What Is Docker?

November 9, 2021

In This Article

Docker Defined

A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly.

Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle.

Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products.

Define Docker Terminology

Docker:
A software container platform designed for developing, shipping, and running apps leveraging container technology. Docker comes in two versions: enterprise edition and community edition

Container:
Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.

Docker engine:
The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.

Docker images:
Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.

Docker Registry:
Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.

Cloud Container Registry

Who Uses Docker?

Docker is an open application development framework that's designed to benefit DevOps and developers. Using Docker, developers can easily build, pack, ship, and run applications as lightweight, portable, self-sufficient containers, which can run virtually anywhere. Containers allow developers to package an application with all of its dependencies and deploy it as a single unit. By providing prebuilt and self-sustaining application containers, developers can focus on the application code and use without worrying about the underlying operating system or deployment system.

Additionally, developers can leverage thousands of open source container applications that are already designed to run within a Docker container. For DevOps teams, Docker lends itself to continuous integration and development toolchains and reduces the constraints and complexity needed within their system architecture to deploy and manage the applications. With the introduction of container orchestration cloud services, any developer can develop containerized applications locally in their development environment, and then move and run those containerized applications in production on cloud services, such as managed Kubernetes services.

Docker and Developers

Containers can be packaged by any kind of developer. Individuals in the software industry often separate developers by specialization—front end, back end, or any concentration between. While you mostly may see back-end developers packaging containers, anyone familiar with CaaS basic concepts can succeed in this particular area of the software development life cycle. Before you're ready to package your application's dependencies, check out developer.oracle.com and familiarize yourself with tools you can use to build your application or program.

Docker Versus Kubernetes

Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.

Docker Basics

The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.

A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.

