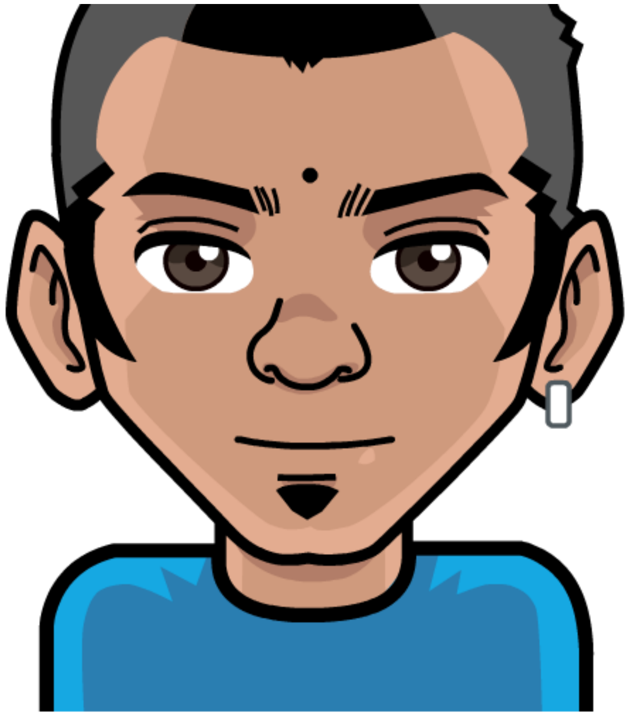
Raju Gandhi

DOCKER WORKSHOP



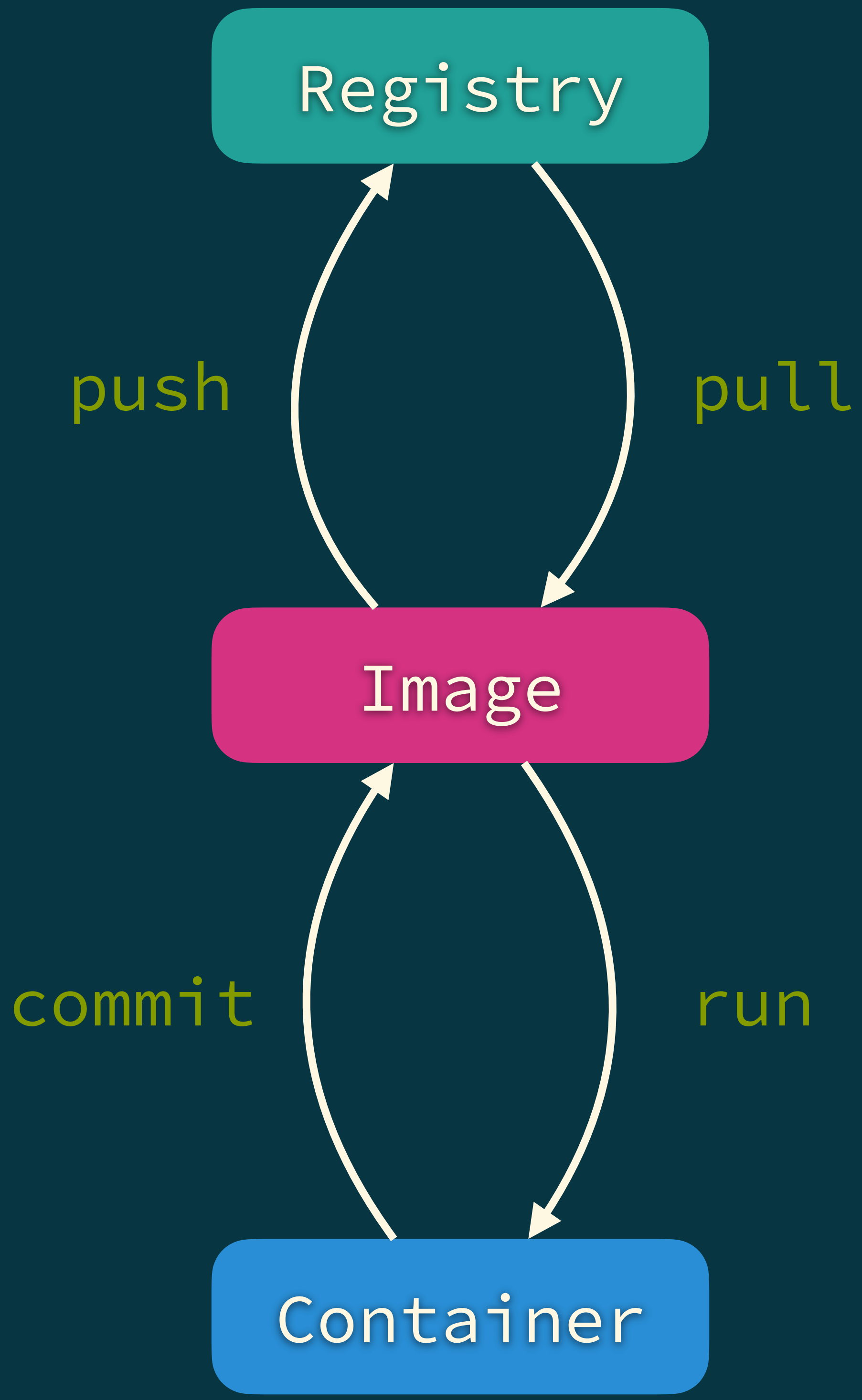
RAJU GANDHI

@LOOSELYTYPED

FOUNDER - DEFMACRO SOFTWARE

WHY?

BUILD ONCE, RUN ANYWHERE



Registry

push

pull

Image

commit

run

Container

WHY?

- Local application development and testing
- Team (and OSS) collaboration
- Ci/Cd

IMAGES

docker image

IMAGES

- The final artifact
- Consider it to be opaque
- Shared using a registry like http://hub.docker.com/

@EXERCISE

EXERCISE

- Figure out what images are cached on your machine
  - Pay attention to what information Docker offers you for every image
- Pull ubuntu:19.10 from hub.docker.com
- Now list the images again and see what changed

HINTS

- `docker help pull;`
- `docker help image; # ask for help`
- `docker image ls --help; # more help!`

HELLO WORLD!

@EXERCISE

EXERCISE

- Run your first ubuntu:19.10 container and make it echo a message

HINTS

- Just for grins “time” it

INTERACTIVE CONTAINERS

```
docker run
```

@EXERCISE

EXERCISE

- Start an interactive bash shell inside a `ubuntu:18.10` container
  - Explore the following
    - Who are you logged in as?
    - What directory are you in?
    - What does the file system look like?
  - Can you `ping www.google.com`?
  - See if you can install a utility, like `iputils-ping`
    - Now can you ping google?

HINTS

- whoami;
- pwd; # print working directory
- ls -al; # listing files
- apt update && apt install -y iputils-ping; # installing items

@EXERCISE

EXERCISE

- Try to start an interactive session using “alpine:3.9”
  - Did it work?
  - Can you `ping www.google.com`?
  - Why is this different than the behavior you saw with ubuntu?

@EXERCISE

EXERCISE

- Try to start an interactive session using “jenkins/jenkins:2.225”
  - Explore the following
    - Who are you logged in as?
    - What directory are you in?
    - What does the file system look like?
  - Change directory to the home directory
    - What do you see?

HINTS

- whoami;
- pwd; # print working directory
- ls -al; # listing files
- cd; # go to home

THE RUNTIME

docker container

@EXERCISE

EXERCISE

- Figure out what containers are running on your machine
- Figure out what was run, but is no longer running
- Remove the containers that are no longer needed

HINTS

- `docker container;`
- `docker help container; # ask for help!`
- `docker container rm --help; # more help`

INTROSPECTION

```
docker exec
```

@EXERCISE

EXERCISE

- Start a “jenkins/jenkins:2.225” interactive container
- In another terminal ask that container of its env variables
- Also see if you can list the processes running within that container
- Shut down the container and delete it

HINTS

- docker help exec;
- env # see env variables
- ps aux # list processes

@EXERCISE

EXERCISE

- Start a “jenkins/jenkins:2.225” in daemon mode
- In another terminal list all the containers running
- Tail the logs of the jenkins container

HINTS

- docker help container;
- docker container ls --help;
- docker logs --help;

```
docker logs
```

