Red Hat Consulting

YOU Telecom
OpenShift Container Platform
HLD document

Version 0.2

Preface

Confidentiality, Copyright, and Disclaimer

This document is a customer-facing document between Red Hat, Inc., Computing ERA, and YOU Telecom.

Except as is required to share this information as provided with the confidential parties.

This document is not a quote and does not include any binding commitments by Red Hat.
Upon request, Red Hat can issue a formal quotation, including the scope of work, cost, and any customer requirements, as necessary.

Trademarks

Trademarked names may appear throughout this document. We use these names only for editorial purposes and the trademark owner's benefit, with no intention of infringing upon that trademark.

Audience

Red Hat intends this document for the Client technical staff responsible for the YOU Telecom cloud environment. This document is confidential between Red Hat, Inc. Computing ERA and YOU Telecom to provide the High-Level Design for deploying the OpenShift Container Platform.

Additional Background and Related Documents

This document does not contain step-by-step Installation details or other tasks that the relevant documentation from http://access.redhat.com/ covers. When necessary, we provide links to the appropriate records.

This document captures the output from the requirements gathering in sessions hosted with the Project Team, Computing ERA and YOU Telecom, where Computing ERA exposed the requirements and needs; it uses several aspects that are in the document named " YOU Telecom OpenShift Container Platform installation’s preparation".

The scope of this document is to describe the high-level architecture that Computing ERA offers as per the Red Hat best practices and that enables the deployment and the support by leveraging the embedded automation provided by the solutions offered.

The target is to design and implement the OpenShift Container Platform on bare-metal servers while offering CNI/CSI for Kubernetes OCP-hosted clusters.

About Computing ERA

Computing ERA was formed in 2012 as an IT, implementing technology services and providing consultancy for their clients in many infrastructures and technologies.

The organisation has become an implementer company for digital transformation at scale, data-engineering platforms and solutions. Its mission is to implement, configure and integrate what Computing ERA believes will work for its customers.

About YOU Telecom

Yemen Oman United (YOU) telecommunication company was launched on 7th March 2022 as a new brand for the previous brand, MTN Yemen.

MTN Yemen started its operation in Yemen in 2006 to 2022. Then, Spacetel Yemen launched its local mobile service in February 2001. It has enjoyed a global reputation for providing GSM mobile services based on a rapidly expanding network and continuous improvement of coverage quality.

Version History

VERSION	DATE	CONTRIBUTOR	ROLE	DESCRIPTION
0.1	08/09/2023	Sid Ramdane	Senior Telco Architect	Initial document
0.2	15/09/2023	Sid Ramdane	Senior Telco Architect	Minor changes: spelling, table of content and format

Glossary of terms for OCP installation

This glossary defines standard terms used in the OCP installation context that help you understand installation and preparation of installation effectively.

- **Assisted Installer** – An installer hosted at console.redhat.com that provides a web user interface or a RESTful API for creating a cluster configuration. The Assisted Installer generates a discovery image. Cluster machines boot with the discovery image, which installs RHCOS and an agent. The Assisted Installer and agent provide pre-installation validation and installation for the cluster.
- **Agent-based installer** – An installer similar to the Assisted Installer, but you must download the agent-based installer first. The agent-based installer is ideal for air-gapped/restricted networks.
- Bootstrap node – A temporary machine that runs a minimal Kubernetes configuration to deploy the OpenShift Container Platform control plane.
- **Control plane** – A container orchestration layer that exposes the API and interfaces to define, deploy, and manage the lifecycle of containers. They are also known as control plane machines.
- **Compute node** – Nodes that are responsible for executing workloads for cluster users. They are also known as worker nodes.
- **Disconnected installation** – Some parts of a data centre might not have access to the internet, even via proxy servers. You can still install the OpenShift Container Platform in these environments, but you must download the required software and images and make them available to the disconnected environment.
- **The OpenShift Container Platform installation program** – is a program that provisions the infrastructure and deploys a cluster.
- Installer-provisioned infrastructure – The installation program deploys and configures the cluster's infrastructure.
- **Ignition config files** – This is a file that Ignition uses to configure Red Hat Enterprise Linux CoreOS (RHCOS) during operating system initialisation. The installation program generates different Ignition config files to initialise the bootstrap, control plane, and worker nodes.
- **Kubernetes manifests** – Specifications of a Kubernetes API object in a JSON or YAML format. A configuration file can include deployments, config maps, secrets, daemonsets, etc.
- **Kubelet** – This **primary node** agent runs on each node in the cluster to ensure containers run in a pod.
- **Load balancers** – A load balancer serves as clients' single point of contact. Load balancers for the API distribute incoming traffic across control plane nodes.
- **Machine Config Operator** – An Operator that manages and applies configuration and updates of the base operating system and container runtime, including everything between the kernel and kubelet for the nodes in the cluster.

- **Operators** – The preferred method of packaging, deploying, and managing a Kubernetes application in an OpenShift Container Platform cluster. An operator encodes human operational knowledge into software quickly packaged and shared with customers.
- **User-provisioned infrastructure** – You can install the OpenShift Container Platform on the provided infrastructure and use the installation program to generate the assets required to provision the cluster infrastructure, create the cluster infrastructure, and then deploy the cluster to the provided infrastructure.

Table of Contents

Project Overview

YOU Telecom is looking to improve its private cloud business to provide business end users with the capability of provisioning business resilient workloads, as required, through the adoption of OpenShift virtualisation, OpenShift for application and ansible to innovate and containerise their applications.

In this context, Computing ERA has requested Red Hat to provide Professional Services around the installation and configuration of the OpenShift components., using a solution-oriented approach, whereby the design and deliverables are tailored to the requirements of YOU Telecom.

The service includes an assessment of YOU Telecom's current environment, architectural design, servers, network, and the storage configuration required for the deployment.

The goal is to assist with architecture and implementation for an on-premises OpenShift Platform that enforces Red Hat best practices as a base for the design described in this document.

Scope of Work

Red Hat will engage in a time and material model where YOU Telecom will consume an allocated number of weeks to design and deploy Red Hat products that realise YOU Telecom's needs.

During this engagement, Red Hat assists the Client with the following tasks list:

Architecture design

Red Hat conducts a design workshop to adopt the Customer requirements, defines the design as per the Red Hat reference architecture and maps the requirements to an overall architecture and design.

Deployment of Red Hat OpenShift Container Platform

- OpenShift installer to perform base installation of OpenShift Container Platform on bare metal (x86) system in one data centre with the following:
    - Three (3) masters configured in high-availability (HA) for OpenShift system functions.
    - Three (3) infrastructure nodes hosting OpenShift infrastructure pods/containers (monitoring, logging, routers, etc.).
    - Up to ten (10) application nodes for application pods/containers.
- Built-In Docker Registry, 2 x Application Routers
- Up to three (3) RBAC for Common Requirements (Project User, Project Admin, Cluster Admin, etc.) based on customer needs.
- Up to one (1) authentication provider (htpasswd, Active Directory, etc.).
- Basic Projects Creation and Node Mapping up to five (5) Projects.
- OpenShift monitoring and aggregated logging deployment and configuration.

- Cluster Validation.

OpenShift Virtualization Implementation

Red Hat will provide Professional Services to assist the Client with the Red Hat OpenShift Virtualization Platform Core Implementation, up to ten (10) nodes per cluster and up to two sites, each consisting of four clusters, all located in the same datacentre.

Deployment of OpenShift Data Foundations (ODF)

The engagement deploys on one (1) datacentre with up to two (2) highly-available ODF clusters. The existing servers, resources, and roles are documented in the proposal document and will be discussed during the Architecture Design Workshops.

Ceph Deployment

- The consultant will operate at the customer's location or remotely as agreed by the customer.
- The consultant will conduct a series of workshops for the high-level design of the Red Hat Ceph Storage.

Ansible Deployment

Red Hat conducts design workshops to adopt the Customer requirements, defines the functional requirements of the integration, and provides the use case description and third-party requirements to be configured by the customer to perform the integration.

Scope of Document

This document focuses on the following items in the scope of work:

- Architecture design for OpenShift and ODF: Red Hat conducted several workshops for the design and deployment of both OpenShift and ODF.
- Deployment of OpenShift and ODF: several workshops were conducted for deploying both OpenShift and ODF with a document as an outcome of the Method of Procedure (MoP) to meet the requirements for deploying OpenShift and ODF.
- OpenShift virtualisation: several workshops were conducted about virtualisation In OpenShift.

The remaining Items In the scope of the work are still open and remain to be proceeded by initially conducting related workshops with the Client.

Out of Scope

- Application Backup: This can be part of a future CSI functionality project.
- Cluster Deployment Automation: This can be part of a future E-2-E OpenShift Cluster Deployment Automation project.
- Ceph deployment. Ceph design must first be scoped through workshops with the Client, and Ceph deployed before ODF is configured to use external Ceph.
- Ansible deployment: requirements/ integration must first be defined through conducting workshops with the Client.

Deployment Phases

	PHASE	DETAIL	DEPENDENCIES
1	Customer Workshops		
2	Design environment	• Preparing Solution Document. • Compatibility matrix Verification. • IP VLAN Schema for the OCP cluster. • Nodes dimensioning. • Connectivity Matrix. • Certificate-Based Authentication Generation. • OCP External Integration Points & Services.	

3	Configuring the environment and Deploying the OCP cluster	• Applying Connectivity Matrix to the Environment. • Prepare the Installer Machine. • Software packages Downloading for OCP. • DNS server Configuration. • Prepare the OCP configuration files.	• Site readiness and computing resources. • Configuring the IP network in the customer Network. • Configure load balancer for UPI installation method (aligning between customer and load balancer solution provider is required). • Customer DNS server readiness. • Red hat Subscription.
4	Verifying	Internal Acceptance test	

OCP Hardware

Computing resources depend on application requirements; cluster nodes must meet the following requirements for the production environment.

- Red Hat Recommends three infrastructure nodes per cluster.
- Worker node sizing depends on application requirements.
- One vCPU is equivalent to one physical core when simultaneous multithreading (SMT), or hyperthreading, is not enabled. When enabled, use the following formula to calculate the corresponding ratio: (threads per core × cores) × sockets = vCPUs.
- OpenShift Container Platform and Kubernetes are sensitive to disk performance, and faster storage is recommended, particularly for etcd on the control plane nodes, which require a 10 ms p99 fsync duration. Note that storage size and IOPS scale together on many cloud platforms, so you might need to over-allocate storage volume to obtain sufficient performance.

OpenShift Cluster Sizing

Number of Clusters

As per the design discussion, one (1) OpenShift cluster will be deployed/configured using Dell Bare Metal servers. The cluster will consist of three (3) controller nodes, three (3) Infra nodes and ten (10) worker/application nodes.

Machine Sizing

As per the design workshops conducted, the sizing of the master will be set to 4 CPUs and 32 GiB of RAM as per the following OpenShift minimum requirements:

- https://docs.openshift.com/container-platform/4.13/installing/installing_bare_metal/installing-bare-metal.html

The infra nodes will be 16 CPU and 64GB per the estimated workloads.

Based on the application's estimated workload, the worker nodes will be sized according to the table below.

The table below details the OpenShift nodes and the sizes during the cluster's initial deployment.

FUNCTION	CPU	MEMORY	DISK	NETWORK	OS
Bastion	4 CPU	16GB	200GB	Cluster Network	RHEL
bootstrap	4 CPU	16GB	200GB	Cluster Network	RHCOS
master1	4 CPU	32GB	200GB	Cluster Network	RHCOS
master2	4 CPU	32GB	200GB	Cluster Network	RHCOS
master3	4 CPU	32GB	200GB	Cluster Network	RHCOS
Infra1	16 CPU	64GB	200GB 4 TB	Cluster Network	RHCOS
Infra2	16 CPU	64GB	200GB 4 TB	Cluster Network	RHCOS
Infra3	16 CPU	64GB	200GB 4 TB	Cluster Network	RHCOS
worker1	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker2	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker3	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker4	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker5	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker6	16 CPU	128GB	300GB	Cluster Network	RHCOS
worker7	16 CPU	128GB	300GB	Cluster Network	RHCOS

OpenShift Infrastructure Components Sizing

Image Registry

The image registry in OpenShift is responsible for storing and downloading all the container images used by all the nodes in the cluster. It is one of the critical components of the cluster and should be treated as such.

The registry can run in 2 different modes:

1. Non-Scaled registry: This is a single replica of the registry container running on one node.
2. Scaled registry: This is multiple replicas of the registry container running on different nodes.

ODF will be used with an initial size of 500GB for the registry. It can be increased later if more storage is required for the image registry.

Elasticsearch

Elasticsearch is the back-end used to store the cluster-wide logging information. All container logs within the cluster are shipped using the EFK stack and are stored in the Elasticsearch back-end.

Elasticsearch requires a ReadWriteOnce (RWO) (Block) storage for maintaining the logs and storing them persistently.

An initial estimate of 100GB will be used as the persistent storage for Elasticsearch. Elasticsearch can be redeployed to accommodate the application's actual workloads.

If Elastic search accommodates YOU Telecom applications and Openshift internal components, we must add more storage to the persistent storage assigned.

An estimate of 2 TB will be assigned then for the elastic search to support YOU Telecom applications.

Retention and scheduling maintenance operations can be done on the elastic search either on a project level (application level) or on the global level.

Prometheus

Prometheus is the back-end used to store the cluster-wide metrics information. All the cluster metrics (CPU/Memory utilisation, Pod count, etc.) are stored there. To maintain these metrics and keep them persistently, Prometheus requires a ReadWriteOnce (RWO) Block storage.

An initial estimate of 100GB will be used as the persistent storage for Prometheus. Prometheus can be redeployed to accommodate metrics and actual workloads.

By default, the Prometheus Cluster Monitoring stack configures the retention time for Prometheus data to be 15 days. You can modify the retention time to change how soon the data is deleted.

Network connectivity requirements

You must configure the network connectivity between machines to allow OpenShift Container Platform cluster components to communicate. Each node (machine host) must be able to resolve the hostnames of all other nodes in the cluster.

During cluster provisioning, the following URLs should be accessible:

URL	PORT	FUNCTION
registry.redhat.io	443, 80	Provides core container images
quay.io	443, 80	Provides core container images
cdn.quay.io	443, 80	Provides core container images
cdn01.quay.io	443, 80	Provides core container images
cdn02.quay.io	443, 80	Provides core container images
cdn03.quay.io	443, 80	Provides core container images
sso.redhat.com	443, 80	The https://console.redhat.com/openshift site uses authentication from sso.redhat.com
rhcos-redirector.apps.art.xq1c.p1.openshiftapps.com	443, 80	Provides Red Hat Enterprise Linux CoreOS (RHCOS) images

Further information about firewall requirements can be found here:
https://docs.openshift.com/container-platform/4.13/installing/install_config/configuring-firewall.html

Ports that are used for all-machine to all-machine communications:

PROTOCOL	PORT	DESCRIPTION
ICMP	N/A	Network reachability tests
TCP	1936	Metrics
	9000-9999	Host-level services, including the node exporter on ports 9100-9101 and the Cluster Version Operator on port 9099.
	10250-10259	The default ports that Kubernetes reserves
UDP	4789	VXLAN and Geneve
	6081	VXLAN and Geneve
	9000-9999	Host-level services, including the node exporter on ports 9100-9101.
	500	IPsec IKE packets
	4500	IPsec NAT-T packets
TCP/UDP	30000-32767	Kubernetes node port
ESP	N/A	IPsec Encapsulating Security Payload (ESP)

Ports used for all-machine to control plane communications:

PROTOCOL	PORT	DESCRIPTION
TCP	6443	OpenShift API
TCP	443	OpenShift dashboard

Allow list the following URLs:

URL	PORT	FUNCTION
registry.connect.redhat.com	443, 80	Required for all third-party images and certified operators.
rhc4tp-prod-z8cxf-image-registry-us-east-1-evenkyleffocxqvofrk.s3.dualstack.us-east- 1.amazonaws.com	443, 80	Provides access to container images hosted on registry.connect.redhat.com
oso-rhc4tp-docker-registry.s3-us-west-2.amazonaws.com	443, 80	Required for Sonatype Nexus, F5 Big IP operators.
mirror.openshift.com	443, 80	Required to access mirrored installation content and images. This site is also a source of release image signatures, although the Cluster Version Operator needs only a single functioning source.
storage.googleapis.com/openshift-release	443, 80	A source of release image signatures, although the Cluster Version Operator needs only a single functioning source.
*.apps.<cluster_name>.<base_domain>	443, 80	Required to access the default cluster routes unless you set an ingress wildcard during installation.
quayio-production-s3.s3.amazonaws.com	443, 80	Required to access Quay image content in AWS.
api.openshift.com	443, 80	Required both for your cluster token and to check if updates are available .
art-rhcos-ci.s3.amazonaws.com	443, 80	Required to download Red Hat Enterprise Linux CoreOS (RHCOS) images.
console.redhat.com/openshift	443, 80	Required for your cluster token.

registry.access.redhat.com	443, 80	Required for Odo CLI.
sso.redhat.com	443, 80	The https://console.redhat.com/openshift site uses authentication from sso.redhat.com
registry.redhat.io	443, 80	Provides core container images
quay.io	443, 80	Provides core container images
*.quay.io	443, 80	Provides core container images
*.openshiftapps.com	443, 80	Provides Red Hat Enterprise Linux CoreOS (RHCOS) images

Software version

VENDOR	SOFTWARE NAME	SOFTWARE VERSION	DESCRIPTION
RedHat	OpenShift Container Platform	4.13	Cloud-based Kubernetes container platform.
Redhat	Openshift Data Foundation	4.13	Container storage platform for Openshift provided by Red Hat

OpenShift Container Platform (RHSA-2023:1326) is available and uses Kubernetes 1.26 with CRI-O runtime.

