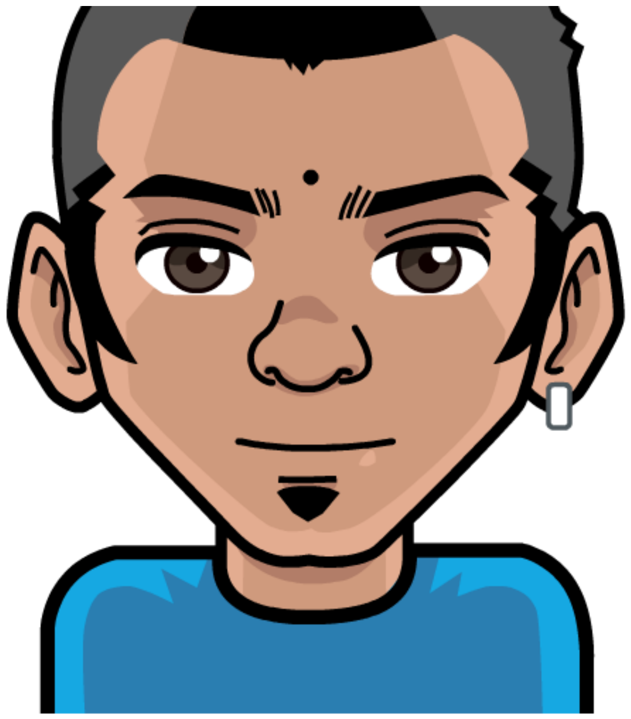
Raju Gandhi

PRACTICAL DOCKER



RAJU GANDHI

@LOOSELYTYPED

FOUNDER - DEFMACRO SOFTWARE

(POLL - Single Choice)
Embracing DevOps

- We do it all — Ci/Cd pipelines, Everything-as-Code, Centralized log/event/monitoring etc
- We have a few pieces in the works
- Just getting started

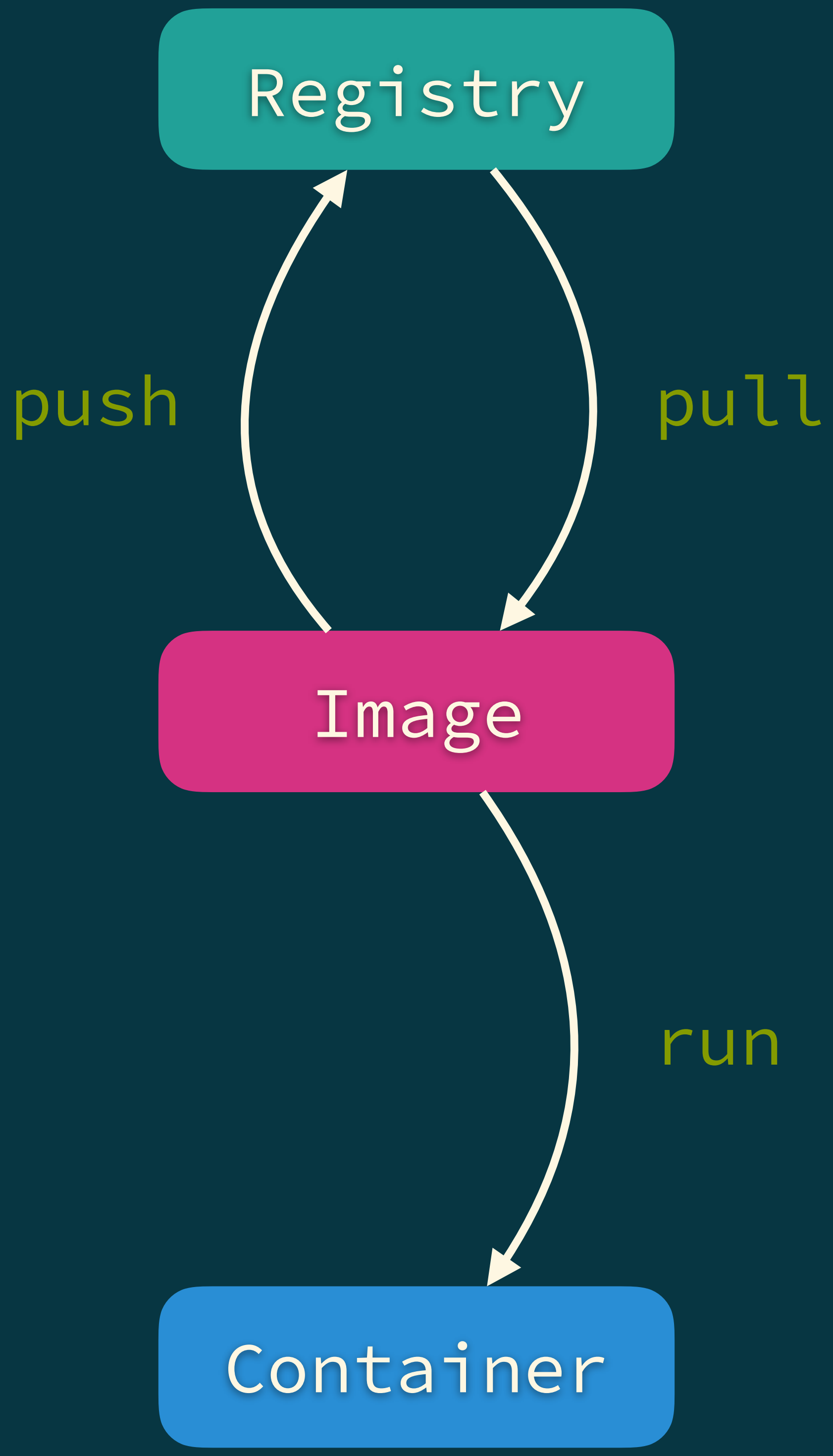
WHY?

BUILD ONCE, RUN ANYWHERE

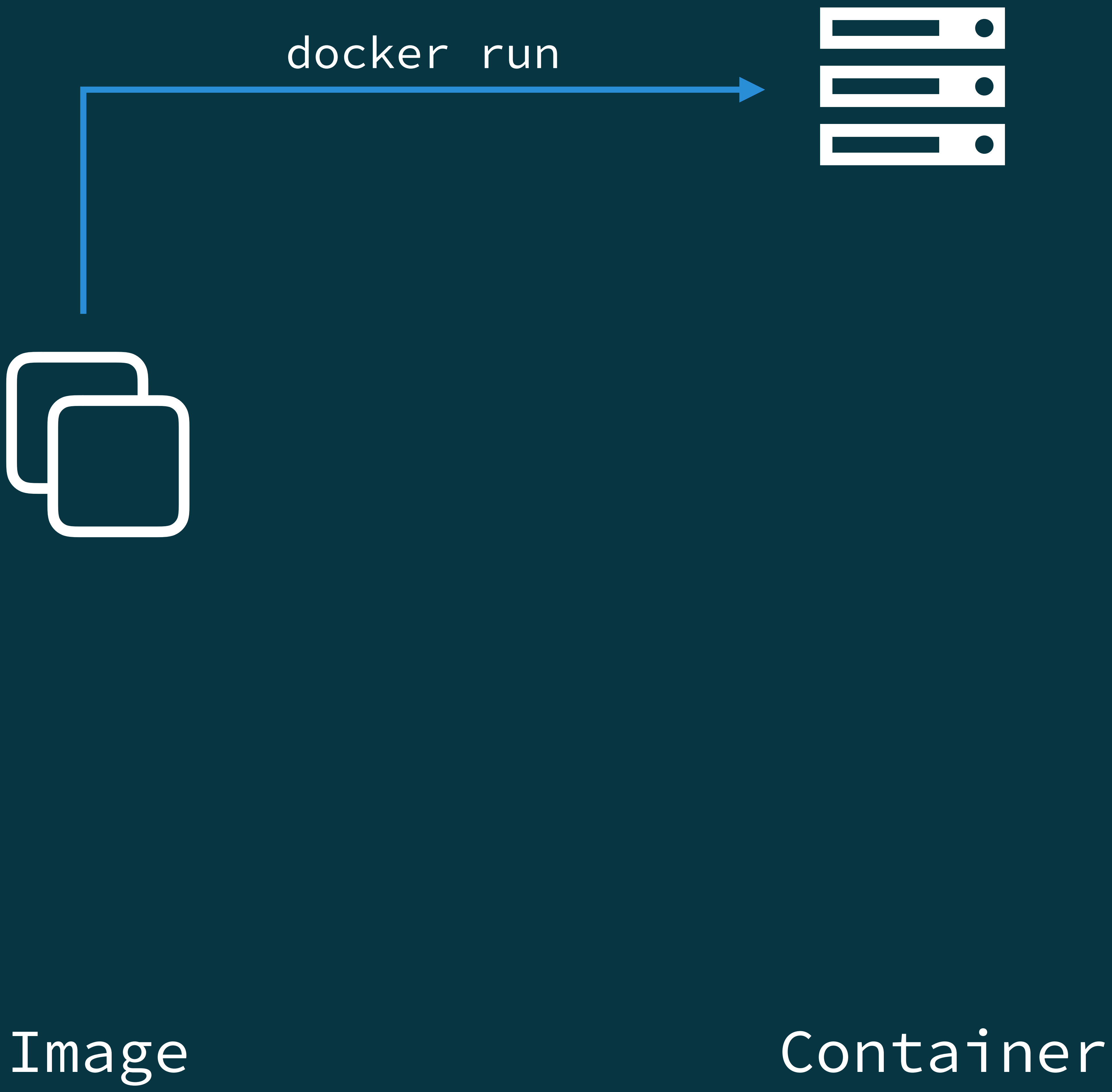
WHY?