@EXERCISE

EXERCISE

- Tail the logs of the Jenkins container you just started

HINTS

- docker help logs;

```
docker inspect
```

@EXERCISE

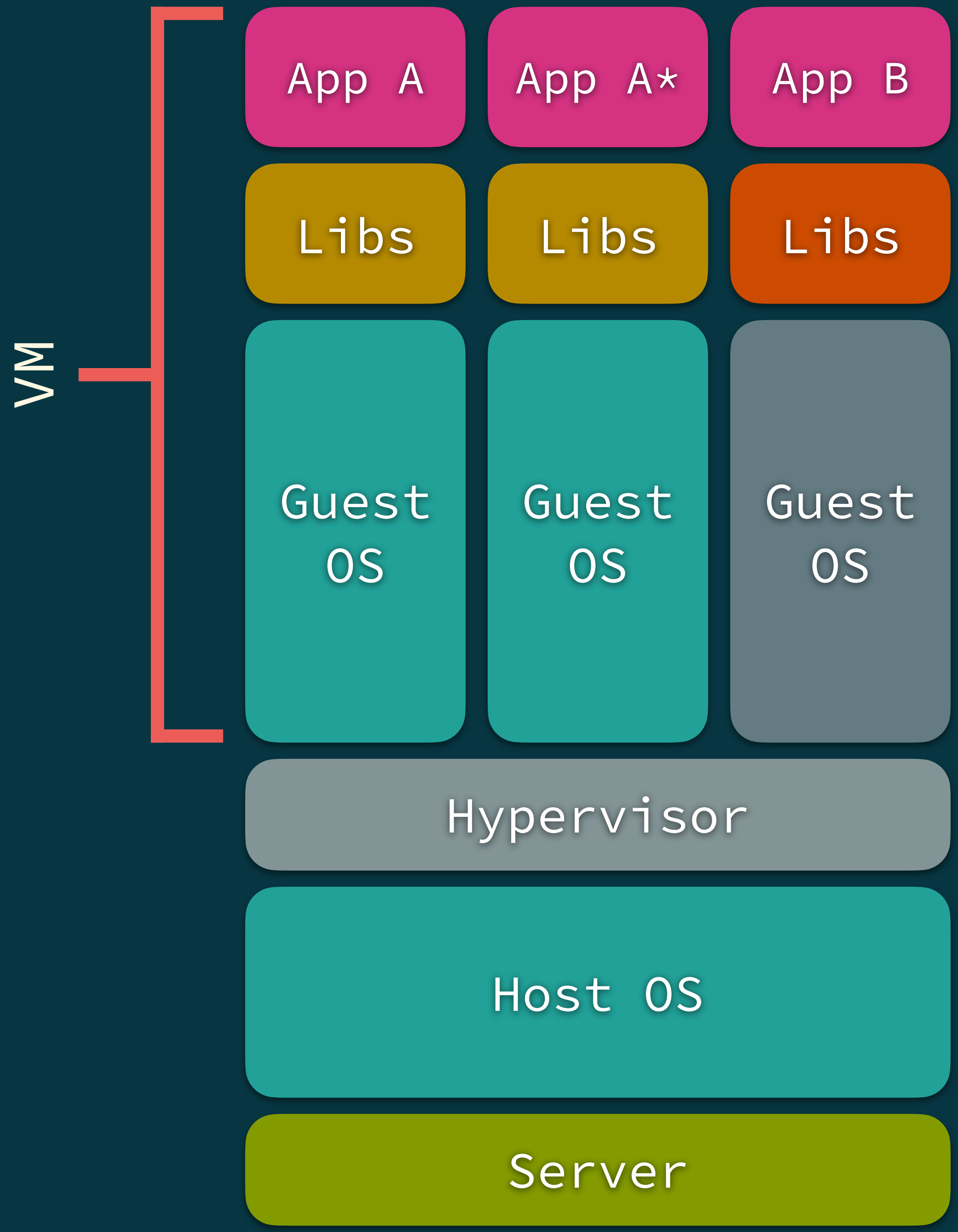
EXERCISE

- Inspect the "jenkins/jenkins:2.225" image
  - Pay attention to the ContainerConfig, Config, and RootFs sections
- Next inspect the container you have running
  - Pay attention to the HostConfig
- Shut down the container

HINTS

- docker image inspect --help;
- docker container inspect --help;

VM? CONTAINERS?



VM

App A

App A*

App B

Libs

Libs

Libs

Guest OS

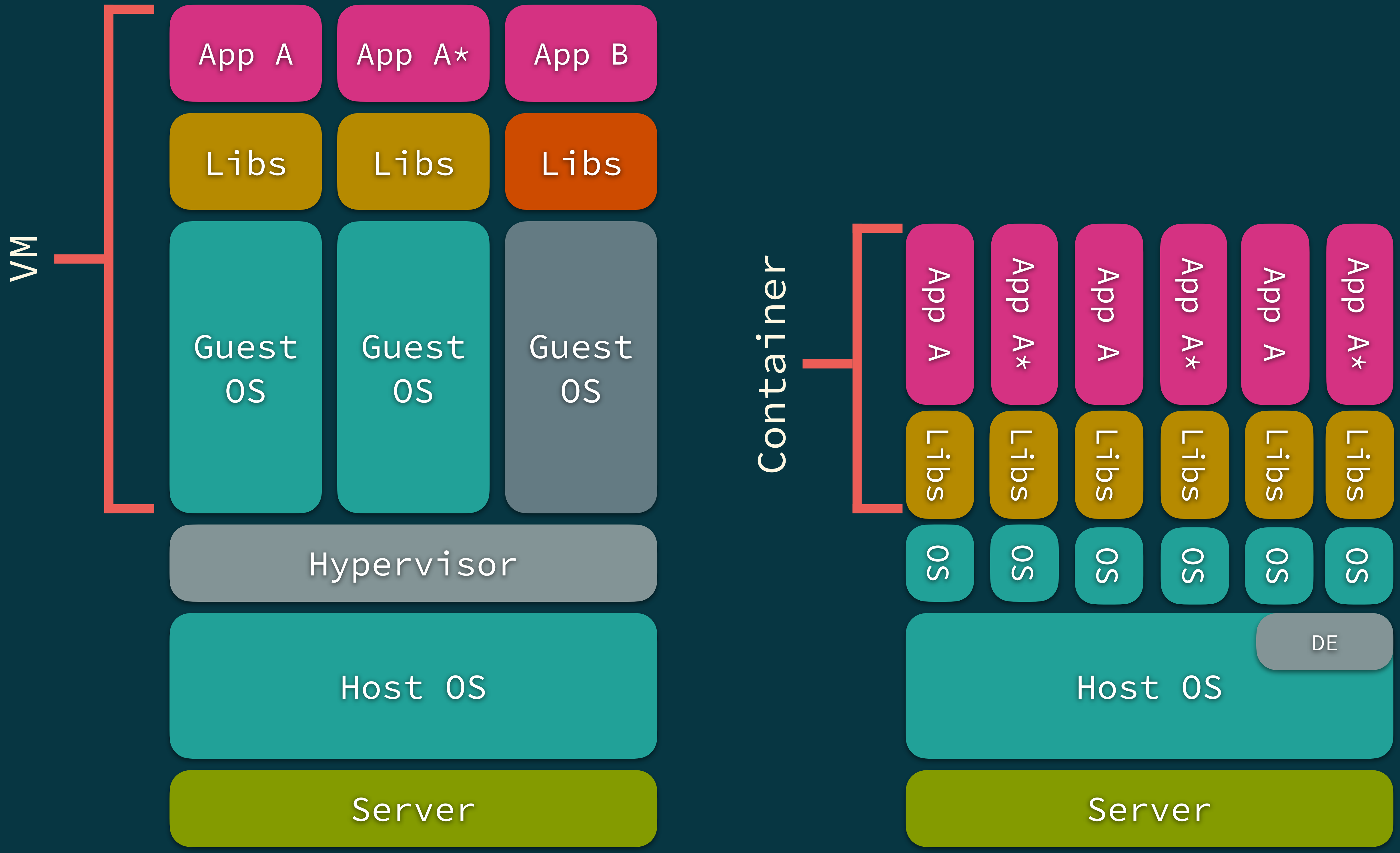
Guest OS

Guest OS

Hypervisor

Host OS

Server



VM

App A

App A*

App B

Libs

Libs

Libs

Guest OS

Guest OS

Guest OS

Hypervisor

Host OS

Server

Container

App A

App A*

App A

App A*

App A

App A*

Libs

Libs

Libs

Libs

Libs

Libs

OS

OS

OS

OS

OS

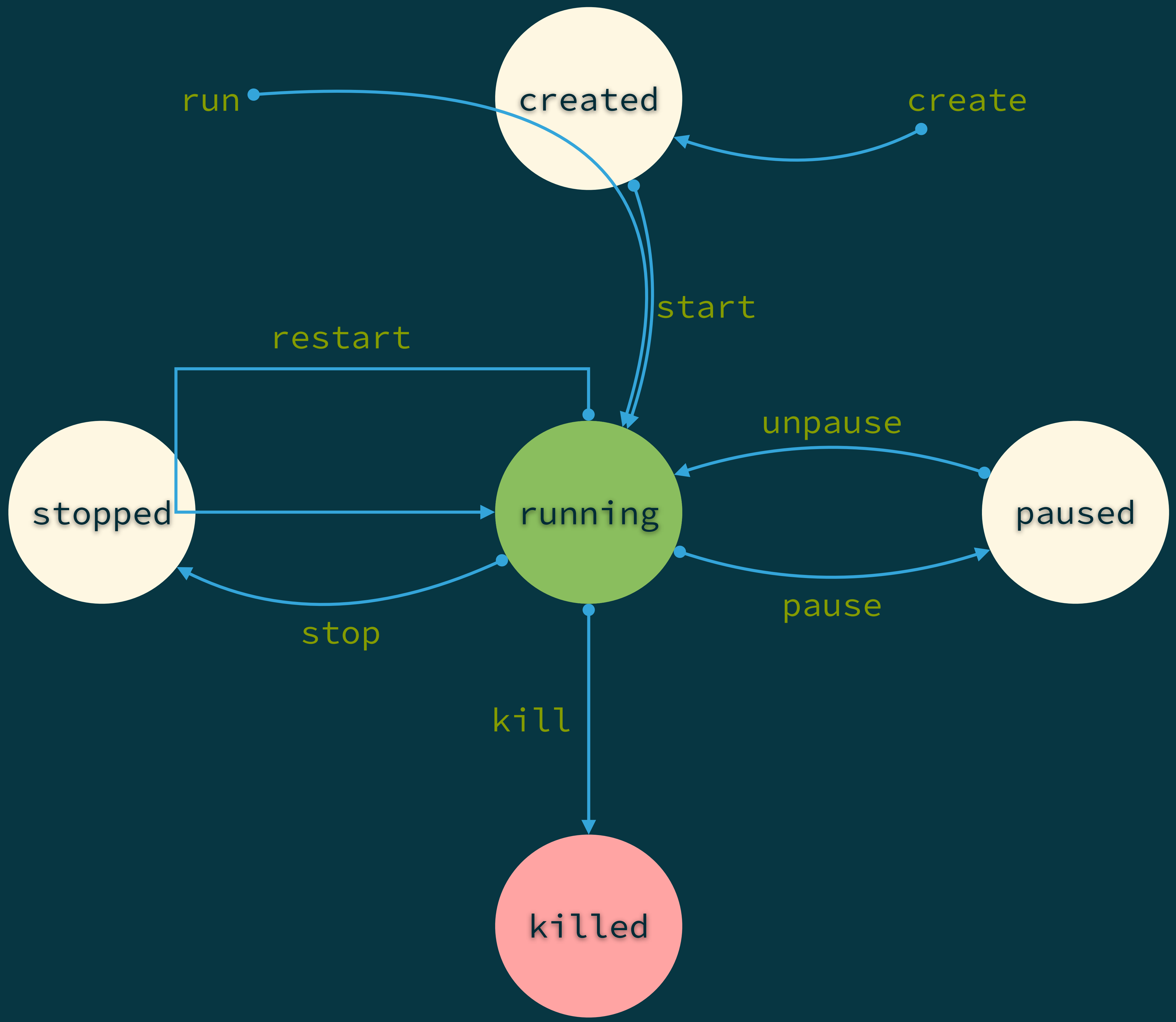
OS

DE

Host OS

Server

DOCKER LIFECYCLE


run
created
create
start
restart
unpause
stopped
running
paused
stop
pause
kill
killed

@EXERCISE

EXERCISE

- Use create + start to run jenkins:2.60.3 (Be sure to name it!)
- Trace the logs
- Be sure to `stop` it, and then remove it

HINTS

- create (use the --name or -n flag)
- start
- logs (use the --follow or -f flag)
- stop
- rm

CONTAINERS?

