Docker

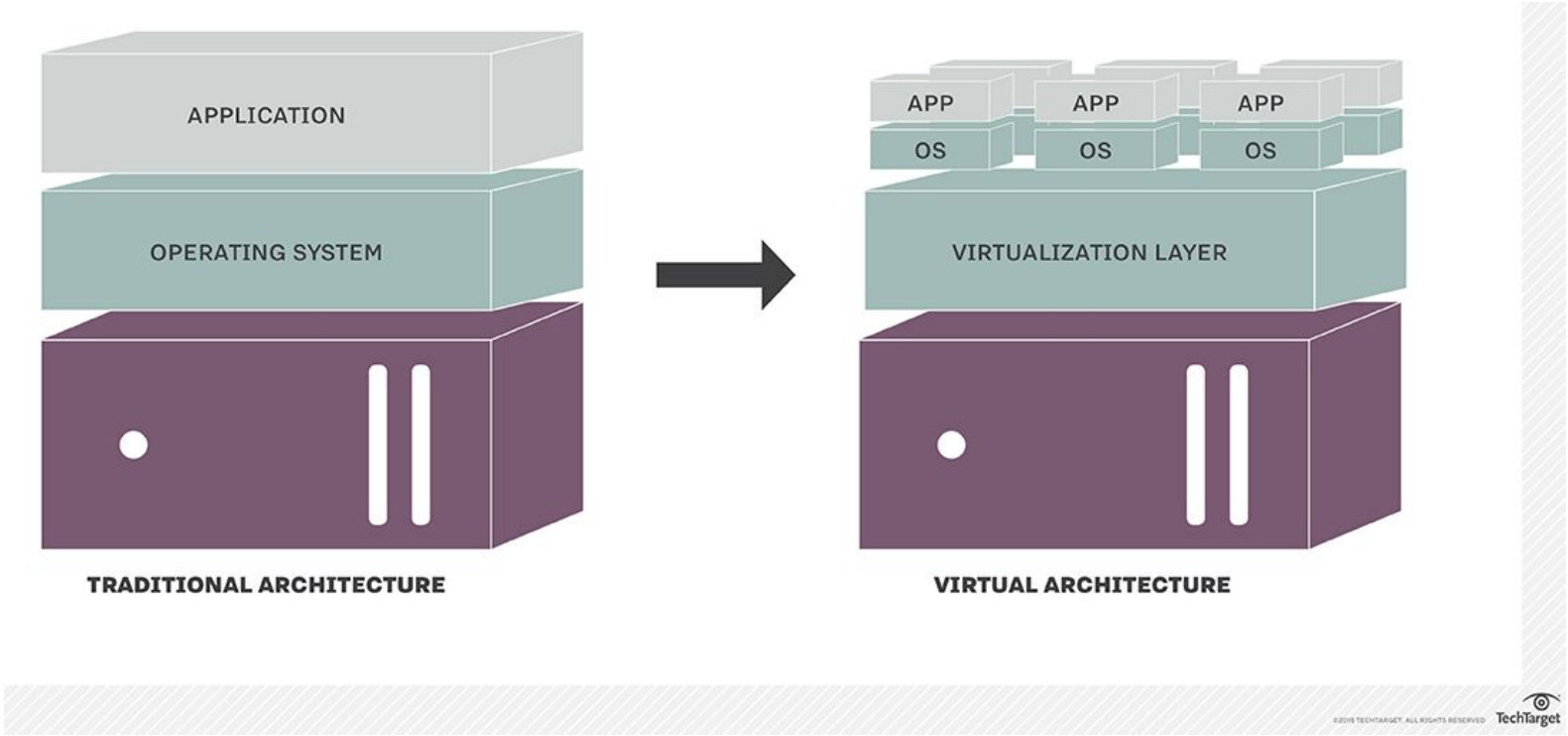
Anurak Theanpurmpul

Agenda

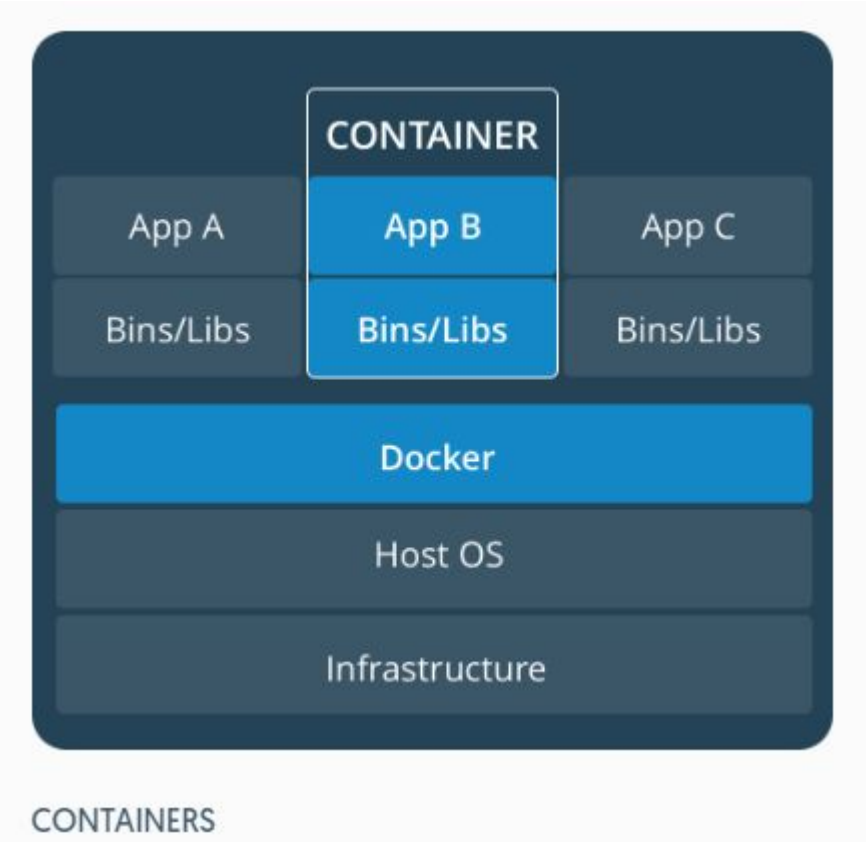
- **Docker Overview**
- **Docker Registry**
- **Docker Images**
- **Docker Volume**
- **Docker Network**
- **Docker Build (Dockerfile)**
- **Multi-state Build**
- **Docker Compose**
- **Docker Swarm Mode**
- **Docker Stack**

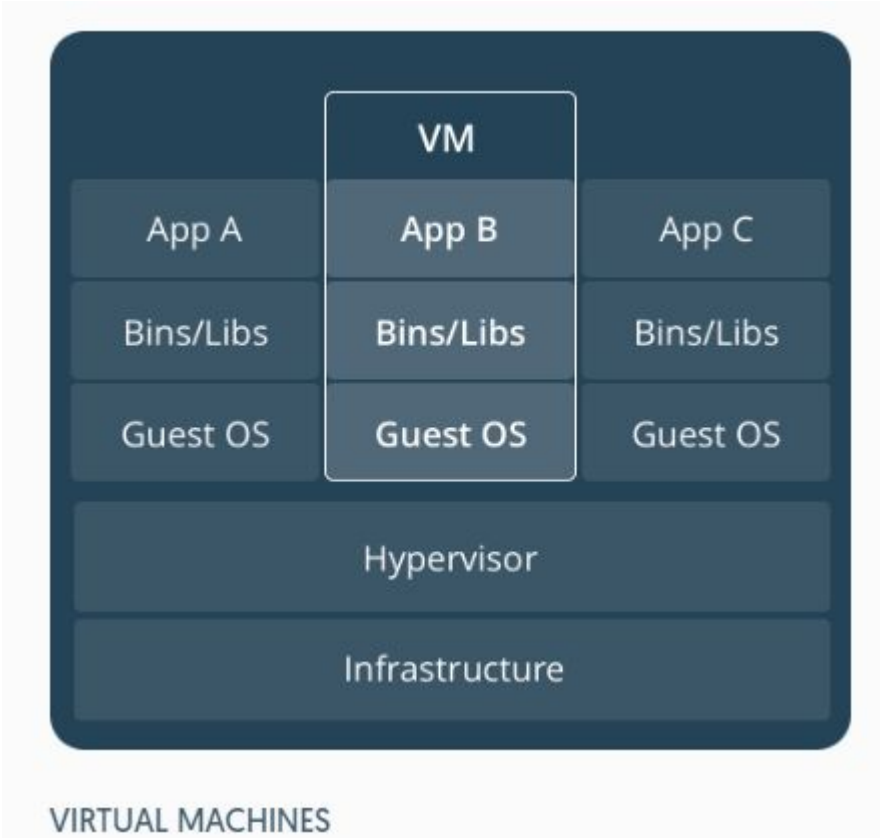
Docker Overview

Traditional and Virtual Architecture



Containers vs VIrtual Machines





- **LXC (LinuX Containers)**

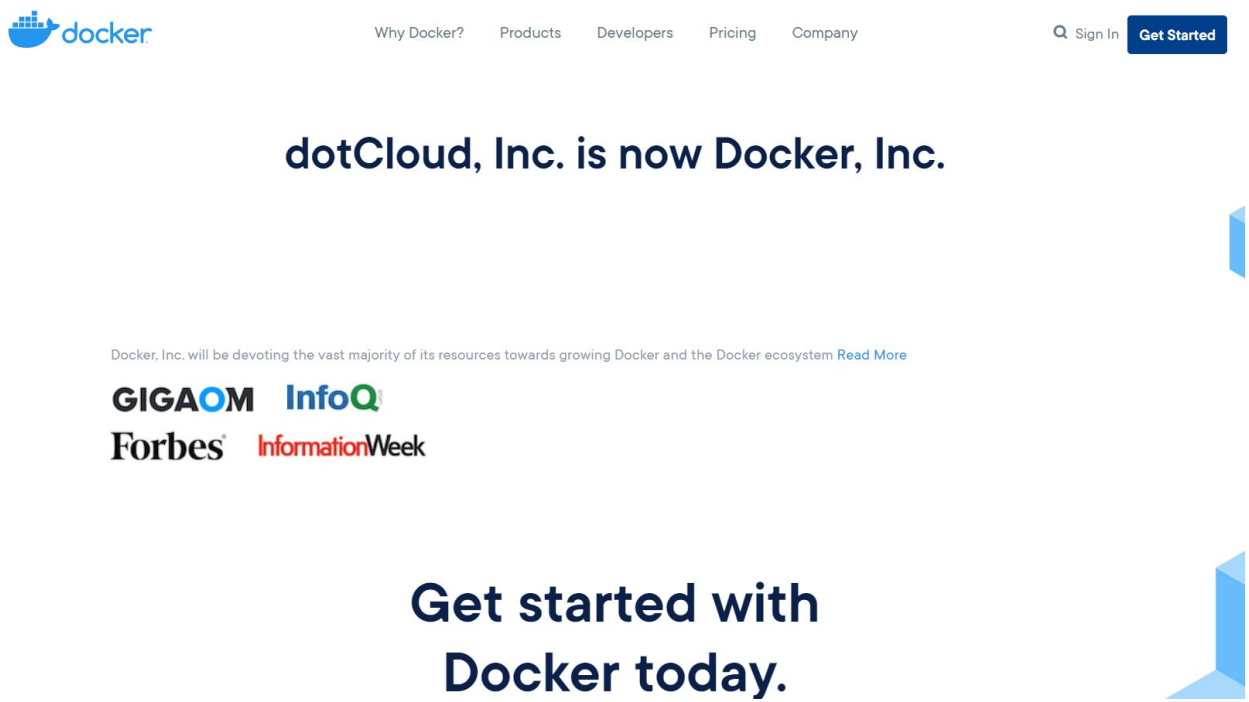
Docker

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications.

http://www.docker.com/

Docker

- Docker was born in 2013 by dotCloud, Inc.



https://www.docker.com/docker-news-and-press/dotcloud-inc-now-docker-inc

Docker Toolbox

- WIndows 7,8 or 10 Home (uses Hyper-V)

Get Docker Toolbox for Windows

Docker Toolbox is for older Mac and Windows systems that do not meet the requirements of Docker for Mac and Docker for Windows and Microsoft Hyper-V.

Docker Desktop

- Docker on Windows 10 Pro/Ent (uses Hyper-V and check for latest updates)

Get Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

You can download and install Docker on multiple platforms. Refer to the following section and choose the best installation path for you.

Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.

Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.

Docker for Linux

Install Docker on a computer which already has a Linux distribution installed.

