SANDEEP ANURAGI

Red Hat Trainer Instructor

ABSTRACT

Introduction of Docker

Sandeep Anuragi

Docker Series.

What is Docker?

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers, containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By Used so, the developer can rest assured that the application will run on any other Linux machine regardless of any customized setting that machine might have that could differ from the machine used for writing and testing the code.

On the other way, **Docker** is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer. This give a significant performance boost and reduces the size of the application.

Docker and Security

Docker brings security to applications running in a shared environment, but containers by themselves are not an alternative to taking proper security measures.

Understanding containers

Containers can be thought of as necessitating three categories of software:

Builder: technology used to build a container.

Engine: technology used to run a container.

Orchestration: technology used to manage many containers.

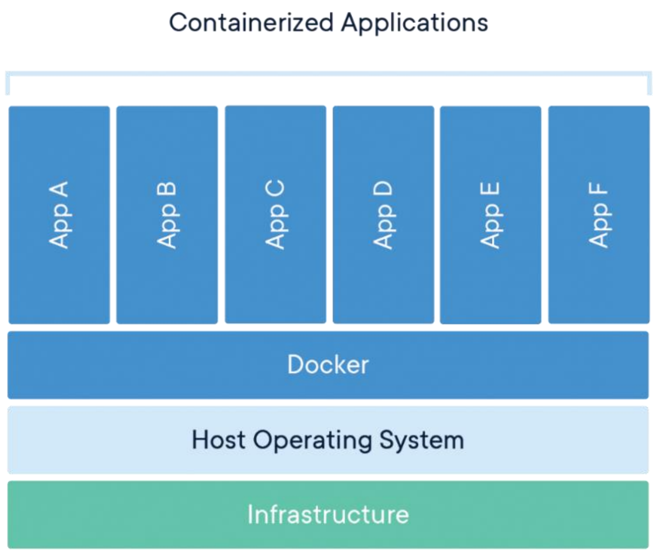
Container Orchestration

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprise that need to deploy and manage hundreds or thousands of Linux Containers and hosts can benefit from container orchestration.

Container orchestration can be used in any environment where you use containers. It can help you to deploy the same applications across different environments without needing to redesign it, And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles.

Managing the lifecycle of containers with orchestration also supports DevOps teams who integrate it into CI/CD workflows. Along with application programming interfaces (APIs) and DevOps teams, containerized microservices are the foundation for cloud-native applications.



What is container orchestration used for?

Use container orchestration to automate and manage tasks such as:

Provisioning and Deployment

Configuration and scheduling

Resource allocation

Container availability

Scaling or removing containers based on balancing workloads across your infrastructure.

Load balancing and traffic routing

Monitoring container health

Configuration applications based on the container in which they will run

Keeping interaction between containers secure

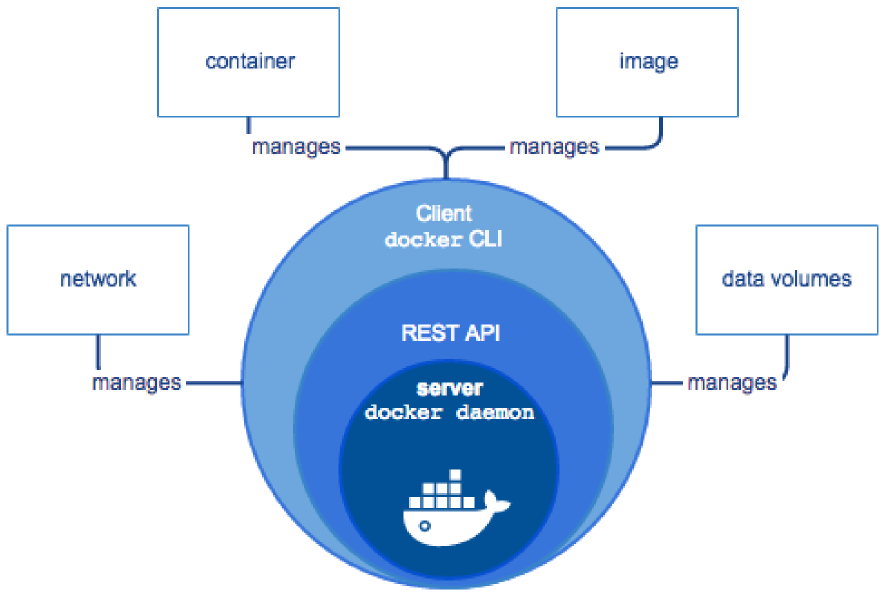
Docker Engine

Docker Engine is a client-server application with these major components:

A server which is a type of long-running program called a daemon process (the **dockerd** command).