This topic includes new features, changes, and known issues about OpenShift Container Platform 4.13.

OpenShift Container Platform 4.13 is supported on Red Hat Enterprise Linux (RHEL) 8.6, 8.7, and 8.8, as well as on Red Hat Enterprise Linux CoreOS (RHCOS) 4.13.

It would help if you used RHCOS machines for the control plane, and you can use either RHCOS or RHEL for compute machines.

Red Hat Enterprise Linux CoreOS (RHCOS) is a RHEL-based container operating system tailored and designed for containerised workloads. The following are some key features related to RHCOS:

- Controlled immutability: RHCOS Management is performed remotely from the OpenShift Container Platform cluster. When you set up your RHCOS machines, you can modify only a few system settings. This controlled immutability allows the

OpenShift Container Platform to store the latest state of RHCOS systems in the cluster to create additional machines and perform updates based on the latest RHCOS configurations.

- **CRI-O container runtime:** Although RHCOS contains features for running the OCI- and libcontainer-formatted containers that Docker requires, it incorporates the CRI-O container engine instead of the Docker container engine. By focusing on features needed by Kubernetes platforms, such as OpenShift Container Platform, CRI-O can offer specific compatibility with different Kubernetes versions. CRI-O also provides a smaller footprint and reduced attack surface than is possible with container engines that offer a more extensive feature set. Currently, CRI-O is the only engine available within OpenShift Container Platform clusters.
- **Set of container tools:** For tasks such as building, copying, and otherwise managing containers, RHCOS replaces the Docker CLI tool with a compatible set of container tools. The Podman CLI tool supports many container runtime features, such as running, starting, stopping, listing, and removing containers and container images. The Skopeo CLI tool can copy, authenticate, and sign images. You can use the crictl CLI tool to work with containers and pods from the CRI-O container engine. While direct use of these tools in RHCOS is discouraged, you can use them for debugging.
- **rpm-ostree upgrades:** RHCOS features transactional upgrades using the rpm-ostree system. Updates are delivered using container images and are part of the OpenShift Container Platform update process. When deployed, the container image is pulled, extracted, and written to disk, and then the bootloader is modified to boot into the new version. The machine will reboot into the update in a rolling manner to ensure cluster capacity is minimally impacted.
- **bootupd firmware and bootloader updater:** Package managers and hybrid systems such as rpm-ostree do not update the firmware or the bootloader. With bootupd, RHCOS users have access to a cross-distribution, system-agnostic update tool that manages firmware and boot updates in UEFI and legacy BIOS boot modes that run on modern architectures, such as x86_64, ppc64le, and aarch64.
- **Updated through the Machine Config Operator:** In the OpenShift Container Platform, the Machine Config Operator handles operating system upgrades. Instead of upgrading individual packages, as is done with yum upgrades, rpm-ostree delivers upgrades of the OS as an atomic unit. The new OS deployment is staged during upgrades and goes into effect on the next reboot. If something goes wrong with the upgrade, a single rollback and reboot returns the system to the previous state. RHCOS upgrades in the OpenShift Container Platform are performed during cluster updates.

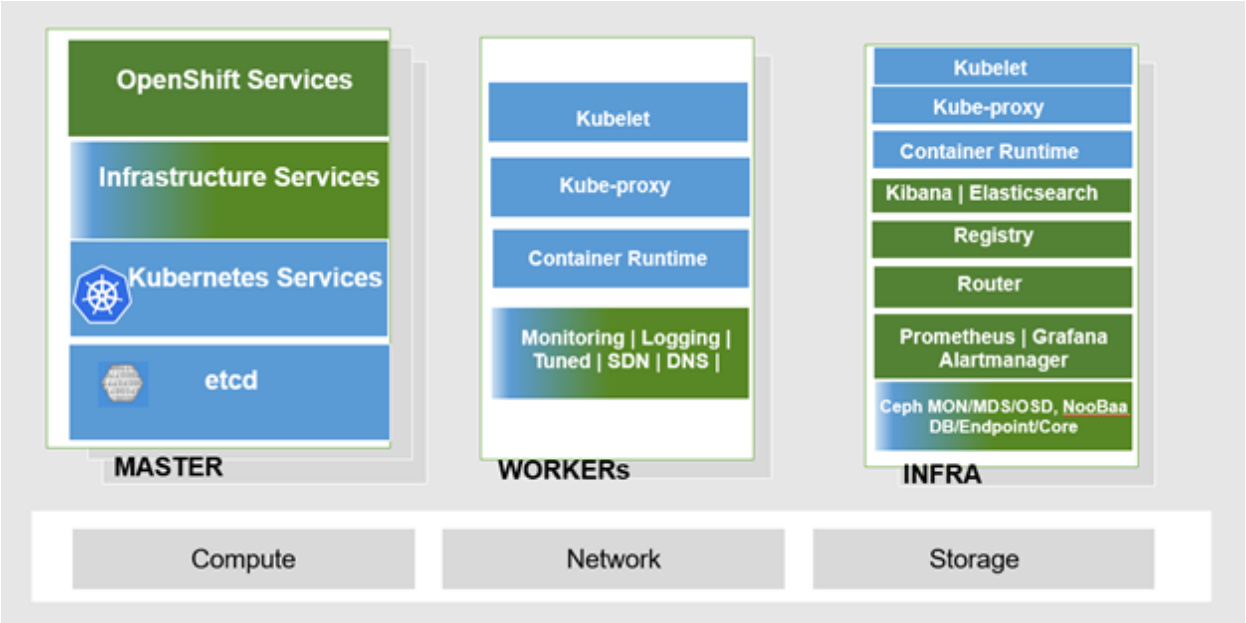
Further details on RHCOS can be found here: https://docs.openshift.com/container-platform/4.13/architecture/architecture-rhcos.html.

Kubernetes & OpenShift Architecture

Kubernetes is an open-source container orchestration project. It helps users manage clustered groups of hosts running Linux containers, which are processes containing everything needed to run in isolation.

Kubernetes was initially developed and designed by engineers at Google – one of the early contributors to Linux container technology—before it was donated to the Cloud Native Computing Foundation (CNCF) in 2015. That means the CNCF is responsible for maintaining the Kubernetes community, while volunteer contributors and administrators are responsible for Kubernetes development, maintenance, and releases.

Red Hat OpenShift is an enterprise open-source container orchestration platform. It's a software product that includes components of the Kubernetes container management project but adds productivity and security features that are important to large-scale companies.



OpenShift components & node types

An OpenShift Kubernetes cluster consists of a control plane and one or more worker nodes.

Control plane nodes (controller nodes)

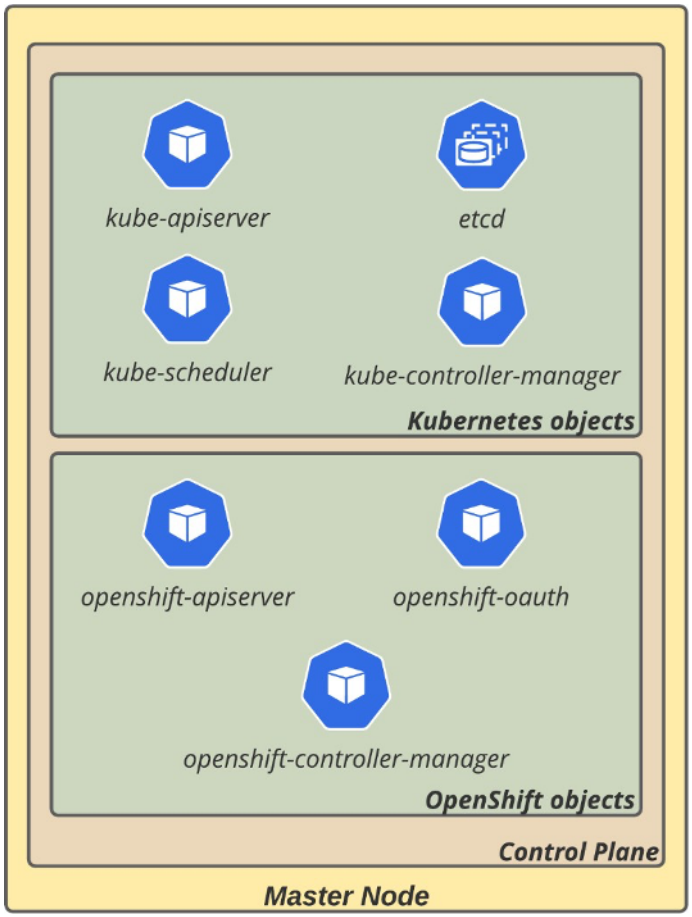
This server contains the control plane of a Kubernetes cluster. Controller servers on OpenShift run over the Red Hat Enterprise Linux CoreOS (RHCOS) operating system and are composed of several main components, such as the following:

- Application programming interface (API) server (kube-apiserver): Responsible for exposing all Kubernetes APIs. All actions performed on a Kubernetes cluster are done through an API call—whenever you use the command-line interface (CLI) or a user interface (UI), an API call will always be used.
- Database (etcd): The database stores all cluster data. etcd is a highly available distributed key-value database. For in-depth information about etcd, refer to its documentation here: https://etcd.io/docs/latest/.
- Scheduler (kube-scheduler): It is the responsibility of the kube-scheduler to assign a node for a Pod to run over. It uses complex algorithms that consider many aspects to decide which node is best to host the pod, such as computing resources available versus required node selectors, affinity and anti-affinity rules, etc.
- Controller manager (kube-controller-manager): Controllers are an endless loop that ensures an object is always in the desired state. As an illustration, think about smart home automation equipment: a controller is responsible for orchestrating the equipment to make sure the environment will always be in the desired programmed state – for example, by turning the air conditioning on and off from time to time to keep the temperature as close as possible to the desired state. Kubernetes controllers have the same function – they monitor objects and respond accordingly to keep them in the selected states. A bunch of controllers are used in a Kubernetes cluster, such as replication, endpoints, namespace, and service accounts. For more information about controllers, check out this page: https://kubernetes.io/docs/concepts/architecture/controller/.

These are the components of the Kubernetes control plane that runs on the controller nodes; however, OpenShift has some additional services to extend Kubernetes functionality, as outlined here:

- OpenShift API Server (openshift-apiserver): This validates and configures data for OpenShift-exclusive resources, such as routes, templates, and projects.
- OpenShift controller manager (openshift-controler-manager): This ensures that OpenShift exclusive resources reach the desired state.
- OpenShift Open Authorization (OAuth) server and API (openshift-oauth-apiserver): Validates and configures data to authenticate a user, group, and token with OpenShift.

The following figure shows the main control plane components of Kubernetes and OpenShift:



These components can be found in multiple namespaces, as you can see in the following table:

CONTROL PLANE COMPONENT	NAMESPACE	MANAGED BY
Kube-API server	openshift-kube-apiserver	kube-apiserver operator
etcd	openshift-etcd	etcd Operator
kube-scheduler	openshift-kube-scheduler	kube-scheduler operator
kube-controller-manager	openshift-kube-controller-manager	kube-controller-manager operator
openshift-apiserver	openshift-apiserver	openshift-API-server Operator
openshift-controller-manager	openshift-controller-manager	openshift-controller-manager Operator
openshift-oauth	openshift-oauth-apiserver	Authentication operator

Bootstrap node

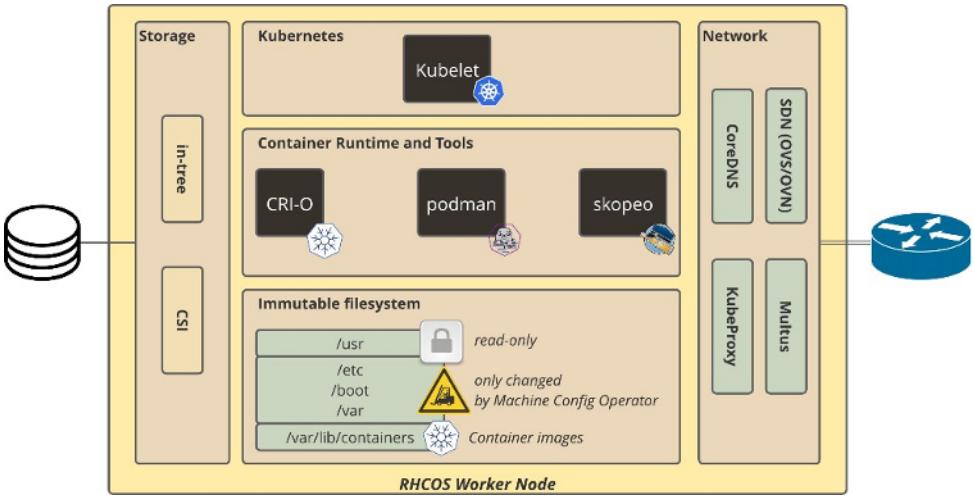
The bootstrap node is a temporary server—used only during cluster deployment—that is responsible for injecting the cluster's components into the control plane. It is removed by the installation program when the bootstrap is finished successfully. As it is a temporary server that lives only during the deployment, it is usually not considered in the OpenShift architecture.

Worker node

Workers are the servers where the workloads themselves run. On OpenShift, workers can run over RHCOS or Red Hat Enterprise Linux (RHEL). Although RHEL is also supported for OpenShift workers, RHCOS, in general, is preferred for the following reasons:

- **Immutable:** RHCOS is a tight operating system designed to be managed remotely by the OpenShift Container Platform. This way, it enables consistency and makes upgrades a much easier and safer procedure, as OpenShift will always know and manage the actual and desired states of the servers.
- **rpm-ostree:** RHCOS uses the rpm-ostree system, which enables transactional upgrades and adds consistency to the infrastructure. Check out this link for more information: https://coreos.github.io/rpm-ostree/.
- **CRI-O container runtime and tools:** RHCOS's default container runtime is CRI-O, optimised for Kubernetes (see https://cri-o.io/). It also has tools for working with containers, such as Podman and Skopeo. During normal functioning, you are not encouraged to access and run commands on workers directly (as the OpenShift platform manages them); however, those tools are helpful for troubleshooting purposes.
- **Based on RHEL:** RHCOS is based on RHEL—it uses the same well-known and safe RHEL kernel with some services managed by Systemd, which ensures the same security and quality you would have using the standard RHEL operating system.
- **Managed by Machine Config Operator (MCO):** To allow a high level of automation and keep secure upgrades, OpenShift uses the MCO to manage the configurations of the operating system. It uses the rpm-ostree system to make atomic upgrades, which allows safer and easier upgrades and rollback (if needed).