CGROUPS

NAMESPACES

JAILS

CONTAINERS

- A container is a lightweight virtual runtime*
- Share the host kernel
- CPU/Memory/Network/File system isolation
- Own their on hostname, users, networking stack

NAMESPACES

“What you can see”

NAMESPACES

- Isolation of
  - Users
  - Filesystem
  - Process trees
  - Network
  - IPC

CGROUPS

“What you can use”

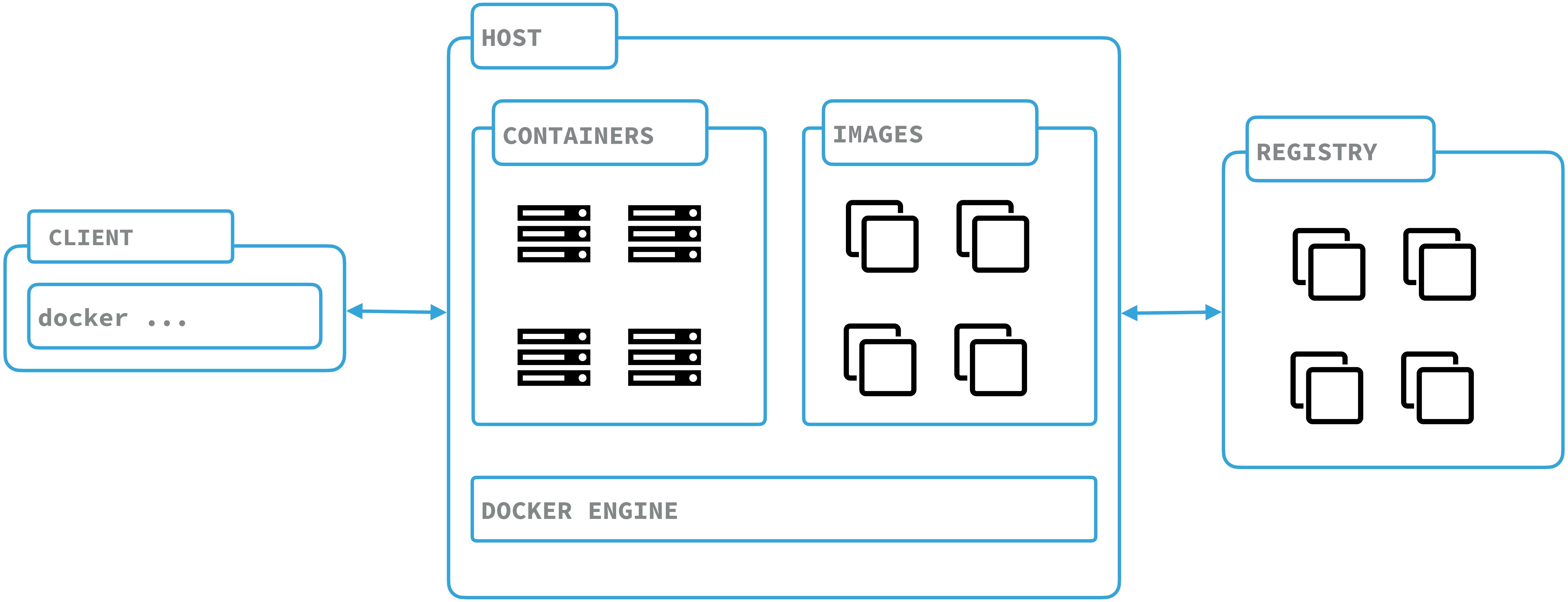
CGROUPS

- Limiting/Metering/ACL
  - CPU
  - Memory
  - I/O
  - Network
  - Device permissions

TERMINOLOGY

TERMINOLOGY

- Docker Engine
- Docker client
- Dockerfile
- Docker Machine
- Docker Compose
- Docker Stack
- Docker Swarm
- Docker Hub


CLIENT
docker ...
HOST
CONTAINERS
IMAGES
DOCKER ENGINE
REGISTRY

DOCKER CLIENT

HTTP

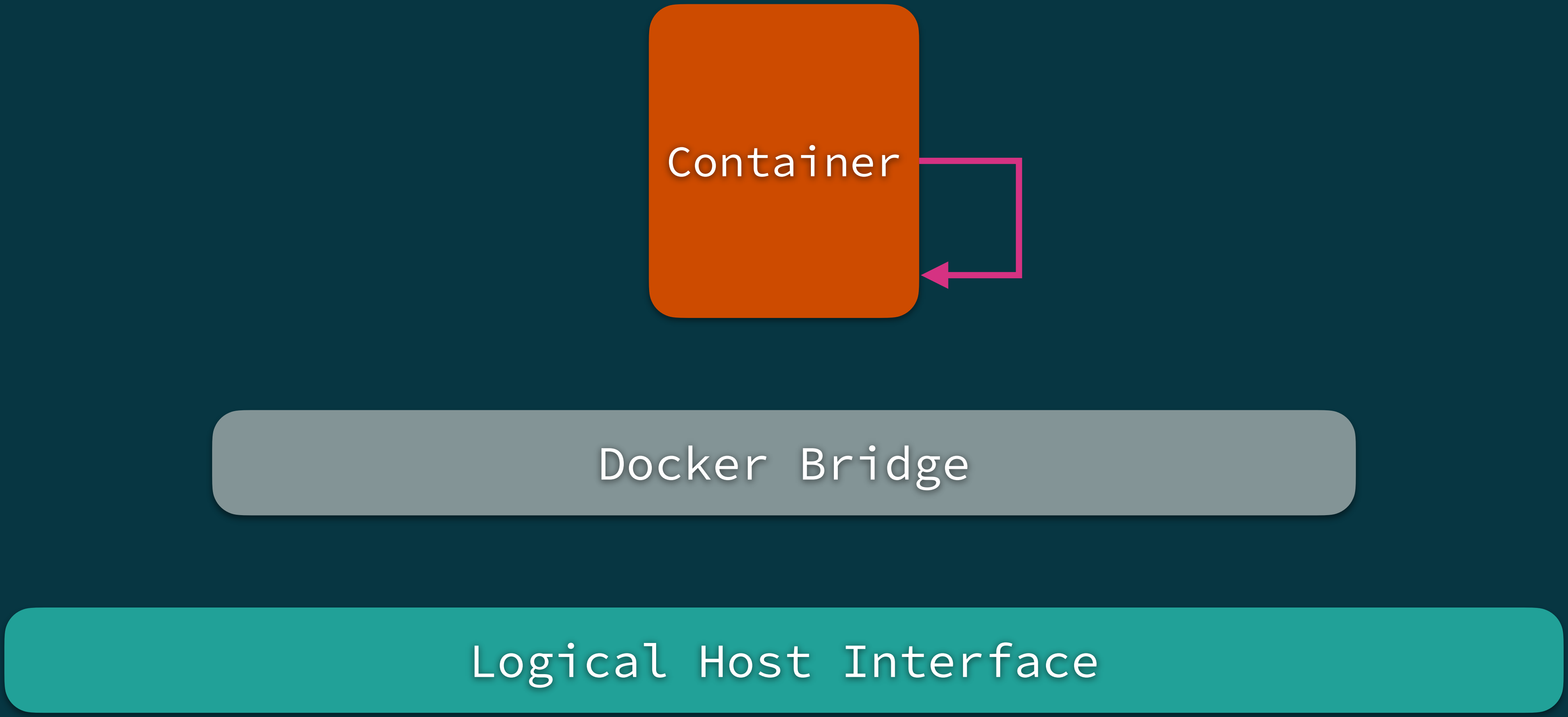
```
curl --unix-socket /var/run/docker.sock \
     http:/docker/images/json

curl --unix-socket /var/run/docker.sock \
     http:/docker/containers/json

curl --unix-socket /var/run/docker.sock \
     http:/docker/containers/<container-id>/logs?stdout=1
```

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

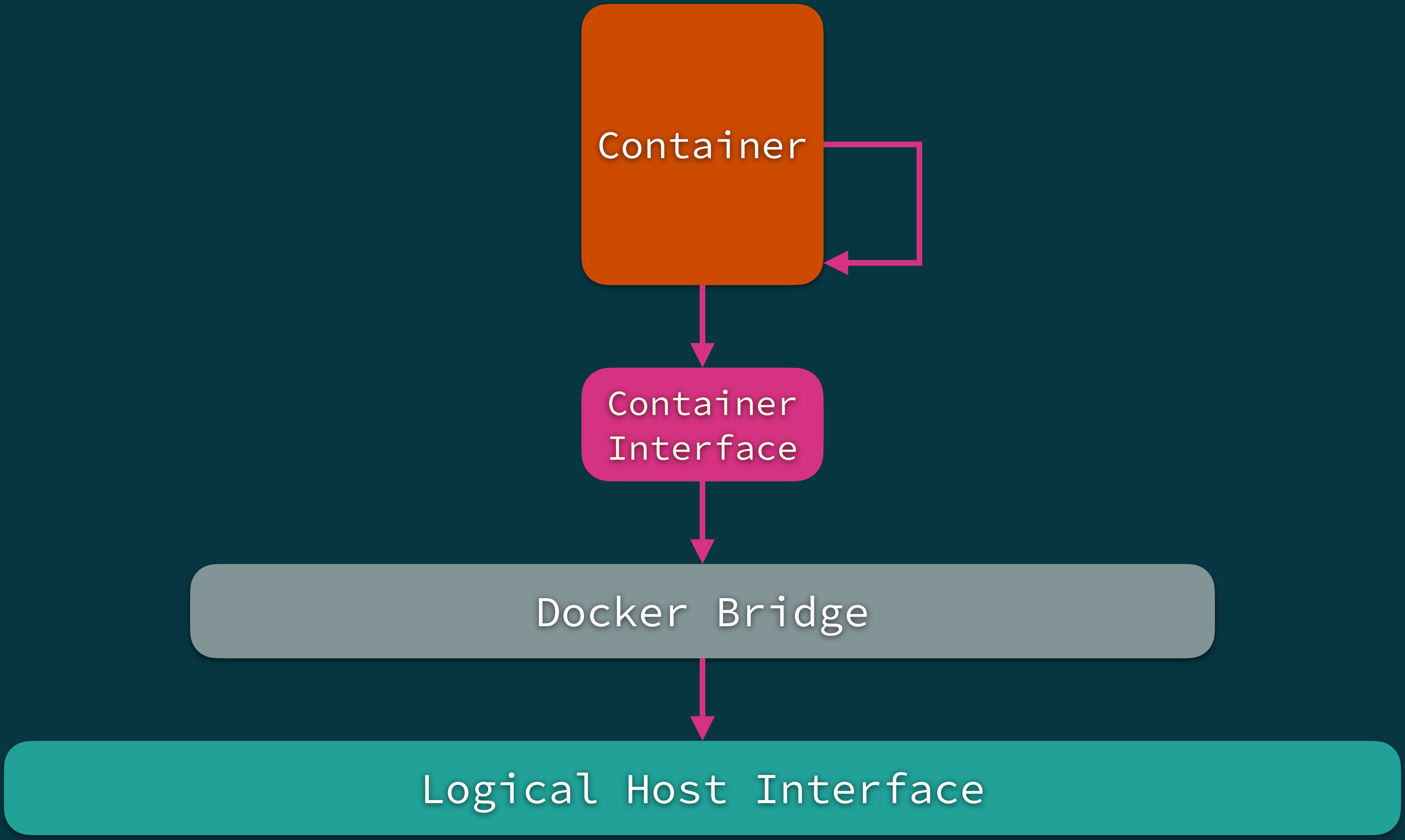
```
docker run -d \
  -p 9000:9000 \
  -v /var/run/docker.sock:/var/run/docker.sock portainer/portainer

# visit http://localhost:9000
```

