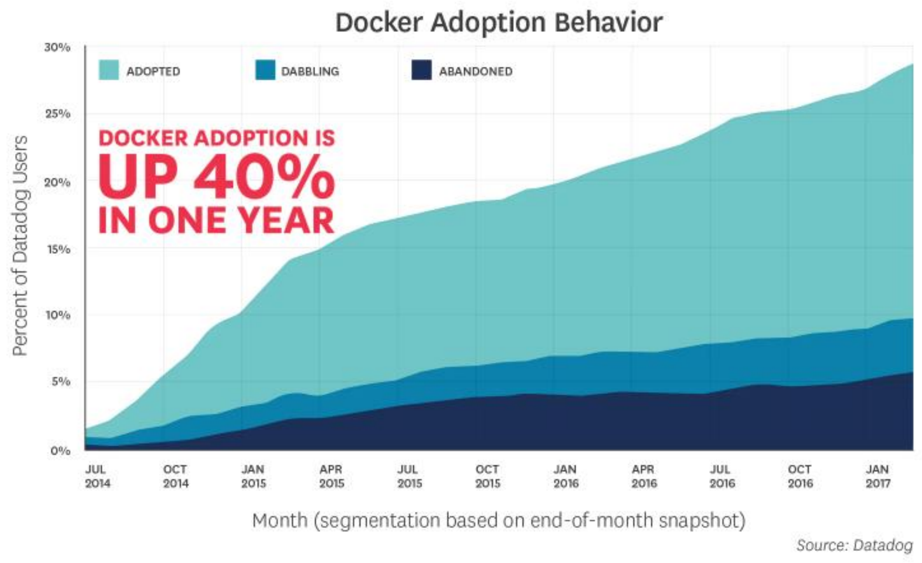
IBM Cloud Container Workshop

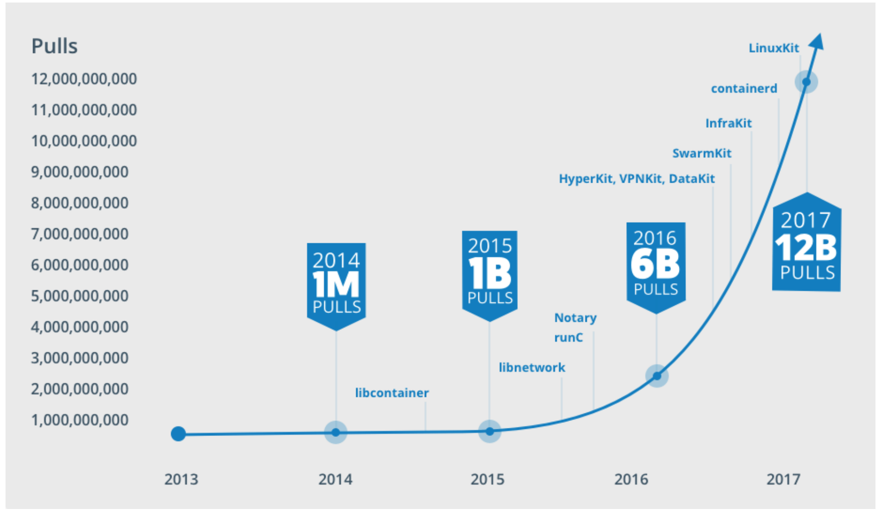
Part1 - Containers

Market Dynamics and Use Cases

- Container Adoption Drivers
  - Microservice Patterns
  - Cloud Native Applications
  - Hybrid Cloud
  - CD/CI in DevOps
  - Modernizing Applications

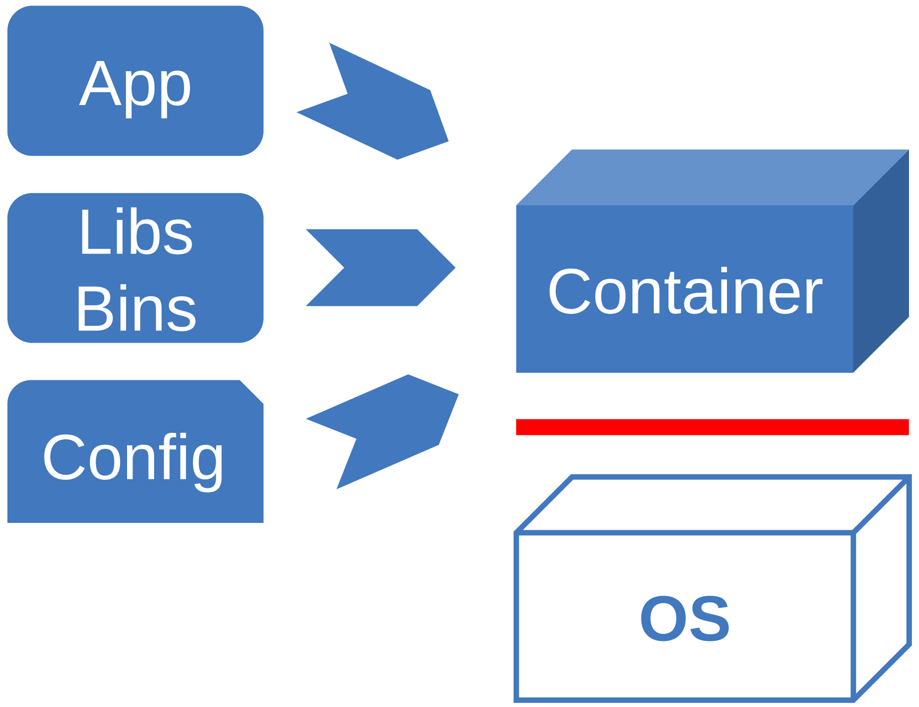
- All industries are impacted





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.



Why Customers are interested in Containers

- #1 : Application Portability
    - Isolated containers package the application, dependencies and configurations together. These containers can then seamlessly move across environments and infrastructures.
- Ship More Software
    - Accelerate development & deployment, CI and CD pipelines by eliminating headaches of setting up environments and dealing with differences between environments. On average, Docker users ship software 7X more frequently[1].
- Resource Efficiency
    - Lightweight containers run on a single machine and share the same OS kernel while images are layered file systems sharing common files to make efficient use of RAM and disk and start instantly.

