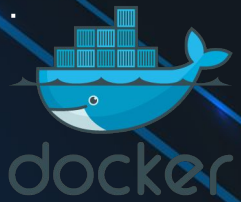
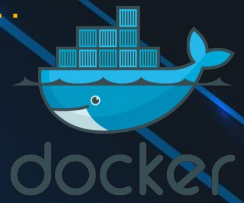


Docker

It's all about apps

We Isolate Services

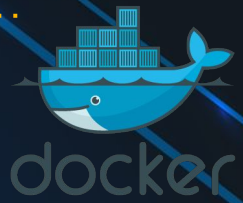


- To host our apps we need Infrastructure.
- We Use VM's/Cloud Computing to setup Infra
- We Isolate our service in OS of VM

- Because of Isolation we end up setting up multiple VM's/Instances.
- VM's/Instances will be overprovisioned.
- Results in High CapEx and OpEx

VM's are expensive

- Every VM has OS
- OS needs nurturing
- OS Needs Licensing
- OS takes time to boot
- VM's are Portable but Bulky.
- VM needs Resources for its OS
- All this to Isolate services



Point to be Noted.

- Isolating services are IMP (Need OS)
- High availablity achived by multiple instances/vm's
- Portablity Matters or Eases the Deployment.
- All this raises CapEx and OpEx

Isolation without OS?

Imagine Multiple Services running in same OS but isolated.

Containers

Process running in a Directory.

Container

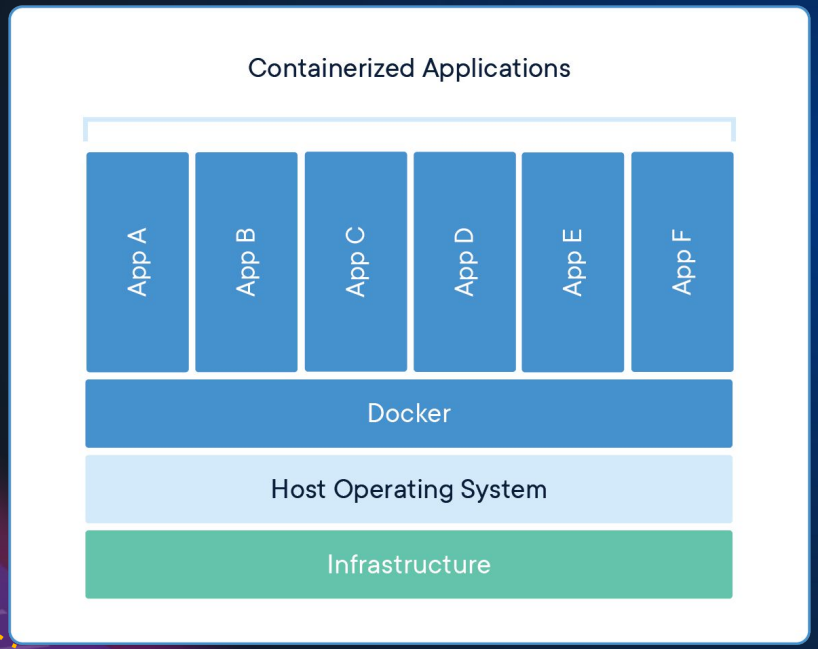
- A Process[Isolated]
- A Directory[Namespace, cgroup]
- Necessary bin/lib in the Directory
- A directory with IP address to connect.