NETWORK

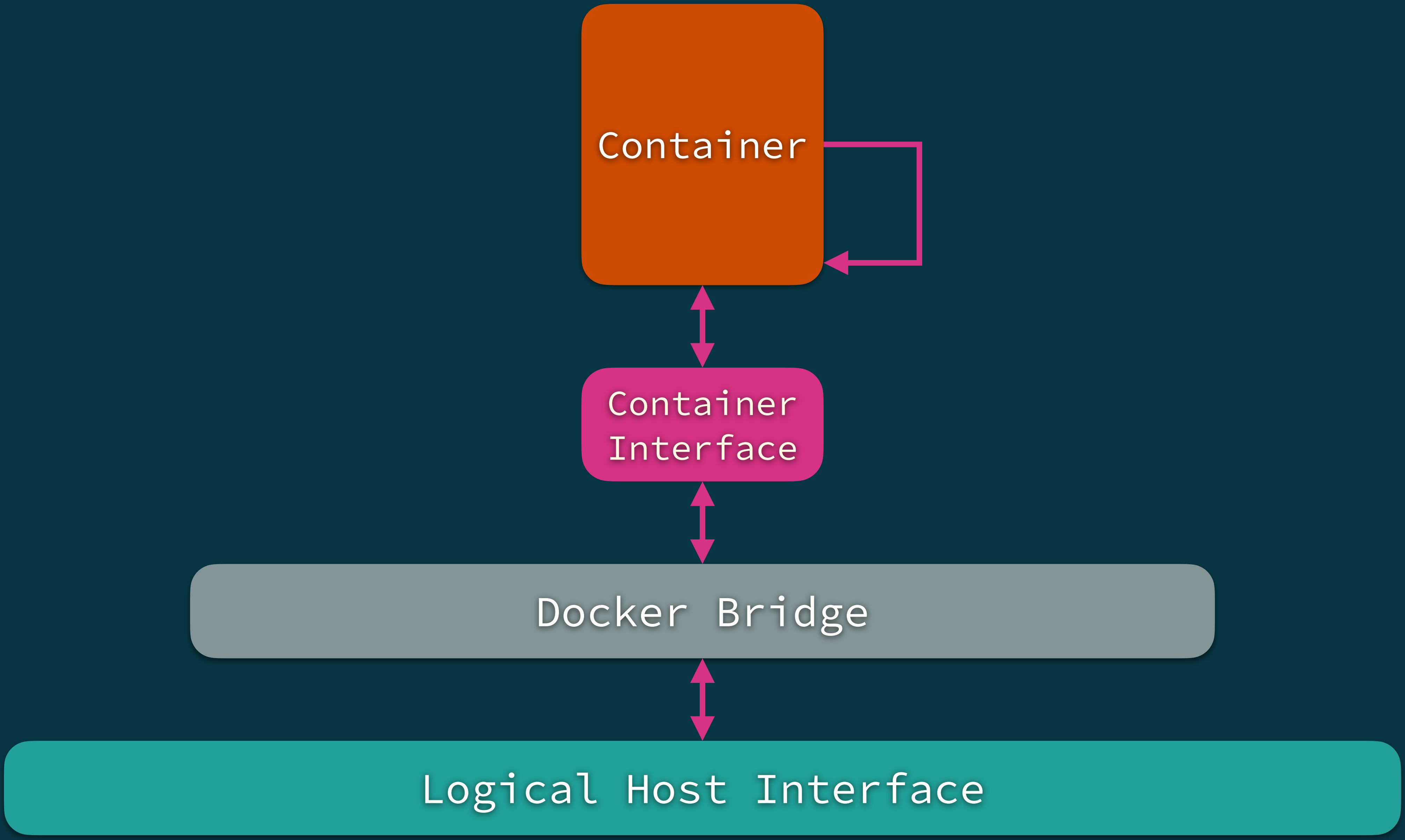
```
docker run -it --net none --rm alpine /bin/sh
```



```
docker run -it --rm alpine /bin/sh
```



```
docker run -it --rm -p 8080:8080 alpine /bin/sh
```



@EXERCISE

EXERCISE

- Use run to start a named "jenkins/jenkins:2.225" container on port 8080
    - Trace the logs
    - Visit http://locahost:8080
- Stop and remove that container, start another one on another port, and see if you can get to it

HINTS

- run (use the --port or -p flag) # export the port

```
# start a webserver
docker run -d --name frontend nginx:1.17.9-alpine

# inspect the env of the running container
docker exec frontend env

# inspect the env of a basic alpine container
docker run alpine:3.9 env

# inspect the env of a alpine container linked to the webserver
docker run --link frontend:webserver alpine:3.9 env

# see how the linked container sees the webserver
docker run --link frontend:webserver alpine:3.9 cat /etc/hosts
```

@EXERCISE

EXERCISE

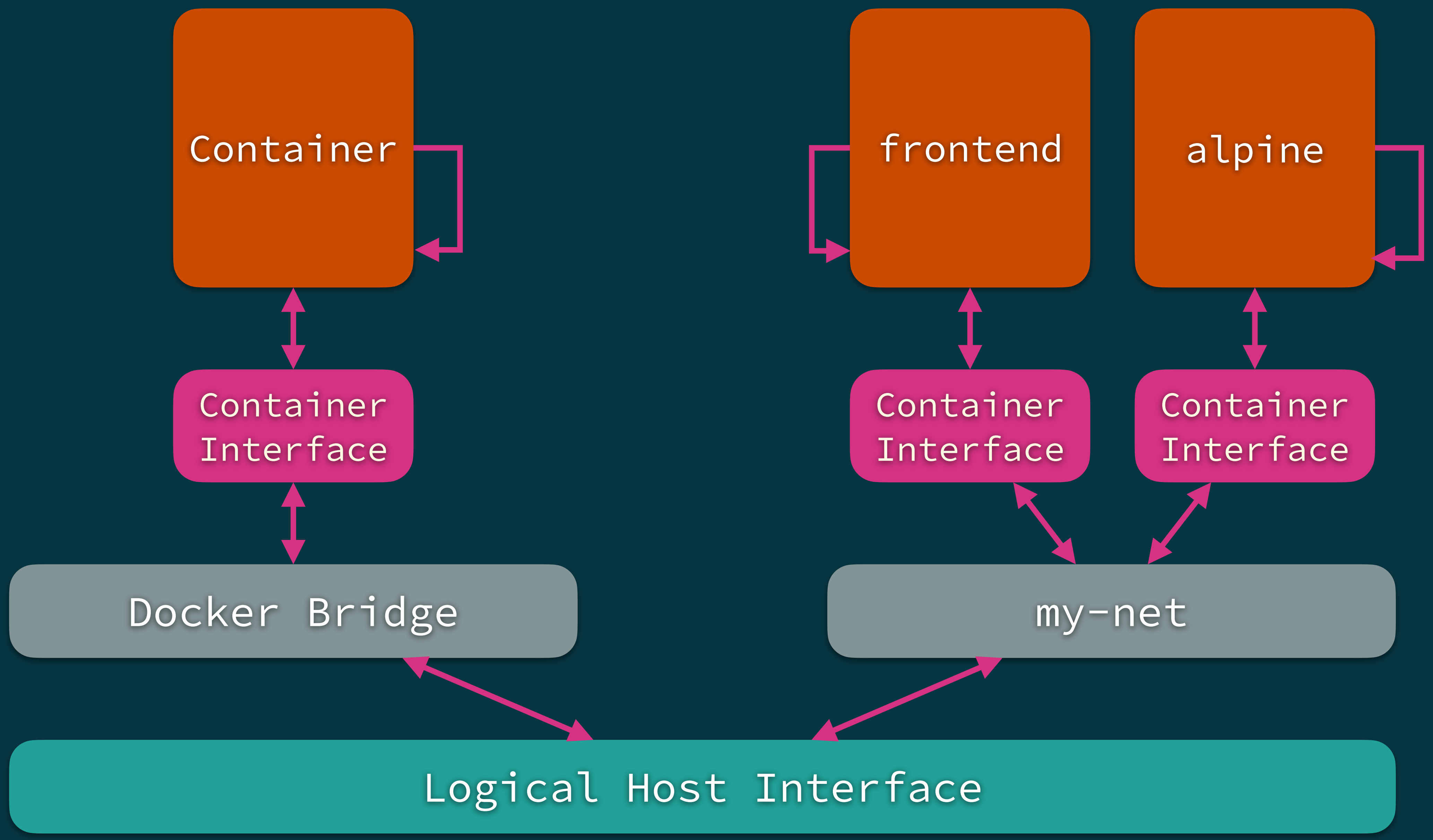
- Start a "nginx:1.17.9-alpine" container with a specific name
- Start an interactive "alpine:3.9" container linked to the nginx container
    - Install curl
    - curl the linked container