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

cgroups

2007

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

warden

2011

2012

Docker

rocket

2014

rkt

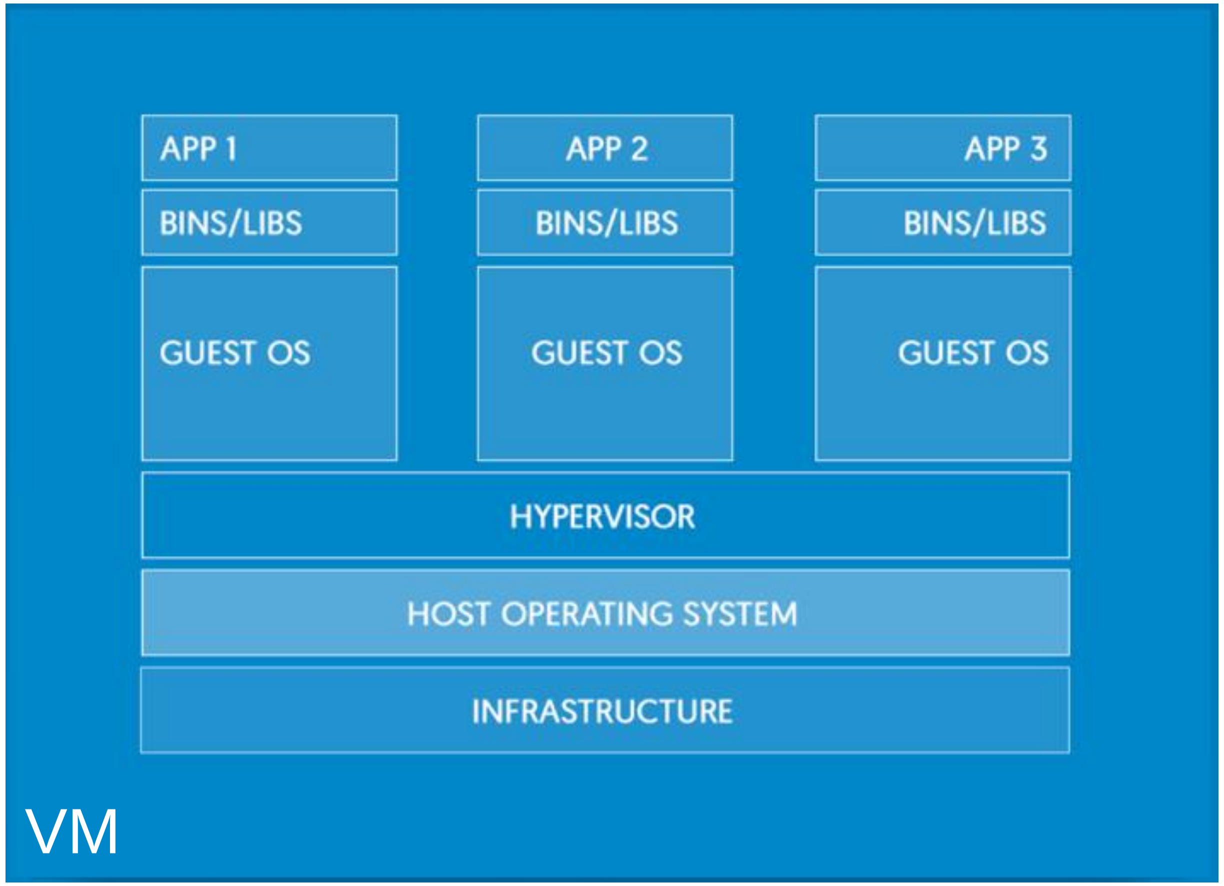
2016

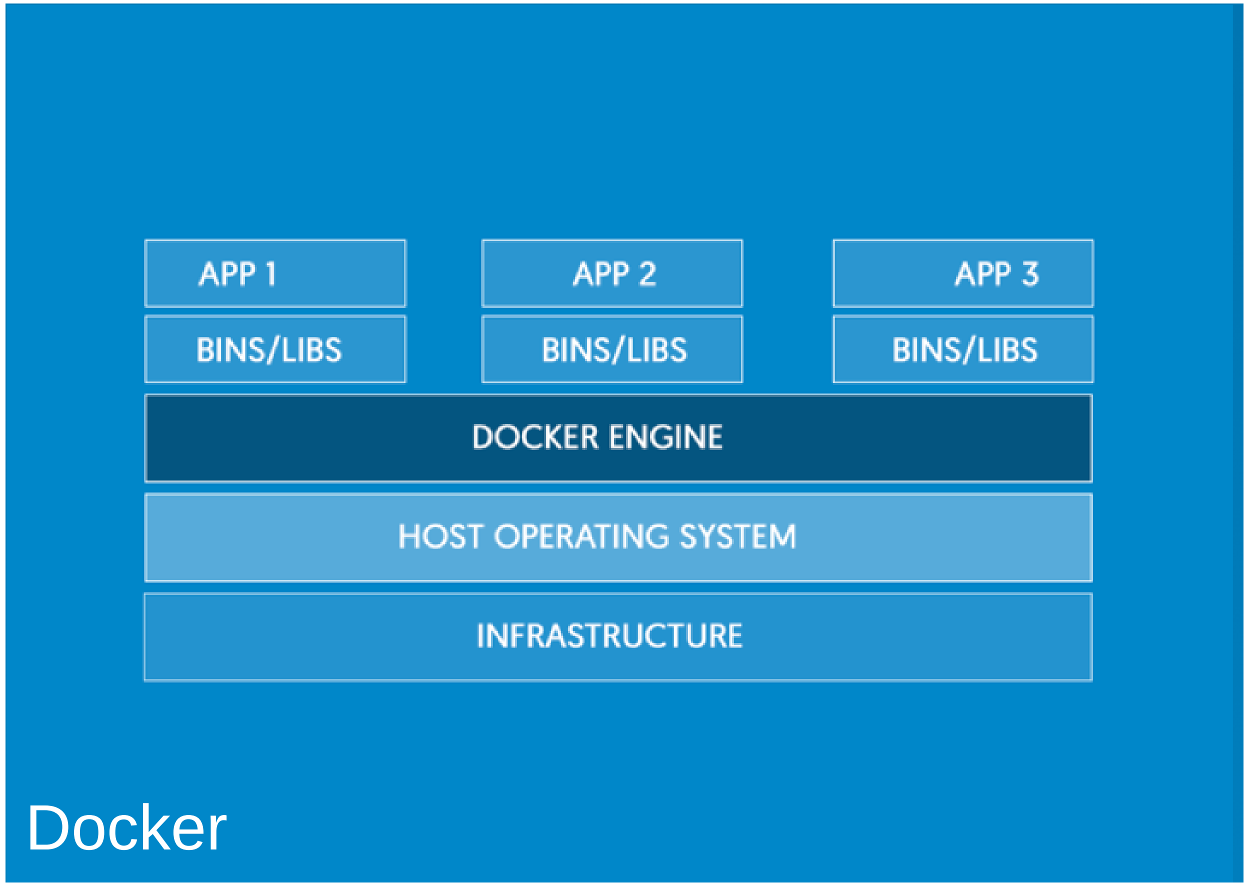
Garden, runC

Multiple De Facto Container Standards

- Docker
  - The most common standard, made Linux containers usable by masses
- Rocket (rkt)
  - An emerging container standard from CoreOS, the company that developed etcd
- Garden
  - The format Cloud Foundry builds using buildpacks
- Open Container Initiative (OCI)
  - A Linux Foundation project developing a governed container standard

