Agenda

Containers
Platform
Docker Machine
Docker Engine
Docker Hub
First Steps (Exercise)
Dockerfile
Simple Web App (Excercise)
Dockerfile Best Practices
Complex Web App (Excercise)
Summary


Docker Containers

Motivation


~2000
Today
Monolithic
Slow changing
Big Servers
Loosely Coupled Services
Rapidly updated
Many Small Servers

Why Containers?

Containers are a standard format

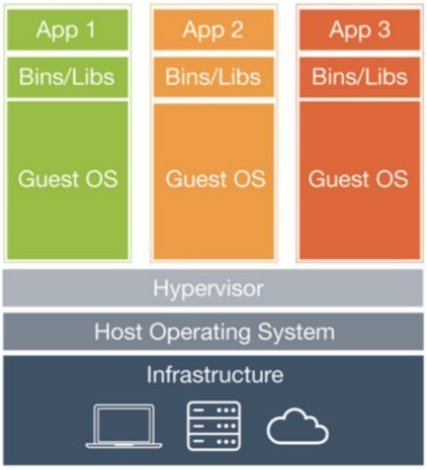
Easily portable across environment

Packages up software binaries and dependencies

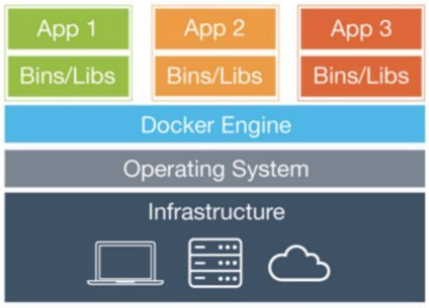
Isolates software from each other

Ecosystem has developed around containers

What is a Container?



Virtual Machines



Containers

- filesystem
- processes
- memory
- network ports
- devices

Why should devs care?

- A fast, clean, safe, hygienic and portable runtime
- No missing dependencies, packages and other pain points
- Own isolated container, various versions of libraries
- Automate testing, integration, packaging
- Reduce concerns about compatibility on platforms
- Easy service deployment

Why should ops care?

- Make lifecycle more efficient, consistent and repeatable
- Increase the quality of code produced by devs
- Eliminate inconsistencies between dev, test, prod and customer environments
- Support segregation of duties
- Significantly improve the speed and reliability of CI/CD
- Lightweight, address significant performance issues, deployment costs

Solutions

Docker Engine (free/commercial)
Docker Compose

Docker Machine
Docker Swarm

Docker Hub (service, public/private)
vs. Docker Registry (free, CLI only)
vs. Docker Trusted Registry (enterprise)
Docker Store (beta, commercial)

Docker Cloud (service, depends on providers)
vs. Docker Datacenter with DTR & UCP (enterprise)

Setup

A) Docker for Linux / Mac / Windows

- Docker Engine [via HyperKit see Guide @justincormack]
- docker client
- docker-compose

B) Docker Toolbox for Mac / Windows

- Docker Engine (via VirtualBox with boot2docker)
- docker client
- docker-compose
- docker-machine and Docker Quickstart Terminal
- Kitematic GUI

Details: https://docs.docker.com

Docker Machine

A tool to provision Docker Hosts

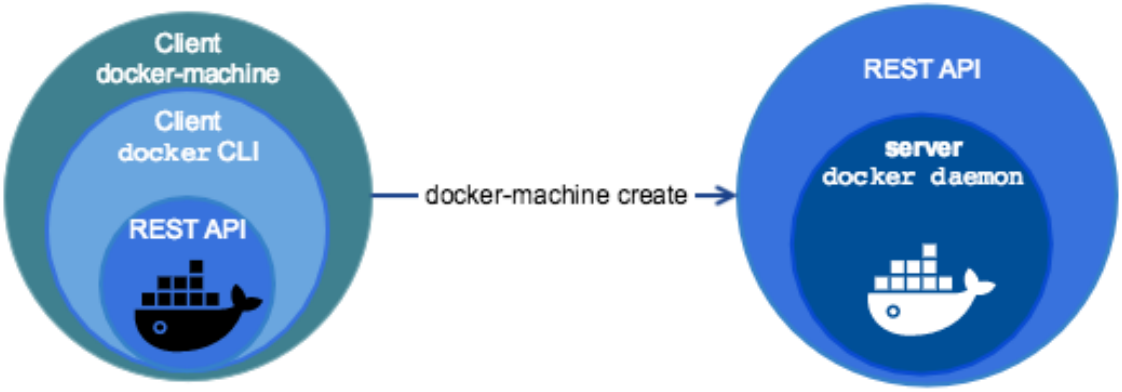
`docker-machine`

Docker Hosts
on remote/local systems

"Each **Machine** (managed host) is the combination of a **Docker Host** and a configured **docker-machine** client!"

Docker Machine

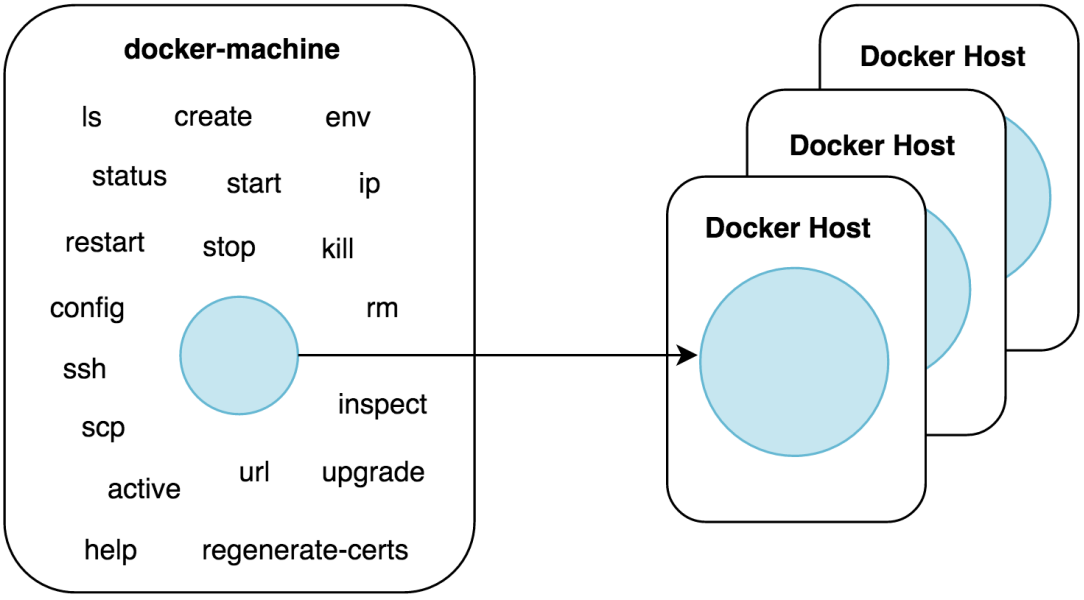
A tool to provision Docker Hosts



Create = install the Docker Engine on a virtual host.

Machine CLI Reference

Commands



docker-machine Client

Provision a local VM as Docker Host

ls = list availible machines

```
docker-machine ls
```

create a machine named "default" as local VirtualBox VM

```
docker-machine create [--driver virtualbox default]
docker-machine create --driver virtualbox xpdays
```

env = prints "export env variables". run them to connect.

