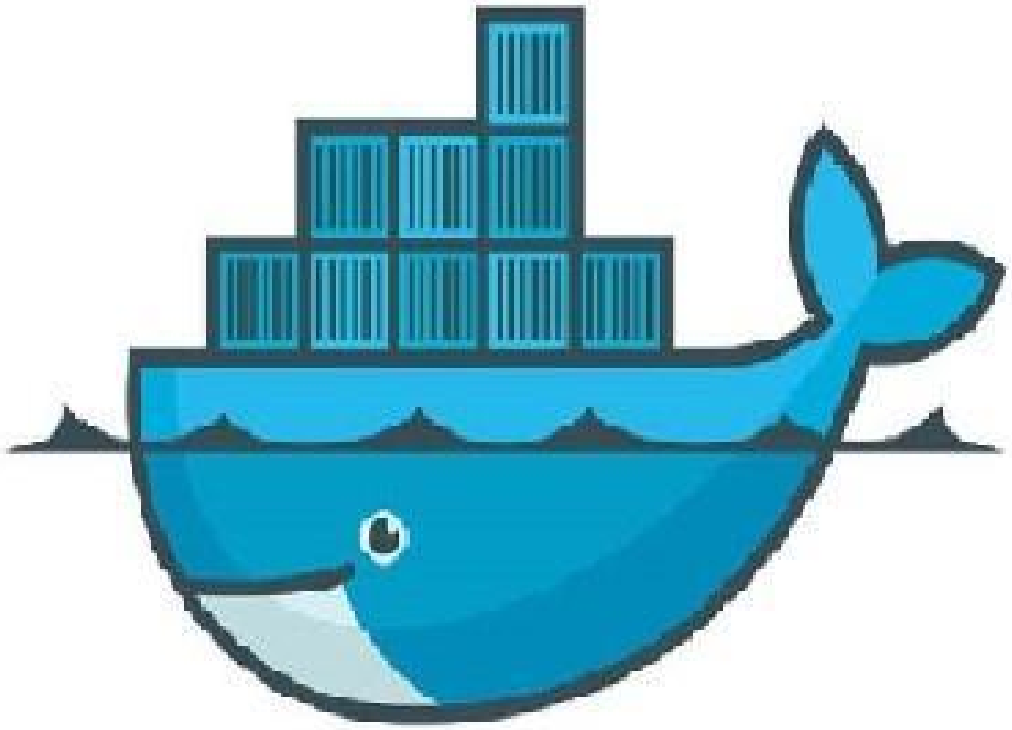
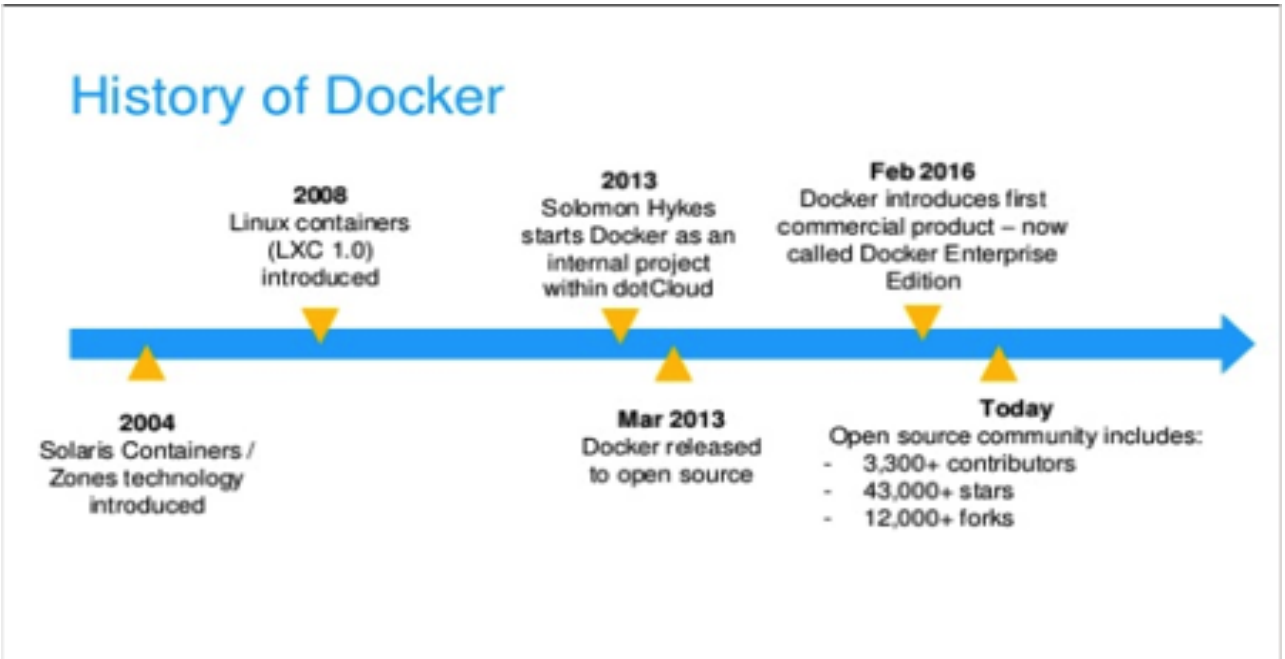