VMs, Containers and Docker

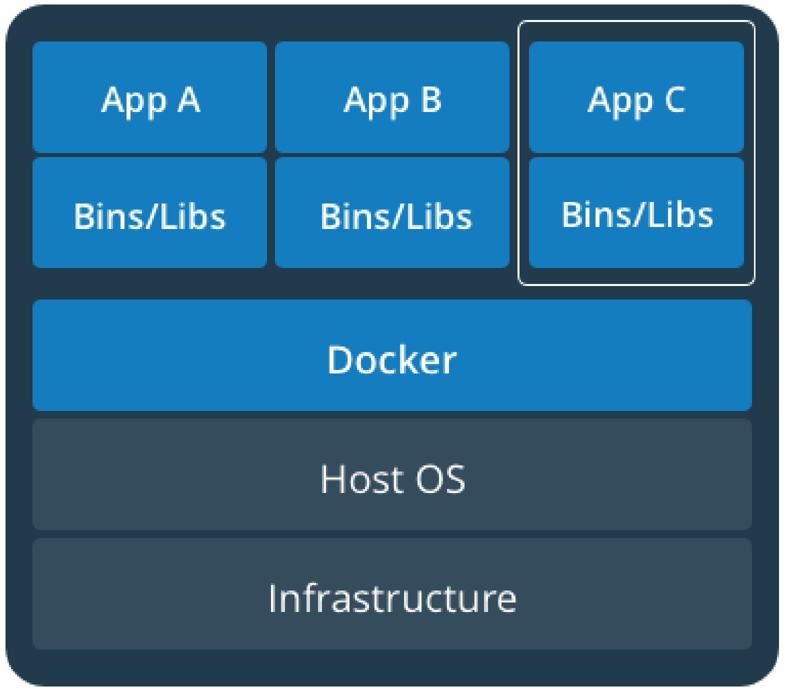




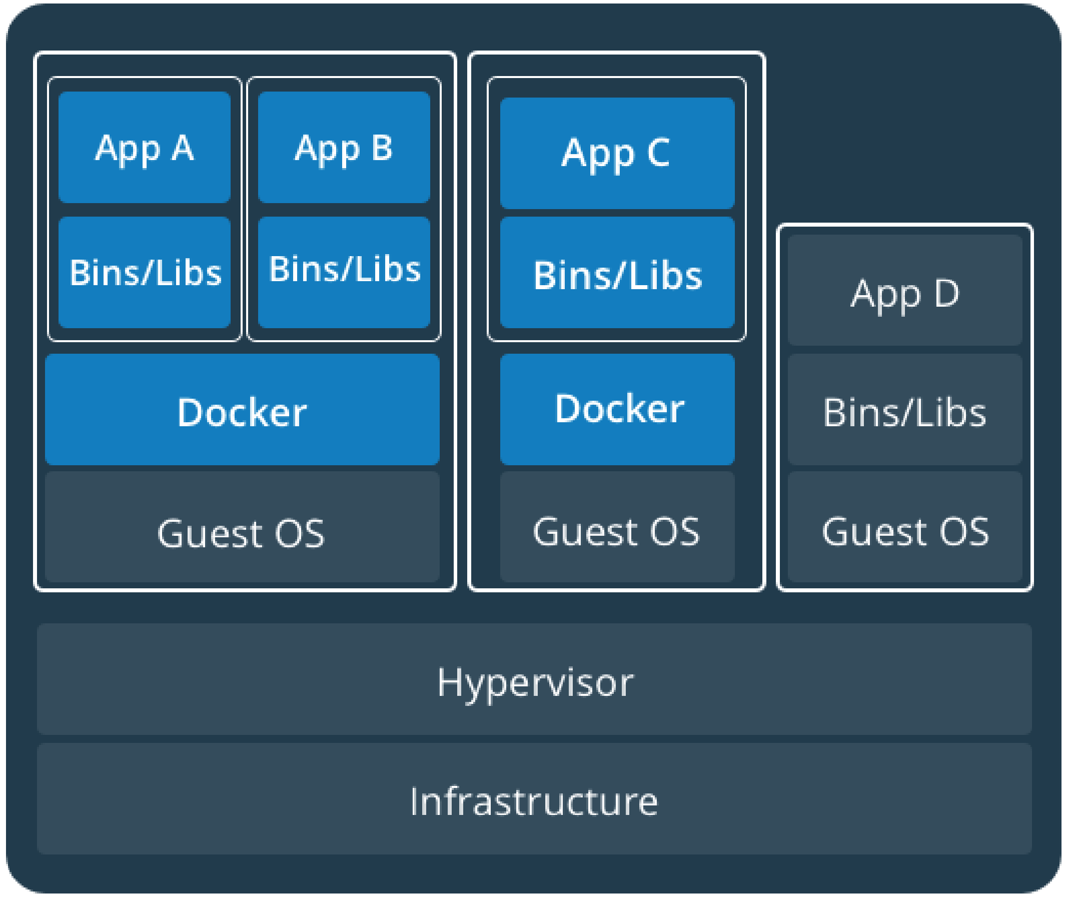
Docker = Linux namespaces + cgroups + overlay (union) file system + image format

Containers and VMs Together

Containers on Bare Metal

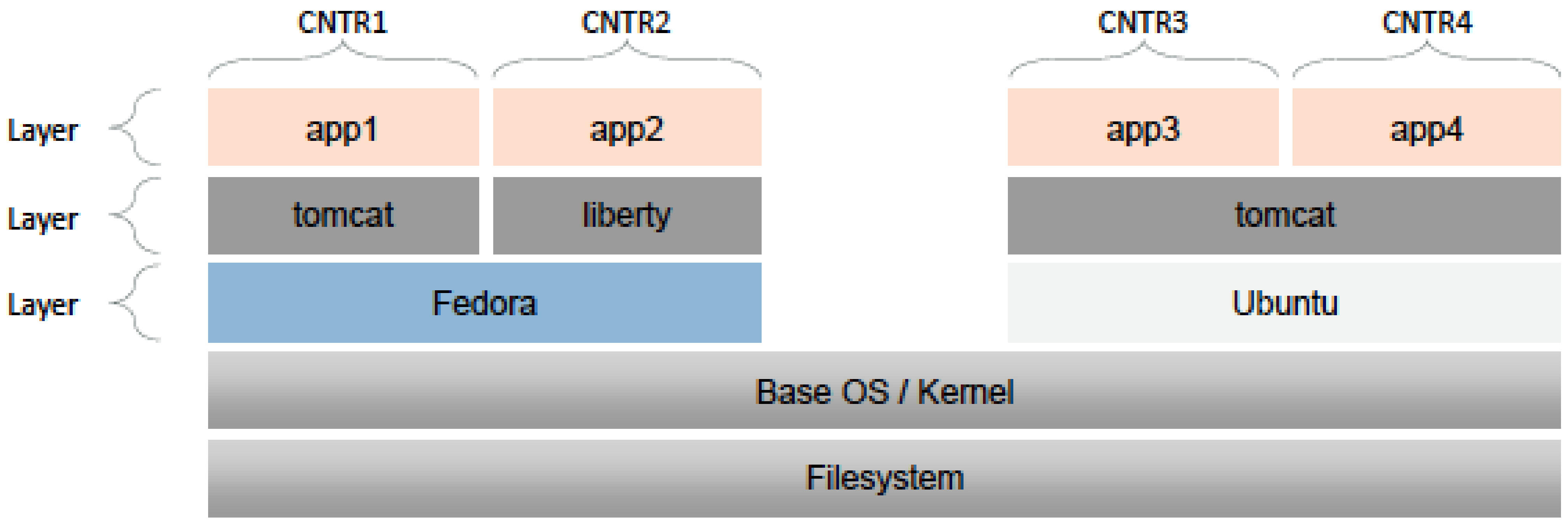


Containers on Virtual Machines



Docker Containers

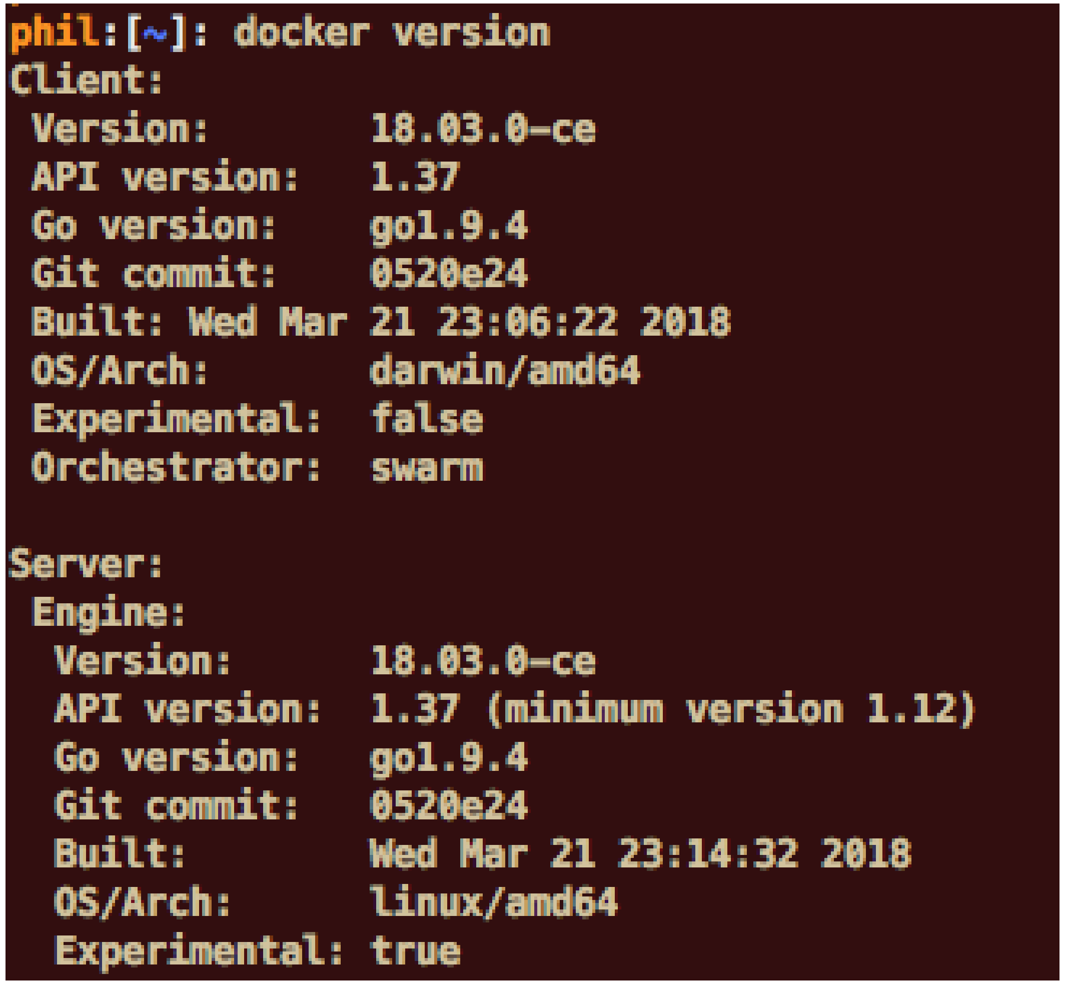
- Docker uses a copy-on-write (union) filesystem
- New files (& edits) are only visible to current/above layers (used for reuse)



