Chapter 1. Introduction and Overview of Containers

Introduction to Containers
Quiz: Introduction to Containers
Introduction to Kubernetes and OpenShift
Quiz: Introduction to Kubernetes and OpenShift
Summary

Abstract

Goal	Describe how containers facilitate application development.
Objectives	• Describe the basics of containers and how containers differ from Virtual Machines. • Describe container orchestration and the features of Red Hat OpenShift.
Sections	• Introduction to Containers (and Quiz) • Introduction to Kubernetes and OpenShift (and Quiz)

Introduction to Containers

Objectives

- Describe the basics of containers and how containers differ from Virtual Machines.

What is a Container

In computing, a *container* is an encapsulated process that includes the required runtime dependencies for the program to run. In a container, application-specific libraries are independent of the host operating system libraries. Libraries and functions that are not specific to the containerized application are provided by the operating system and kernel. The provided libraries and functions help to ensure that the container remains compact, and that it can quickly execute and stop as needed.

A container engine creates a union file system by merging container image layers. Because container image layers are immutable, a container engine adds a writable layer for runtime file modifications. Containers are *ephemeral* by default, which means that the container engine removes the writable layer when you remove the container.

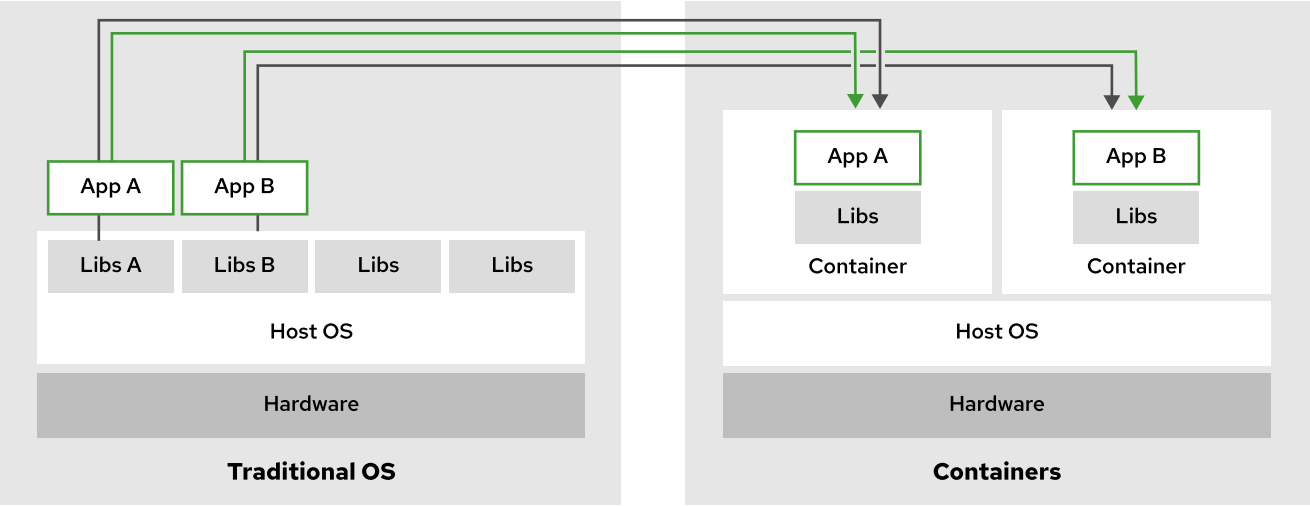

Figure 1.1: Applications in containers versus on host operating system

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 in particular provide the functionality to isolate processes within containers from each other and from the host system. As such, the environment within a container is Linux-based, regardless of the host operating system. When using containers on non-Linux operating systems, these Linux-specific features are often virtualized by the container engine implementation.

Containerization originated from technologies such as `chroot`, a method to partially or fully isolate an environment, and evolved to the *Open Container Initiative (OCI)*, which is a governance organization that defines standards for creating and running containers. Most container engines conform to the OCI specifications, so developers can confidently build their deployable target artifacts to run as OCI containers.

Images versus Instances

Containers can be split into two similar but distinct ideas: *container images* and *container instances*. A *container image* contains effectively immutable data that defines an application and its libraries. *Container instances* are the running versions of these images, incorporating runtime resources such as networking, storage, and other necessities.