Docker Desktop WSL 2 backend

- No more Hyper-V, Windows Subsystem for Linux (WSL) 2 introduces a significant architectural change as it is a full Linux kernel built by Microsoft, allowing Linux containers to run natively without emulation.
- https://www.microsoft.com/en-us/software-download/windows10

docs.docker.com/docker-for-windows/wsl/

docker docs | Search the docs | Home | Guides | Product manuals | Reference | Samples

- Install Docker Desktop for Windows
- Install Docker Desktop on Windows Home
- User manual
- Deploy on Kubernetes
- Networking
- Logs and troubleshooting
- FAQs
- Stable release notes
- Edge release notes
- Docker Desktop WSL 2 backend
- Dashboard
- Open source licensing
- Docker Hub
- Open-source projects
- Release notes
- Superseded products and tools

Docker Desktop WSL 2 backend

Estimated reading time: 5 minutes

Windows Subsystem for Linux (WSL) 2 introduces a significant architectural change as it is a full Linux kernel built by Microsoft, allowing Linux containers to run natively without emulation. With Docker Desktop running on WSL 2, users can leverage Linux workspaces and avoid having to maintain both Linux and Windows build scripts. In addition, WSL 2 provides improvements to file system sharing, boot time, and allows access to some cool new features for Docker Desktop users.

Docker Desktop uses the dynamic memory allocation feature in WSL 2 to greatly improve the resource consumption. This means, Docker Desktop only uses the required amount of CPU and memory resources it needs, while enabling CPU and memory-intensive tasks such as building a container to run much faster.

Additionally, with WSL 2, the time required to start a Docker daemon after a cold start is significantly faster. It takes less than 10 seconds to start the Docker daemon when compared to almost a minute in the previous version of Docker Desktop.

Prerequisites

Before you install the Docker Desktop WSL 2 backend, you must complete the following steps:

1. Install Windows 10, version 1903 or higher.
2. Enable WSL 2 feature on Windows. For detailed instructions, refer to the Microsoft documentation.
3. Download and install the Linux kernel update package.

Best practices

- To get the best out of the file system performance when bind-mounting files, we recommend storing source code and other data that is bind-mounted into Linux containers (i.e., with `docker run -v <host-path>:<container-path>`) in the Linux file system, rather than the

Edit this page

Request docs changes

On this page:

- Prerequisites
- Best practices
- Download
- Install
- Develop with Docker and WSL 2
- Feedback

https://docs.docker.com/docker-for-windows/wsl/

Try It!

Basic Docker Commands

- *docker ps*
- *docker ps -a*
- *docker images*
- *docker run --name some-nginx -p 8080:80 -d nginx*
- *docker ps*

```
❯ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
   NAMES
79ed823b6a79        nginx               "/docker-entrypoint.…"   10 minutes ago      Up 10 minutes       0.0.0.0:8080->80/tcp
   some-nginx
```

- Open browser and try http://localhost>:8080
- *docker images*

Basic Docker Commands

```
Commands:
    attach    Attach 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
    diff      Inspect changes 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 a container, image or task
    kill      Kill one or more running container
    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
    network   Manage Docker networks
    node      Manage Docker Swarm nodes
    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 a container
    rm        Remove one or more containers
    rmi       Remove one or more images
    run       Run a command in a new container
    save      Save one or more images to a tar archive (streamed to STDOUT by de
fault)
    search    Search the Docker Hub for images
    service   Manage Docker services
    start     Start one or more stopped containers
    stats     Display a live stream of container(s) resource usage statistics
    stop      Stop one or more running containers
    swarm     Manage Docker Swarm
    tag       Tag an image into a repository
    top       Display the running processes of a container
    unpause   Unpause all processes within one or more containers
    update    Update configuration of one or more containers
    version   Show the Docker version information
    volume    Manage Docker volumes
    wait      Block until a container stops, then print its exit code
```

- docker ps
- docker ps -a
- docker run
- docker start
- docker stop
- docker logs
- docker pull
- docker images
- docker rm
- docker rmi
- docker info
- docker inspect

Basic Docker Commands (con’t)

- *docker images*
- *docker logs <containerId>*
- *docker logs -f <containerId>*
- *docker stop <containerId>*
- *docker ps -a*
- *docker start <containerId>*
- *docker inspect <containerId>*
- **Limit a container's resources** <https://docs.docker.com/engine/admin/resource_constraints/>

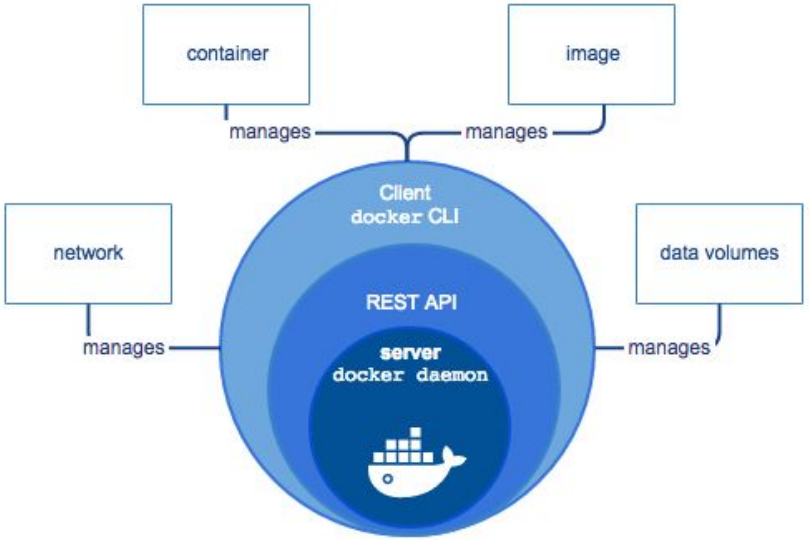
Note:

- Can use **container name** instead of ***containerId***
- *docker <command> --help*
- *Docker <command> <subcommand> --help*

Docker Engine

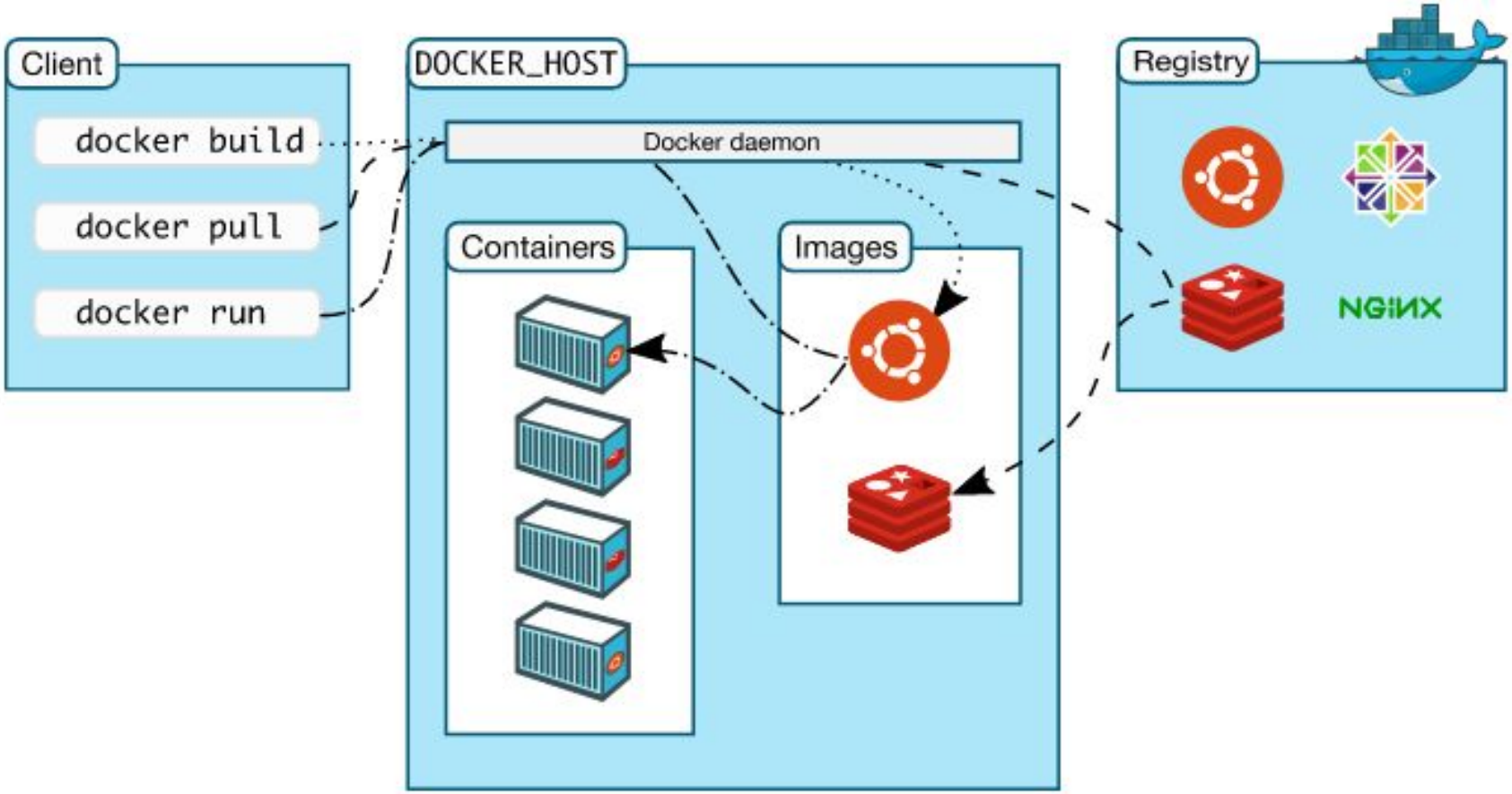
Docker Engine is a **client-server application** with these major components:

- A server which is a type of long-running program called **a daemon process** (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).
- *curl --unix-socket /var/run/docker.sock http://localhost/v1.40/containers/json*

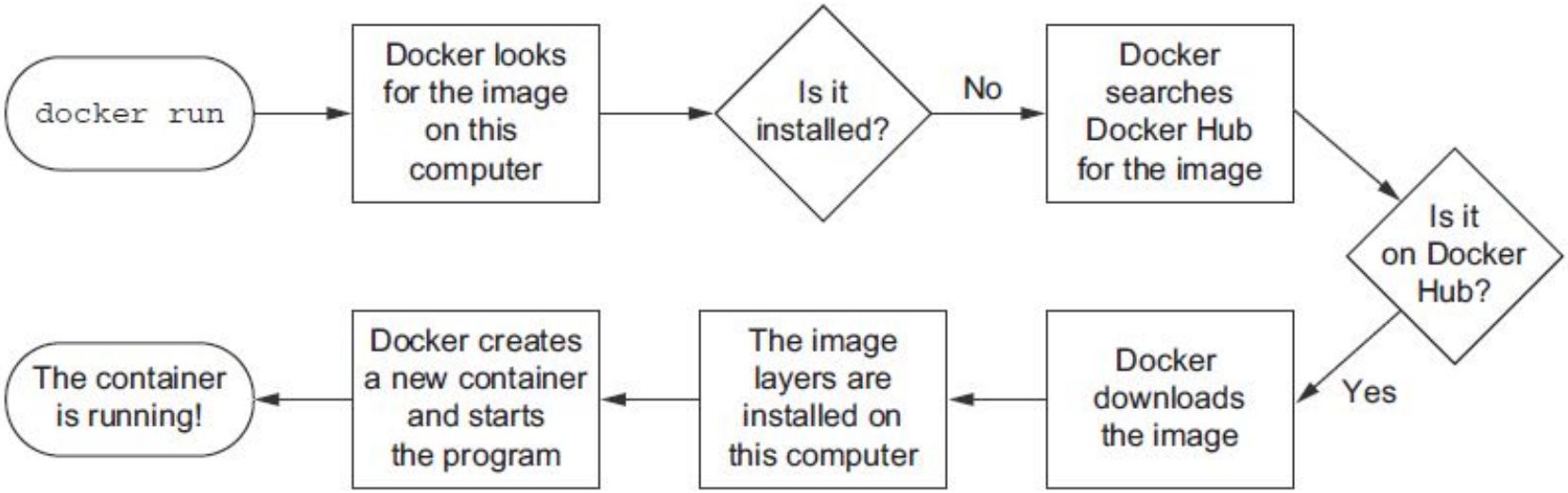


https://docs.docker.com/engine/api/v1.40/

Docker Architecture



Flow of docker run command



Inside Docker Desktop

Mac

- *screen ~/Library/Containers/com.docker.docker/Data/vms/0/tty*
- *cd /var/lib/docker*

Windows

- *docker run -it --privileged --pid=host debian nsenter -t 1 -m -u -i sh*
- *cd /var/lib/docker*

