Docker & Containers Introduction

Everybody Loves Containers

A **standard way to package an application and all its dependencies** so that it can be moved between environments and run without changes

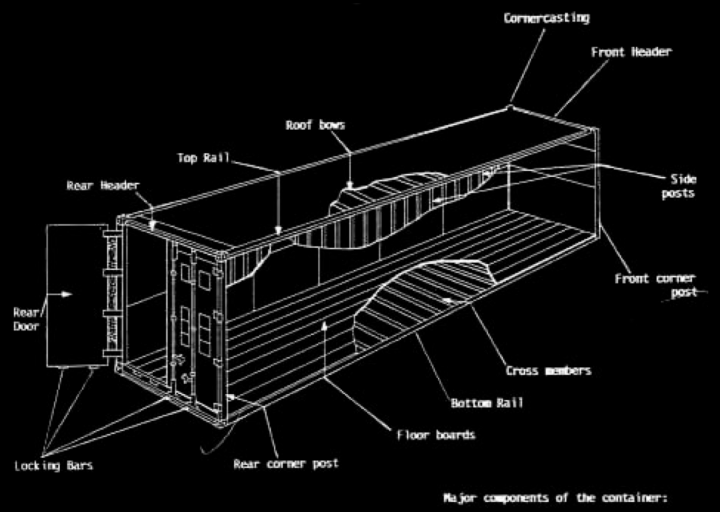
Containers work by **isolating the differences between applications** inside the container so that everything outside the container can be standardized

Microservices implementation with Containers

Why it works – separation of concerns

Development

- Worries about what's "**inside**" the container
  - Code
  - Libraries
  - Package Manager
  - Apps
  - Data
- All Linux servers look the same



Major components of the container:

Operations

- Worries about what's "**outside**" the container
  - Logging
  - Remote Access
  - Monitoring
  - Network Config
- All containers start, stop, copy, attach, migrate, etc… the same way

Clear ownership boundary between
Dev and IT Ops drives DevOps adoption and fosters agility

Container History

1979

chroot

A chroot on Unix operating systems is an operation that changes the apparent root directory for the current running process and its children. A program that is run in such a modified environment cannot name (and therefore normally cannot access) files outside the designated directory tree. … The modified environment is called a "chroot jail".

https://en.wikipedia.org/wiki/ChrootChr

2007

cgroups

2008

LXC

LXC (Linux Containers) is an operating-system-level virtualization environment for running multiple isolated Linux systems (containers) on a single Linux control host.

…

LXC combines kernel's **cgroups** and support for **isolated namespaces** to provide an isolated environment for applications.