HINTS

- apk add curl # install curl in alpine

docker network

```
# create a network
docker network create my-net

# start a webserver using that network
docker run -d --name frontend --net my-net nginx:1.17.9-alpine

# see how the linked container sees the world
docker run --net my-net alpine:3.9 cat /etc/hosts
docker run --net=my-net alpine:3.9 cat /etc/resolv.conf

# see if the two can talk to each other
docker run --net=my-net alpine:3.9 ping frontend

# see if some other container on another network can see it
docker run alpine:3.9 ping frontend
docker network create my-net
```



@EXERCISE

EXERCISE

- Create a docker network
- Start a "nginx:1.17.9-alpine" container with a specific name using that network
- Start an interactive "alpine:3.9" container using that network
    - See if you can ping the nginx container using its name
- Start another interactive "alpine:3.9" container using the default network
    - See if you can ping the nginx container using its name

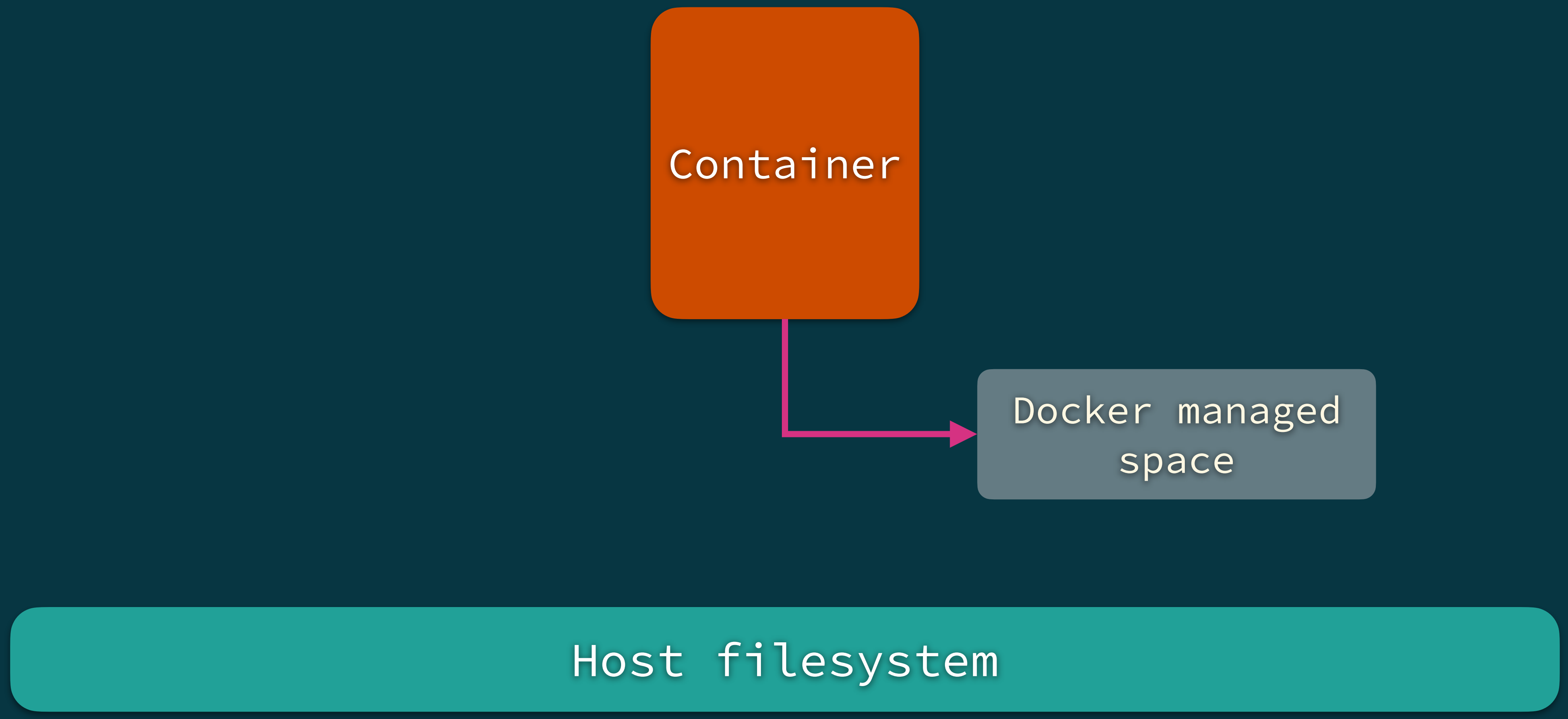
HINTS

- docker network create <some-name> # create a network
- --net # flag for assigning a container a network

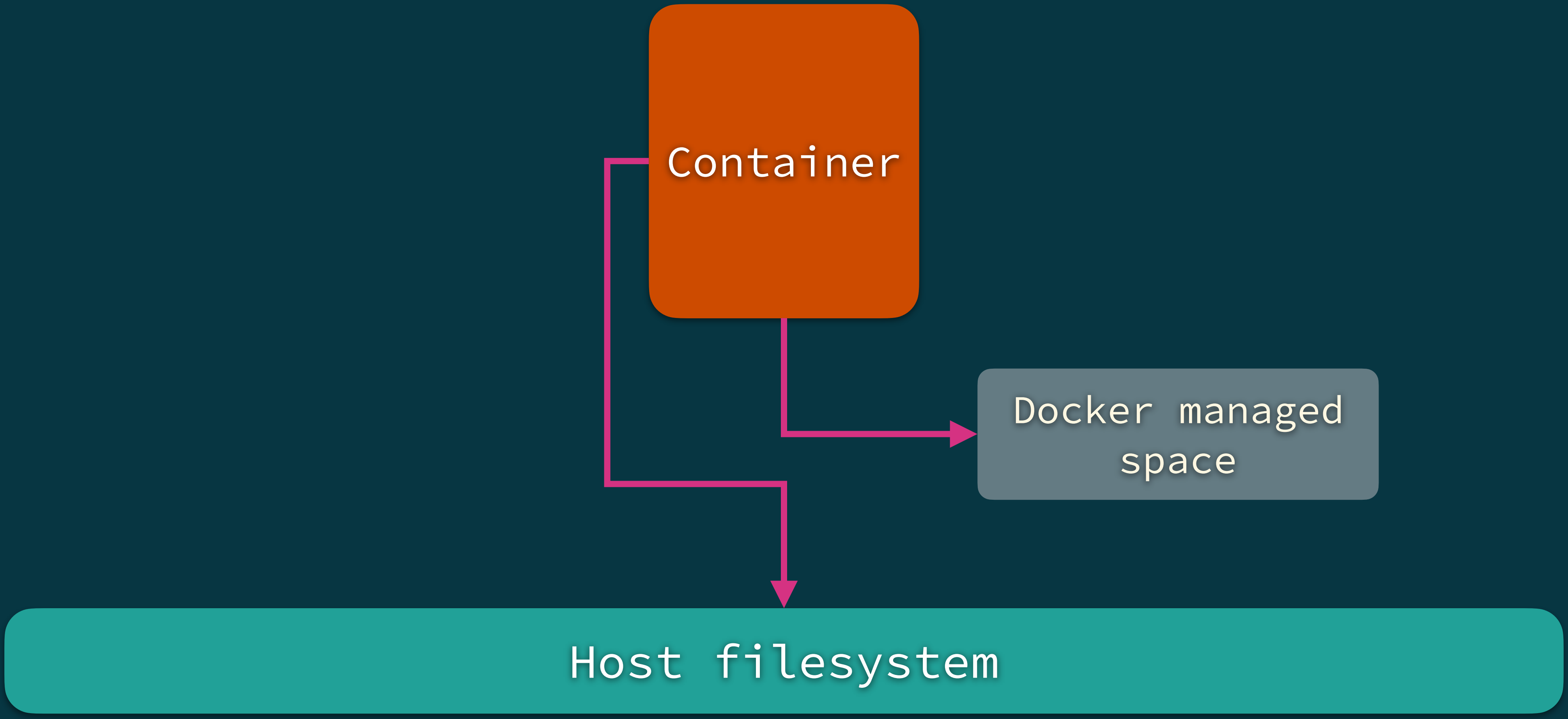
VOLUMES

```
--volume
```

```
docker run -it --rm ubuntu /bin/bash
```



```
docker run -it -v /host/path:/tmp ubuntu /bin/bash
```



@EXERCISE

EXERCISE

- Start a nginx:1.14-alpine container in daemon mode mounting the "nginx-files" to "/usr/share/nginx/html" exposing port 8080
    - Visit http://localhost:8080
    - Modify index.html file in nginx-files and refresh your browser