In the following figure, you can view how these objects and concepts are used in an OpenShift worker node:



Types of workers

There are some common types of workers, the most usual being these:

- **Application workers:** Responsible for hosting the workloads–this is where the application containers run.
- **Infrastructure workers:** This type of server is usually used to host the platform infrastructure tools, such as the ingress (routers), internal registry, the monitoring stack (Prometheus and Grafana), and the logging tool (Elasticsearch and Kibana).
- **Storage workers:** Container storage solutions, such as Red Hat OpenShift Data Foundation, usually require some dedicated worker nodes to host their Pods. In such cases, a best practice is to use a dedicated node group for them.

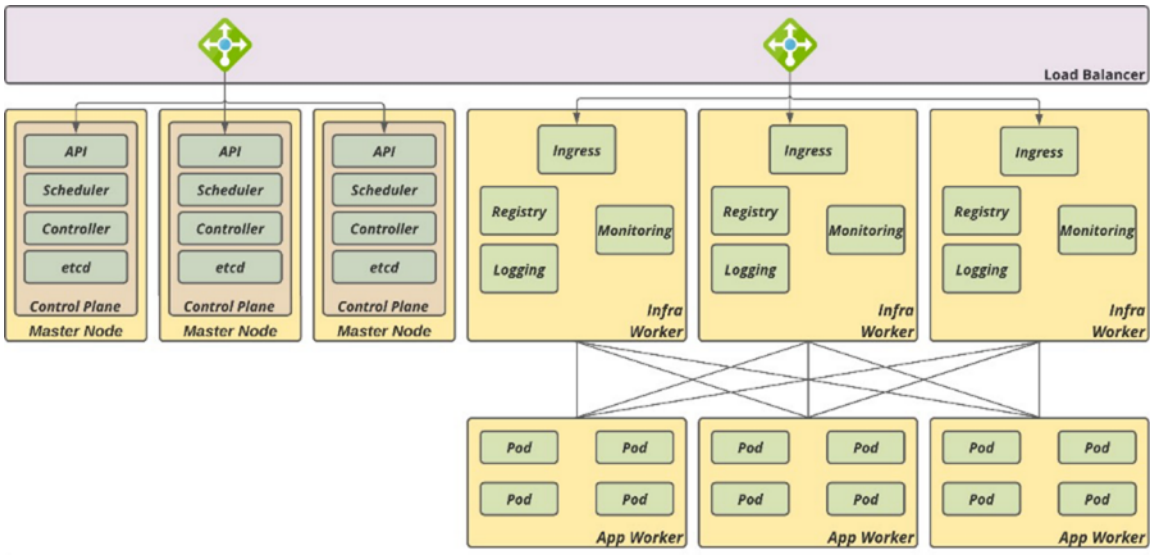
Highly available cluster

It is not uncommon for OpenShift clusters to become critical for the enterprise –sometimes, they start small but become large quickly. Due to that, you should consider in your OpenShift cluster architecture non-functional requirements (NFRs) such as high availability (HA) from day one. A highly available cluster is comprised of the following aspects:

- **Controller nodes:** etcd uses a distributed consensus algorithm named Raft protocol, which requires at least three nodes to be highly available. It is not the focus of this book to explain the Raft protocol, but if you want to understand it better, refer to these links:
  - Raft description: https://raft.github.io/.
  - Illustrated example: http://thesecretlivesofdata.com/raft/.
- **Infrastructure worker nodes:** At least two nodes must have ingress highly available. However, Red Hat supports at least three infra nodes for HA per OpenShift deployment.

- **Application worker nodes:** At least two nodes must be considered highly available; however, you may have as many nodes as required to provide enough capacity for expected workloads. In this chapter, we will walk through some sizing guidance to determine the number of workers needed for a workload if you have an estimated required capacity.

The following figure shows what a highly available cluster on bare metal architecture looks like for YOU Telecom:



About OpenShift installation

The OpenShift Container Platform installation program offers four methods for deploying a cluster:

- **Interactive**: You can deploy a cluster with the web-based Assisted Installer. This is the recommended approach for clusters with networks connected to the internet. The Assisted Installer is the easiest way to install the OpenShift Container Platform. It provides intelligent defaults and performs pre-flight validations before installing the cluster. It also provides a RESTful API for automation and advanced configuration scenarios.
- **Local Agent-based**: You can deploy a cluster locally with the agent-based installer for air-gapped or restricted networks. It provides many of the benefits of the Assisted Installer, but you must download and configure the agent-based installer first. Configuration is done with a command line interface. This approach is ideal for air-gapped or restricted networks.
- **Automated**: You can deploy a cluster on installer-provisioned infrastructure and the cluster it maintains. The installer uses each cluster host's baseboard management controller (BMC) for provisioning. You can deploy clusters with both connected or air-gapped or restricted networks.

- **Complete control**: You can deploy a cluster on the infrastructure you prepare and maintain, providing maximum customizability. You can deploy clusters with both connected or air-gapped or restricted networks.

The clusters have the following characteristics:

- Highly available infrastructure with no single points of failure is embedded by default.
- Administrators maintain control over what updates are applied and when.

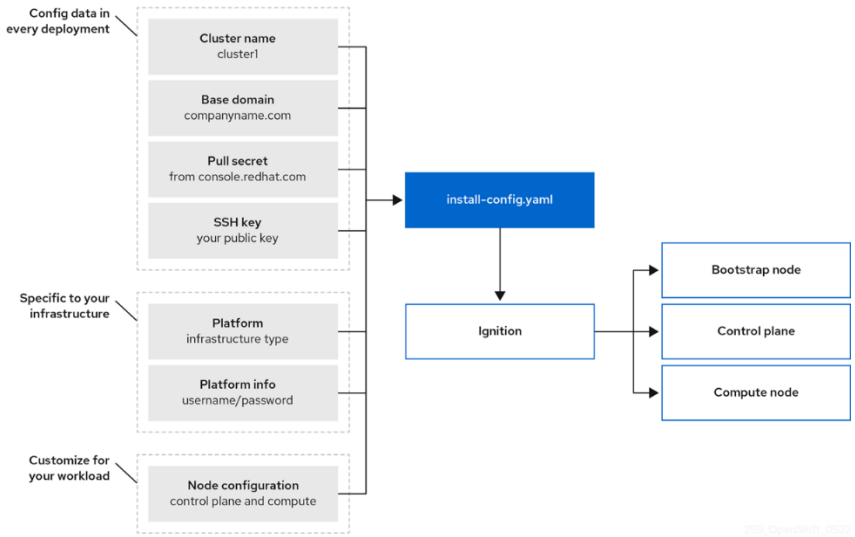
About the installation program

You can use the installation program to deploy each type of cluster. The installation program generates main assets such as Ignition config files for the bootstrap, control plane (master), and worker machines.

Start an OpenShift Container Platform cluster with these three configurations and correctly configured infrastructure.

The OpenShift Container Platform installation program uses a set of targets and dependencies to manage cluster installations. The installation program has a bunch of targets that it must achieve, and each target has a set of dependencies. Because each target is only concerned with its dependencies, the installation program can act to achieve multiple targets in parallel, with the ultimate target being a running cluster. The installation program recognises and uses existing components instead of running commands to create them again because it meets dependencies.

The following picture shows you OpenShift Container Platform installation targets and dependencies.



About Red Hat Enterprise Linux CoreOS (RHCOS)

Post-installation, each cluster machine uses Red Hat Enterprise Linux CoreOS (RHCOS) as the operating system. RHCOS is the immutable container host version of Red Hat Enterprise Linux (RHEL) and features a RHEL kernel with SELinux enabled by default. It includes the kubelet, the Kubernetes node agent, and the CRI-O container runtime, which is optimised for Kubernetes.

Every control plane machine in an OpenShift Container Platform 4.13 cluster must use RHCOS, which includes a critical first-boot provisioning tool called Ignition. This tool enables the cluster to configure the machines. Operating system updates are delivered as a bootable container image, using **OSTree** as a back-end, deployed across the cluster by the Machine Config Operator. Actual operating system changes are made on each machine as an atomic operation using **rpm-ostree**. Together, these technologies enable the OpenShift Container Platform to manage the operating system like any other application on the cluster by in-place upgrades that keep the entire platform up-to-date. These in-place updates can reduce the burden on operations teams.

If you use RHCOS as the operating system for all cluster machines, the cluster manages all aspects of its components and machines, including the operating system. Because of this, only the installation program and the Machine Config Operator can change machines. The installation program uses Ignition config files to set the exact state of each machine, and the Machine Config Operator completes more changes to the machines, such as the application of new certificates or keys, after installation.

Installation process

Except for the Assisted Installer, when you install an OpenShift Container Platform cluster, you download the installation program from the appropriate Infrastructure Provider page on the OpenShift Cluster Manager site. This site manages:

- REST API for accounts
- Registry tokens, which are the pull secrets that you use to obtain the required components
- Cluster registration, which associates the cluster identity to your Red Hat account to facilitate the gathering of usage metrics

In OpenShift Container Platform 4.13, the installation program is a Go binary file that performs a series of file transformations on a set of assets.

Both installation methods, Installer Provisioned Infrastructure (IPI) - for small deployments -, and User Provisioned Infrastructure (UPI) - for enterprise deployments -, are an essential primer to understand and conceptualise the requirements for deploying a thriving OpenShift cluster in a cloud orchestration model.

However, the UPI installation is highly customisable and tunable. The infrastructure is not configured within the installation of OpenShift, and the cluster heavily relies on the proper configuration of the following services:

- DHCP
- DNS
- Proxy
- Router
- NAT
- Firewall
- web hosting
- LDAP (Active Directory or equivalent)
- TFTP/SFTP server
- NFS

Given the customisation and flexibility of a UPI installation, this methodology is the most representative of an on-premises enterprise deployment.

The IPI installation provides a turnkey solution and includes all the necessary infrastructure services within the OpenShift cluster. Significant planning must be done before deployment to ensure you calculate and size the capacity size of the deployment.

Furthermore, the load balancer built into the OpenShift cluster that fronts the installed applications cannot be replaced. This load balancer could quickly become a bottleneck if the application workloads suddenly require an enterprise load balancer.

In the case of YOU Telecom, we select deploying OpenShift using the user-provisioned infrastructure method.

In this approach, you provision and manage the infrastructure for your cluster. You must provide the cluster infrastructure and resources, including the bootstrap machine, networking, load balancing, storage, and individual cluster machines.

In General, the installer uses three sets of files during installation: an installation configuration file named install-config.yaml, Kubernetes manifests, and Ignition config files for your machine types.

The installation configuration file is transformed into Kubernetes manifests, which are then wrapped into Ignition config files. The installation program uses these Ignition config files to create the cluster.

The installation configuration files are pruned when you run the program, so be sure to back up all configuration files you want to use again.

User-provisioned infrastructure method

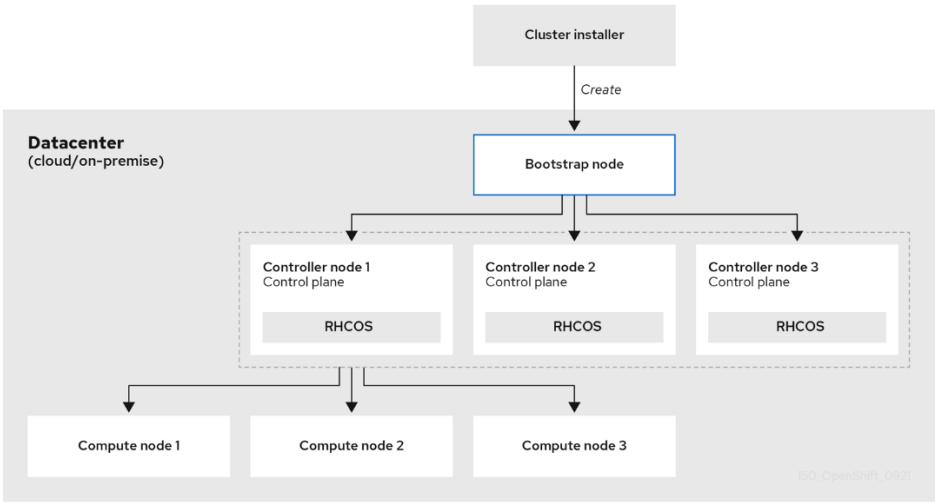
You use the installation program to generate the assets required to provision the cluster infrastructure, create the cluster infrastructure, and then deploy the cluster to the provided infrastructure. You must manage and maintain the cluster resources yourself, including:

- The underlying infrastructure for the control plane and compute machines that make up the cluster
- Load balancers
- Cluster networking, including the DNS records and required subnets
- Storage for the cluster infrastructure and applications

Installation process details

Because each machine in the cluster requires information about the cluster when it is provisioned, the OpenShift Container Platform uses a temporary **bootstrap** machine during initial configuration to provide the necessary information to the permanent control plane. It boots using an Ignition config file describing how to create the cluster. The bootstrap machine establishes the control plane machines that make up the control plane. The control plane machines create the compute machines, also known as worker machines.

The following figure illustrates this process of creating the bootstrap, control plane, and compute machines:



After the cluster machines initialise, the bootstrap machine is destroyed. All clusters use the bootstrap process to initialise the cluster, but if you provision the infrastructure for your cluster, you must complete many steps manually.

Bootstrapping a cluster involves the following steps:

1. The bootstrap machine boots and starts hosting the remote resources required to boot the control plane machines (Requires manual intervention if you provision the infrastructure).
2. The bootstrap machine starts a single-node etcd cluster and a temporary Kubernetes control plane.
3. The control plane machines fetch the remote resources from the bootstrap machine and finish booting. (Requires manual intervention if you provision the infrastructure).

4. The temporary control plane schedules the production control plane to the production control plane machines.
5. The Cluster Version Operator (CVO) installs the etcd Operator online. The etcd Operator scales up etcd on all control plane nodes.
6. The temporary control plane shuts down and passes control to the production control plane.
7. The bootstrap machine injects OpenShift Container Platform components into the production control plane.
8. The installation program shuts down the bootstrap machine. (Requires manual intervention if you provision the infrastructure).
9. The control plane sets up the compute nodes.
10. The control plane installs additional services as a set of Operators.

The result of this bootstrapping process is a running OpenShift Container Platform cluster. The cluster then downloads and configures the components needed for the day-to-day operation, including creating compute machines in supported environments.

After the cluster machines initialise, the bootstrap machine is destroyed. All clusters use the bootstrap process to initialise the cluster, but if you provision the infrastructure for your cluster, you must complete many steps manually.

PORT	BACK-END MACHINES (POOL MEMBERS)	INTERNAL	EXTERNAL	DESCRIPTION
6443	Bootstrap and control plane: You remove the bootstrap machine from the load balancer after the bootstrap machine initialises the cluster control plane.	X	X	Kubernetes API server
22623	Bootstrap and control plane: You remove the bootstrap machine from the load balancer after the bootstrap machine initialises the cluster control plane.	X		Machine config server

OpenShift Operators

Operators are a method of packaging, deploying, and managing a Kubernetes application by encoding human operational knowledge into software that is more easily shared with consumers.

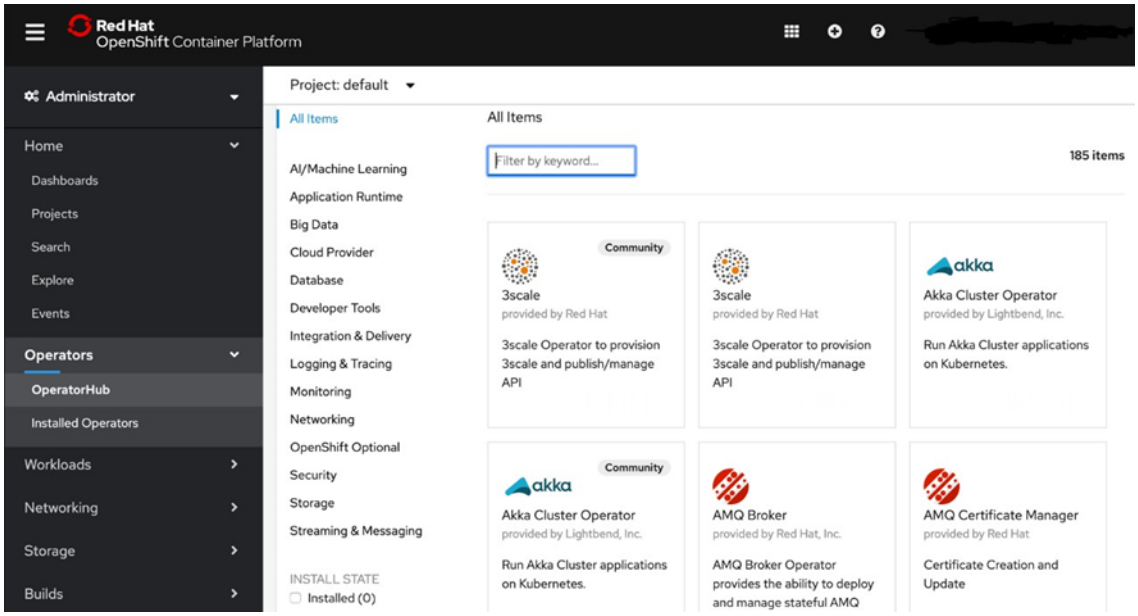
Operators are software that ease the operational complexity of running another piece of software. They act like an extension of the software vendor's engineering team, monitoring a Kubernetes environment (such as OpenShift Container Platform) and using its current state to make decisions in real-time. Advanced Operators are designed to handle upgrades seamlessly, react to failures automatically, and not take shortcuts, like skipping a software backup process to save time.