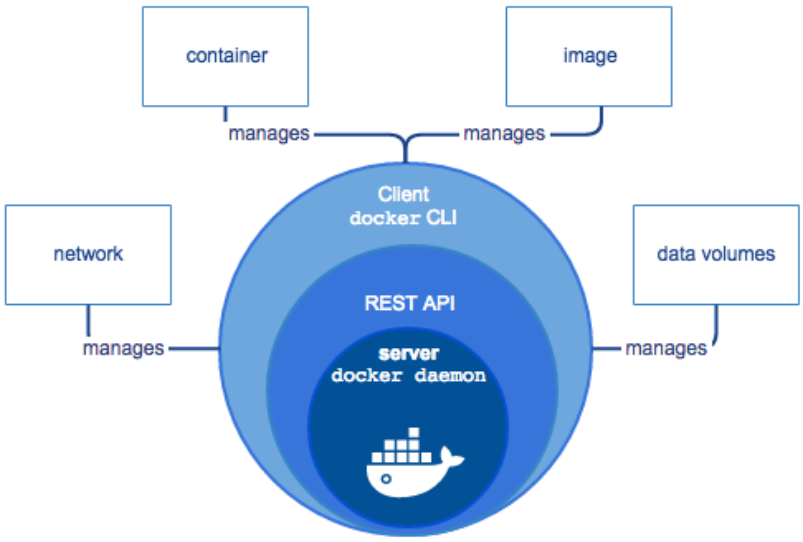
```
docker-machine env default
> export DOCKER_TLS_VERIFY="1"
> export DOCKER_HOST="tcp://172.16.62.130:2376"
> export DOCKER_CERT_PATH="/Users/<you>/.docker/machine/machines/default"
> export DOCKER_MACHINE_NAME="default"
eval "$(docker-machine env default)"
```

get ip address / stop / start the "default" machine

```
docker-machine ip default
docker-machine stop default
docker-machine start default
```

Docker Engine

A Client-Server Application



A Docker Daemon runs on a Docker Host and manages Docker objects (images, containers etc.).

docker Client

help

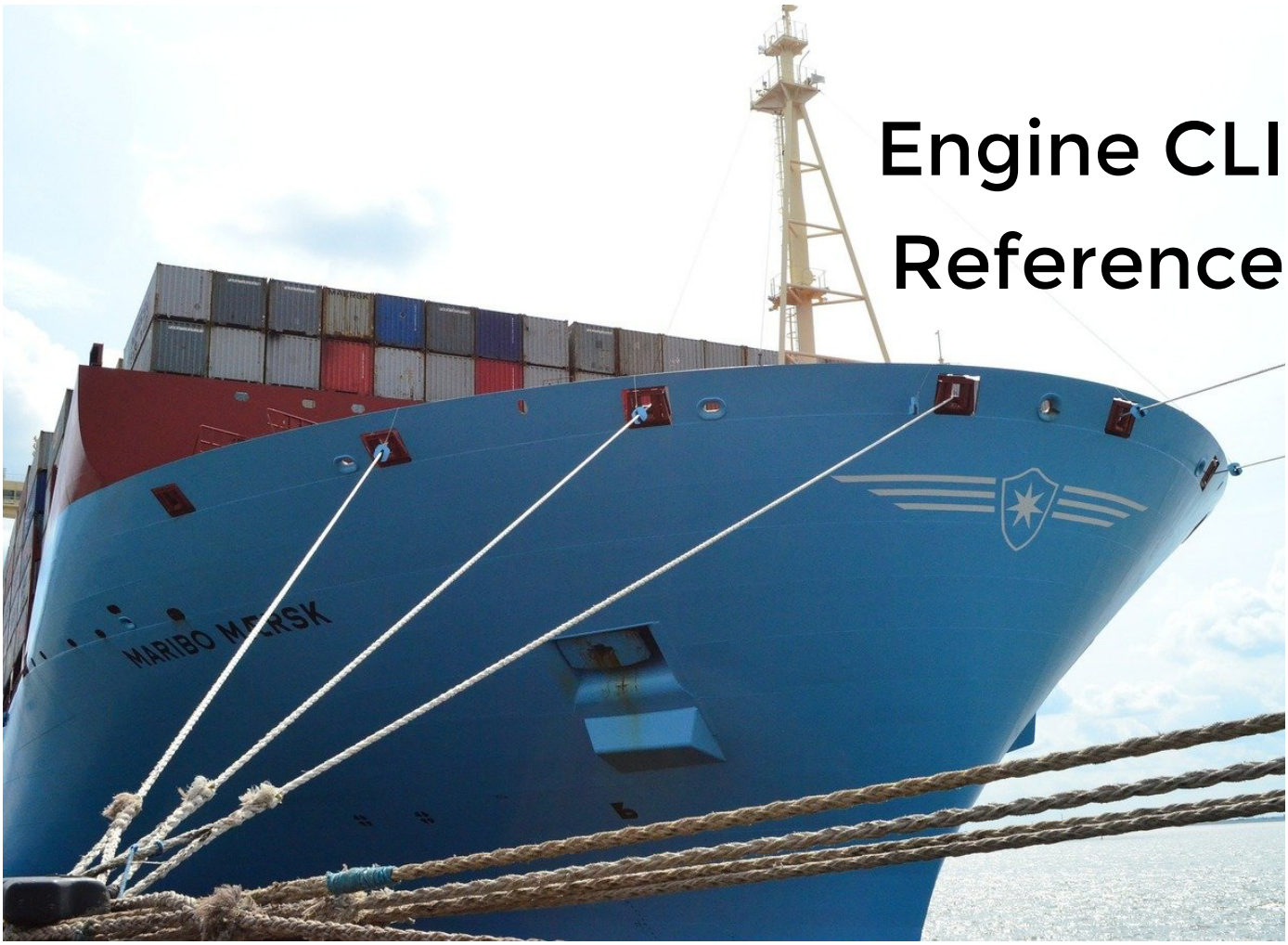
```
docker
docker --help
docker COMMAND --help
```

"Hello World" run in a Alpine container

```
docker run -it --name my-first-container alpine:3.4 \
    /bin/echo "Hello World"
```

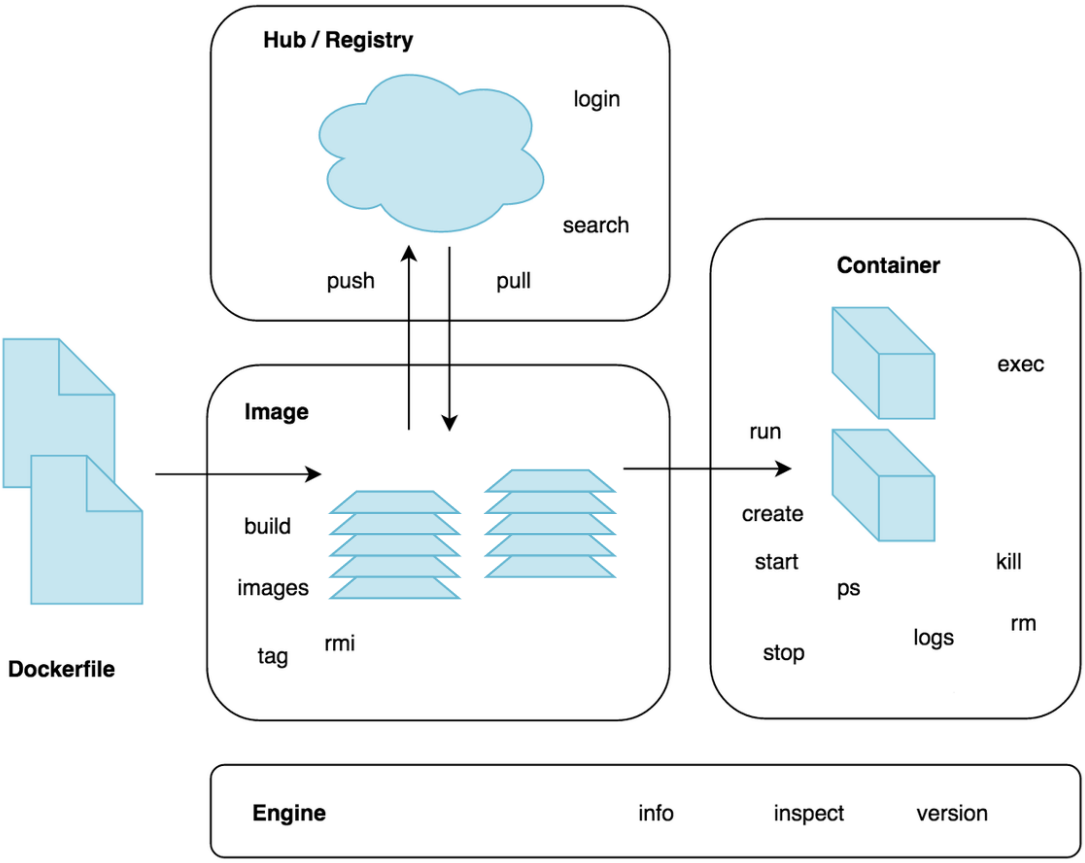
version and info

```
docker version
docker info
docker inspect my-first-container
```