You can use a single container image multiple times to create many distinct container instances. You can also run these instances across multiple hosts. The application within a container is independent of the host environment.

> **NOTE**
>
> OCI container images are defined by the `image-spec` specification, whereas OCI container instances are defined by the `runtime-spec` specification.
>
> Another way to think about *container images* versus *container instances* is that an instance relates to an image as an object relates to a class in object-oriented programming.

Comparing Containers to Virtual Machines

Containers generally serve a similar role to *virtual machines* (VMs), where an application resides in a self-contained environment with virtualized networking for communication. Although this use case initially seems to be the same, containers have a smaller footprint, and start and stop faster than a virtual machine. For both memory and disk usage, VMs are often measured in gigabytes, whereas containers are measured in megabytes.

A VM is useful when an additional full computing environment is required, such as when an application requires specific, dedicated hardware. Additionally, a VM is preferable when an application requires a non-Linux operating system or a different kernel from the host.

Virtual Machines versus Containers

Virtual machines and containers use different software for management and functionality. Hypervisors, such as KVM, Xen, VMware, and Hyper-V, are applications that provide the virtualization functionality for VMs. The container equivalent of a hypervisor is a container engine, such as Podman.

	Virtual machines	**Containers**
Machine-level functionality	Hypervisor	Container engine
Management	VM management interface	Container engine or orchestration software
Virtualization level	Fully virtualized environment	Only relevant parts
Size	Measured in gigabytes	Measured in megabytes
Portability	Generally only same hypervisor	Any OCI-compliant engine

You can manage hypervisors with additional management software, which can be included with the hypervisor, or be external, such as Virtual Machine Manager with KVM. In contrast, you can manage containers directly through the container engine itself. Additionally, you can use container orchestration tools, such as Red Hat OpenShift Container Platform (RHOCP) and Kubernetes, to run and manage containers at scale. RHOCP manages both containers and virtual machines from a common interface.

With VMs, interoperability is uncommon. A VM that runs on one hypervisor is usually not guaranteed to run on a different hypervisor. In contrast, containers that follow the OCI specification do not require a particular container engine to function. Many container engines can function as drop-in replacements for each other.

Deployment at Scale

Both containers and VMs can work well at various scales. Because a container requires considerably fewer resources than a VM, containers have performance and resource benefits at a larger scale. A common method in large-scale environments is to use containers that run inside VMs. This configuration takes advantage of the strong points in each technology.

Development for Containers

Containerization provides many advantages for the development process, such as easier testing and deployments, by providing tools for stability, security, and flexibility.

Testing and Workflows

One of the greatest advantages of containers for developers is the ability to scale. A developer can write software and test locally, and then deploy the finished application to a cloud server or a dedicated cluster with few or no changes. This workflow is especially useful when creating microservices, which are small and ephemeral containers that are designed to spin up and down as needed. Additionally, developers that use containers can take advantage of *Continuous Integration/Continuous Development (CI/CD)* pipelines to deploy containers to various environments. In particular, RHOCP offers various integration features with CI/CD pipelines and workflows in mind.

Stability

As mentioned earlier, container images are a stable target for developers. Software applications require specific versions of libraries to be available for deployment, which can result in dependency issues or specific OS requirements. Because libraries are included in the container image, a developer can be confident of no dependency issues in a deployment. Having the libraries integrated within the container removes variability between testing and production environments. For example, a container with a specific version of Python ensures that the same version of Python is used in every testing or deployment environment.

Multi-container Applications

A multi-container application is distributed across several containers. You can run the containers from the same image for *high availability* (HA) replicas, or from several different images. For example, a developer can create an application that includes a database container that runs separately from the application's web API container. An application can rely on the container management software to provide HA replicas with multi-container options, such as Podman Pods or the `compose-spec` specification.

REFERENCES

Red Hat topic for containers

Red Hat topic for virtualization

Podman official documentation

About OCI specifications

Kubernetes official documentation

For more information about Red Hat OpenShift Container Platform, refer to the documentation at https://access.redhat.com/documentation/en-us/openshift_container_platform/4.18