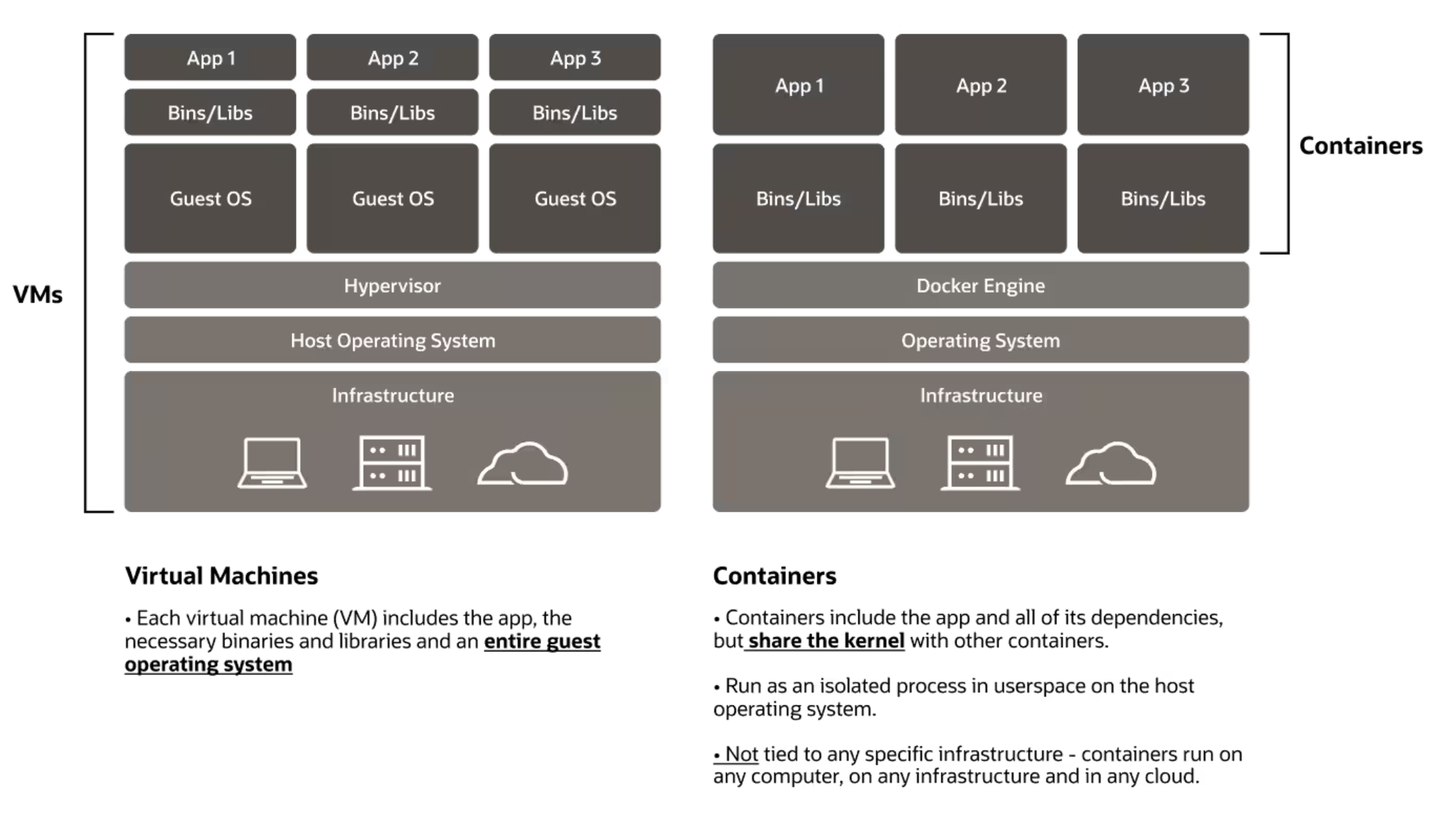


Figure 1

Isolation vs. Virtualization

Every Docker container has its own file system, its own network stack (and therefore its own IP address), its own process space, and defined resource limitations for CPU and memory. Since a Docker container does not have to boot an operating system, it starts up instantly. Docker is about isolation, i.e., separating the resources of a host operating system, as opposed to virtualization, i.e., providing a guest operating system on top of the host operating system.

Incremental Files System

The file system of a Docker image is layered, with copy-on-write semantics. This enables inheritance and reuse, saves resources on disk, and enables incremental image download.

As illustrated in Figure 2, a Docker image with a WebLogic deployment could be based on an image with an Oracle WebLogic Server domain, which could be based on a WebLogic image, which is based on a Java Development Kit (JDK) image, which in turn is based on an Oracle Linux base image.