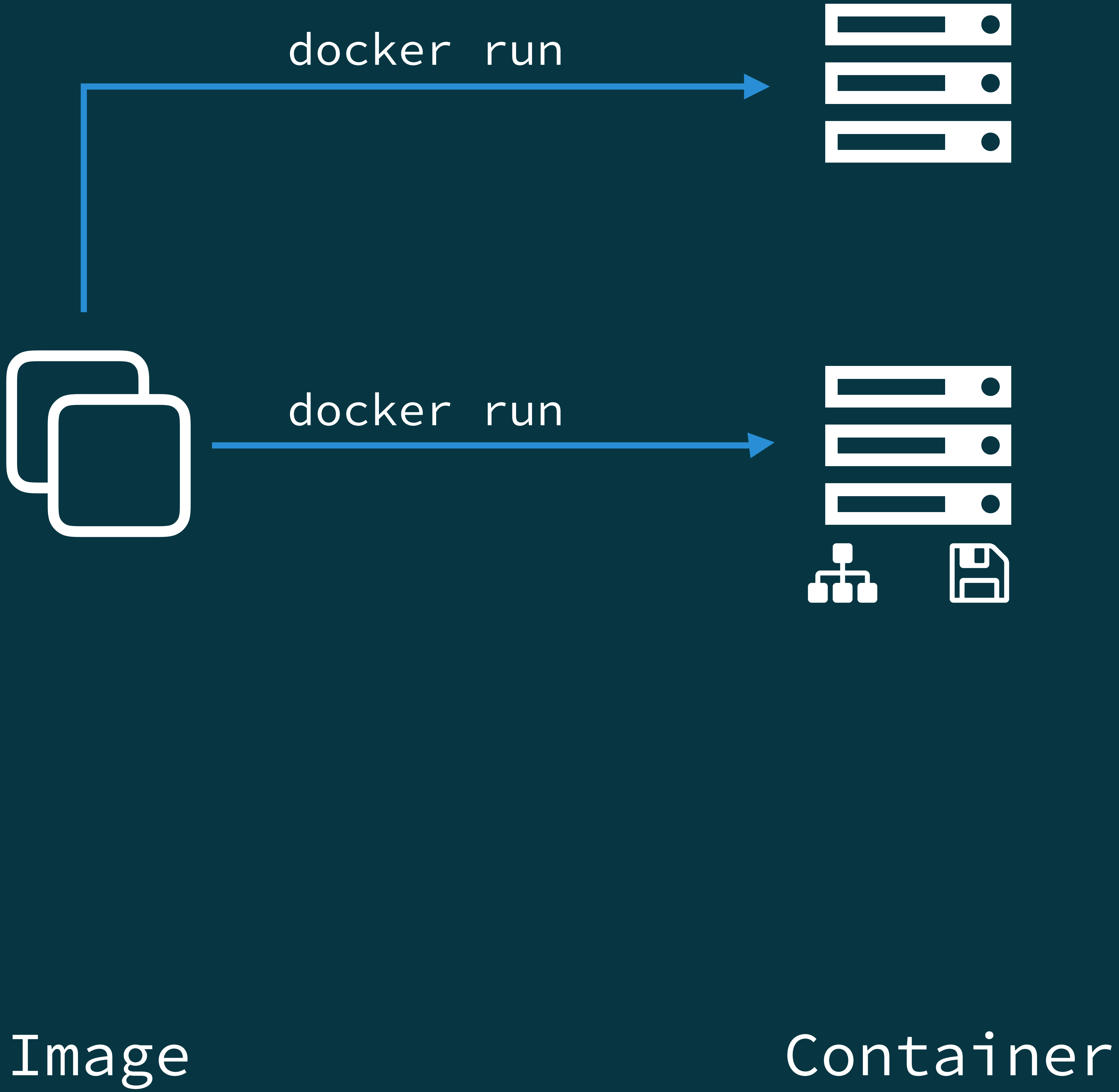
- Local application development and testing
- Team (and OSS) collaboration
- Ci/Cd
- Higher density deployments
- Deterministic

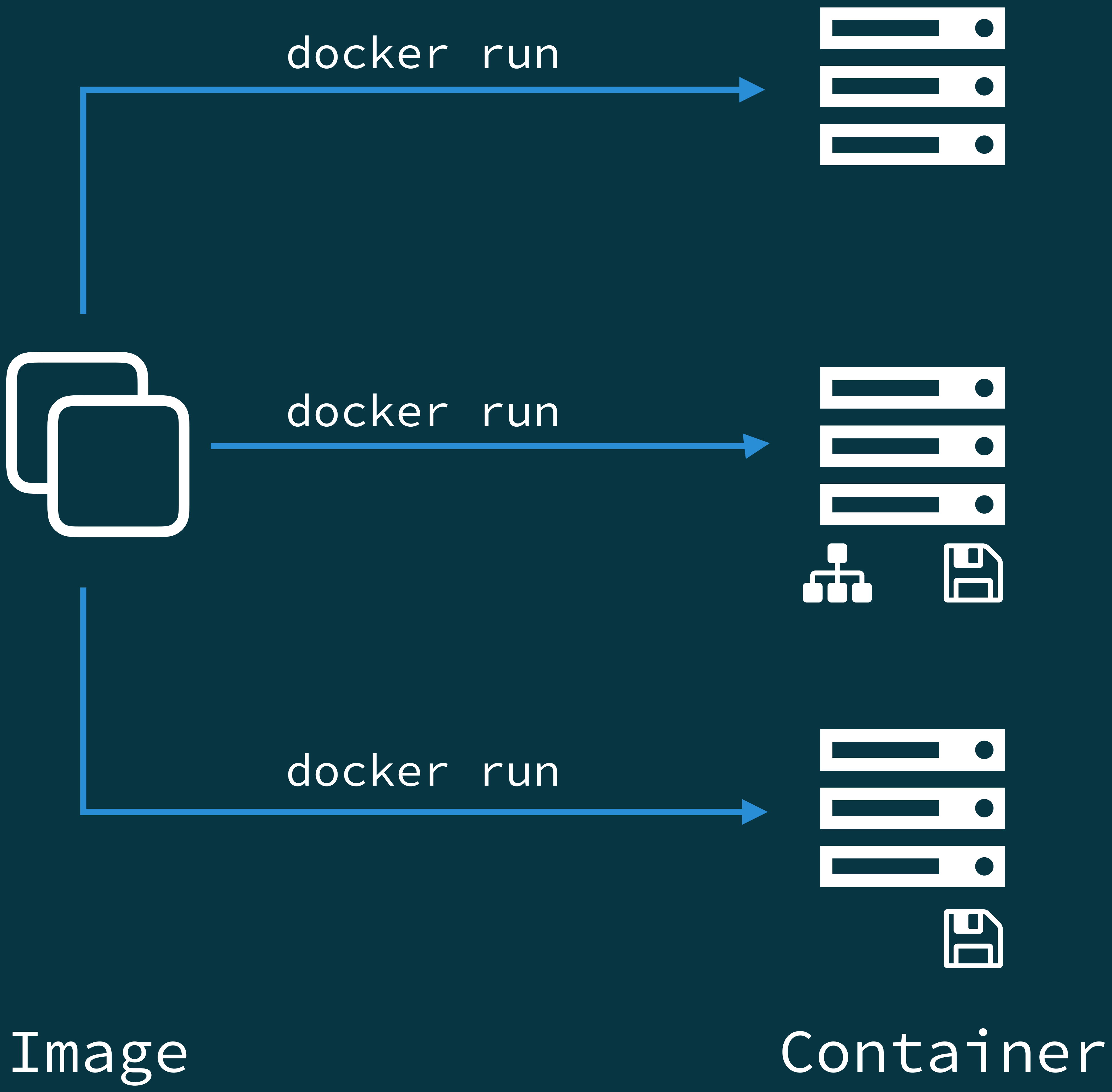
IMAGES


Registry
push
pull
Image
run
Container

Image


docker run
Image
Container


docker run
docker run
Image
Container


docker run
docker run
docker run
Image
Container

IMAGES

- The docker artifact
- Opaque
- Shared using a registry like http://hub.docker.com/

IMAGES

- The "template" for containers
  - Figure out what you need "fixed" a.k.a "compile" time settings
  - Figure out what you want the consumer to specify a.k.a "runtime"