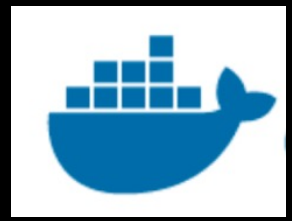
https://en.wikipedia.org/wiki/LXC

2011

warden

2012

Docker



2014

rkt

2016

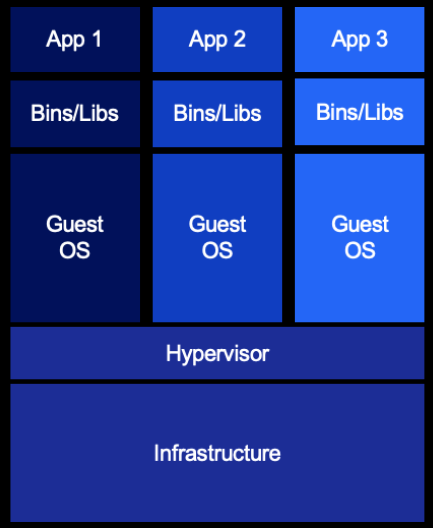
Garden runC

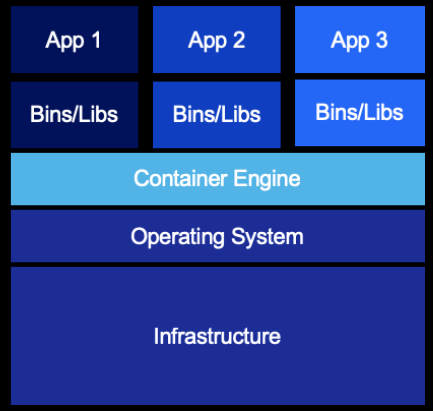
2018

containerd

https://containerd.io/

VMs vs. Containers

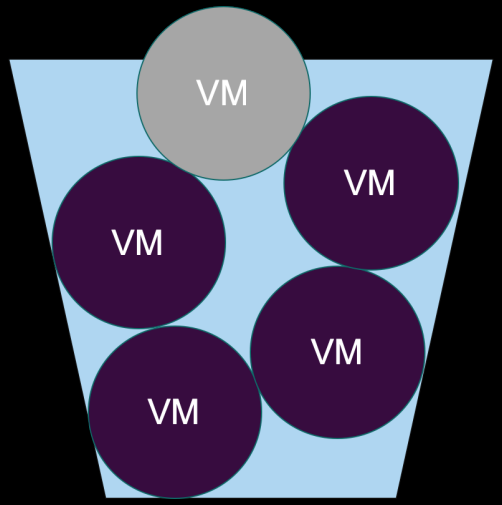


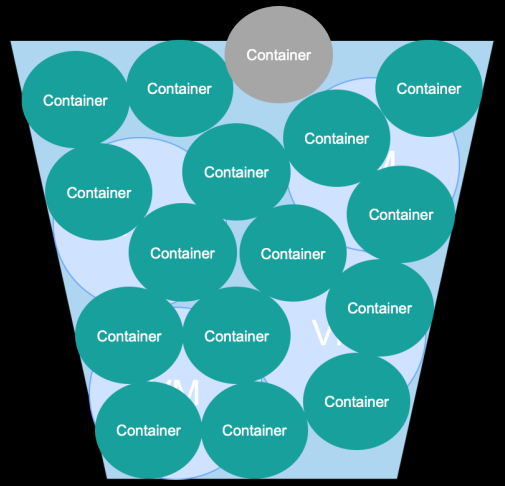


+ VM Isolation
- Complete OS
- Static Compute
- Static Memory

+ Container Isolation
+ Shared Kernel
+ Burstable Compute
+ Burstable Memory

VMs vs. Containers





+ VM Isolation
- Complete OS
- Static Compute
- Static Memory
- Low Resource Utilization

+ Container Isolation
+ Shared Kernel
+ Burstable Compute
+ Burstable Memory
+ High Resource Utilization

Advantages of Containers

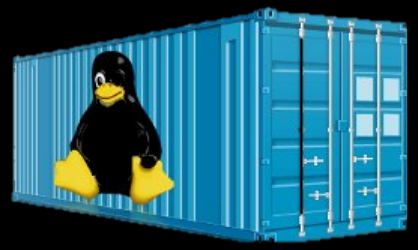
Containers are **portable**

Containers are **easy to manage**

Containers provide **“just enough” isolation**

Containers use hardware **more efficiently**

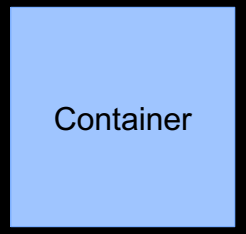
Containers are **immutable**



Docker Components

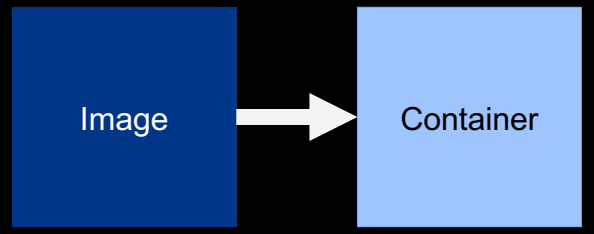
Container

Smallest compute unit



Docker Components

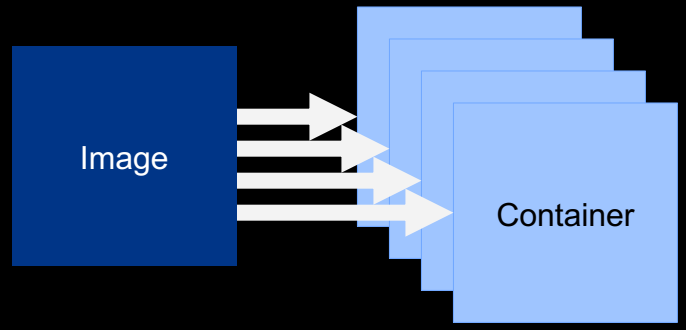
Containers
are created from
Images



one process per container

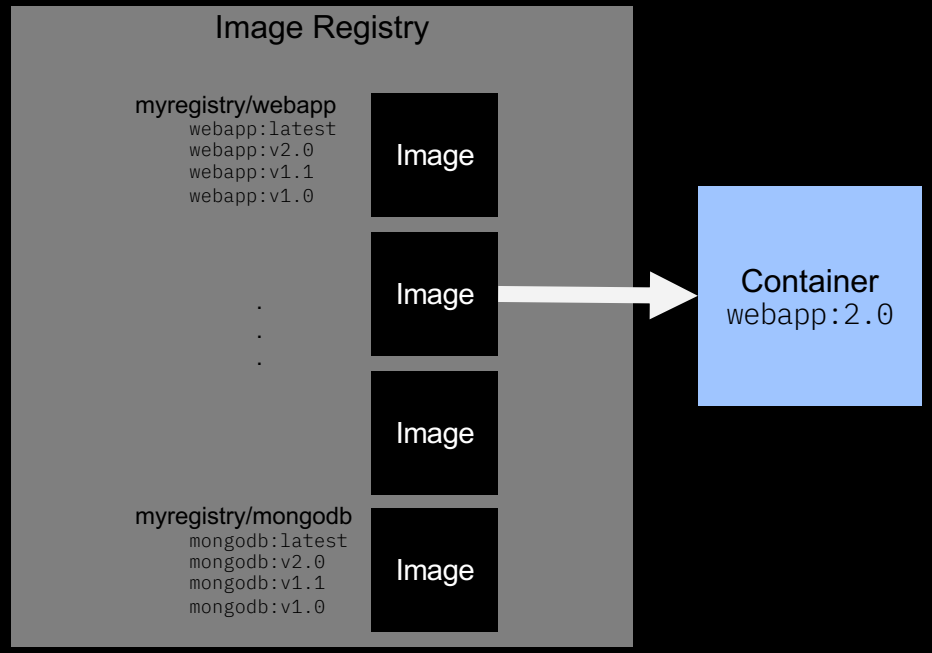
Docker Components

As **many Containers**
as needed can be created from
Images



Docker Components

The **Image Registry**
stores the versioned
Images
to create
Containers



Docker Components

Image

A read-only snapshot of a container stored in Docker Hub to be used as a template for building containers