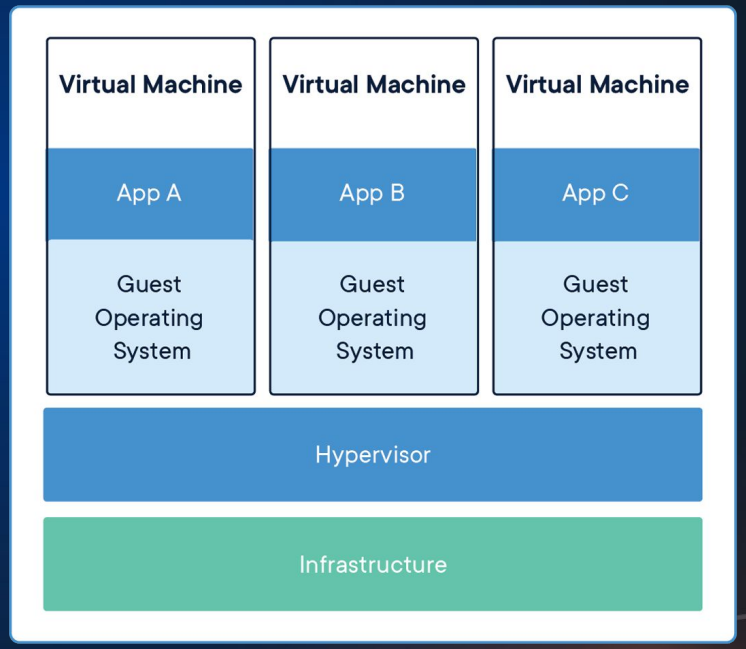
Container

- Containers share the machine's OS system kernel and therefore do not require an OS per application.

- A container is a standard unit of software that packages up
  - Code
  - Dependencies

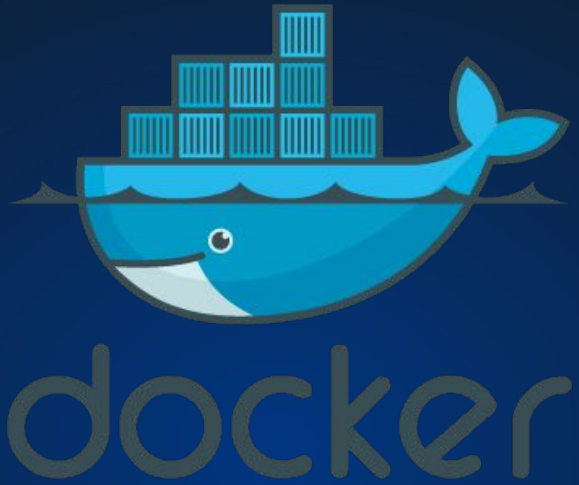
Container vs VM





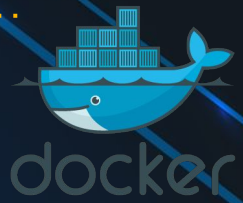
VM vs Container

- Containers offer Isolation not Virtualization
- Containers are OS virtualization
- VM's are Hardware virtualization
- VM needs OS
- Containers don't need OS.
- Containers uses Host OS for Compute Resource



Docker

Manages your Containers



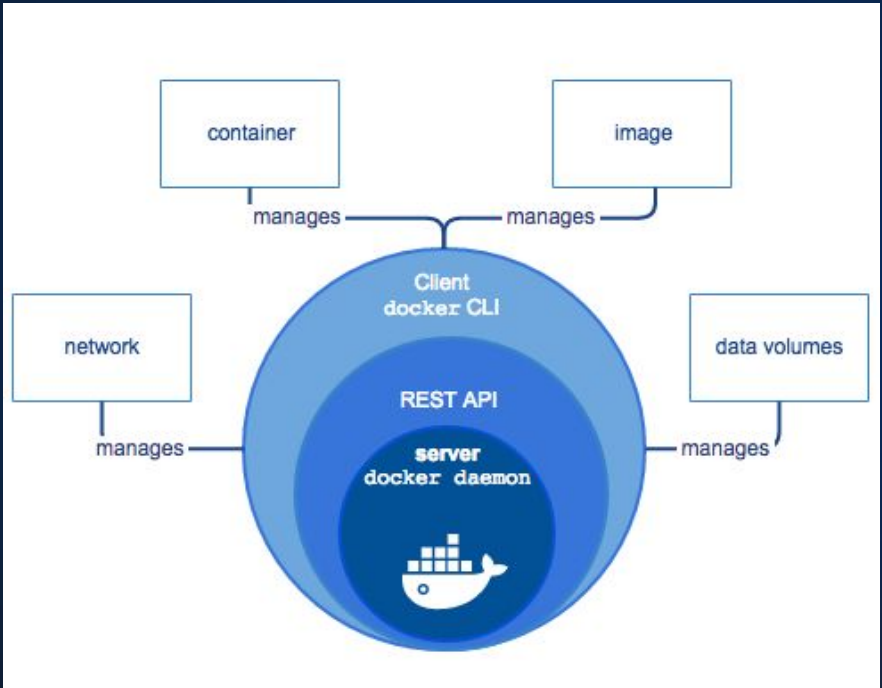
Docker History

- Formerly Known as DotCloud Inc
- Into PAAS Business
- Used LXC (Linux Containers)
- Saved CapEx by using Containers instead of VM's
- Developed TOOLS to manage containers.
- Business Failed.
- Made their tools OpenSource project knows as Docker.
- Got Funding
- Changed name to Docker Inc