Container List

```
✔  docker ps
CONTAINER ID        IMAGE                                   COMMAND                  CREATED             STATUS              PORTS
                                                    NAMES
6fb1c7b7bd84        portainer/portainer                         "/portainer"             About an hour ago   Up About an hour    0.0.0.0:9001->9000/tcp
                                                    portainer
9adb7c194b0e        docker.elastic.co/kibana/kibana:6.3.2   "/usr/local/bin/kiba…"   11 days ago         Up 11 days          0.0.0.0:5601->5601/tcp
                                                    kibana
639718c9aa3e        bitnami/redis-sentinel:latest               "/opt/bitnami/script…"   3 months ago        Up 3 weeks          0.0.0.0:26380->26379/tcp
                                                    bitnami_redis-sentinel_2
fd00c9463a41        bitnami/redis-sentinel:latest               "/opt/bitnami/script…"   3 months ago        Up 3 weeks          0.0.0.0:26381->26379/tcp
                                                    bitnami_redis-sentinel_3
76bcc50ac645        bitnami/redis-sentinel:latest               "/opt/bitnami/script…"   3 months ago        Up 3 weeks          0.0.0.0:26379->26379/tcp
                                                    bitnami_redis-sentinel_1
b5c69f162304        bitnami/redis:latest                        "/opt/bitnami/script…"   3 months ago        Up 3 weeks          0.0.0.0:32769->6379/tcp
                                                    bitnami_redis-slave_1
99303b866e8c        bitnami/redis:latest                        "/opt/bitnami/script…"   3 months ago        Up 3 weeks          0.0.0.0:32768->6379/tcp
                                                    bitnami_redis_1
e7cd519a70a9        prom/prometheus:v2.20.0                     "/bin/prometheus --c…"   3 months ago        Up 3 weeks          9090/tcp
                                                    prometheus
7361c250a9ea        prom/alertmanager:v0.21.0                   "/bin/alertmanager -…"   3 months ago        Up 3 weeks          9093/tcp
                                                    alertmanager
304fe3414086        gcr.io/cadvisor/cadvisor:v0.37.0            "/usr/bin/cadvisor -…"   3 months ago        Up 3 weeks (healthy)   8080/tcp
                                                    cadvisor
2adcef6ed50e        stefanprodan/caddy                          "/sbin/tini -- caddy…"   3 months ago        Up 3 weeks          0.0.0.0:3000->3000/tcp, 0.0
.0.0:9090-9091->9090-9091/tcp, 0.0.0.0:9093->9093/tcp   caddy
2f83908fa48e        grafana/grafana:7.1.1                       "/run.sh"                3 months ago        Up 3 weeks          3000/tcp
                                                    grafana
5a63958e75df        prom/node-exporter:v1.0.1                   "/bin/node_exporter …"   3 months ago        Up 3 weeks          9100/tcp
                                                    nodeexporter
65e9e2ec7ffe        prom/pushgateway:v1.2.0                     "/bin/pushgateway"       3 months ago        Up 3 weeks          9091/tcp
                                                    pushgateway
c21d70f731d5        elkozmon/zoonavigator:latest                "./run.sh"               4 months ago        Up 3 weeks (healthy)   0.0.0.0:9000->9000/tcp
                                                    zoonavigator
74c1f48fdc44        anutech2001/reader                          "/usr/lib/jvm/java-8…"   5 months ago        Up 3 weeks          0.0.0.0:8010->8010/tcp
                                                    training_reader_1
d4d277e37279        anutech2001/bookstore                       "/usr/lib/jvm/java-8…"   5 months ago        Up 3 weeks          0.0.0.0:8011->8011/tcp, 888
8/tcp                                               training_bookstore_1
48743fb7b571        openzipkin/zipkin                           "/busybox/sh run.sh"     5 months ago        Up 3 weeks          9410/tcp, 0.0.0.0:9411->941
1/tcp                                               training_zipkin_1
```

Container Process List

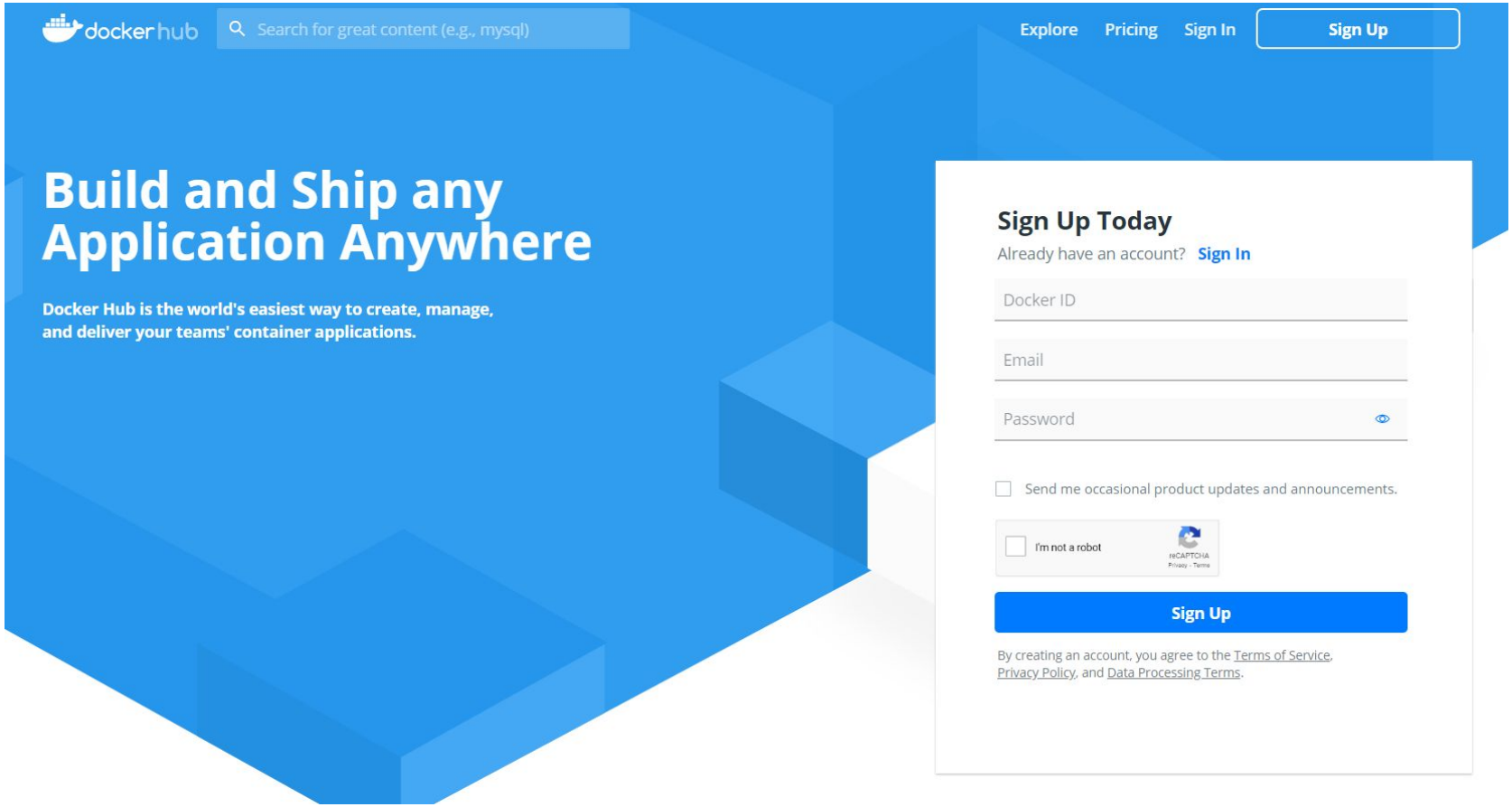
- *$ top*

```
Mem: 7725500K used, 438752K free, 1696K shrd, 509312K buff, 3157616K cached
CPU:   1% usr   3% sys   0% nic  94% idle   0% io   0% irq   0% sirq
Load average: 0.51 0.53 0.62 1/1231 55022
  PID  PPID USER     STAT   VSZ %VSZ CPU %CPU COMMAND
 9177  7901 root     S     205m   3%   7   3% /usr/bin/cadvisor -logtostderr
 1640  1632 root     S    2297m  28%   4   0% /usr/local/bin/dockerd -H unix:///
 8958  7714 65534    S     702m   9%   6   0% /bin/node_exporter --path.procfs=/
 7514  7369 1000     S    6412m  78%   1   0% /usr/local/openjdk-11/bin/java -Du
 7868  7627 65534    S    1438m  18%   5   0% /bin/prometheus --config.file=/etc
 1654  1640 root     S    2033m  25%   3   0% containerd --config /var/run/docke
 1090     1 root     S     142m   2%   1   0% /usr/bin/containerd
  484     1 root     S     104m   1%   5   0% /usr/bin/memlogd -fd-log 3 -fd-que
 8855  7686 1001     S    52792   1%   0   0% redis-sentinel 0.0.0.0:26379 [sent
 7199  6986 1001     S    52792   1%   1   0% redis-sentinel 0.0.0.0:26379 [sent
 7217  7084 1001     S    52792   1%   6   0% redis-sentinel 0.0.0.0:26379 [sent
 7369  1654 root     S     106m   1%   5   0% containerd-shim -namespace moby -w
 7169  6844 472      S     740m   9%   6   0% grafana-server --homepath=/usr/sha
 7302  7124 1001     S     138m   2%   3   0% redis-server 0.0.0.0:6379
 7901  1654 root     S     106m   1%   7   0% containerd-shim -namespace moby -w
 7473  7276 root     S    5594m  68%   0   0% {java} /usr/lib/jvm/java-8-openjdk
 7182  6994 root     S    5458m  67%   2
```

Docker Registry

Docker Hub (Public Registry)

Docker Hub (https://hub.docker.com/) provides a free-to-use, hosted Registry, plus additional features (organization accounts, automated builds, and more).



Try It!

Private Docker Registry

For faster access and shared an images, create our private registry.

- Run your private registry
  docker run -d -p 5000:5000 --restart=always --name registry registry:2
- Pull some image from the Docker Hub
  docker pull hello-world
- Tag the image so that it points to your registry
  docker tag hello-world localhost:5000/myhelloworld
- Push it
  docker push localhost:5000/myhelloworld
- Pull it back (from another docker machine)
  docker pull localhost:5000/myhelloworld
- List Catalog
  http://<registryHost>:5000/v2/_catalog
- See Tags of an Image, for example from "*myhelloworld*" image
  http://localhost:5000/v2/myhelloworld/tags/list

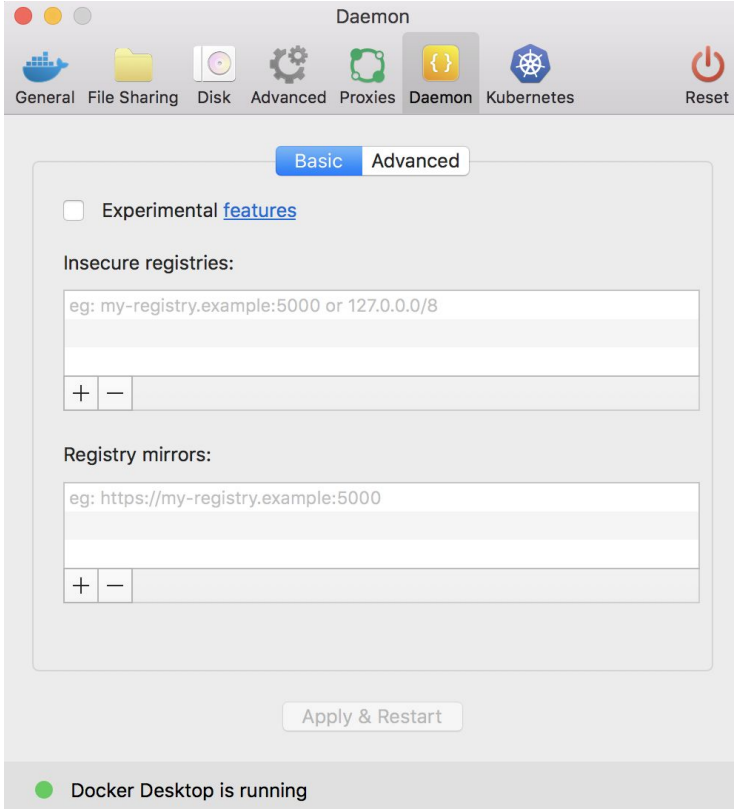
Try It!

