Docker Fundamentals

Keywords

Docker, Container, Images, Volumes, Swarm, Continous Integration, Docker Networking, Compose,

Private Registry, Swarm, Scaling

References

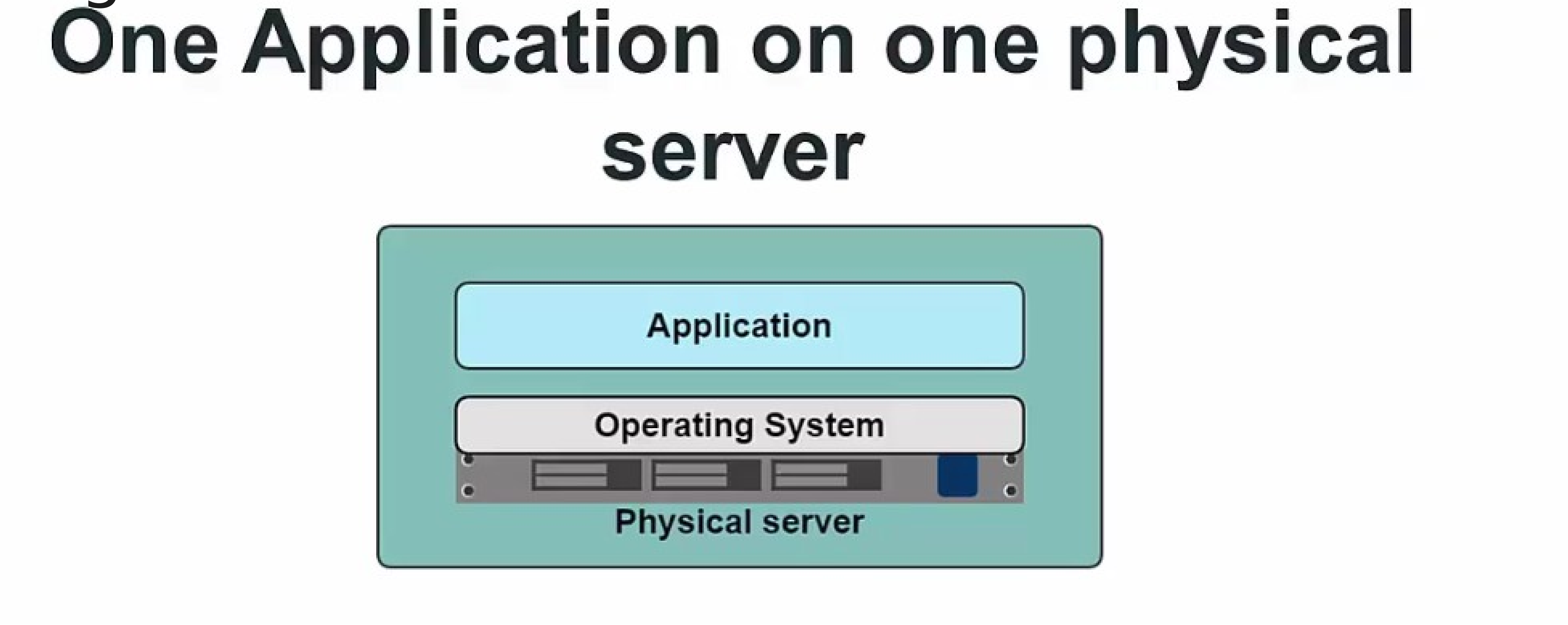
- Docker Documentation https://docs.docker.com/
- Learning Docker Second Edition – 2017

  Jeeva S. Chelladhurai, Vinod Singh, Pethuru Raj

Intoduction to Docker

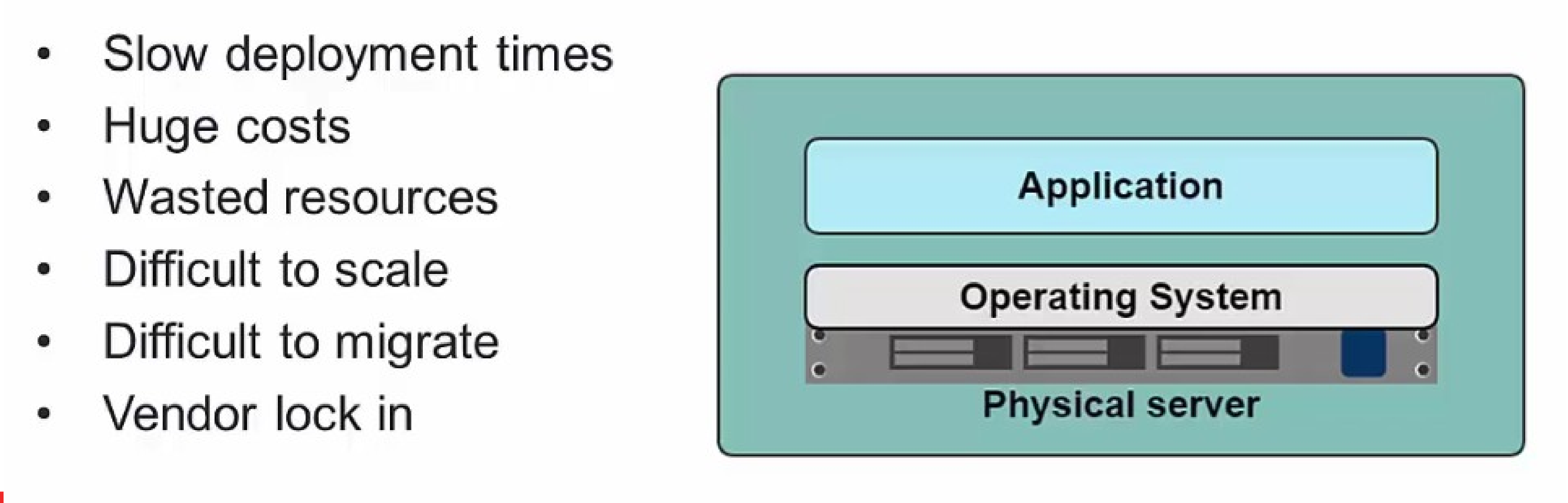
A History lesson

- In the dark ages



One Application on one physical server

Problem in the past



- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in

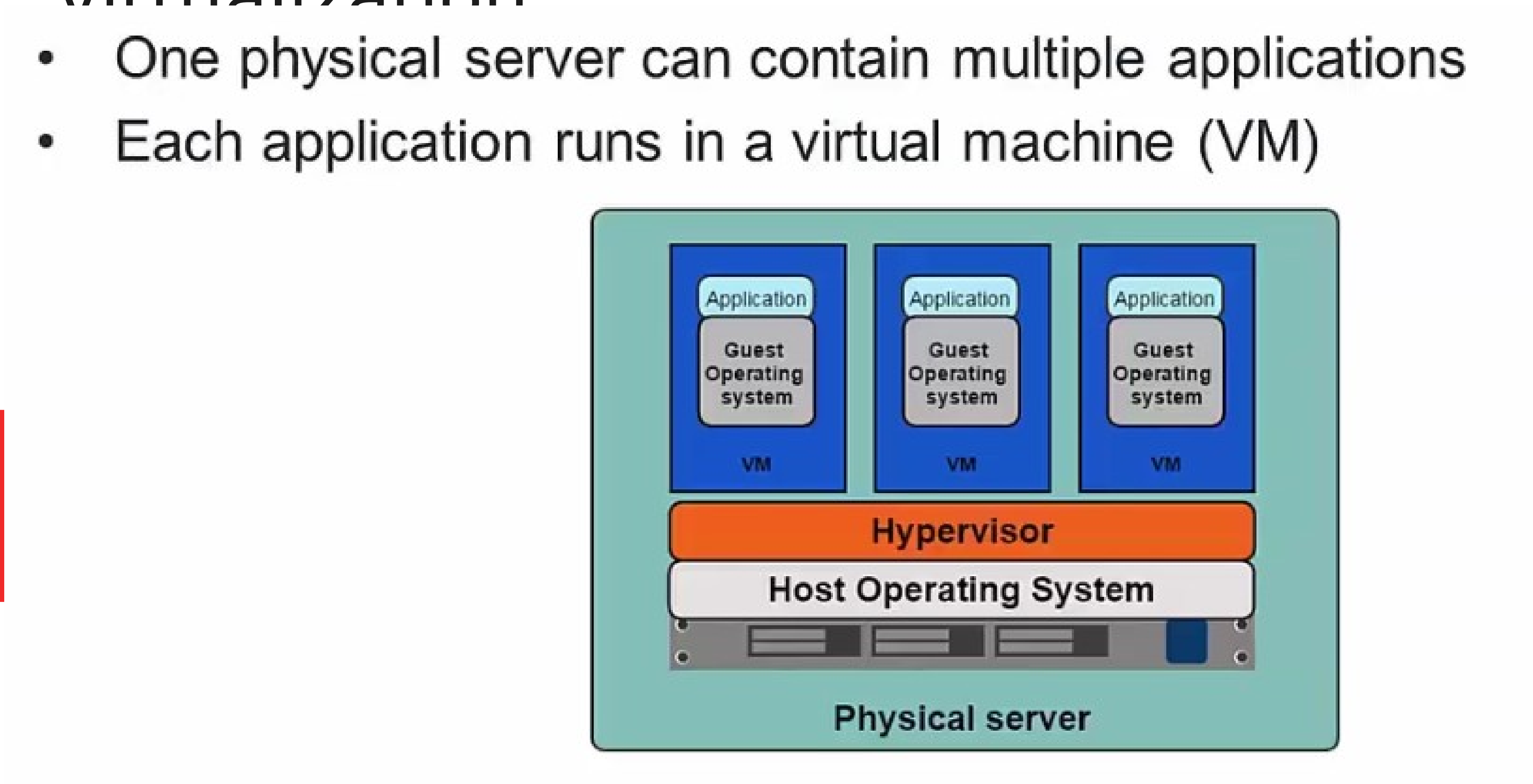
Application

Operating System

Physical server

A History lesson

- Hypervisor-based virtualization



- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)