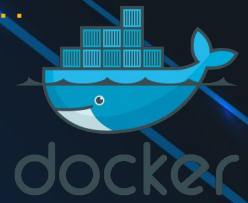
So What's Docker?

- Docker Inc
- Docker Engine
- Docker Project (OpenSource)

Docker Engine



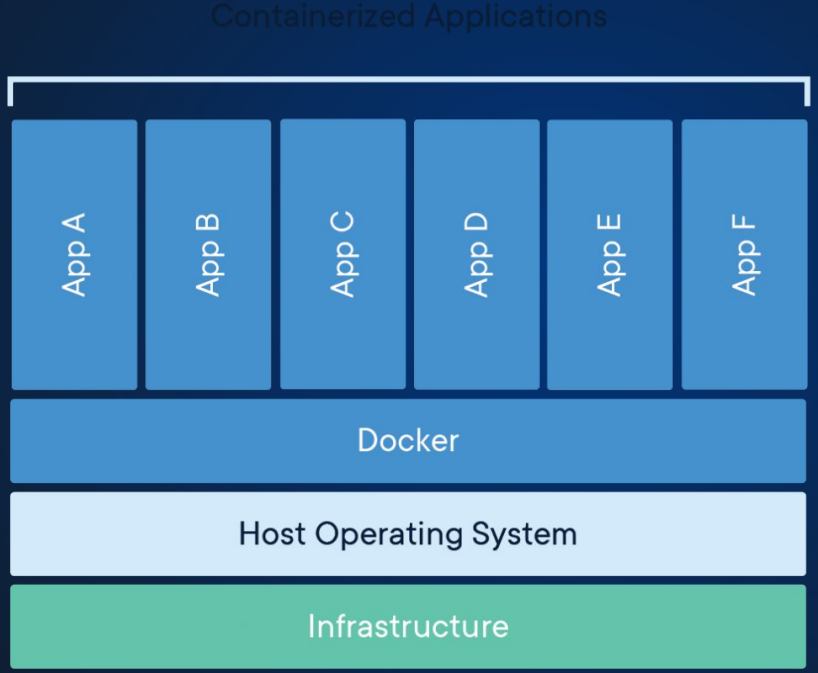
Docker Containers



Docker containers that run on Docker Engine:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

Docker Containers



Docker Installation

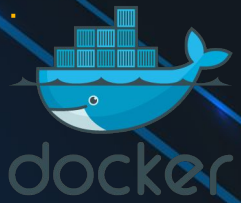
- Linux or Windows
- Windows Containers runs on Windows OS
- Linux Containers runs on Linux OS
- Docker Desktop

3

DockerHub

Registry for Docker Images

Docker Image

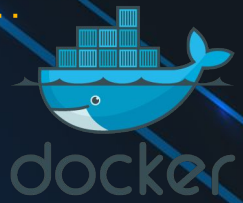


- A stopped Container like vm Image.
- Consist of multiple layers.
- An app will be bundled in an Image.
- Containers runs from Images
- Images are called as Repositories in Registries.