Why We Need Docker?

- Developing apps today requires so much more than writing code.
- Multiple languages, frameworks, architectures, and discontinuous interfaces between tools for each lifecycle stage creates enormous complexity.
- Docker simplifies and accelerates your workflow, while giving developers the freedom to innovate with their choice of tools, application stacks, and deployment environments for each project.

Introduction

- Docker is an open platform for developing, shipping, and running 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 ship it all out as one package.
- It is a bit like virtual machine.
- Can run on popular 64-bit Linux distribution.
- Supported by several cloud platform including Amazon EC2, Google Computer Engine and Rackspace.



What Is Docker?

- When we say Docker we are talking about the entire ecosystem or Docker products.
- **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.

- It include Docker Client, Docker Server, Docker Hub, Docker Compose.

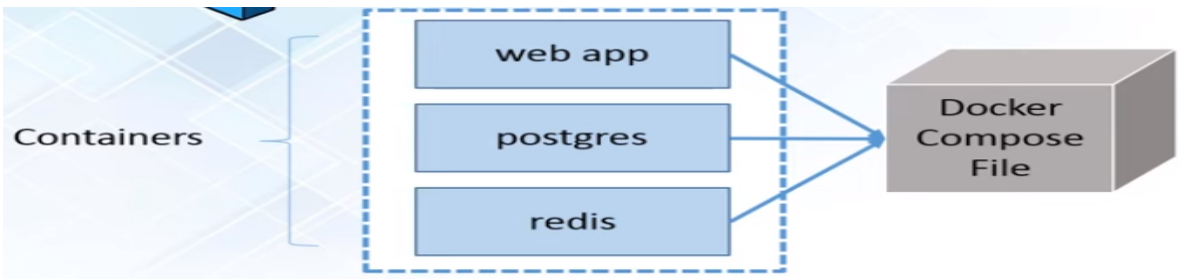
Docker Tools.

The Docker are consists of two basic tools. There are a follows:

- Docker Compose.
- Docker Swarm.

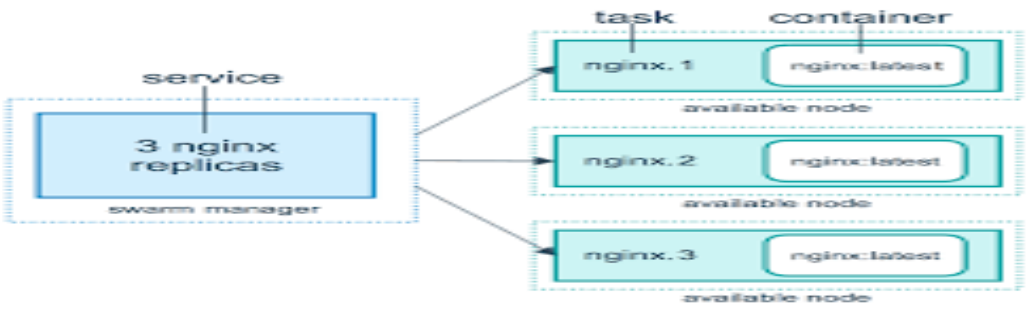
1. Docker Compose

- Docker Compose is a tool for defining and running multi-container Docker applications.
- It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command.