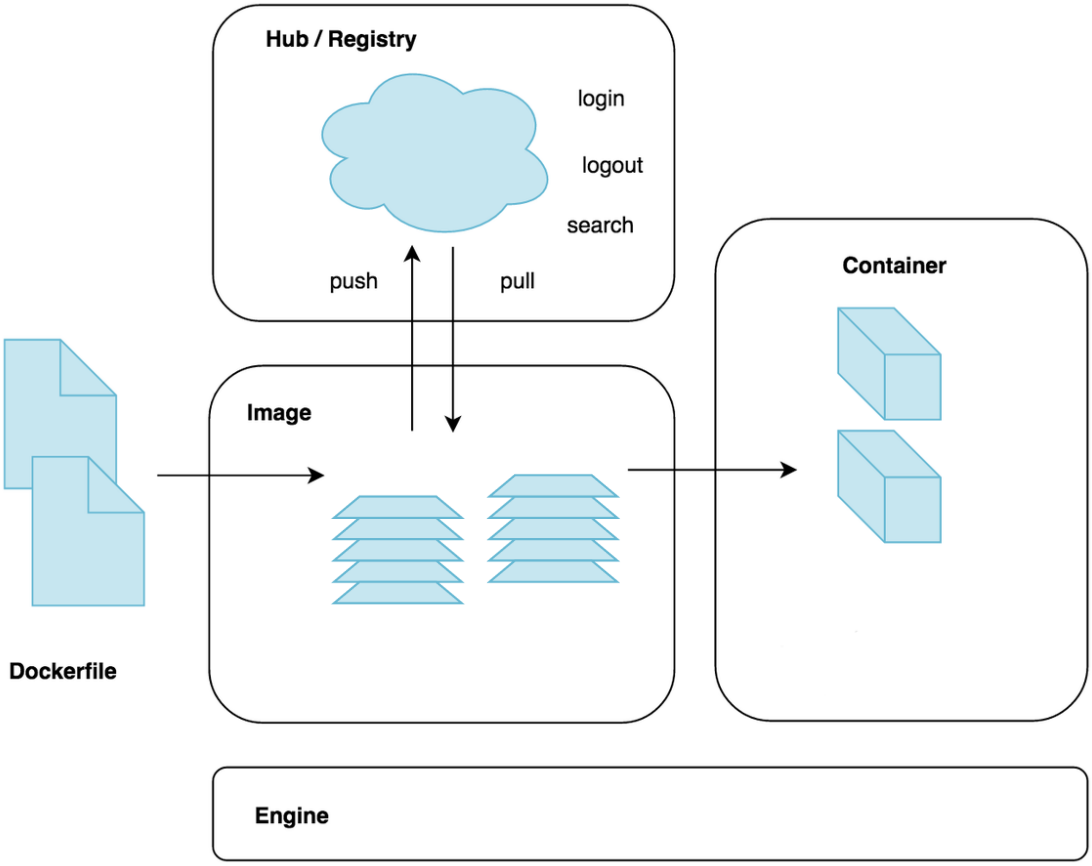


Engine CLI Reference

Commands



Hub Commands



Hub Commands

pull or push an image

```
docker pull $IMAGE_NAME[:$TAG]
docker pull $IMAGE_NAME:$@DIGEST
docker push $IMAGE_NAME:$TAG
docker push $REPO/$IMAGE_NAME:$TAG
docker push $REGISTRY:$PORT/REPO/$IMAGE_NAME:$TAG
```

search for images

```
docker search $IMAGE_NAME
docker search --filter stars=3 busybox
docker search --filter is-automated busybox
docker search --filter is-offical busybox
docker search --filter "is-offical=true" --filter "stars=3" --no-trunc busybox
```

login and logout to Docker Hub or other Registry

```
docker login [$REGISTRY_HOSTNAME]
docker login -u $USERNAME -p $PASSWORD -e $EMAIL
docker logout [$REGISTRY_HOSTNAME]
```

credentials store

Image Commands

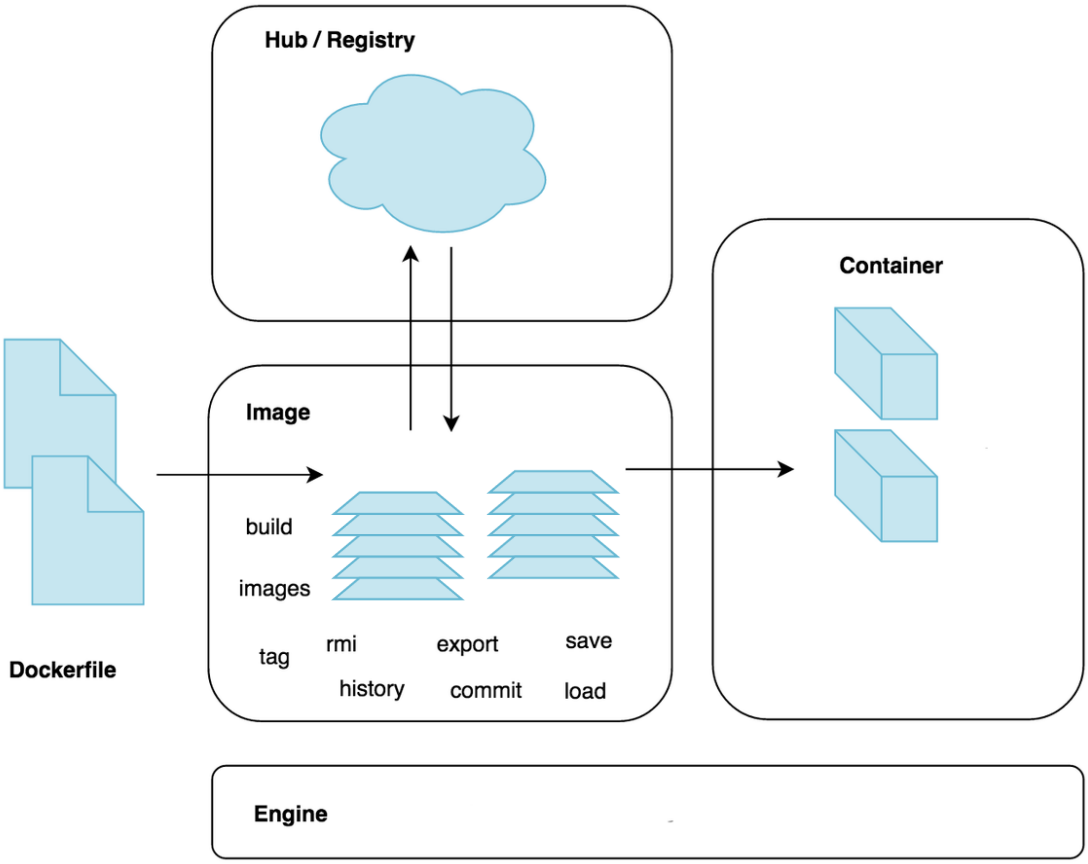


Image Commands

build an image from a Dockerfile

```
docker build -t $IMAGE_NAME:$TAG $DIR
docker build -t $IMAGE_NAME:$TAG -f $DOCKERFILE
docker build -t $IMAGE_NAME:$TAG -t $IMAGE_NAME:$TAG .
```

list images

```
docker images
docker images --no-trunc
docker images --digests
docker images --filter "dangling=true"
docker images --filter "before=image1"
docker images --filter "since=image3"
docker images --format "table {{.ID}}\t{{.Repository}}\t{{.Tag}}"
```

tag images

```
docker tag $ID          $REPO/$NAME:$TAG
docker tag $NAME        $REPO/$NAME:$TAG
docker tag $NAME:$TAG   $REPO/$NAME:$TAG
docker tag $ID          $REGISTRY_HOSTNAME:$PORT/$REPO/$NAME:$TAG
```

rmi = remove images

```
docker rmi $ID
docker rmi $NAME:$TAG
docker rmi $(docker images -f "dangling=true" -q)
```

