Chapter 1. Introduction to Kubernetes and OpenShift

Abstract

Goal	Identify the main Kubernetes cluster services and OpenShift platform services and monitor them by using the web console.
Sections	• Containers and Kubernetes (and Matching Quiz) • Red Hat OpenShift Components and Editions (and Matching Quiz) • Navigate the OpenShift Web Console (and Guided Exercise) • Monitor an OpenShift Cluster (and Guided Exercise)
Lab	• Introduction to Kubernetes and OpenShift

Containers and Kubernetes

Objectives

- Describe the main characteristics of containers and Kubernetes.

Introduction to Containers and Kubernetes

Red Hat OpenShift Container Platform (RHOCP) is a complete platform to run applications in clusters of servers. OpenShift is based on existing technologies such as *containers* and Kubernetes. Containers provide a way to package applications and their dependencies that can be executed on any system with a container runtime.

Kubernetes provides many features to build clusters of servers that run containerized applications. However, Kubernetes does not intend to provide a complete solution, but rather provides extension points so system administrators can complete Kubernetes. OpenShift builds on the extension points of Kubernetes to provide a complete platform.

Containers Overview

A *container* is a process that runs an application independently from other containers on the host. Containers are created from a *container image*, which includes all the runtime dependencies of the application. Containers can then be executed on any host, without requiring the installation of any dependency on the host, and without conflicts between the dependencies of different containers.

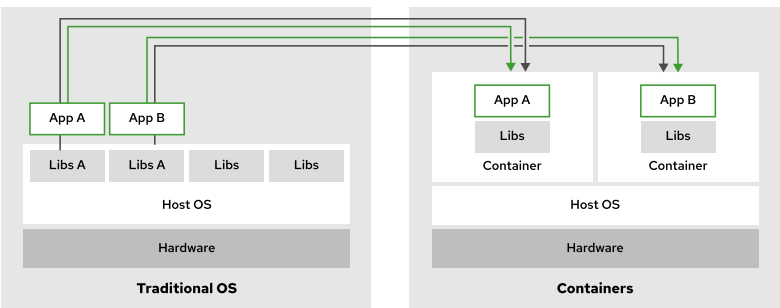

Figure 1.1: Applications in containers versus on the host operating system

Containers use Linux kernel features, such as namespaces and control groups (cgroups). For example, containers use cgroups for resource management, such as CPU time allocation and system memory. Namespaces isolate a container's processes and resources from other containers and the host system.

The Open Container Initiative (OCI) maintains standards about containers, container images, and container runtimes. Because most container engine implementations are designed to conform to the OCI specifications, applications that are packaged according to the specification can run on any conforming platform.

Kubernetes Overview

Kubernetes is a platform that simplifies the deployment, management, and scaling of containerized applications.

You can run containers on any Linux system by using tools such as Podman. Further work is required to create containerized services that, for example, can run across a cluster for redundancy.

Kubernetes creates a cluster that runs applications across multiple nodes. If a node fails, then Kubernetes can restart an application in a different node.

Kubernetes uses a declarative configuration model. Kubernetes administrators write a definition of the workloads to execute in the cluster, and Kubernetes ensures that the running workloads match the definition. For example, an administrator defines several workloads. Each workload defines the amount of required memory. Kubernetes then creates the necessary containers in different nodes, to meet the memory requirements.

Kubernetes defines workloads in terms of *pods*. A pod is one or multiple containers that run in the same cluster node. Pods with multiple containers are useful in certain situations, when two containers must run in the same cluster node to share some resource. For example, a job is a workload that runs a task in a pod until completion.

Kubernetes Features

Kubernetes offers the following features on top of a container infrastructure:

Service discovery and load balancing
Distributing applications over multiple nodes can complicate communication between applications.

Kubernetes automatically configures networking and provides a DNS service for pods. With these features, pods can communicate with services from other pods transparently across nodes by using only hostnames instead of IP addresses. Multiple pods can back a service for performance and reliability. For example, Kubernetes can evenly split incoming requests to an NGINX web server by considering the availability of the NGINX pods.

Horizontal scaling
Kubernetes can monitor the load on a service, and create or delete pods to adapt to the load. With some configurations, Kubernetes can also provision nodes dynamically to accommodate cluster load.

Self-healing
If applications declare health check procedures, then Kubernetes can monitor, restart, and reschedule failing or unavailable applications. Self-healing protects applications both from internal failure (the application stops unexpectedly) or external failure (the node that runs the application becomes unavailable).

