Introduction to Docker

IDA Embedded - May 2017

Jan Krag

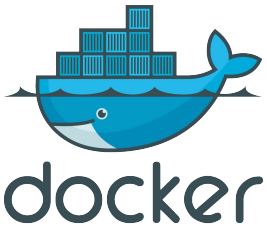
Continuous Improvement Agent

Trainer and mentor in Git, Docker, Continuous Delivery, Cat genetics and related topics

jak@praqma.net

@jankrag

A brief tour of Docker



By the end of this session you will understand:

- What is a container and why you may want one
- How to run pre-built containers
- How to create your own containers
- How to share your containers
- How to run multi-container applications
- How Docker supports Continuous Delivery


MAERSK LINE

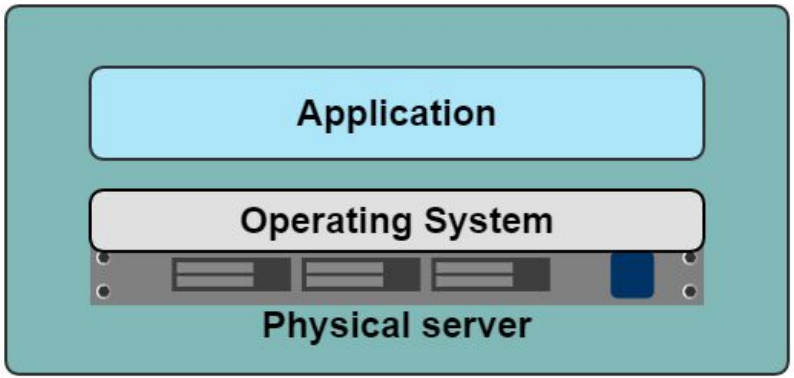
What the why now?

If docker is the answer, what is the question?

A History Lesson

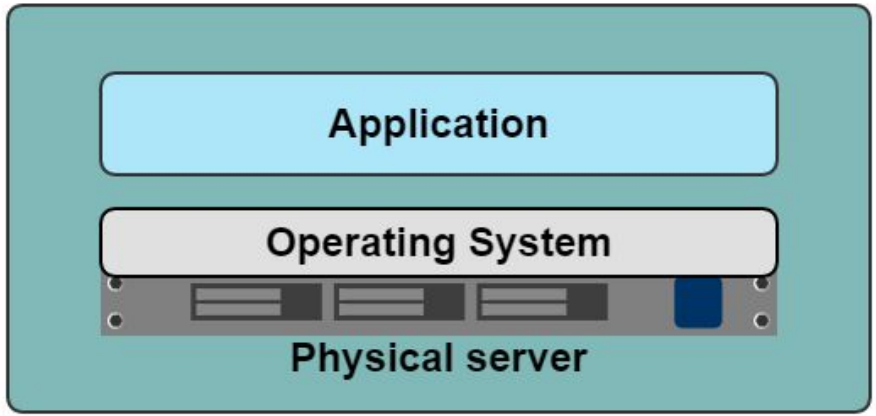
In the Dark Ages

One application on one physical server



Historical limitations of application deployment

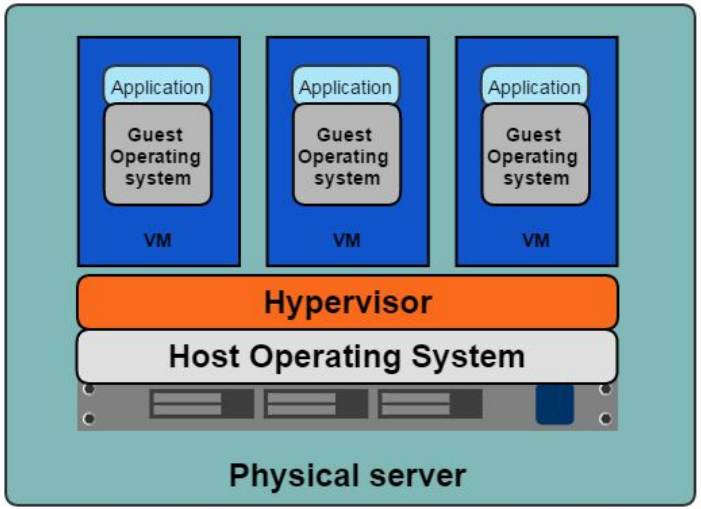
- Slow deployment times
- Huge costs
- Wasted resources
- Difficult to scale
- Difficult to migrate
- Vendor lock in



A History Lesson

Hypervisor-based Virtualization

- One physical server can contain multiple applications
- Each application runs in a virtual machine (VM)



Benefits of VM's

- Better resource pooling
  - One physical machine divided into multiple virtual machines
- Easier to scale
- VM's in the cloud
  - Rapid elasticity
  - Pay as you go model

Limitations of VM's

- Each VM stills requires
  - CPU allocation
  - Storage
  - RAM
  - An entire guest operating system