Benefits of VMs

- Better resource pooling

  - One physical machine divide into multiple virtual machines

- Easier to scale
- VM’s in the cloud

  - Rapid elasticity

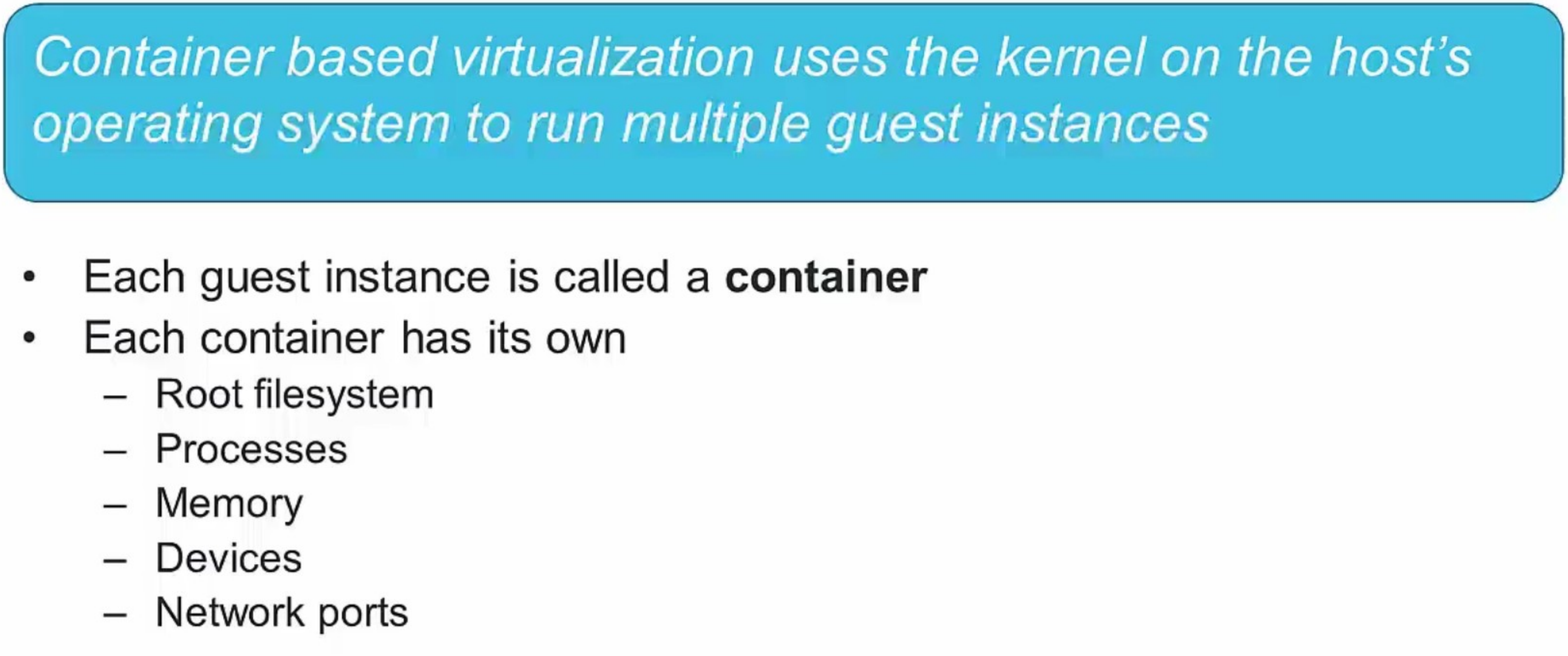
  - Pay as you go model

Limitations of VMs

- Each VM still requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system
- The more VM’s you run, the more resources you need
- Guest OS means wasted resources

Introducing Containers

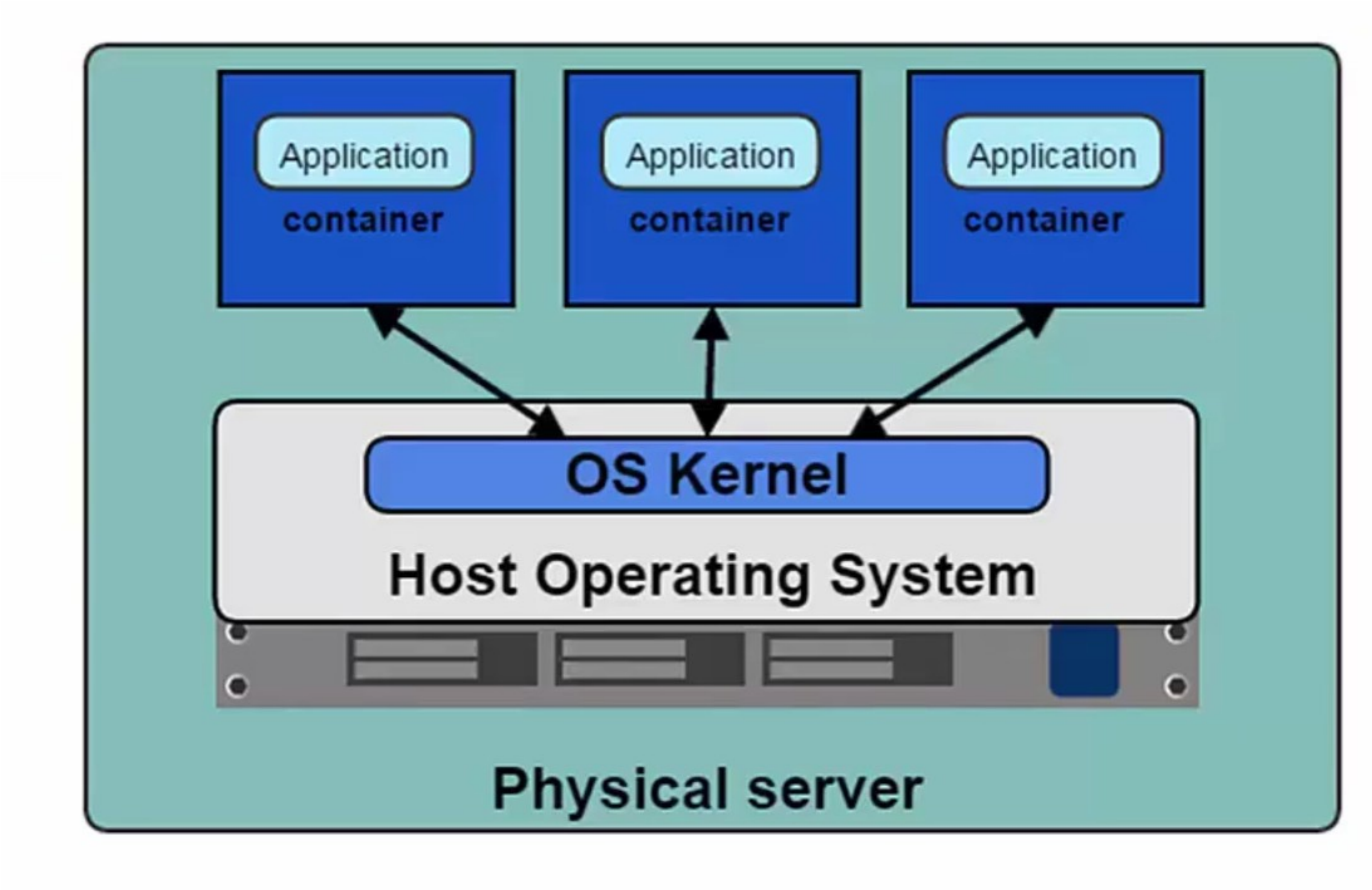
Containers



Container based virtualization uses the kernel on the host's operating system to run multiple guest instances

- Each guest instance is called a **container**
- Each container has its own
  - Root filesystem
  - Processes
  - Memory
  - Devices
  - Network ports

