✓ 100 XP

What is Docker?

5 minutes

Before we start our quick tour of Docker containers, let's have a look at how our team develops and deploy applications. We'll also briefly discuss some of the challenges faced by our teams.

The process of developing and managing applications in your company typically includes one or more teams. There's a development team that creates the software, and an operations team responsible for the deployment of these applications. The operations team is also responsible for the management of the application hosting infrastructure.

For example, assume we're developing an order tracking portal that our company's various outlets will use. Several environments host our applications during the app's development and publish process. First, the development team develops and tests the software in a development environment. From here, the software is then deployed to a quality assurance (QA) environment, followed by pre-production, and a final production environment.

There are several challenges that we'll need to consider in the above scenario:

- **Management of hosting environments**

  The different environments all require both software and hardware management. We have to ensure that both the installed software and configured hardware in each is the same. We also need to configure aspects such as network access, data storage, and security per environment in a consistent and easily reproducible manner.

- **Continuity in software delivery**

  The deployment of applications to our environments must happen consistently. Each deployment package must include all system packages, binaries, libraries, configuration files, and other items that will ensure a fully functional application. We also need to make sure that all of these dependencies match software versions and architecture.

- **Efficient use of hardware**

  Each deployed application must execute in such a way that it's isolated from other applications running on the same hardware. We aim to run more than one application per server to make the best use of resources without compromising each other.

- **Application portability**

> There are several reasons why application portability is essential. A hosting environment might fail, or we need to scale out our application. In both instances, the potential result is a redeployment of our software to a new environment. We want to move software from one host to another even if the underlying infrastructure is different. Such a move needs to happen as fast as possible to reduce downtime for our customers.

Before we look at the Docker features that help solve these challenges, we'll discuss a few concepts and look at a brief overview of the Docker architecture.

What is a container?

A container is a loosely isolated environment that allows us to build and run software packages. These software packages include the code and all dependencies to run applications quickly and reliably on any computing environment. We call these packages *container images*.

The container image becomes the unit we use to distribute our applications.

What is software containerization?

Software containerization is an OS virtualization method that is used to deploy and run containers without using a virtual machine (VM). Containers can run on physical hardware, in the cloud, VMs, and across multiple OSs.

What is Docker?

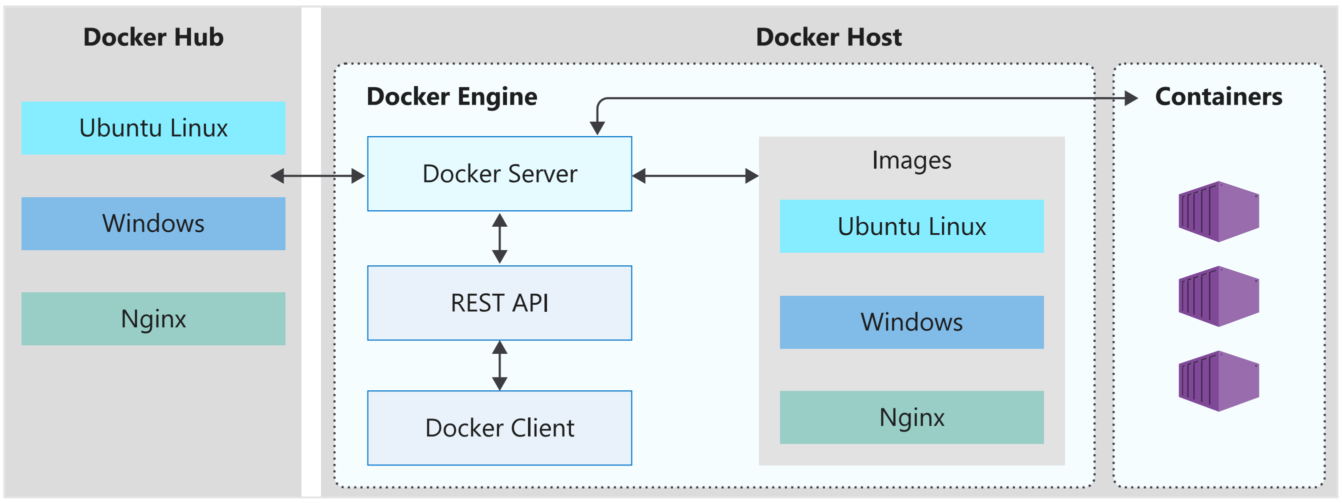
Docker is a containerization platform used to develop, ship, and run containers. Docker doesn't use a hypervisor, and you can run Docker on your desktop or laptop if you're developing and testing applications. The desktop version of Docker supports Linux, Windows, and macOS. For production systems, Docker is available for server environments, including many variants of Linux and Microsoft Windows Server 2016 and above. Many clouds, including Azure, supports Docker.

Docker architecture

The Docker platform consists of several components that we use to build, run, and manage our containerized applications.

Docker Engine

The Docker Engine consists of several components configured as a client-server implementation where the client and server run simultaneously on the same host. The client communicates with the server using a REST API, which allows the client to also communicate with a remote server instance.



The Docker client

The Docker client is a command-line application named `docker` that provides us with a command line interface (CLI) to interact with a Docker server. The `docker` command uses the Docker REST API to send instructions to either a local or remote server and functions as the primary interface we use to manage our containers.

The Docker server

The Docker server is a daemon named `dockerd`. The `dockerd` daemon responds to requests from the client via the Docker REST API and can interact with other daemons. The Docker server is also responsible for tracking the lifecycle of our containers.

Docker objects

There are several objects that you'll create and configure to support your container deployments. These include networks, storage volumes, plugins, and other service objects. We won't cover all of these objects here, but it's good to keep in mind that these objects are items that we can create and deploy as needed.

Docker Hub

Docker Hub is a Software-as-a-Service (SaaS) Docker container registry. Docker registries are repositories that we use to store and distribute the container images we create. Docker Hub is the default public registry Docker uses for image management.

Keep in mind that you can create and use a private Docker registry or use one of the many cloud provider options available. For example, you can use Azure Container Registry to store Docker containers to use in several Azure container enabled services.

Next unit: How Docker images work

Continue >