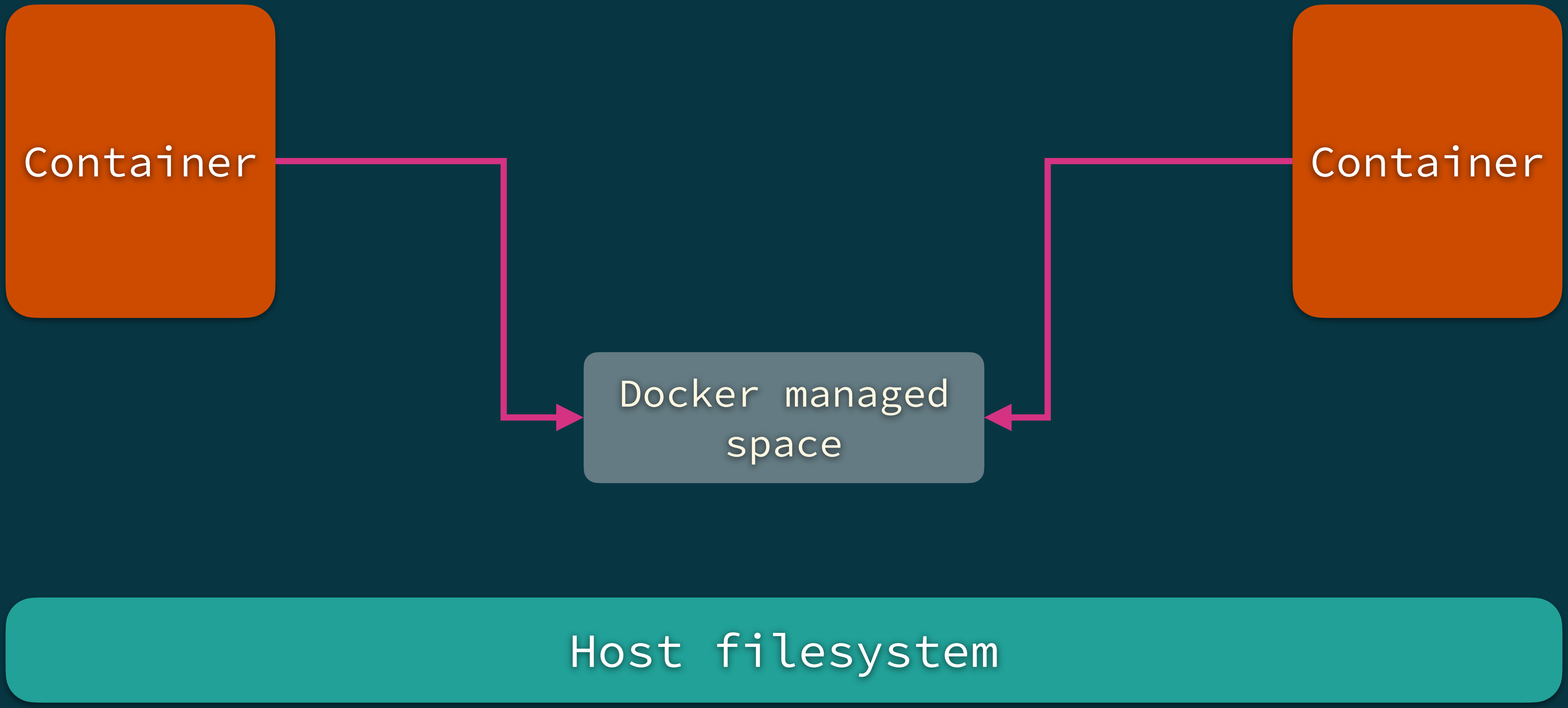
HINTS

- run with the --port or -p flag AND the --volume or -v flag

docker volume

```
# create a volume
docker volume create my-volume

# create a container using that volume
docker run -it --rm -v my-volume:/tmp ubuntu:19.10 bash

# inside the container
root@cf00e17b54ae:/tmp# echo 'hello world' >> /tmp/my-file.txt
root@cf00e17b54ae:/tmp# cat /tmp/my-file.txt

# start another container with the same volume
docker run -it --rm -v my-volume:/tmp ubuntu:18.10 bash

# in that container
root@c945ba2aa878:/tmp# ls /tmp
my-file.txt
root@c945ba2aa878:/tmp# cat /tmp/my-file.txt
hello world
```

```
docker run -it -v volume-name:/tmp ubuntu /bin/bash
```



@EXERCISE

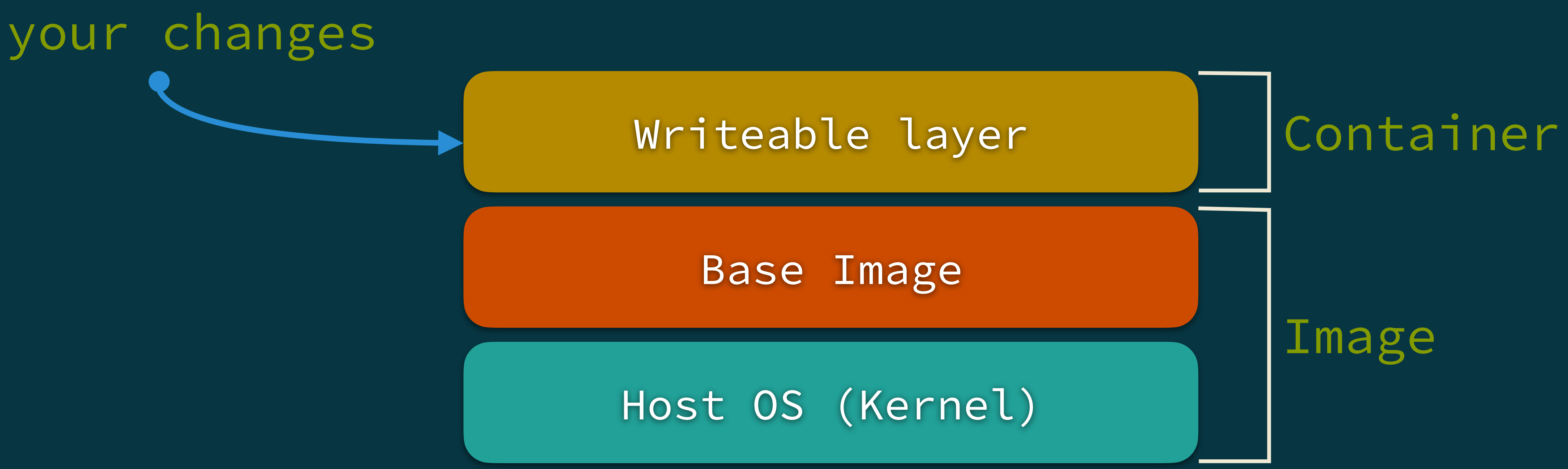
EXERCISE

- Create a docker volume
- Start a interactive "ubuntu:19.10" container with that volume mounted
  - In that container add some files to the mounted directory
  - Exit that container
- Start another interactive "ubuntu:19.10" container with that volume mounted
  - Make sure that the files still exist

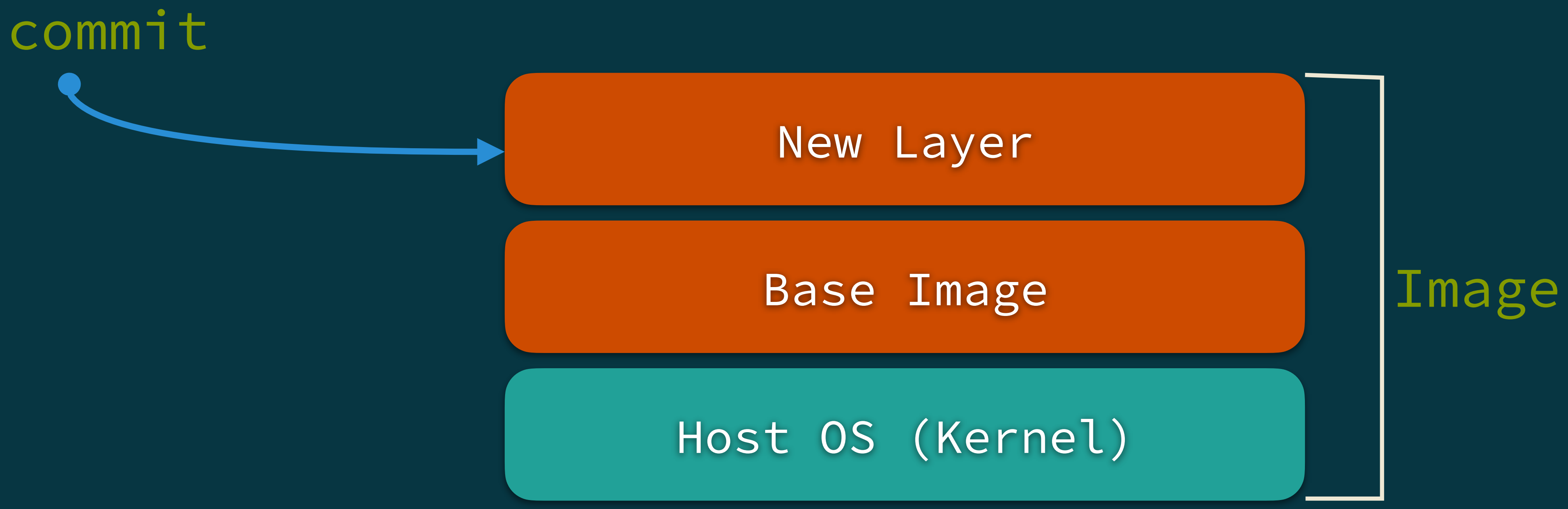
HINTS

- docker volume create <some-name> # create a volume
- --volume or -v # flag for assigning a container a volume

WHAT IS A CONTAINER?



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



commit

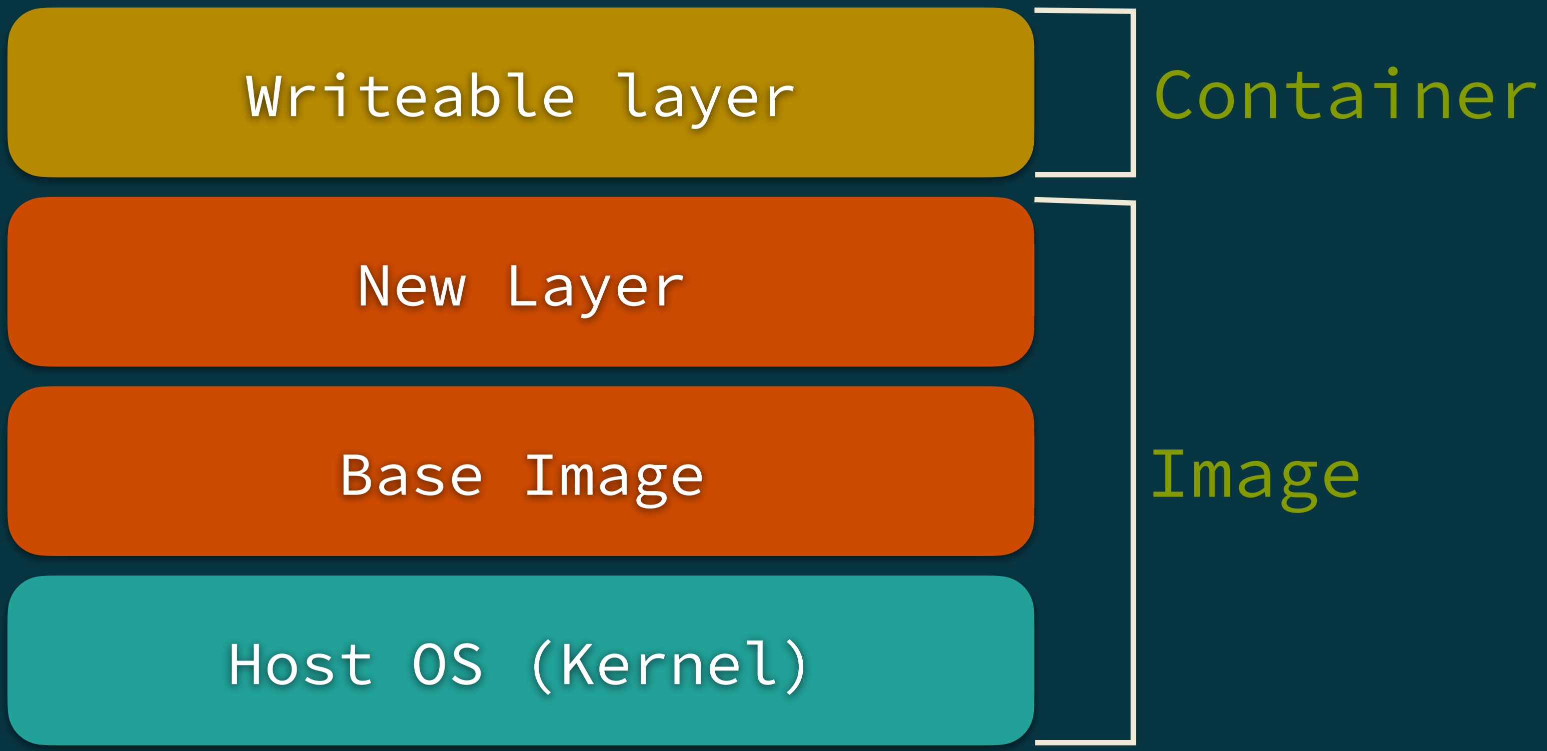
New Layer

Base Image

Host OS (Kernel)