Containers

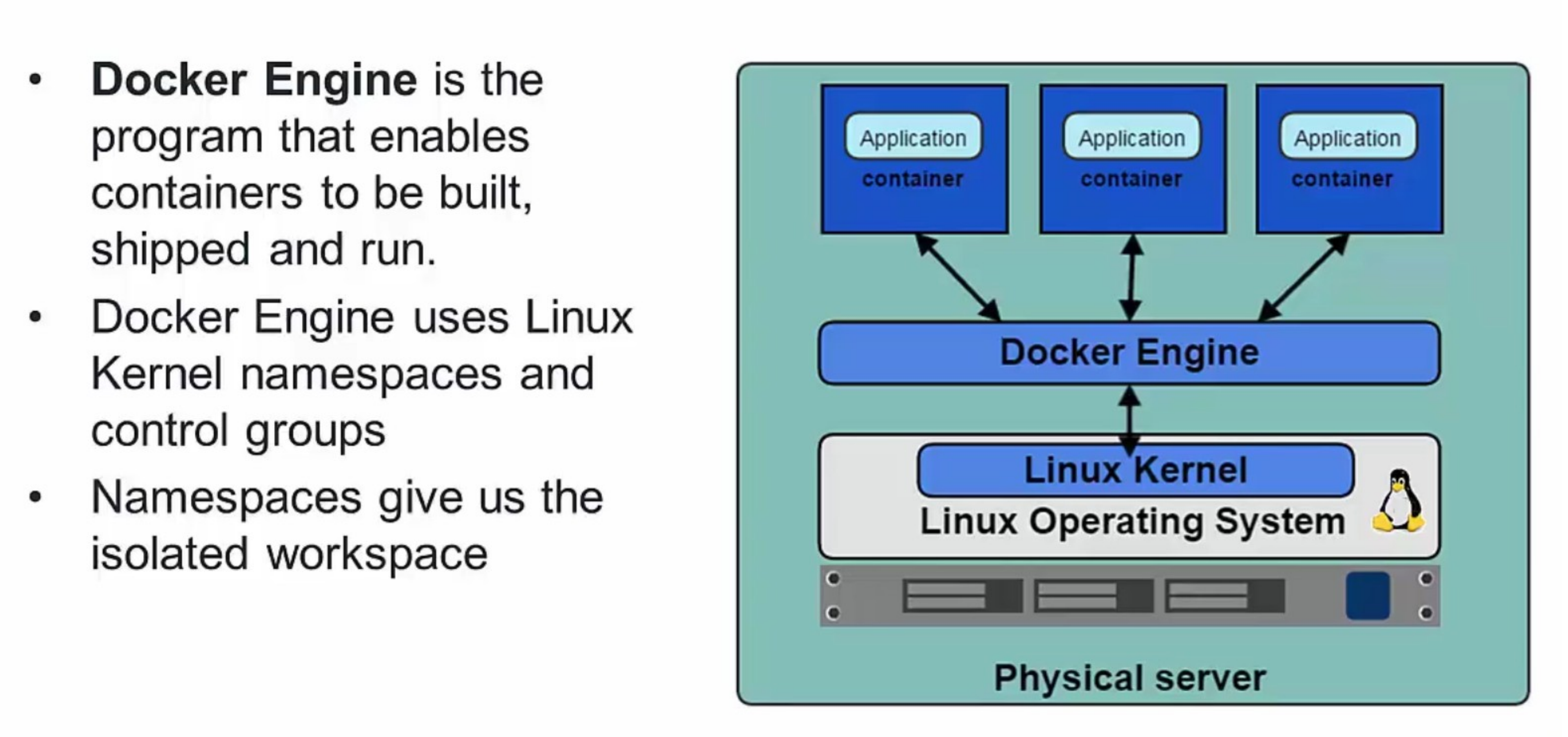

Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

Containers vs VMs

- Containers are more lighweight
- No need to install guest OS
- Less CPU, RAM, Storage space required
- More containers per machine than VM

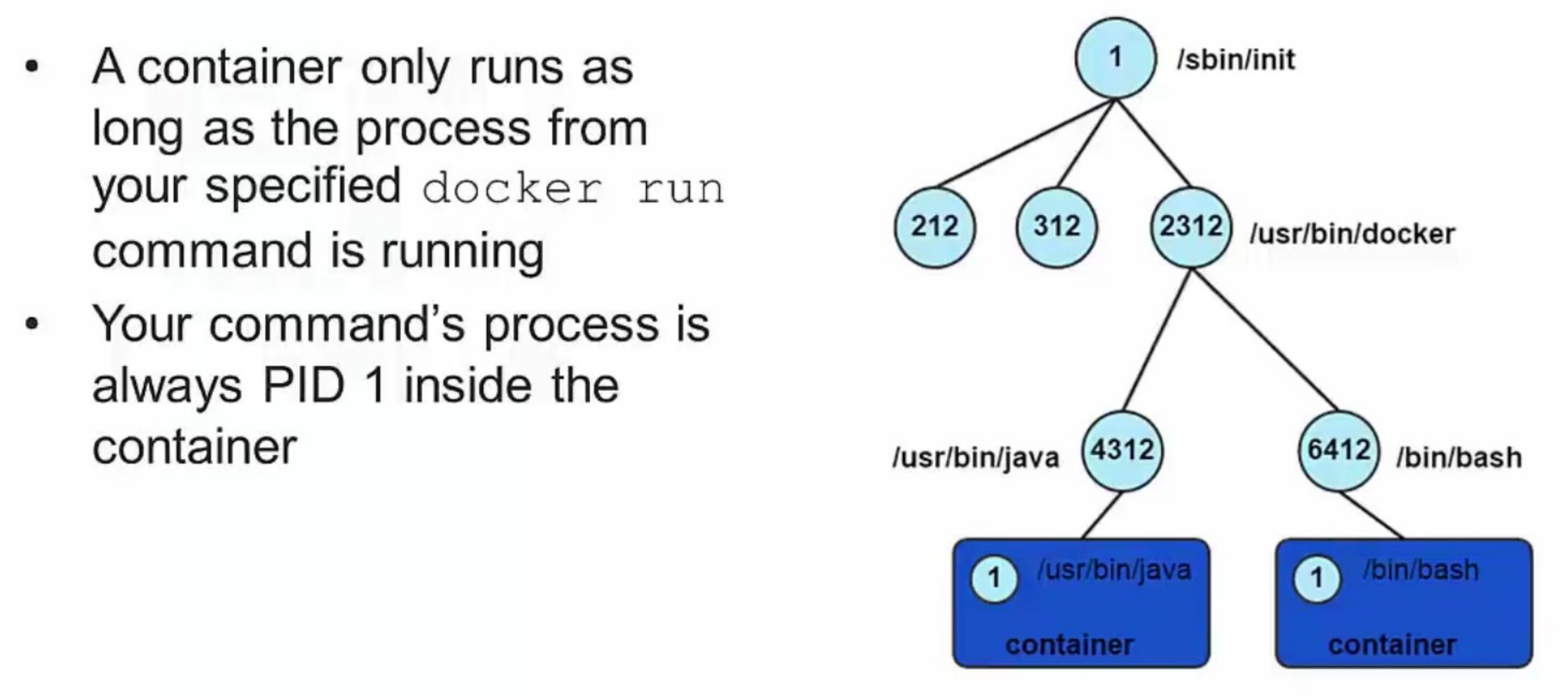
Docker Concepts and Terms

Docker and the Linux Kernel



- **Docker Engine** is the program that enables containers to be built, shipped and run.
- Docker Engine uses Linux Kernel namespaces and control groups
- Namespaces give us the isolated workspace

Container Processes

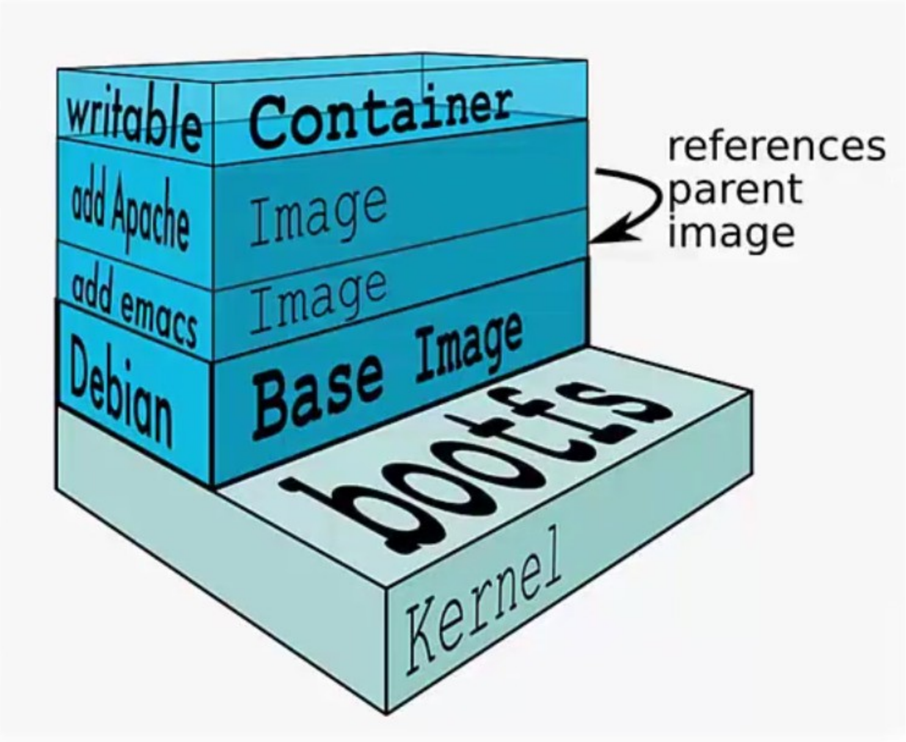


- A container only runs as long as the process from your specified `docker run` command is running
- Your command’s process is always PID 1 inside the container

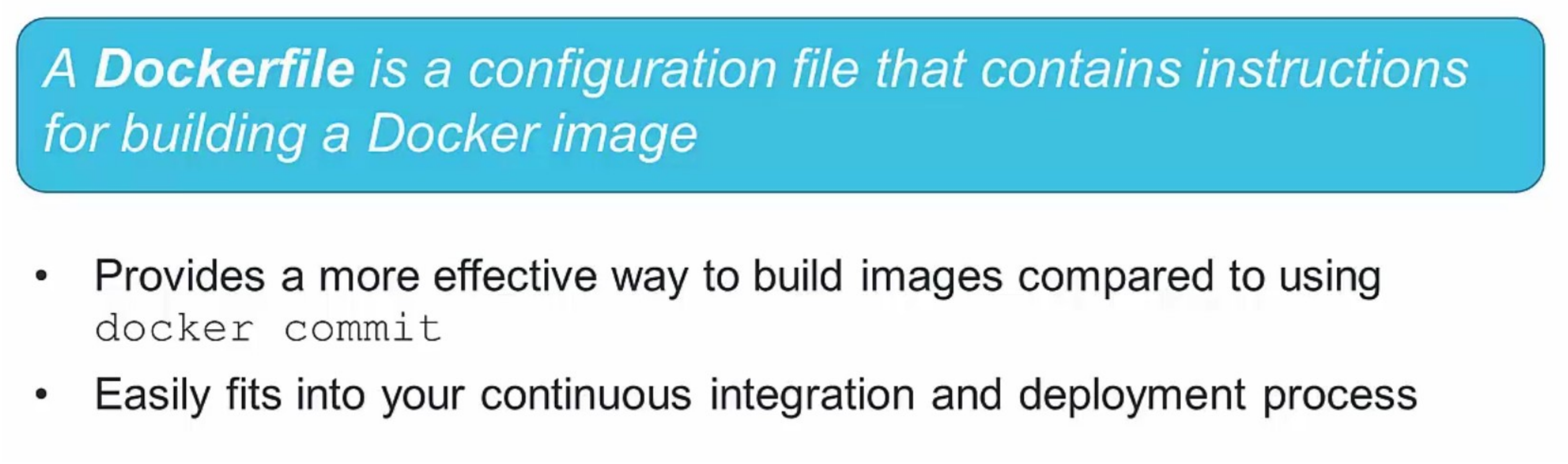
Images & Volumes

Images Layers

- Images are comprised of multiple layers
- A layer is also just another image
- Every image contains a base layer
- Docker uses a copy on write system
- Layers are read only