Operator Framework

The Operator Framework is a family of tools and capabilities to deliver on the customer experience described above. It is not just about writing code; testing, producing, and updating Operators is just as important.

The Operator Framework components consist of open-source tools to tackle these problems:

- **Operator SDK:** The Operator SDK assists Operator authors in bootstrapping, building, testing, and packaging their Operator based on their expertise without requiring knowledge of Kubernetes API complexities.
- **Operator Lifecycle Manager:** Operator Lifecycle Manager (OLM) controls the installation, upgrade, and role-based access control (RBAC) of Operators in a cluster. It is deployed by default in OpenShift Container Platform 4.13.
- **Operator Registry:** The Operator Registry stores cluster service versions (CSVs) and custom resource definitions (CRDs) for creation in a cluster and stores Operator metadata about packages and channels. It runs in a Kubernetes or OpenShift cluster to provide this Operator catalogue data to OLM.
- **Operator Hub:** Operator Hub is a web console for cluster administrators to discover and select Operators to install on their cluster. It is deployed by default in the OpenShift Container Platform.
- With one click, an Operator can be pulled from its off-cluster source, installed and subscribed on the cluster, and made ready for engineering teams to self-service manage the product across deployment environments using Operator Lifecycle Manager (OLM)



OpenShift High Availability

The architecture consists of three (3) Controller nodes (odd numbers) to provide a highly available design. Each controller node will have the following control components:

- ETCD
- Scheduler
- Controller manager
- API Server

The table and figure below depict the HA role matrix of OpenShift control components:

SERVICE NAME	HA MODE	DESCRIPTION
API Server	Active-Active	Managed by HAProxy
Etcd	Active-Active	Fully redundant deployment with load balancing
Controller Manager	Active-Passive	One instance is selected as a cluster leader at a time
HAProxy	Active-Passive	Balances load between API master endpoints.

ETCD HA Design

Etcd is the distributed data store where all the API objects will be kept. The architecture includes three instances of etcd, one on each controller node for high availability. Each etcd instance will have the IP address and port number of other instances, which means that every etcd instance will have information of the other two instances.

Etcd will replicate data across all instances. If one instance fails or gets disconnected, the other two nodes would maintain the majority can accept the cluster state changes and would be able to communicate with the clients (API server). Once the failed node returns, it will be updated with the latest information by the other instances.

API HA Design

The API server would remain stateless, although it does cache, etc. The architecture proposes to run three (3) instances of API server, one on each controller node, to make it accessible through load-balancer.

Scheduler and Controller HA Design

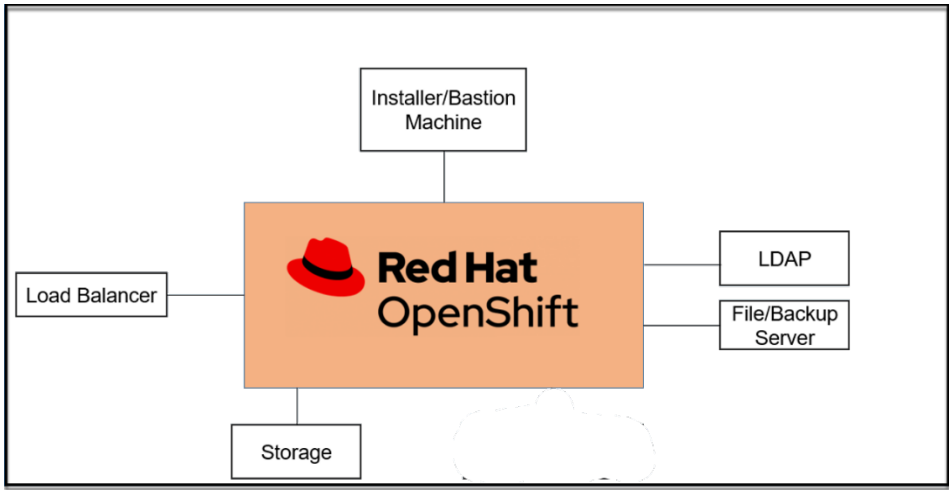
As the Scheduler and controller actively watch the cluster state for changes, running multiple active instances of the Scheduler and controller could cause race conditions and affect the performance. For this reason, only one instance may be active when running multiple instances of these components.

The controller and Scheduler take care of this by following the leader-election procedure. The instance will only be active when it's the elected leader. Only the leader performs actual work, whereas all other instances are standing by and waiting for the current leader to fail. When it does, the remaining instances elect a new leader who takes over the work.

This mechanism ensures that two components never operate simultaneously and do the same work. The controller and Scheduler would run collocated with API, etcd.

OCP High-level Architecture Overview

Solution Architecture



OpenShift infrastructure requirements

DHCP

During the initial boot, the machines require an IP address configuration set through a DHCP server or statically by providing the needed boot options. After establishing a network connection, the machines download their Ignition config files from an HTTP or HTTPS server. The Ignition config files are then used to set the exact state of each machine. After installation, the Machine Config Operator completes more machine changes, such as applying new certificates or keys.

Using a DHCP server for long-term management of the cluster machines is recommended. Ensure the DHCP server is configured to provide the cluster machines with persistent IP addresses, DNS server information, and host names.

The Kubernetes API server must be able to resolve the node names of the cluster machines. If the API servers and worker nodes are in different zones, you can configure a default DNS search zone to allow the API server to resolve the node names. Another supported approach is constantly referring to hosts by their fully qualified domain names in the node objects and all DNS requests.

- All nodes must be in the same VLAN.
- For scaling the cluster, you can use a second VLAN as a Day 2 operation.
- The following networking resources must be configured before you install the OpenShift Container Platform cluster.

Setting node hostnames through DHCP

The hostname is set through Network Manager on Red Hat Enterprise Linux CoreOS (RHCOS) machines. By default, the machines obtain their hostname through DHCP. If DHCP does not provide

the hostname, set statically through kernel arguments or another method, it is obtained through a reverse DNS lookup. Reverse DNS lookup occurs after the network has been initialised on a node and can take time to resolve. Other system services can start before this and detect the hostname as localhost or similar. You can avoid this by using DHCP to provide the hostname for each cluster node.

Additionally, setting the hostnames through DHCP can bypass manual DNS record name configuration errors in environments with a DNS split-horizon implementation.

NTP

An NTP server must be provided and be reachable.

Openshift CIDR blocks

Machine CIDR ranges cannot be changed after creating your cluster. When specifying subnet CIDR ranges, ensure the Subnet CIDR range is within the defined Machine CIDR.

Since the OpenShift cluster will communicate with YOU Telecom's on-premises resources, the OpenShift CIDR blocks must be separate from these resources.

YOU Telecom needs to make sure that the bellow-CIDR blocks do not conflict with the on-premises resources the applications will communicate with:

- In the Machine CIDR field, you must specify the IP address range for machines or cluster nodes. This range must encompass all CIDR address ranges for your virtual private cloud (VPC) subnets. Subnets must be contiguous. A minimum IP address range of 128 addresses, using the subnet prefix /25, is supported for single availability zone deployments. A minimum address range of 256 addresses, using the subnet prefix /24, is supported for deployments that use multiple availability zones. The default is 10.0.0.0/16. This range must not conflict with any connected networks.
- **Service CIDR**: in the Service CIDR field, you must specify the IP address range for services. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed within the cluster. The default is 172.30.0.0/16. This address block needs to be the same between clusters.
- **Pod CIDR**: in the pod CIDR field, you must specify the IP address range for pods. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed within the cluster. The default is 10.128.0.0/14. This address block needs to be the same between clusters.
- **Host Prefix**: in the Host Prefix field, you must Specify the subnet prefix length assigned to pods scheduled to individual machines. The host prefix determines the pod IP address pool for each machine. For example, if the host prefix is set to /23, each machine is assigned a /23 subnet from the pod CIDR address range. The default is /23, allowing 512 cluster nodes and 512 pods per node (both of which are beyond the maximum supported).

User-provisioned DNS requirements

In OpenShift Container Platform deployments, DNS name resolution is required for the following components:

- The Kubernetes API
- The OpenShift Container Platform application wildcard
- The bootstrap, control plane, and computer machines

Reverse DNS resolution is also required for the Kubernetes API, bootstrap, control plane, and compute machines.

DNS A/AAAA or CNAME records are used for name resolution, and PTR records are used for reverse name resolution. The reverse records are essential because Red Hat Enterprise Linux CoreOS (RHCOS) uses the reverse records to set the hostnames for all the nodes unless DHCP provides the hostnames. The reverse records are also used to generate the certificate signing requests (CSR) that the OpenShift Container Platform needs to operate.

The following DNS records are required for a user-provisioned OpenShift Container Platform cluster and must be in place before installation. In each record, <cluster name> is the cluster name, and <base domain> is the cluster base domain that you specify in the <Install_config.yaml> file.

A complete DNS record takes the form of <component>.<cluster name>.<base domain>.

DNS	IP ADDRESS
Primary DNS	
Secondary DNS	

Following is the cluster to be deployed and its names:

ENVIRONMENT	NAME	CONTROLLER NODES	WORKER NODES	INFRA NODES	BASE DOMAIN
Production		3	+3	3	

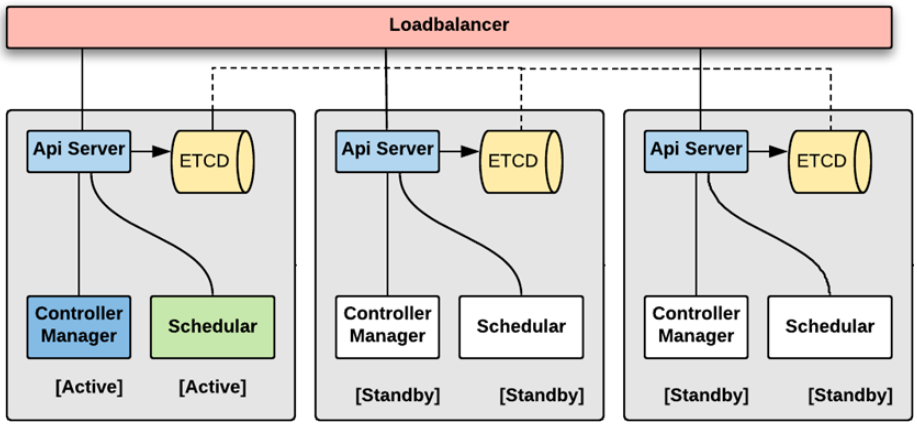
YOU Telecom needs to provide the information above according to the naming convention required in this setup.

COMPONENT	RECORD	DESCRIPTION
Kubernetes API	api.<cluster_name>.<base_domain>.	A DNS A/AAAA or CNAME record and a DNS PTR record to identify the API load balancer. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster.
	api-int.<cluster_name>.<base_domain>.	A DNS A/AAAA or CNAME record and a DNS PTR record for identifying the internal load balancer API. These records must be resolvable from all the nodes within the cluster. The API server must be able to resolve the worker nodes by the hostnames that are recorded in Kubernetes. If the API server cannot resolve the node names, proxied API calls can fail, and you cannot retrieve the log from the pod.

Routes	*. apps.<cluster_name>.<base_domai n> .	A wildcard DNS A/AAAA or CNAME record that refers to the application ingress load balancer. The application ingress load balancer targets the machines that run the Ingress Controller pods. The Ingress Controller pods run on the computer machines by default. These records must be resolvable by both clients external to the cluster and from all the nodes within the cluster. For example, console-openshift-console.apps.<cluster_name>.<base_dom ai n> is used as a wildcard route to the OpenShift Container Platform console
Bootstrap machine	bootstrap.<cluster_name>.<base_ do main>.	A DNS A/AAAA or CNAME record and a DNS PTR record to identify the bootstrap machine. These records must be resolvable by the nodes within the cluster.
Control plane machines	<master><n>.<cluster_name>.<base _ domain>.	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the control plane nodes. These records must be resolvable by the nodes within the cluster.
Compute machines	<worker><n>.<cluster_name>.<base _domain>.	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the worker nodes. These records must be resolvable by the nodes within the cluster.
Infra Nodes	<infra><n>.<cluster_name>.<base_d o main>.	DNS A/AAAA or CNAME records and DNS PTR records to identify each machine for the infra nodes. These records must be resolvable by the nodes within the cluster.

Load balancing requirements for UPI

Third-party load-balancing infrastructure must meet the following requirements:



Before you install the OpenShift Container Platform, you must provision the API and application Ingress load balancing infrastructure. In production scenarios, you can deploy the API and application Ingress load balancers separately to scale the load balancer infrastructure for each in isolation.

The load-balancing infrastructure must meet the following requirements:

1. **API load balancer**: Provides a common endpoint for human and machine users to interact with and configure the platform. Configure the following conditions:

    - Layer 4 load balancing only. It is known as Raw TCP, SSL Passthrough, or SSL Bridge mode. If you use SSL Bridge mode, you must enable Server Name Indication (SNI) for the API routes.
    - A stateless load balancing algorithm. The options vary based on the load balancer implementation.

    Do not configure session persistence for an API load balancer. Configuring session persistence for a Kubernetes API server might cause performance issues from excess application traffic for your OpenShift Container Platform cluster and the Kubernetes API that runs inside the cluster.

2. Configure the following ports on both the front and back of the load balancers:

PORT	BACK_END MACHINES (POOL MEMBERS)	INTERNAL	EXTERNAL	DESCRIPTION
6443	Bootstrap and control plane: You remove the bootstrap machine from the load balancer after the bootstrap machine initialises the cluster control plane. You must configure the /readyz endpoint for the API server health check probe.	X	X	Kubernetes API server
22623	Bootstrap and control plane: You remove the bootstrap machine from the load balancer after the bootstrap machine initialises the cluster control plane.	X		Machine config server

3. The load balancer must be configured to take 30 seconds from when the API server turns off the /readyz endpoint to remove the API server instance from the pool. The endpoint must have been removed or added within the time frame after /readyz returns an error or becomes healthy. Probing every 5 or 10 seconds, with two successful requests to become healthy and three to become unhealthy, are well-tested values.
4. **Application Ingress load balancer**: Provides an ingress point for application traffic outside the cluster. A working configuration for the Ingress router is required for an OpenShift Container Platform cluster.