Docker Images

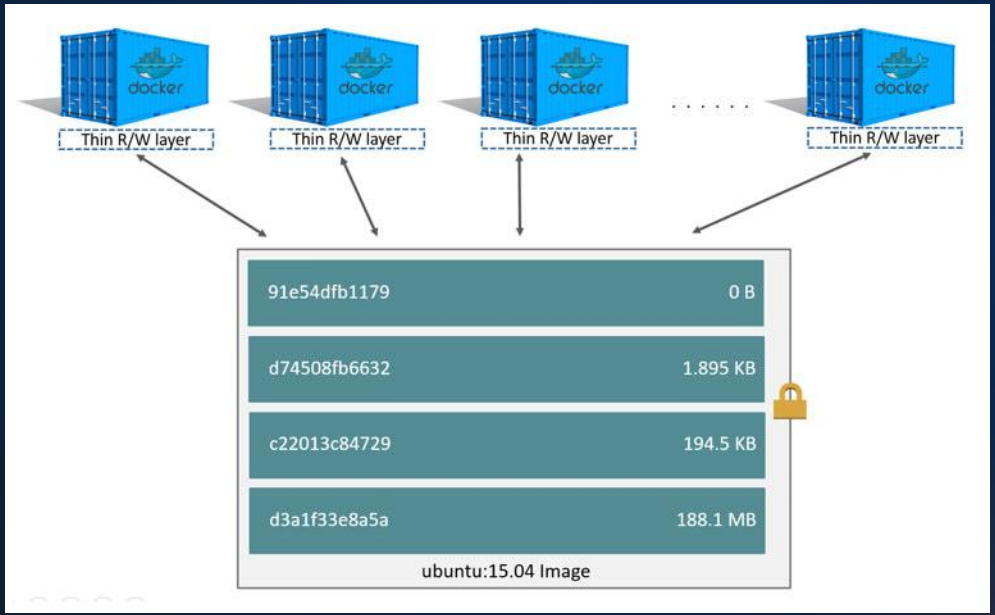
Images become containers when they run on Docker Engine.

Docker Registries



- Storage for Docker Images.
- Dockerhub is default registry
- Cloud based Registries.
  - Dockerhub
  - GCR (Google Container Registry)
  - Amazon ECR
- Inhouse or Local Registries
  - Nexus 3 +
  - Jfrog Artifactory
  - DTR (Docker trusted Registry)

Containers Runs from Images

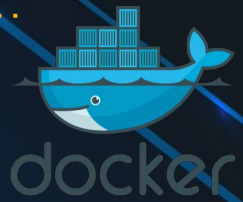


4

Creating Container

docker run

Docker Commands



- # docker images => Lists Images locally
- # docker run => command creates a new container.
- # docker ps => Lists running container
- # docker ps –a => Lists all the containers
- # docker exec => executes commands on containers.
- # docker start/stop/restart/rm
- # docker rmi => Remove docker images.
- # docker inspect => Detail of container & Image

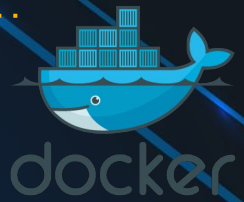
https://docs.docker.com/engine/reference/commandline/cli/

4

Container Volumes

Persistent storage for volatile containers

Container Data

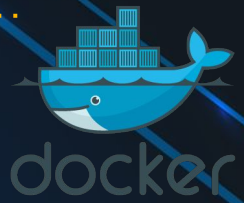


- The data doesn't persist when that container no longer exists, and it can be difficult to get the data out of the container if another process needs it.
- A container's writable layer is tightly coupled to the host machine where the container is running. You can't easily move the data somewhere else.

Docker has two options for containers to store files in the host machine

- *Volumes*
  - Managed by Docker (/var/lib/docker/volumes/ on Linux
- *Bind Mounts*
  - Stored *anywhere* on the host system

Container Data



Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.

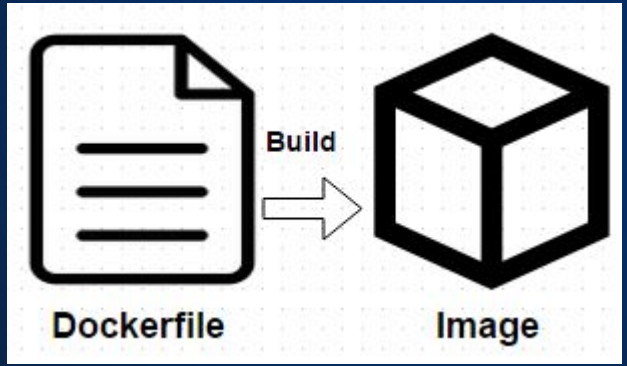
Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.

tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

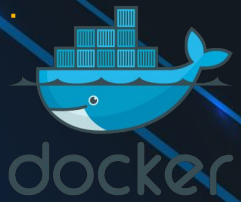
Build Images

Dockerfile contains information to build Images

Dockerfile build Image

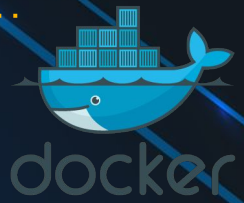


Dockerfile Instructions



- FROM => Base Image
- LABEL => Adds metadata to an image
- RUN => execute commands in a new layer and commit the results.
- ADD/COPY => Adds files and folders into image.
- CMD => Runs binaries/commands on docker run
- ENTRYPOINT => Allows you to configure a container that will run as an executable.
- VOLUME => Creates a mount point and marks it as holding externally mounted volumes.
- EXPOSE => Container listens on the specified network ports at runtime

Dockerfile Instruction

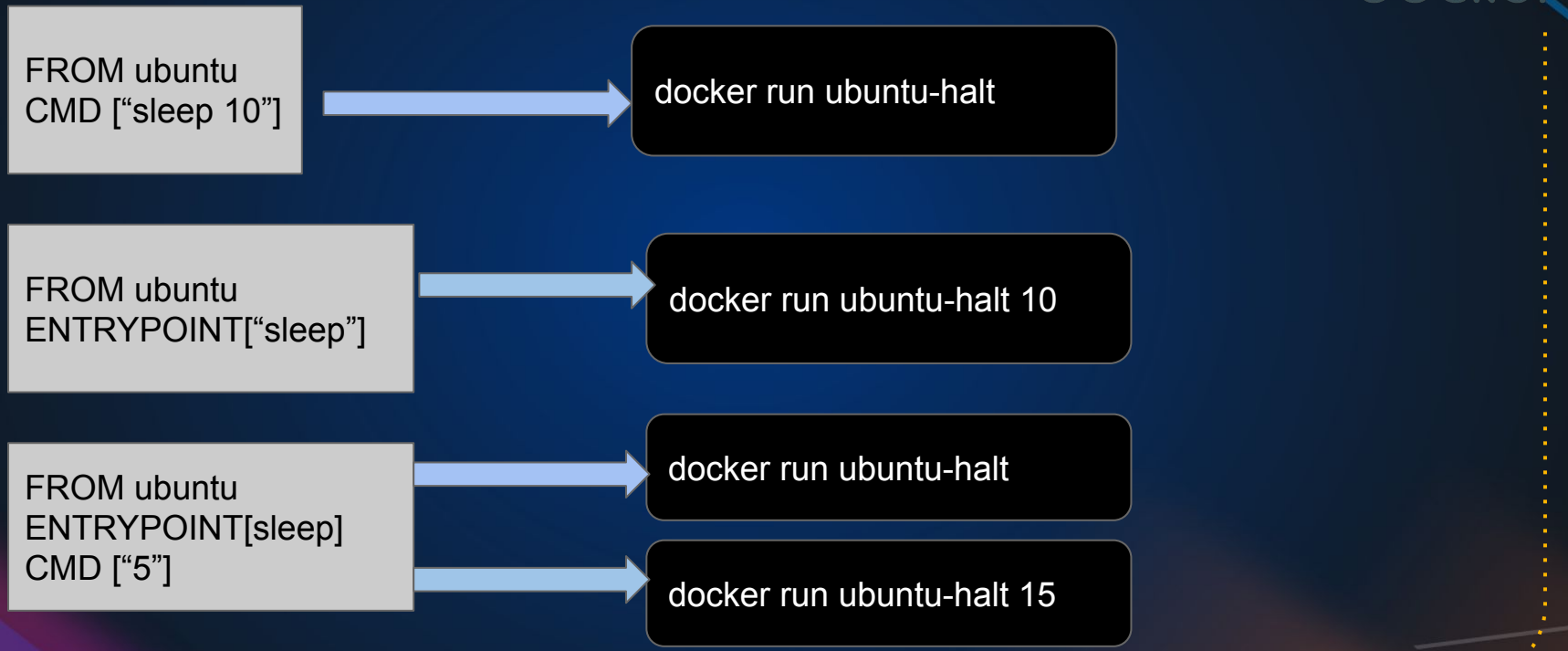


- ENV => Sets the environment variable
- USER => Sets the user name (or UID)
- WORKDIR => Sets the working directory
- ARG => Defines a variable that users can pass at build-time
- ONBUILD => Adds to the image a *trigger* instruction to be executed at a later time