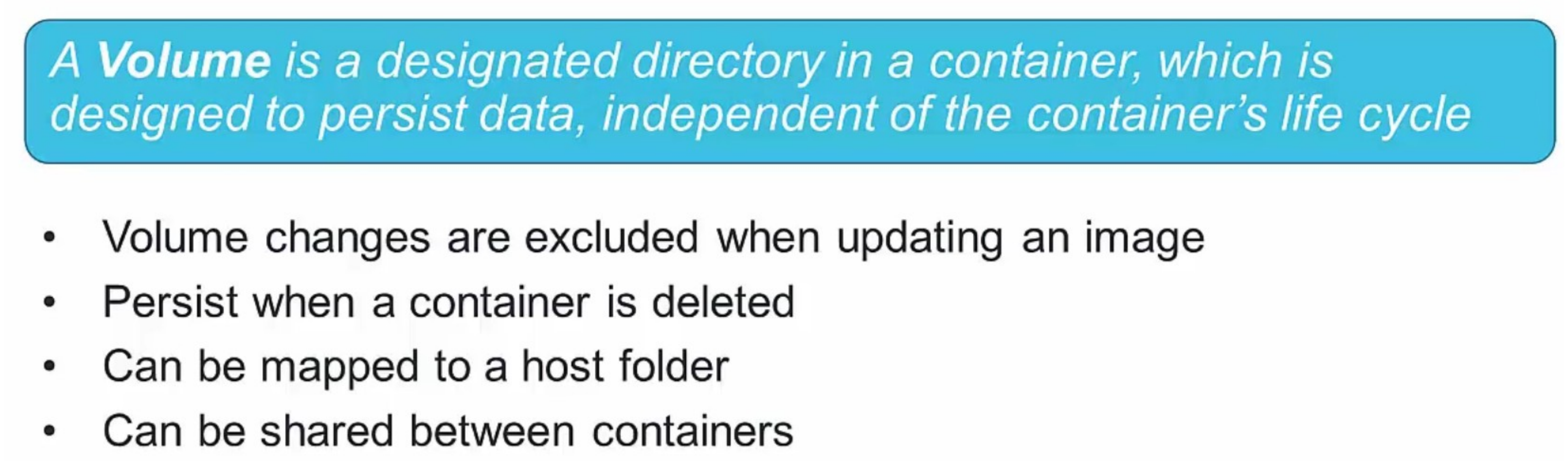
Intro to Dockerfile



*A **Dockerfile** is a configuration file that contains instructions for building a Docker image*

- Provides a more effective way to build images compared to using `docker commit`
- Easily fits into your continuous integration and deployment process

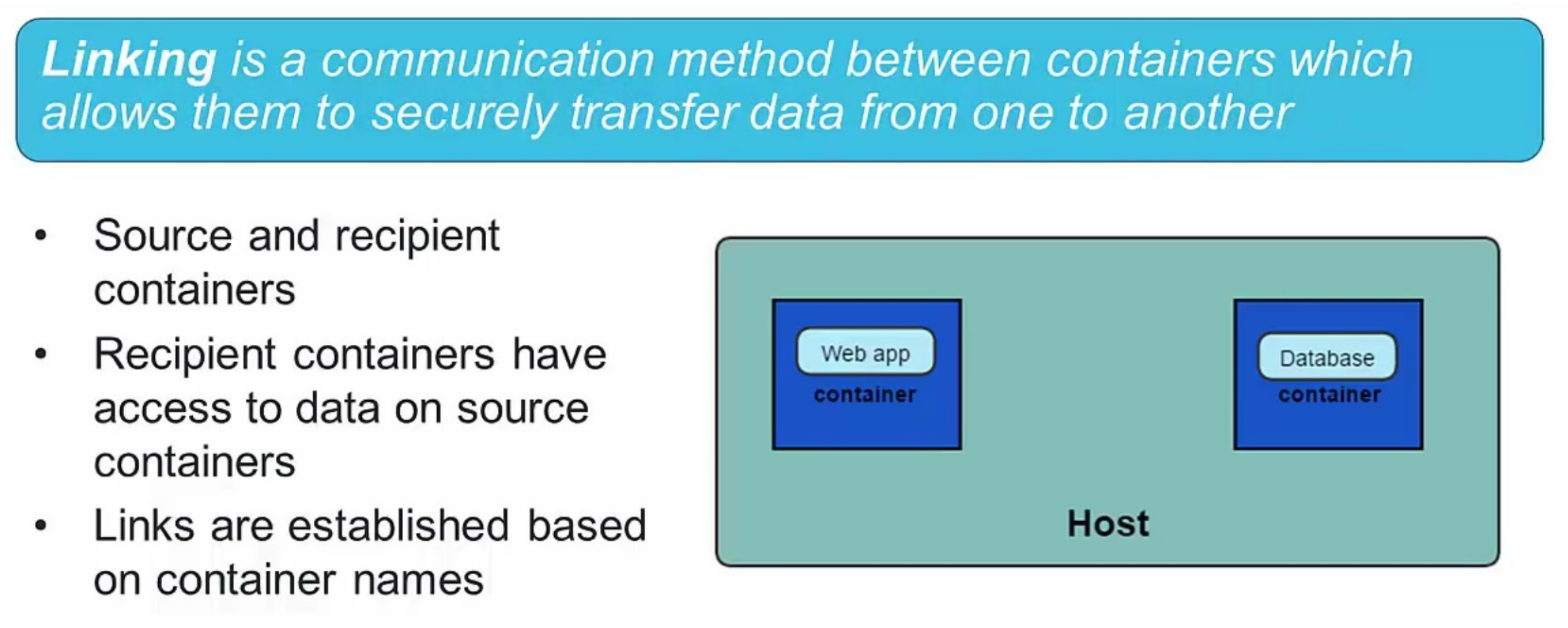
Volumes



*A **Volume** is a designated directory in a container, which is designed to persist data, independent of the container's life cycle*

- Volume changes are excluded when updating an image
- Persist when a container is deleted
- Can be mapped to a host folder
- Can be shared between containers

Linking Containers

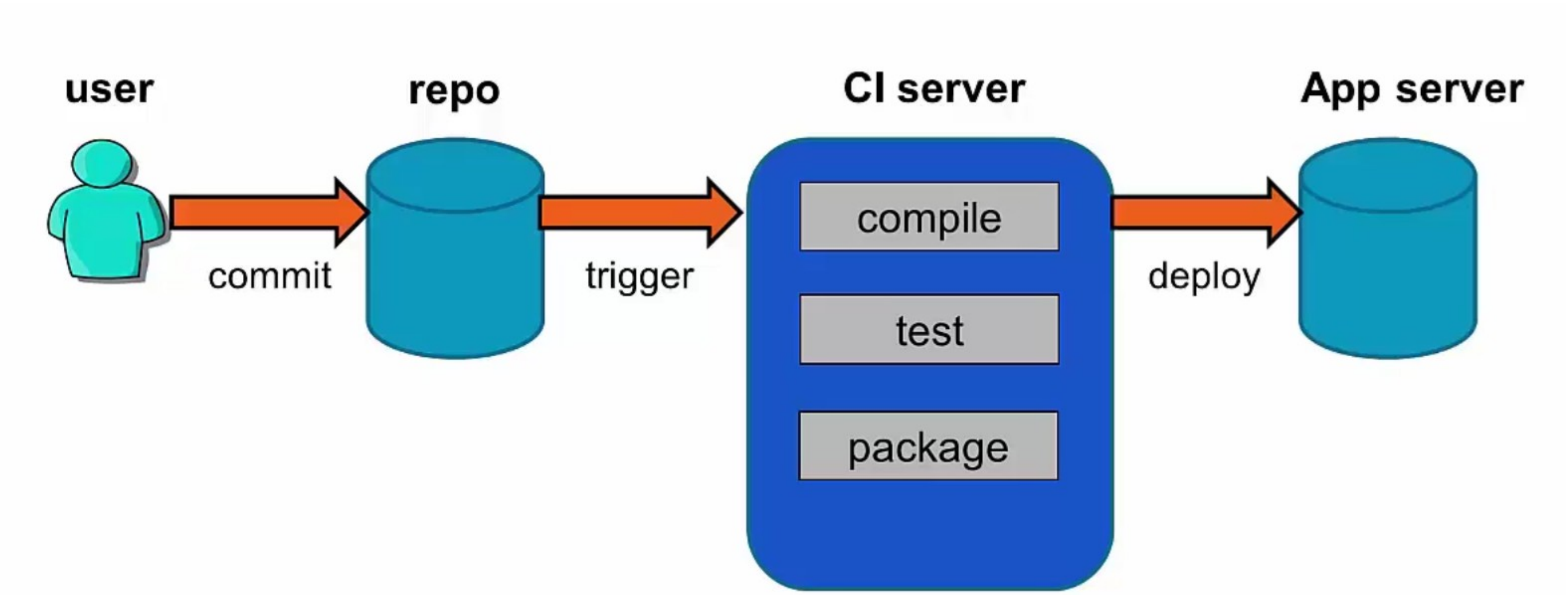


***Linking** is a communication method between containers which allows them to securely transfer data from one to another*