   Configure the following conditions:

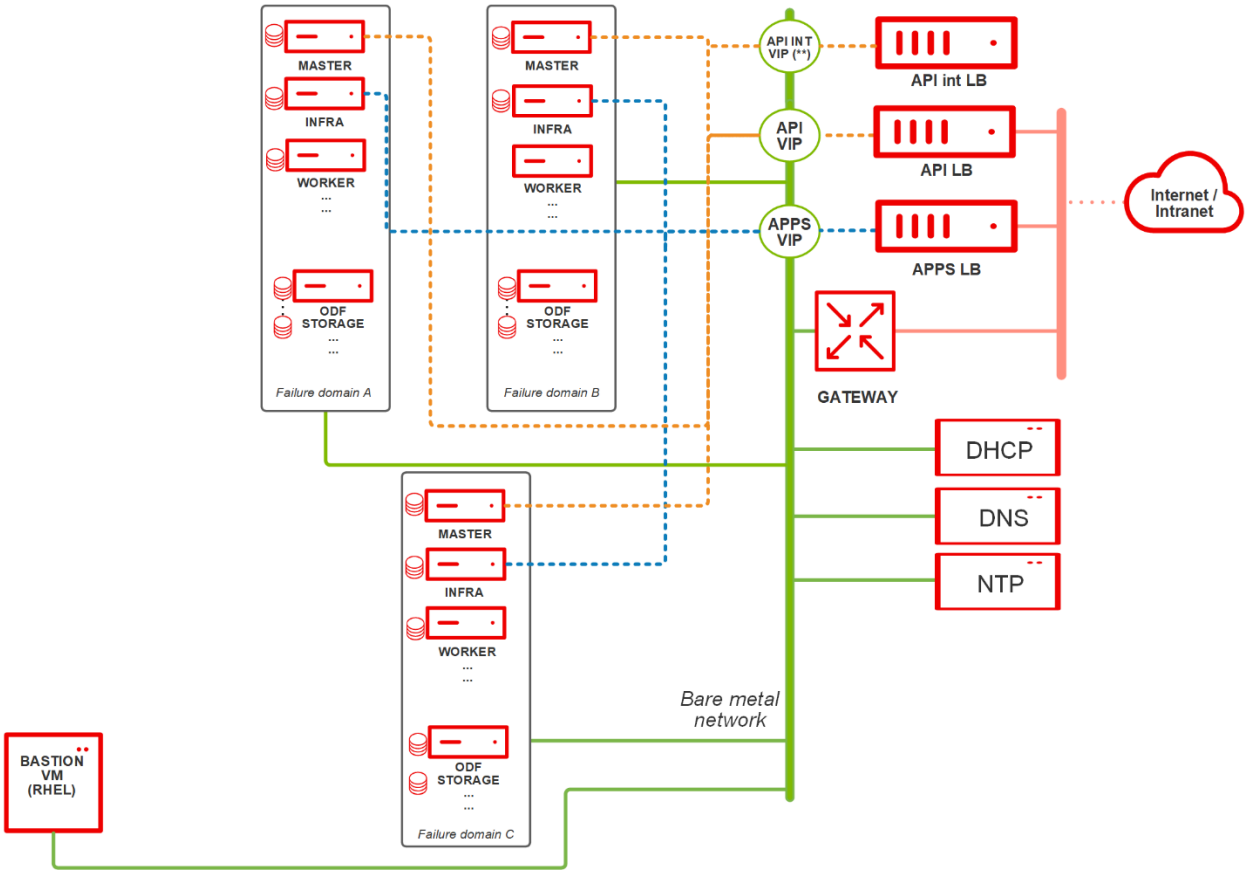
   - Layer 4 load balancing only. It is known as Raw TCP, SSL Passthrough, or SSL Bridge mode. If you use SSL Bridge mode, you must enable Server Name Indication (SNI) for the ingress routes.
   - Connection-based or session-based persistence is recommended based on the options available and the types of applications that will be hosted on the platform.

   If the actual IP address of the Client can be seen by the application Ingress load balancer, enabling source IP-based session persistence can improve performance for applications that use end-to-end TLS encryption.

For applications ingress load balancer, configure the following ports on both the front and back of the load balancers:

PORT	BACK-END MACHINES (POOL MEMBERS)	INTERNAL	EXTERNAL	DESCRIPTION
443	By default, the machines that run the Ingress Controller pods, compute, or worker nodes.	X	X	HTTPS traffic
80	The machines that run the Ingress Controller pods, compute or worker nodes by default.	X	X	HTTP traffic
	If you deploy a three-node cluster with zero compute nodes, the Ingress Controller pods run on the control plane nodes. In three-node cluster deployments, you must configure your application Ingress load balancer to route HTTP and HTTPS traffic to the control plane nodes.			

Platform node connectivity



Storage in OpenShift

OpenShift Container Platform storage is broadly classified into ephemeral and persistent.

Storage Types

Ephemeral Storage

Pods and containers can require ephemeral or transient local storage for their operation. The lifetime of this ephemeral storage does not extend beyond the life of the individual pod, and this ephemeral storage cannot be shared across pods.

Pods use ephemeral local storage for scratch space, caching, and logs.

Issues related to the lack of local storage accounting and isolation include the following:

- Pods need to find out how much local storage is available to them.
- Pods cannot request guaranteed local storage.
- Local storage is a best-effort resource.
- Pods can be evicted due to other pods filling the local storage, after which new pods are only admitted once sufficient storage has been reclaimed.

Types of ephemeral storage

Ephemeral local storage is always made available in the primary partition. Two basic ways of creating the primary partition are root and runtime.

Root

This partition holds the kubelet root directory, /var/lib/kubelet/ by default, and /var/log/ directory. This partition can be shared between user pods, the OS, and Kubernetes system demons. Pods can consume this partition through EmptyDir volumes, container logs, image layers, and container-writable layers. Kubelet manages shared access and isolation of this partition. This partition is ephemeral, and applications cannot expect performance SLAs, such as disk IOPS, from this partition.

Runtime.

It is an optional partition that runtimes can use for overlay file systems. OpenShift Container Platform attempts to identify and provide shared access and isolation to this partition. Container image layers and writable layers are stored here. If the runtime partition exists, the root partition does not hold any image layer or other writable storage.

Persistent Storage

Stateful applications deployed in containers require persistent storage. OpenShift Container Platform uses a pre-provisioned storage framework called persistent volumes (PV) to allow cluster administrators to provision persistent storage. The data inside these volumes can exist beyond the lifecycle of an individual pod. Developers can use persistent volume claims (PVCs) to request storage requirements.

PVs are resources in the cluster. PVCs are requests for those resources and act as claim checks to the resource. The interaction between PVs and PVCs has the following lifecycle.

Persistent Volumes (PV)

Cluster administrators can create several PVs in advance that carry the details of the actual storage that is available for use. PVs exist in the API and are available for use.

Persistent Volume Claim (PVC)

When you create a PVC, you request a specific amount of storage, specify the required access mode, and create a storage class to describe and classify the storage. The control loop in the master watches for new PVCs and binds the new PVC to an appropriate PV. If a proper PV does not exist, a provisioner for the storage class creates one.

The size of all PVs might exceed your PVC size. This is especially true with manually provisioned PVs. To minimise the excess, the OpenShift Container Platform binds to the smallest PV that matches all other criteria.

Reclaim policy for persistent volumes.

The reclaim policy of a persistent volume tells the cluster what to do with it after it is released. A volume's reclaim policy can be Retain, Recycle, or Delete.

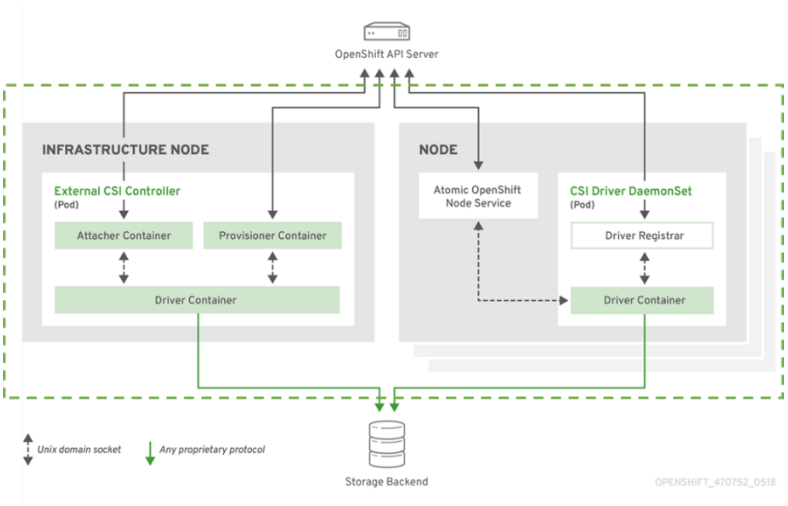
- The "Retain" reclaim policy allows manual reclamation of the resource for those volume plug-ins that support it.
- The "Recycle" reclaim policy recycles the volume into the pool of unbound persistent volumes once released from its claim.
- The "Delete" reclaim policy deletes the PersistentVolume object from the OpenShift Container Platform and the associated storage asset in external infrastructure.

Container Storage Interface (CSI)

Container Storage Interface (CSI) allows OpenShift Container Platform to consume storage from storage back ends that implement the CSI interface as persistent storage.

CSI drivers are typically shipped as container images. These containers must be aware of the OpenShift Container Platform where they run. To use the CSI-compatible storage back end in the OpenShift Container Platform, the cluster administrator must deploy several components that serve as a bridge between the OpenShift Container Platform and the storage driver.

The following diagram provides a high-level overview of the components running in pods in the OpenShift Container Platform cluster.



OpenShift Container Platform supports the following persistent volume plug-ins:

- AWS Elastic Block Store (EBS).
- Azure Disk.
- Azure File.
- Cinder.
- Fibre Channel.
- GCE Persistent Disk.
- HostPath.
- iSCSI.
- Local volume.
- NFS.
- OpenStack Manila.
- Red Hat OpenShift Container Storage, also known as RH OpenShift Data Foundation.
- VMware vSphere.

In YOU Telecom deployment, we will use Red Hat OpenShift Data Foundation plug-ins.

An external CSI attacher container translates attach and detach calls from the OpenShift Container Platform to “ControllerPublish" and "ControllerUnpublish" calls to the CSI respective driver.

An external CSI provisioner container that translates "provision" and "delete" calls from the OpenShift Container Platform to respective "CreateVolume" and "DeleteVolume" to respective CSI drivers.

A CSI driver container.

The CSI attacher and CSI provisioner containers communicate with the CSI driver container using UNIX Domain Sockets, ensuring that no CSI communication leaves the pod. The CSI driver is not accessible from outside of the pod.

CSI driver daemon set

The CSI driver daemon set runs a pod on every node that allows OpenShift Container Platform to mount storage provided by the CSI driver to the node and use it in user workloads (pods) as persistent volumes (PVs). The pod with the CSI driver installed contains the following containers:

- A CSI driver registrar registers the CSI driver into the openshift-node service running on the node. The openshift-node process running on the node then directly connects with the CSI driver using the UNIX Domain Socket available on the node.
- A CSI driver.

In YOU Telecom design, OpenShift Data Foundation will be used as a storage solution based on a discussion with the client infrastructure team.

OpenShift Data Foundation

Red Hat OpenShift Data Foundation is a highly integrated collection of cloud storage and data services for the Red Hat OpenShift Container Platform. It is available as part of the Red Hat OpenShift Container Platform Service Catalog, packaged as an operator to facilitate simple deployment and management.

OpenShift Data Foundation Components

Red Hat OpenShift Data Foundation services are primarily made available to applications by way of storage classes that represent the following components:

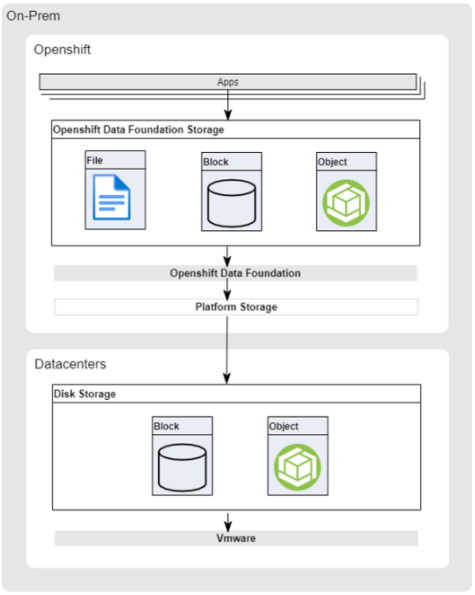
- Block storage devices, catering primarily to database workloads. Prime examples include Red Hat OpenShift Container Platform logging and monitoring and PostgreSQL.
- Shared and distributed file system, catering primarily to software development, messaging, and data aggregation workloads. Examples include Jenkins build sources and artefacts, WordPress uploaded content, Red Hat OpenShift Container Platform registry, and JBoss AMQ messaging.
- Multicloud object storage features a lightweight S3 API endpoint that can abstract the storage and retrieval of data from multiple cloud object stores.
- On-premises object storage features a robust S3 API endpoint that scales to tens of petabytes and billions of objects, primarily targeting data-intensive applications. Examples include storing and accessing row, columnar, and semi-structured data with applications like Spark, Presto, Red Hat AMQ Streams (Kafka), and even machine learning frameworks like TensorFlow and Pytorch.

Red Hat OpenShift Data Foundation version 4.xx integrates a collection of software projects, including:

- **Ceph** provides block storage, a shared and distributed file system, and on-premises object storage.
- **Ceph CSI**, to manage provisioning and lifecycle of persistent volumes and claims.
- **NooBaa** provides a Multicloud Object Gateway.
- **OpenShift Data Foundation,** Rook-Ceph, and NooBaa operators initialise and manage OpenShift Data Foundation services.

OpenShift Data Foundation Architecture

Red Hat OpenShift Data Foundation provides services for and can run internally from the Red Hat OpenShift Container Platform. The following diagram provides the high-level architecture of Redhat Openshift Data Foundation:



OpenShift Data Foundation Operators

Red Hat OpenShift Data Foundation comprises three leading operators, codifying administrative tasks and custom resources to automate task and resource characteristics easily. Administrators define the desired end state of the cluster, and the OpenShift Data Foundation operators ensure the cluster is either in that state, or approaching that state with minimal administrator intervention.

OpenShift Data Foundation operator

A meta-operator that codifies and enforces the recommendations and requirements of a supported Red Hat OpenShift Data Foundation deployment by drawing on other operators in specific, tested ways. This Operator provides the storage cluster resource that wraps resources supplied by the Rook-Ceph and NooBaa operators.

Rook-Ceph Operator

This Operator automates the packaging, deployment, management, upgrading, and scaling of persistent storage and file, block, and object services. It creates block and file storage classes for all environments and creates an object storage class and services object bucket claims made against it in on-premises environments.

Additionally, for internal mode clusters, it provides the Ceph cluster resource, which manages the deployments and services representing the following:

- Object storage daemons (OSDs)
- Monitors (MONs)
- Manager (MGR)
- Metadata servers (MDS)
- Object gateways (RGW) on-premises only

MCG operator

This Operator automates the packaging, deployment, management, upgrading, and scaling of the Multicloud Object Gateway object service. It creates an object storage class and services object bucket claims made against it.

Additionally, it provides the NooBaa cluster resource, which manages the deployments and services for the NooBaa core, database, and endpoint.

OpenShift Data Foundation Deployment Approach

Red Hat Openshift Data Foundation will be deployed within the Openshift Container Platform using the internal approach. This approach follows the operator-based deployment and management using a local storage operator and local storage devices

There are two different deployment modalities available when Openshift Data Foundation is running entirely with Openshift Container platform:

- Simple
- Optimised

We will follow the optimised modality rather than the simple one because we have dedicated infra nodes for OpenShift Data Foundation services.

OpenShift Data Foundation Security Considerations

The Federal Information Processing Standard Publication 140-2 (FIPS-140-2) is a standard defining a set of security requirements for the use of cryptographic modules. Red Hat OpenShift Data Foundation uses FIPS-validated cryptographic modules delivered by Red Hat Enterprise Linux OS/CoreOS (RHCOS).

FIPS mode must be enabled on the OpenShift Container Platform before installing OpenShift Data Foundation. OpenShift Container Platform must run on RHCOS nodes, as OpenShift Data Foundation deployment on RHEL 7 is not supported for this feature.

OpenShift Data Foundation Infrastructure Requirements

Disks greater than 4TB will need a support exception (SE) from Red Hat. Using such a disk is not tested; it is a function that is still a tech preview feature, and that is not within the regular support limits of Red Hat.