Container

The standard unit in which the application service resides or transported

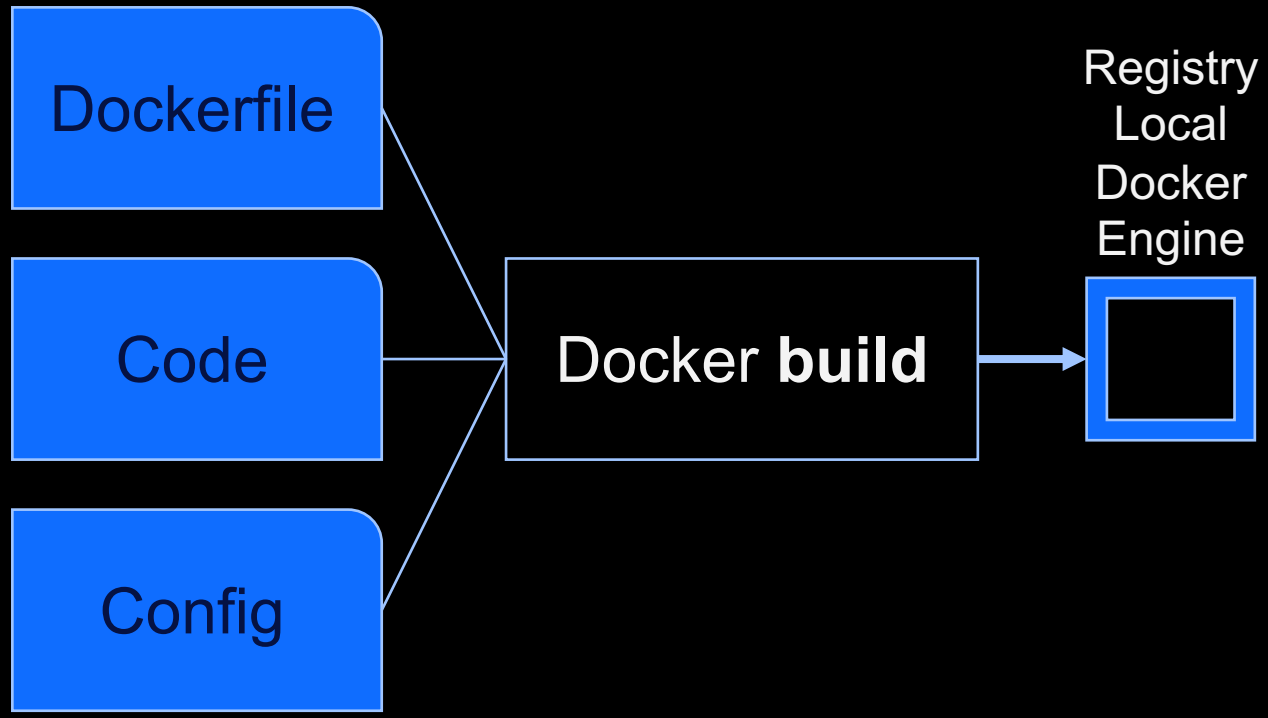
Docker Hub/Registry

Available in SaaS or Enterprise to deploy anywhere you choose
Stores, distributes, and shares container images

Docker Engine

A program that creates, ships, and runs application containers
Runs on any physical and virtual machine or server locally, in private or public cloud. Client communicates with Engine to execute commands

Docker Basics – Build

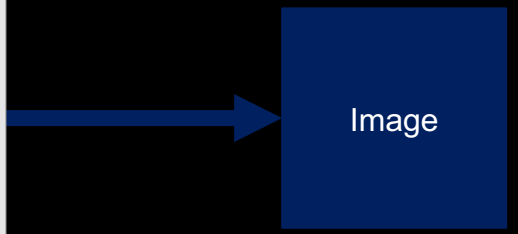


Docker Basics – Build - Dockerfile

Dockerfile

- Text based file describing:
  - Previous Layer
  - Environment Variables
  - Commands used to populate data/software/frameworks/etc...
  - Command to run when executed

```
# Pull base image
FROM tomcat:8-jre8

# Maintainer
MAINTAINER "youremailaddress"

# Run command
RUN apt-get update && apt-get -y upgrade

# Set variables
ENV myName John Doe

# Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/

# Run server
CMD ["catalina.sh", "run"]
```



Docker Basics – Build - Dockerfile

- A text file that builds an image using Docker directives

- FROM
- RUN
- COPY
- ENTRYPOINT
- LABEL
- ENV
- ARG
- USER
- EXPOSE

```
# Pull base image
FROM tomcat:8-jre8

# Maintainer
MAINTAINER "youremailaddress"

# Run command
RUN apt-get update && apt-get -y upgrade

# Set variables
ENV myName John Doe

# Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/

# Expose the port of the web application
EXPOSE 80

# Run server
CMD ["catalina.sh", "run"]
```

```
docker build –t myimage ./Dockerfile
```

Base Image from:
- hub.docker.com
- quay.io
- your own

Run commands
apt-get, yum, chown, mv, cp, ...

Set environment variables
Can be overriden when running

Add assets to the image
Jar, sources, golang app, ...