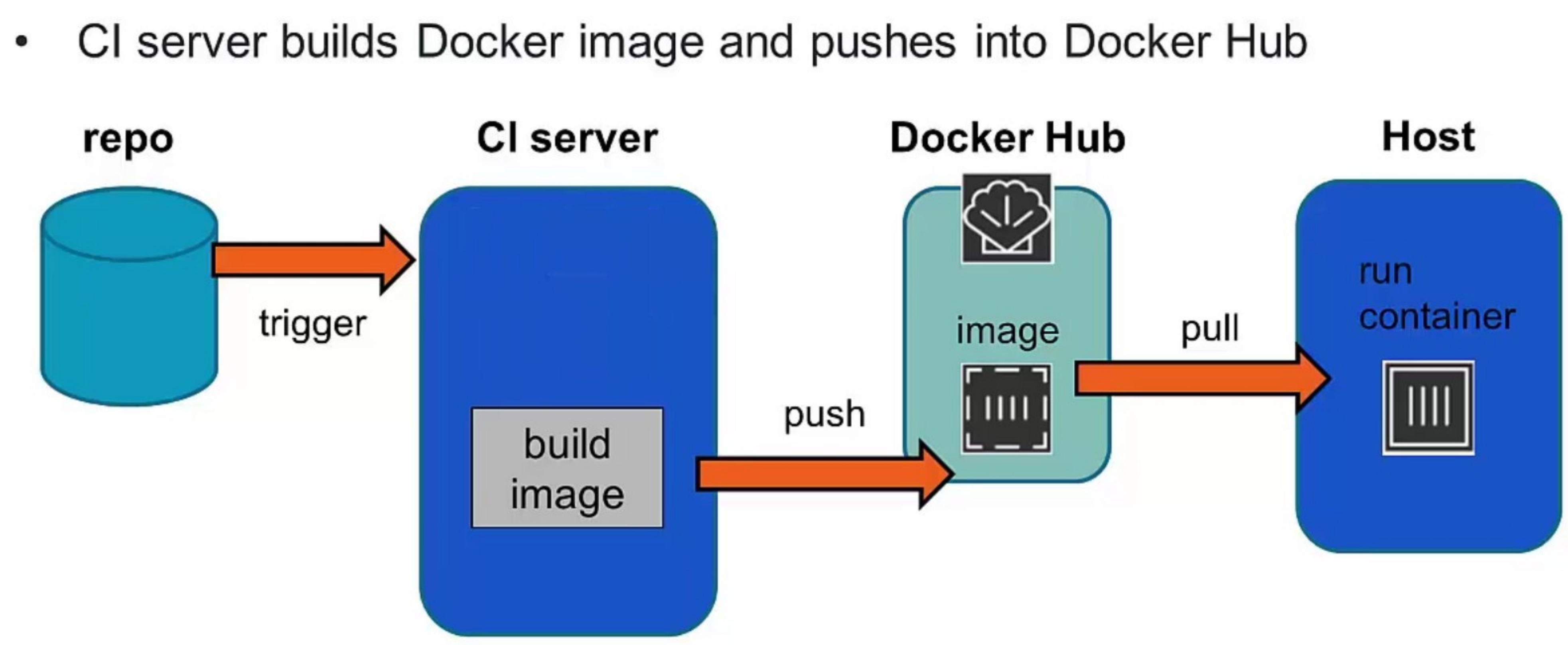
- Source and recipient containers
- Recipient containers have access to data on source containers
- Links are established based on container names

Docker Continous Integration

Traditional Continous Integration

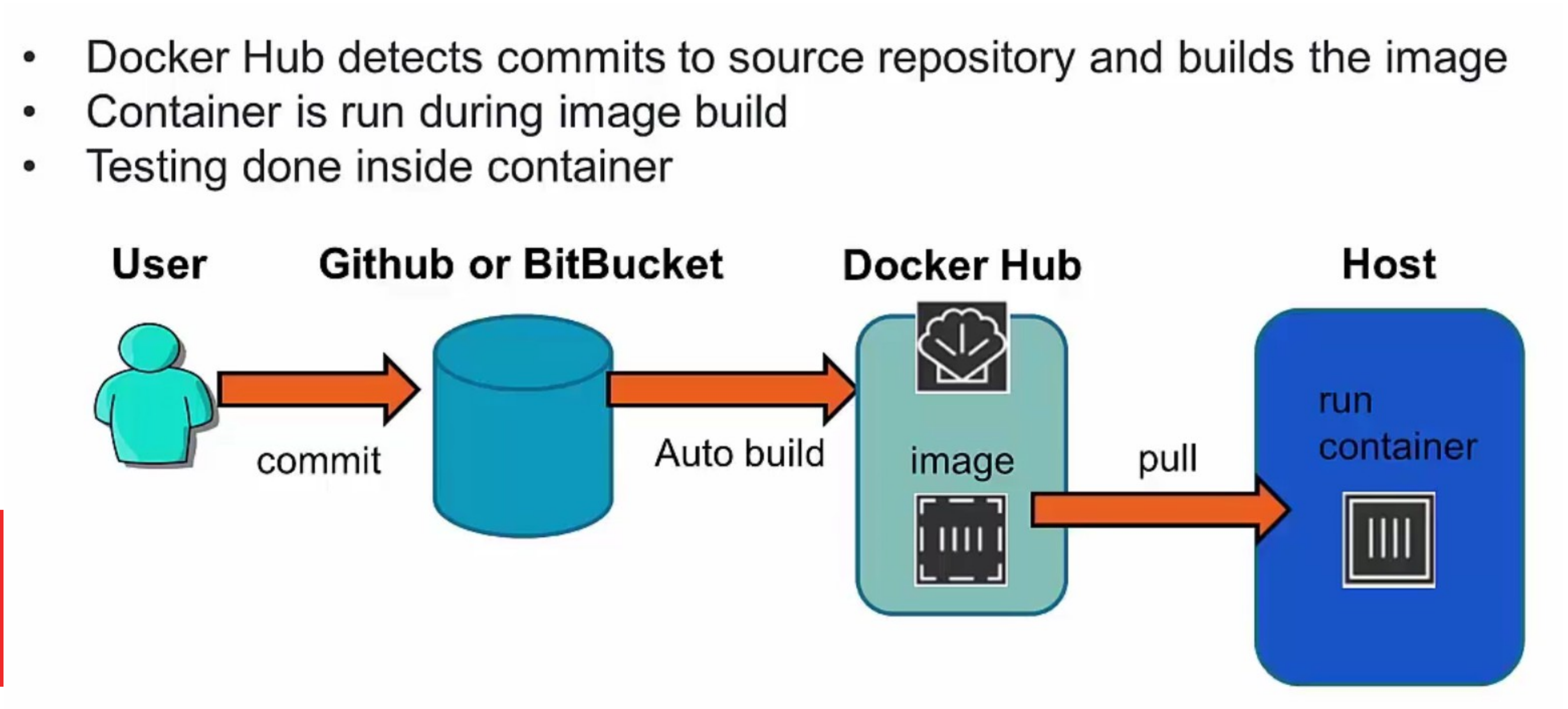


Using Docker in CI



- CI server builds Docker image and pushes into Docker Hub

Docker Hub Auto Build

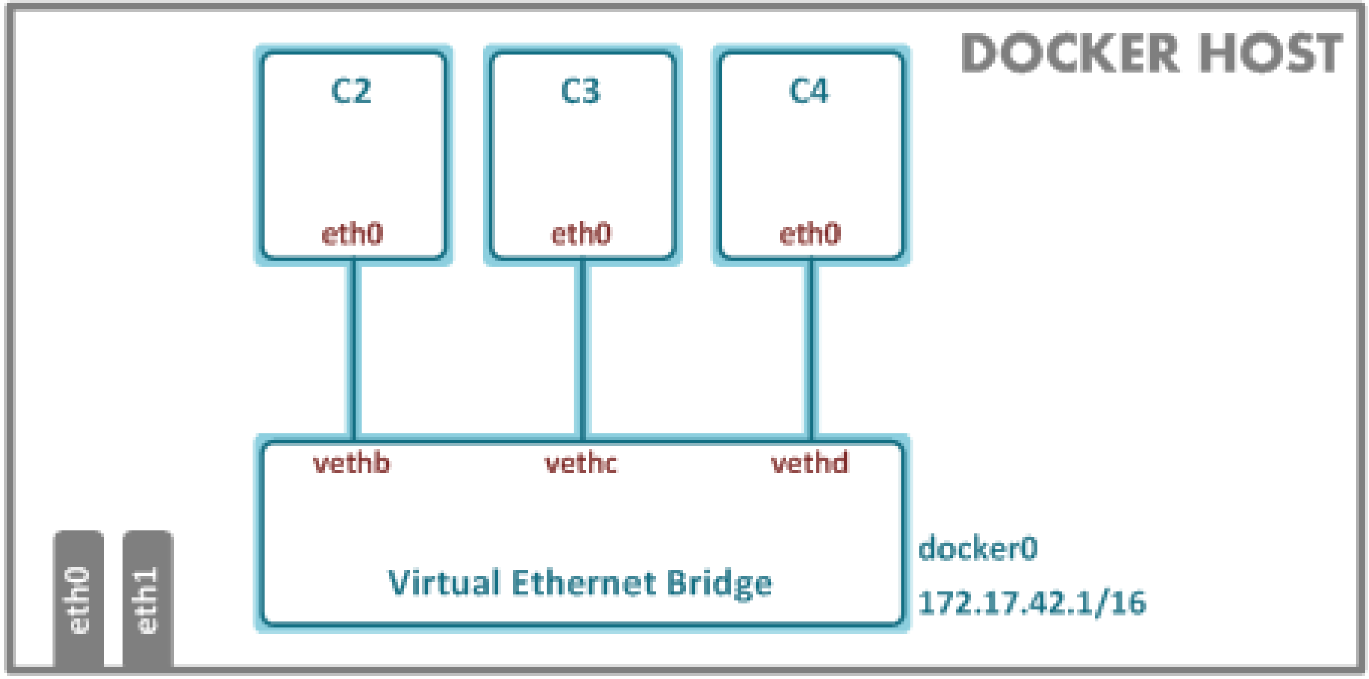


- Docker Hub detects commits to source repository and builds the image
- Container is run during image build
- Testing done inside container