Image

```
run <new-image>
```


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

@EXERCISE

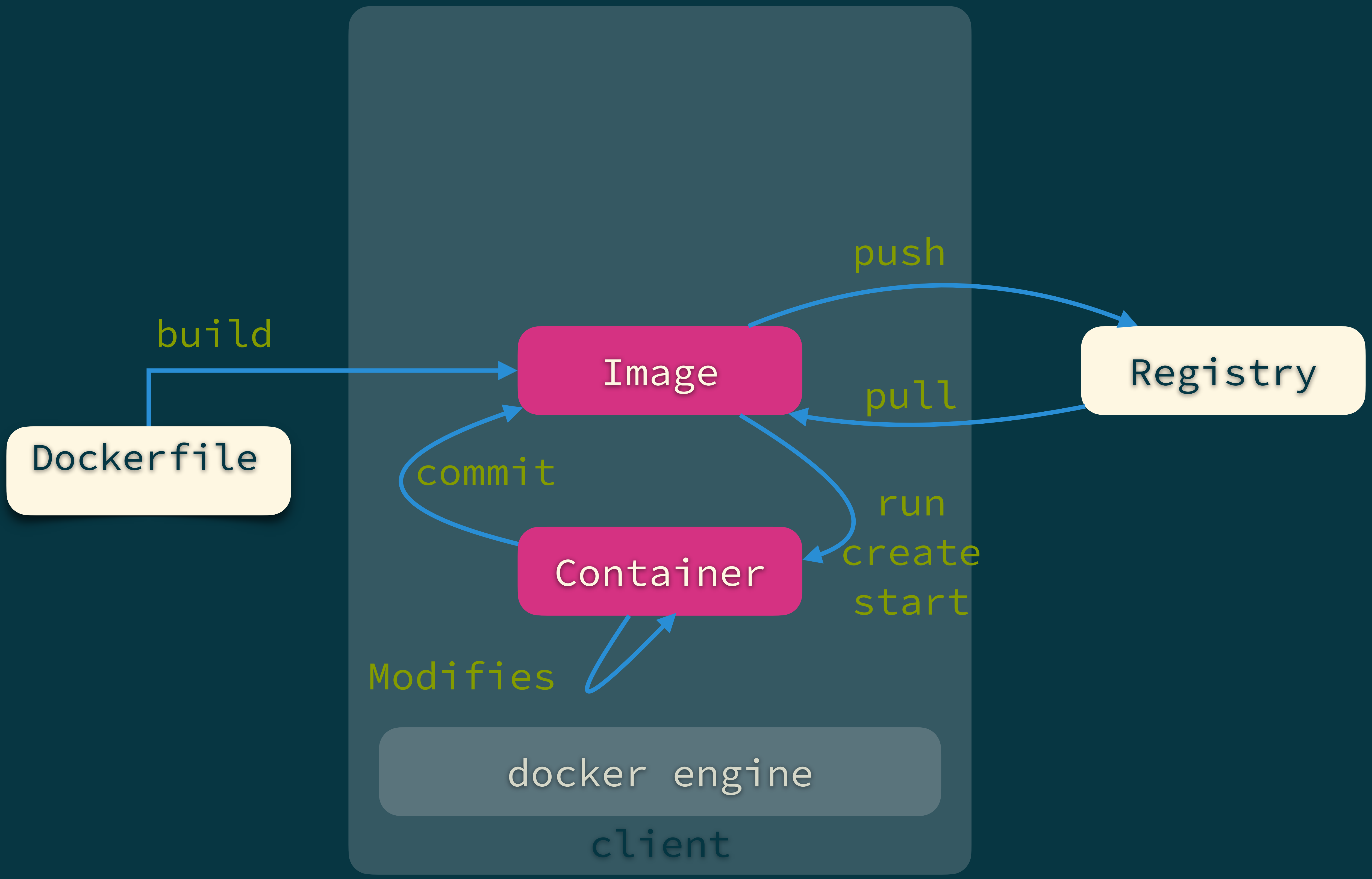
EXERCISE

- Use run to start a ubuntu:19.10 container (Be sure to name it!)
- touch a couple of new files
- Exit, then commit to create a new image named “ubuntu-addons”
- Create a new interactive container using the image “ubuntu-addons”
- See if your files are there

HINTS

- run (use the --name or -n flag) # create a new container
- echo “My OWN image” >> myFile.txt # create a new file with contents
- commit <contain-name> <new-image-name> # create a new image
- cat someFile # displays the contents of a file

WORKFLOW


build

Dockerfile

Image

push

pull

Registry

commit

run
create
start

Container

Modifies

docker engine

client

DOCKERFILE

DOCKERFILES

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

FROM / COPY

@EXERCISE

EXERCISE

- Create a Dockerfile in the "nginx-files" folder
    - Base it on "nginx:1.17.9-alpine"
    - COPY the index.html file into /usr/share/nginx/html/
- Build a new image from this Dockerfile and run it exposing port 8080
- Visit http://localhost:8080
- Inspect your new image

HINTS

- FROM <base-image>
- COPY src dest
    - If you are COPY-ing a file, you need to name the file in the dest
- docker help build;

CMD

@EXERCISE

EXERCISE

- Prior to doing this exercise make sure you have run "./gradlew compileJava shadowJar" in the "code/hello-vertx" folder
- Create a Dockerfile in the "hello-vertx" folder
  - Base it on "openjdk:8u131-jre"
  - COPY ./build/libs/docker-workshop-0.0.1-SNAPSHOT-fat.jar
  - Use CMD to run "java -jar"
- Build a new image from this Dockerfile
- Inspect your new image

RUNNING IT

EXERCISE

- Create a new network
- Start a "mongo:3.6.17" container named "mongo" and attach it to the network
- Start a container with your newly built image using the same network and exposing port 8080
- Visit http://localhost:8080

WORKDIR / .DOCKERIGNORE

@EXERCISE

EXERCISE

- Create a Dockerfile.build in the "hello-vertx" folder
  - Base it on "openjdk:8u131-jdk"
  - Create a workdir
  - Copy all of the source files into the work directory
  - Use CMD to run "./gradlew shadowJar --no-daemon"
- Build a new image from this Dockerfile
- Start a new container from this image and list all the files in the working directory
  - What do you see being copied over? Do you need all that?

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
- Create a base image with `FROM scratch`

DO’S ✓

- Pin down the exact tag (or even better the digest)
  - Do not use “latest” tag
- Inspect ancestor images for `USER`s, `PORT`s, `ENV`s, `VOLUME`s, `LABEL`s and anything that can be inherited