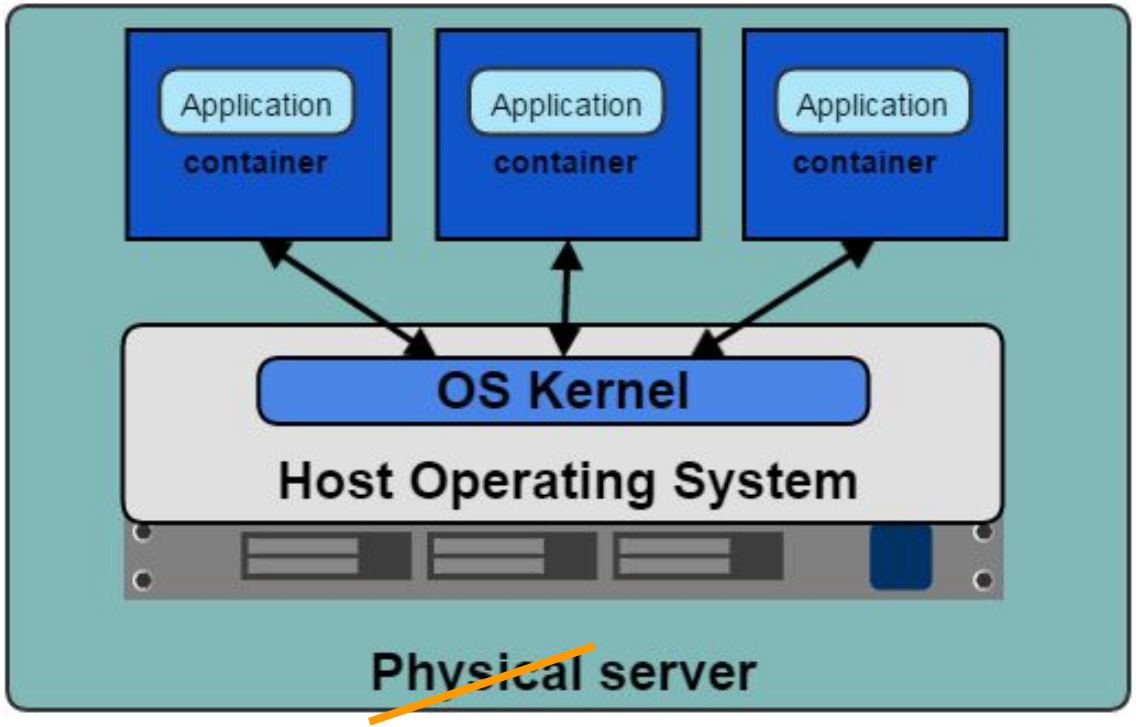
- The more VM's you run, the more resources you need
- Guest OS means wasted resources
- Application portability not guaranteed

Introducing Containers

Containerization uses the kernel on the host operating system to run multiple root file systems

- Each root file system is called a **container**
- Each container also has its own
  - Processes
  - Memory
  - Devices
  - Network stack

Containers


Application
container
Application
container
Application
container
OS Kernel
Host Operating System
Physical server

Containers vs VM's

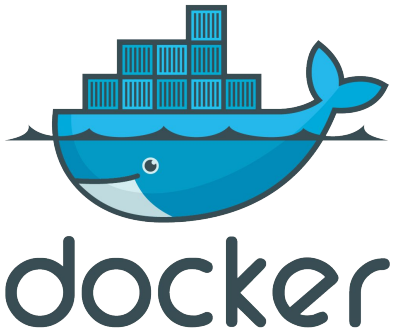
- Containers are more lightweight and faster
- No need to install guest OS
- Less CPU, RAM, storage space required
- More containers per machine than VMs
- Greater portability
- Containers are easy to manage as they share a common OS
  - Share multiple workloads on a single OS
- Containers are a better way to develop and deploy microservices compared with VMs.

Docker is a platform

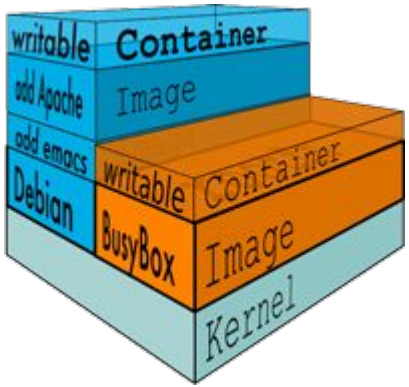
Docker is a platform for developing, shipping and running applications using container technology

The Docker Platform consists of multiple products/tools:

- Docker Engine
- Docker Hub
- Docker Trusted Registry
- Docker Machine
- Docker Swarm
- Docker Compose
- Kitematic



Dependency management



Docker provides a means to package and application with all its **dependencies** into standardized unit for software development

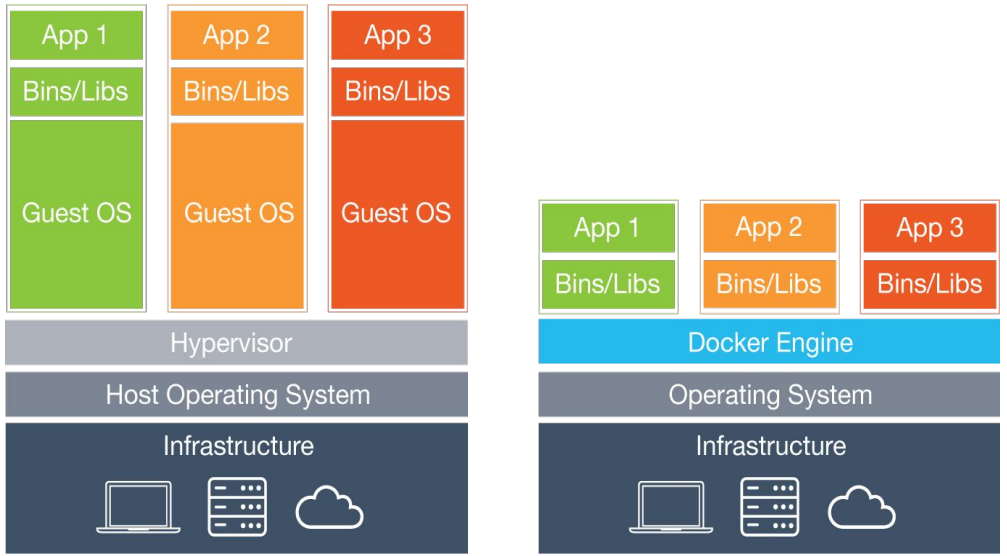
It provides **isolation**, so applications on the same host and stack can avoid dependency conflict