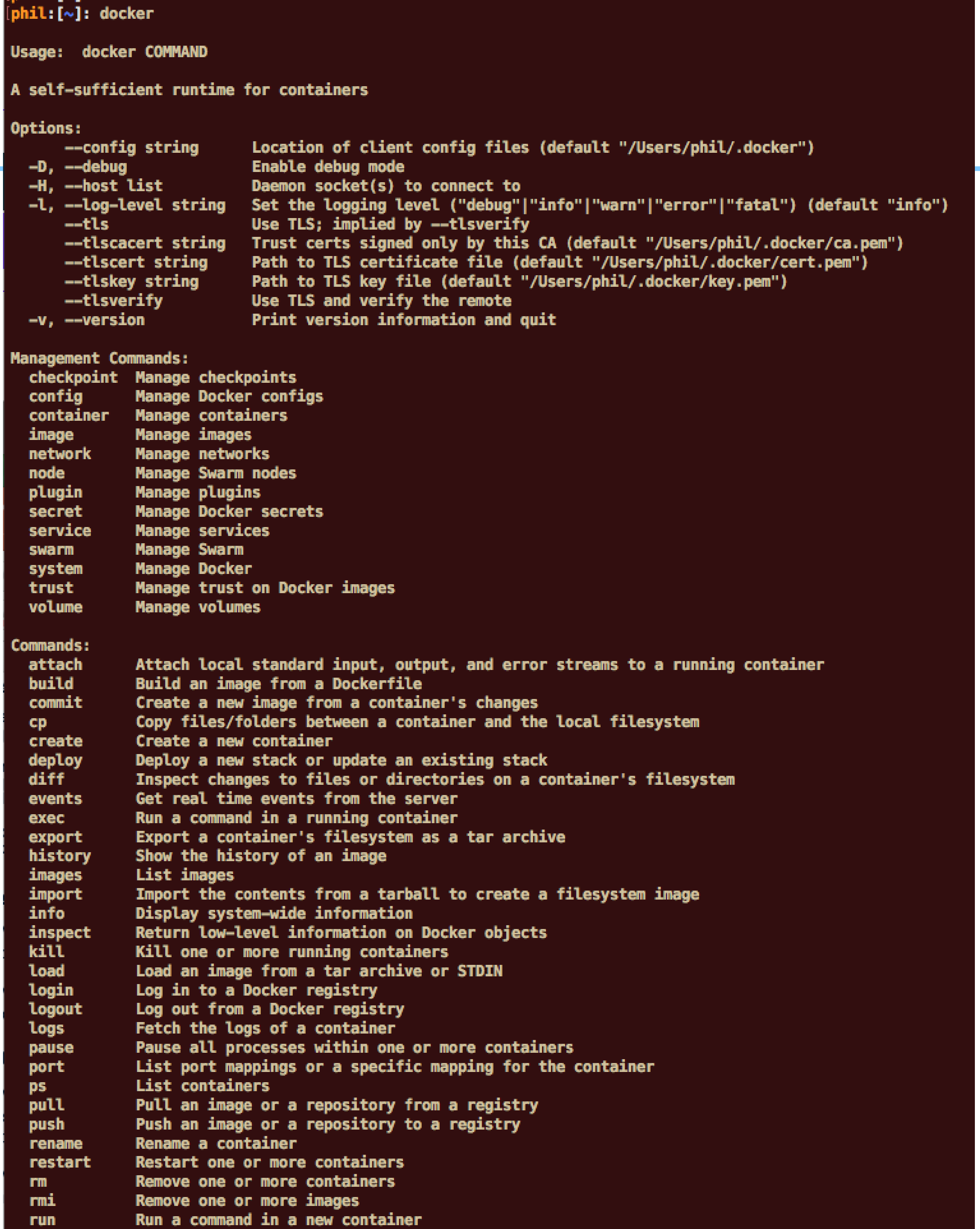
Docker Terminology

- Image
  - A read-only snapshot of a container stored in a registry to be used as a template for building containers. At rest.
- Container
  - The image when it is ‘running.’ The standard unit for app service
- Registry
  - Stores, distributes and manages Docker images
- Engine
  - The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.
- Control Plane
  - Management plane for container and cluster **orchestration**

Docker Commands (CLI)



```
phil:[~]: docker version
Client:
 Version:       18.03.0-ce
 API version:   1.37
 Go version:    go1.9.4
 Git commit:    0520e24
 Built: Wed Mar 21 23:06:22 2018
 OS/Arch:       darwin/amd64
 Experimental:  false
 Orchestrator:  swarm

Server:
 Engine:
  Version:      18.03.0-ce
  API version:  1.37 (minimum version 1.12)
  Go version:   go1.9.4
  Git commit:   0520e24
  Built:        Wed Mar 21 23:14:32 2018
  OS/Arch:      linux/amd64
  Experimental: true
```

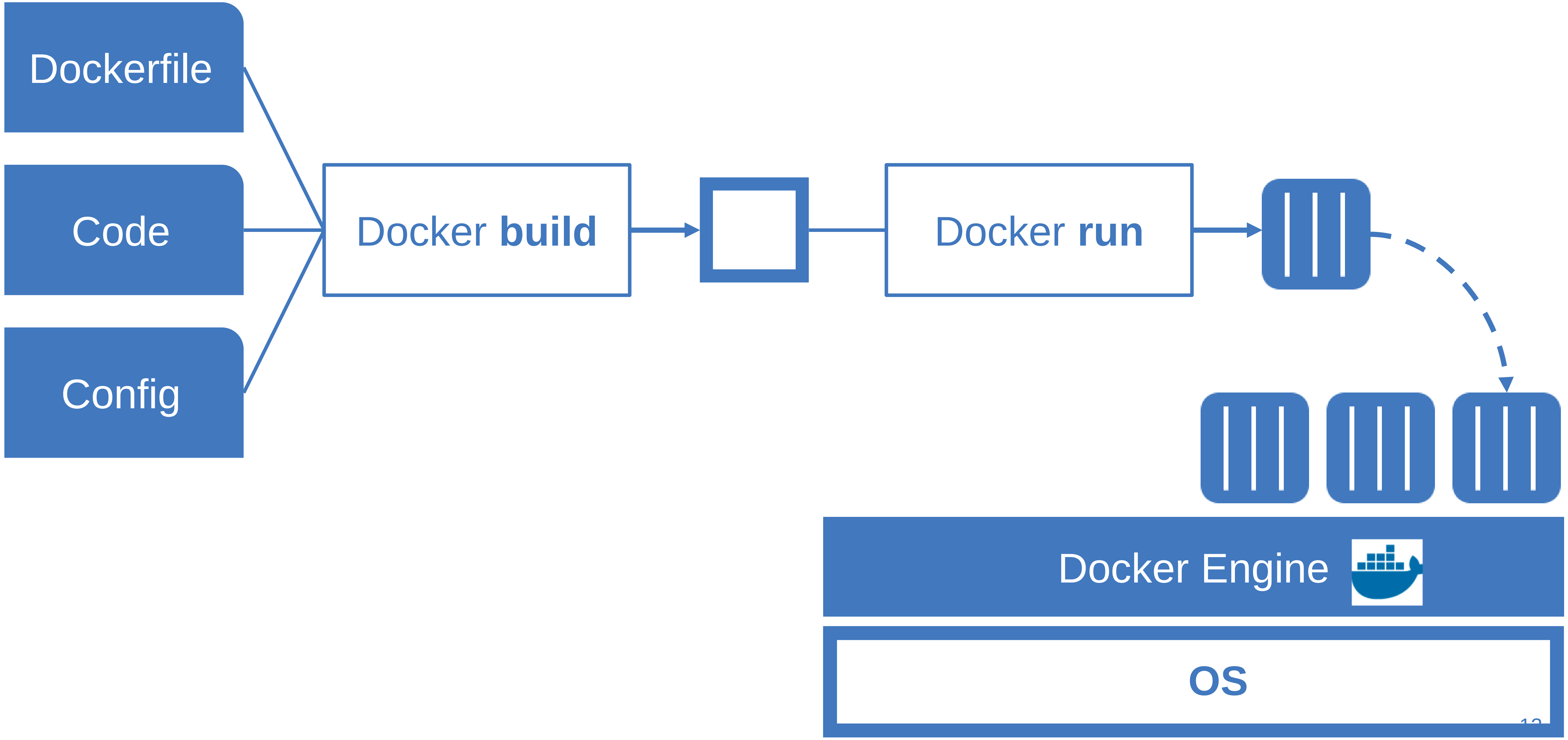


```
phil:[~]: docker

Usage:  docker COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/Users/phil/.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/Users/phil/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/Users/phil/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/Users/phil/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  checkpoint  Manage checkpoints
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  deploy      Deploy a new stack or update an existing stack
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
```

