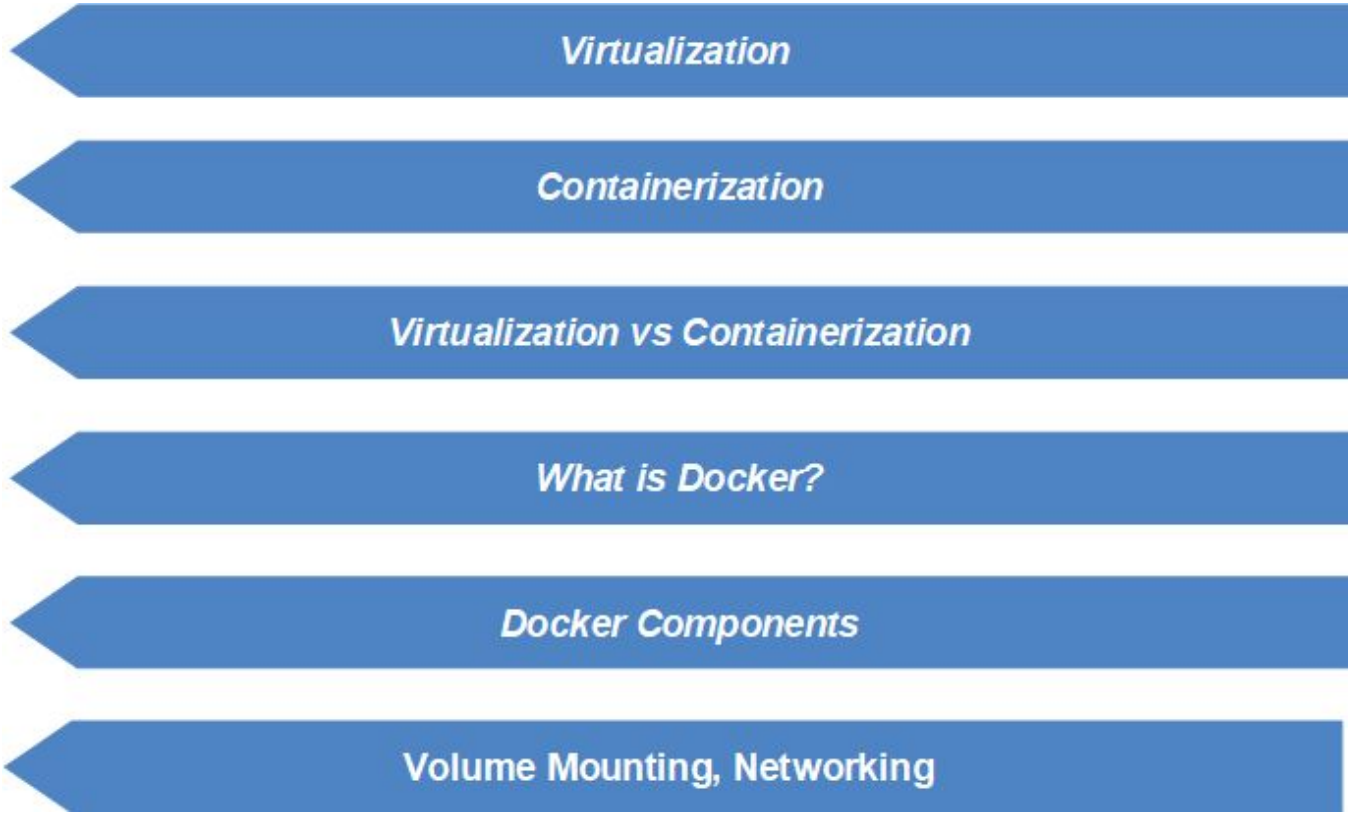
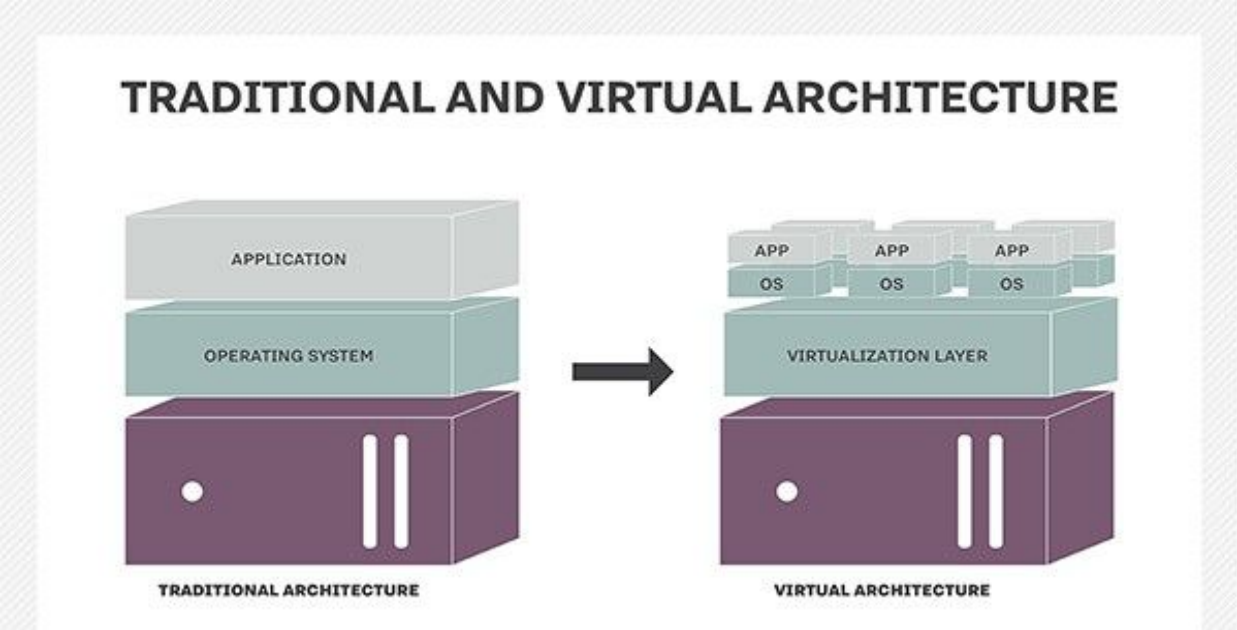
DOCKER