Image Commands

history of an image

```
docker history $IMAGE:$TAG
```

save / load an image as/from tar

```
docker save busybox > busybox.tar
docker save --output ubuntu.tar ubuntu:lucid ubuntu:saucy
docker load < busybox.tar.gz
docker load --input ubuntu.tar
```

commit = create a new image from container changes

```
docker commit $CONTAINER_ID $REPO/$IMAGE:$TAG
docker commit --change "ENV key=value" $ID $REPO/$IMAGE:$TAG
docker inspect -f "{{ .Config.Env }}" $ID    # to check
docker commit --change='CMD ["apachectl", "-DFOREGROUND"]' \
    -c "EXPOSE 80" $ID $REPO/$IMAGE:$TAG
```

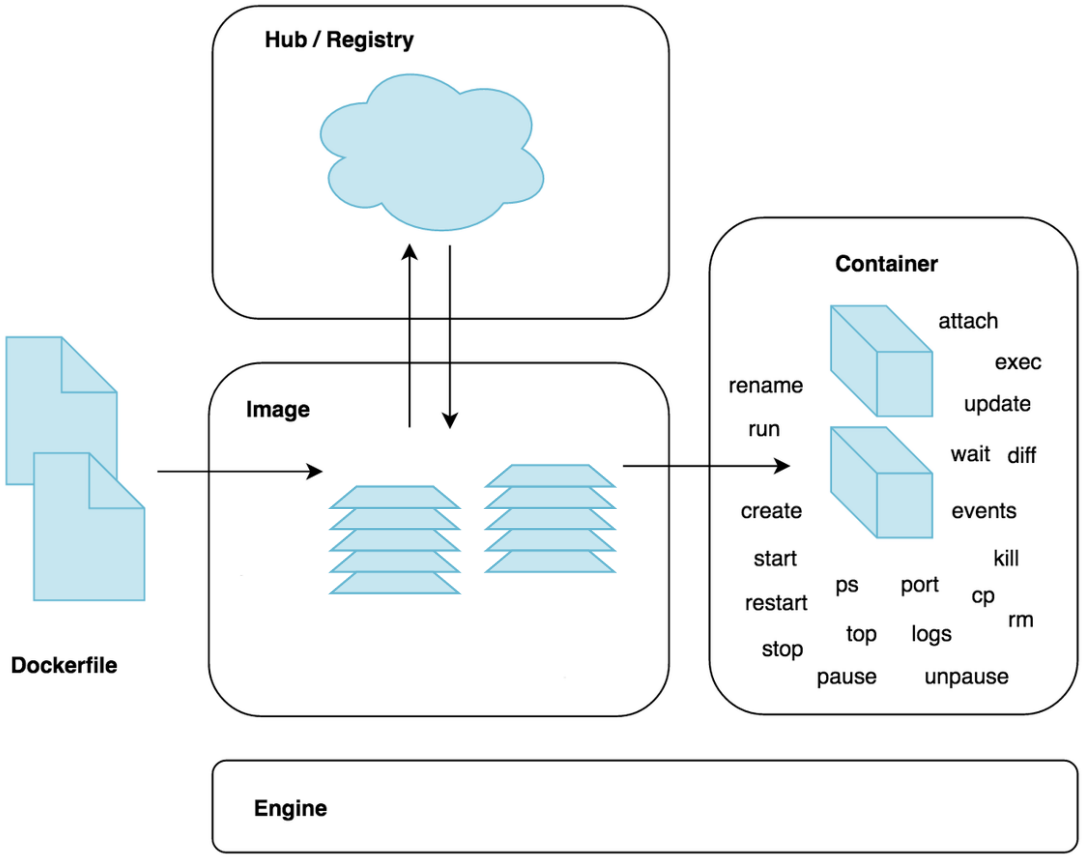
export filesystem as tar (without VOL)

```
docker export CONTAINER > latest.tar
docker export --output="latest.tar" CONTAINER
```

import filesystem from tar (without VOL)

```
docker import http://example.com/exampleimage.tgz
docker import /path/to/exampleimage.tgz
cat exampleimage.tgz | docker import --message "New image imported from tarball" \
  - exampleimagelocal:new
```

Container Commands



Container Commands

run a container (= pull/create/start)

```
docker run --rm \
    --env-file "$PATH_TO_ENV_FILE" \
    -e "key=value" \
    -p $HOST_HTTP_PORT:$DOCKER_HTTP_PORT \
    -p $HOST_TCP_PORT:$DOCKER_TCP_PORT \
    -v $HOST_DATA_DIR:$DOCKER_DATA_VOL \
    -v $HOST_CONFIG_DIR:$DOCKER_CONFIG_VOL \
    --name $CONTAINER_NAME \
     $IMAGE_NAME:$TAG [$COMMAND]
```

create a container from an image

```
docker create --name $CONTAINTER_NAME $IMAGE_NAME:$TAG
// options are the same as at docker run
```

start and stop a container

```
docker stop ${ID}
docker stop ${CONTAINTER_NAME}
docker start ${CONTAINTER_NAME}
docker start -a -i ${ID}

    -a attach STDOUT/STDERR
    -i attach STDIN
```

Container Commands

ps = show containers

```
docker ps
docker ps -a
docker ps --filter status=paused
docker ps --filter "label=color=blue"
docker ps --filter "name=ubun"
docker ps --filter ancestor=ubuntu
docker ps --filter volume=/data --format "table {{.ID}}\t{{.Mounts}}"
docker ps --filter network=net1
```

rm = remove containers

```
docker rm $ID
docker rm /$LINK
docker rm $(docker ps -a -q)
docker rm -v redis
```

top = display processes in a container

```
docker top CONTAINER [ps OPTIONS]
```

kill one or more containers

```
docker kill CONTAINER [CONTAINER...]
docker kill -s SIGTERM CONTAINER
```

Container Commands

list port mappings of a running container

```
docker port CONTAINER
```

logs of a running container

```
docker logs CONTAINER
docker logs --follow CONTAINER
docker logs --tail 10 CONTAINER
docker logs --since 1h30m CONTAINER
```