Lab I

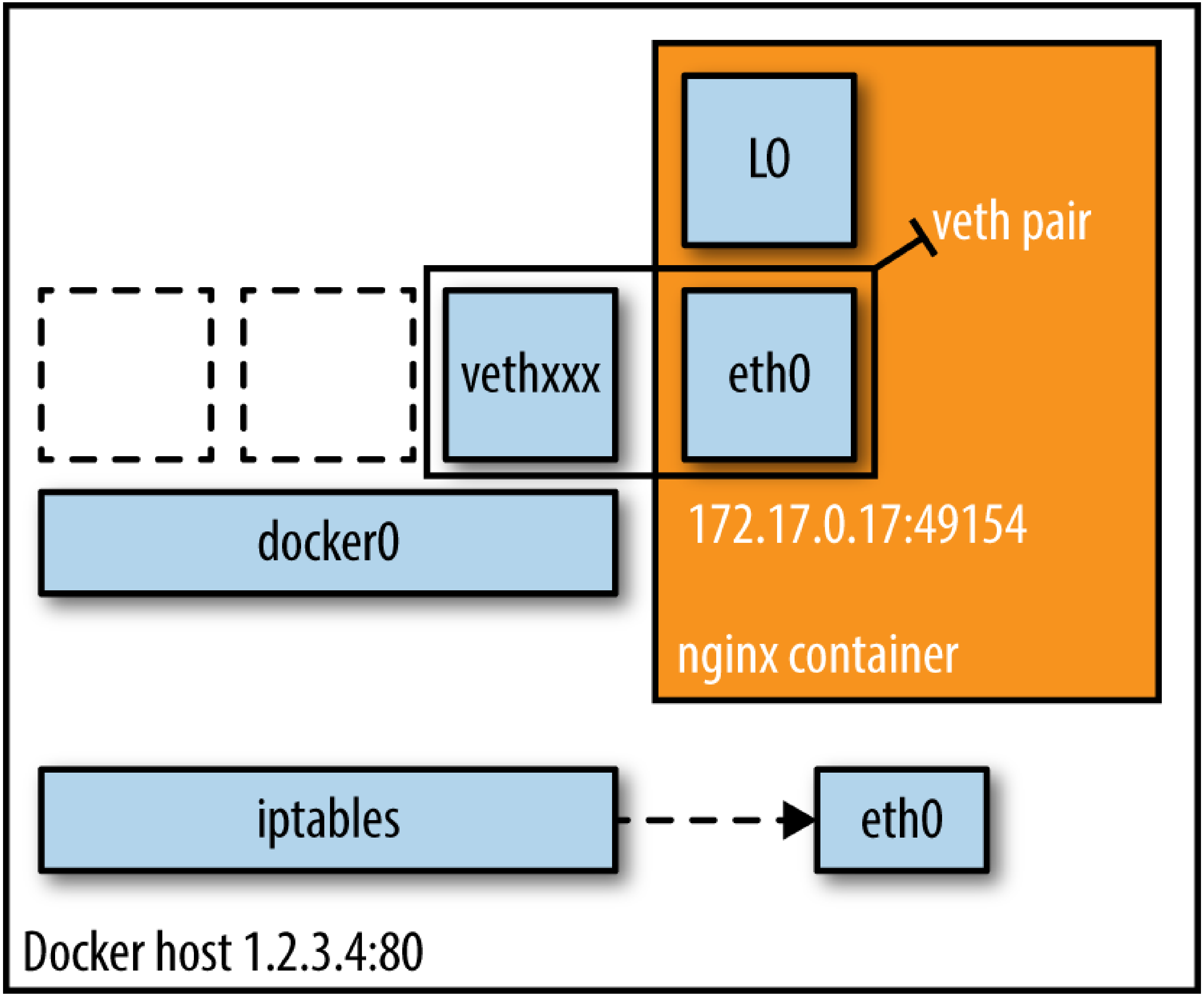
Docker Networking Basic

Container Networking

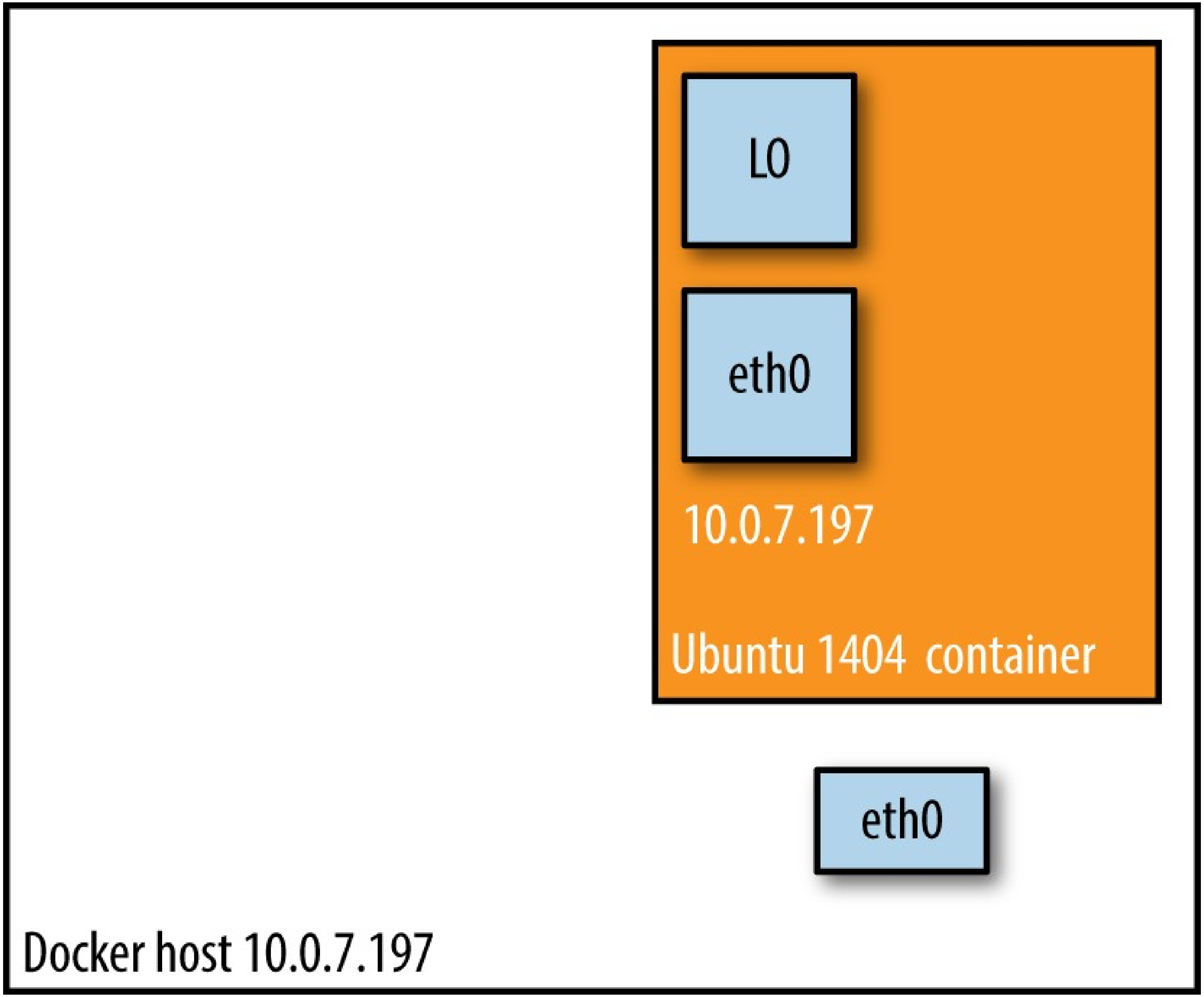


- docker0 – a Virtual Ethernet Bridge
- docker0 is present in the namespace of Docker host
- Subnet shared by docker0 interface and Container interface