Agenda


Virtualization
Containerization
Virtualization vs Containerization
What is Docker?
Docker Components
Volume Mounting, Networking

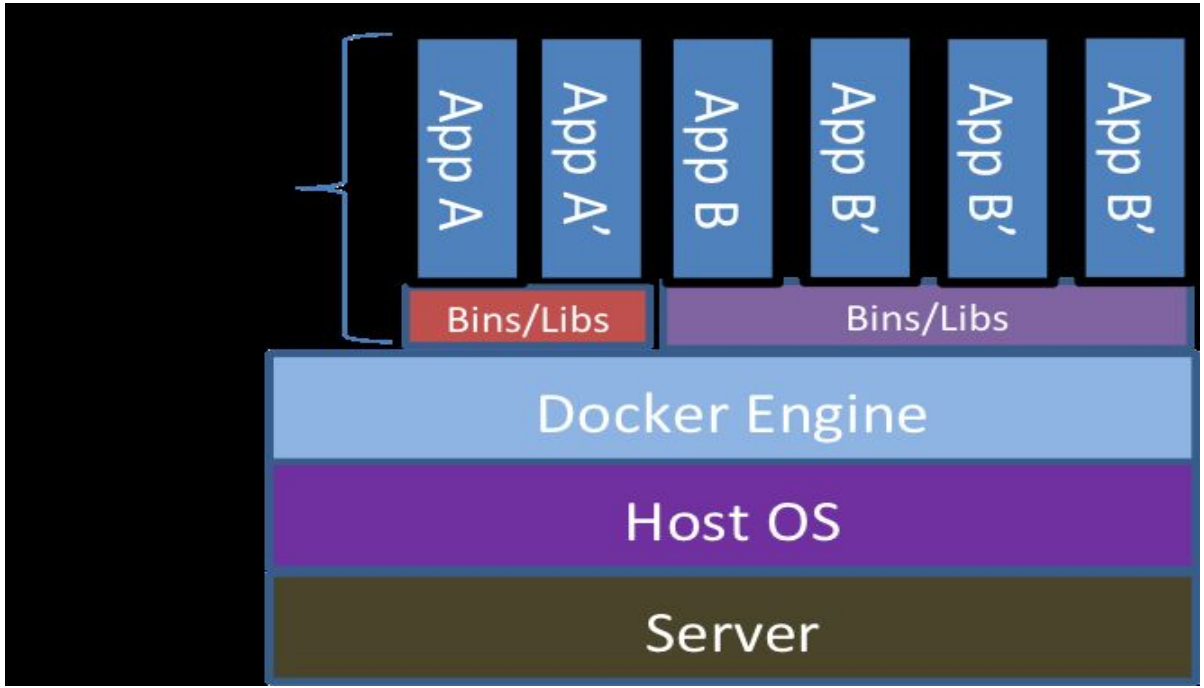
Virtualization

Virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, storage devices, and computer network resources