A **REST API** which specifies interfaces that programs can use to talk to daemon and instruct it what to do.

A **command line interface (CLI)** client (the **docker** command).



The CLI uses the docker REST API to control or interact with the Docker Daemon through scripting or direct CLI commands. Many other docker applications use the underlying API and CLI.

The demon creates and manages docker objects, such as images, containers, networks, and volumes.

Who is Docker for?

Docker is a tool that is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains. For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on. It also allows them to get a head start by using one of thousands of programs already designed to run in a Docker container as a part of their application. For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead.

What can I use Docker for?

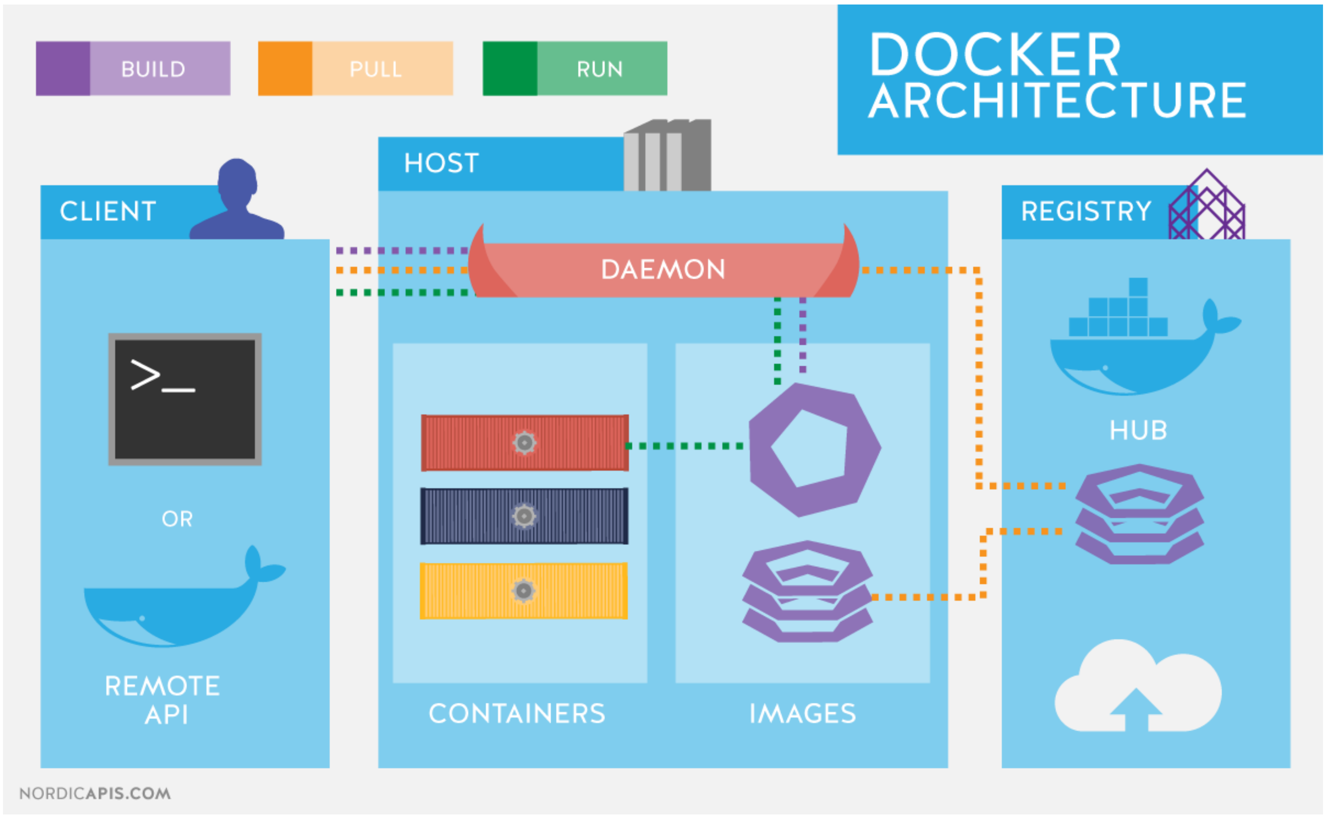
Fast, consistent delivery of your applications.

Responsive deployment and scaling

Running more workloads on the same hardware.

Docker Architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.



The 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.