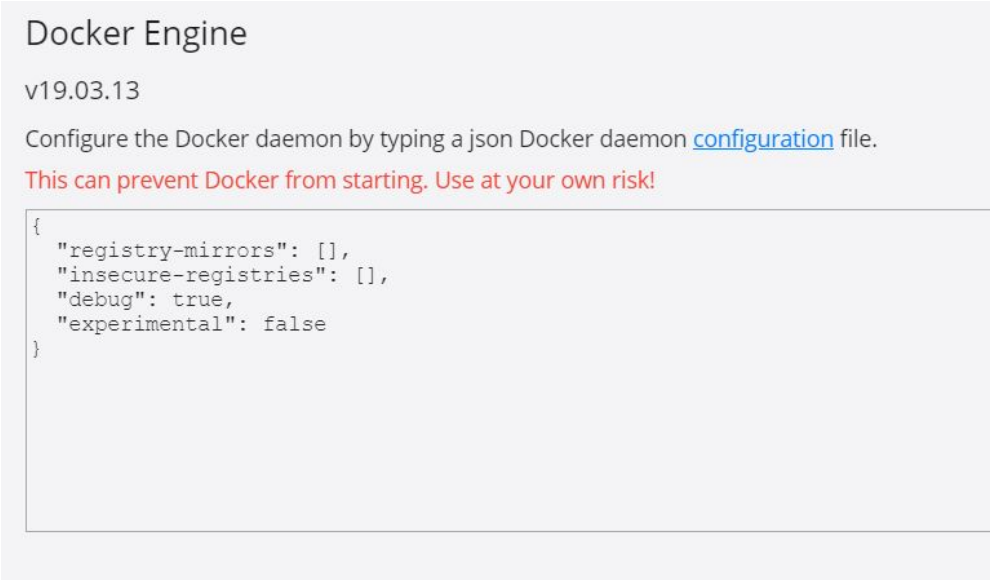
Upload Image to Docker Hub

- Step to Upload Image to Docker Hub
  - Register Free Docker Hub accounts (https://hub.docker.com/)
  - *docker tag <image name>:<version> <docker hub user>/<image name>:<version>*
  - *docker login –u <docker hub user> –p <docker hub password>*
  - *docker push <docker hub user>/<image>:<version>*
- Example
  - *docker tag hello-world <docker hub user>/myhello-world*
  - *docker login –u <docker hub user>*
  - *docker push <docker hub user>/myhello-world*
  - Delete image

    docker rmi <image name>

Setting External Private Registry





Advantage of Private Registry

- Speed
- Security
- Reliability
- Regulation
- Cost

JFrog ARTIFACTORY

Azure Container Registry

aws Amazon ECR

Google Container Registry

HARBOR

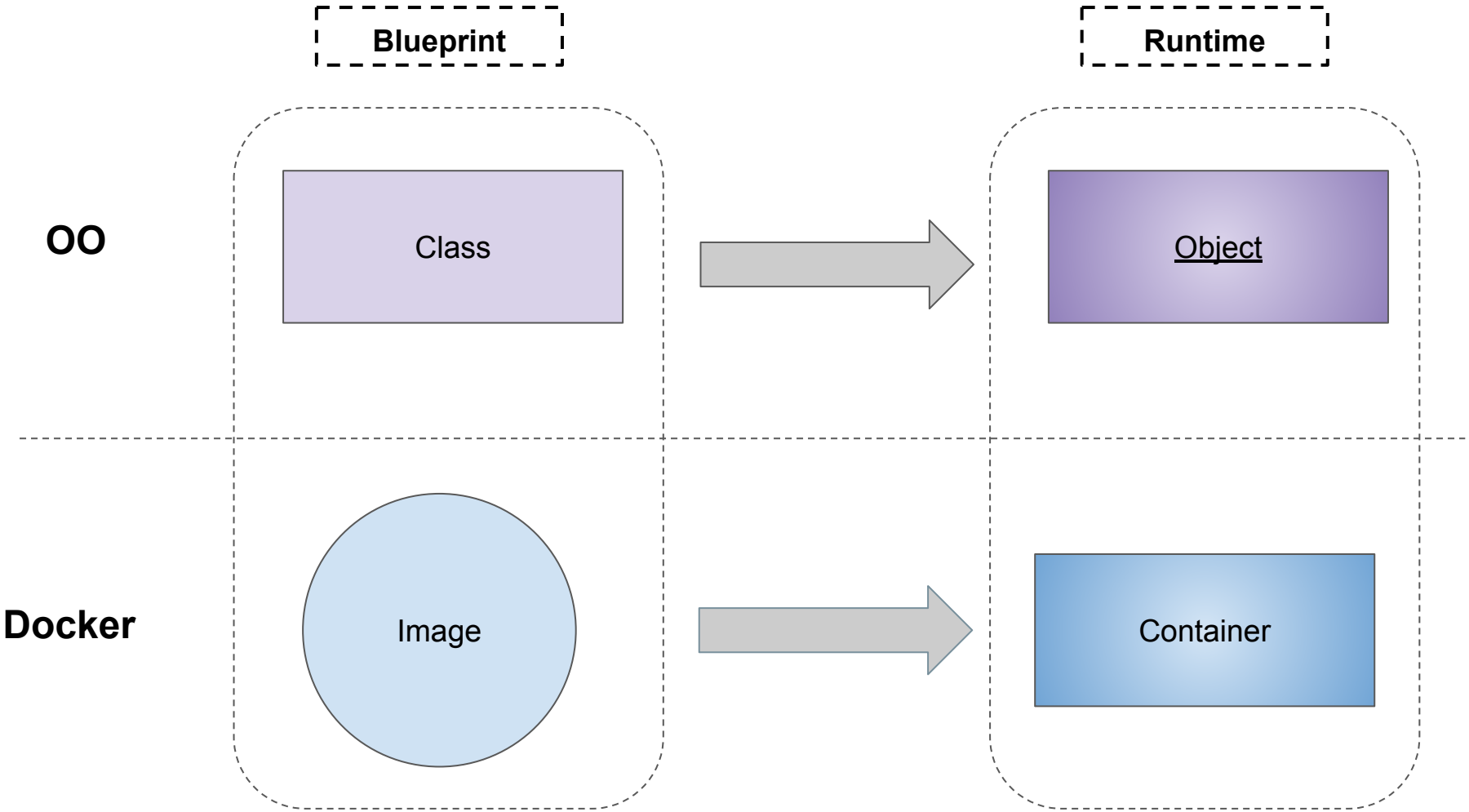
QUAY

Gitlab Container Registry

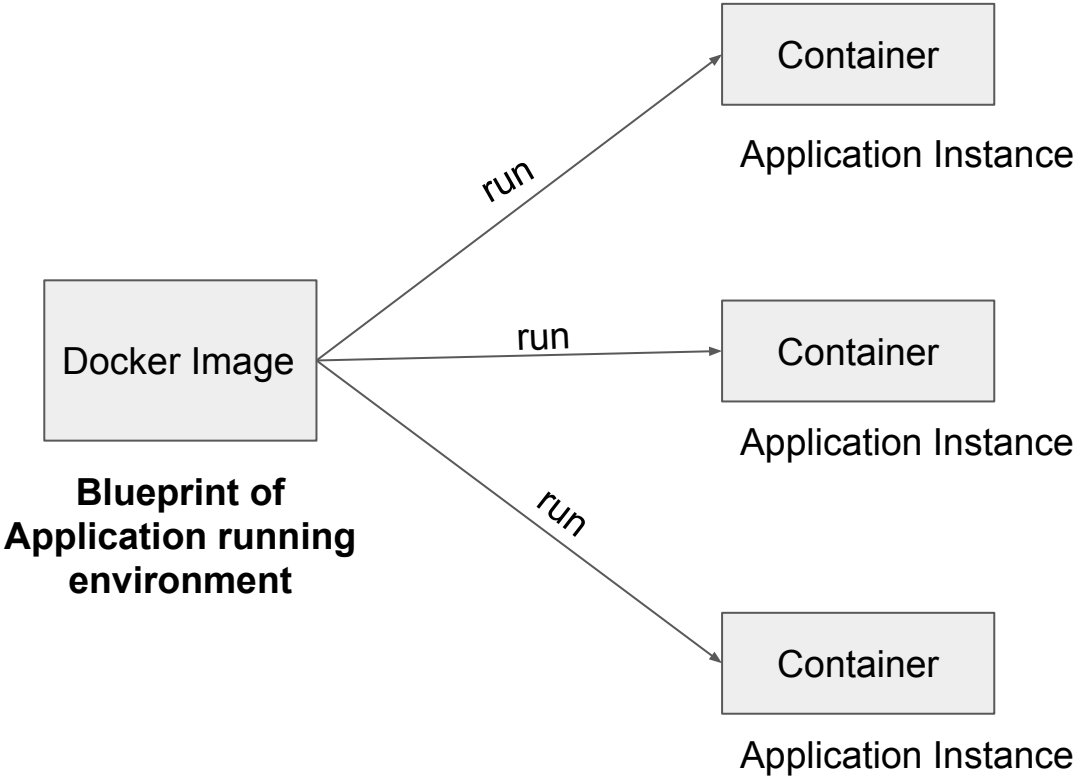
opensuse/portus

Docker Images

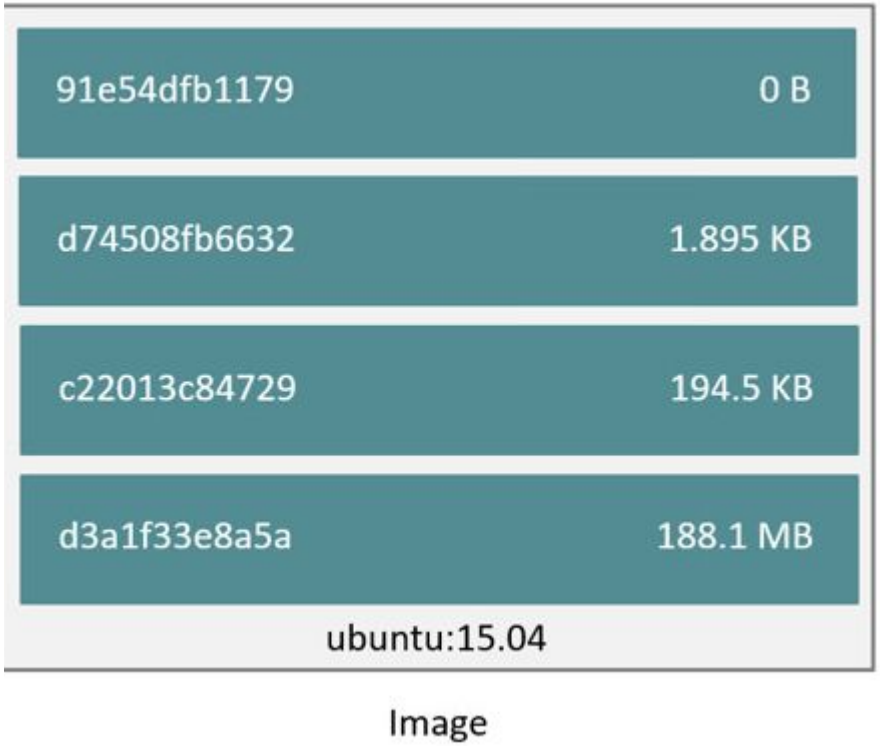
Compare with Object-Oriented



Docker Image



Docker Images and Layers



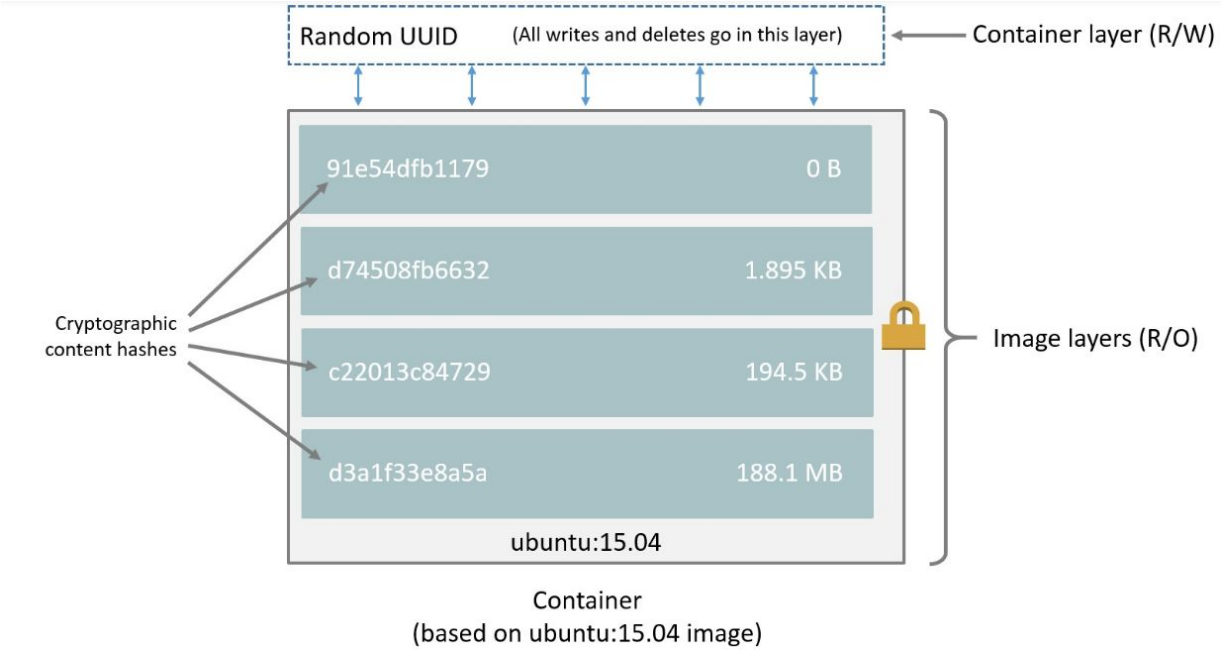
- Each Docker image references a **list of read-only layers** that represent filesystem differences.
- The **Docker storage driver** is responsible for stacking these layers and providing a single unified view.