exec = run a command in running container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
docker exec -it CONTAINER bash
docker exec -d CONTAINER touch /tmp/file
```

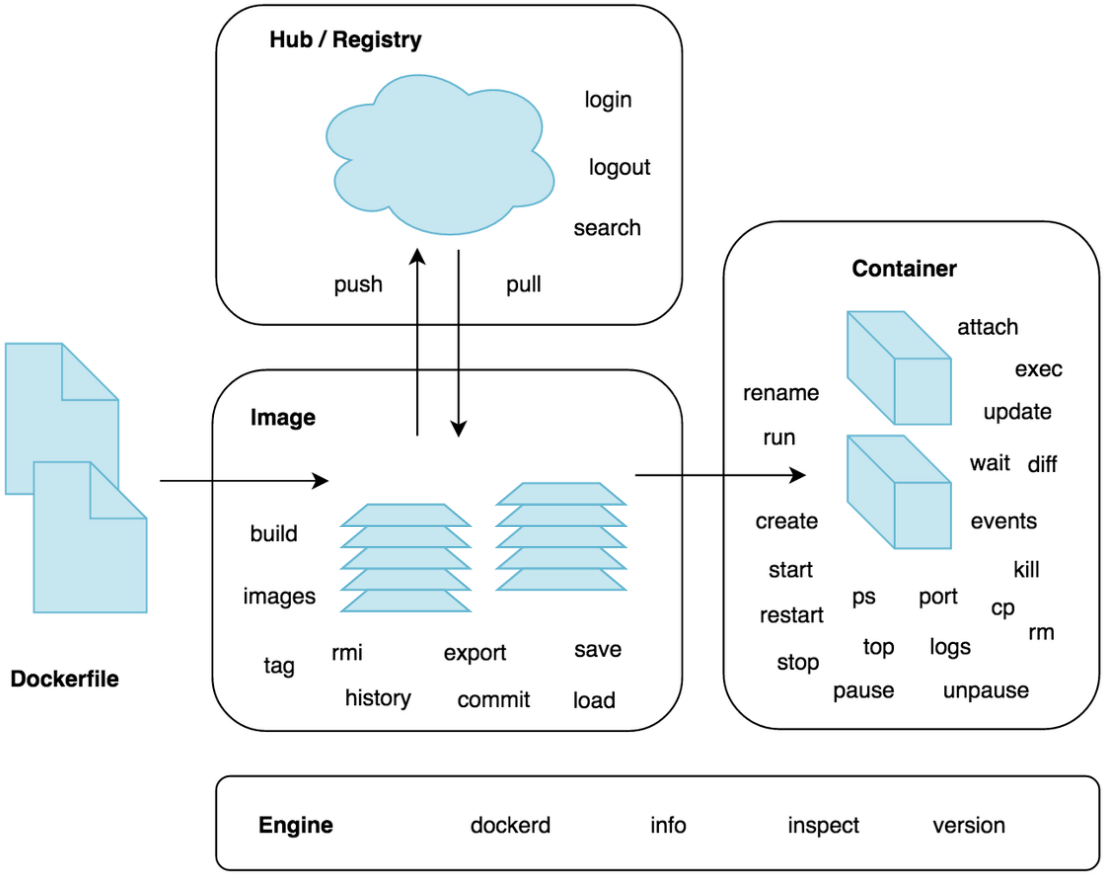
attach to a running container shell (PID 1)