It is **portable**, so you can be sure to have exactly the same dependencies at runtime during development, testing and in production

From https://www.docker.com/what-docker

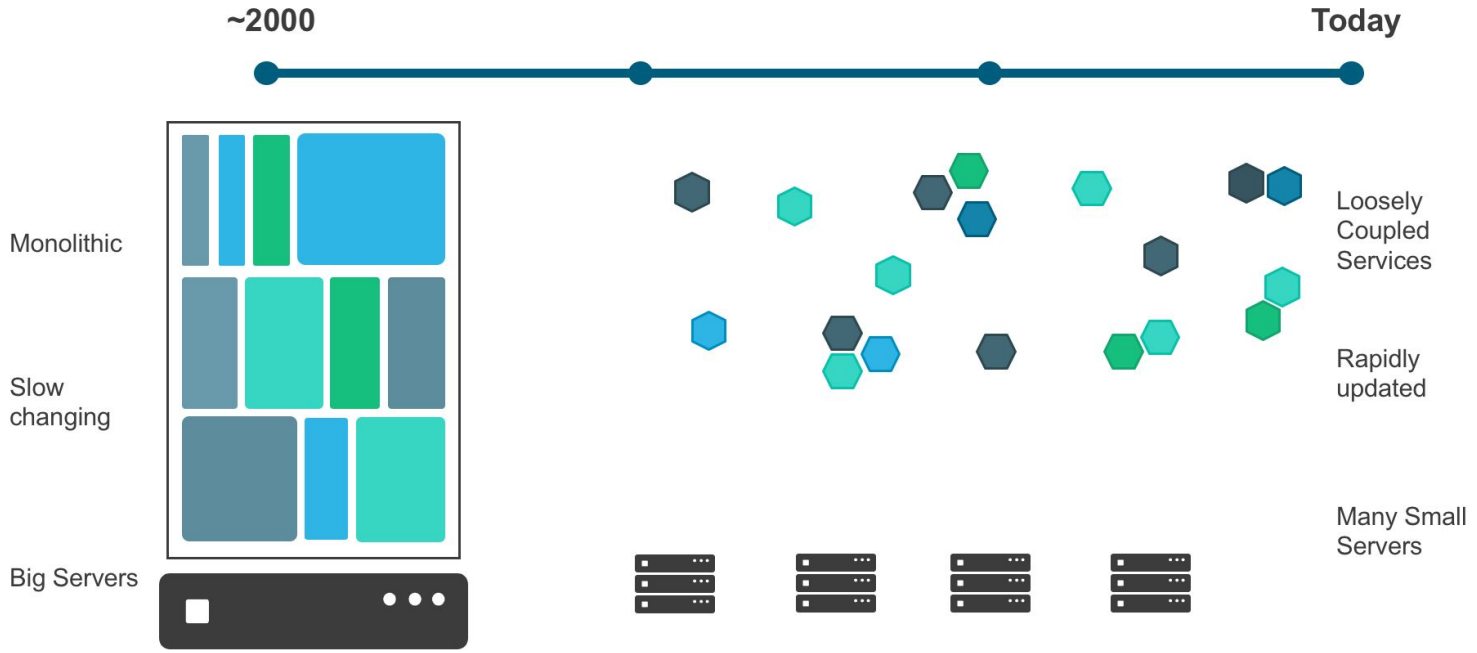
Resource Utilization

Better utilization, more portable, shared operating system



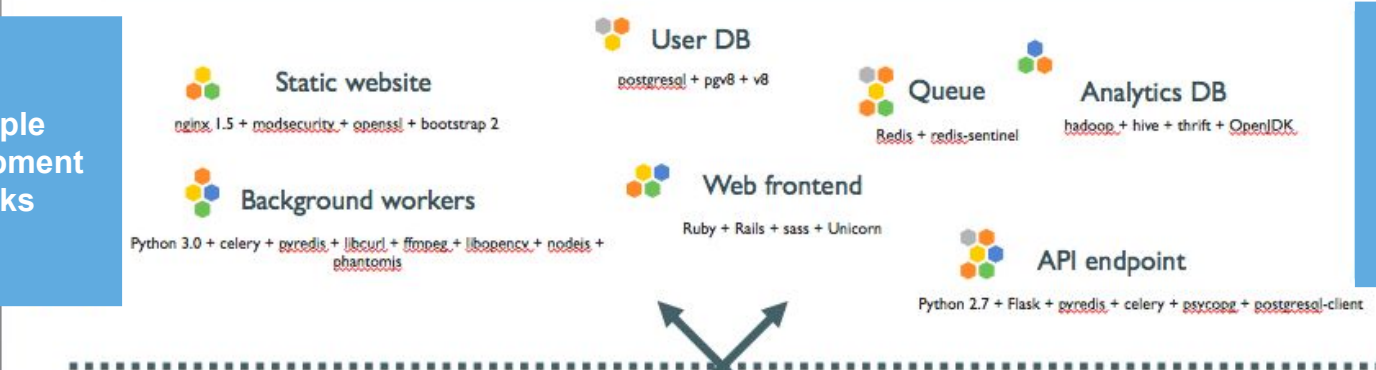
From https://www.docker.com/what-docker

Transforming the Application Landscape



The deployment nightmare

Multiple development stacks



Do services and apps interact appropriately?