Expose a port of the process running in the container

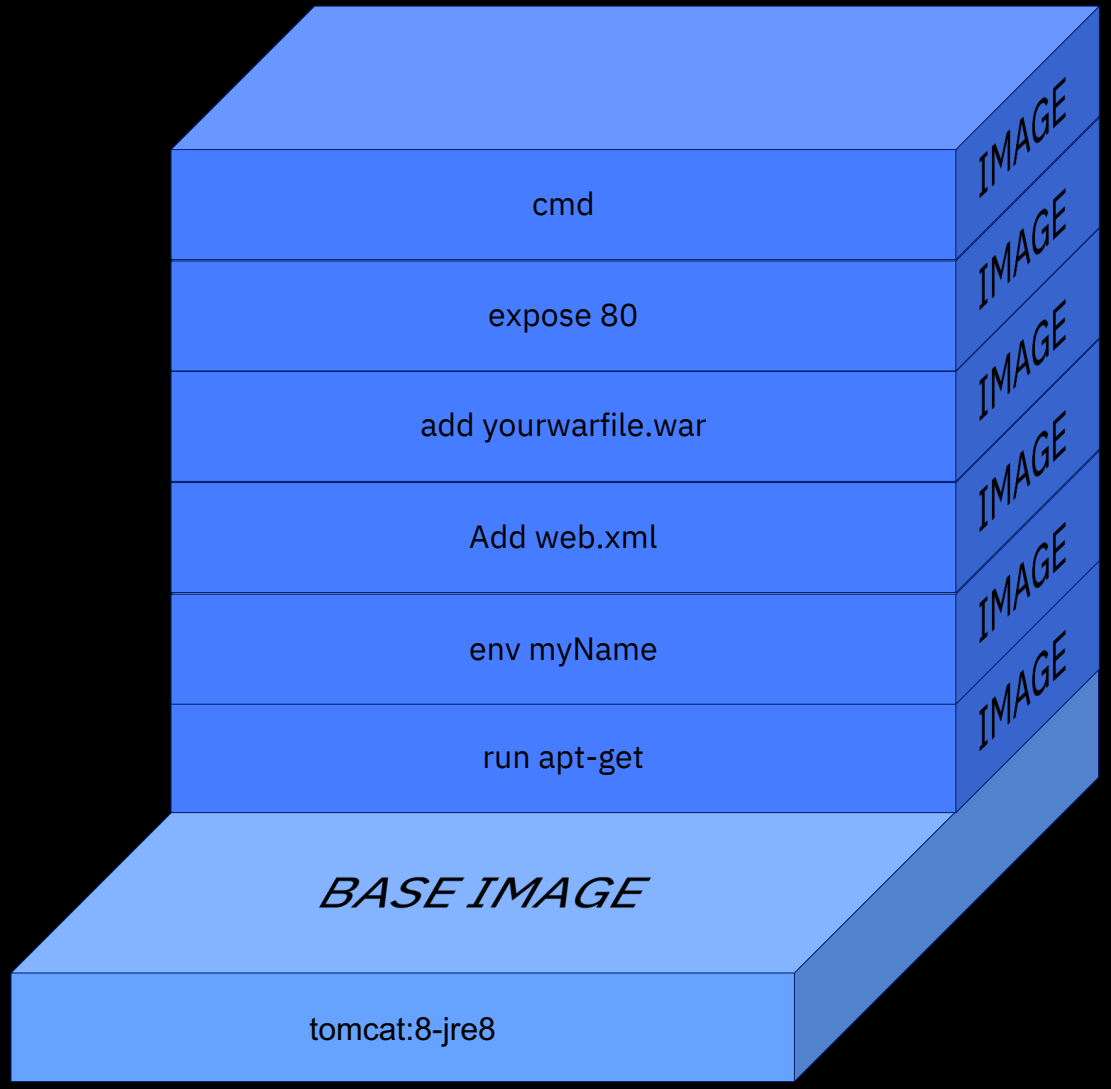
Startup command when image is being run

Docker Basics – Layered File System

Docker Layers
Inheritance

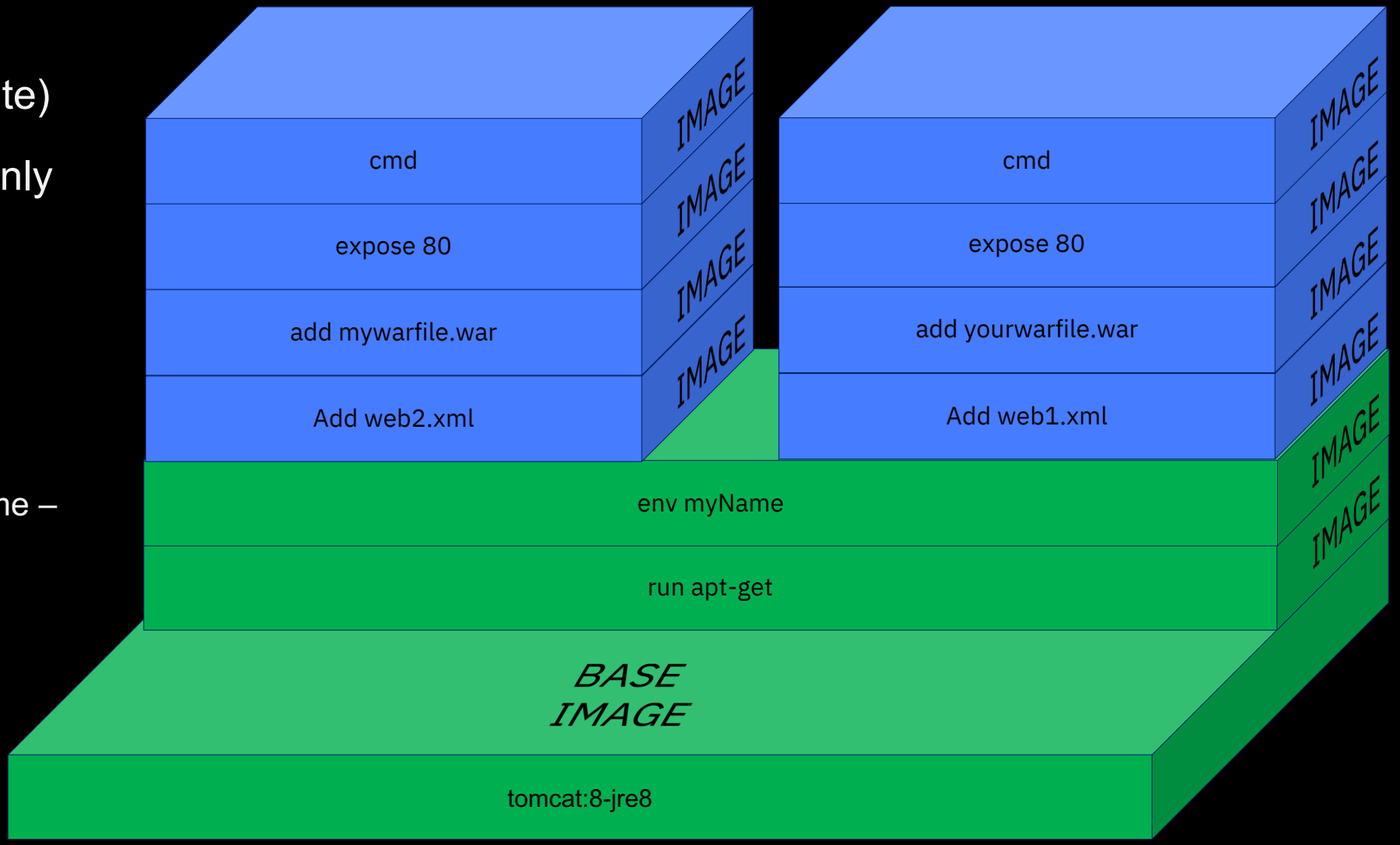
- Build on the work of those who came before you

```
# Pull base image
FROM tomcat:8-jre8

# Maintainer
MAINTAINER  youremailaddress"

# Run command
RUN apt-get update && apt-get -y upgrade

# Set variables
ENV myName John Doe

# Copy to images tomcat path
ADD web.xml /usr/local/tomcat/conf/
ADD yourwarfile.war /usr/local/tomcat/webapps/

# Expose the port of the web application
EXPOSE 80

# Run server
CMD ["catalina.sh", "run"]
```