Bridge Mode Networking

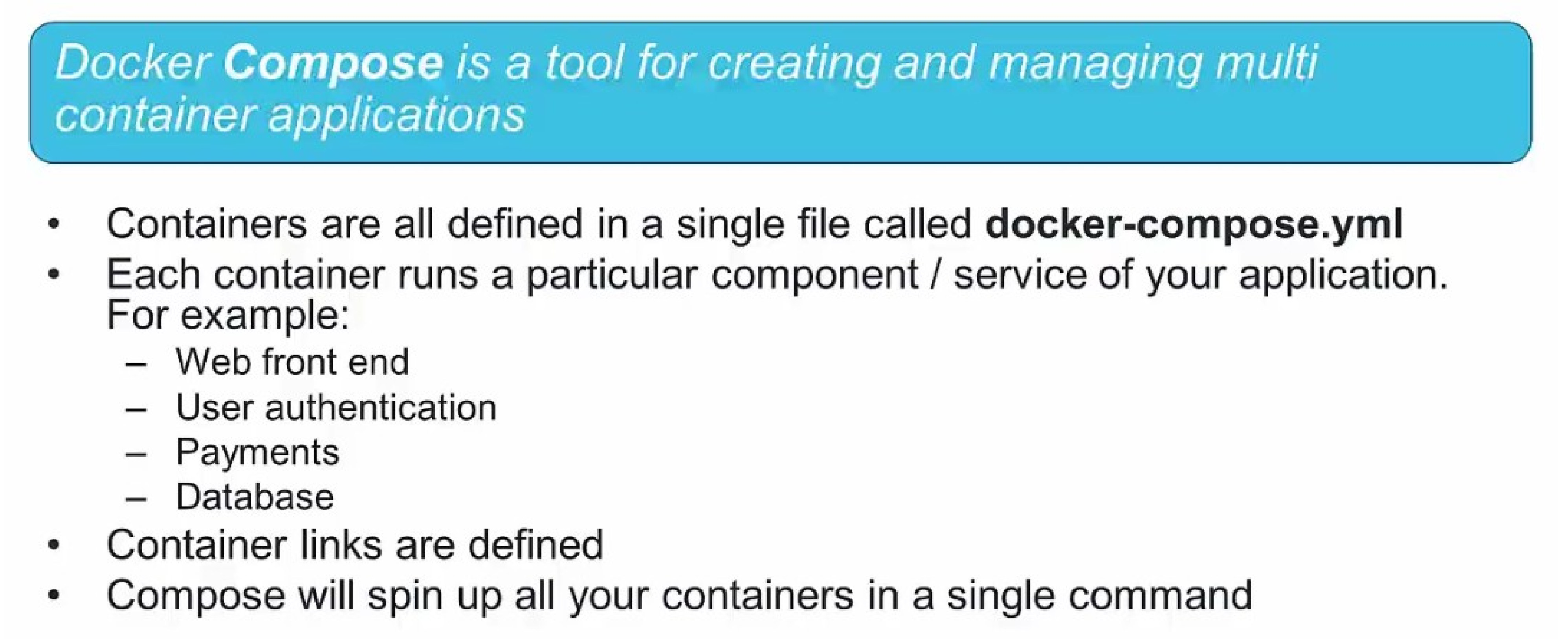


Host Mode Networking



Docker Compose

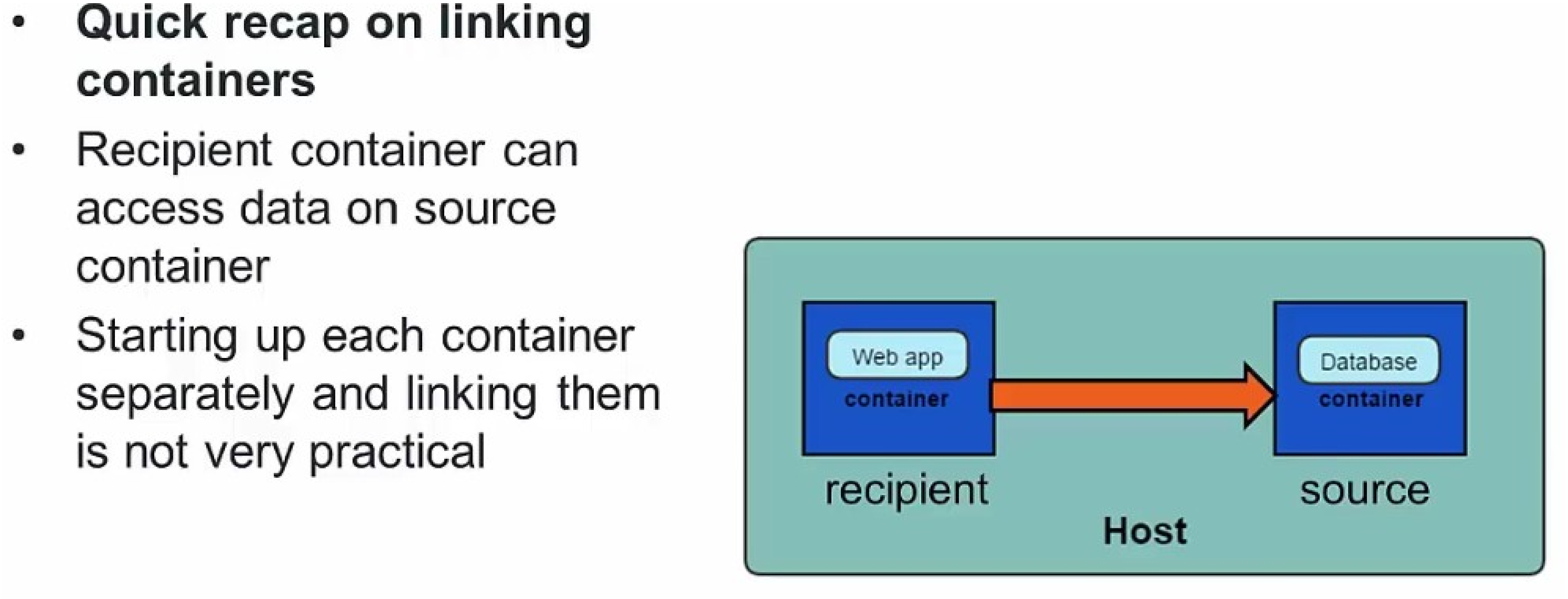
What is Compose ?



*Docker **Compose** is a tool for creating and managing multi container applications*

- Containers are all defined in a single file called **docker-compose.yml**
- Each container runs a particular component / service of your application. For example:
  - Web front end
  - User authentication
  - Payments
  - Database
- Container links are defined
- Compose will spin up all your containers in a single command

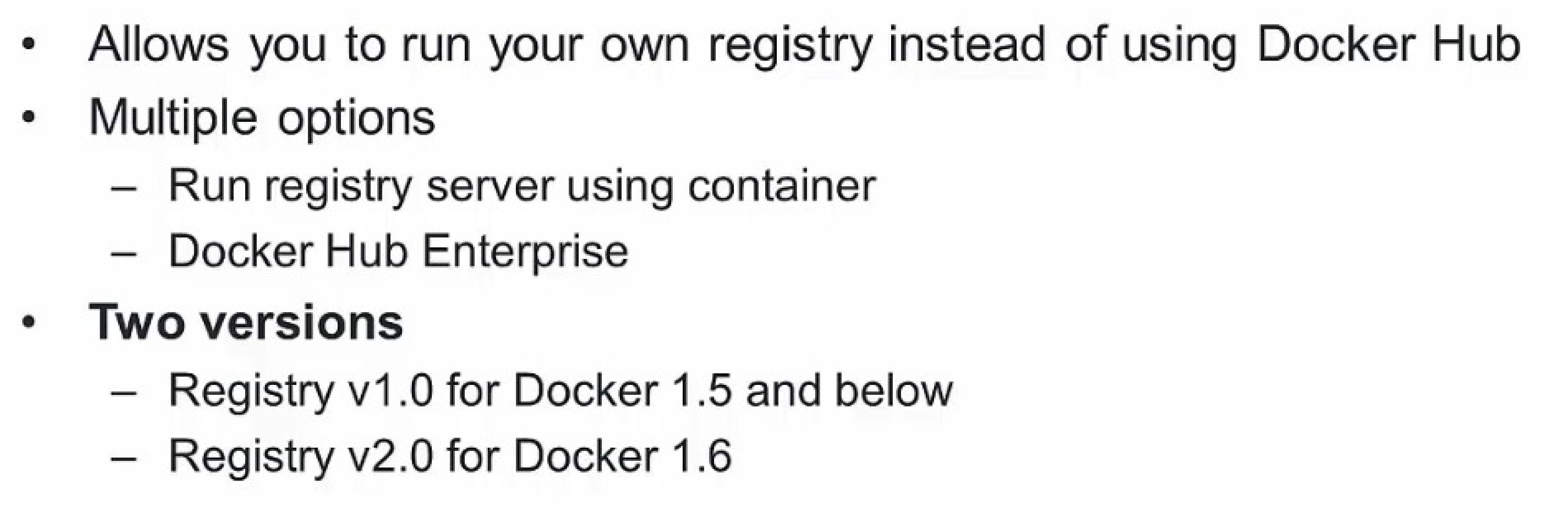
Benefit of Compose



- **Quick recap on linking containers**
- Recipient container can access data on source container
- Starting up each container separately and linking them is not very practical