Automated rollout
Kubernetes can gradually roll out updates to your application's containers. If something goes wrong during the rollout, then Kubernetes can roll back to the previous version of the deployment. Kubernetes routes requests to the rolled out version of the application, and deletes pods from the previous version when the rollout completes.

Secrets and configuration management
You can manage the configuration settings and secrets of your applications without requiring changes to containers. Application secrets can be usernames, passwords, and service endpoints, or any configuration setting that must be kept private.

> **NOTE**
> Kubernetes does not encrypt secrets.

Operators
Operators are packaged Kubernetes applications that can manage Kubernetes workloads in a declarative manner. For example, an operator can define a database server resource. If the user creates a database resource, then the operator creates the necessary workloads to deploy the database, configure replication, and back up automatically.

Kubernetes Architectural Concepts

Kubernetes uses several servers (also called *nodes*) to ensure the resilience and scalability of the applications that it manages. These nodes can be physical or virtual machines. The nodes come in two types, each of which provides a different aspect of the cluster operations.

Control plane nodes provide global cluster coordination for scheduling the deployed workloads. These nodes also manage the state of the cluster configuration in response to cluster events and requests.

Compute plane nodes run the user workloads.

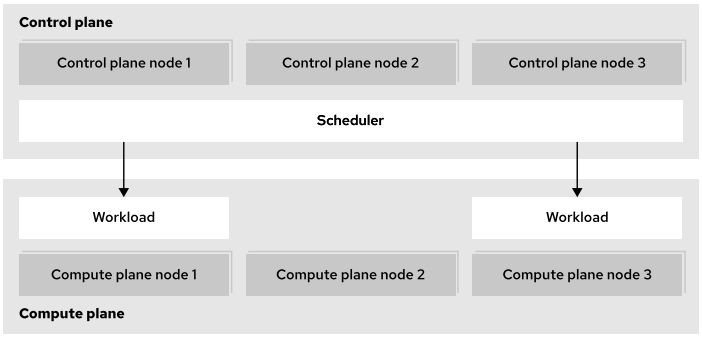

Figure 1.2: Kubernetes components

Although a server can act as both a control plane node and a compute node, the two roles are usually separated for increased stability, security, and manageability. Kubernetes clusters can range from small single-node clusters, to large clusters with up to 5,000 nodes.

The Kubernetes API and Configuration Model

Kubernetes provides a model to define the workloads to run on a cluster.

Administrators define workloads in terms of resources.

All resource types work in the same way, with a uniform API. Tools such as the `kubectl` command can manage resources of all types, even custom resource types.

Kubernetes can import and export resources as text files. By working with resource definitions in text formats, administrators can describe their workloads instead of performing the right sequence of operations to create them. This approach is called *declarative resource management*.

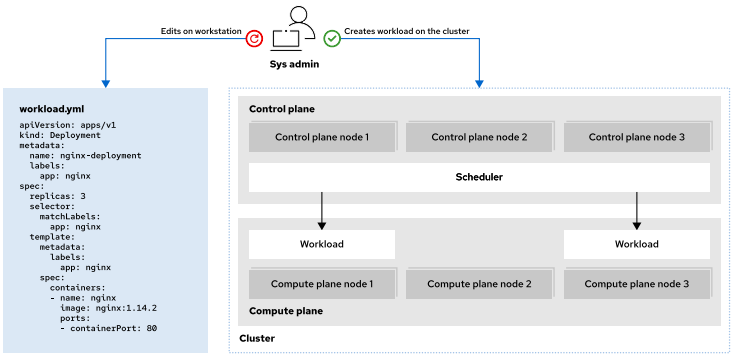

Figure 1.3: Declarative resource management

Declarative resource management can reduce the work to create workloads and improve maintainability. Additionally, when using text files to describe resources, administrators can use generic tools such as version control systems to gain further benefits such as change tracking.

To support declarative resource management, Kubernetes uses *controllers* that continuously track the state of the cluster and perform the necessary steps to keep the cluster in the intended state. This process means that changes to resources often require some time to be effective. However, Kubernetes can automatically apply complex changes. For example, if you increase the RAM requirements of an application that cannot be satisfied on the current node, then Kubernetes can move the application to a node with sufficient RAM. Kubernetes redirects traffic to the new instance only when the movement is complete.

Kubernetes also supports *namespaces*. Administrators can create namespaces, and most resources must be created inside a namespace. Besides helping organize large amounts of resources, namespaces provide the foundations for features such as resource access. Administrators can define permissions for namespaces, to allow specific users to view or modify resources.

REFERENCES

Containers

Kubernetes Concepts Overview

What Kubernetes Is Not

Considerations for Large Clusters