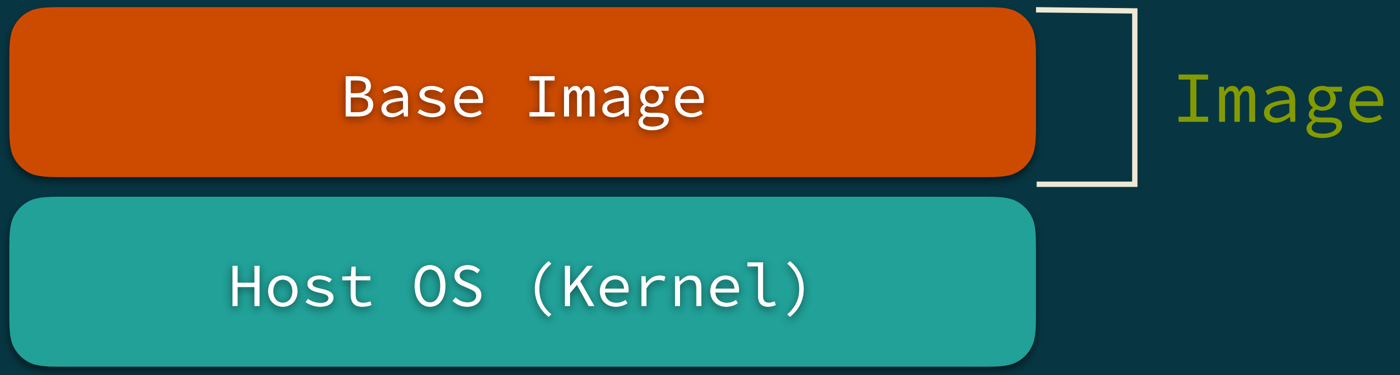
(POLL - Single Choice)
Docker usage

- Everywhere we can! — dev/qa/prod/Ci-Cd pipelines
- Not all the way to production yet (only development or lower tiers)
- Just testing the waters
- Not there yet

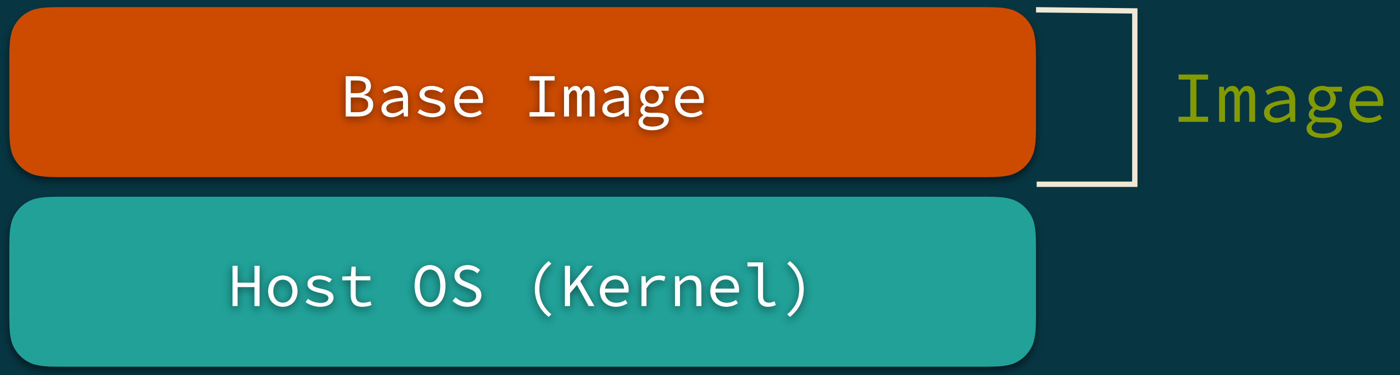
WHAT IS A CONTAINER?

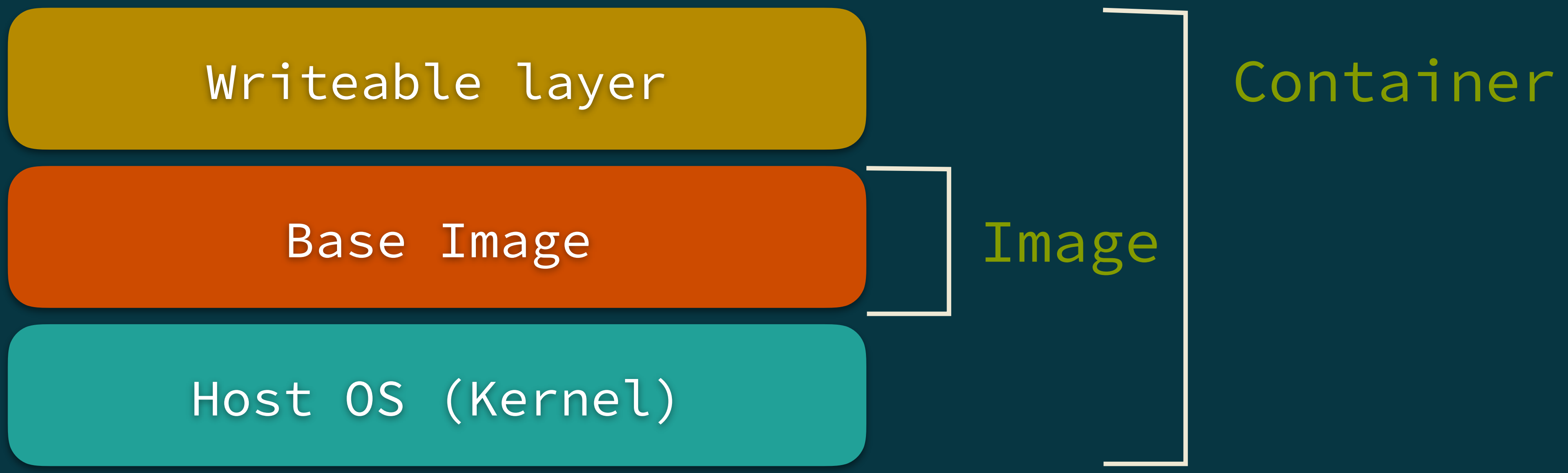
WHAT IS A CONTAINER

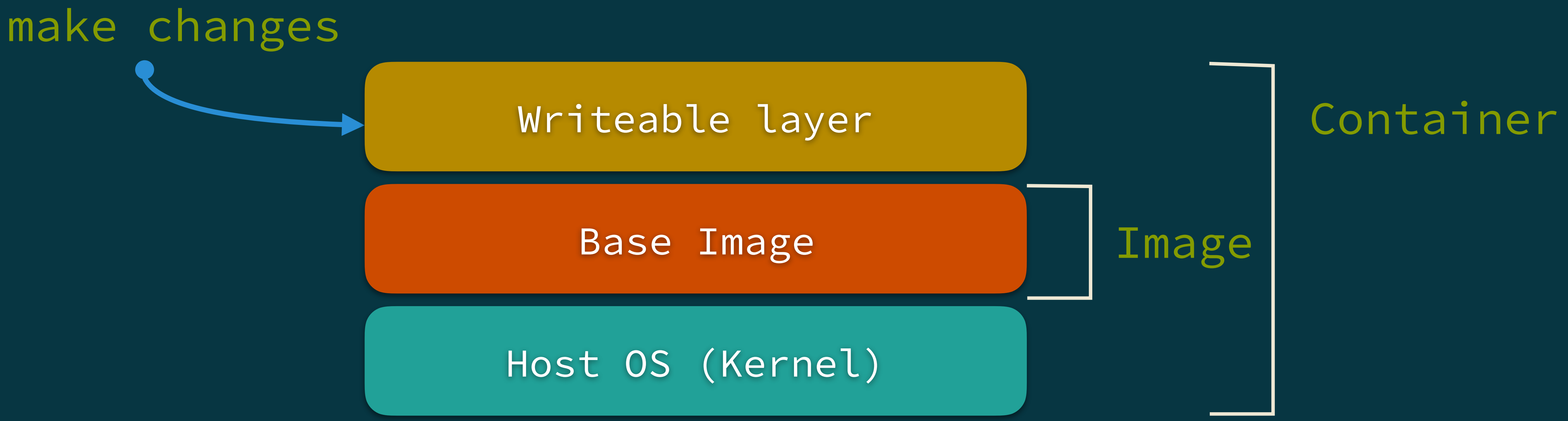
- An instance of an image
- With a r/w layer on top
- Configured with resource limits (cpu/memory), network settings and volume mounts etc on "create"

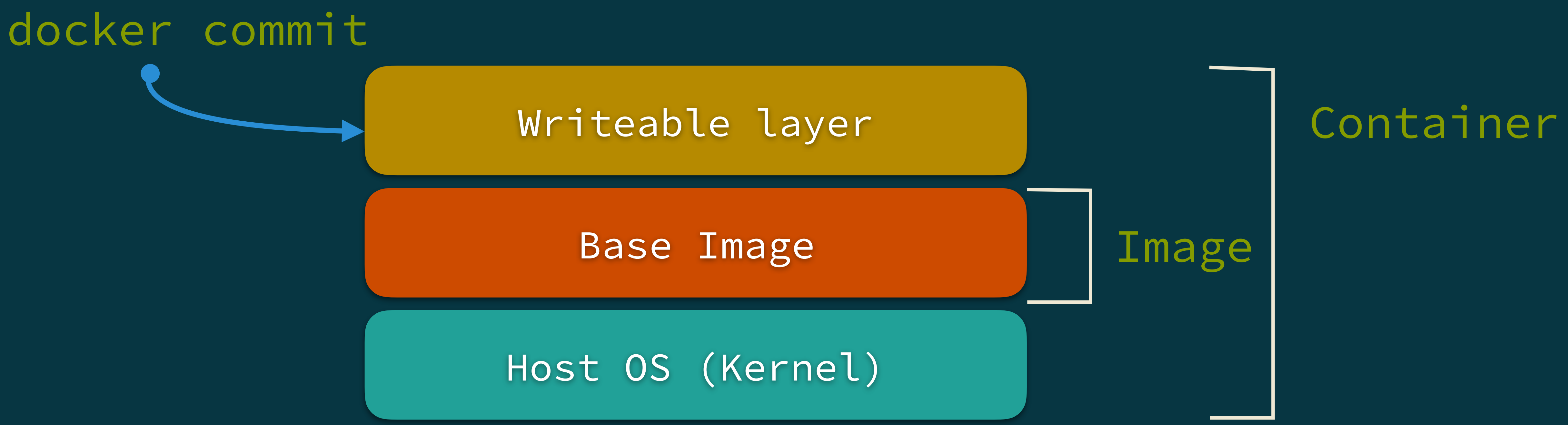

Base Image
Image
Host OS (Kernel)