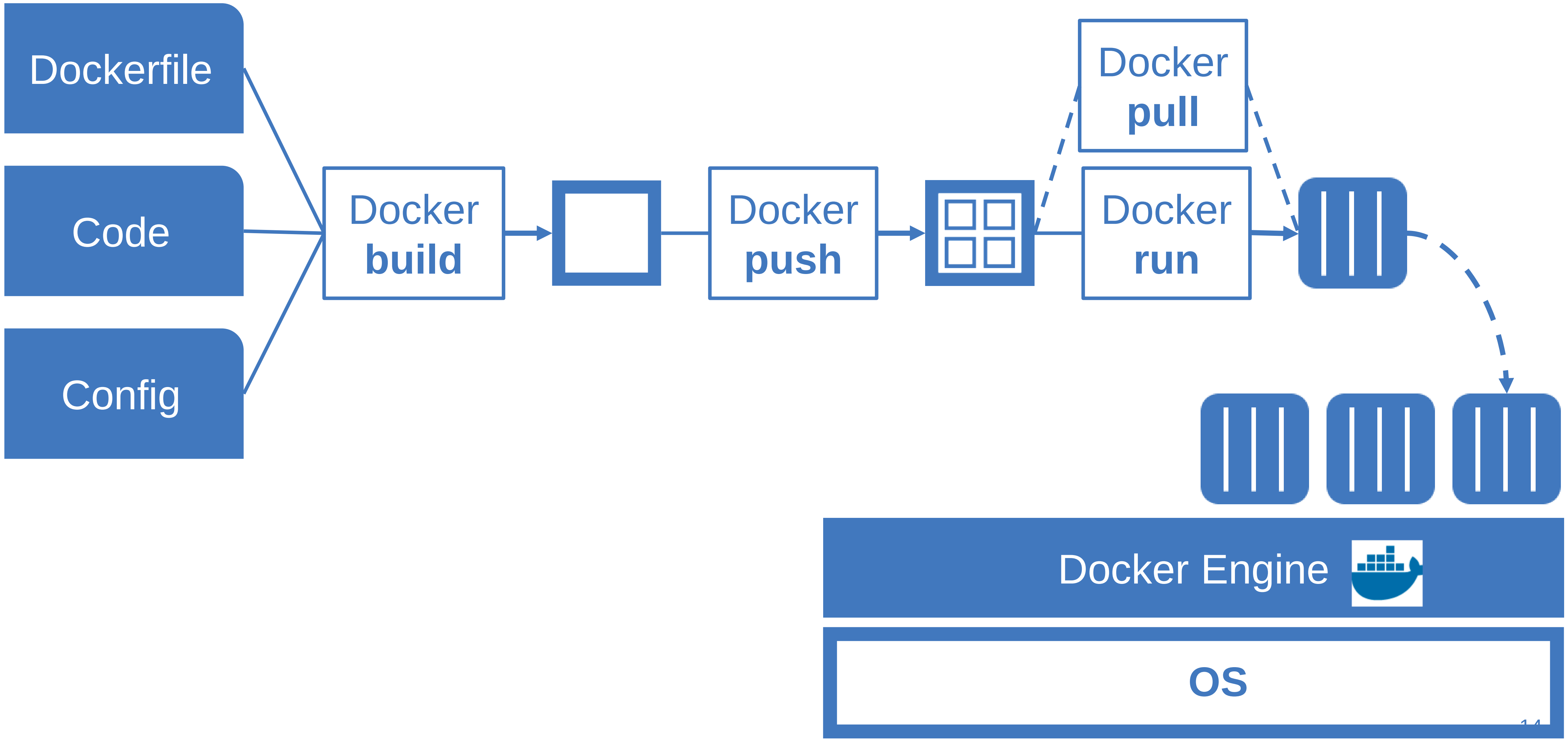
Most Useful Docker Commands

• docker build	used to build the image with the help of the Dockerfile
• docker push	push the image into a registry
• docker images	list images in a registry
• docker run	run the container or a command in a container
• docker ps	list containers
• docker kill	kill one or more container
• docker exec	run a command in a running container
• docker top	display the running processes in a container
• docker container	manages container details
• docker network	manages networks for containers

Docker Supply Chain (local)



Docker Supply Chain (with a registry)



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 Docker Hub
  - https://hub.docker.com
- Login into the registry
  - Docker login domain:port

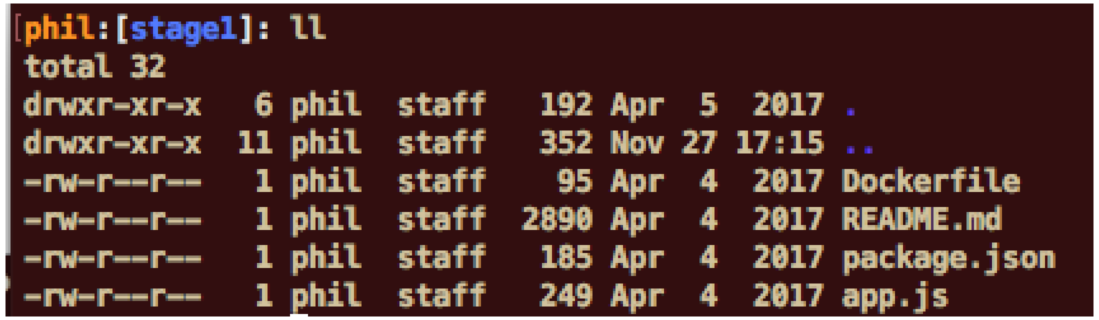
Dockerfile

- Build an image automatically
- Specifies base image and instructions:
  - FROM <existing image>
  - ADD <local file> <path inside image>
  - RUN <cmd>
  - EXPOSE <port>
  - ENV <name> <value>
  - CMD <cmd>

```
# Use latest jboss/base-jdk:7 image as the base
FROM jboss/base-jdk:7

# Set the WILDFLY_VERSION env variable
ENV WILDFLY_VERSION 8.1.0.Final

# Add the WildFly distribution to /opt
RUN cd $HOME && curl http://download.jboss.org/wildfly/$W
$WILDFLY_VERSION.tar.gz | tar zx && mv $HOME/wildfly-$

# Set the JBOSS_HOME env variable
ENV JBOSS_HOME /opt/jboss/wildfly

# Expose the ports we're interested in
EXPOSE 8080 9990

# Set the default command to run on boot
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0"]
```

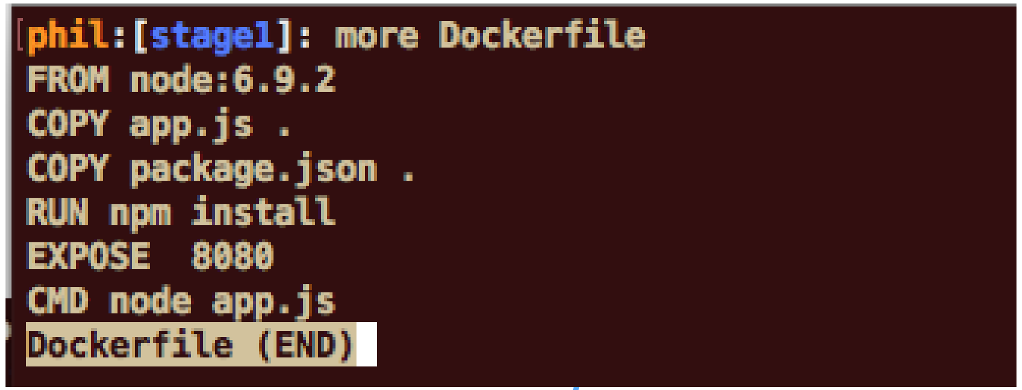
Dockerfile > Build > Run > Push > Run

1



```
phil:[stage1]: ll
total 32
drwxr-xr-x   6 phil  staff   192 Apr  5  2017 .
drwxr-xr-x  11 phil  staff   352 Nov 27 17:15 ..
-rw-r--r--   1 phil  staff    95 Apr  4  2017 Dockerfile
-rw-r--r--   1 phil  staff  2890 Apr  4  2017 README.md
-rw-r--r--   1 phil  staff   185 Apr  4  2017 package.json
-rw-r--r--   1 phil  staff   249 Apr  4  2017 app.js
```