The device requirement and storage class are given below using a disk of 4TB:

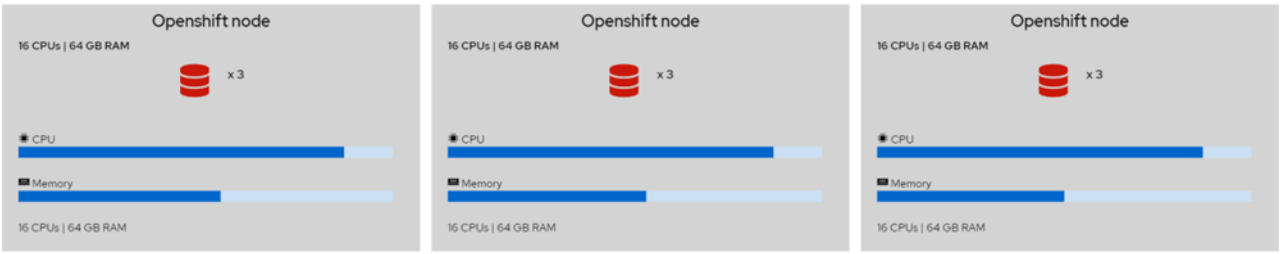
- DirectPath storage devices via the Local Storage Operator.

The node layout diagram below shows how to reach the target capacity of 12 TB. In summary:

3 OCP nodes will run ODF services. (NOTE: OCP clusters often contain additional OCP worker nodes which do not run ODF services.). Each OCP node running ODF services has:

- 16 CPU units, 64 GB memory, the disk size is 4 TB, and the number of disks is three (3).

The ODF deployment type is internal. With this target capacity, you can use up to 9.00 TB before receiving a capacity alert and up to 10.20 TB before the cluster goes into read-only mode.



With the above sizing, the Infra nodes can have five (5) additional disks, making a total usable capacity of 32TB without the scale of infra nodes. Any requirements more than this will need an extra set of nodes (3) to be deployed. Expanding the cluster into multiples of three (3) nodes, one node in each failure domain, is an easy way to satisfy pod placement rules for RHODF.

Capacity alerts are issued when cluster storage capacity reaches 75% (near-full) and 85% (full) of total capacity.

Networking in OpenShift

By default, Kubernetes allocates each pod an internal IP address for applications running within the pod. Pods and their containers can network, but clients outside the cluster do not have networking access. It ensures all containers within the pod behave as if they were on the same host. When you expose your application to external traffic, giving each pod its IP address means that pods can be

treated like physical hosts or virtual machines in terms of port allocation, networking, naming, service discovery, load balancing, application configuration, and migration.

OpenShift platform Network Types

1. **MachineCIDR Subnet:**

    - This Subnet is for Cluster Nodes .
    - Cluster Nodes IPs allocation is through DHCP.
    - The Subnet is routed.
    - It Includes Kubernetes API (API load balancer) (api.<cluster_name>.<base_domain>.) (api- int.<cluster_name>.<base_domain>.).
    - It Includes Routes (*. apps.<cluster_name>.<base_domain>.) (Application ingress load balancer).
    - Including Nodes Management IPs and the VIPs, and Should be accessible to the customer Operations team.
    - Cluster Nodes IPs should have internet access.

2. **ClusterNetwork subnet:**

    - This Subnet is for PODs IP.
    - Used for internal cluster PODs communication.
    - Managed by the OCP.
    - This Subnet is Not Routed.
    - The OpenShift performs PODs IP assignment.

3. **ServiceNetwork Subnet:**

    - This Subnet is for Cluster Services.
    - The Subnet is not routed.
    - This Subnet has the IP address of Service type ClusterIP.

Openshift CNI overview

OpenShift Container Platform offers two supported choices, OpenShift SDN and OVN-Kubernetes, for the default Container Network Interface (CNI) network provider.

OpenShift Container Platform Registry

OpenShift container Platform has an integrated registry and can be combined with third-party registries. The below sections describe both options in more detail.

Integrated OpenShift Container Platform Registry

OpenShift Container Platform provides a built-in container image registry running as a standard cluster workload. The registry is configured and managed by an infrastructure Operator. It gives an out-of-the-box solution for users to control the images that run their workloads, and this solution runs on top of the existing cluster infrastructure. This registry can be scaled up or down like any other cluster workload and does not require specific infrastructure provisioning. In addition, it is integrated into the cluster user authentication and authorisation system, which means that access to create and retrieve images is controlled by defining user permissions on the image resources.

The registry is typically used as a publication target for images built on the cluster and as a source of images for workloads running on the cluster. When a new image is pushed to the registry, the cluster is notified of the new image and other components can react to and consume the updated image.

Image data is stored in two locations. The image data is stored in a configurable storage location, such as cloud storage or a filesystem volume. The image metadata, exposed by the standard cluster APIs and used to perform access control, is stored as common API resources, specifically images and image streams.

In this project, integrated OpenShift container platform Registry will deployed and used. The file storage is recommended to be present for the registry setup to be highly available.

Image registry storage configuration

The Image Registry Operator is not initially available for platforms that do not provide default storage. After installation, configure the registry to use storage to make the Registry Operator available.

Prerequisites:

- Cluster administrator permissions.
- A cluster deployed.
- Persistent storage is provisioned for your cluster.

OpenShift Container Platform supports ReadWriteOnce access for image registry storage when you have only one replica. Deploying an image registry supporting high availability with two or more replicas requires ReadWriteMany access.

Third-party registries

OpenShift Container Platform can create containers using images from third-party registries. Still, these registries will likely offer a different image notification support than the integrated OpenShift Container Platform registry. In this situation, the OpenShift Container Platform will fetch tags from the remote registry upon imagestream creation.

Refreshing the fetched tags is as simple as running the command "oc import-image <stream>". The previously described build and deployment reactions occur when new images are detected.

OCP Monitoring

Monitoring overview

OpenShift Container Platform includes a preconfigured, preinstalled, and self-updating monitoring stack that monitors core platform components. You also have the option to enable monitoring for user-defined projects.

A cluster administrator can configure the monitoring stack with the supported configurations. OpenShift Container Platform delivers monitoring best practices out of the box.

The OpenShift Container Platform web console lets you view and manage metrics and alerts and review monitoring dashboards. OpenShift Container Platform provides access to third-party interfaces like Prometheus, Alertmanager, and Grafana.

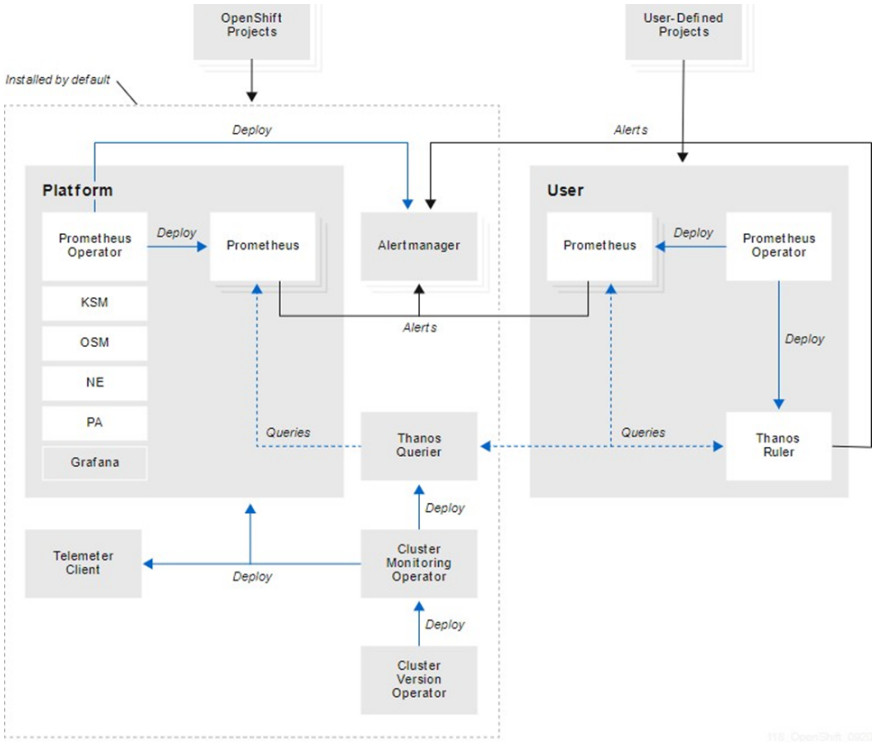
After installing OpenShift Container Platform 4.10, cluster administrators can optionally enable monitoring for user-defined projects. This feature lets cluster administrators, developers, and other users specify how their projects' services and pods are monitored.

Monitoring stack Components

The OpenShift Container Platform monitoring stack is based on the Prometheus open-source project and its broader ecosystem. The monitoring stack includes the following:

- Default platform monitoring components: A set of platforms monitoring components are installed in the openshift-monitoring project by default during an OpenShift Container Platform installation. It provides monitoring for core OpenShift Container Platform components, including Kubernetes services. The default monitoring stack also enables remote health monitoring for clusters. The following diagram illustrates These components in the Installed by default section.
- Components for monitoring user-defined projects: After optionally enabling monitoring for user-defined projects, additional monitoring components are installed in the "openshift-user-workload-monitoring" project. It provides monitoring for user-defined projects.

These components are illustrated in the User section in the following diagram:



Default monitoring components

By default, the OpenShift Container Platform 4.10 monitoring stack includes these components:

COMPONENTS	DESCRIPTION
Cluster Monitoring Operator	The Cluster Monitoring Operator (CMO) is a central component of the monitoring stack. It deploys and manages Prometheus instances, the Thanos Querier, the Telemeter Client, and metrics targets, ensuring they are current. The CMO is deployed by the Cluster Version Operator (CVO).
Prometheus Operator	The Prometheus Operator (PO) in the openshift-monitoring project creates, configures, and manages platform Prometheus instances and Alertmanager instances. It also automatically generates monitoring target configurations based on Kubernetes label queries.
Prometheus	Prometheus is the monitoring system on which the OpenShift Container Platform monitoring stack is based. Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.
Prometheus Adapter	The Prometheus Adapter (PA in the preceding diagram) translates Kubernetes node and pod queries for use in Prometheus. The resource metrics that are translated include CPU and memory utilisation metrics. The Prometheus Adapter exposes the cluster resource metrics API for horizontal

	pod autoscaling. The oc adm top nodes and oc adm top pods commands also use the Prometheus Adapter.
Alertmanager	The Alertmanager service handles alerts received from Prometheus. Alertmanager is also responsible for sending the alerts to external notification systems.
kube-state-metrics agent	The kube-state-metrics exporter agent (KSM in the preceding diagram) converts Kubernetes objects to metrics, which Prometheus can use.
openshift-state-metrics agent	The openshift-state-metrics exporter (OSM in the preceding diagram) expands upon kube-state-metrics by adding metrics for OpenShift Container Platform-specific resources.
node- exporter agent	The node-exporter agent (NE in the preceding diagram) collects metrics about every node in a cluster. The node-exporter agent is deployed on every node.
Thanos Querier	The Thanos Querier aggregates and optionally deduplicates core OpenShift Container Platform metrics and metrics for user-defined projects under a single, multi-tenant interface.
Grafana	The Grafana analytics platform provides dashboards for analysing and visualising the metrics. The Grafana instance that is provided with the monitoring stack, along with its dashboards, is read-only.
Telemeter Client	The Telemeter Client sends a subsection of the data from platform Prometheus instances to Red Hat to facilitate Remote Health Monitoring for clusters.

The stack monitors all components in the monitoring stack and is automatically updated when the OpenShift Container Platform is updated.

Default monitoring targets

In addition to the components of the stack itself, the default monitoring stack monitors:

- CoreDNS
- Elasticsearch (if logging is installed)

- etcd
- Fluentd (if logging is installed)
- HAProxy
- Image registry
- Kubelet
- Kubernetes API server
- Kubernetes controller manager
- Kubernetes scheduler
- Metering (if Metering is installed)
- OpenShift API server
- OpenShift Controller Manager
- Operator Lifecycle Manager (OLM)

Components for monitoring user-defined projects

OpenShift Container Platform 4.13 includes an optional enhancement to the monitoring stack that enables you to monitor services and pods in user-defined projects. This feature consists of the following components.

Monitoring targets for user-defined projects

COMPONENT	DESCRIPTION
Prometheus Operator	The Prometheus Operator (PO) in the openshift-user-workload-monitoring project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.
Prometheus	Prometheus is the monitoring system through which monitoring is provided for user-defined projects. Prometheus sends alerts to Alertmanager for processing.
Thanos Ruler	The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Container Platform 4.10, Thanos Ruler provides rules and alerting evaluation for monitoring user-defined projects.

When monitoring is enabled for user-defined projects, you can monitor:

- Metrics are provided through service endpoints in user-defined tasks.
- Pods running in user-defined tasks.

Configuring alert notifications

In the OpenShift Container Platform, an alert is fired when the conditions defined in an alerting rule are actual. An alert provides a notification that circumstances are apparent within a cluster. Firing alerts can be viewed in the Alerting UI in the OpenShift Container Platform web console by default.

After installation, you can configure the OpenShift Container Platform to send alert notifications to external systems.

Sending notifications to external systems

In OpenShift Container Platform 4.10, firing alerts can be viewed in the Alerting UI. Alerts are not configured by default to be sent to any notification systems. You can configure OpenShift Container Platform to send alarms to the following receiver types:

- PagerDuty.
- Webhook.
- Email.
- Slack.

Routing alerts to receivers enables you to send timely notifications to the appropriate teams when failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or a critical response team. Alerts that provide non-critical warning notifications might instead be routed to a ticketing system for non-immediate review.

OCP Logging

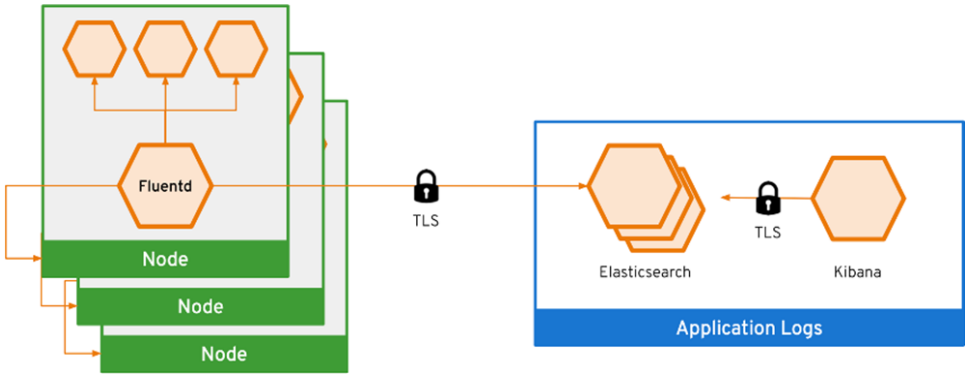
Introduction

OpenShift Logging aggregates all the logs from the OpenShift Container Platform cluster, such as node system audit logs, application container logs, and infrastructure logs. OpenShift Logging aggregates these logs throughout the cluster and stores them in a default log store. The Kibana web console is used to visualise log data.

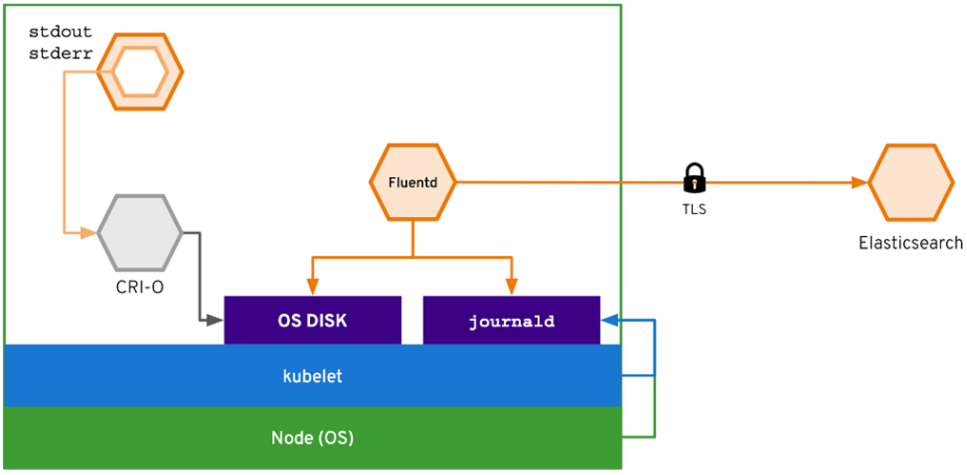
Types of Logs

OpenShift Logging aggregates the following types of logs:

- Application: these are container logs generated by user applications running in the cluster, except infrastructure container applications.