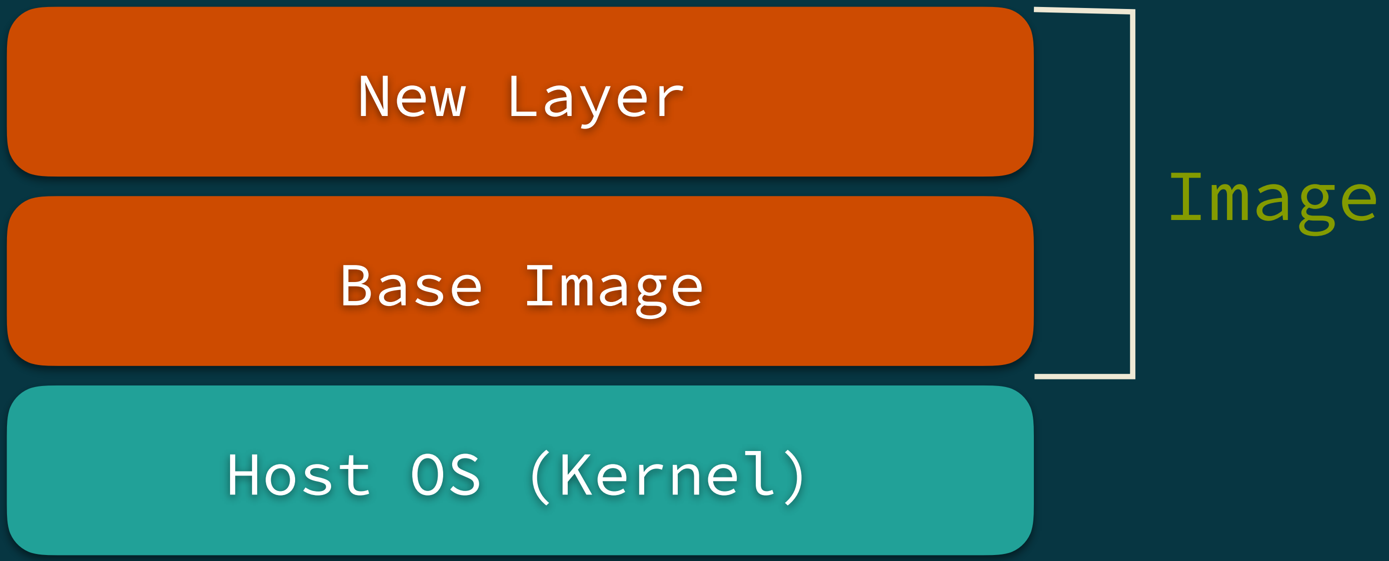
`docker run`



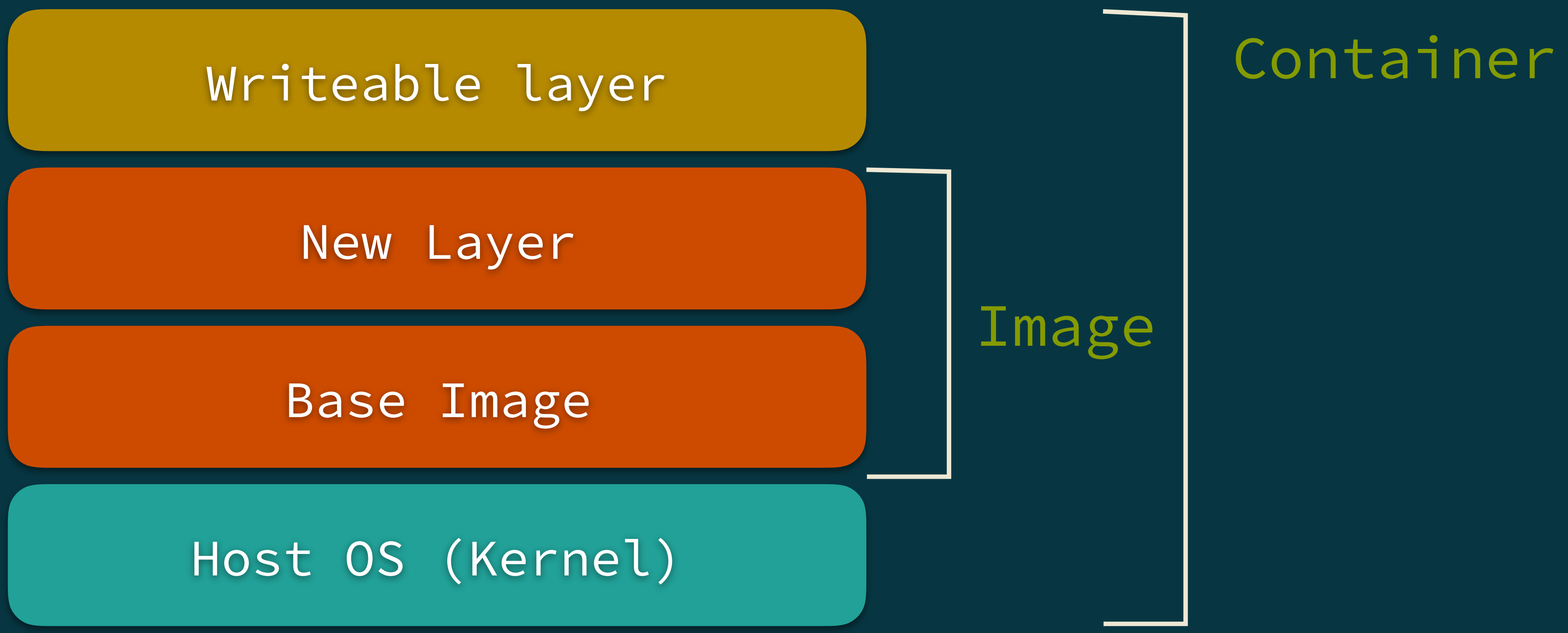

Writeable layer
Base Image
Host OS (Kernel)
Image
Container

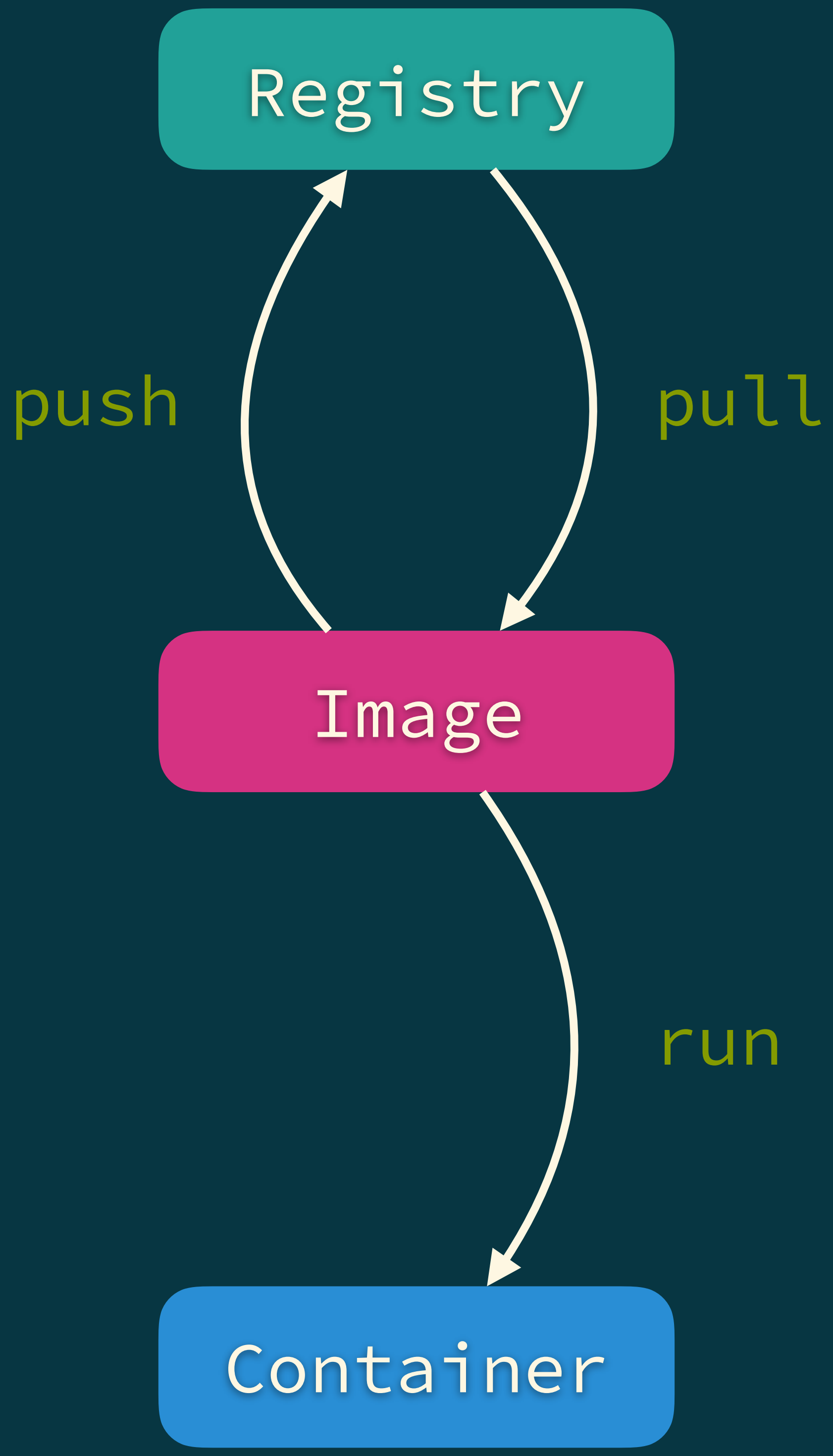

make changes
Writeable layer
Base Image
Host OS (Kernel)
Image
Container

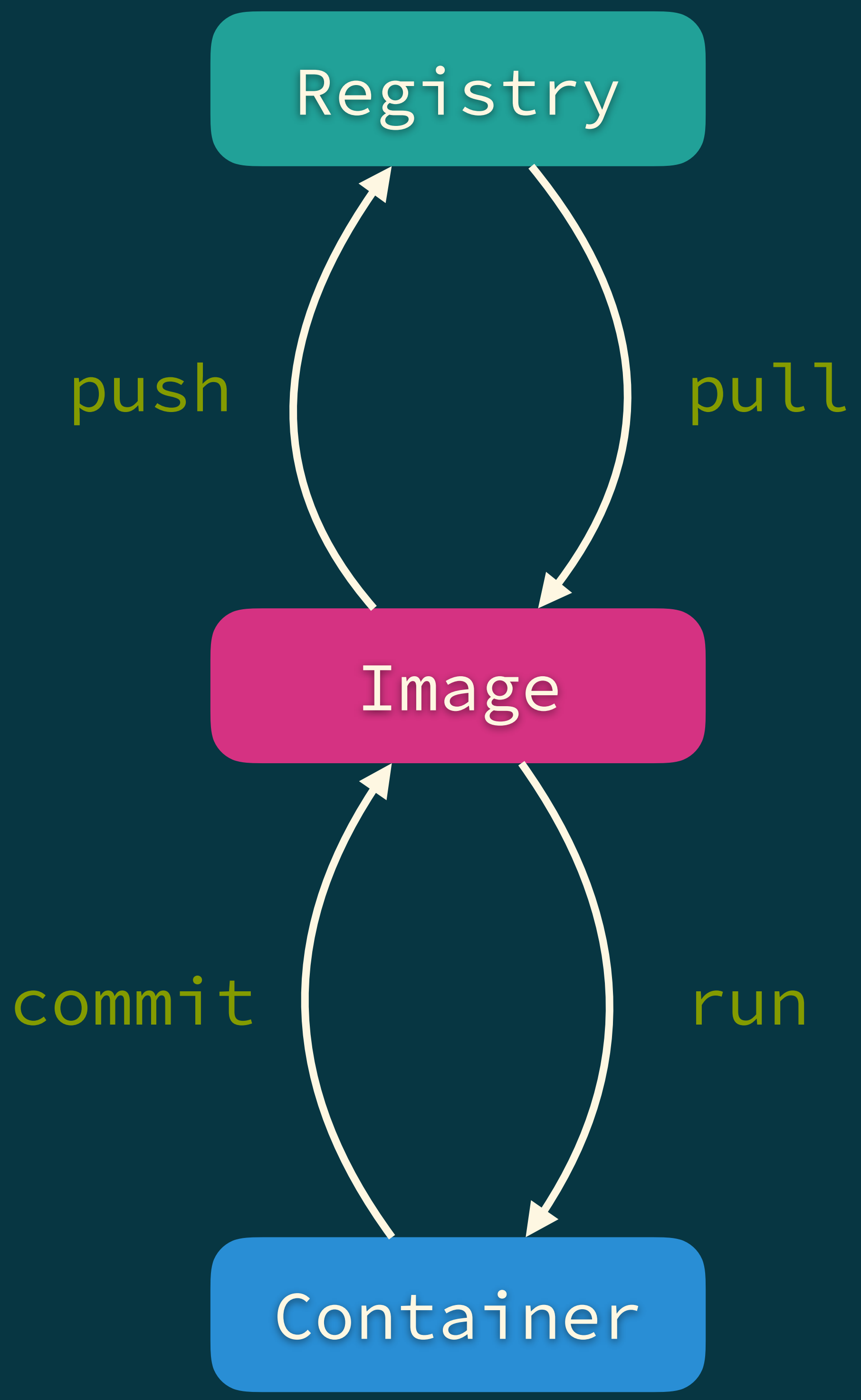

docker commit
Writeable layer
Base Image
Host OS (Kernel)
Image
Container

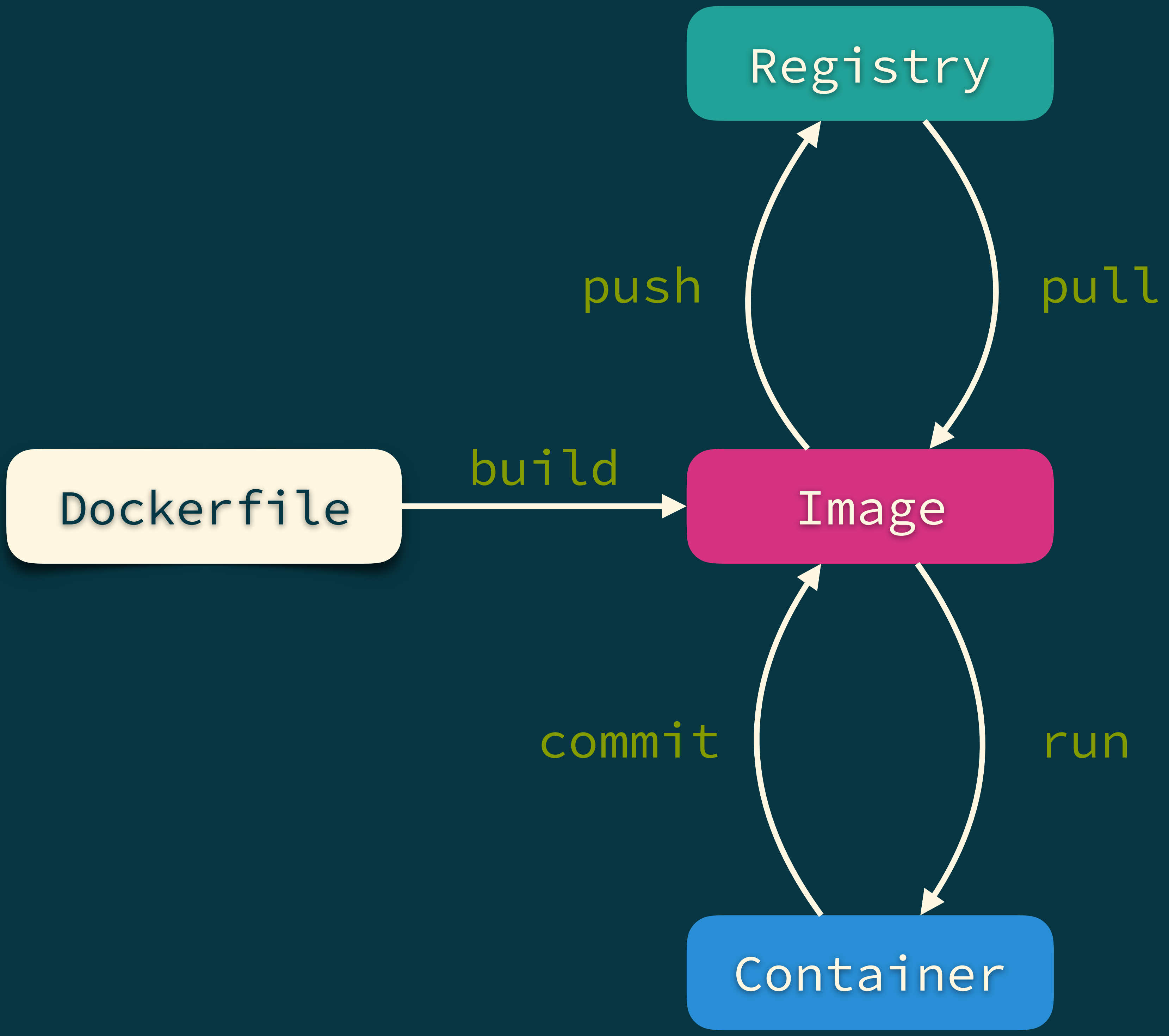

New Layer
Base Image
Host OS (Kernel)
Image

`docker run`


Writeable layer
New Layer
Base Image
Host OS (Kernel)
Image
Container


Registry
push