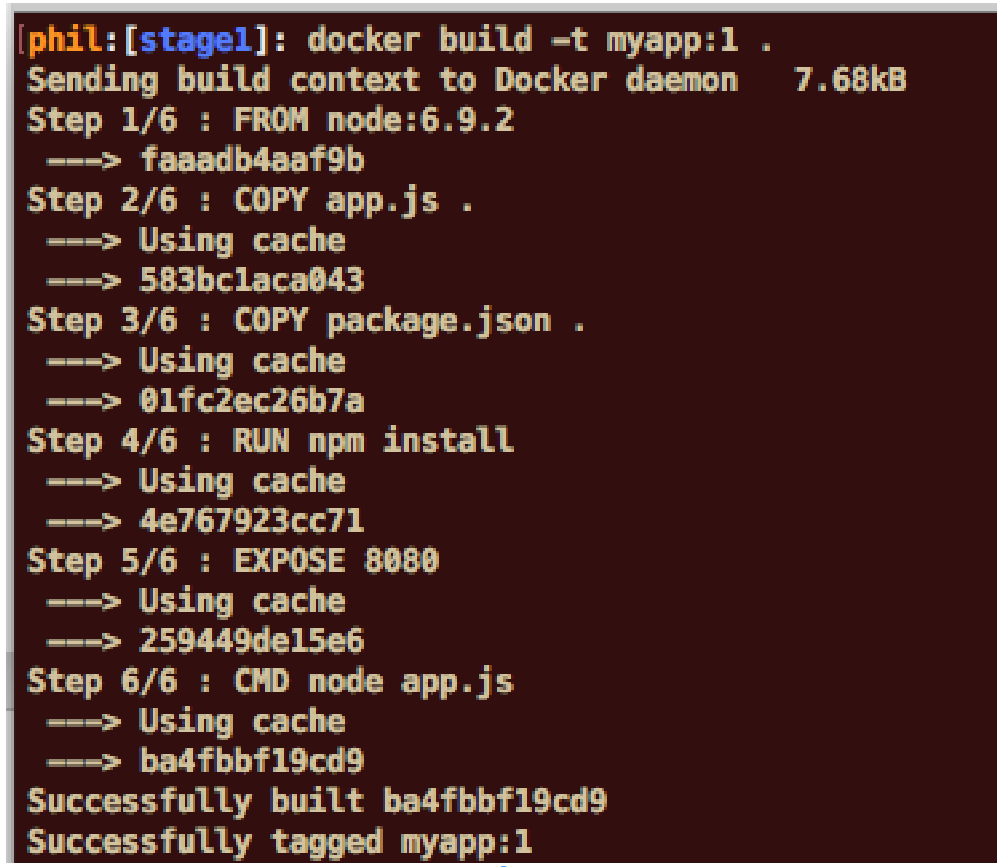
2



```
phil:[stage1]: more Dockerfile
FROM node:6.9.2
COPY app.js .
COPY package.json .
RUN npm install
EXPOSE  8080
CMD node app.js
Dockerfile (END)
```

Configuring Dockerfile

3

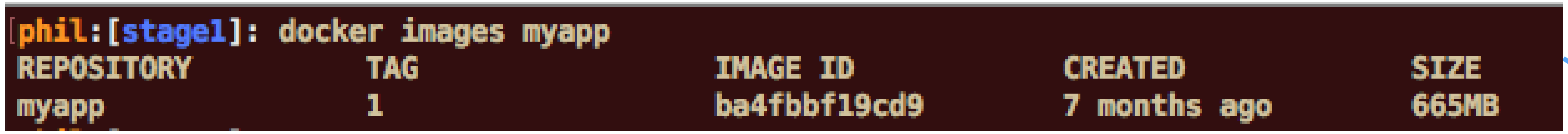


```
phil:[stage1]: docker build -t myapp:1 .
Sending build context to Docker daemon   7.68kB
Step 1/6 : FROM node:6.9.2
 ---> faaadb4aaf9b
Step 2/6 : COPY app.js .
 ---> Using cache
 ---> 583bc1aca043
Step 3/6 : COPY package.json .
 ---> Using cache
 ---> 01fc2ec26b7a
Step 4/6 : RUN npm install
 ---> Using cache
 ---> 4e767923cc71
Step 5/6 : EXPOSE 8080
 ---> Using cache
 ---> 259449de15e6
Step 6/6 : CMD node app.js
 ---> Using cache
 ---> ba4fbbf19cd9
Successfully built ba4fbbf19cd9
Successfully tagged myapp:1
```

Building image

Dockerfile > Build > Run > Push > Run

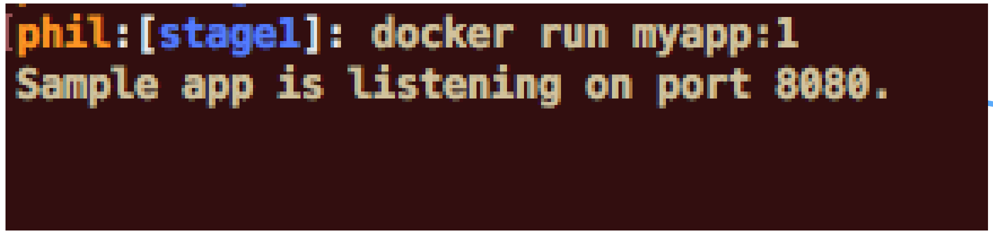
4



```
phil:[stage1]: docker images myapp
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
myapp               1                   ba4fbbf19cd9        7 months ago        665MB
```

Listing images

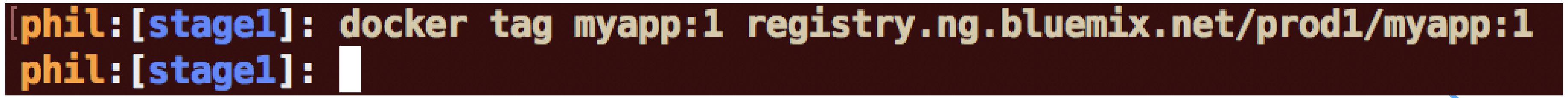
5



```
phil:[stage1]: docker run myapp:1
Sample app is listening on port 8080.
```

Running the image locally

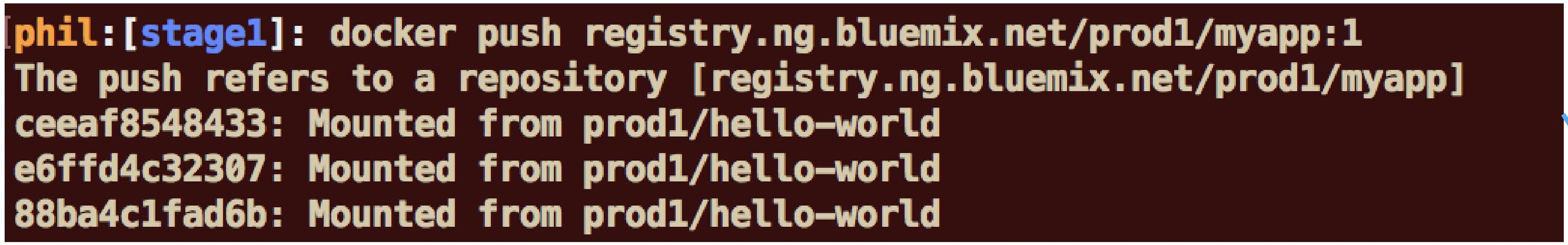
6



```
phil:[stage1]: docker tag myapp:1 registry.ng.bluemix.net/prod1/myapp:1
phil:[stage1]: 
```