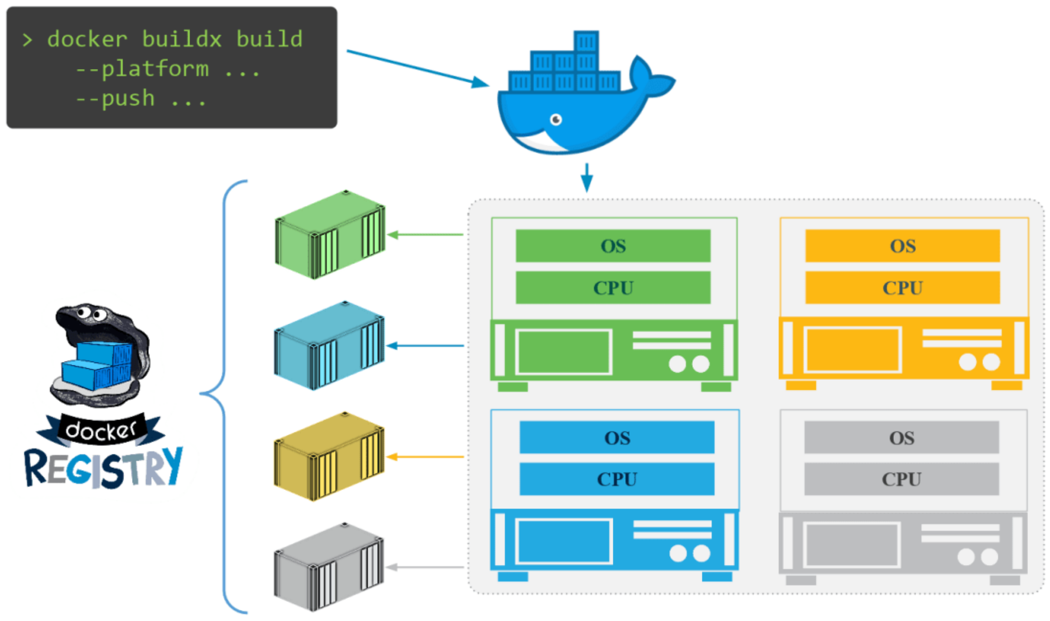
The Docker Client

The Docker Client 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 Registries.

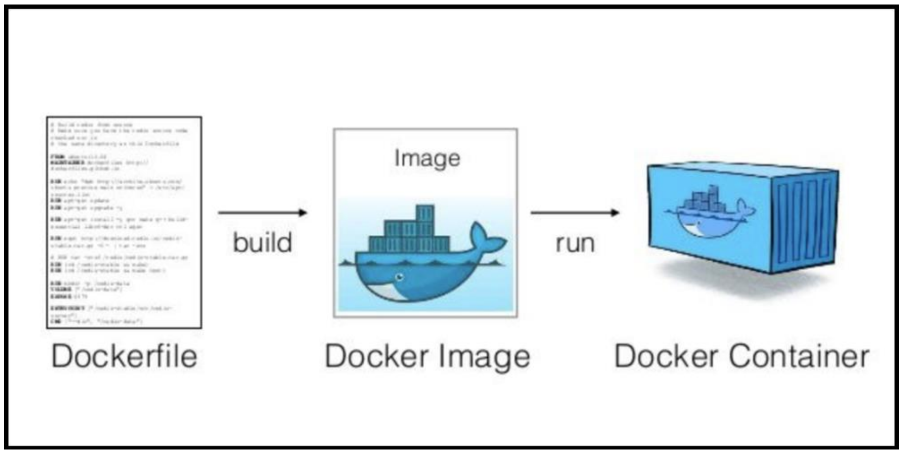
A Docker registry stores docker images. Docker hub is a public registry that anyone can use, and docker is configured to look for images on docker hub by default. You can even run your own private registry. If you use docker datacenter (DDC), it includes docker trusted Registry (DTR).