pull
Image
run
Container


Registry
push
pull
Image
commit
run
Container


Registry
push
pull
Dockerfile
build
Image
commit
run
Container

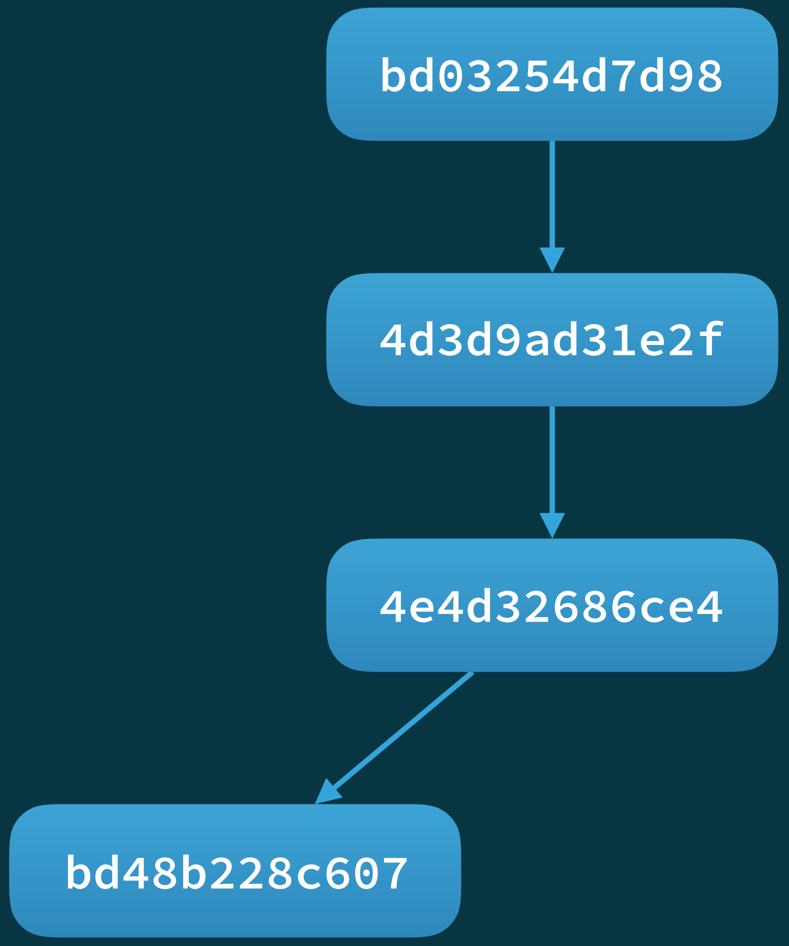
DOCKERFILE

DOCKERFILES

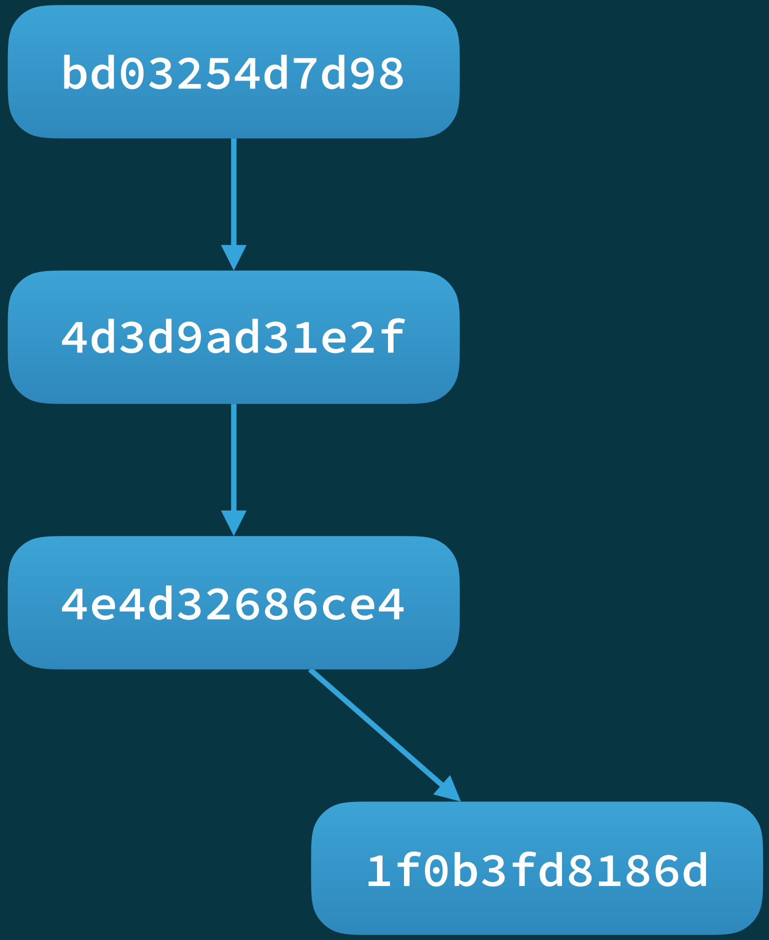
- A set of instructions to build a Docker image
- Plain text, version controlled
- Provides insight into the image needs/capabilities/intents

```
# sample Dockerfile
FROM openjdk:8u131-jre

RUN apt-get update \
  && apt-get install -y netcat

COPY build/libs/app-fat.jar /var/app.jar

CMD ["java", "-jar", "/var/app.jar"]
```