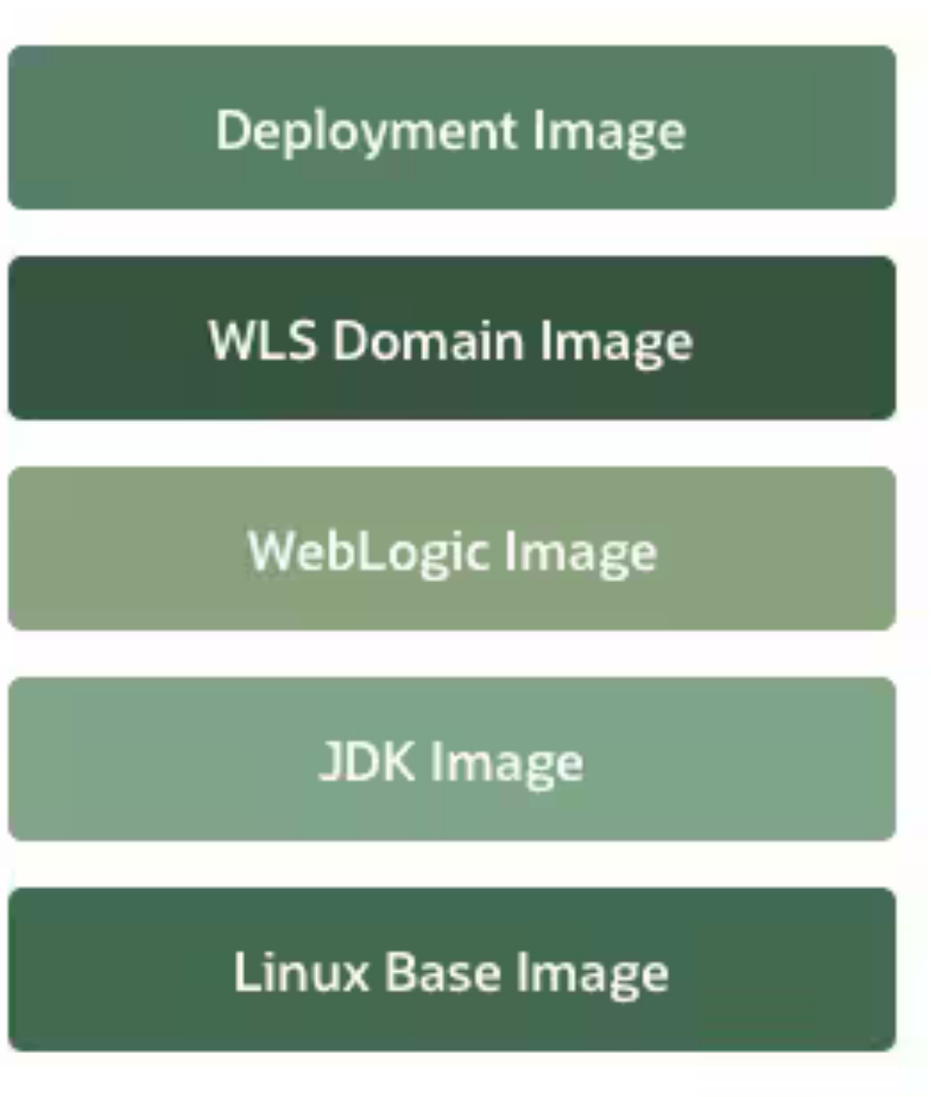


Figure 2

Docker Registry

While Docker images are easy to build and developers love the simplicity and portability of Docker images, they quickly discovered that managing thousands of Docker images is very challenging. Docker Registry address this challenge. Docker Registry is a standard way to store and distribute Docker images. The Registry is an open source-based repository under the permissive Apache license.

Docker Registry also helps improve access control and security of the Docker images stored in its repository. It manages the distribution of images and also can integrate with application development workflows. Developers can setup their own Docker Registry, or use a hosted Docker Registry service such as Docker Hub, Oracle Container Registry, Azure Container Registry, etc.

Docker Hub is a hosted Docker registry managed by Docker. Docker Hub has over 100,000 container images from software vendors, open source projects, and the community. Docker Hub contains software and applications from official repositories such as NGINX, Logstash, Apache HTTP, Grafana, MySQL, Ubuntu, and Oracle Linux.

When starting a container, Docker will by default automatically pull the corresponding image from the public Docker Hub if it is not available locally. Moreover, you can also create your own images and push them to Docker Hub into either a public or private repository.