Container’s Key Technologies

Two key technologies behind Docker image and container management.

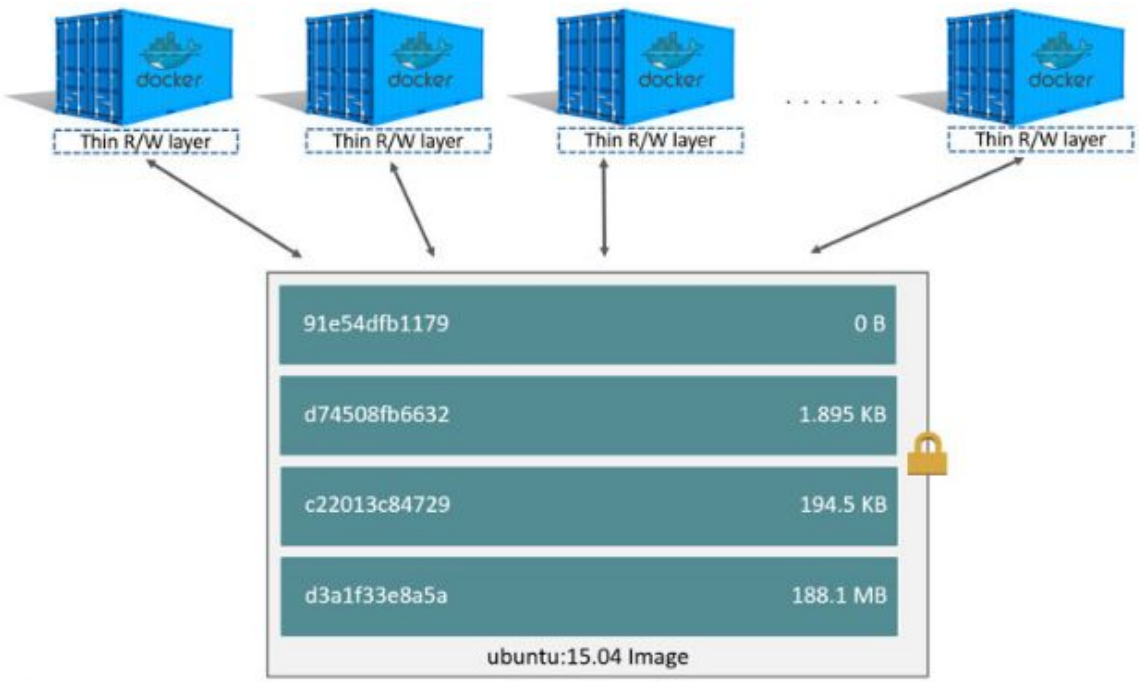
- ***Stackable image layers***
- ***The copy-on-write (CoW) strategy***:
  - System processes that need the same data share the same instance of that data.
  - If one process needs to modify or write to the data, only then does the operating system make a copy of the data for that process to use and only the process that needs to write has access to the data copy.
- **These strategies optimizes both image disk space usage and the performance of container start times**

Container and Layers



- New R/W layer was created when the container was created.
- All changes are written to this layer.
- When the container is deleted the writable layer is also deleted.
- Since Docker 1.10

Container and Layers (cont')



- Multiple containers can share access to the same underlying image and yet have their own data state.
- Sharing is a good way to optimize resources.

Dockerfile of jdk:8

```
FROM buildpack-deps:jessie-scm

RUN apt-get update && apt-get install -y --no-install-recommends \
        bzip2 \
        unzip \
        xz-utils \
    && rm -rf /var/lib/apt/lists/*
RUN echo 'deb http://deb.debian.org/debian jessie-backports main' > /etc/apt/sources.list.d/jessie-backports.list
ENV LANG C.UTF-8
RUN { \
        echo '#!/bin/sh'; \
        echo 'set -e'; \
        echo; \
        echo 'dirname "$(dirname "$(readlink -f "$(which javac || which java)")")"'; \
    } > /usr/local/bin/docker-java-home \
    && chmod +x /usr/local/bin/docker-java-home

ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64

ENV JAVA_VERSION 8u111
ENV JAVA_DEBIAN_VERSION 8u111-b14-2~bpo8+1
ENV CA_CERTIFICATES_JAVA_VERSION 20140324

RUN set -x \
    && apt-get update \
    && apt-get install -y \
        openjdk-8-jdk="$JAVA_DEBIAN_VERSION" \
        ca-certificates-java="$CA_CERTIFICATES_JAVA_VERSION" \
    && rm -rf /var/lib/apt/lists/* \
    && [ "$JAVA_HOME" = "$(docker-java-home)" ]
RUN /var/lib/dpkg/info/ca-certificates-java.postinst configure
```

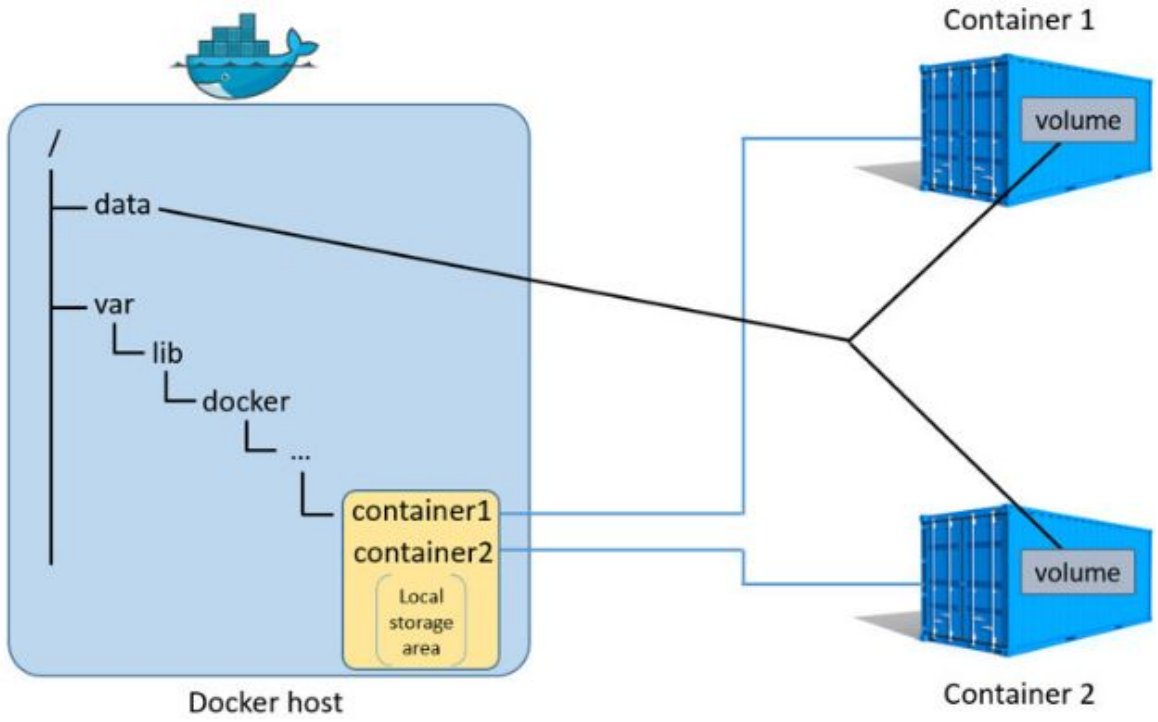
Docker Image of java:8

Image
192.168.99.117:5000/java:8

Image	Cmd	Size
54970924248	/var/lib/dpkg/info/ca-certificates-java.postinst configure	285 KB
71b6ed1bb5e	/bin/sh -c set -x && apt-get update && apt-get install -y openjdk-8-jdk="$JAVA_DEBIAN_VERSION" ca-certificates-java="$CA_CERTIFICATES_JAVA_VERSION" && rm -rf /var/lib/apt/lists/* && ["$JAVA_HOME" = "$(docker-java-home)"]	131.6 MB
c2e26dad9eb	/bin/sh -c #(nop) ENV CA_CERTIFICATES_JAVA_VERSION=20140324	0 B
3305d7e2c8a	/bin/sh -c #(nop) ENV JAVA_DEBIAN_VERSION=8u102-b14.1-1~bpo8+1	0 B
a001ffeb3ab	/bin/sh -c #(nop) ENV JAVA_VERSION=8u102	0 B
a76495019d4	/bin/sh -c #(nop) ENV JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64	0 B
031d51a8c22	/bin/sh -c { echo '#!/bin/sh'; echo 'set -e'; echo; echo 'dirname "$(dirname "$(readlink -f "$(which javac || which java)")")"'; } > /usr/local/bin/docker-java-home && chmod +x /usr/local/bin/docker-java-home	240 B
a43d5ea7d36	/bin/sh -c #(nop) ENV LANG=C.UTF-8	0 B
6141a4dc4ed	/bin/sh -c echo 'deb http://httpredir.debian.org/debian jessie-backports main' > /etc/apt/sources.list.d/jessie-backports.list	218 B
eecb52a8466	/bin/sh -c apt-get update && apt-get install -y --no-install-recommends bzip2 unzip xz-utils && rm -rf /var/lib/apt/lists/*	599 KB
5d459c46dd1	/bin/sh -c apt-get update && apt-get install -y --no-install-recommends bzr git mercurial openssh-client subversion procps && rm -rf /var/lib/apt/lists/*	43.2 MB
488a5a26b89	/bin/sh -c apt-get update && apt-get install -y --no-install-recommends ca-certificates curl wget && rm -rf /var/lib/apt/lists/*	19.2 MB
51dd77e0947	/bin/sh -c #(nop) CMD ["/bin/bash"]	0 B
57f55a8a684	/bin/sh -c #(nop) ADD file:f2453b914e7e026efd39c6321c7b14509b6d09dd3cf5567a8f6bd38466e06954 in /	52.5 MB

Total size: 247.3 MB

The storage driver and Data volumes



- A single Docker host running two containers.
- Each container own address space within the Docker host's local storage area *(/var/lib/docker/...)*
- A single shared data volume located at */data* on the Docker host

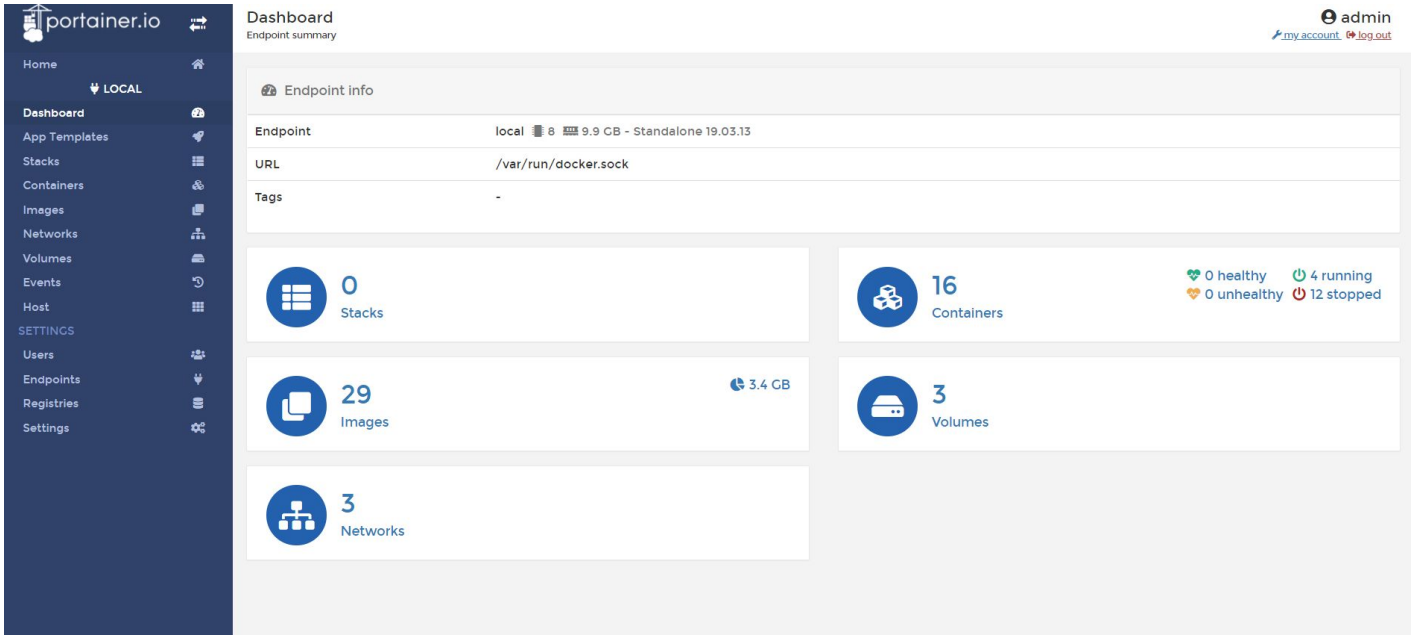
Portainer

An open-source toolset that allows you to easily build and manage containers in Docker, Swarm, Kubernetes and Azure ACI.

```
$ docker volume create portainer_data
```