UNION FILE SYSTEM



```
FROM openjdk:8u131-jre

RUN apt-get update \
    && apt-get install -y netcat

COPY build/libs/app-fat.jar /var/app.jar

CMD ["java", "-jar", "/var/app.jar"]
```



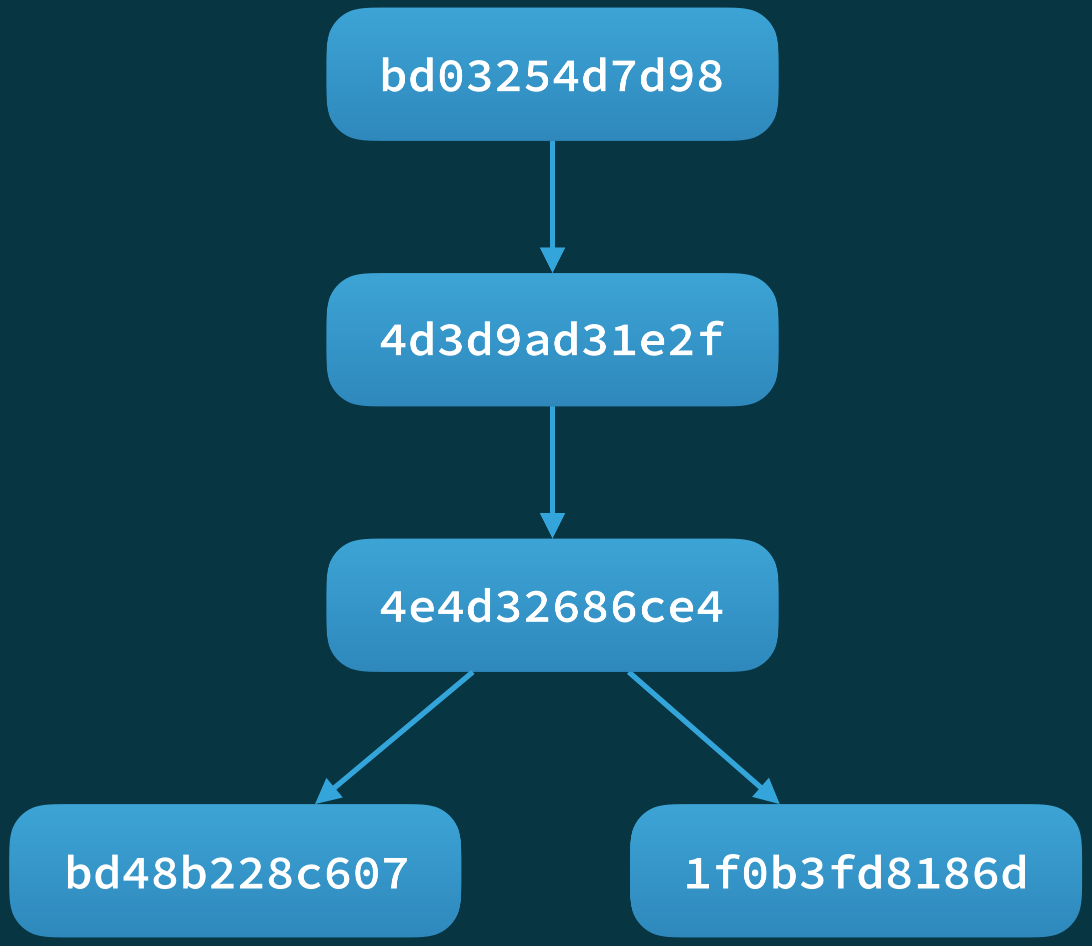
```
FROM openjdk:8u131-jre

RUN apt-get update \
    && apt-get install -y netcat

COPY build/libs/app-fat.jar /var/app.jar

CMD ["ls", “-al"]
```

```bash
#!/usr/bin/env bash

last=alpine:3.8
for i in `seq 200`; do
  rm -f cid
  docker run --cidfile=cid $last touch file$i;
  docker commit `cat cid` tag$i;
  docker rm `cat cid`;
  last=tag$i;
done
```