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

ADD/COPY

DON'TS

- Avoid ADD
- Do not leave “residual” artifacts

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

ENTRYPOINT/CMD

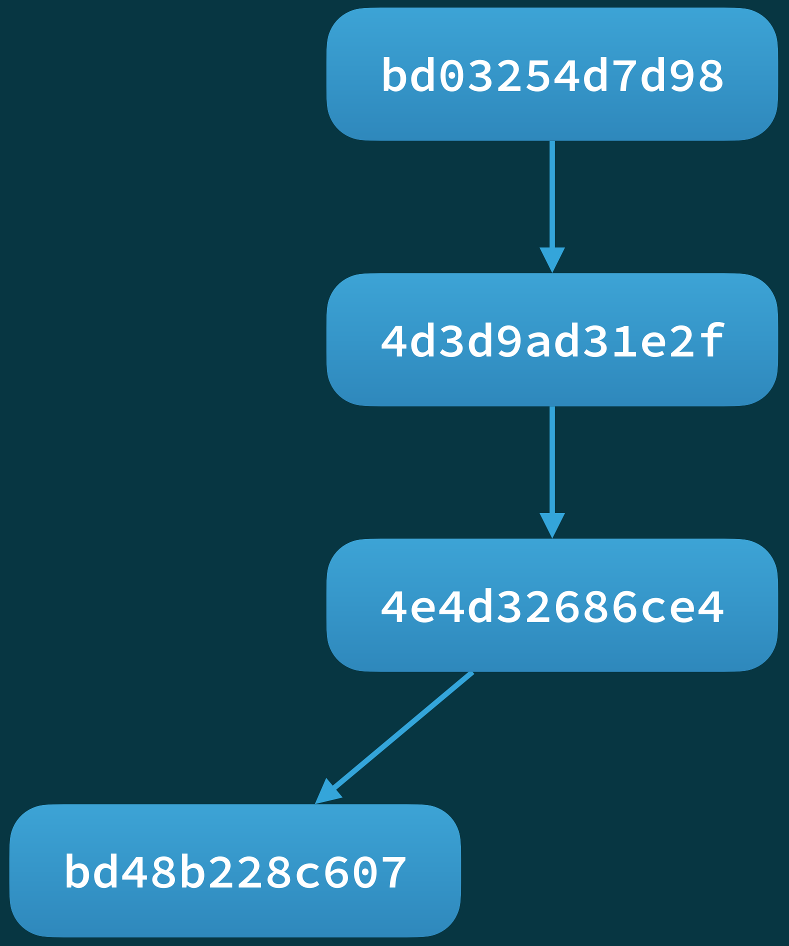
DONT'S

- Avoid the “shell” form

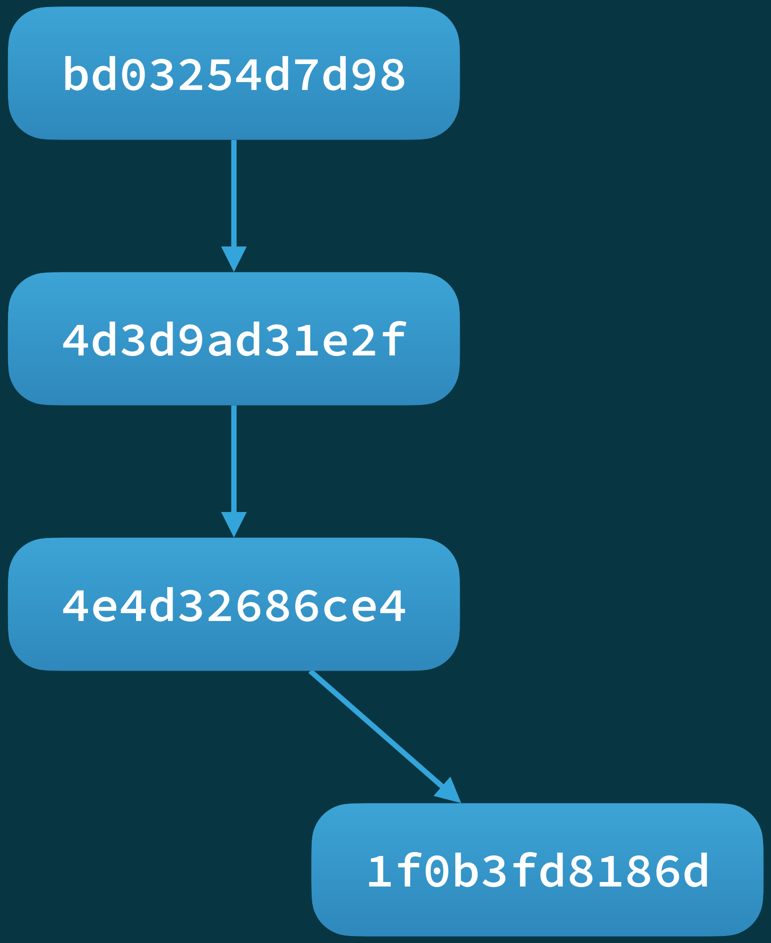
DO'S

- Use the “exec” form
  - Shell expansion will **not** happen!
- Use `ENTRYPOINT` and `CMD` together
- Use a “entrypoint-script”
  - Always “exec” (or “gosu”)

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"]
```



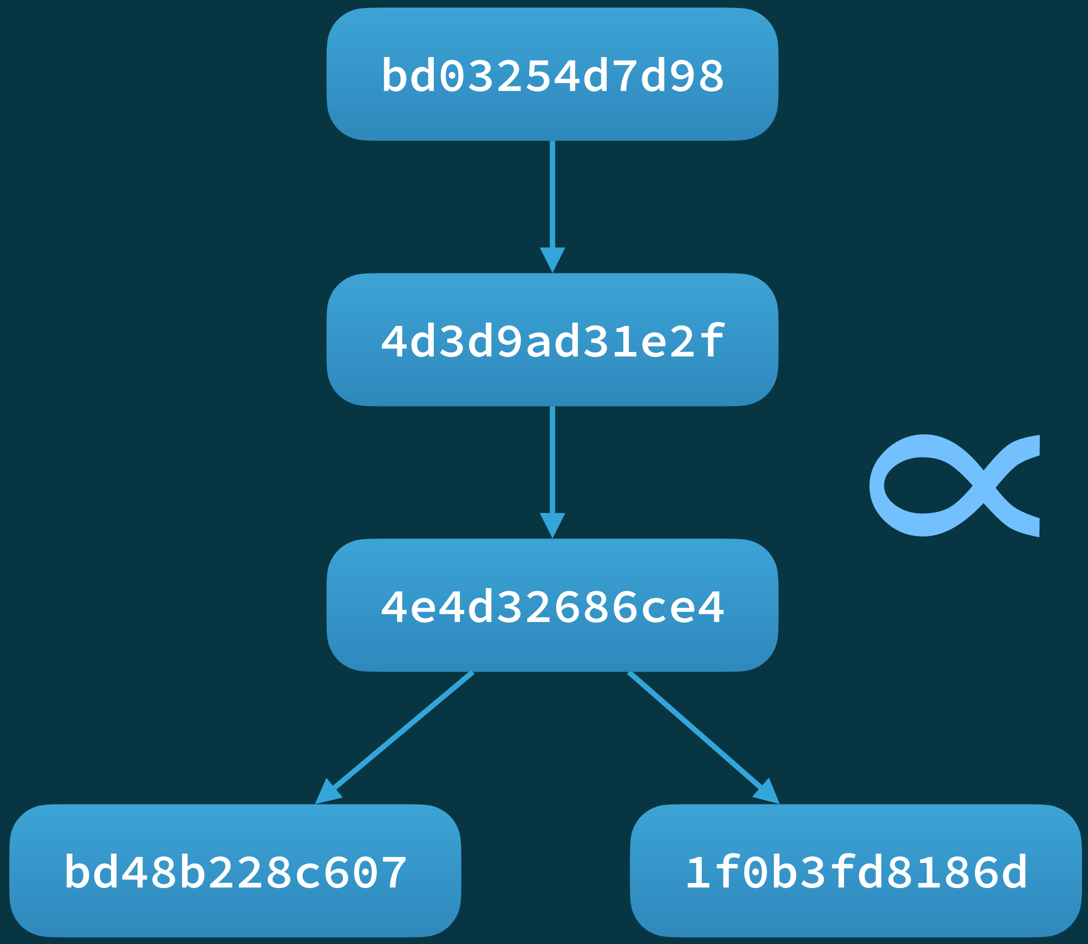
bd03254d7d98

4d3d9ad31e2f

4e4d32686ce4

1f0b3fd8186d

```
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"]
```



bd03254d7d98

4d3d9ad31e2f

4e4d32686ce4

bd48b228c607

1f0b3fd8186d

∝

```
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"]
```

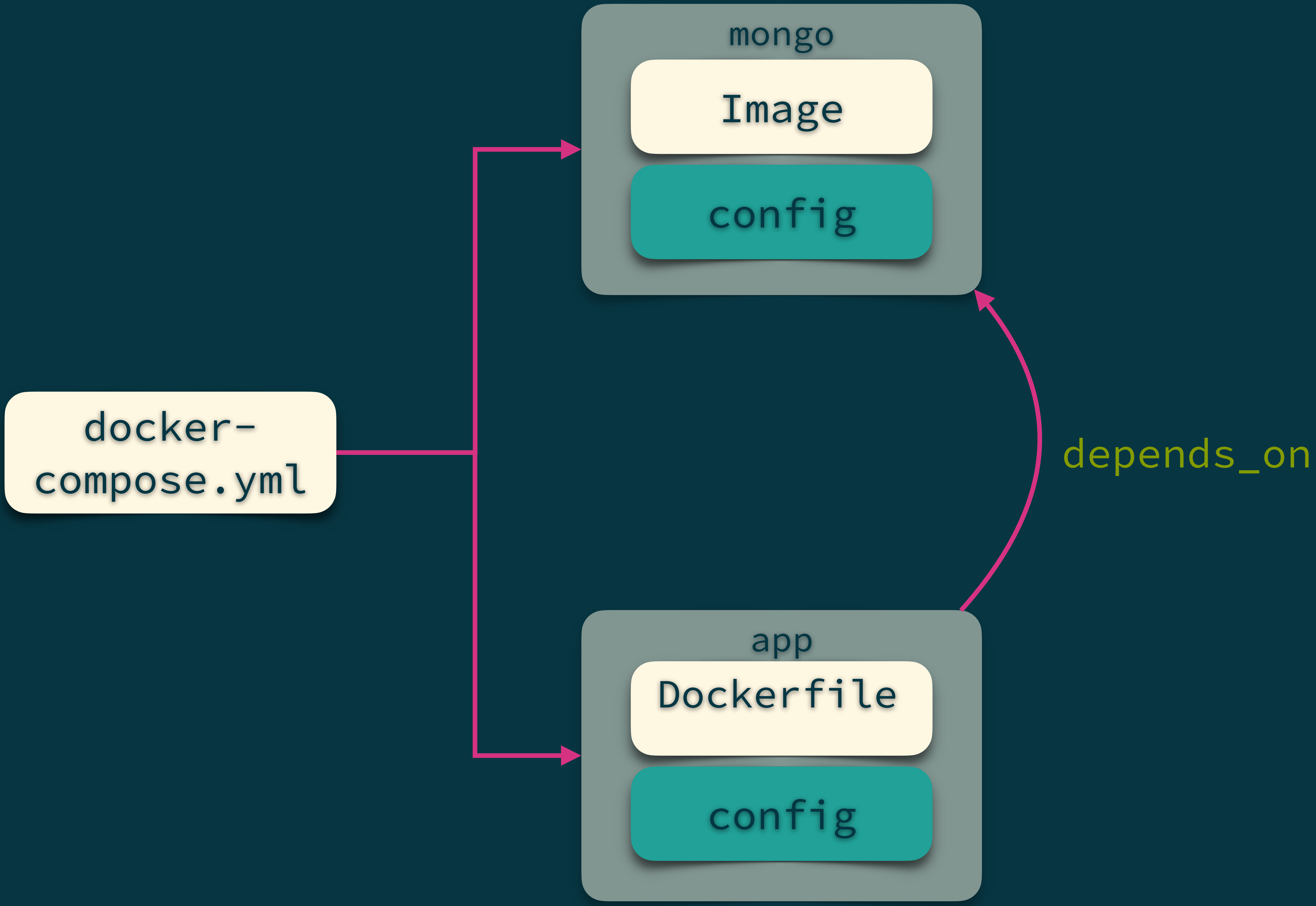
DOCKER COMPOSE

DOCKER COMPOSE

- A system is usually made up of multiple containers
- Containers depend on each other
  - Orchestration
- Single host

DOCKER COMPOSE

- Define multi-container applications in a single file
- Supports scaling, healing
- Single host



docker-compose.yml

mongo

Image

config

app

Dockerfile

config

depends_on

THANKS!!