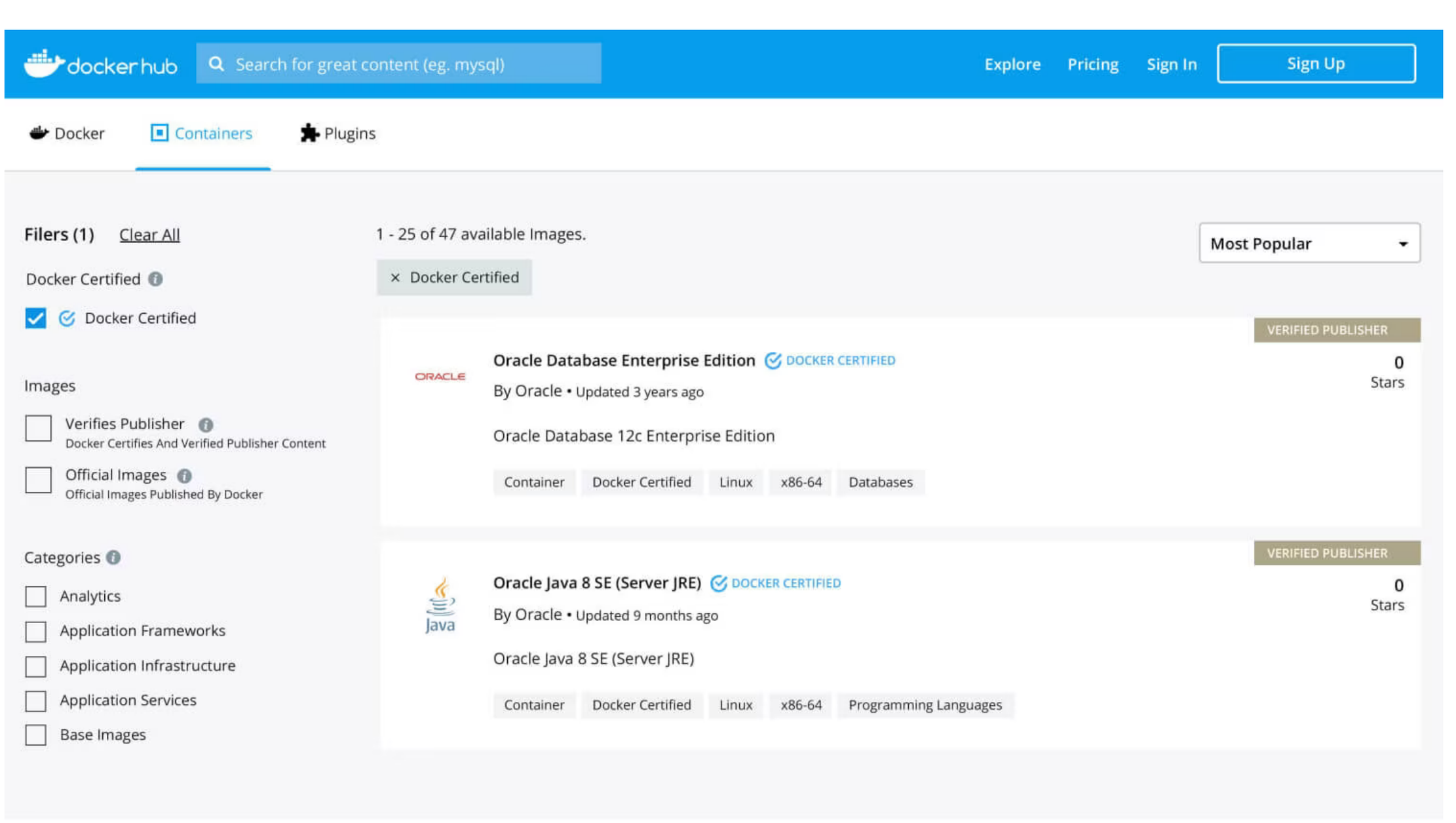


Figure 3

Docker as a Microservices Runtime

The idea of cutting monolithic applications into smaller chunks of microservices attracts a lot of attention these days among software developers.

Microservices are independently deployed as a process, use light-weight protocols to communicate with each other, and every service owns its data. Since microservices follow a decentralized governance approach, they require a rather high amount of infrastructure automation, automated testing, fully automated CD pipelines, and skilled, agile DevOps teams.

There is still a lot of discussion about this architectural style, yet it would be naive to assume that an application decomposed into microservices can be simply operated as a set of processes. To name only a few requirements, a microservice needs to be host-independent, and isolated on an operating system level. It must run within its resource limits, must be scaled up and down, restarted if failed, and discovered and connected to other microservices via a software defined network layer.

Therefore, running a microservice in a Docker container puts you at an excellent jumping-off point to achieve most of these goals.

Docker—Two Key Dimensions

Docker changes the way we build, ship, and run software in two different dimensions:

- It enhances the process to get applications reliably from development to production.
- It provides a standards image format to get from on-premises to cloud.

Both dimensions are explained in more detail in the following paragraphs.

Docker Image—Development to Production

Creating a Docker image with all of its dependencies solves the "but it worked for me on my development machine" problem. The key idea is that a Docker image is created automatically by a build pipeline from a source-code repository like Git and initially tested in a development environment. This immutable image will then be stored in a Docker registry.

As shown in the Figure 4, the same image will be used for further load tests, integration tests, acceptance tests, and more. In every environment, the same image will be used. Small but necessary environmentally specific differences, such as a JDBC URL for a production database, can be fed into the container as environment variables or files.

