5th December 2023

Docker (Day 1)

- **What is a Container?**

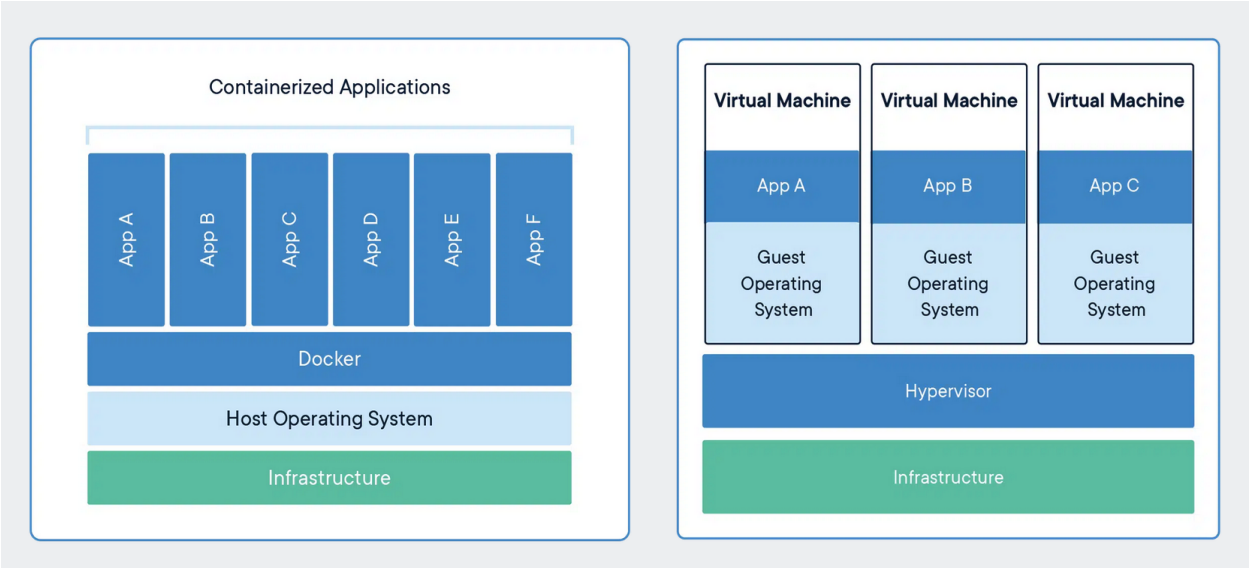
A **Container** is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers are a form of virtualisation but are more lightweight than traditional virtual machines.

A container is a bundle of Applications, Application libraries required to run your application and the minimum system dependencies.

Key features include :

1) Isolation
2) Portability
3) Efficiency
4) Consistency
5) Scalability

Docker is one of the most widely used containerization platforms, and it has played a significant role in popularizing container technology. On the other hand, **Kubernetes** is a popular container orchestration platform that automates containerised applications' deployment, scaling, and management. Image taken from - "https://github.com/iam-veeramalla/Docker-Zero-to-Hero".



- **Containers vs Virtual Machines**

Containers and **virtual machines (VMs)** are both technologies used for virtualization, but they serve different purposes and have distinct characteristics. Here are the key differences between containers and virtual machines:

1) Isolation:

Containers: Containers share the host operating system's kernel, which means they are lightweight and start up quickly. They provide process and file system isolation, but they all run on the same OS kernel.

Virtual Machines: VMs, on the other hand, emulate an entire operating system, including the kernel. Each VM runs its operating system, providing stronger isolation but requiring more resources.

2) Resource Efficiency:

Containers: Containers are more resource-efficient than VMs because they share the host OS kernel and don't require a full operating system for each instance.

Virtual Machines: VMs have more overhead since each VM includes a full operating system and has its kernel.

3) Performance:

Containers: Containers generally offer better performance due to their lightweight nature and reduced overhead.