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 buildx build
    --platform ...
    --push ...
```

Docker images

An image is a read-only template with instructions for creating 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 applications, as well as the configuration details needed to make your application run.



Containers

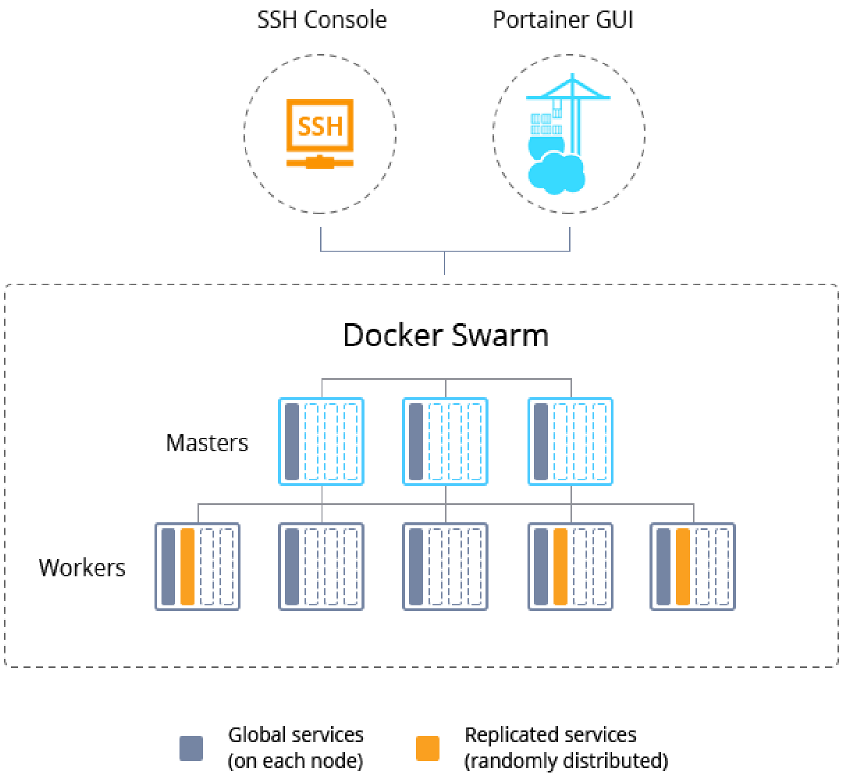
A container is a runnable instance of an image. You can create, start, stop, move or delete a container using the docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Services

Services allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and all the daemons communicate using the Docker API. A service allows you to define the desired state, such as the number of replicas of the service that must be available at any given time. By default, the service is load-balanced across all worker nodes. To the consumer, the Docker service appears to be a single application. Docker Engine supports swarm mode in Docker 1.12 and higher.


SSH Console
Portainer GUI
SSH
Docker Swarm
Masters
Workers
Global services (on each node)
Replicated services (randomly distributed)

The underlying technology

Docker is written in [Go](https://golang.org/) and takes advantage of several features of the Linux kernel to deliver its functionality.

Namespaces

Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

The pid namespace: Process isolation (PID: Process ID).

The net namespace: Managing network interfaces (NET: Networking).

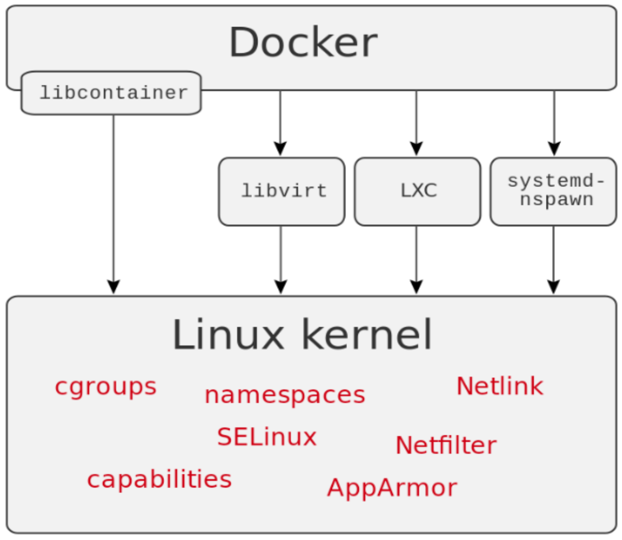
The ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).

The mnt namespace: Managing filesystem mount points (MNT: Mount).

The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups

Docker Engine on Linux also relies on another technology called control groups (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.



Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Container format

Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones

Components of Docker

Docker has the following components

Docker for Mac − It allows one to run Docker containers on the Mac OS.

Docker for Linux − It allows one to run Docker containers on the Linux OS.

Docker for Windows − It allows one to run Docker containers on the Windows OS.

Docker Engine − It is used for building Docker images and creating Docker containers.

Docker Hub − This is the registry which is used to host various Docker images.

Docker Compose − This is used to define applications using multiple Docker containers

Docker Hub

Docker Hub is a service provided by Docker for finding and sharing container images with your team. It provides the following major features:

Repositories: Push and pull container images.

Teams & Organizations: Manage access to private repositories of container images.

Official Images: Pull and use high-quality container images provided by Docker.

Publisher Images: Pull and use high- quality container images provided by external vendors. Certified images also include support and guarantee compatibility with Docker Enterprise.

Builds: Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.

Webhooks: Trigger actions after a successful push to a repository to integrate Docker Hub with other services.