Win: docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce

Mac: docker run -d -p 9001:9000 --name portainer --restart always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer

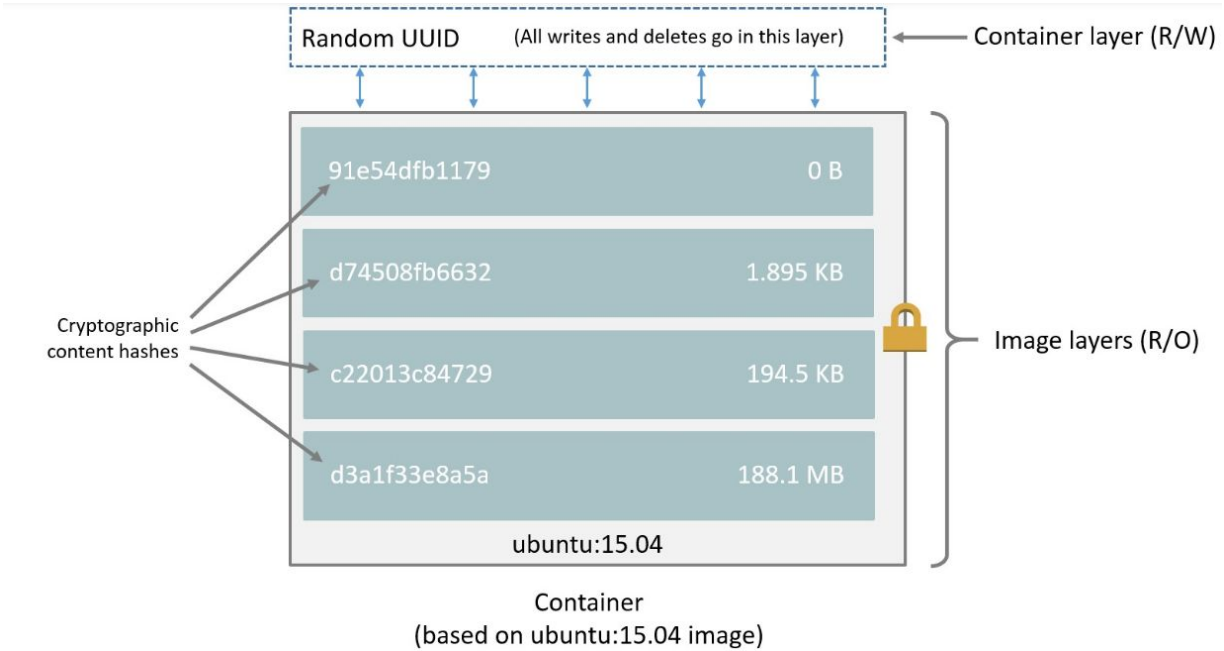


Docker Volume

Manage data in Docker

By default all files created inside a container are **stored on a writable container layer.**

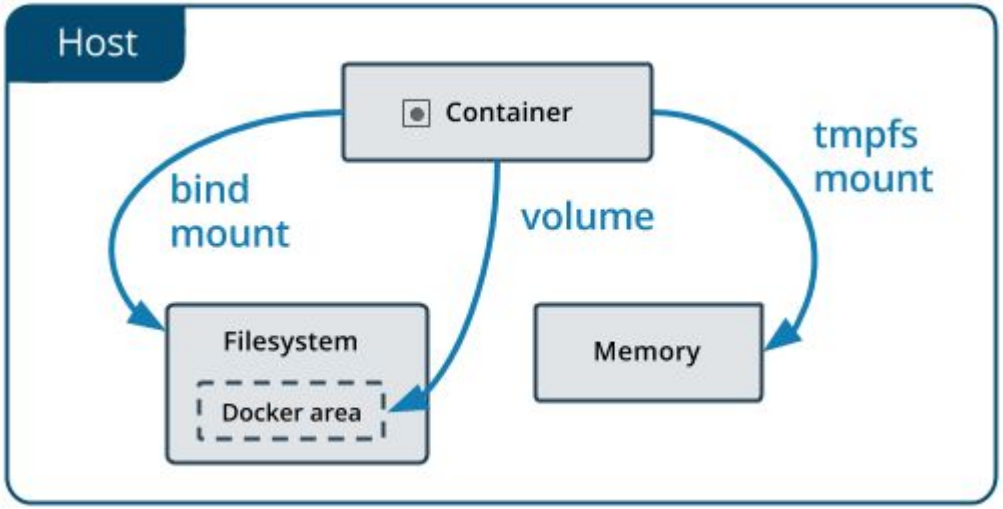
- The data doesn't persist when that container no longer exists.
- This extra abstraction reduces performance.



Different Types of Manage Data

Two options for containers to store files in the host machine.

- **Volumes** are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/)
- **Bind mounts** may be stored anywhere on the host system.



Note: No matter which type of mount you choose to use, **the data looks the same from within the container.**

Docker run with Bind mounts

- *docker run --name some-nginx-v -v /c/Users/<user>/nginx/:/usr/share/nginx/html:ro -p 8080:80 -d nginx*
- Locate a volume
  - ***docker inspect***


```
"Mounts": [
    {
        "Type": "volume",
        "Name": "1fb394327342b55036eea22e3b24c2e4a45a352ba673ab0696af01a103c274ba",
        "Source": "/var/lib/docker/volumes/1fb394327342b55036eea22e3b24c2e4a45a352ba673ab0696af01a103c274ba/_data",
        "Destination": "/tmp/a",
        "Driver": "local",
        "Mode": "",
        "RW": true,
        "Propagation": ""
    }
],
```


- https://docs.docker.com/storage/volumes/

Docker run with Volume (anonymous volume and named volume)

Create a volume explicitly

docker volume create awesome

*docker run --rm -v **/foo** -v **awesome:/bar** busybox top*

Or just:

docker run --rm -v /foo -v awesome:/bar busybox top

Inspect Volume

docker volume inspect <volume name>

Remove volumes

docker volume prune

docker volume rm <volume name>

Good use cases for volumes and Bind mounts

Volumes

- Sharing data among multiple running containers.
- When you want to store your container's data on a remote host or a cloud provider, rather than locally.
- When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice.
- When your application requires high-performance I/O on Docker Desktop.

Bind mounts

- Sharing configuration files from the host machine to containers.
- Sharing source code or build artifacts between a development environment on the Docker host and a container.

Advanced Topics

Use a volume driver

Backup, restore, or migrate data volumes

Docker storage drivers

Docker Network

Network driver summary

- **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 Networks

- A software bridge which allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network.
- Bridge networks apply to containers running on the same Docker daemon host
- When you start Docker, a default bridge network (also called bridge) is created automatically
- The default network driver. If you don’t specify a driver.
- Basic network commands
    - docker network ls
    - docker network inspect <network name>
    - docker network create --driver <driver type> <network name>
    - docker network rm <network name>

  Noted: driver type can be “bridge”, “host”, “overlay”, “macvlan”, or “none”.

Bridge Network Examples

$ docker run -dit --name alpine1 alpine ash

$ docker run -dit --name alpine2 alpine ash

$ docker network inspect bridge - note ip of alpine2

$ docker attach alpine1

$ ping -c 2 <alpine ip>

$ ping -c 2 alpine2

What happened in the last command?

User-defined Bridge Networks

- Try these commands
  - $ docker network create my-net
  - $ docker run -dit **--network my-net** --name alpine1 alpine ash
  - $ docker run -dit **--network my-net** --name alpine2 alpine ash
  - $ docker network inspect **my-net** --- noted an ip of alpine2
  - $ docker attach alpine1
  - $ ping -c 2 <alpine ip>
  - $ ping -c 2 alpine2
- What happened in the last command?
- $ *docker network rm my-net*

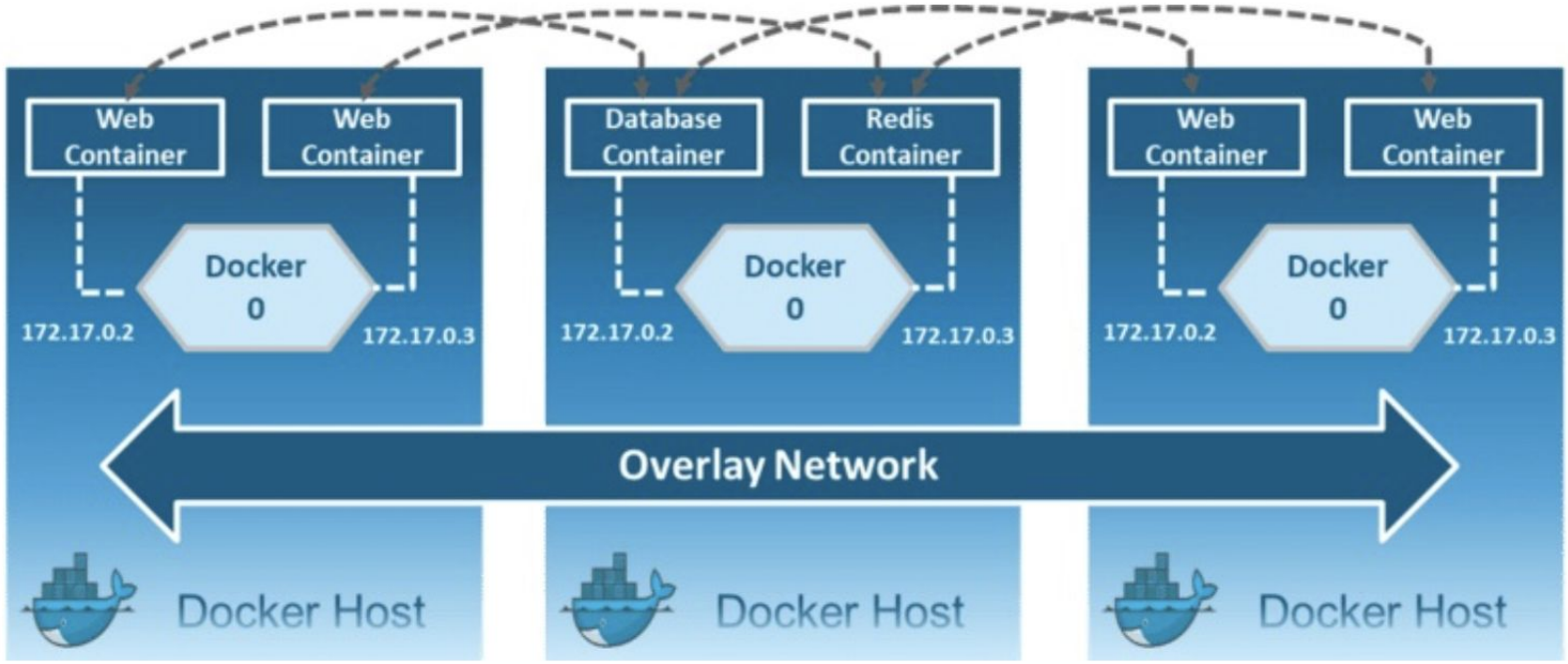
- *$ docker rm $(docker ps -a -q -f status=exited)*

User-defined Bridge Networks vs Default Bridge

- Differences Between User-defined Bridges and The Default Bridge
  - User-defined bridges provide automatic DNS resolution between containers.
  - User-defined bridges provide better isolation.
  - Containers can be attached and detached from user-defined networks on the fly.
  - Each user-defined network creates a configurable bridge.
  - Linked containers on the default bridge network share environment variables.
- https://docs.docker.com/network/bridge/#differences-between-user-defined-bridges-and-the-default-bridge
- https://hub.docker.com/_/busybox

Overlay Networks

- The overlay network driver creates a distributed network among multiple Docker daemon hosts.
- Creates an internal private network that spans across all the nodes participating in the swarm cluster.
- Overlay networks facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker Daemons.