Multiple hardware environments

Can I migrate quickly and smoothly?

A shipping analogy

Multiple types of goods

Do I worry about how goods interact? (i.e. place coffee beans next to spices)

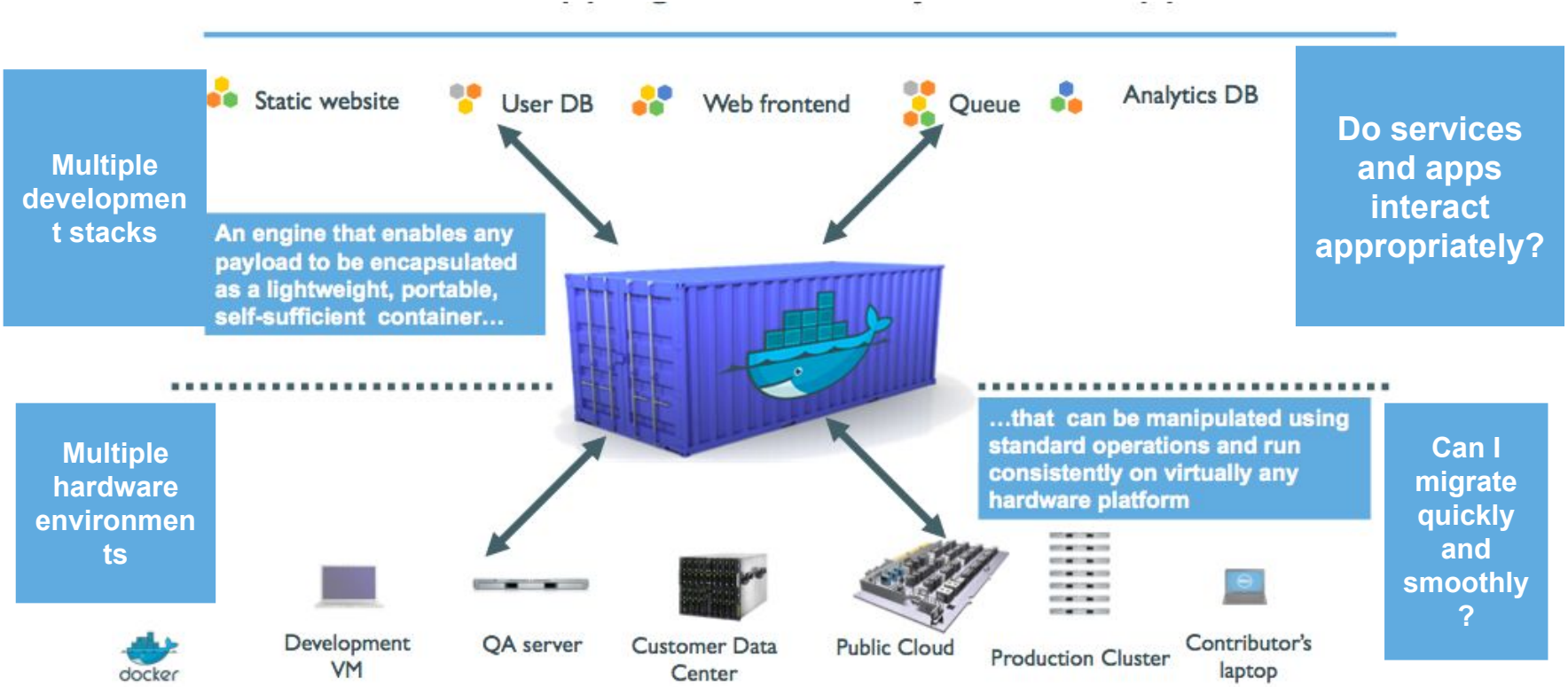
Multiple methods of transportation

Can I transport quickly and smoothy? (i.e unload from ship onto train)

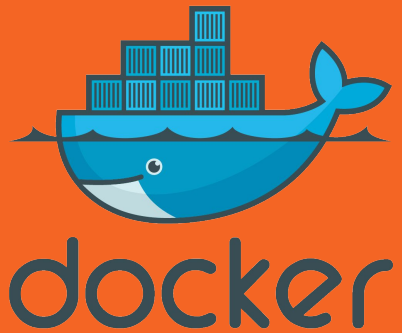
docker

The shipping container

Docker containers



Let's get started...



http://docs.docker.com/

Running Docker - Linux

Installed via your favourite

Installing Docker

Documented install procedure on many platforms

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

Changes faster than I can update my slides

Recently: Split between Community and Enterprise Edition

https://www.docker.com/community-edition

Running Docker - Linux

Docker is built on Linux kernel features

Runs natively on Linux

Most modern Linux flavours (Min. kernel 3.10)

Running Docker - Mac

Requires a virtual Linux - but don't worry

"Docker for Mac"

Integrated with the MacOS Hypervisor framework, networking and filesystem

Running Docker - Windows

Requires a virtual Linux - but don't worry

"Docker for Windows"

Native Windows app deeply integrated with Hyper-V virtualization, networking and file system

Ability to toggle between Linux and Windows Server environments to build applications

Are we there yet?

```
$  docker info
```

Are we there yet?