2. Docker Swarm

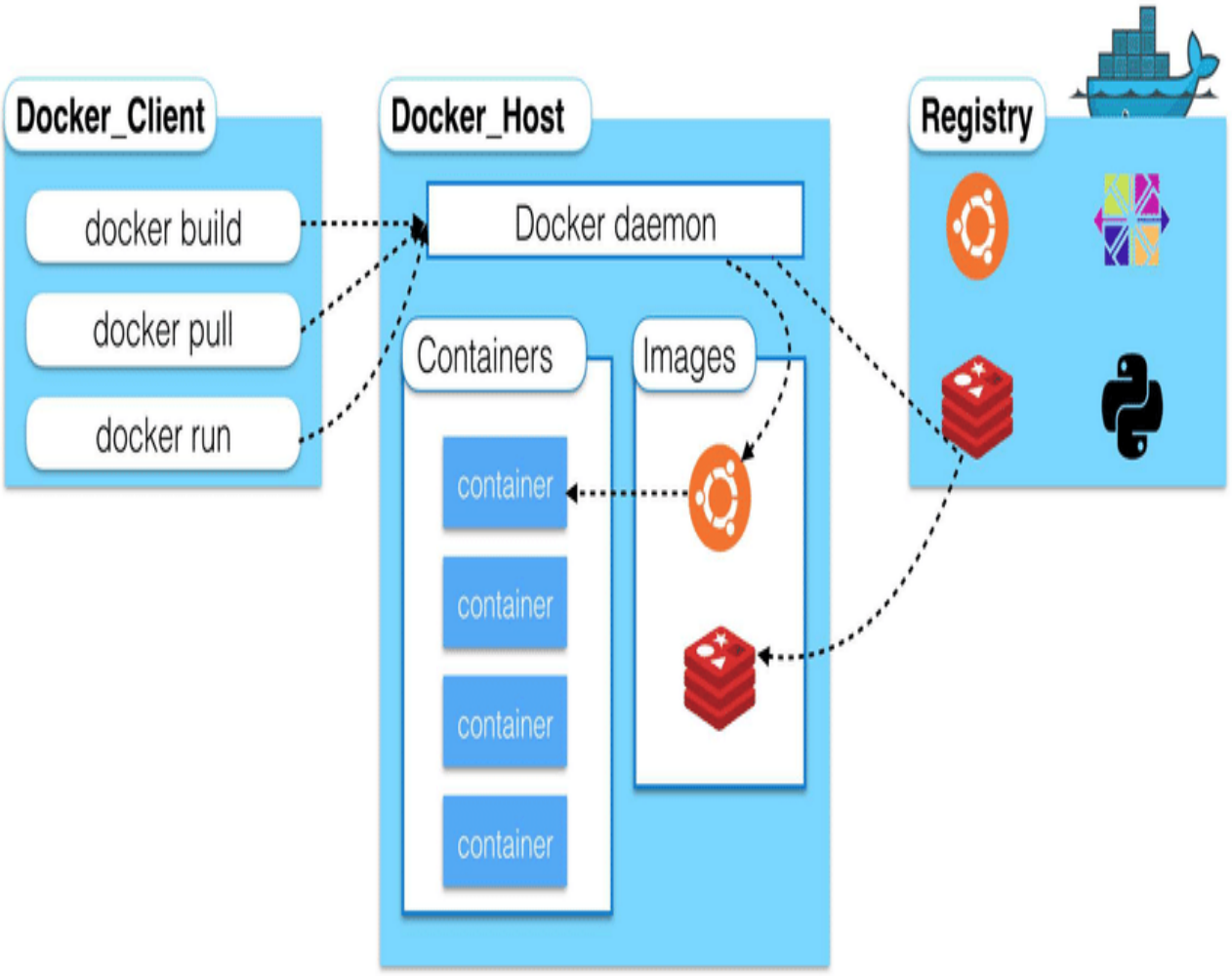
- A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster.
- The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster are referred to as nodes.



Features

- Light-weight
- Portable
- Self sufficient
- Easy and faster configuration
- Application isolation
- Increase productivity
- Swarm
- Routing Mesh
- Services
- Security Managment

Docker Architecture



Docker Engine

It is the core part of the whole Docker system. Docker Engine is an application which follows client-server architecture. It is installed on the host machine.

❑ There are three components in the Docker Engine:

- Server: It is the Docker daemon It can create and manage Docker images. Containers, networks, etc.
- Rest API: It is used to instruct Docker daemon what to do.
- Command Line Interface (CLI): It is a client which is used to enter Docker command.

Docker Daemon

- The Docker daemon is a service that runs on your host operating system.
- A daemon can also communicate with other daemons to manage Docker services.
- The Docker daemon itself exposes a REST API.
- The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

Docker Client

- To communicate with the Docker daemon to execute a certain task like running a container, build some image are all done via Docker client.
- When you use commands such as Docker run, the client sends these commands to Docker Daemon.
- The Docker client can communicate with more than one Docker daemon.