```
docker attach CONTAINER              # CTRL-c for SIGKILL or CTRL-p CTRL-q to leave
docker attach --no-stdin CONTAINER
```

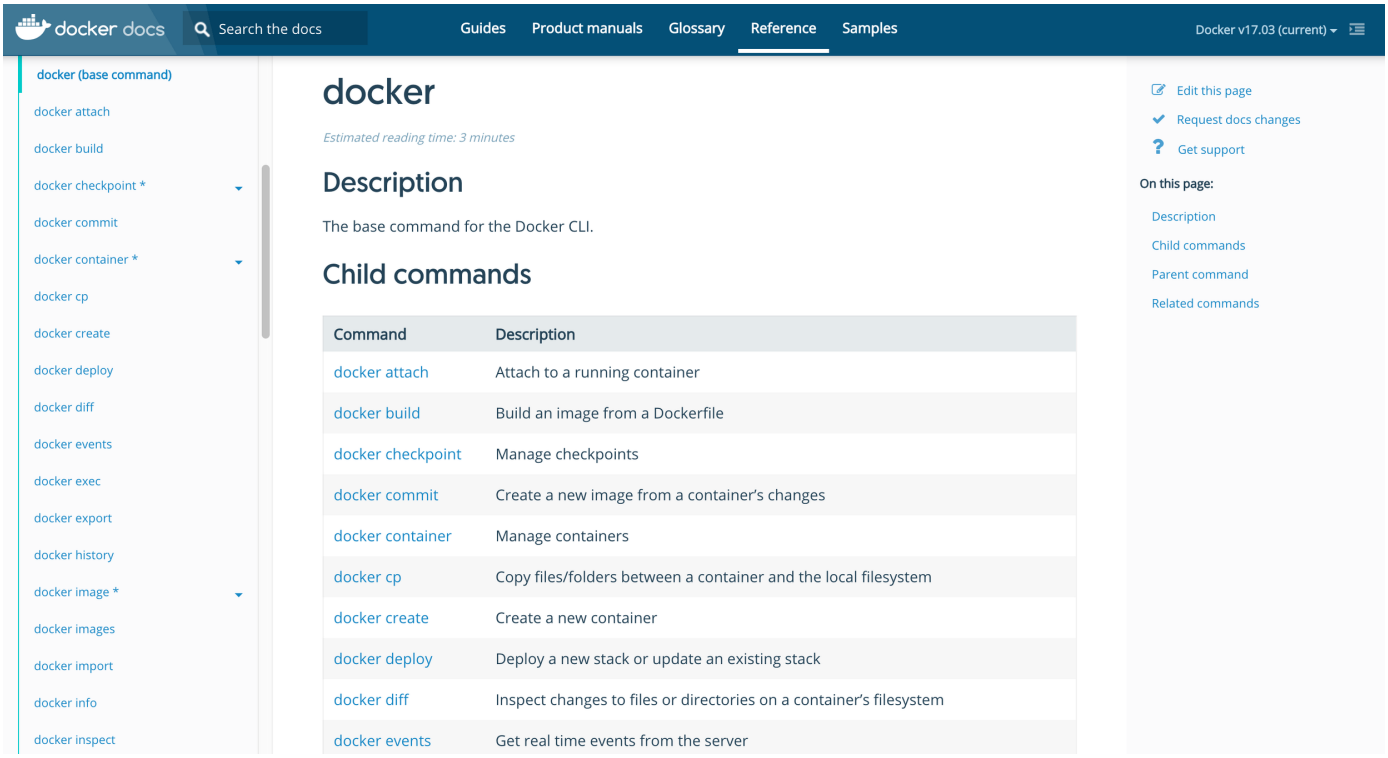
update config of a running container

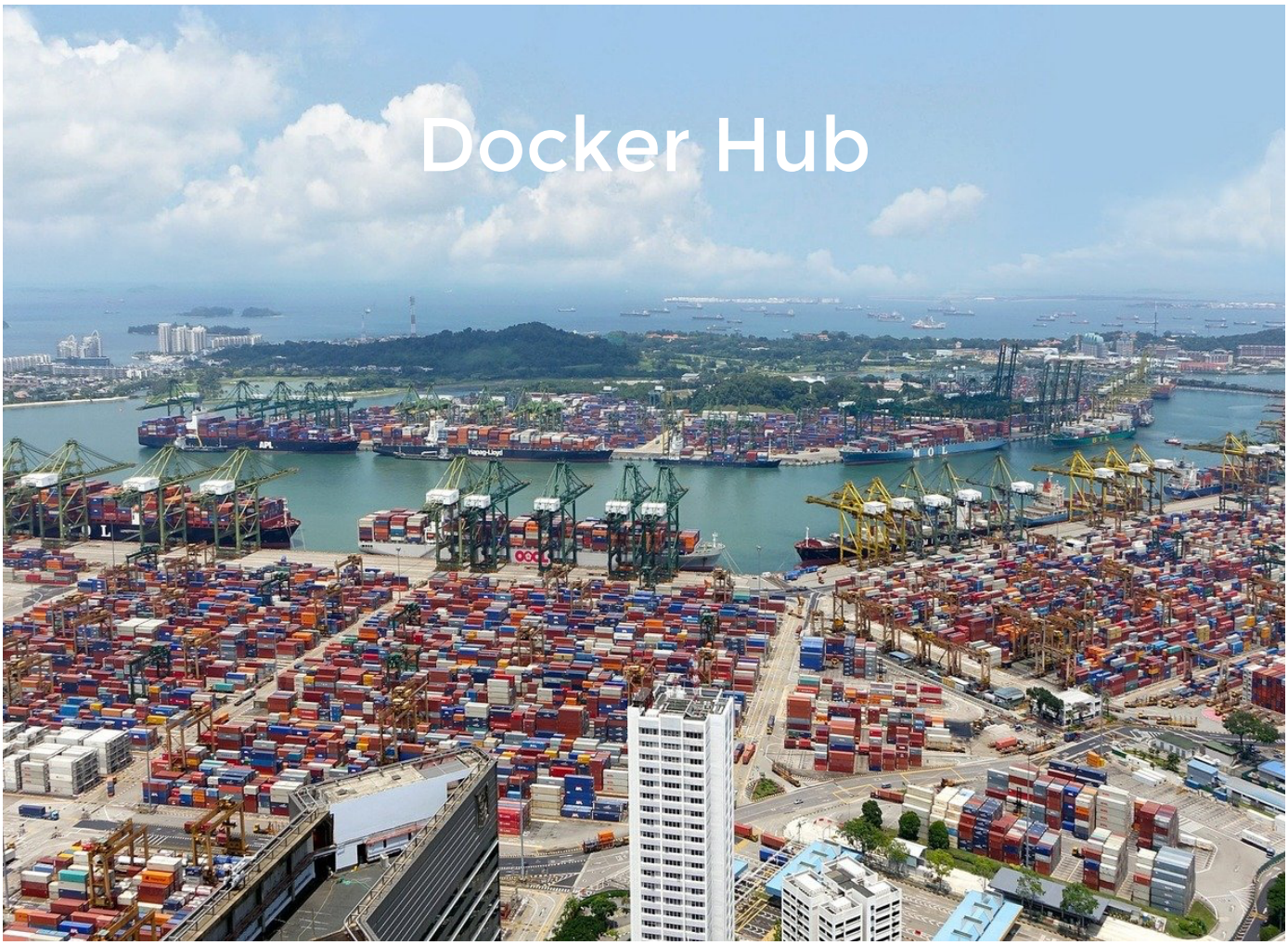
```
docker update --cpu-shares 512 -m 300M CONTAINER1 CONTAINER2
```

All Commands

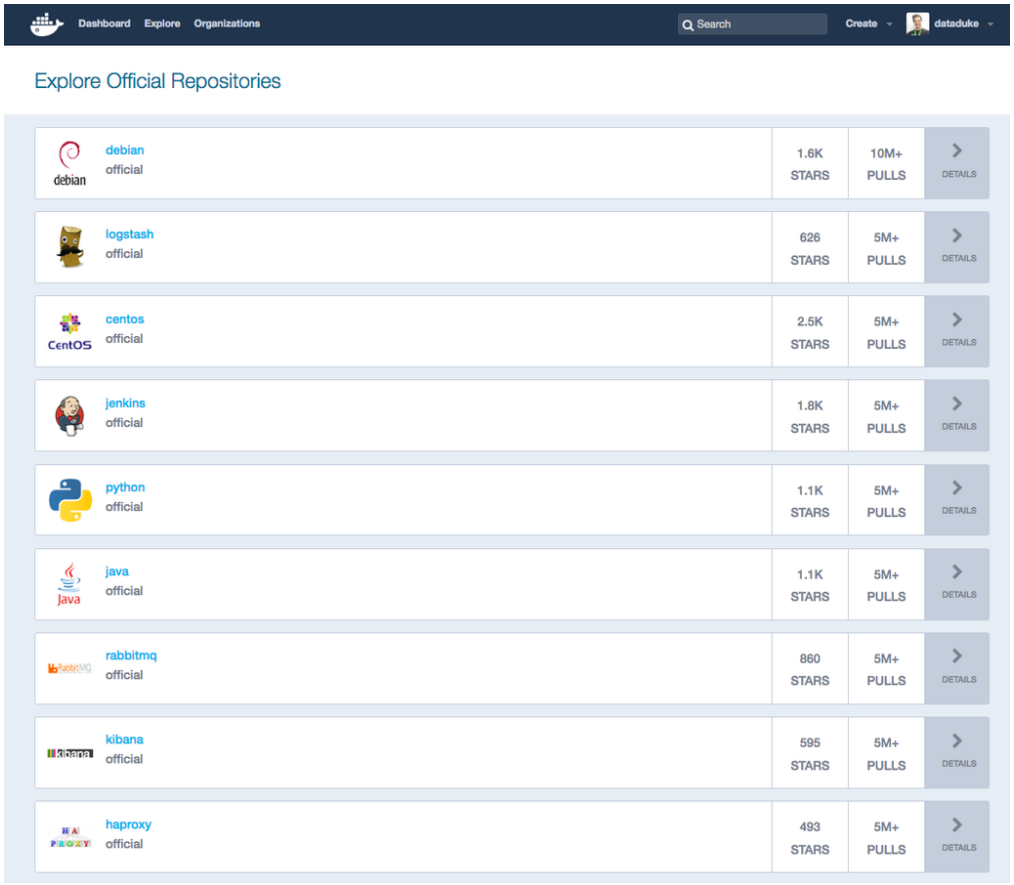


Docker Docs


docker docs
Search the docs
Guides
Product manuals
Glossary
Reference
Samples
Docker v17.03 (current)
docker (base command)
docker attach
docker build
docker checkpoint *
docker commit
docker container *
docker cp
docker create
docker deploy
docker diff
docker events
docker exec
docker export
docker history
docker image *
docker images
docker import
docker info
docker inspect
docker
Estimated reading time: 3 minutes
Description
The base command for the Docker CLI.
Child commands
Command
Description
docker attach
Attach to a running container
docker build
Build an image from a Dockerfile
docker checkpoint
Manage checkpoints
docker commit
Create a new image from a container's changes
docker container
Manage containers
docker cp
Copy files/folders between a container and the local filesystem
docker create
Create a new container
docker deploy
Deploy a new stack or update an existing stack
docker diff
Inspect changes to files or directories on a container's filesystem
docker events
Get real time events from the server
Edit this page
Request docs changes
Get support
On this page:
Description
Child commands
Parent command
Related commands


Docker Hub

Docker Hub


Dashboard
Explore
Organizations
Search
Create
dataduke
Explore Official Repositories
debian
official
1.6K STARS
10M+ PULLS
DETAILS
logstash
official
626 STARS
5M+ PULLS
DETAILS
centos
official
2.5K STARS
5M+ PULLS
DETAILS
jenkins
official
1.8K STARS
5M+ PULLS
DETAILS
python
official
1.1K STARS
5M+ PULLS
DETAILS
java
official
1.1K STARS
5M+ PULLS
DETAILS
rabbitmq
official
860 STARS
5M+ PULLS
DETAILS
kibana
official
595 STARS
5M+ PULLS
DETAILS
haproxy
official
493 STARS
5M+ PULLS
DETAILS

Docker Hub

Base Images

- Busybox
- Alpine
- Debian
- Ubuntu

Environment Images

- PHP
- Ruby
- Open JDK
- Python
- Node

Tool Images

- Magento
- LimeSurvey
- Tomcat
- JBoss WildFly
- PHP Mentors Symfony

More Images

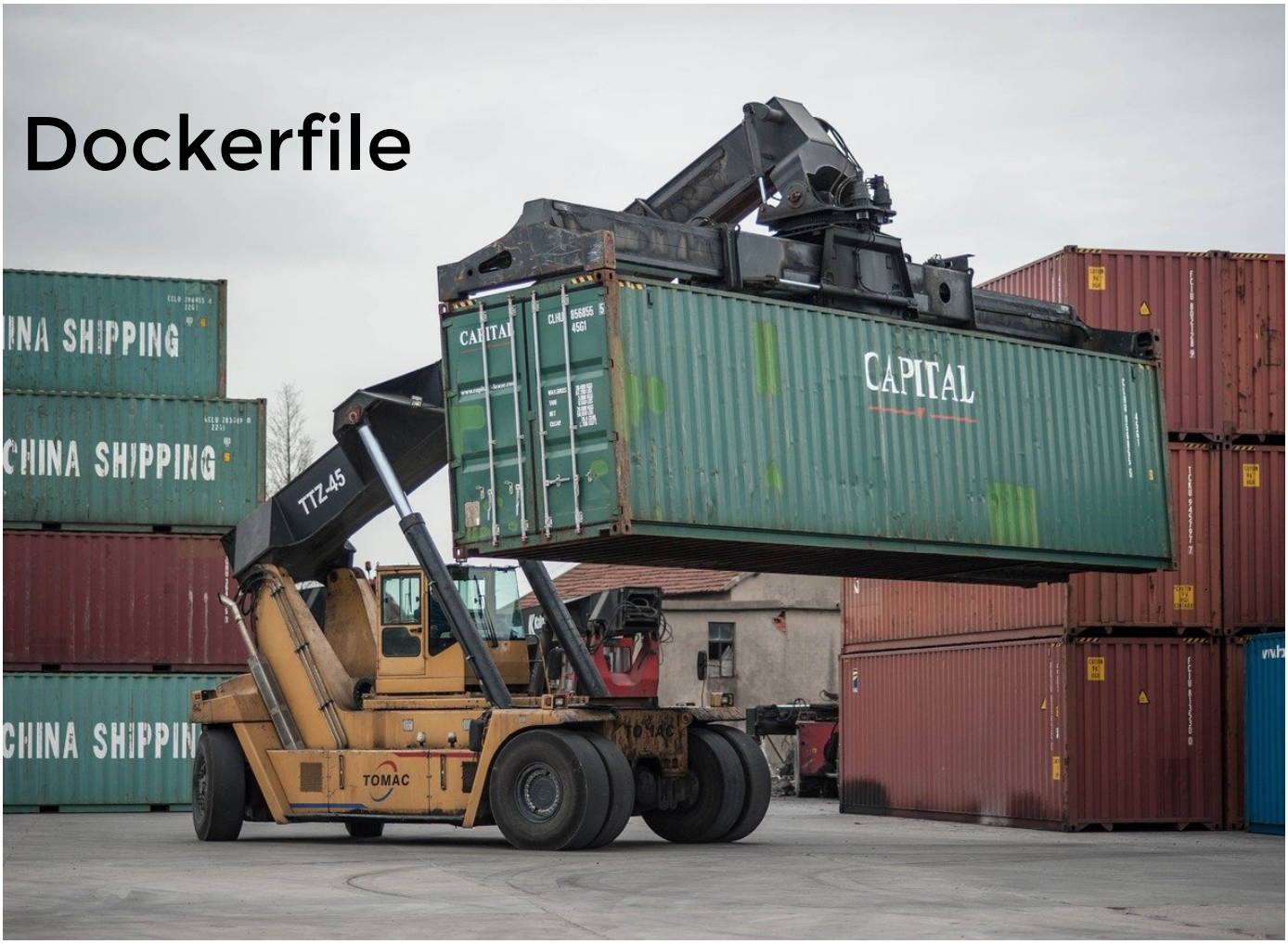
- github.com/docker-library
- All Official Images

Run your first container

- Pull alpine image
- List all images
- Run alpine image with `echo "Hello World"` command
- List running containers
- Start container and step into it

Create your first image

- Start ubuntu container and step into it
- Install curl and exit
- List all containers
- Commit the currently changed container
- List all images
- Name the new image curl_example:1.0
- Run new image with `curl https://www.google.com`



Dockerfile

Instructions