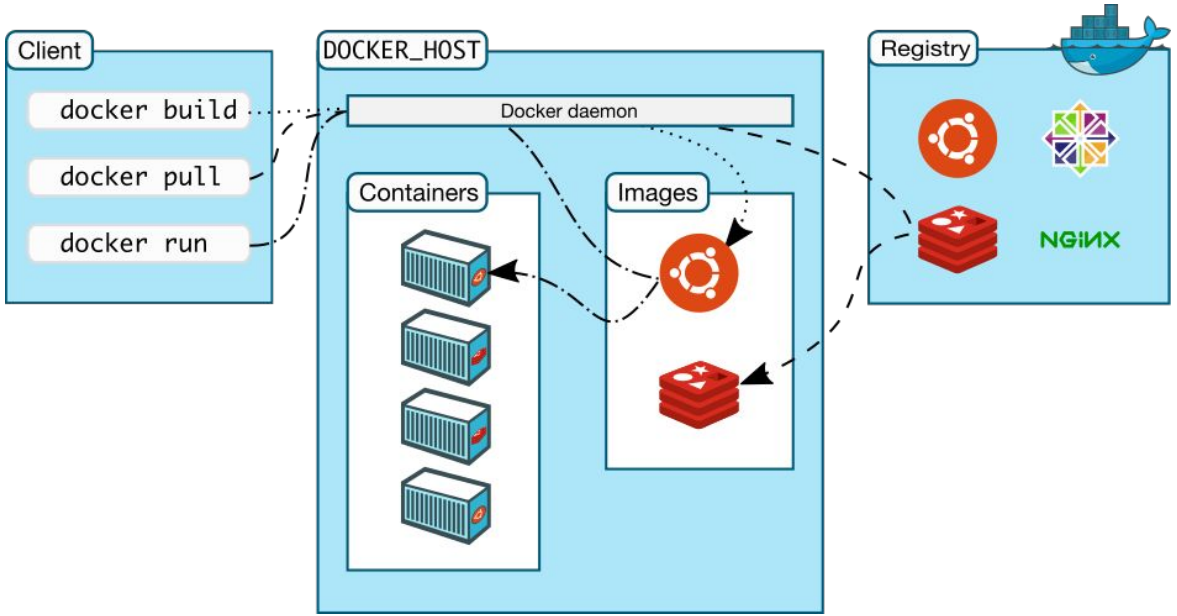
```
$  docker version
```

Let’s create some containers!

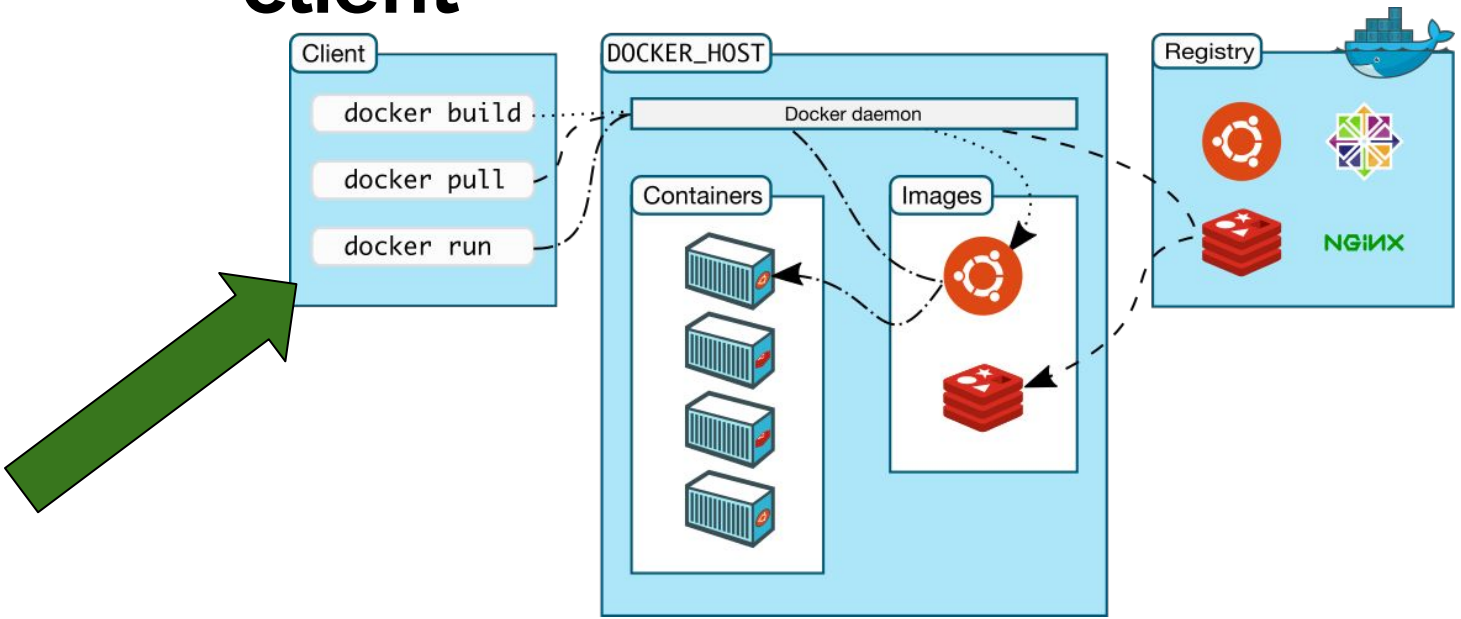
Hello, IDA!

```
$ docker run hello-world
```