Docker Registry

- It is the location where the Docker images are stored. It can be a public Docker registry or a private Docker registry. Docker Hub is the default place of Docker images, its stores' public registry. You can also create and run your own Private registry.
- When you execute Docker pull or Docker run commands, the required Docker image is pulled from the configured registry. When you execute Docker push command, the Docker image is stored on the configured registry.

Docker Object

❑ There are mainly 5 objects:

- Images
- Container
- Volumes
- Network
- Plugins

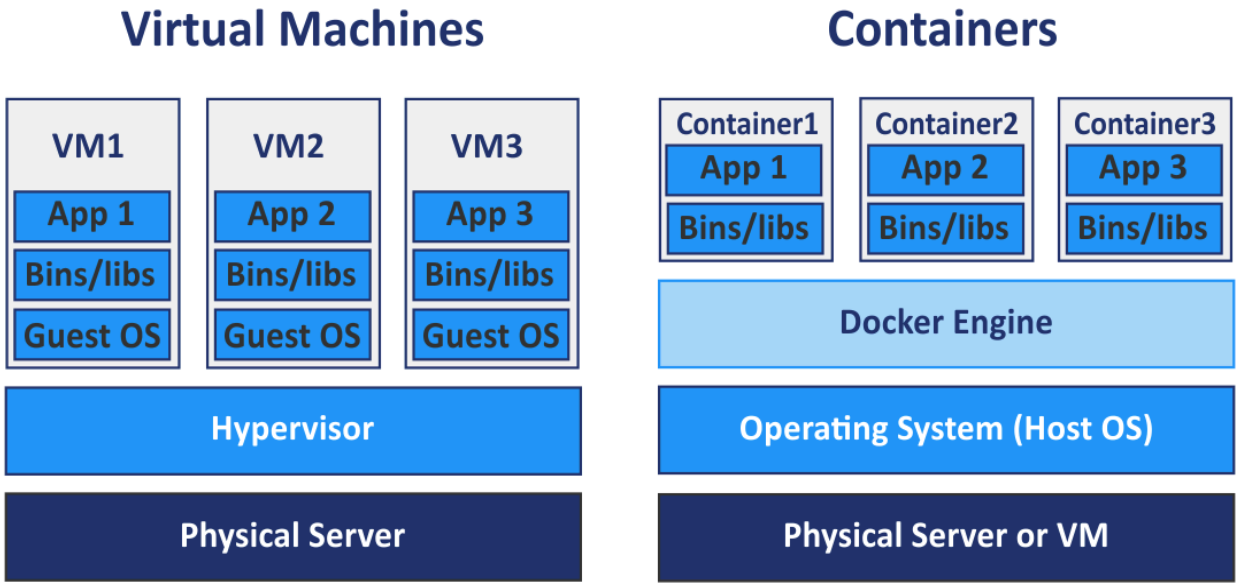
1. Docker Images

- Docker images are read-only templates with instructions to create a Docker container.
- Docker image can be pulled from a Docker hub and used as it is, or you can add additional instructions to the base image and create a new and modified Docker image.
- Docker image has a base layer which is read-only, and the top layer can be written. When you edit a Docker file and rebuild it, only the modified part is rebuilt in the top layer.

2. Docker Container

- After run a Docker image, it creates a Docker container.
- All the applications and their environment run inside this container.
- You can use Docker API or CLI to start, stop, delete a docker container.
- 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.

Docker Container vs. Virtual Machine



Compare VMs vs. Container

Virtual Machine	Docker Container
Each VM runs in its own OS	All containers share the host OS.
Host OS can be different than the guest OS	Host OS and Container OS has to be the same (Linux Kernel)
VM are always running.	Containers stop when the command it is started with completes
Startup time in minutes	Startup time in milliseconds
VMs snapshots are used sparingly	Images are built incrementally on top of another like layers.
You can run multiple VMs on a laptop for example	Can run many docker containers on a laptop.
Fully isolated and hence more secure	Process-level isolation and hence less secure

Container and VM Together

- Container and VMs used together provide a great deal of flexibility in deploying and managing app.

3. Docker Volumes

- Docker volumes are file systems mounted on Docker containers to preserve data generated by the running container.
- The volumes are stored on the host, independent of the container life cycle.
- This allows users to back up data and share file systems between containers easily.