```
FROM <IMAGE_NAME>:<IMAGE_TAG>

MAINTAINER <NAME> <SURNAME> "name.surname@company.com"

LABEL <KEY>=<VALUE>

ARG <NAME>[=<DEFAULT VALUE>]

ENV <VAR>=<VALUE> \
    <VAR>=<VALUE>

RUN <SHELL_COMMAND> \
    <SHELL_COMMAND>

WORKDIR <PATH>

USER <NAME>

ADD ["<DIR>", "<FILE>", "<URL>", "<TAR>", "<CONTAINER_DEST_DIR>"]

COPY <HOST_DIR/FILE> <CONTAINER_DEST_DIR> // * ? file wildcards, relative/absolute dest path

VOLUME ["<MOUNT_PATH_DIR>", "<MOUNT_PATH_DIR>"]

EXPOSE <PORT>

ONBUILD [INSTRUCTION]

ENTRYPOINT ["<PROGRAM>"]

CMD ["<PARAM1>", "<PARAM2>"]
```

Get it running

1. Finalise Dockerfile
2. Build Docker image
3. Run Docker container
4. Open website to check if it is working

Basic Docker Commands

1. List running containers
2. Stop Docker container
3. List all Docker containers
4. Remove created container
5. Run Docker container with removal

Running containers in background

1. Run Docker container in background
2. Step into running container
3. Stop container
4. Start container
5. Step into the container
6. Kill running python process
7. Is the container still running?

Some more useful commands

1. Run Docker container and step into it
2. Run Docker container with mounted volume
    1. Change HTML files and refresh page

Inspect a container

1. Inspect the running flask-test-app container
   - Which information can be retrieved?
2. Create persistent volume with specific name
3. Run alpine container with created volume mounted to /data und the command `sh -c 'ping 8.8.8.8 > /data/ping.txt'`
4. Inspect the volume and have a look on the Mountpoint

Inspect images

1. Inspect flask-test-app image
    - Have a closer look which information can be found there
2. ls /graph/overlay2
    - We can find every layer here
    - If we pull another image there should appear more files then before

Best Practices

Dockerfile: Guidelines

1. Containers should be ephemeral
2. Use a .dockerignore file
3. Use small base images
4. Use tags
5. Group common operations
6. Avoid installing unnecessary packages
7. Run only one process per container
8. Minimize the number of layers
9. Sort mult-line arguments and indent 4 spaces:

```
RUN apt-get update && apt-get install --yes \
    cvs \
    git \
    mercurial \
    subversion
```

Best Practices

Dockerfile: Guidelines

10. Build Cache

- **CACHING:** Use whenever possible. Saves time.
- **DISABLE:** *docker build --no-cache=true -t NAME:TAG .*
- **CHECKSUMS:** For *ADD* and *COPY* the contents of the file(s) in the image are examined and a checksum is calculated for each file. During the cache lookup, the checksum is compared against the checksum in the existing images. Cache is invalid if anything has changed (besides file access dates)!
- **NO CACHE LOOKUP:** All other commands are not evaluted on a file level to determine a cache match/hit. Just the command string itself is used to find a match when processing files updated in the container, e.g. *RUN apt-get -y update.*

Best Practices

Dockerfile: Instructions

1. **FROM**: use current official Repositories,
   e.g. Debian is tightly controlled and kept minimal: 150 mb.

2. **RUN**: split long or complex RUN statements on multiple lines separated

```
RUN command-1 \
    command-2 \
    command-3
```

3. Avoid **RUN apt-get upgrade** or **dist-upgrade**
   as many of the “essential” packages from the base images
   won’t upgrade inside an unprivileged container.

Best Practices

Dockerfile: Instructions

4. **RUN apt-get update**

- **CACHE BUSTING**: Always combine *RUN apt-get update && apt-get install -y ….* Using *apt-get update* alone in a *RUN* statement causes caching issues and subsequent apt-get install instructions fail.

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y curl

FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y curl ngnix
```

- **VERSION PINNING:** forces the build to retrieve a particular version regardless of what's in the cache. new versions cause a cache bust of *apt-get update*.

```
RUN apt-get update && apt-get install
    package-foo=1.3.* \
    s3cmd=1.1.* \
```

Best Practices

Dockerfile: Instructions

5. **CMD**

- alway use this format:

```
CMD ["executable", "param1", "param2"…]
CMD ["apache2","-DFOREGROUND"]
CMD ["perl", "-de0"]
CMD ["python"]
CMD ["php", "-a"]
```

- do only rarely use *CMD ["param", "param"]* in conjunction with *ENTRYPOINT* unless you/your users are familiar with *ENTRYPOINT*

Best Practices

Dockerfile: Instructions

6. **EXPOSE**

- use the common, traditional port for your application, e.g.

```
EXPOSE 80 # Apache
EXPOSE 27017 # MongoDB
```