Tagging the image

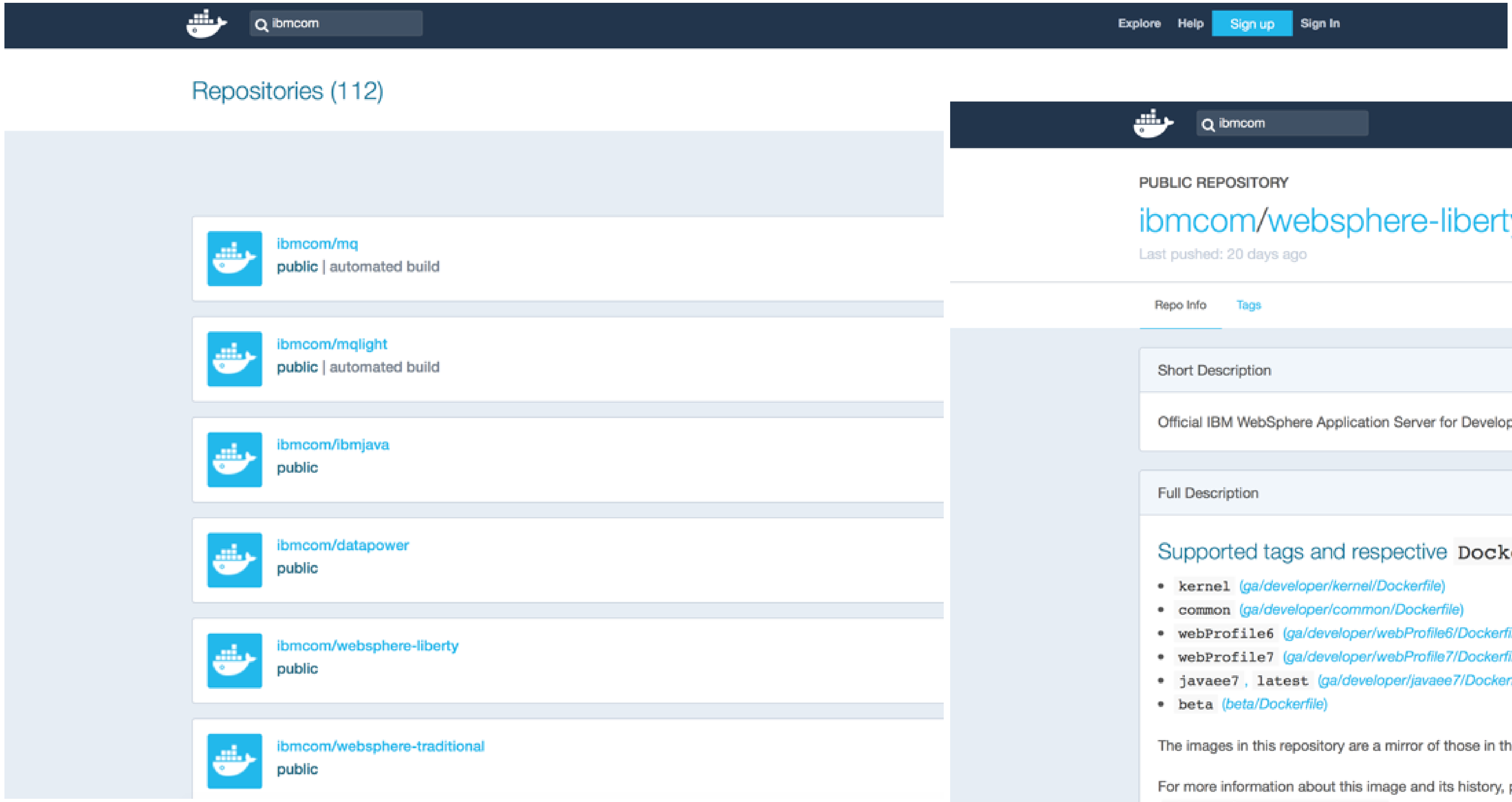
7



```
phil:[stage1]: docker push registry.ng.bluemix.net/prod1/myapp:1
The push refers to a repository [registry.ng.bluemix.net/prod1/myapp]
ceeaf8548433: Mounted from prod1/hello-world
e6ffd4c32307: Mounted from prod1/hello-world
88ba4c1fad6b: Mounted from prod1/hello-world
```

Pushing the image

IBM Software on Docker Hub



Repositories (112)

ibmcom/mq
public | automated build

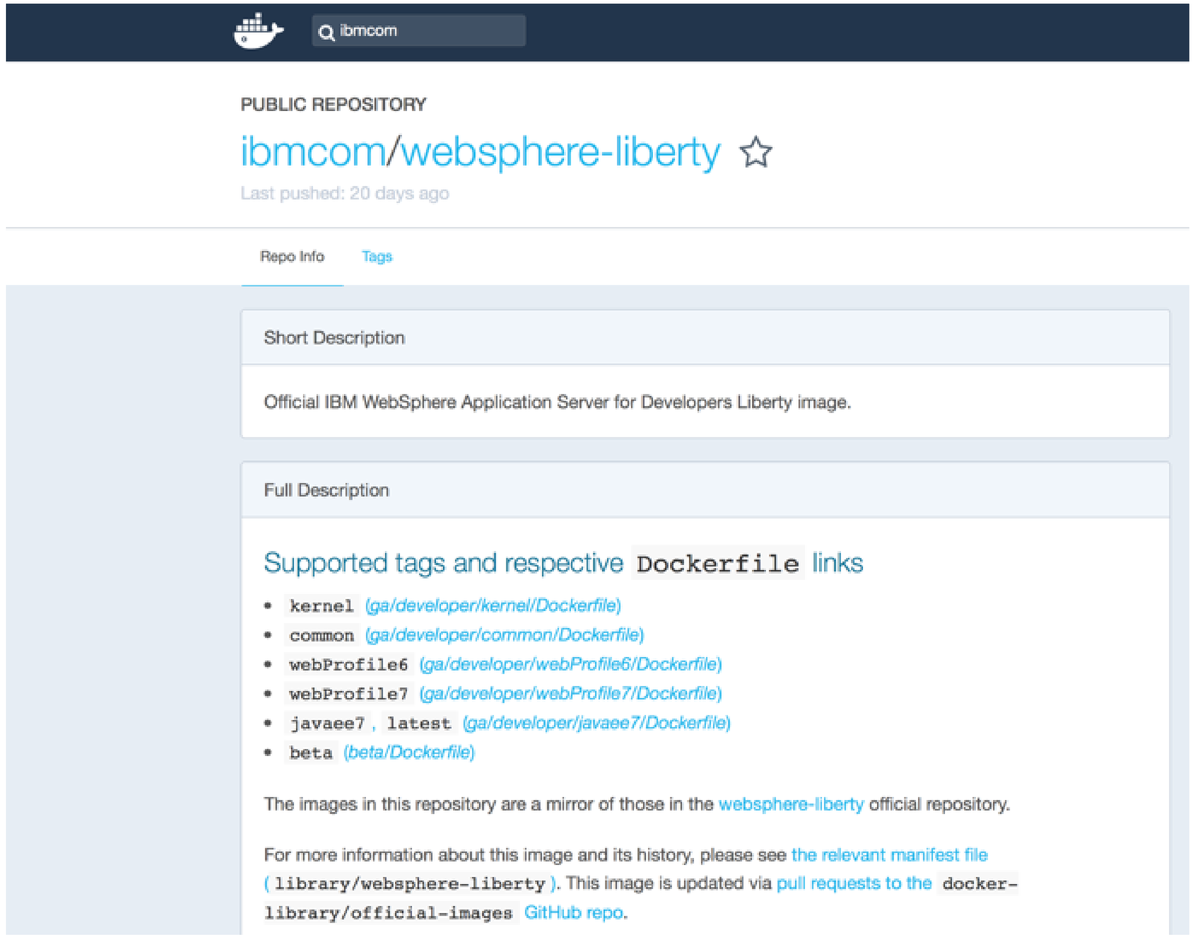
ibmcom/mqlight
public | automated build

ibmcom/ibmjava
public

ibmcom/datapower
public

ibmcom/websphere-liberty
public

ibmcom/websphere-traditional
public



PUBLIC REPOSITORY

ibmcom/websphere-liberty

Last pushed: 20 days ago

Short Description

Official IBM WebSphere Application Server for Developers Liberty image.