4. Docker Networks

- Docker networking allows you to attach a container to as many networks as you like. You can also attach an already running container.
- Docker secures the network by managing rules that block connectivity between different Docker networks.
- the Docker Engine creates the necessary Linux bridges, internal interfaces, iptables rules, and host routes to make this connectivity possible.

5. Docker Plugin

- **Docker plugins** are out-of-process extensions which add capabilities to the **Docker** Engine.

- **Docker** Engine's **plugin** system allows you to install, start, stop, and remove **plugins** using **Docker** Engine.
- A plugin is a process running on the same or a different host as the docker daemon, Which registers itself by placing a file on the same docker host in one of the plugin directories described in Plugin discovery
- Plugin have human-readable names, which are short, lowercase string. For example, flocker or weave.

How Does Docker Work?

- Docker uses a client-server architecture.
- We can build Docker images that hold your applications.
- We can create Docker containers from those Docker images to run your applications.
- We can share those Docker images via Docker Hub or your own registry.

Docker Workflow?

Docker has three parts:

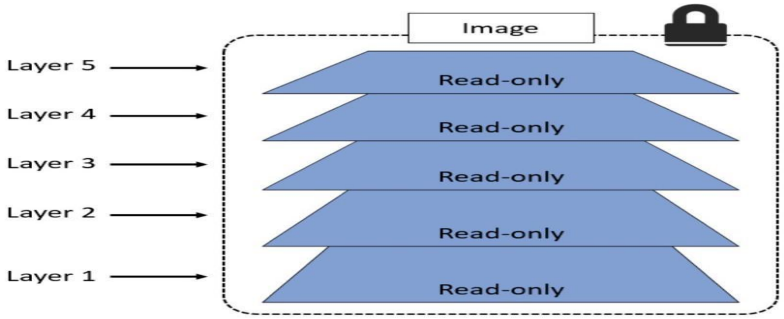
- Docker File
- Docker Image.
- Docker Containers.

Docker File?

- It is a text document that contains all commands.
- It describes step-by-step instructions of all the commands you need to run to assemble a Docker Image.

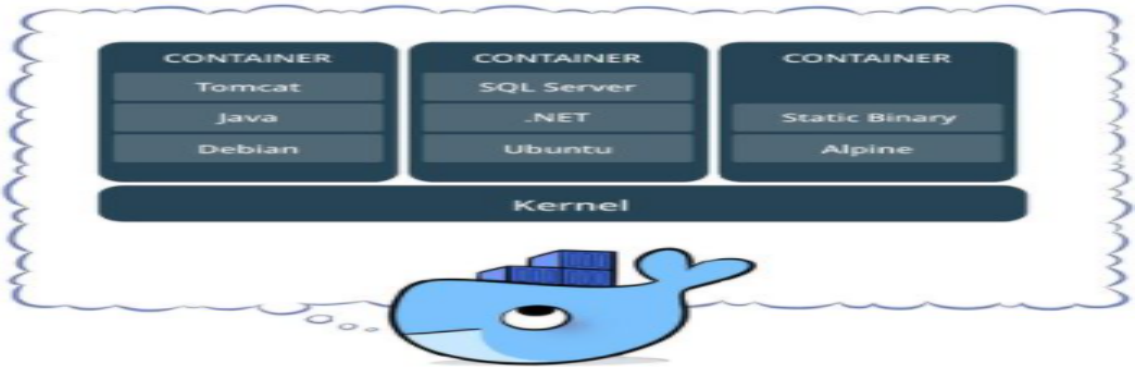
Docker Images

- A Docker image is a read-only template.
- We can use Docker Run to run the image and create a container.
- Docker Images are stored in the Docker Registry. Eg : Docker Hub



Docker Containers

- Containers are a form of operating system virtualization.
- A single container might be used to run anything from a small micro service or software process to a larger application.



Docker Container Lifecycle

❖ The Life of a Container

Conception

- Build an Image from a Docker File

Birth

- RUN (create + start) a container

Reproduction

- COMMIT a container to a new image
- RUN a new container from an image

Sleep

- KILL a running container

Wake

- Start a stopped container

Death

- RM (delete) a stopped container

Extinction

- RMI a container image (delete image)

Conclusion

- Easy to build, run and share containers.
- Rapidly expanding ecosystem.
- Better performance vs. Virtual machine.

- Layered file system gives us git like control of images
- Reduce Complexity
- Google is expected to tightly intergrade container with its LaaS and PaaS offerings