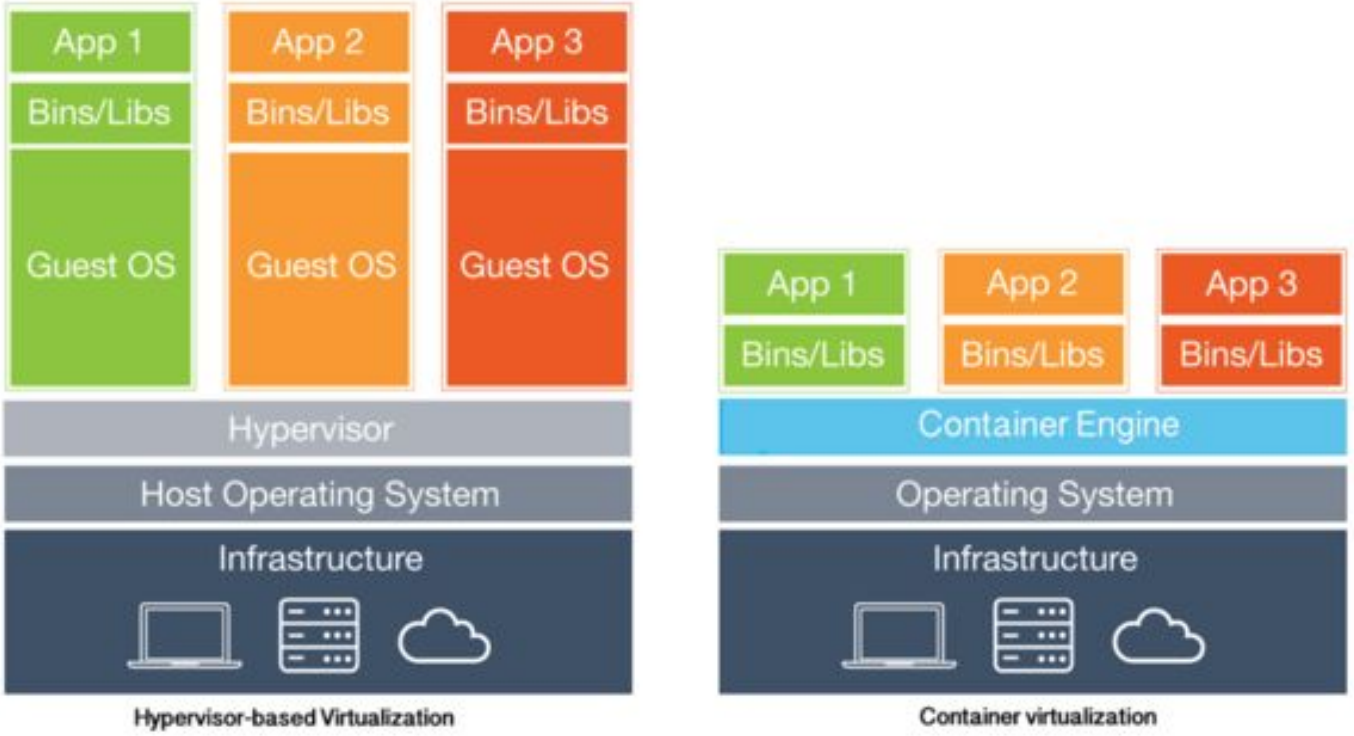


Containerization

Containerization is an operating system level (OS-level) virtualization method for deploying and running distributed applications without launching an entire virtual machine (VM) for each app.