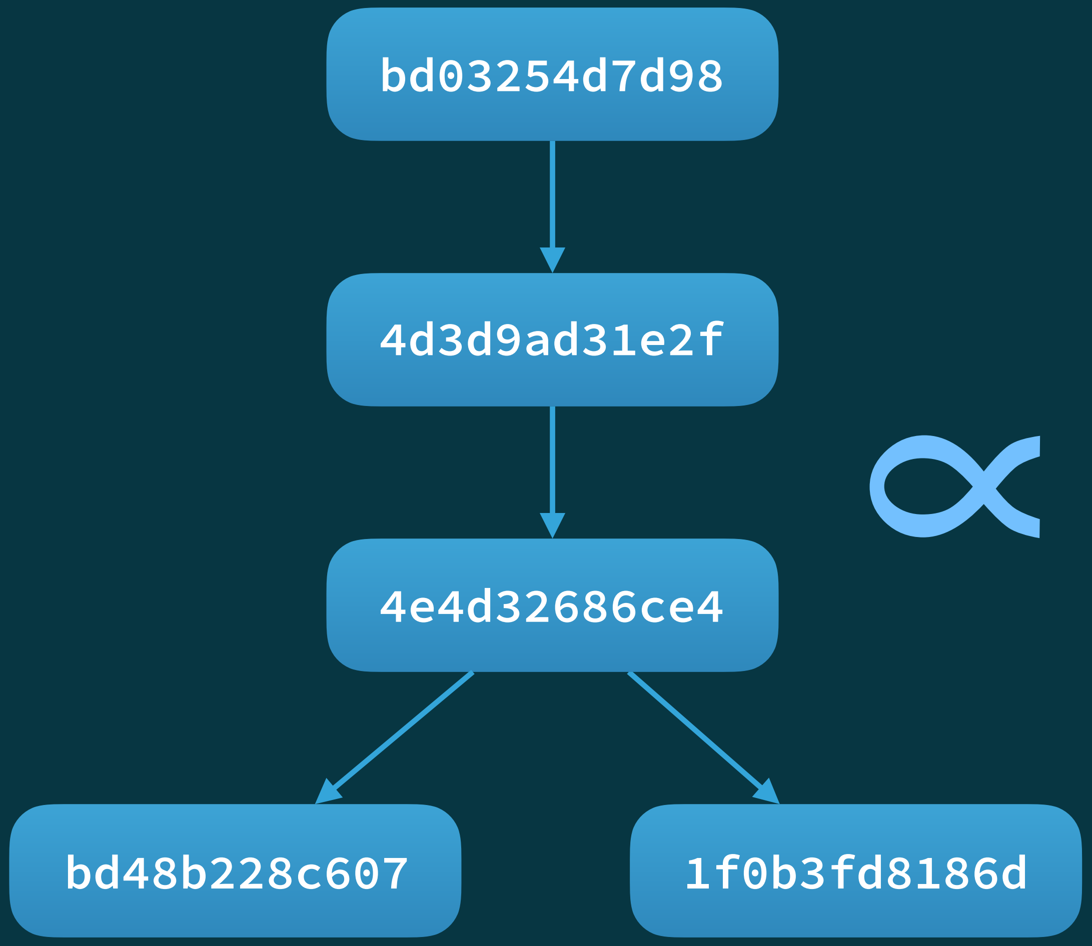
```bash
#!/usr/bin/env bash

last=alpine:3.8
for i in `seq 200`; do
  rm -f cid
  docker run --cidfile=cid $last touch file$i;
  docker commit `cat cid` tag$i;
  docker rm `cat cid`;
  last=tag$i;
done

# Error response from daemon: max depth exceeded
# Unable to find image 'tag125:latest' locally
```

```
FROM alpine:3.8

RUN touch file1
RUN touch file2
# 125 times more
RUN touch file127
```

```
FROM alpine:3.8

RUN touch file1
RUN touch file2
# 125 times more
RUN touch file127

# Error response from daemon: max depth exceeded
```



```
FROM openjdk:8u131-jre

RUN apt-get update \
    && apt-get install -y netcat

COPY build/libs/app-fat.jar /var/app.jar

CMD ...
```



$\propto$

```
FROM openjdk:8u131-jre

RUN apt-get update \
    && apt-get install -y netcat

COPY build/libs/app-fat.jar /var/app.jar

CMD ...
```

ONE CONCERN PER CONTAINER

@LOOSELYTYPED

(POLL - Single Choice)
Base image usage

- We try and use as much as we can from public registries
- Outside of a handful of base images we build everything in-house
- Not concerned about this (yet)

FROM

NOTES

- Implies “ancestry”
- **Has** to be the first line (Except if preceded by `ARG`)
- Has implications on `WORKDIR`, `USER`, `ENTRYPOINT` (and `CMD`), and `ONBUILD`, `EXPOSE` and other commands
    - Use "`docker inspect`" for this
- Create a base image with `FROM scratch`

DO'S

- Pin down the exact tag (or even better the digest)
  - Do not use "latest" tag
- Choose your parent image wisely
  - Vet it!
  - Inspect ancestor images for `USER`s, `PORT`s, `ENV`s, `VOLUME`s, `LABEL`s and anything that can be inherited
- Most likely you will build the lineage yourself internal to your team and organization

```
# Don't
FROM alpine

# Do
# Pin the version
FROM alpine:3.8
# OR Use ARG to set it at build time
ARG version=3.8
FROM alpine:${version}
```

ARG

DO'S

- Use `ARG` for tweaking the build dynamically
  - Use them to set `FROM`, `ENV`, `LABEL`, `RUN`
- Default them appropriately

```
# Do
# Use default value
ARG AUTHOR="Raju Gandhi"
ARG BUILD_DATE
ARG VCS_REF
```

```
# default author
docker build --build-arg -t test .
# set at build time
docker build --build-arg AUTHOR="Solomon Hykes" -t test .
```

ENV

DONT'S

- Put secrets or sensitive information in ENV variables
- Override parent image ENV's unless absolutely necessary

DO'S

✓

- Use them for documentation and modifying runtime behavior
  - They are baked in the final image
- Use `docker run <image-name> env`
  - Or `docker inspect`
- Default then appropriately
- Be cognizant of inherited `ENV` variables

```
ARG PROJECT_VERSION
# Do
# Default them if set dynamically
ENV PROJECT_VERSION ${PROJECT_VERSION:-2.3}
```

LABEL

DONT'S

- Define individual labels separately

DO'S

- Use them liberally
- Labels can see ARG variables. Use this!
    - BUILD_NUMBER, GIT_SHA
- Apply a standard convention
    - Build tooling on top of the conventions

```
ARG AUTHOR="Raju Gandhi"
ARG BUILD_DATE
ARG VCS_REF

# Don't
LABEL org.label-schema.author=$AUTHOR
LABEL org.label-schema.build-date=$BUILD_DATE
LABEL org.label-schema.vcs-ref=$VCS_REF

# Do
LABEL org.label-schema.author=$AUTHOR \
  org.label-schema.build-date=$BUILD_DATE \
  org.label-schema.vcs-ref=$VCS_REF
```

(POLL - Multiple Choice)
Concerns with image size

- Affects our build/runtimes
- Security concerns
- Disk usage for storage (local/registry/servers)
- Not a concern (yet)

RUN

DON'TS

- Be cognizant of the effects (and drawbacks) of caching
- Do not do OS level upgrades (eg. `RUN dist-upgrade`)

DOS

- Group common operations
  - Clean up as well (reduces image sizes)
- Use multiline (\) to make PR / auditing easier

```
# Don't
RUN apt-get update
RUN apt-get install -y netcat
RUN apt-get clean

# Do
RUN apt-get update \
  && apt-get install -y \
  netcat  \
  && apt-get clean
```

ENTRYPOINT/CMD

NOTES

- `CMD` can be overridden if the user supplies an argument to create or run
    - You can (also) override `ENTRYPOINT` by explicitly supplying `--entrypoint` flag
- The default command run in a container is (`ENTRYPOINT + CMD`)

DONT'S

- Avoid the “shell” form

DO'S

- Use the "exec" form
    - Shell expansion will **not** happen!
- Use `ENTRYPOINT` and `CMD` together
- Use a "entrypoint-script"
    - Allows you to set error (-e) flags and traps
    - Your editor is syntax aware
    - Always "exec"

```
# Do
# Use ENTRYPOINT and CMD together
ENTRYPOINT [ "echo", "hello" ]
CMD [ "world" ]

# > docker build -t entrypoint-cmd .
# > docker run entrypoint-cmd
# hello world
# > docker run entrypoint-cmd raju
# hello raju
# Do
COPY entrypoint.sh /usr/local/bin/
ENTRYPOINT ["entrypoint.sh"]
CMD ["default"]


# entrypoint.sh
if [ "$1" = 'default' ]; then
  # do default thing here
  echo "Running default"
else
  echo "Running user supplied arg"
  # if the user supplied say /bin/bash
  exec "$@"
fi
```