Virtual Machines: VMs may have slightly more performance overhead because of the need to emulate a full operating system.

4) Portability:

Containers: Containers are highly portable because they encapsulate the application and its dependencies. They can run consistently across different environments.

Virtual Machines: VMs are less portable because they include the entire operating system. Moving VMs between different environments may require more effort.

5) Startup Time:

Containers: Containers start up quickly since they don't need to boot an entire operating system.

Virtual Machines: VMs typically have a longer startup time because they need to boot a full OS.

6) Scaling:

Containers: Containers are well-suited for microservices architectures and can be easily scaled up or down based on demand.

Virtual Machines: While VMs can also be scaled, they are generally heavier and may have longer startup times.

7) Use Cases:

Containers: Ideal for microservices architectures, continuous integration/continuous deployment (CI/CD), and cloud-native applications.

Virtual Machines: Often used for running legacy applications, environments with different operating systems, and situations where stronger isolation is required.

In summary, containers are generally favoured for their lightweight nature, fast startup times, and efficiency, while virtual machines provide stronger isolation and may be preferred in certain scenarios where complete OS separation is necessary.

- **What is a Docker?**

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that can run applications and their dependencies isolated from the host system and other containers. Docker provides a set of tools and a platform to automate the deployment and scaling of containerized applications.

Key components of Docker include:

Docker Engine:

The core component responsible for building, running, and managing containers. It includes a server, a REST API, and a command-line interface (CLI).

Docker Image:

A lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Images are used to create containers.

Container:

An instance of a Docker image that runs in isolation on the host system. Containers share the host OS kernel but have their own filesystem, processes, and network space.

Dockerfile:

A text file that contains instructions for building a Docker image. It specifies the base image, application code, dependencies, and other configuration settings.

Docker Hub:

A cloud-based registry service where Docker images can be stored, shared, and distributed. It allows users to publish and access pre-built images.

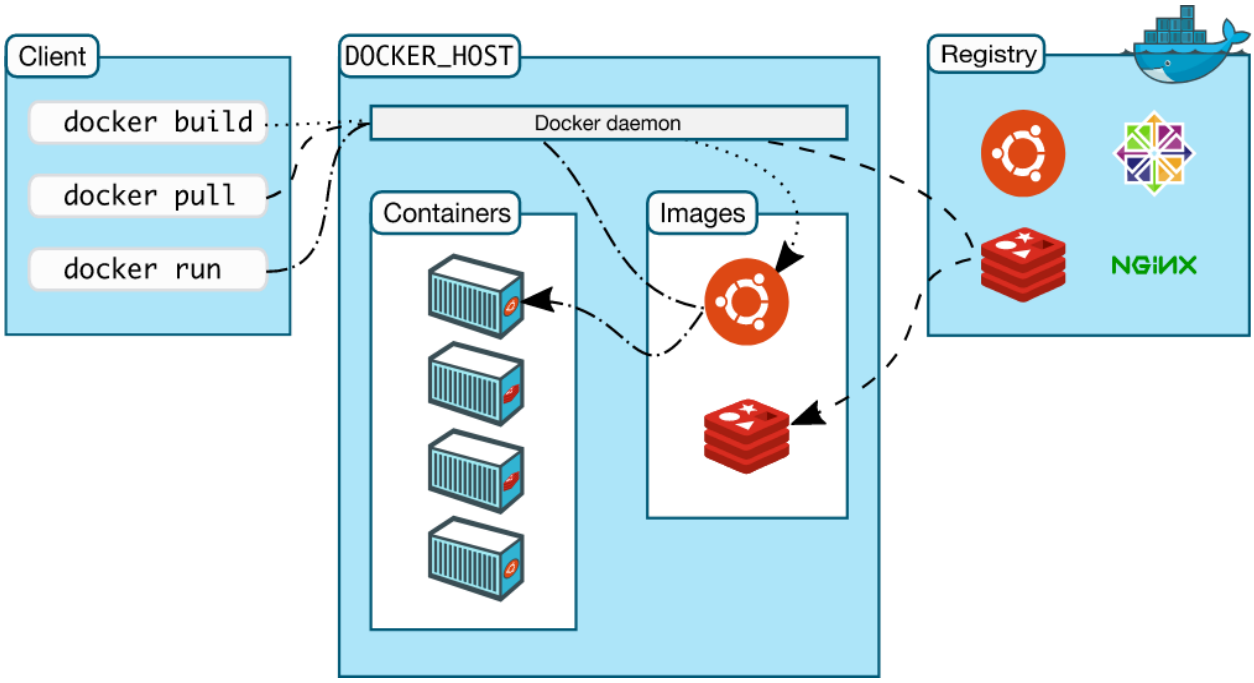
How Docker works:

Build:
Developers create a Dockerfile that describes the application and its dependencies.
Docker builds an image based on the Dockerfile.

Ship:
The Docker image is pushed to a registry (like Docker Hub) or a private registry.
Other developers or systems can pull the image from the registry.

Run:
The Docker image is used to create and run containers on any system that has Docker installed.
Containers are isolated and share the same underlying infrastructure.

Docker Architecture - from "https://github.com/iam-veeramalla/Docker-Zero-to-Hero"



- **Understanding the terminology (Inspired from Docker Docs)**

Docker daemon
The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

Docker client
The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop
Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries
A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry. Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Dockerfile
Dockerfile is a file where you provide the steps to build your Docker Image.

Images
An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

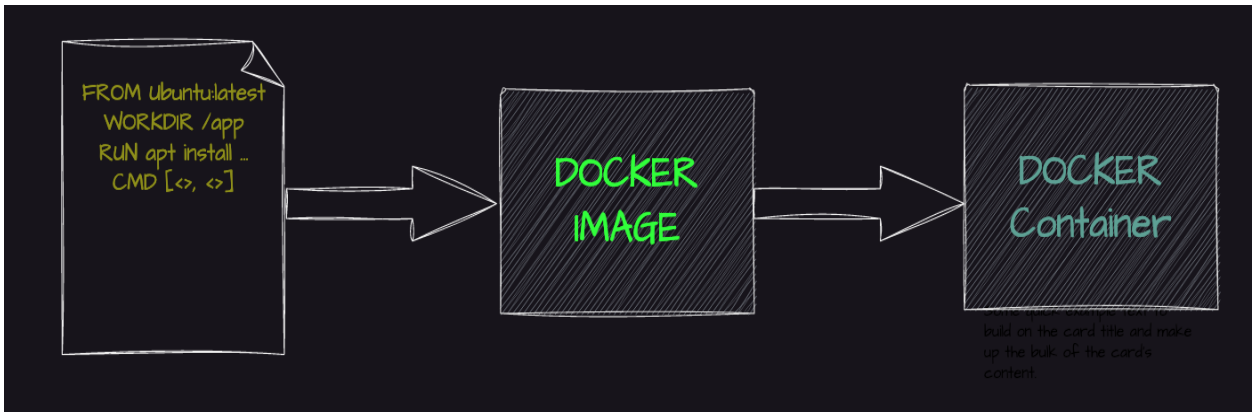
You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

- **Docker LifeCycle**

We can use the above Image as reference to understand the lifecycle of Docker.

There are three important things,

1) docker build -> builds docker images from Dockerfile
2) docker run -> runs container from docker images
3) docker push -> push the container image to public/private regestries to share the docker images.
Image from - “https://github.com/iam-veeramalla/Docker-Zero-to-Hero”.



You can follow this link for the Docker installation part - “https://github.com/iam-veeramalla/Docker-Zero-to-Hero”.

You can follow the below playlist for Docker - “https://youtube.com/playlist?list=PLdpzxOOAlwvLjb0vTD9BXLOwwLD_GWCmC&si=O11HT8baHQdnF8QX”.