- Infrastructure: These are logs generated by infrastructure components running in the cluster and OpenShift Container Platform nodes, such as journal logs. Infrastructure components are pods that run in the OpenShift*, kube*, or default projects.



- Audit - Logs generated by auditd, the node audit system, stored in the /var/log/audit/audit.log file, and the audit logs from the Kubernetes apiserver and the OpenShift apiserver.

OpenShift Logging Components

The OpenShift Logging components include a collector deployed to each node in the OpenShift Container Platform cluster that collects all node and container logs and writes them to a log store. You can use a centralised web UI to create rich visualisations and dashboards with the aggregated data.

The significant components of OpenShift Logging are:

- Collection - This component collects logs from the cluster, formats them, and forwards them to the log store. The current implementation is Fluentd.
- Log store - This is where the logs are stored. The default implementation is Elasticsearch. You can use the default Elasticsearch log store or forward logs to external log stores. The default log store is optimised and tested for short-term storage.
    - From the documentation, the OpenShift Container Platform uses Elasticsearch to store log data. This data is stored for seven (7) days max. If you want to extend it, it must be sent to a third party.
    - Regarding the local elastic store, as mentioned in the first note: "Because the internal OpenShift Container Platform Elasticsearch log store does not provide secure storage for audit logs, audit logs are not stored in the internal Elasticsearch instance by default. If you want to send the audit logs to the default internal Elasticsearch log store, for example, to view the audit logs in Kibana, you must use the Log Forwarding API."
- Visualisation - This is the UI component you can use to view logs, graphs, charts, and so forth. The current implementation is Kibana.

Logging Collector (Fluentd)

OpenShift Container Platform uses Fluentd to collect container and node logs. By default, the log collector uses the following sources:

- journald for all system logs
- /var/log/containers/*.log for all container logs

If the log collector is configured to collect audit logs, it gets them from /var/log/audit/audit.log.

The logging collector is a daemon set that deploys pods to each OpenShift Container Platform node. System and infrastructure logs are generated by journald log messages from the operating system, the container runtime, and the OpenShift Container Platform.

The CRI-O container engine generates application logs. Fluentd collects the logs from these sources and forwards them internally or externally as configured in the OpenShift Container Platform.

Log Store (Elasticsearch)

By default, the OpenShift Container Platform uses Elasticsearch (ES) to store log data. Optionally, you can use the log forwarding features to forward logs to external log stores using Fluentd protocols, Syslog protocols, or the OpenShift Container Platform Log Forwarding API.

The OpenShift Logging Elasticsearch instance is optimised and tested for short-term storage for approximately seven days.

Elasticsearch organises the log data from Fluentd into datastores, or indices, then subdivides each index into multiple pieces called shards, which spread across a set of Elasticsearch nodes in an Elasticsearch cluster.

Elasticsearch can be configured to make copies of the shards, called replicas, which Elasticsearch also spreads across the Elasticsearch nodes. The ClusterLogging custom resource (CR) allows you to specify how the shards are replicated to provide data redundancy and resilience to failure. You can also choose how long the different types of logs are retained using a retention policy in the ClusterLogging CR.

Logging Visualisation (Kibana)

OpenShift Container Platform uses Kibana to display the log data collected by Fluentd and indexed by Elasticsearch.

Kibana is a browser-based console interface to query, discover, and visualise your Elasticsearch data through histograms, line graphs, pie charts, and other visualisations.

Forwarding logs to external logging systems

By default, OpenShift Logging sends container and infrastructure logs to the default internal Elasticsearch log store defined in the ClusterLogging custom resource. However, it does not send audit logs to the internal store because it does not provide secure storage.

To send logs to other log aggregators, OpenShift Container Platform Cluster Log Forwarder can be used. This API enables sending container, infrastructure, and audit logs to specific endpoints within or outside your cluster. In addition, it is possible to send different types of logs to various systems so that multiple individuals can access each type. It is also possible to enable Transport Layer Security (TLS) support to send logs securely.

When forwarding logs externally, the Red Hat OpenShift Logging Operator creates or modifies a Fluentd config map to send logs using your desired protocols. You are responsible for configuring the protocol on the external log aggregator.

Forwarding cluster logs to external third-party systems requires a combination of outputs and pipelines specified in a ClusterLogForwarder custom resource (CR) to send logs to specific endpoints inside and outside your OpenShift Container Platform cluster. You can also use inputs to forward the application logs associated with a particular project to an endpoint.

If the output URL scheme requires TLS (HTTPS, TLS, or UDPS), then TLS server-side authentication is enabled. In order to allow for client authentication, the output must name a secret in the OpenShift-loggingproject. The secret must have tls.crt, tls.key, and ca-bundle.crt that point to the respective certificates they represent.

- An output is the destination for log data you define or where you want the logs sent. An outcome can be one of the following types:
    - elasticsearch. An external Elasticsearch instance. The elasticsearch output can use a TLS connection.
    - fluentdForward. An external log aggregation solution that supports Fluentd. This option uses the Fluentd forward protocols. The fluentForward result can use a TCP or TLS connection and supports shared-key authentication by providing a

shared_key field in a secret. Shared-key authentication can be used with or without TLS.
    - Syslog. An external log aggregation solution that supports the syslog RFC3164 or RFC5424 protocols. The syslog output can use a UDP, TCP, or TLS connection. (This method is used as per YOU TELECOM CIP Logging Design).
    - cloudwatch. Amazon CloudWatch is a monitoring and log storage service hosted by Amazon Web Services (AWS).
    - Loki. Loki is a horizontally scalable, highly available, multi-tenant log aggregation system.
    - Kafka. A Kafka broker. The Kafka output can use a TCP or TLS connection.
    - Default. The internal OpenShift Container Platform Elasticsearch instance. You are not required to configure the default output. If you configure a default output, you receive an error message because the default output is reserved for the Red Hat OpenShift Logging Operator.
- A pipeline defines simple routing from one log type to one or more outputs or which logs you want to send. The log types are one of the following:
    - Application. Container logs are generated by user applications running in the cluster, except infrastructure container applications.
    - Infrastructure. Container logs from pods that run in the OpenShift*, kube*, or default projects and journal logs sourced from node file system
    - audit. The node audit system, auditd, Kubernetes API server, OpenShift API server, and OVN network generate audit logs.

Labels can be added to outbound log messages using key-value pairs in the pipeline. For example, add a label to messages forwarded to other data centres or label the logs by type. Labels that are added to objects are also forwarded with the log message.

An input forwards the application logs associated with a specific project to a pipeline. In the pipeline, you define which log types to forward using an inputRef parameter and where to forward the logs to using an outputRef parameter.

Logs Types – Summary

LOG TYPE	DESCRIPTION	SOURCE	FREQUENCY & TRIGGERS
Infrastructure	Logs generated by infrastructure components running in the cluster and OpenShift Container Platform nodes, such as journal logs. Infrastructure components are pods that run in the OpenShift *, kube*, or default projects.	Systemd-journald /var/log/journal/	Configurable recommended setting RateLimitIntervalSec=30s and RateLimitBurst=10000 (e.g., 1000 messages within 30 sec) https://docs.openshift.com/container-platform/4.9/logging/config/cluster- logging-systemd.html

Application	Container logs generated by user applications running in the cluster, except infrastructure container applications. The CRI-O container engine generates application logs.	/var/log/containers/*.log	Same as Infrastructure.
System Audit	Logs generated by auditd, the node audit system, and the audit logs from the Kubernetes apiserver and the OpenShift apiserver.	/var/log/audit/audit.log / path=kube- apiserver/<log_name> path=openshift- apiserver/<log_name>	Default - Logs only metadata for read and write requests; does not log request bodies except for OAuth access token requests. It is the default policy.

Logs forwarding

To configure Red Hat OpenShift to forward logs from the ClusterLogging instance to an external 3rd party system, in this case, using ClusterLogForwarder.

This API lets you send container, infrastructure, and audit logs to specific endpoints within or outside your cluster. In addition, you can send different types of logs to various systems so that multiple individuals can access each type.

For more information, see https://docs.openshift.com/container-platform/4.13/logging/cluster-logging-external.html.

OCP Control Plane Backup & Restore

Backup and Restore Overview

As a cluster administrator, you should stop an OpenShift Container Platform cluster for a period and restart it later. Some reasons for restarting a cluster are that you need to perform maintenance on a cluster or want to reduce resource costs. In the OpenShift Container Platform, you can perform a graceful shutdown of a cluster to restart the cluster later quickly.

You must back up etcd data before shutting down a cluster; etcd is the key-value store for the OpenShift Container Platform, which persists in the state of all resource objects. An etcd backup plays a crucial role in disaster recovery. In the OpenShift Container Platform, you can replace an unhealthy etcd member.

You might run into several situations where the OpenShift Container Platform does not work as expected, such as:

- You have a cluster that is not functional after the restart because of unexpected conditions, such as node failure or network connectivity issues.
- You have deleted something critical in the cluster by mistake.
- You have lost the majority of your control plane hosts, leading to etcd quorum loss.

Backing up etcd

etcd is the key-value store for the OpenShift Container Platform, which persists in the state of all resource objects.

Back up your cluster's etcd data regularly and store it in a secure location, ideally outside the OpenShift Container Platform environment. Do not take an etcd backup before the first certificate rotation completes, which occurs 24 hours after installation; otherwise, the backup will contain expired certificates. It is also recommended to take etcd backups during non-peak usage hours because the etcd snapshot has a high I/O cost.

Be sure to take an etcd backup after you upgrade your cluster. It is essential because when you restore your cluster, you must use an etcd backup from the same z-stream release. For example, an OpenShift Container Platform 4.y.z cluster must use an etcd backup from 4.y.z.

After Backing up etcd, two files are created in the /home/core/assets/backup/ directory on the control plane host:

- "snapshot_<datetimestamp>.db: this file Is the etcd snapshot. The "cluster- backup.sh script confirms Its validity.
- static_kuberesources_<datetimestamp>.tar.gz: This file contains the resources for the static pods. If etcd encryption is enabled, it also contains the encryption keys for the etcd snapshot.

Control Plane Restore

To restore the cluster to a previous state, you must have previously backed up etcd data by creating a snapshot. You will use this snapshot to restore the cluster state.

ETCD backup is used to restore the cluster to a previous state. It can be used to recover from the following situations:

- The machine is not running, or the node is not ready
- The etcd pod is crash looping.

Replacement of unhealthy controller nodes (UPI Mechanism)

For Replacing an unhealthy etcd member whose machine is not running or whose node is not ready.

Perquisites:

- Before starting the node replacement process, Take an etcd backup before replacing an unhealthy etcd member.
- Shut Down the controller node, which needs replacement, before starting with the above-mentioned procedure.
- For re-provisioning the controller node, Boot the new machine using the master ignition file.
- Ensure all the changes related to disk, storage or network are done before booting the new controller node.
- Once the master boots up, approve the pending CSRs(if any) and let the master come in a Ready state.

Replacing an unhealthy etcd member whose etcd pod is crash looping

Prerequisites

- You have identified the unhealthy etcd member.
- You have verified that the etcd pod is crashlooping.
- You have access to the cluster as a user with the cluster-admin role.
- You have taken an etcd backup.

Failure of All Cluster Controller Nodes

If all controller nodes are down, the control plane will be impacted while the data plane might still work.

- No OpenShift API calls will be returned because both the api-server and controller manager will be in a failed state.
- etcd won't be available, so oc command will not return any output, and the web console won't be able to propagate the data.

- The cluster cannot create new resources, move pods to further nodes, etc, until the controller node returns online.
- Infra nodes will be fine, and the router can reach application routes.
- Application deployed on worker nodes will only be impacted if the horizontal pod autoscaling controller (HPA) triggers more replicas of the application.

The life for applications will continue as normal unless nodes are rebooted, or there is a dramatic failure during this time.

Application Backup

The dynamic nature of Kubernetes environments makes it harder for more traditional backup systems and techniques to work well in the context of Kubernetes nodes and applications. Both recovery time and recovery point are more strict since applications must constantly be up and running. Backup and restore involves backing up the entire application offsite from a local Kubernetes cluster. For example, the backup software needs to understand what is part of the OpenShift app.

- App configuration
- Kubernetes resources
- Data

The backup tool needs to save all of the above as a single resource for it to be helpful for restoration in case of failure in the cluster. Also, backup software should be able to select specific resources, such as particular applications, specific groups of applications, and the entire Kubernetes namespace.

OCP Security & Compliance

Container security

Securing a containerised application relies on multiple levels of security:

- Container security begins with a trusted base container image and continues through the container build process as it moves through your CI/CD pipeline.
- When a container is deployed, its security depends on it running on secure operating systems and networks and establishing firm boundaries between the container itself and the users and hosts that interact with it.
- Continued security relies on being able to scan container images for vulnerabilities and having an efficient way to correct and replace vulnerable images.

Securing containers on RHCoreOS

Red Hat CoreOS is a Red Hat Enterprise Linux (RHEL) version specially configured to work as a controller node and worker node on OpenShift Container Platform clusters. So RHCOS is tuned to efficiently run container workloads, along with Kubernetes and OpenShift Container Platform services.

Containers simplify deploying many applications to run on the same host, using the same kernel and container runtime to spin up each container. Many users can own the applications and, because they are kept separate, can run different, and even incompatible, versions of those applications simultaneously without issue.

In Linux, containers are just a process, so securing containers is similar to ensuring any other running process. An environment for running containers starts with an operating system that can secure the host kernel from containers and other processes running on the host and secure containers from each other.

Because OpenShift Container Platform 4.10 runs on RHCOS hosts, with the option of using Red Hat Enterprise Linux (RHEL) as worker nodes, the following concepts apply by default to any deployed OpenShift Container Platform cluster. These RHEL security features are at the core of what makes running containers in the OpenShift Container Platform more secure:

- Linux namespaces enable the creation of an abstraction of a particular global system resource to make it appear as a separate instance to processes within a namespace. Consequently, several containers can use the same computing resource simultaneously without conflict. By default, container namespaces from the host include mount table, process table, network interface, user, control group, UTS, and IPC namespaces. Those containers needing direct access to host namespaces need elevated permissions to request that access.
- SELinux provides an additional layer of security to keep containers isolated from each other and the host. SELinux allows administrators to enforce mandatory access controls (MAC) for every user, application, process, and file.
- CGroups (control groups) limit, account for, and isolate a collection of processes' resource usage (CPU, memory, disk I/O, network, etc.). CGroups ensure that containers on the same host are not impacted by each other.
- Secure computing mode (seccomp) profiles can be associated with a container to restrict available system calls.
- Deploying containers using RHCOS reduces the attack surface by minimising and tuning the host environment for containers. The CRI-O container engine further reduces that attack surface by implementing only those features that the Kubernetes and OpenShift Container Platform require to run and manage containers, unlike other container engines that implement desktop-oriented standalone features.

Hardening RHCoreOS

RHCOS was created and tuned to be deployed in the OpenShift Container Platform with a few changes to RHCOS nodes. Every organisation adopting the OpenShift Container Platform has its

requirements for system hardening. As a RHEL system with OpenShift-specific modifications and features added (such as Ignition, ostree, and a read-only /usr to provide limited immutability), RHCOS can be hardened just as you would any RHEL system. Differences lie in the ways you manage the hardening.

A vital feature of the OpenShift Container Platform and its Kubernetes engine is quickly scaling applications and infrastructure up and down as needed. Unless unavoidable, you do not want to make direct changes to RHCOS by logging into a host and adding software or changing settings. You want to have the OpenShift Container Platform installer and control plane manage changes to RHCOS so new nodes can be spun up without manual intervention.

Choosing what to harden in RHCoreOS

The RHEL 8 Security Hardening The guide describes how to approach security for any RHEL system.

Use this guide to learn how to approach cryptography, evaluate vulnerabilities, and assess threats to various services. Likewise, you can learn to scan for compliance standards, check file integrity, perform auditing, and encrypt storage devices.

Knowing what features you want to harden, you can decide how to set them in RHCOS.

Choosing how to harden RHCOS

Direct modification of RHCOS systems in the OpenShift Container Platform is discouraged. Instead, it would be best if you thought of modifying systems in pools of nodes, such as worker nodes and control plane nodes. When a new node is needed in non-bare metal installations, you can request a new node of the type you want, and it will be created from an RHCOS image plus the modifications you made earlier.