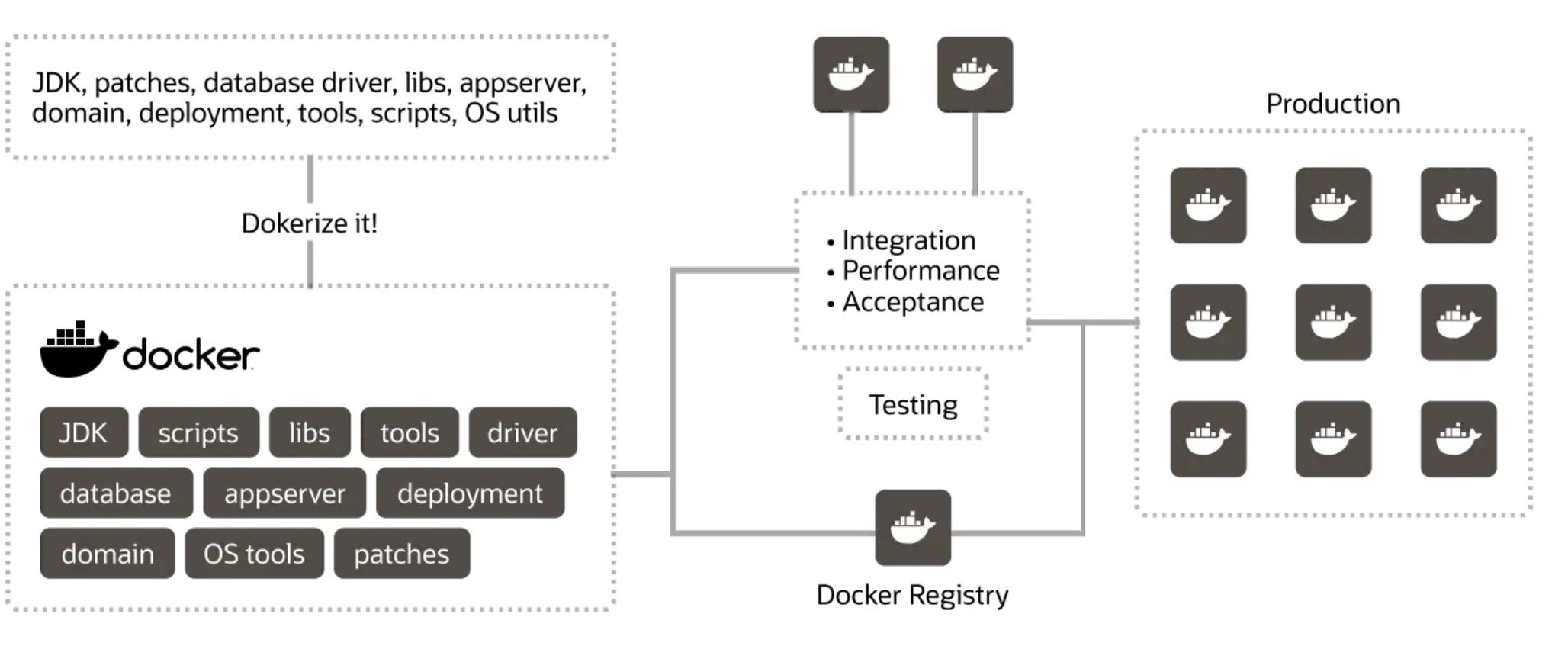


Figure 4

Statistics show that 65% of all current Docker use cases are in development, and 48% use Docker for continuous integration.

Docker Cloud

Docker changed the adoption of public clouds: On one hand, with a Docker image, for the first time in history, a common package format exists that can be run on premises as well as on every major cloud provider. Docker containers run on a laptop the same way they run on Oracle Cloud.

On the other hand—since Docker containers run on every major public cloud—they are a major contribution to overcoming a long curated prejudice against public clouds: vendor lock-in. Every major cloud provider now offers Docker as a PaaS.

Docker Versions—Maturity of Underlying Technology

The pace of Docker releases is much faster than the release cycle of the traditional enterprise software. Sometimes the sheer pace of Docker releases, together with the newness of the Docker project, raises concerns about the security and stability of Docker.