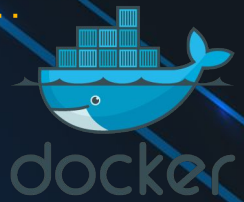
Refer Documentation

https://docs.docker.com/engine/reference/builder/

Command & Entrypoint

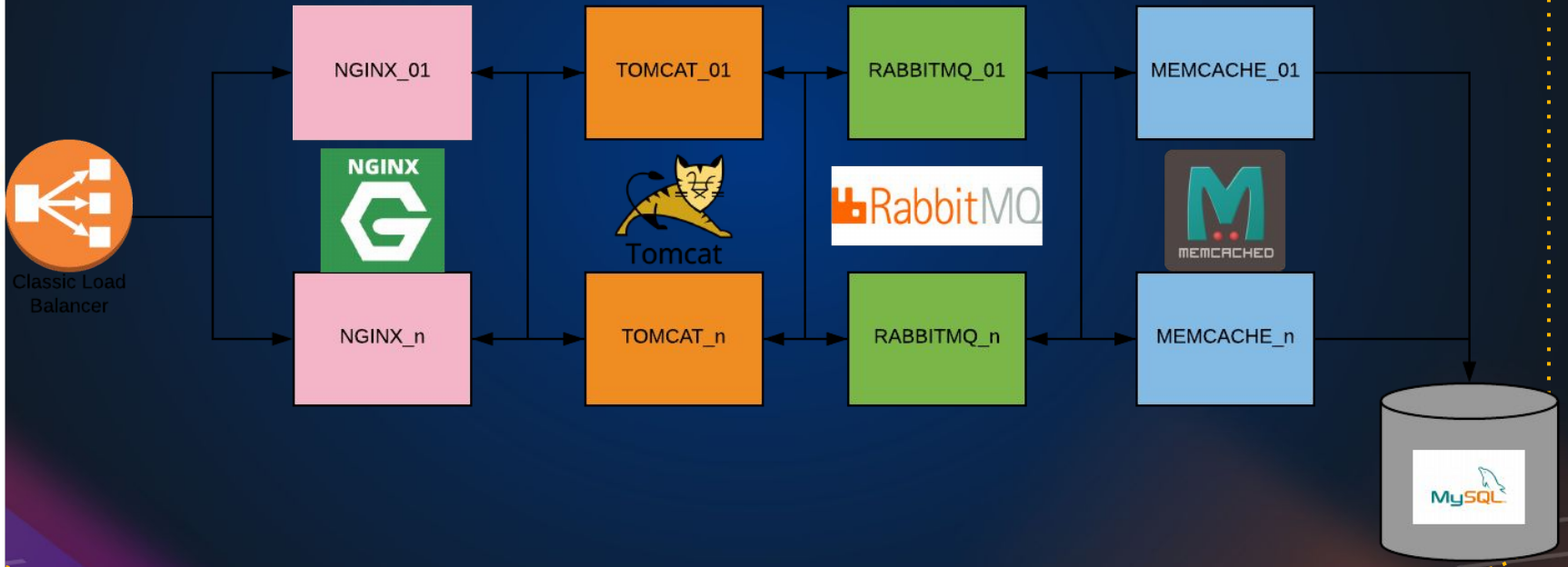


Docker Build & Publish



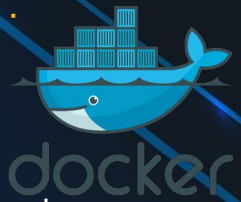

```
# docker build –t Account-Name/Image-Name Dockerfile-Path

# docker login

# docker push Account-Name/Image-Name
```

Vprofile Project's Architecture



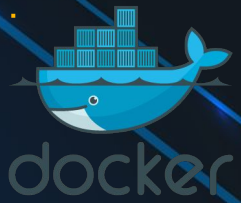
Docker Networking

Network Drivers



- bridge: The default network driver. Bridge networks are usually used when your applications run in standalone containers that need to communicate.
- host: For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly
- overlay: Connect multiple Docker daemons together and enable swarm services to communicate.