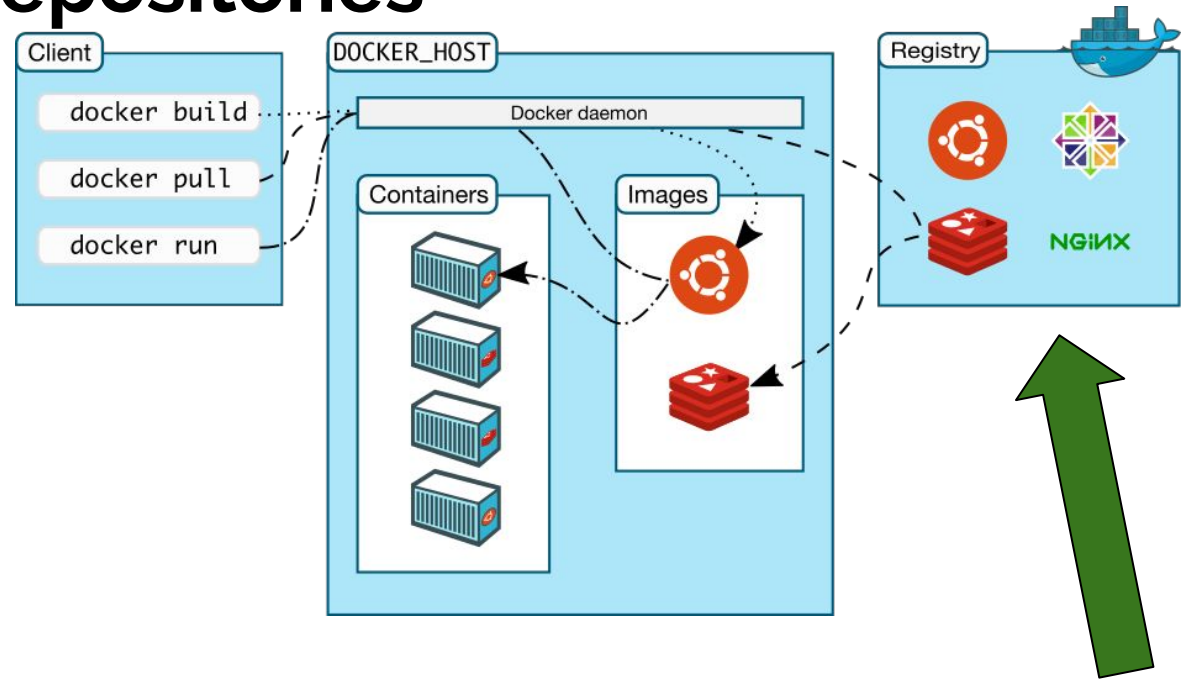
What just happened there then?



