=True
openshift_logging_es_cluster_size=3
openshift_logging_es_nodeselector={"region": "infra"}
openshift_logging_kibana_nodeselector={"region": "infra"}
openshift_logging_curator_nodeselector={"region": "infra"}
```

Run the Ansible playbook below to deploy the logging components.

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-logging/config.yml
```

3.3. METRICS

Red Hat OpenShift Container Platform metrics requires persistent storage. Modify the inventory to add in the variables for the installation of OpenShift metrics.

```
# vi /etc/ansible/hosts
...omitted...
openshift_metrics_install_metrics=true
openshift_metrics_cassandra_pvc_size=20Gi
openshift_metrics_cassandra_storage_type=dynamic
openshift_metrics_cassandra_replicas="1"
openshift_metrics_hawkular_nodeselector={"region":"infra"}
openshift_metrics_cassandra_nodeselector={"region":"infra"}
openshift_metrics_heapster_nodeselector={"region":"infra"}
```

Run the Ansible playbook below to deploy the metrics components.

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-metrics/config.yml
```

3.4. SERVICE CATALOG

The service catalog also requires persistent storage to be used for an **`etcd`** pod that runs within the Red Hat OpenShift Container Platform project. Modify the inventory to change the value of **`openshift_enable_service_catalog`** from false to true.

```
# vi /etc/ansible/hosts
...omitted...
#openshift_enable_service_catalog=false
openshift_enable_service_catalog=true
```

Run the Ansible playbook below to deploy the service catalog components.

```
# ansible-playbook /usr/share/ansible/openshift-
ansible/playbooks/openshift-service-catalog/config.yml
```

3.5. 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.

3.5.1. 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
```

3.5.2. 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

3.5.3. Adding Microsoft Azure to Cloudforms

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

https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.6/html/managing_providers/cloud_providers#azure_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 Microsoft Azure environments with the following tasks:

- Deciding between different internal network technologies
- Provisioning instances within Microsoft Azure 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.