Although Docker and its command line, the Docker daemon, its API, and tools such as Docker Swarm, Docker Machine, and Docker Compose only evolved in the last three years, the underlying kernel features have been available in every Linux kernel for nearly a decade.

A prominent example of an early adopter of container technology is Google. Google has been using Linux containers even before Docker was around. Furthermore, Google runs everything in a container. It is estimated that Google launches several billion containers per week.

Cgroups and Namespaces History

The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers[1]. Cgroups limit and account for the resource usage of a set of operating system processes.

The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.[2]

Container Cloud Services

The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container.

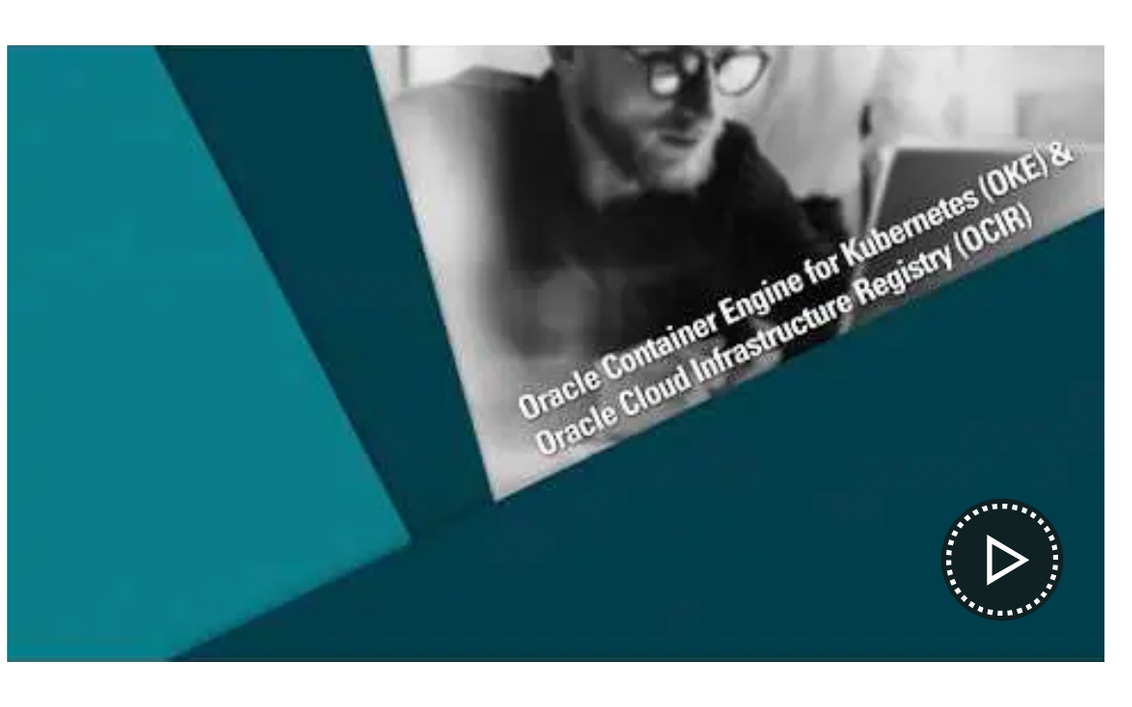
To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming.

To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services.

Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications.

Docker Images From Oracle

Containers can be packaged by any kind of developer. Individuals in the software industry often separate developers by specialization—front end, back end, or any concentration between. While you mostly may see back-end developers packaging containers, anyone familiar with CaaS basic concepts can succeed in this particular area of the software development life cycle. Before you're ready to package your application's dependencies, check out developer.oracle.com and familiarize yourself with tools you can use to build your application or program.



Below are some sources for obtaining or building Docker images for Oracle products. The Oracle GitHub repository for Docker images contains Dockerfiles and samples to build Docker images for Oracle commercial products and Oracle sponsored open source projects.

- Oracle GitHub repository for Docker images
- Oracle Container Registry

Docker Hands-On Lab—Containerized Development With Docker

- Containerized Development with Docker on Oracle Cloud

References

1. Cgroups (Wikipedia)
2. Linux Namespaces (Wikipedia)