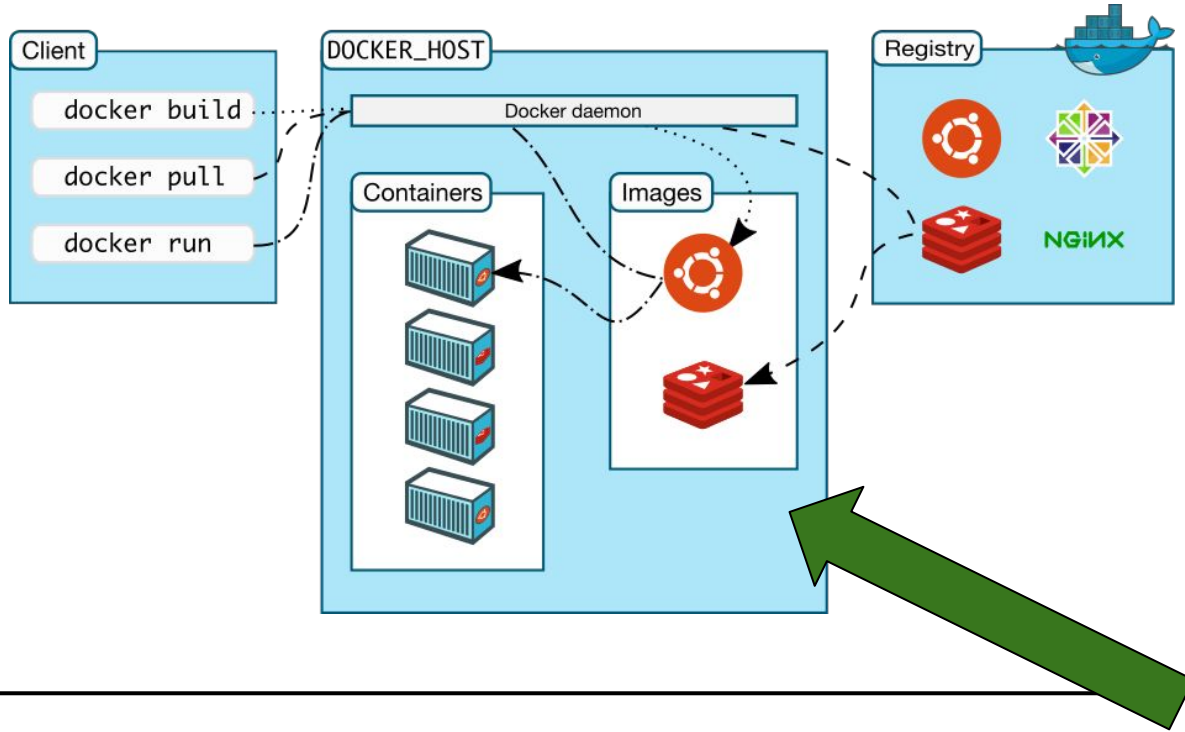
Commands are executed on the client



Images are pulled from repositories



Containers are run from images



An container is...

- an isolated and secure application platform
- run, started, stopped, moved, and deleted
- created from a Docker image

Let's get another

```
$ docker pull alpine
```

Find out what images you have

```
# docker images
```

Image Tags

Images are specified by repository:tag

Default tag is latest

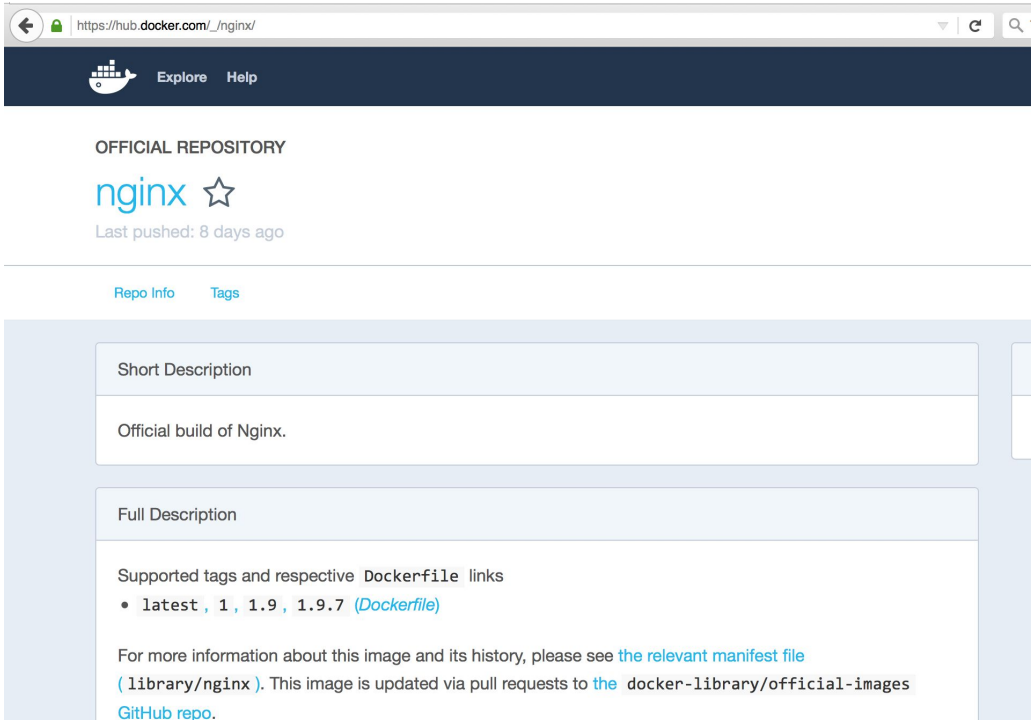
Run a command

```
$ docker run alpine ls -l
```

```
$ docker run alpine echo "hello from alpine"
```


PRAQMA

Docker hub


https://hub.docker.com/_/nginx/
Explore
Help
OFFICIAL REPOSITORY
nginx
Last pushed: 8 days ago
Repo Info
Tags
Short Description
Official build of Nginx.
Full Description
Supported tags and respective Dockerfile links
latest, 1, 1.9, 1.9.7 (Dockerfile)
For more information about this image and its history, please see the relevant manifest file (library/nginx). This image is updated via pull requests to the docker-library/official-images GitHub repo.

Let's saturate the network!

```
$ docker run ubuntu:14.04 echo "hello world"
$ docker run ubuntu:14.04 ps aux
```

The second run should be faster because there is no download

Let’s run a container with a terminal

```
$ docker run -i -t ubuntu:14.04 /bin/bash
```

`-i` flag tells docker to connect to STDIN on the container
`-t` flag specifies to get a pseudo-terminal

Look at our running containers

```
$ docker ps
```

List all containers
Containers have ID’s and Names

```
$ docker ps -a
```

Use the -a flag to include stopped containers

Use detached mode to run a container in the background

```
$ docker run -d ubuntu:14.04 ping 127.0.0.1 -c 50
```

Use `docker logs [containerID]` to get the output
`-f` is a useful flag

What is happening inside?

```
$ docker logs [containerID]
$ docker logs -f [containerID]
```

Time for a web server!

```
$ docker run -d -P nginx
```

Use `docker ps` to get the nginx port mapping

Container processes

```
$ docker run ubuntu:14.04 echo "hello"
$ docker run -ti ubuntu:14.04 /bin/bash
root@1234dfs:/# ps -ef
CTRL + P + Q
$ ps -ef
```