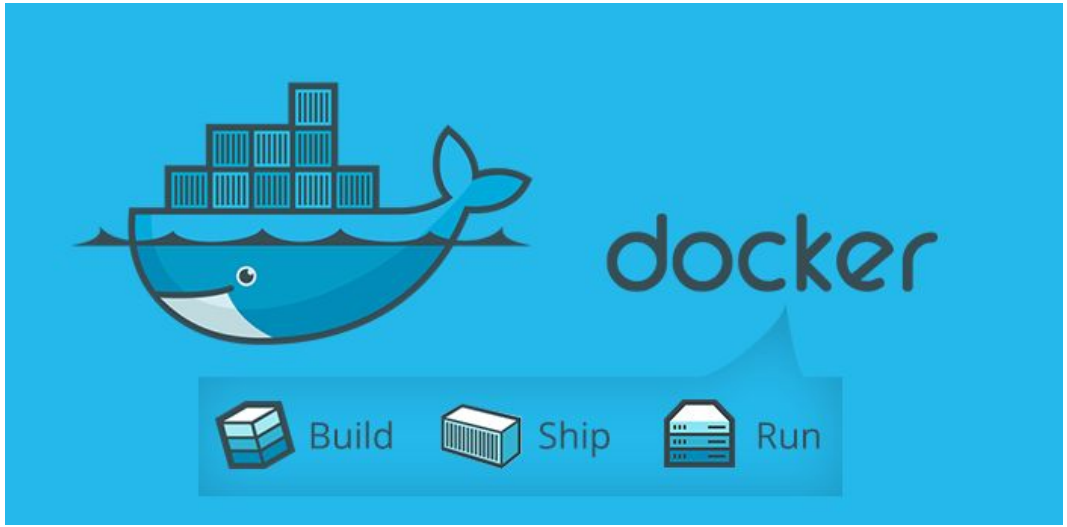


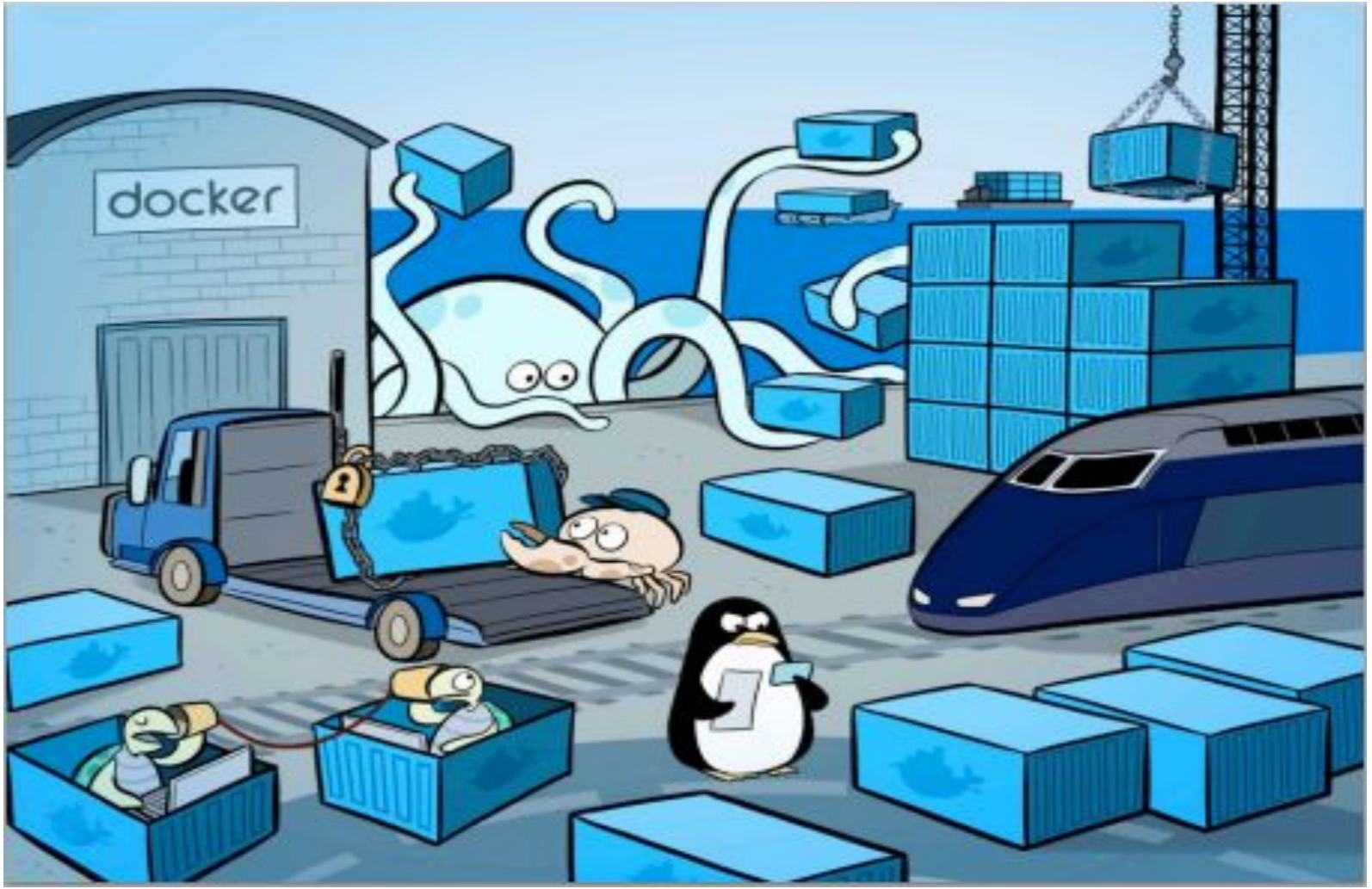
Virtualization vs Containerization



Introduction of Docker

- open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.
- provides an additional layer of abstraction and automation of OS level virtualization on Windows and Linux.
- open source project that automates the deployment of applications inside Software containers.




docker

Docker features

- Docker containers are lightweight and fast
- Docker containers run (almost) everywhere. You can deploy containers on desktops, physical servers, virtual machines, into data centers, and up to public and private clouds.
- Docker containers are Portable
- Can run on any Linux system that supports LXC (today).
- 0.7 release includes support for RedHat/Fedora family.
- Future plans to support other container tools (lmctfy, etc.)
- Possible future support for other operating systems (Solaris, OSX, Windows?)

Docker Components

1. Docker Deamon-Runs on host machine
2. Docker client-Primary User interface to Docker
3. Docker images-Read-only templates
4. Docker registries-Hold images
5. Docker Containers-Hold everything needed for the application to run