Full Description

Supported tags and respective `Dockerfile` links

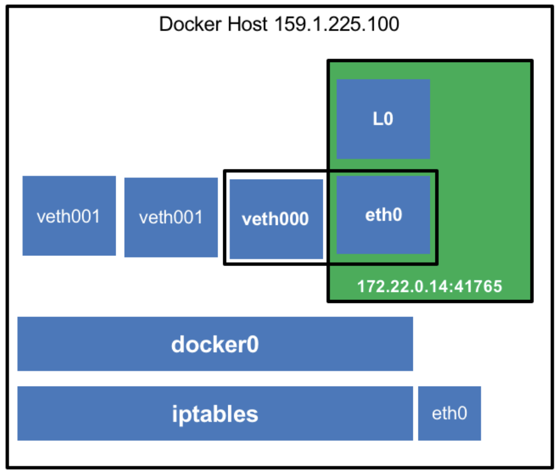
- `kernel` (ga/developer/kernel/Dockerfile)
- `common` (ga/developer/common/Dockerfile)
- `webProfile6` (ga/developer/webProfile6/Dockerfile)
- `webProfile7` (ga/developer/webProfile7/Dockerfile)
- `javaee7` , `latest` (ga/developer/javaee7/Dockerfile)
- `beta` (beta/Dockerfile)

The images in this repository are a mirror of those in the websphere-liberty official repository.

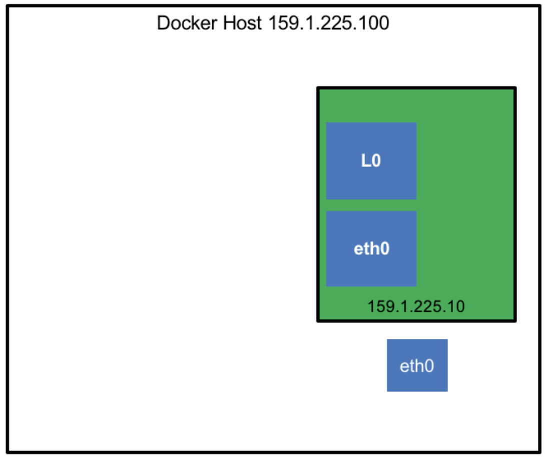
For more information about this image and its history, please see the relevant manifest file (`library/websphere-liberty`). This image is updated via pull requests to the `docker-library/official-images` GitHub repo.

Docker Networking in a Host

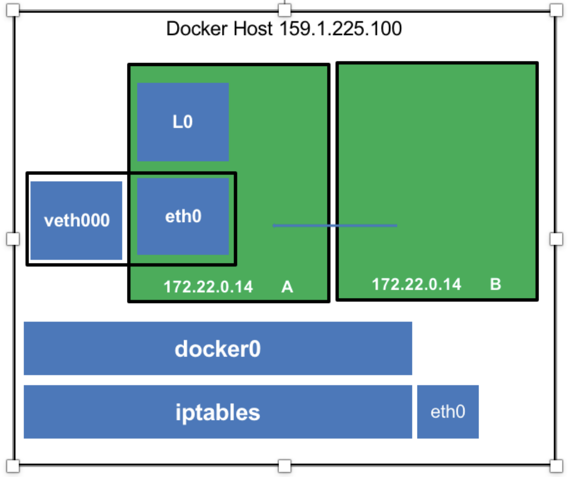
Bridge Mode



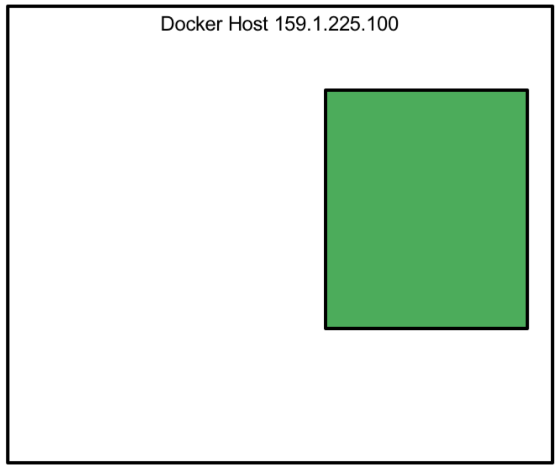
Host Mode



Container Mode



No Networking



```
docker run -d --net=bridge nginx:1.9.1
docker run -d –P --net=host ubuntu:14.04
docker run -it --net=container:anothercontainer ubuntu:14.04 ip addr …
docker run -d --net=none ubuntu:latest
```

Docker Networking for Multiple Hosts

- SDN = Software Defined Network
- L2 solution (overlay netwo*rk)* :
    - Docker Networking (default)
    - Flannel
    - Weave Net
    - Open vSwitch
    - OpenVPN

- Project **Calico** (L3 solution & BGP)

Docker-Compose

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a YAML file to configure your application's services.
- Then, with a **single command**, you create and start all the services from your configuration.
    - docker-compose up

docker-compose.yml

```yaml
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

IBM and Docker Partnership

- Strategic partnership announced December, 2014
  - https://www-03.ibm.com/press/us/en/pressrelease/45597.wss

- Partnership extended February, 2016
  - IBM initially only partner to resell and support Docker Datacenter

- Objective: Deliver next generation enterprise-grade, portable, distributed applications that are composed of interoperable Docker containers
  - Enables hybrid cloud use cases for the enterprise
  - **IBM Cloud Container Service** since 2014

- Initiatives Underway especially with **IBM Cloud Private**

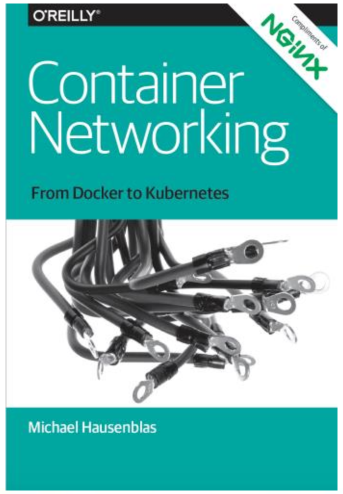
Advantages of Containers

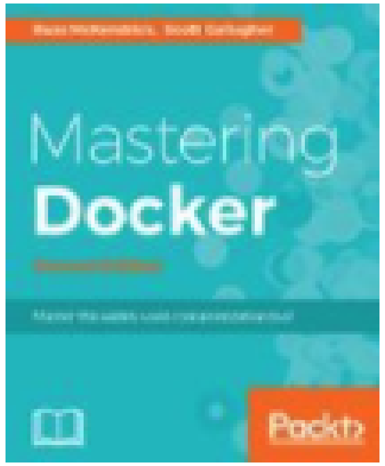
- Containers are **portable**
  - Any platform with a container engine can run containers
- Containers are **easy** to manage
  - Container images are easy to share, download, and delete
    - Especially with Docker registries
  - Container instances are easy to create and delete
  - Each container instance is easy and fast to start and stop
- Containers provide “just enough” **isolation**
  - Processes share the operating system kernel but are segregated
- Containers use hardware more **efficiently**
  - Greater density than virtual machines
  - Especially Docker containers, which can share layers
- Containers are **immutable**

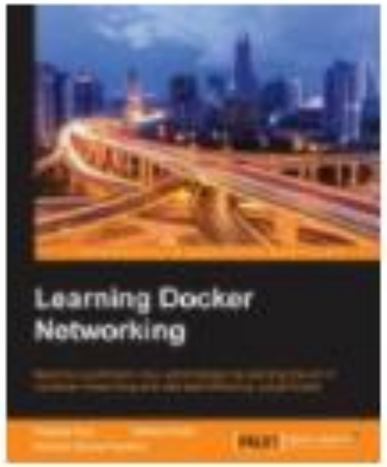
Books, eBooks and links

- Mastering Docker (second edition)
- Learning Docker Networking
- Container Networking
- Monitoring Docker

- https://docs.docker.com/







Preparation Lab & Docker Lab

Labs

Labs

- https://github.com/Azzoz06/ContainerWkshp/blob/master/2-DockerLab.md

PrepareLab

- Installing Docker on your laptop
- Installaing the ibmcloud (ic) commands

DockerLab

- Working with Docker
- Building Docker images
- Running Web Application