Network Drivers



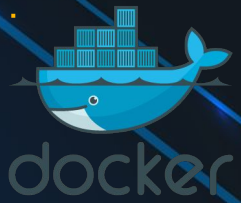
- macvlan: Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network.

- Network plugins: You can install and use third-party network plugins with Docker.:

Refer Documentation

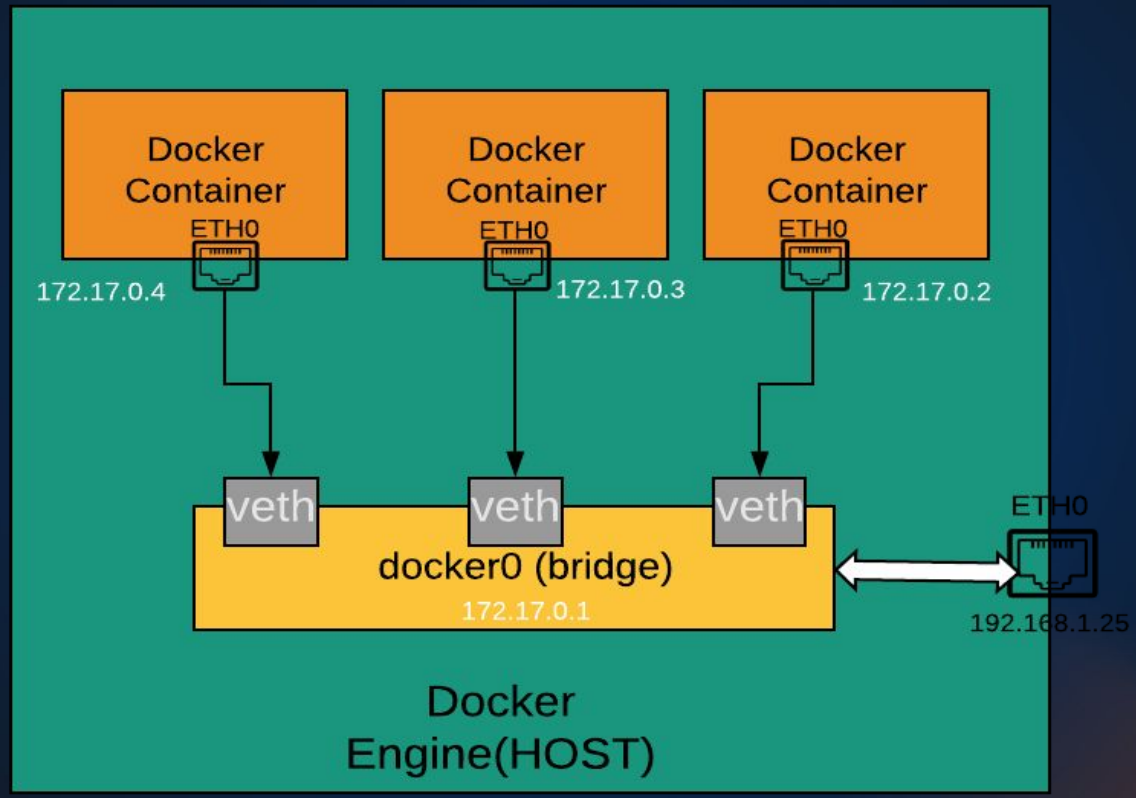
https://docs.docker.com/network/

Network Drivers Use Cases

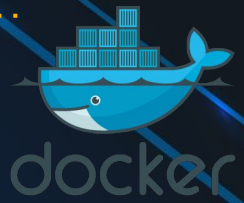


- User-defined bridge networks are best when you need multiple containers to communicate on the same Docker host.
- Host networks are best when the network stack should not be isolated from the Docker host, but you want other aspects of the container to be isolated.
- Overlay networks are best when you need containers running on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- Macvlan networks are best when you are migrating from a VM setup or need your containers to look like physical hosts on your network, each with a unique MAC address.
- Third-party network plugins allow you to integrate Docker with specialized network stacks.

Bridge Network



Container Bridge Networking



- Container created gets Name & IP address
- Container default gateway is bridge
- Containers can connect each other with IP & Name
- Container's name resolution is done automatically

Docker Compose

Manage containers from docker-compose.yml file