Docker Life cycle

The Life of a Container

– **Conception**

• BUILD an Image from a Dockerfile

– **Birth**

• RUN (create+start) a container

– **Reproduction**

• COMMIT (persist) a container to a new image

• RUN a new container from an image

– **Sleep**

• KILL a running container

– **Wake**

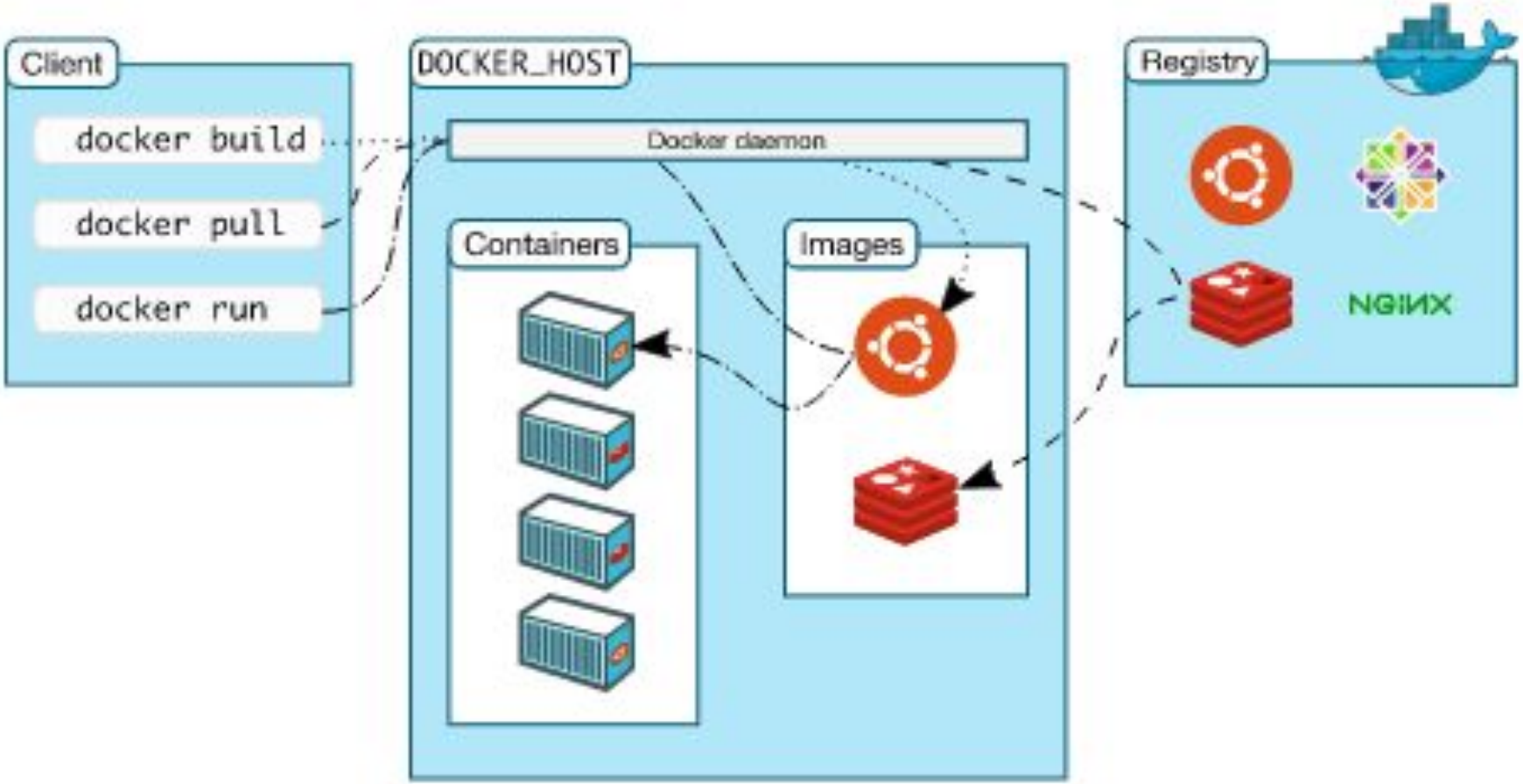
• START a stopped container

– **Death**

• RM (delete) a stopped container

• Extinction

– **RMI a container image (delete image)**


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

Docker Images

Persisted snapshot that can be run

images: List all local images

run: Create a container from an image and execute a command in it

tag: Tag an image

pull: Download image from repository

rmi: Delete a local image This will also remove intermediate images if no longer used Layered File System

Docker File

- Create images automatically using a build script:«Dockerfile»
- Can be versioned in a version control system like Gitor SVN, along with all dependencies
- Docker Hub can automatically build images based on dockerfiles on Github

Dockerfile:

FROM ubuntu

ENV DOCK_MESSAGE Hello My World

ADD dir /files

CMD ["bash", "someScript"]

docker build [DockerFileDir]

docker inspect [imageId]

Some examples of Dockerfile instructions are:

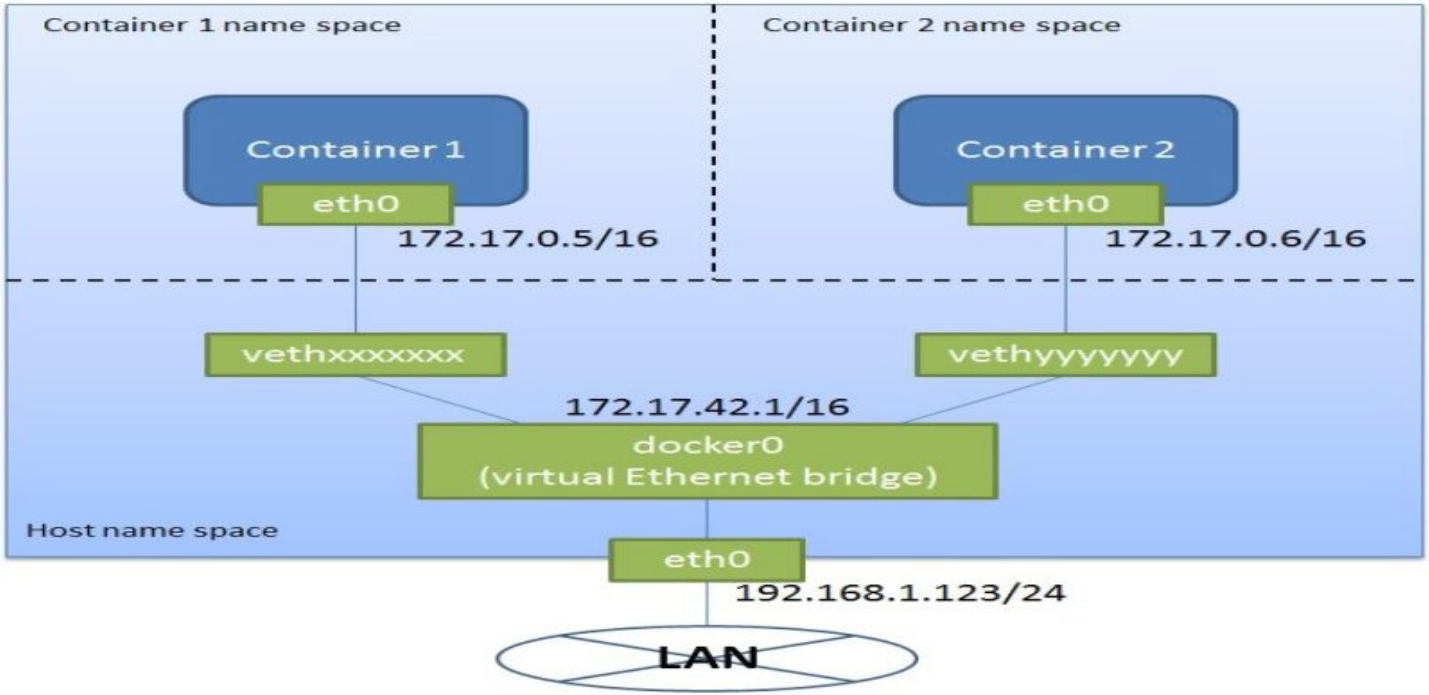
- Specify the base image (FROM)
- Specify image metadata (LABEL)
- Run a command (RUN)
- Add a file or directory (ADD)
- Create an environment variable (ENV)
- What process to run when launching a container from this image (CMD)

Docker Volumes

- Data volumes provide several useful features for persistent or shared data:
- Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization. (Note that this does not apply when mounting a host machine.)
- Data volumes can be shared and reused among containers.
- Changes to a data volume are made directly.
- Changes to a data volume will not be included when you update an image.
- Data volumes persist even if the container itself is deleted.
- Example for mount a folder from host machine to a container
- docker run -d -P --name web -v /webapp training/webapp python app.py