There are opportunities for modifying RHCOS before, during, and after the cluster is up and running.

Hardening after the cluster is running

After the OpenShift Container Platform cluster is up and running, there are several ways to apply hardening features to RHCOS:

- Machine config: "machineconfig" objects contain a subset of Ignition configs in the same format.

By applying machine configs to all worker or control plane nodes, you can ensure that the next node of the same type added to the cluster has identical changes applied.

All features noted here are described in the OpenShift Container Platform product documentation.

Etcd encryption

By default, etcd data is not encrypted in the OpenShift Container Platform. You can enable etcd encryption for your cluster to provide an additional layer of data security. For example, it can help protect against losing sensitive data if an etcd backup is exposed to the incorrect parties.

When you enable etcd encryption, the following OpenShift API server and Kubernetes API server resources are encrypted:

- Secrets
- Config maps
- Routes
- OAuth access tokens
- OAuth authorise tokens

When you enable etcd encryption, encryption keys are created. These keys are rotated every week. It would be best to have these keys to restore from an etcd backup.

In YOU Telecom design ETCD encryption will be enabled. A cluster admin role is required to allow for encryption for ETCD.

Securing container content

To ensure the security of the content inside your containers, you need to start with trusted base images, such as Red Hat Universal Base Images, and add trusted software. For checking the security of your container images, there are both Red Hat and third-party tools for scanning images.

Security scanning in RHEL

For Red Hat Enterprise Linux (RHEL) systems, OpenSCAP scanning is available from the "openscap-utils" package. In RHEL, you can use the "openscap-podman" command to scan images for vulnerabilities. See Scanning containers and container images for vulnerabilities in the Red Hat Enterprise Linux documentation.

OpenShift Container Platform enables you to leverage RHEL scanners with your CI/CD process. For example, you can integrate static code analysis tools that test for security flaws in your source code and software composition analysis tools that identify open-source libraries to provide metadata on those libraries, such as known vulnerabilities.

Scanning OpenShift images

For the container images running in the OpenShift Container Platform and pulled from Red Hat Quay registries, you can use an Operator to list the vulnerabilities of those images. The Red Hat Quay Container Security Operator can be added to the OpenShift Container Platform to provide vulnerability reporting for images added to selected namespaces.

The Clair security scanner performs container image scanning for Red Hat Quay. In Red Hat Quay, Clair can search for and report vulnerabilities in images built from RHEL, CentOS, Oracle, Alpine, Debian, and Ubuntu operating systems.

Control plane and data plane lockdown

Access to the OpenShift should be limited and only trusted specific IP addresses or subnets should be allowed access to the control and data plane nodes. It must be done outside the cluster firewall.

The following is considered the best practice to be implemented for securing access to the control plane:

- Connection to the cluster will be hardened and only allowed through the Jump/Bastion server.
- The bastion server must be Identity Management (IdM) integrated with AD to provide a centralised and unified way to manage identity stores, authentication, policies, and authorisation policies.
- A firewall will restrict access to the cluster API from Bastion only.

Securing the container platform

OpenShift Container Platform and Kubernetes APIs are crucial to automating container management at scale. APIs are used to:

- Validate and configure the data for pods, services, and replication controllers.
- Perform project validation on incoming requests and invoke triggers on other major system components.

Security-related features in the OpenShift Container Platform that are based on Kubernetes include:

- Multitenancy combines Role-Based Access Controls and network policies to isolate containers at multiple levels.
- Admission plug-ins form boundaries between an API and those making requests to the API.

OpenShift Container Platform uses Operators to automate and simplify the management of Kubernetes-level security features.

Isolating containers with multitenancy

Multitenancy allows applications on an OpenShift Container Platform cluster owned by multiple users and run across various hosts and namespaces to remain isolated from each other and outside attacks. You obtain multitenancy by applying role-based access control (RBAC) to Kubernetes namespaces.

In Kubernetes, namespaces are areas where applications can run in ways that are separate from other applications. OpenShift Container Platform uses and extends namespaces by adding extra annotations, including MCS labelling in SELinux, and identifying these extended namespaces as projects. Within the scope of a project, users can maintain their cluster resources, including service accounts, policies, constraints, and various other objects.

RBAC objects are assigned to projects to authorise selected users to access those projects. That authorisation takes the form of rules, roles, and bindings:

- Rules define what a user can create or access in a project.
- Roles are collections of rules and directions you can bind to selected users or groups.
- Bindings define the association between users or groups and roles.

Local RBAC roles and bindings attach a user or group to a project. Cluster RBAC can attach cluster-wide roles and bindings to all projects in a cluster. There are default cluster roles that can be assigned to provide "admin", "basic user", cluster-admin", and "cluster-status" access.

Protecting control plane with admission plug-ins

While RBAC controls access rules between users, groups, and available projects, admission plug-ins define access to the OpenShift Container Platform master API. Admission plug-ins form a chain of rules that consist of:

- Default admissions plug-ins: These implement a default set of policies and resource limits applied to components of the OpenShift Container Platform control plane.
- Mutating admission plug-ins: These plug-ins dynamically extend the admission chain. They call out to a webhook server and can both authenticate a request and modify the selected resource.
- Validating admission plug-ins: These validate requests for a selected resource and can validate the request and ensure that the resource does not change again.

API requests go through admissions plug-ins in a chain, with any failure in the path causing the request to be rejected. Each admission plug-in is associated with particular resources and only responds to requests for those resources.

Security context constraints (SCCs)

You can use security context constraints (SCCs) to define a set of conditions a pod must run with to be accepted into the system.

Some aspects that can be managed by SCCs include:

- Running of privileged containers
- Capabilities a container can request to be added
- Use of host directories as volumes
- SELinux context of the container
- Container user ID

If you have the required permissions, you can adjust the default SCC policies to be more permissive.

Granting roles to service accounts

You can assign roles to service accounts in the same way that users are given role-based access. There are three default service accounts created for each project.

A service account:

- It is limited in scope to a particular project.
- It derives its name from its project.
- It is automatically assigned an API token and credentials to access the OpenShift Container Registry.

Service accounts associated with platform components automatically have their keys rotated.

Authentication and authorisation

Controlling access using OAuth

You can use API access control via authentication and authorisation for securing your container platform. The OpenShift Container Platform master includes a built-in OAuth server. Users can obtain OAuth access tokens to authenticate themselves to the API.

OpenShift OAuth server can be configured to use many identity providers. The following lists include the more common ones:

- HTPasswd: Validates user names and passwords against a secret that stores credentials generated using the htpasswd command.
- Keystone: Enables shared authentication with an OpenStack Keystone v3 server. LDAP Configures the LDAP identity provider to validate user names and passwords against an LDAPv3 server using simple bind authentication.
- GitHub or GitHub Enterprise Configures a GitHub identity provider to validate user names and passwords against GitHub or the GitHub Enterprises OAuth authentication server.
- OpenID: Connect Integrates with an OpenID Connect identity provider using an Authorization Code Flow

The OAuth custom resource must be updated with your desired identity provider. We can define multiple identity providers of the same or different kinds on the same OAuth custom resource. In the YOU Telecom environment, the LDAP server will be used as an identity provider to authenticate the YOU Telecom users to the OpenShift.

By configuring LDAP authentication and role-based access control for your Red Hat OpenShift cluster, you limit access to critical resources and separate duties and allow for auditability through named access.

The integration with Active Directory involves two steps.

- Configure an LDAP Custom Resource for the Red Hat OpenShift Cluster. The LDAP custom resource allows the cluster OAuth instance to leverage your Active Directory LDAP as an authentication mechanism.

- Configure LDAP Group Sync for the Red Hat OpenShift Cluster. Performing LDAP group sync is necessary to assign user cluster roles by using groups and not assigning permissions per user. A sync job automatically syncs the Users/Groups on a set schedule.

API access control and management

Applications can have multiple independent API services with different endpoints requiring management. OpenShift Container Platform includes a containerised version of the 3scale API gateway so that you can manage your APIs and control access.

3scale gives you a variety of standard API authentication and security options, which can be used alone or in combination to issue credentials and control access: standard API keys, application ID and key pair, and OAuth 2.0.

You can restrict access to specific endpoints, methods, and services and apply access policy for groups of users. Application plans allow you to set rate limits for API usage and control traffic flow for groups of developers.

Red Hat Single Sign-On

The Red Hat Single Sign-On server enables you to secure your applications by providing web single sign-on capabilities based on standards, including SAML 2.0, OpenID Connect, and OAuth 2.0. The server can act as a SAML or OpenID Connect–based identity provider (IdP), mediating with your enterprise user directory or third-party identity provider for identity information and your applications using standards-based tokens. You can integrate Red Hat Single Sign-On with LDAP-based directory services, including Microsoft Active Directory and Red Hat Enterprise Linux Identity Management.

Securing self-service web console

OpenShift Container Platform provides a self-service web console to ensure teams cannot access other environments without authorisation. OpenShift Container Platform ensures a secure multi-tenant master by providing the following:

- Access to the master uses Transport Layer Security (TLS).
- Access to the API Server uses X.509 certificates or OAuth access tokens.
- Project quota limits the damage that a rogue token could do.
- The etcd service is not exposed directly to the cluster.

Managing certificates for the platform

OpenShift Container Platform has multiple components within its framework that use REST-based HTTPS communication leveraging encryption via TLS certificates. OpenShift Container Platform's installer configures these certificates during installation. Some primary features generate this traffic:

- masters (API server and controllers)
- etcd
- nodes
- registry
- router

Configuring custom certificates

You can configure custom serving certificates for the public hostnames of the API server and web console during initial installation or when redeploying certificates. You can also use a custom CA.

Securing networks

Network security can be managed at several levels. At the pod level, network namespaces can prevent containers from seeing other pods or the host system by restricting network access. Network policies give you control over allowing and rejecting connections. You can manage ingress and egress traffic to and from your containerised applications.

Using network namespaces

OpenShift Container Platform uses software-defined networking (SDN) to provide a unified cluster network that enables container communication across the cluster.

Network policy mode, by default, makes all pods in a project accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create "NetworkPolicy" objects to indicate the allowed incoming connections. Using multi-tenant mode, you can provide project-level isolation for pods and services.

Isolating pods with network policies

Using network policies, you can isolate pods from each other in the same project. Network policies can deny all network access to a pod, only allow connections for the ingress controller, reject connections from pods in other projects, or set similar rules for how networks behave.

For additional resources, please see https://docs.openshift.com/container-platform/4.13/networking/network_policy/about-network-policy.html#about-network-policy.

Isolating applications

OpenShift Container Platform enables you to segment network traffic on a single cluster to make multi-tenant clusters that isolate users, teams, applications, and environments from non-global resources.

Securing ingress traffic

There are many security implications for configuring access to your Kubernetes services from outside your OpenShift Container Platform cluster. Besides exposing HTTP and HTTPS routes, ingress routing allows you to set up NodePort or LoadBalancer ingress types. NodePort exposes an application's service API object from each cluster worker. LoadBalancer lets you assign an external load balancer to an associated service API object in your OpenShift Container Platform cluster.

In YOU Telecom, HTTP ingress traffic will be blocked as best practice. Also, firewall rules should be implemented on external firewalls to allow only the needed access to the application hosted on the OpenShift.

The following steps will turn HTTP (port 80) traffic off on the OpenShift ingress.

Securing egress traffic

OpenShift Container Platform provides the ability to control egress traffic using either a router or firewall method. For example, you can use IP whitelisting to manage database access. A cluster administrator can assign one or more egress IP addresses to a project in an OpenShift Container Platform SDN network provider. Likewise, a cluster administrator can use an egress firewall to prevent egress traffic from going outside an OpenShift Container Platform cluster.

By assigning a fixed egress IP address, you can set all outgoing traffic to that IP address for a particular project. With the egress firewall, you can prevent a pod from connecting to an external network, from connecting to an internal network, or limit a pod's access to specific internal subnets.

Securing attached storage

OpenShift Container Platform supports multiple types of storage, both for on-premise and cloud providers. In particular, the OpenShift Container Platform can use storage types that support the Container Storage Interface.

Persistent volume plug-ins

Containers are helpful for both stateless and stateful applications. Protecting attached storage is a crucial element of securing stateful services. Using the Container Storage Interface (CSI), the OpenShift Container Platform can incorporate storage from any back-end supporting the CSI interface.

- Red Hat OpenShift Container Storage (ODF) *
- OpenStack Cinder *
- VMware vSphere *
- Network File System (NFS)
- FlexVolume
- Fibre Channel
- iSCSI

Plug-ins for those storage types with dynamic provisioning are marked with an asterisk (*). Data in transit is encrypted via HTTPS for all OpenShift Container Platform components communicating with each other.

You can mount a persistent volume (PV) on a host in any way supported by your storage type. Different types of storage have additional capabilities, and each PV's access mode is set to the specific methods supported by that particular volume.

For example, NFS can support multiple read/write clients, but a specific NFS PV might be exported on the server as read-only. Each PV has access modes describing a particular PV's capabilities, such as ReadWriteOnce, ReadOnlyMany, and ReadWriteMany.

Shared storage

For shared storage providers like NFS, the PV registers its group ID (GID) to annotate the PV resource. Then, when the pod claims the PV, the annotated GID is added to the supplemental groups of the pod, giving that pod access to the contents of the shared storage.

Block storage

For block storage providers like AWS Elastic Block Store (EBS), GCE Persistent Disks, and iSCSI, OpenShift Container Platform uses SELinux capabilities to secure the root of the mounted volume for non-privileged pods, making the mounted volume owned by and only visible to the container with which it is associated.

Monitoring cluster events and logs

Monitoring and auditing an OpenShift Container Platform cluster is essential to safeguarding the cluster and its users against inappropriate usage.

Two primary sources of cluster-level information are helpful for this purpose: events and logging.

Cluster administrators are encouraged to familiarise themselves with the Event resource type and review the list of system events to determine which events are of interest. Events are associated with a namespace, either the namespace of the resource they are related to or, for cluster events, the default namespace. The default namespace holds relevant events for monitoring or auditing a cluster, such as node and resource events related to infrastructure components.

Logging

Using the "oc" log command, you can view container logs and build configs and deployments in real time.

Different users can have separate access to logs:

- Users who have access to a project can see the logs for that project by default.
- Users with admin roles can access all container logs.

To save your logs for further audit and analysis, you can enable the cluster login add-on feature to collect, manage, and view system, container, and audit logs. You can deploy, manage, and upgrade OpenShift Logging through the OpenShift Elasticsearch Operator and Red Hat OpenShift Logging Operator.

Audit logs

With audit logs, you can follow a sequence of activities associated with how a user, administrator, or other OpenShift Container Platform component behaves. API audit logging is done on each server.

For additional information, please see:

- List of system events
- Understanding OpenShift Logging
- Viewing audit logs

For more additional information, please see the following:

- List of system events.
- Understanding OpenShift Logging.
- Viewing audit logs

Certificates

Various components use certificates to validate access to the cluster. Administrators can replace the default ingress certificate, add API server certificates, or add a service certificate.

You can also review more details about the types of certificates used by the cluster:

- User-provided certificates for the API server
- Proxy certificates
- Service CA certificates
- Node certificates
- Bootstrap certificates
- etcd certificates
- OLM certificates
- User-provided certificates for default ingress
- Ingress certificates
- Monitoring and cluster logging's Operator component certificates
- Control plane certificates

ANNEX

Bastion/Installer Server

To enable the deployment, a virtual machine is provisioned to run the Red Hat OpenShift installation steps and host an HTTP Server. This virtual machine is known as the bastion node. The bastion node is configured to allow SSH access from the jump-server or remote device.

The bastion node runs Red Hat® Enterprise Linux®, which hosts the scripts, files, and tools to provision the bootstrap, control plane, and compute nodes. After the deployment, keeping the bastion node as an administrative node for the cluster is recommended.

The bastion node setup consists of the following steps:

- Provision of a Red Hat virtual machine.
- Register the Red Hat virtual machine.
- Install NGINX (HTTP Server).
- Generate an SSH private key and add it to the agent.

MACHINE	OS	CPU	RAM	STORAGE	COUNT
Bastion Server	RHEL	8	16 GB	256 GB	1