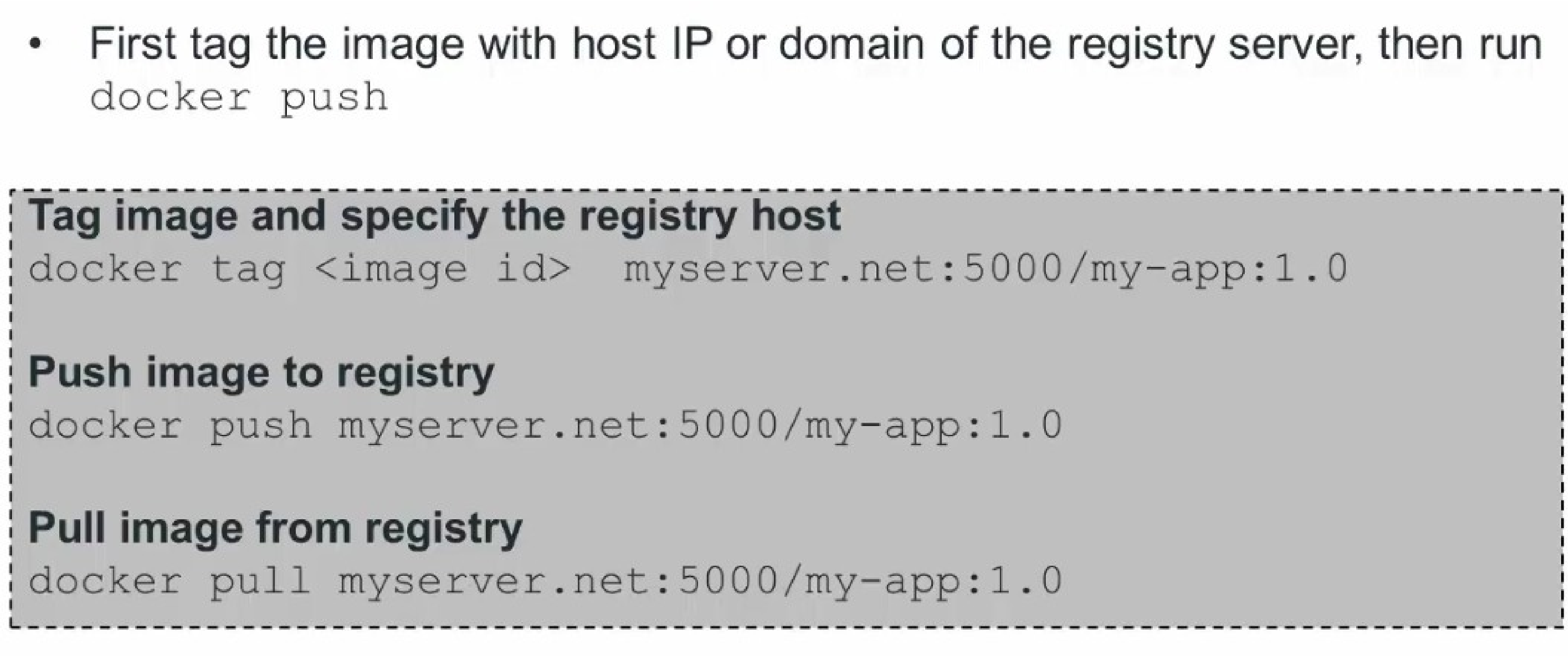
Private Registry

Docker Private Registry



- Allows you to run your own registry instead of using Docker Hub
- Multiple options
  - Run registry server using container
  - Docker Hub Enterprise
- **Two versions**
  - Registry v1.0 for Docker 1.5 and below
  - Registry v2.0 for Docker 1.6

Push and Pull from Private Registry



- First tag the image with host IP or domain of the registry server, then run `docker push`

Tag image and specify the registry host

```
docker tag <image id>  myserver.net:5000/my-app:1.0
```

Push image to registry

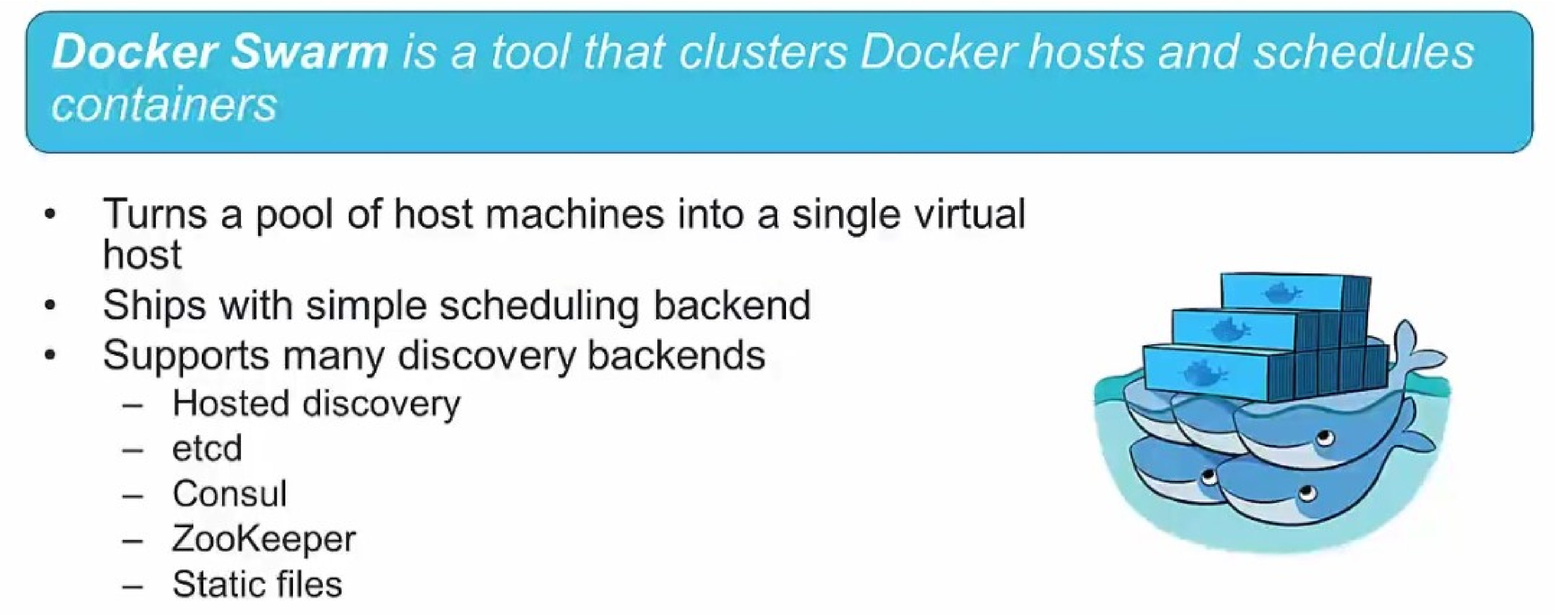
```
docker push myserver.net:5000/my-app:1.0
```

Pull image from registry

```
docker pull myserver.net:5000/my-app:1.0
```

Docker Swarm

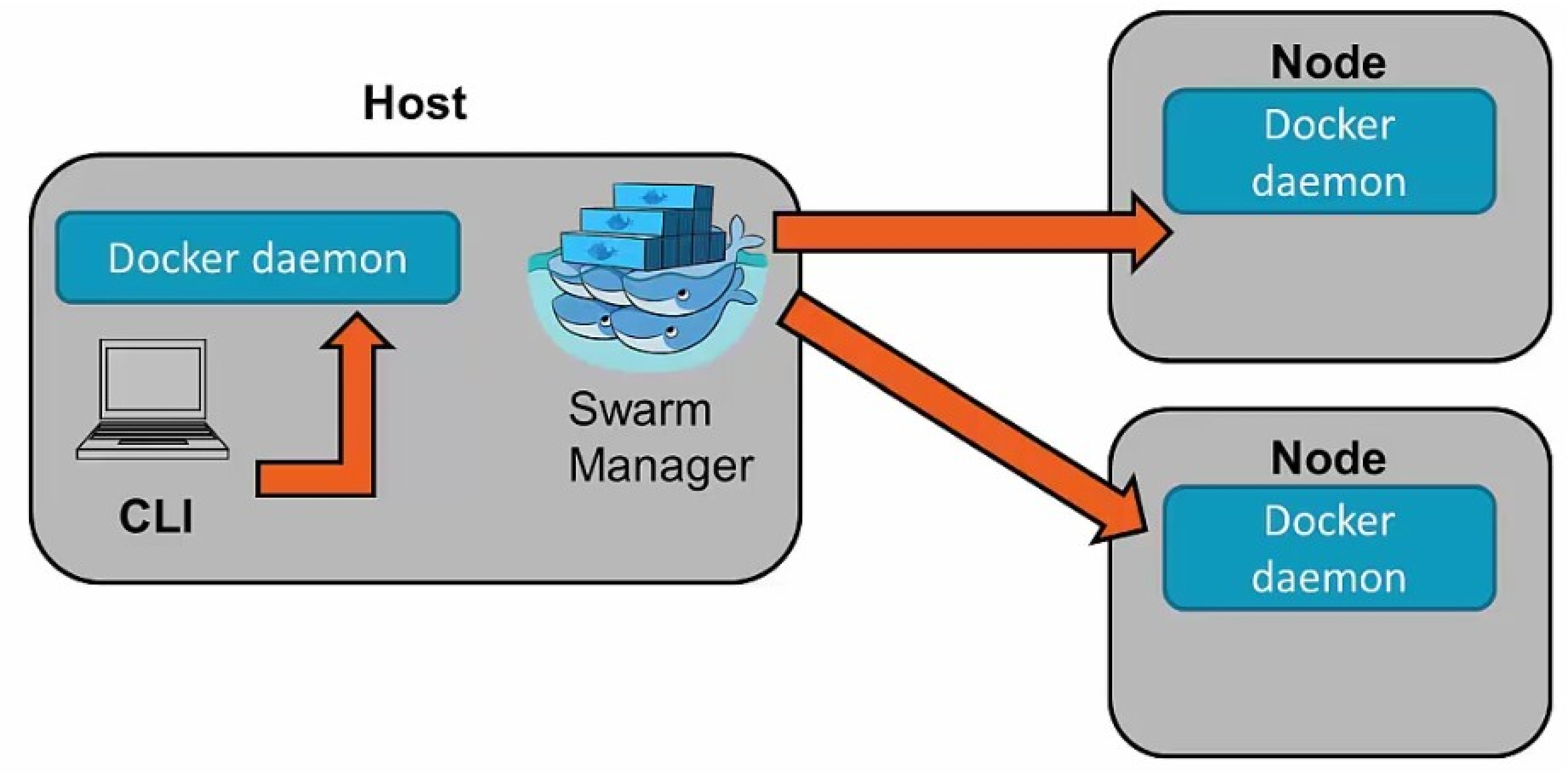
Docker Swarm



***Docker Swarm** is a tool that clusters Docker hosts and schedules containers*

- Turns a pool of host machines into a single virtual host
- Ships with simple scheduling backend
- Supports many discovery backends
  - Hosted discovery
  - etcd
  - Consul
  - ZooKeeper
  - Static files

How Swarm work



Manage nodes in a swarm

- **No value** indicates a worker node that does not participate in swarm management.
- **Leader** means the node is the primary manager node that makes all swarm management and orchestration decisions for the swarm.
- **Reachable** means the node is a manager node participating in the Raft consensus quorum. If the leader node becomes unavailable, the node is eligible for election as the new leader.
- **Unavailable** means the node is a manager that is not able to communicate with other managers. If a manager node becomes unavailable, you should either join a new manager node to the swarm or promote a worker node to be a manager.

Lab II

www.btech.id