- For container linking, Docker provides environment variables for the path from the recipient container back to the source (ie, MYSQL_PORT_3306_TCP)

Best Practices

Dockerfile: Instructions

7. **ENV**

- Update path to ensure commands work:

```
ENV PATH /usr/local/nginx/bin:$PATH
CMD [“nginx”]
```

- Provide needed env vars for services eg. Postgres PGDATA.
- Use for version numbers and pathes (like constant vars):

```
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz \
    | tar -xJC /usr/src/postgress && …
```

Best Practices

Dockerfile: Instructions

8. **COPY**

- **Prefer COPY** more transparent than ADD
- COPY **only supports the basic copying** of local files into the container, while ADD has some features (like local-only tar extraction and remote URL support) that are not immediately obvious.
- **FEWER CACHE INVALIDATIONS:** Reuse multiple COPY steps individually. Ensures that each step's build cache is only invalidated (forcing the step to be re-run) if a file changes.

```
COPY requirements.txt /tmp/
RUN pip install --requirement /tmp/requirements.txt
COPY . /tmp/
```

Best Practices

Dockerfile: Instructions

9. **ADD**

- **TAR AUTO-EXTRACTION:**

```
ADD rootfs.tar.xz /
```

- Because image size matters, **using ADD** to fetch packages from remote URLs is **strongly discouraged.** You should use curl or wget!

```
# Bad
ADD http://example.com/big.tar.xz /usr/src/things/
RUN tar -xJf /usr/src/things/big.tar.xz -C /usr/src/things
RUN make -C /usr/src/things all

# Good
RUN mkdir -p /usr/src/things \
    && curl -SL http://example.com/big.tar.xz \
    | tar -xJC /usr/src/things \
    && make -C /usr/src/things all
```

Best Practices

Dockerfile: Own experience

1. Use fixed version for base image
2. Prefer offical base images
3. Use gosu for easy-stepdown from root
4. Define own entrypoint if needed
5. Write integration tests, e.g. with CircleCI

Best Practices

Some helpful clean up commands

```
# Kill all running containers
docker kill $(docker ps -q)

# Delete all stopped containers (including data-only containers)
docker rm $(docker ps -a -q)

# Remove all containers:
docker rm -f $(docker ps -a -q)

# Remove dangling images:
docker rmi $(docker images -q -f dangling=true)

# Remove all images
docker rmi $(docker images -q)

# Remove all unused images, not just dangling ones
docker image prune -a -f

# Remove all unused containers, volumes, networks and images
# e.g. both dangling and unreferenced
docker system prune

# Remove all stopped containers
docker container prune
```

Advanced Dockerfile

1. Finalise Dockerfile and build Image
2. Start 2 instances of the flask-test-app
    1. Name them differently
    2. Do not expose any ports!
3. Run nginx container
    1. Hand in all necessary environment variables
    2. Add also the DEBUG variable
    3. Does http://localhost:<port> work?

How can the environment variables be inserted easier?

→ **use --env-file instead of -e**

Effects of ENTRYPOINT

1. Try to start and immediately step into the container
    1. What happens?
2. What advantages do we have with such a solution?

Use custom ENTRYPOINT

1. Extend entrypoint script to have default options but use custom options if provided
2. docker run -it --rm --name nginx -p 80:80 --env-file envfile bastianklein/nginx-test-app "-g daemon off;"

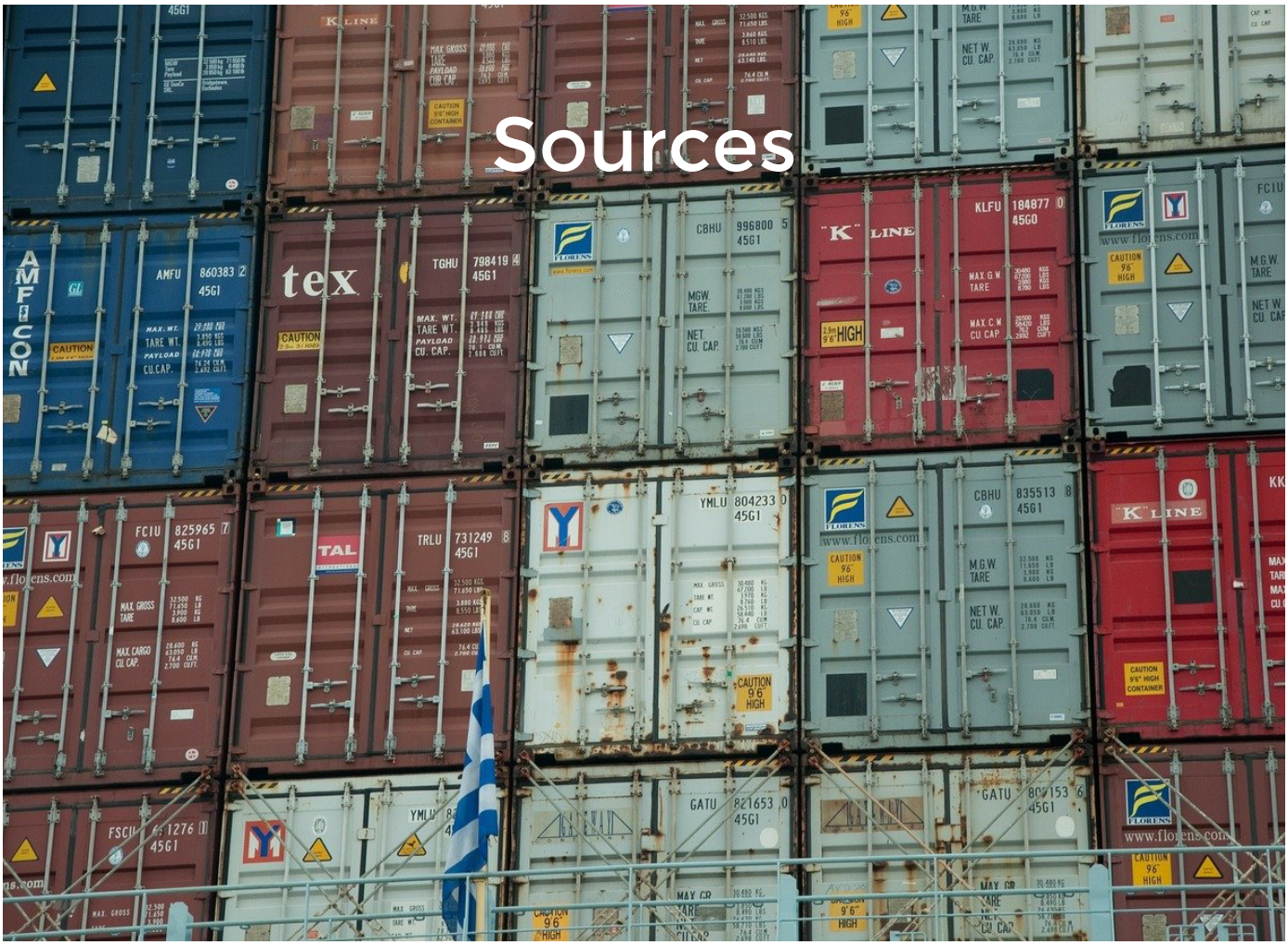
Creating an isolated network

1. Create a new network with driver bridge
2. Start 2 instances of the flask-test-app
    1. Name them differently
    2. Do not expose any ports!
    3. Use the newly created network
3. Run nginx container
    1. Hand in all necessary environment variables
    2. Add also the DEBUG variable
    3. Use the newly created network

Summary
Professional Training
Day 1

Lessons learned

- Running containers
- Writing Dockerfiles
- Inspecting images and containers
- Overriding entrypoint script
- Best practises
- Templating
- Docker network



Sources

Docker

- https://docs.docker.com
- https://docs.docker.com/engine/reference/commandline
- https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices

Examples

- https://github.com/dataduke/gs-spring-boot-docker
- https://github.com/dataduke/logstash-demo