Docker networking

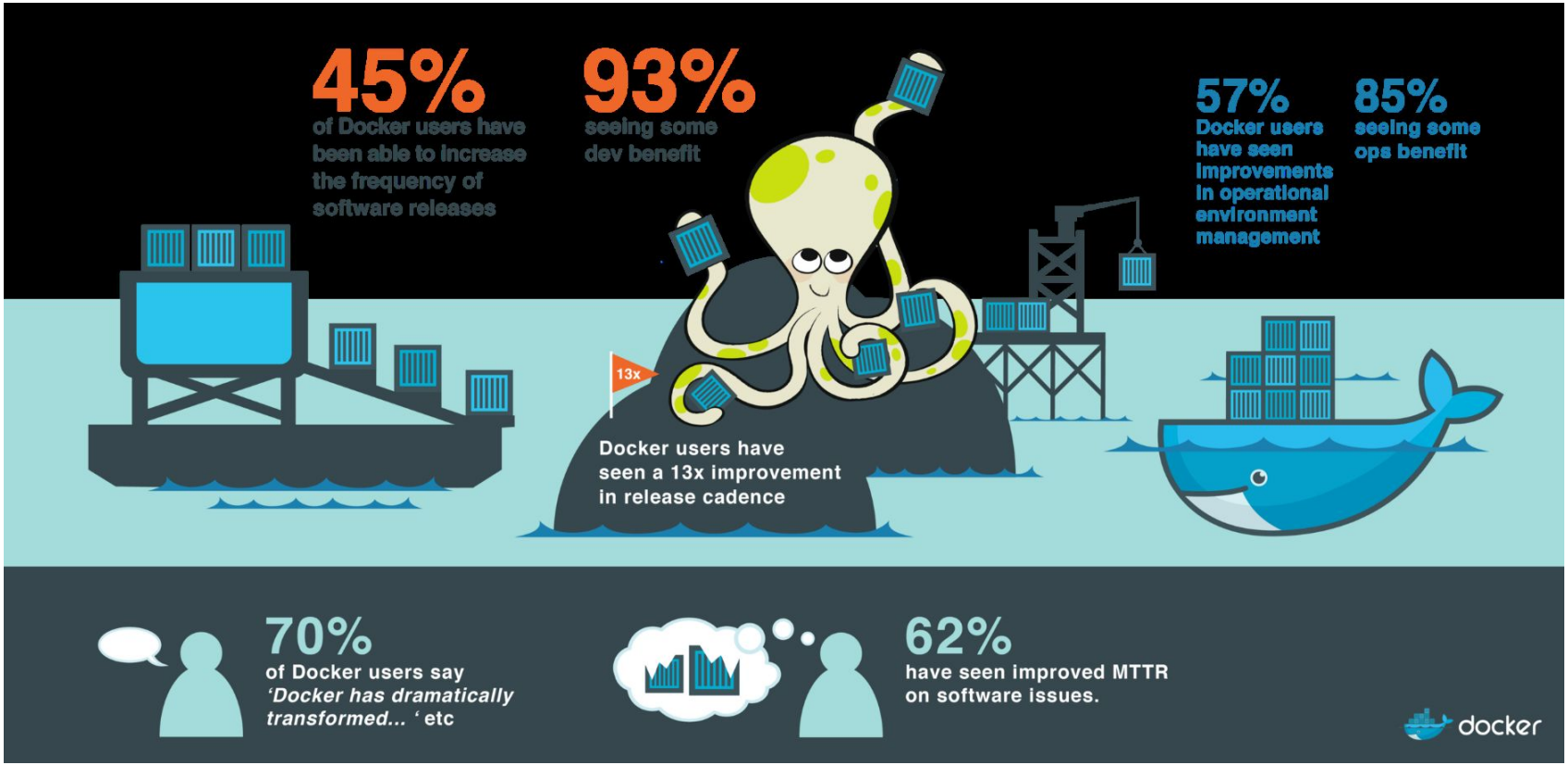
By default, Docker containers are not connected to a local network. They are connected only to a virtual network Docker creates as like below:



Docker Use cases

- DevOps
- Development Environment
- Environments for Integration Tests
- Quick evaluation of software
  - - Microservices
- Unified execution environment (dev -> test -> prod (local,VM, cloud, ...)

Results after using Docker



Thank you