Source: https://borosan.gitbook.io/docker-handbook/docker-networking

Create Overlay Network

- Create an overlay network

  $ *docker network create -d overlay my-overlay*

- To create an overlay network which can be used by swarm services or standalone containers to communicate with other standalone containers running on other Docker daemons, add the *--attachable* flag:

  $ *docker network create -d overlay --attachable my-attachable-overlay*

- You can specify the IP address range, subnet, gateway, and other options. See docker network create --help for details.
- To encrypt application data as well, add *--opt encrypted* when creating the overlay network. This enables IPSEC encryption at the level of the vxlan.**Overlay network encryption is not supported on Windows nodes.**

  $ *docker network create --opt encrypted --driver overlay --attachable my-attachable-multi-host-network*

Docker Cheat Sheet

dockerlux.github.io
@gcuisinier

General Usage

Start a container in background
```
$> docker run -d jenkins
```

Start an interactive container
```
$> docker run -it ubuntu bash
```

Start a container automatically removed on stop
```
$> docker run --rm ubuntu bash
```

Export port from a container
```
$> docker run -p 80:80 -d nginx
```

Start a named container
```
$> docker run --name mydb redis
```

Restart a stopped container
```
$> docker start mydb
```

Stop a container
```
$> docker stop mydb
```

Add metadata to container
```
$> docker run -d \
label=traefik.backend=jenkins jenkins
```

Build Images

Build an image from Dockerfile in current directory
```
$> docker build --tag myimage .
```

Force rebuild of Docker image
```
$> docker build --no-cache .
```

Convert a container to image
```
$> docker commit c7337 myimage
```

Remove all unused images
```
$> docker rmi $(docker images \
-q -f "dangling=true"
```

Volumes

Create a local volume
```
$> docker volume create --name myvol
```

Mounting a volume on container start
```
$> docker run -v myvol:/data redis
```

Destroy a volume
```
$> docker volume rm myvol
```

List volumes
```
$> docker volume ls
```

Debug

Run another process in running container
```
$> docker exec -it c7337 bash
```

Show live logs of running daemon container
```
$> docker logs -f c7337
```

Show exposed ports of a container
```
$> docker port c7337
```

Create a local network
```
$> docker network create mynet
```

Attach a container to a network on start
```
$> docker run -d --net mynet redis
```

Connect a running container from a network
```
$> docker network connect mynet c7337
```

Disconnect container to a network
```
$> docker network disconnect mynet c7337
```

Manage Containers

List running containers
```
$> docker ps
```

List all containers (running & stopped)
```
$> docker ps -a
```

Inspect containers metadatas
```
$> docker inspect c7337
```

List local available images
```
$> docker images
```

Delete all stopped containers
```
$> docker rm $(docker ps --filter status=exited -q)
```

List all containers with a specific label
```
$> docker ps --filter label=traefik.backend
```

Query a specific metadata of a running container
```
$> docker inspect -f '{{ .NetworkSettings.IPAddress }}' c7337
```

Legend

Image name
```
redis, jenkins, nginx
```

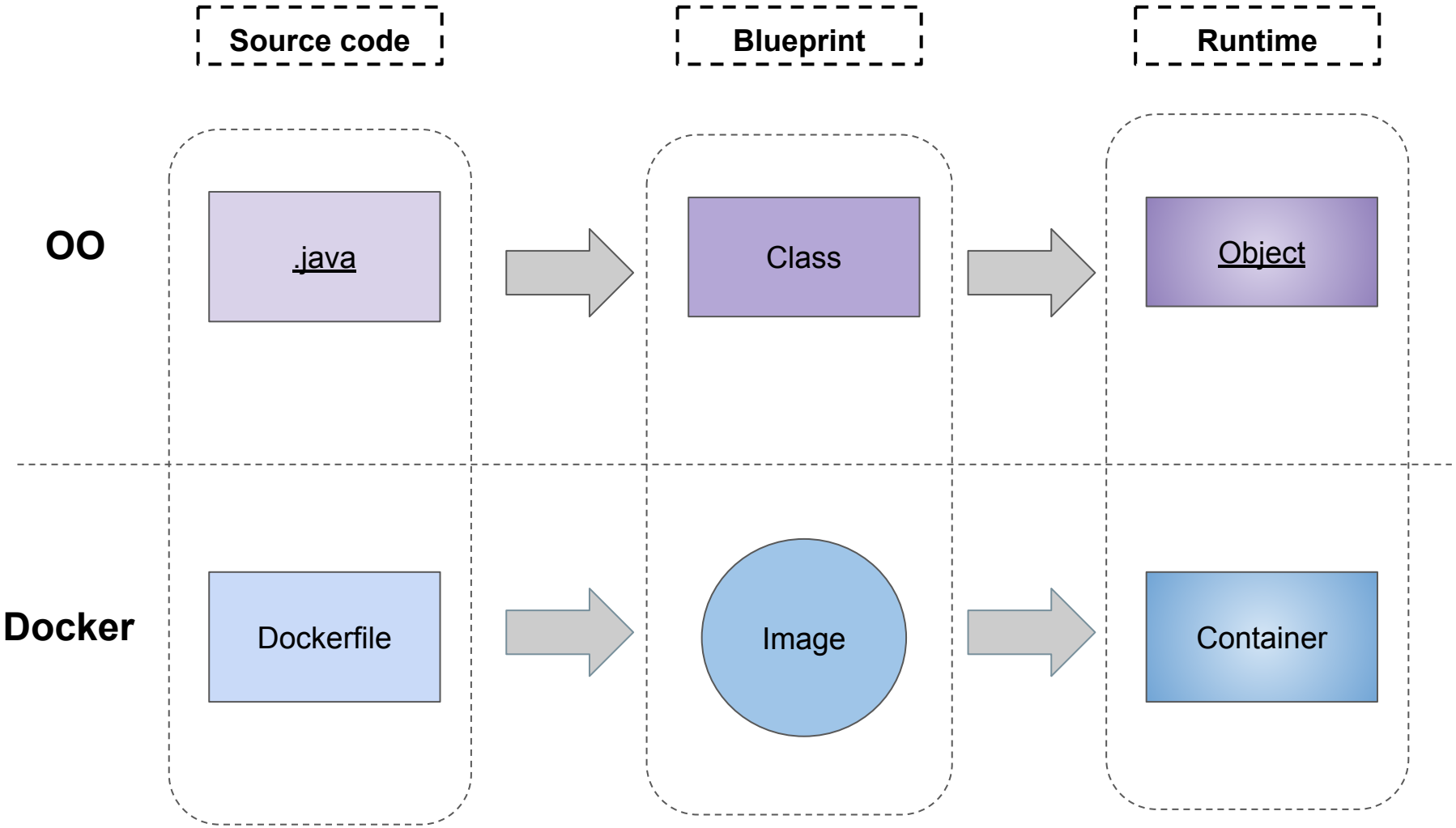
Container name or commit ID
```
mydb      #name
c7337     #commit id
```

Docker Build (Dockerfile)

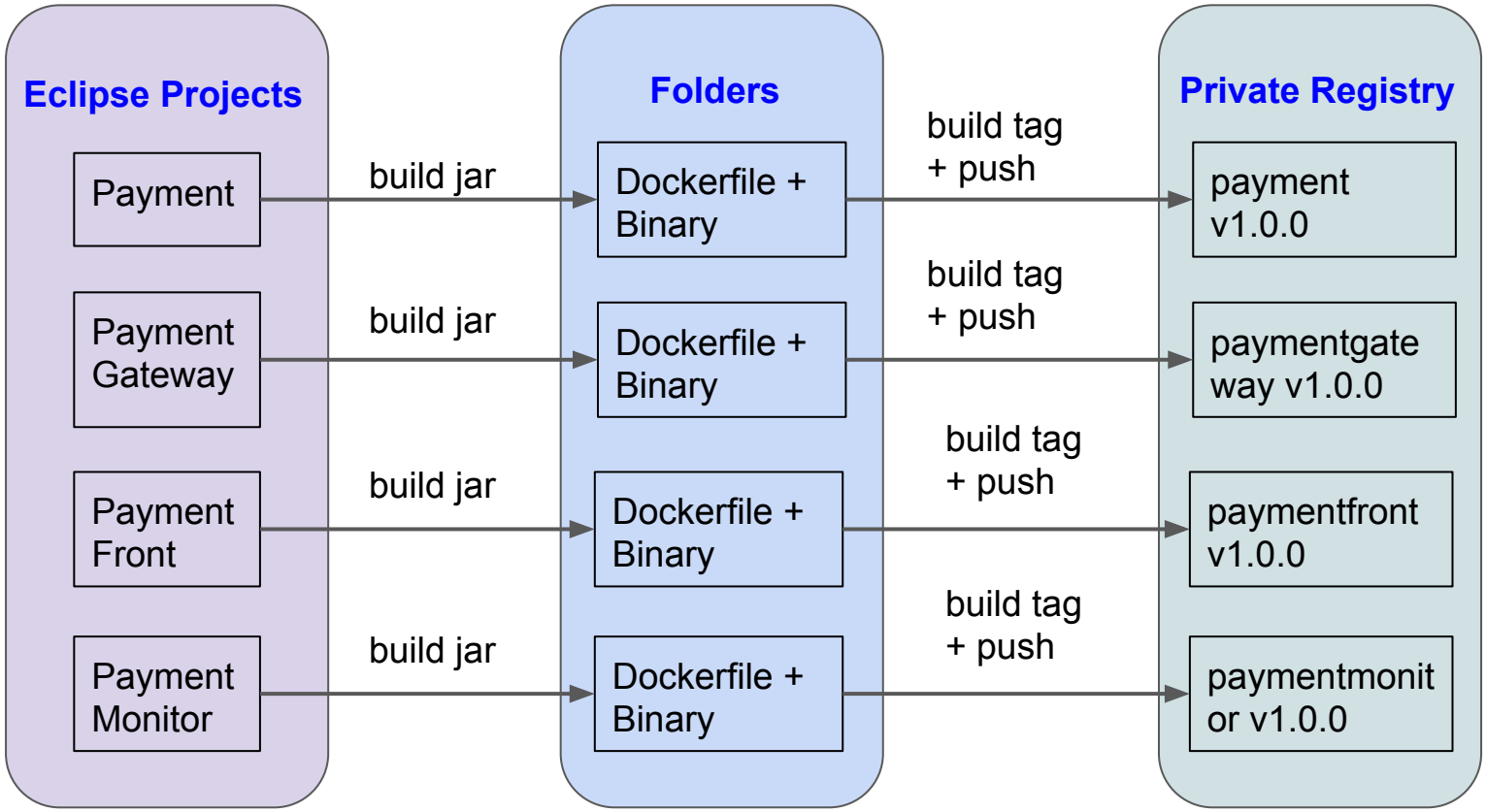
Dockerfile

- A **Dockerfile** is a text based script that **contains instructions and commands** for building the image from the base image.
- Docker reads this Dockerfile when you request a build of an image, executes the instructions, and returns a final image.
- **Docker build** is the process of building a Docker image from a Dockerfile.
- **Docker layer** is each layer in a Docker context represents an instruction included in a Docker image's Dockerfile.The layers can also be referred to as "build steps".
- **Docker build cache**, every time you build a Docker image, each build step is cached. Reuse cached layers that do not change in the image rebuild process to improve the build time.

Compare with Object-Oriented



Container Development Example




```
FROM 192.168.99.117:5000/java:8
MAINTAINER AnurakTh
EXPOSE 8900
COPY payment-1.0.0.jar /home/payment.jar
CMD ["/usr/lib/jvm/java-8-openjdk-amd64/bin/java", "-jar", "/home/payment.jar"]
```

Dockerfile Examples

```
FROM 192.168.99.117:5000/java:8
MAINTAINER AnurakTh
EXPOSE 8900
COPY payment-1.0.0.jar /home/payment.jar
CMD ["/usr/lib/jvm/java-8-openjdk-amd64/bin/java", "-jar", "/home/payment.jar"]
```

Java Application

```
FROM 10.28.104.174:5000/python:2.7
ADD . /code
WORKDIR /code
RUN chmod +r /code
RUN ls /code -la
RUN pip install -r requirements.txt
CMD python app.py
```

PHP Application

Docker Image Building

Example commands

- # build payment image
- $ docker build -t reader .
- $ docker build -f <Dockerfile name> -t reader .
- $ docker build .
- $ *docker tag reader <docker hub account>/reader:latest*
- $ *docker push <docker hub account>/reader:latest*

***** Just version your tags. Every time. *****

***** Do not uses the latest tag in production. *****

*** The ‘latest’ tag does not actually mean latest, it doesn’t mean anything. ***

Dockerfile instructions

- FROM
- RUN
- CMD
- LABEL
- EXPOSE
- ENV
- ADD
- COPY
- ENTRYPOINT
- VOLUME
- USER
- WORKDIR
- ARG

Note: The instruction is not case-sensitive.