HEALTHCHECK

DONT'S

- Be too aggressive with `--interval` period
  - Especially if the check itself is expensive
- Use external tools (like `curl`) if you can

DO’S

- Use a script that leverages the same runtime as your service
    - For example, if you have a node or go service, write a health check using the same runtime
- Be cognizant of the overhead the health check introduces
- Experiment with combinations of interval/timeout/retries

```dockerfile
# Avoid
HEALTHCHECK CMD curl --fail http://localhost:5000/ || exit 1

# Do
COPY healthcheck ./healthcheck
HEALTHCHECK --interval=1s \
  --timeout=1s \
  --start-period=2s \
  --retries=3 CMD [ "/healthcheck" ]
```

ADD/COPY

.DOCKERIGNORE

```
# example .dockerignore file
# ignore these folders
.git
build
!build/libs/*.jar # but NOT this

# ignore these files
.project
.gitignore
.dockerignore

# ignore all Docker files
Dockerfile*
docker-compose.yml

# ignore all markdown files (md) besides README.md
*.md
```

DON'TS

- Avoid ADD
- Do not leave “residual” artifacts
- Be wary of using the array syntax
- Copy over all source in one fell swoop
    - Copy over source files separately and later on since they change often

DO’S

✓

- Instead of `ADD`
  - Combine `COPY` and `RUN`
  - OR `RUN` with wget/curl/tar/unzip
  - See DO’S under `RUN`
- Be mindful of what you put in the `.dockerignore` file

```dockerfile
# Don't
# ADD is confusing (unless you want to copy & explode a tar file)
ADD src .
# Avoid the array syntax (if you can)
COPY ["src", "test.sh"]
# Copy over ALL source in one fell swoop
COPY . .
```

USER

DON'TS

- Do not switch USER often
- Avoid using root

DO'S

- Create a user (if you can) for your service
- Default the container to a non-root user if you can

```dockerfile
FROM ubuntu:18.10

# Do
RUN groupadd -r app \
  && useradd -r -g app appuser
USER appuser
```

EXPOSE

DON'TS

- Avoid "`docker run -P`"

DO'S

- Do document the ports your application needs exposed

COMPILE VS RUNTIME

```
FROM alpine:3.8

RUN apk add --update \
  tzdata \
  && rm -rf /var/cache/apk/*

ARG TZ=America/Los_Angeles
RUN ln -snf \
  /usr/share/zoneinfo/$TZ \
  /etc/localtime && echo $TZ > /etc/timezone
```

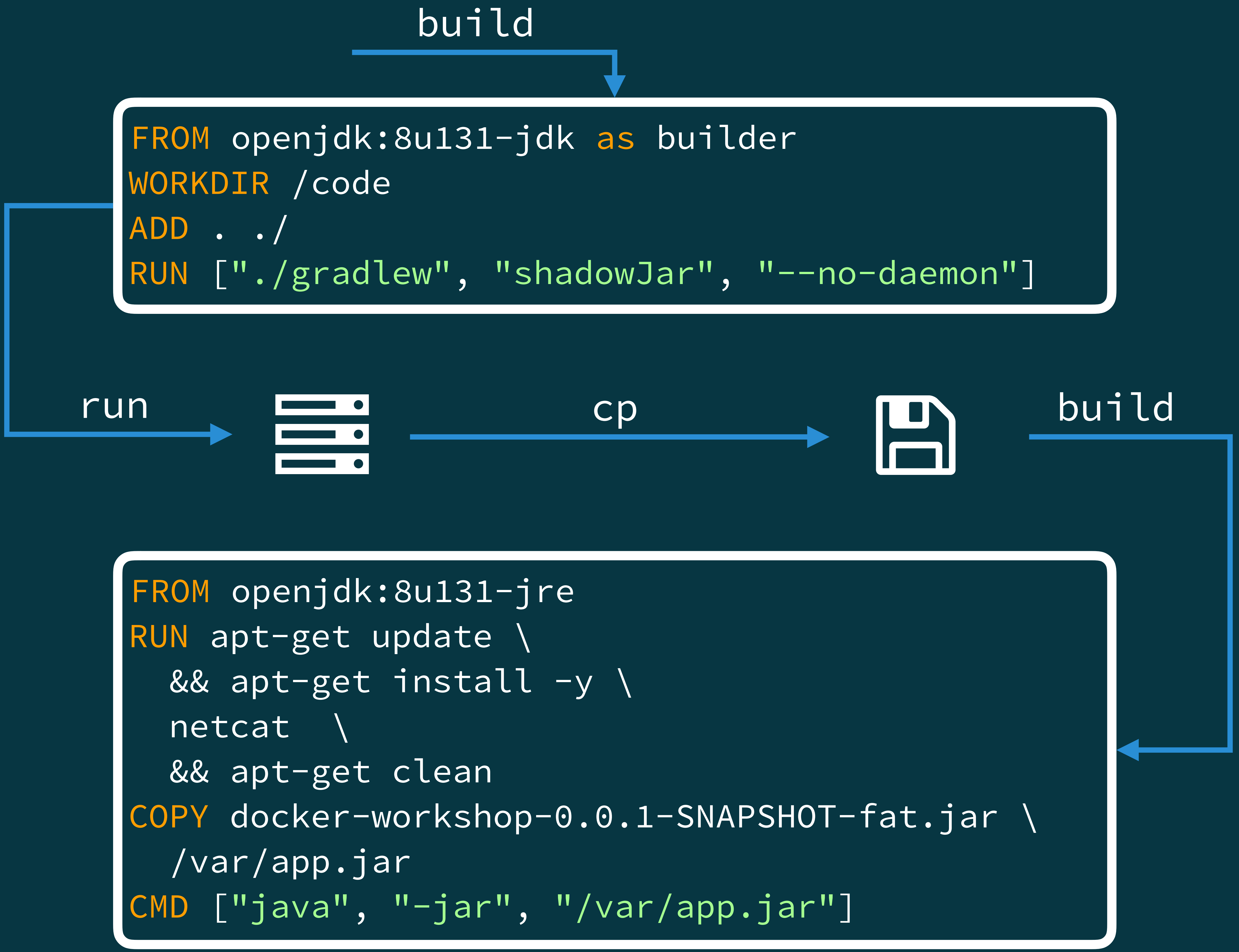
```
FROM alpine:3.8

RUN apk add --update \
  tzdata \
  && rm -rf /var/cache/apk/*

ARG TZ=Americal/Los_Angeles
ENV TZ $TZ

COPY entrypoint.sh /usr/local/bin/
ENTRYPOINT ["entrypoint.sh"]
CMD ["default"]
```

```
#!/bin/sh
ln -snf \
  /usr/share/zoneinfo/$TZ \
  /etc/localtime && echo $TZ > /etc/timezone
```

DO'S

- Use a combination of ARG and ENV with ENTRYPOINT/ CMD
    - Allows you to express what is at "compile" time versus "runtime"
- ARG/ENV are important parts of your image and containers API

MULTI-STAGE BUILDS



```
FROM openjdk:8u131-jdk as builder
WORKDIR /code
ADD . ./
RUN ["./gradlew", "shadowJar", "--no-daemon"]
```

```
FROM openjdk:8u131-jre
RUN apt-get update \
  && apt-get install -y \
  netcat  \
  && apt-get clean
COPY docker-workshop-0.0.1-SNAPSHOT-fat.jar \
  /var/app.jar
CMD ["java", "-jar", "/var/app.jar"]
```

NOTES

- Allow you to do everything using docker containers
- Separates build environment from runtime, with the net effect of leaner production images
  - Keep secrets out of production images
  - Image builds are much faster because docker can leverage the cache

```
FROM openjdk:8u131-jdk as builder
WORKDIR /code
ADD . ./
RUN ["./gradlew", "shadowJar", "--no-daemon"]


FROM openjdk:8u131-jre
RUN apt-get update \
  && apt-get install -y \
  netcat  \
  && apt-get clean
EXPOSE 8080
COPY --from=builder \
  /code/build/libs/docker-workshop-0.0.1-SNAPSHOT-fat.jar \
  /var/app.jar
CMD ["java", "-jar", "/var/app.jar"]
```

```
FROM openjdk:8u131-jdk as builder
WORKDIR /code
ADD . ./
RUN ["./gradlew", "shadowJar", "--no-daemon"]


FROM openjdk:8u131-jre
RUN apt-get update \
  && apt-get install -y \
  netcat  \
  && apt-get clean
EXPOSE 8080
COPY --from=builder \
  /code/build/libs/docker-workshop-0.0.1-SNAPSHOT-fat.jar \
  /var/app.jar
CMD ["java", "-jar", "/var/app.jar"]
```

THANKS!!

RESOURCES

- Best practices for writing Dockerfiles
- Docker Registry V2
- Explaining Docker Image IDs
- Dockerfiles for reference
    - redis
    - Jenkins
    - Postgresql