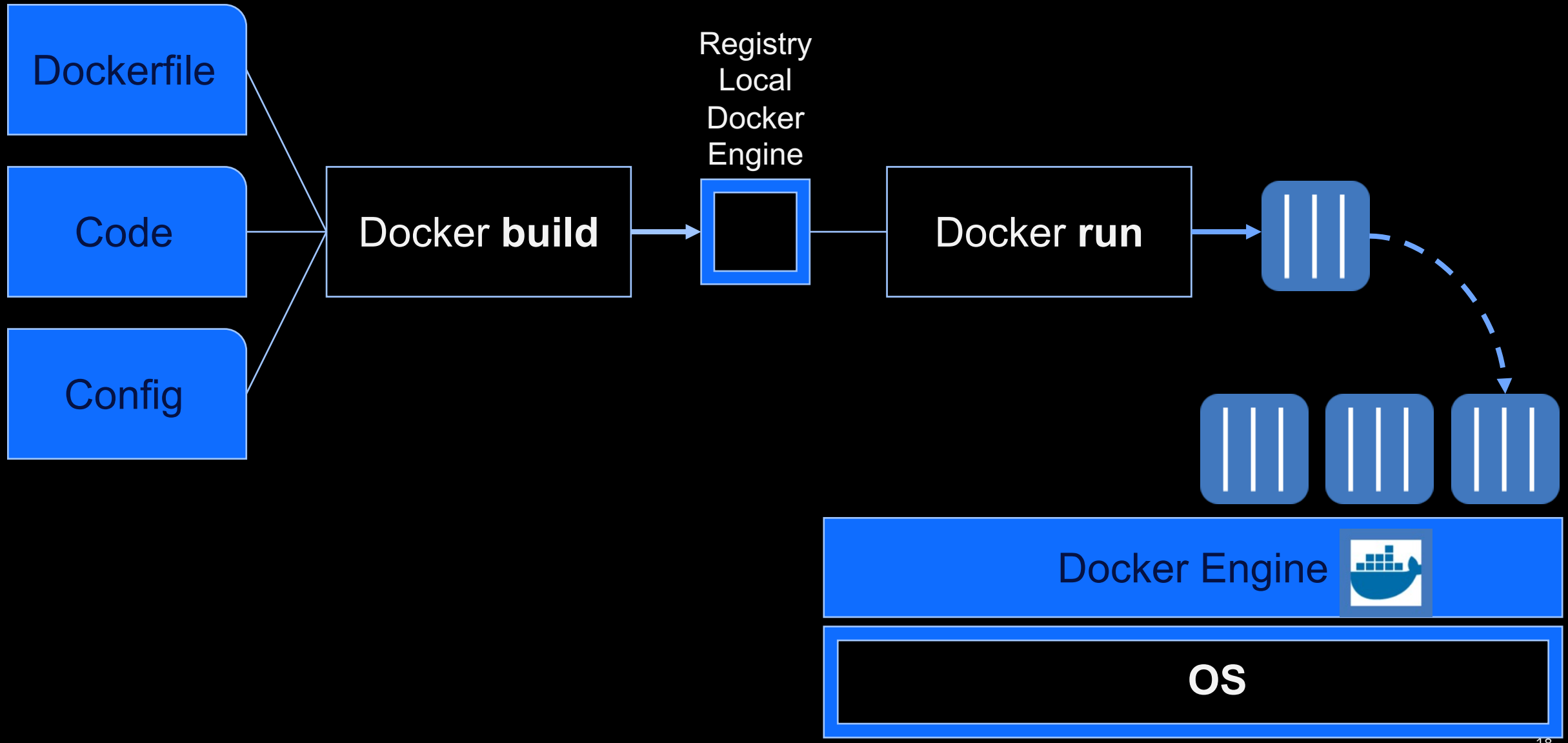


Docker Basics – Layered File System

- Docker uses a **layered filesystem** (copy-on-write)
- New files (& edits) are only visible to current/above layers
- Layers allow for reuse
  - More containers per host
  - Faster start-up/download time – base layers are "cached"



Docker Basics – Run



Docker Basics – Run

Docker Layers

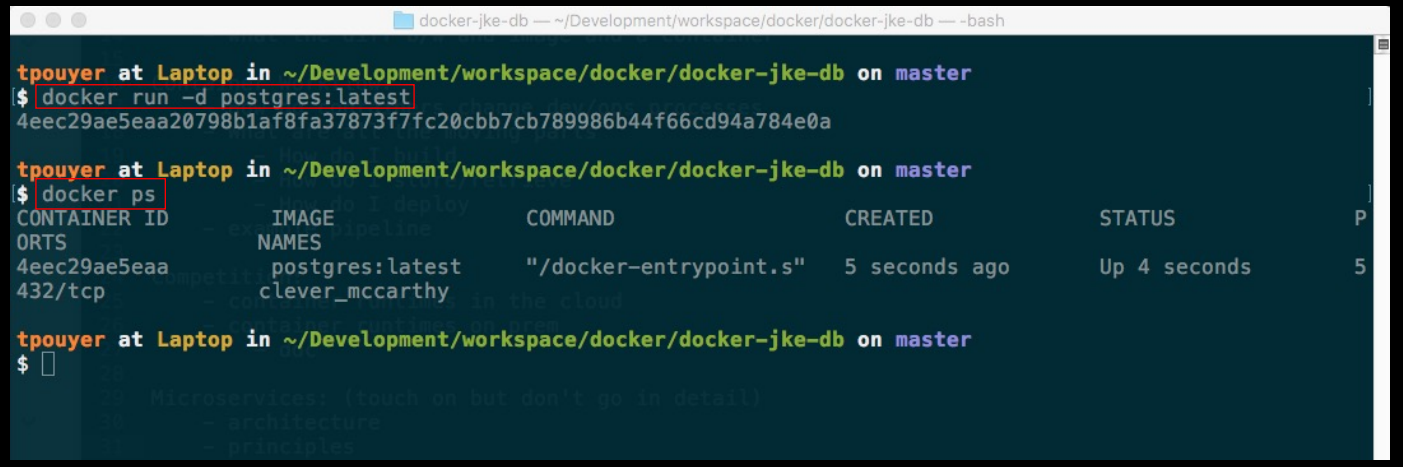
Inheritance

- The biggest difference between a container and an image is the **top writable layer**.
- All writes to the container that add new or modify existing data are stored in this writable layer.
- When a container is deleted, its writable layer is also deleted. The underlying image is unchanged.

Docker Basics – Run

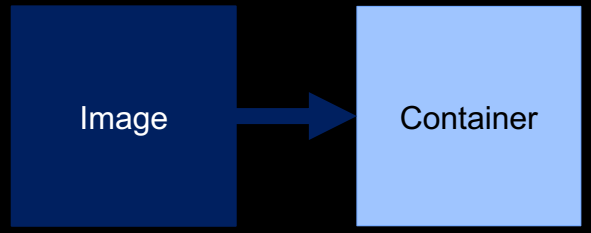
Docker run

- Local (containerd)



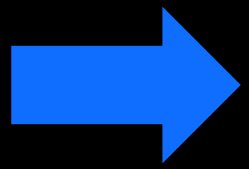

```
docker run -d -e MYVAR=foo -p 8080:80 myimage:1.0.0
docker run -ti myimage:1.0.0 /bin/bash
```

…



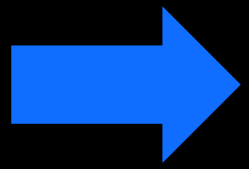
Docker Basics – Run – CMD vs ENTRYPOINT

CMD echo “Hello World”



```
mac> sudo docker run [image_name]
mac> Hello World

mac> sudo docker run [image_name] hostname
mac> my-host-name
```