FROM Instruction

- The FROM instruction initializes a new build stage and sets the Base Image for subsequent instructions.
- A valid Dockerfile must start with a FROM instruction.

```
FROM [--platform=<platform>] <image> [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

Run Instruction

- The RUN instruction will **execute any commands** in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.
- RUN has 2 forms:
  - **RUN <command>** (shell form, the command is run in a shell, which by default is /bin/sh -c on Linux or cmd /S /C on Windows)
  - **RUN ["executable", "param1", "param2"]** (exec form)
- Examples

```
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
```

```
RUN /bin/bash -c 'source $HOME/.bashrc; echo $HOME'
```

```
RUN ["/bin/bash", "-c", "echo hello"]
```

CMD Instruction

- **To provide defaults for an executing container.**
- The CMD instruction has three forms:
  - `CMD ["executable","param1","param2"]` (*exec* form, this is the preferred form)
  - `CMD ["param1","param2"]` (as *default parameters to ENTRYPOINT*)
  - `CMD command param1 param2` (*shell* form)
- Example
  - the **shell form** of the CMD, then the <command> will execute in /bin/sh -c

```
FROM ubuntu
CMD echo "This is a test." | wc -
```

  - **exec form**, must express the command as a JSON array and give the full path to the executable.

```
FROM ubuntu
CMD ["/usr/bin/wc","--help"]
```

- If you would like your container to run the same executable every time, then you should consider using ENTRYPOINT in combination with CMD.

LABEL Instruction

- The LABEL instruction adds metadata to an image.
- A LABEL is a key-value pair.
- Format

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

- Examples

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

```
LABEL multi.label1="value1" multi.label2="value2" other="value3"
```

```
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

- To view an image's labels, use the docker image inspect command. You can use the --format option to show just the labels;

```
docker image inspect --format='' myimage
```

```
{
  "com.example.vendor": "ACME Incorporated",
  "com.example.label-with-value": "foo",
  "version": "1.0",
  "description": "This text illustrates that label-values can span multiple lines.",
  "multi.label1": "value1",
  "multi.label2": "value2",
  "other": "value3"
}
```

- MAINTAINER (deprecated)

```
LABEL maintainer="SvenDowideit@home.org.au"
```

EXPOSE Instruction

- The EXPOSE instruction informs Docker that the container **listens on the specified network ports at runtime.**
- Format

```
EXPOSE <port> [<port>/<protocol>...]
```

- Examples

```
EXPOSE 80/tcp
EXPOSE 80/udp
```

- To actually publish the port when running the container, use the -p flag on docker run

```
docker run -p 80:80/tcp -p 80:80/udp ...
```

ENV Instruction

- The ENV instruction sets the environment variable <key> to the value <value>.
- Format

```
ENV <key>=<value> ...
```

- Examples

```
ENV MY_NAME="John Doe"
ENV MY_DOG=Rex\ The\ Dog
ENV MY_CAT=fluffy
```

```
ENV MY_NAME="John Doe" MY_DOG=Rex\ The\ Dog \
    MY_CAT=fluffy
```

```
ENV MY_VAR my-value
```

The alternative syntax is supported for backward compatibility.

- The environment variables set using ENV will persist when a container is run from the resulting image.
- You can view the values using docker inspect, and change them using
  - docker run --env <key>=<value>

ADD Instruction

- The ADD instruction copies new **files, directories or remote file URLs** from <src> and adds them to the filesystem of the image at the path <dest>.
- Format

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

- Example

```
ADD test.txt relativeDir/
```

```
ADD test.txt /absoluteDir/
```

```
ADD hom* /mydir/
```

```
ADD hom?.txt /mydir/
```

```
ADD --chown=55:mygroup files* /somedir/
ADD --chown=bin files* /somedir/
ADD --chown=1 files* /somedir/
ADD --chown=10:11 files* /somedir/
```

- If <src> is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory.

Users > ktb_user > Documents > dockerTmp > JbossApp1 > Dockerfile > FROM

```
FROM 192.168.99.117:5000/jboss/wildfly:v1.0.1
ADD node-info.war /opt/jboss/wildfly/standalone/deployments/
ADD node-info.war.dodeploy /opt/jboss/wildfly/standalone/deployments/
```

Copy Instruction

- The COPY instruction copies new **files or directories** from <src> and adds them to the filesystem of the container at the path <dest>.
- Formats

```
COPY [--chown=<user>:<group>] <src>... <dest>
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

- Examples

```
COPY hom* /mydir/
```

```
COPY hom?.txt /mydir/
```

```
COPY --chown=55:mygroup files* /somedir/
COPY --chown=bin files* /somedir/
COPY --chown=1 files* /somedir/
COPY --chown=10:11 files* /somedir/
```

ENTRYPOINT Instruction

- An ENTRYPOINT allows you to configure a container that will run as an executable.
- Formats
  - The exec form, which is the preferred form:

    ```
    ENTRYPOINT ["executable", "param1", "param2"]
    ```

  - The shell form:

    ```
    ENTRYPOINT command param1 param2
    ```

- We cannot override the ENTRYPOINT instruction by adding command-line parameters to the docker run command.

Note: There is a way to override the ENTRYPOINT instruction - you need to add the --entrypoint flag prior to the container_name when running the command.

CMD with ENTRYPOINT

- There are some situations in which **combining CMD and ENTRYPOINT** would be the best solution for your Docker container. In such cases, **the executable is defined with ENTRYPOINT, while CMD specifies the default parameter.**
- If you are using both instructions, make sure to keep them in **exec form**.

Dockerfile-cmd-entrypoint

```
FROM ubuntu:18.04
LABEL maintainer=Anurakth
RUN apt-get update
ENTRYPOINT ["echo", "Hello"]
CMD ["World"]
```

VOLUME Instruction

- The VOLUME instruction creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.
- Formats

```
VOLUME ["/data"]
```

- Examples

```
FROM ubuntu
RUN mkdir /myvol
RUN echo "hello world" > /myvol/greeting
VOLUME /myvol
```

- Override volume name during docker run with -v option
  - docker run -v mydata:/myvol <image-name>

USER Instruction

- The USER instruction sets the **user name (or UID)** and optionally the **user group (or GID)** to use when running the image and for any RUN, CMD and ENTRYPOINT instructions that follow it in the Dockerfile.
- Formats

```
USER <user>[:<group>]
```

```
USER <UID>[:<GID>]
```

- Examples

```
FROM ubuntu:latest
RUN useradd -r -u 1001 -g appuser appuser
USER appuser
ENTRYPOINT [“sleep”, “infinity”]
```

- Another option is to run a docker container and specify the username or uid, and also the group name or gid at runtime.
  - *$ docker run -d --user 1001 ubuntu:latest sleep infinity*

WORKDIR Instruction

- The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile.
- If the WORKDIR doesn't exist, it will be created even if it's not used in any subsequent Dockerfile instruction.
- Formats

```
WORKDIR /path/to/workdir
```

- Examples

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

The output of the final pwd command in this Dockerfile would be /a/b/c.

ARG

- ARG is the only instruction that may precede FROM in the Dockerfile
- The ARG instruction defines a variable that users can **pass at build-time** to the builder with the docker build command using the **--build-arg <varname>=<value>**
- **Formats**

```
ARG <name>[=<default value>]
```

- **Examples**

```
FROM ubuntu
ARG CONT_IMG_VER
ENV CONT_IMG_VER=v1.0.0
RUN echo $CONT_IMG_VER
```

```
FROM busybox
ARG user1=someuser
ARG buildno=1
# ...
```

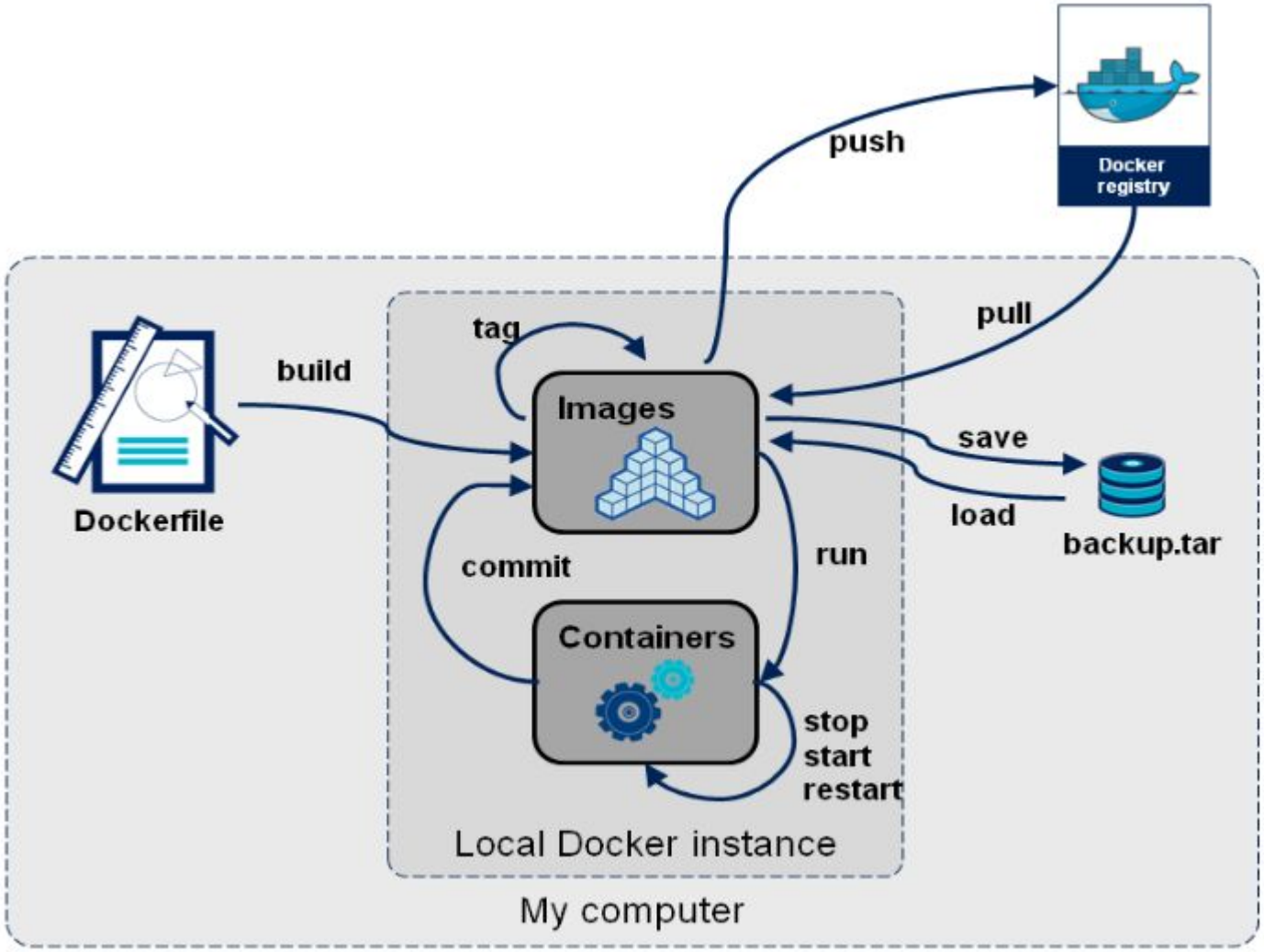
```
$ docker build --build-arg CONT_IMG_VER=v2.0.1 .
```

Predefined ARGs

- `HTTP_PROXY`
- `http_proxy`
- `HTTPS_PROXY`
- `https_proxy`
- `FTP_PROXY`
- `ftp_proxy`
- `NO_PROXY`
- `no_proxy`

```
--build-arg HTTP_PROXY=http://user:pass@proxy.lon.example.com
```

Container Development



Credit: https://blog.octo.com/en/docker-registry-first-steps/

Docker Build Cache

Show docker disk usage

$ *docker system df*

Remove Docker Build Cache

$ *docker images -a*

$ docker builder prune

Dangling Images

$ *docker images --filter "dangling=true"*

$ *docker rmi $(docker images -q --filter "dangling=true")*

Dockerfile best practices

https://docs.docker.com/develop/dev-best-practices/

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/