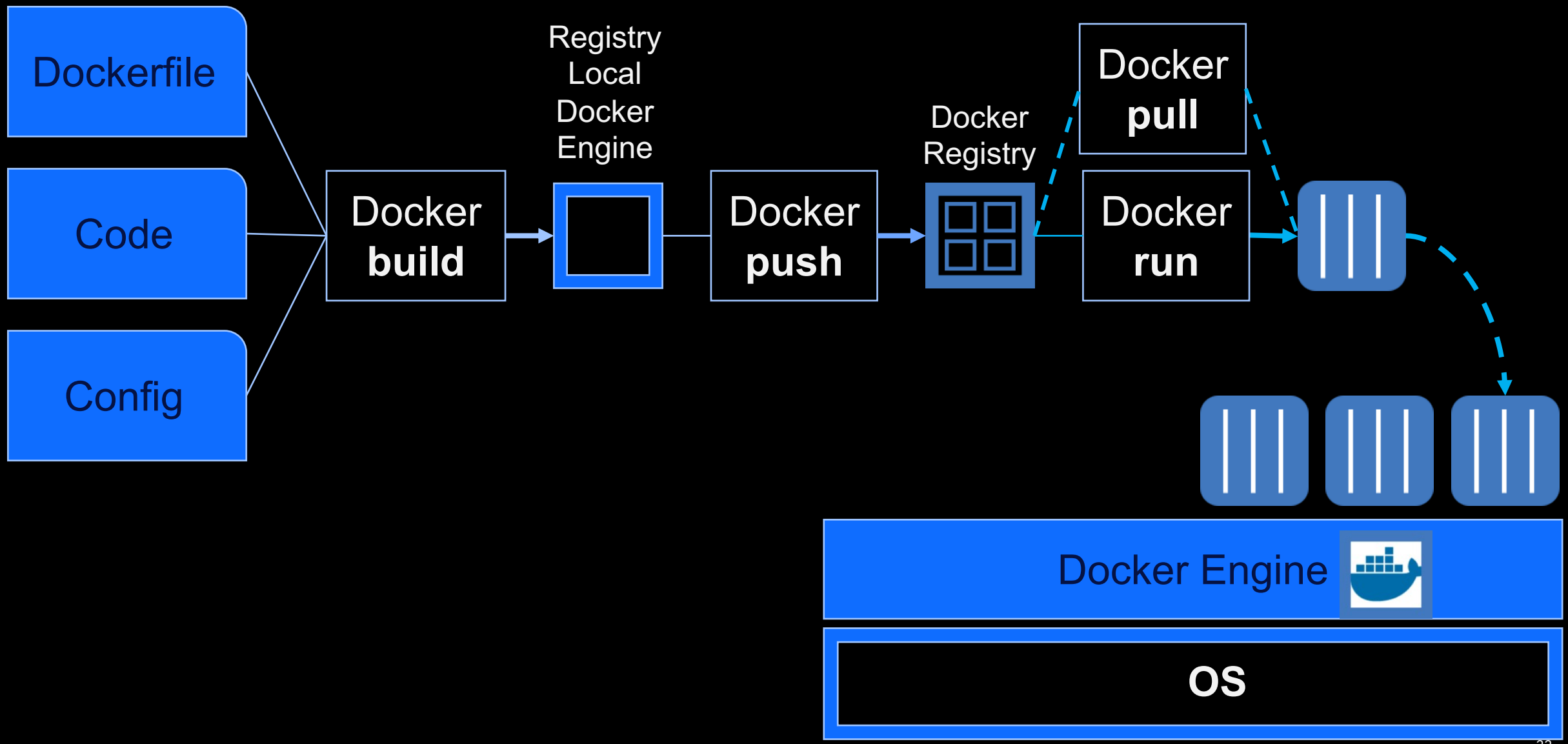
ENTRYPOINT echo "Hello World”



```
mac> sudo docker run [image_name]
mac> Hello World

mac> sudo docker run [image_name] hostname
mac> Hello World  hostname
```

https://phoenixnap.com/kb/docker-cmd-vs-entrypoint

Docker Basics – Store, Retrieve & Run with registry

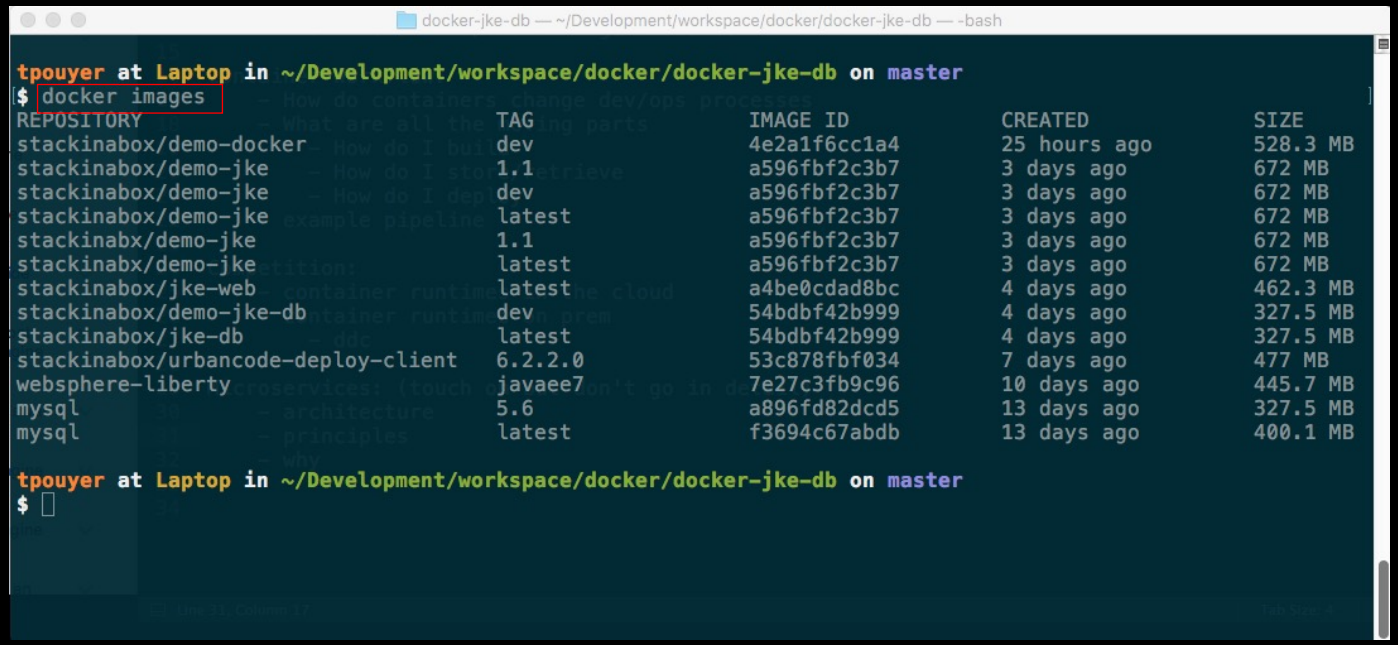


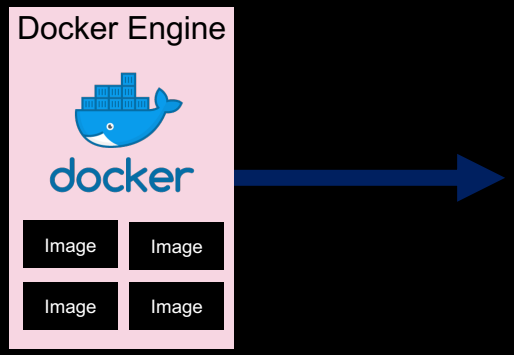
Docker Basics – Store & Retrieve

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker images
```





myregistry/webapp
webapp:latest
webapp:v2.0
webapp:v1.1
webapp:v1.0

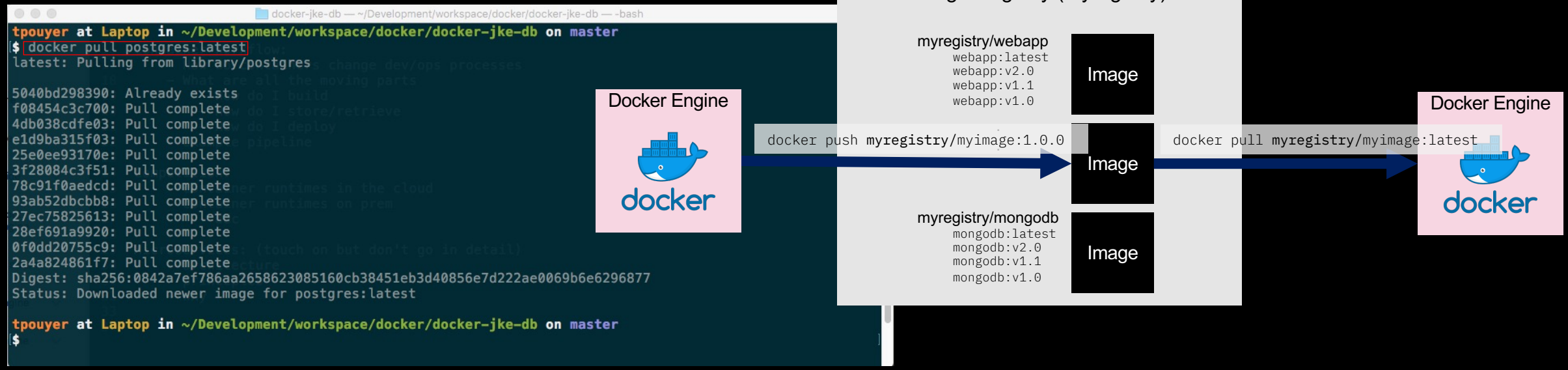
myregistry/mongodb
mongodb:latest
mongodb:v2.0
mongodb:v1.1
mongodb:v1.0

Docker Basics – Store & Retrieve with registry

Docker Registry

- Private Local
- Public Docker Hub
- Private Shared

```
docker tag myimage:1.0.0 myregistry/myimage:1.0.0
docker push myregistry/myimage:1.0.0

docker pull myregistry/myimage:latest
```



Docker Basics – Registries

Hosting image repositories

- You can define your own registry
- A registry is managed by a registry container

Public and Private registries

- Public Registry like **Quay** or **Docker Hub**
- **https://quay.io**
- **https://hub.docker.com**

Login into the registry

- Docker login domain:port

Docker Recap

- **Containers are not VMs**
- **Containers provide many benefits:**
  - Efficiency
  - Portability
  - Consistency
- **New challenges with containers:**
  - Production apps dependent on open-source projects
  - Existing tools may not be sufficient for container
  - Need to focus on business objectives

QUESTIONS?