A container only runs as long as it’s process
Your command’s process is always PID 1 in the container

Getting back in

```
$ docker attach <container-id>
```

Containers have ID’s and Names
Either can be used

Stopping the container

```
$ docker stop <container-id>
```

```
$ docker ps
```

Now it is not listed anymore

What about exited containers?

```
$ docker ps -a -f status=exited
```

Let’s add something to our container

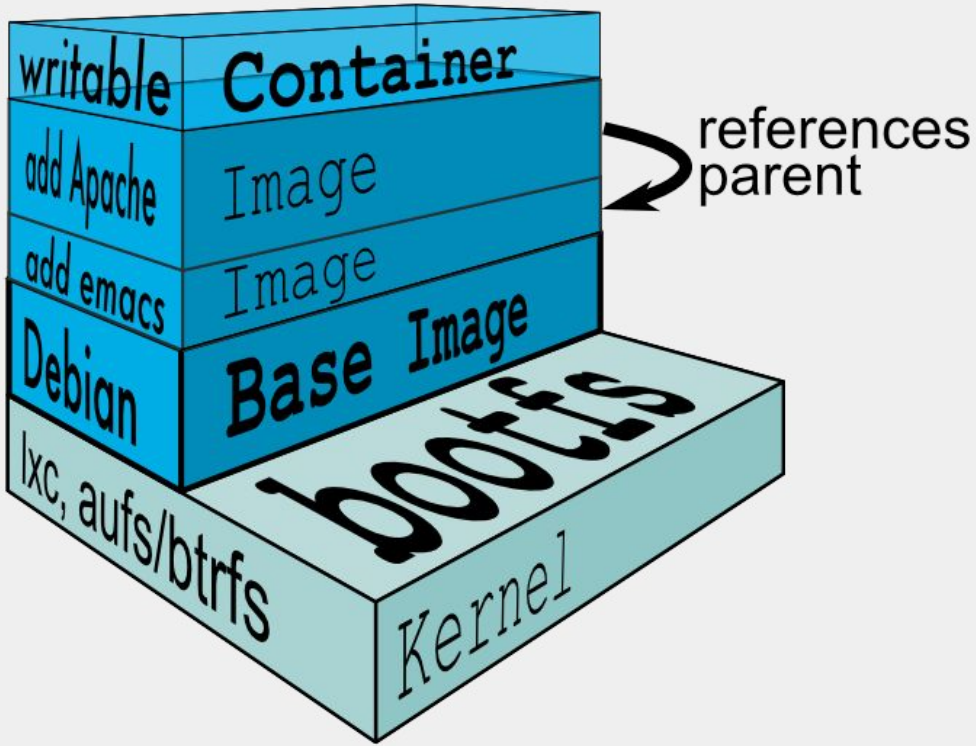
```
$ docker run -i -t ubuntu:14.04 /bin/bash
```

Let’s add something to our container

```
$ apt-get update
$ apt-get install nano
$ nano test.txt
$ exit
```

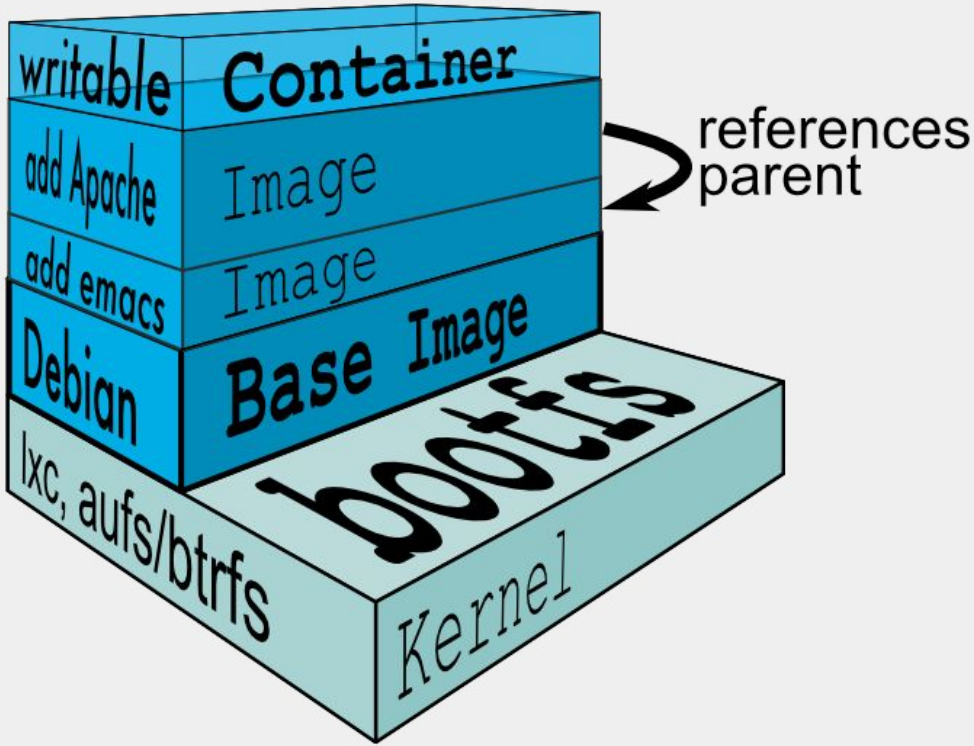
Images

An image is...

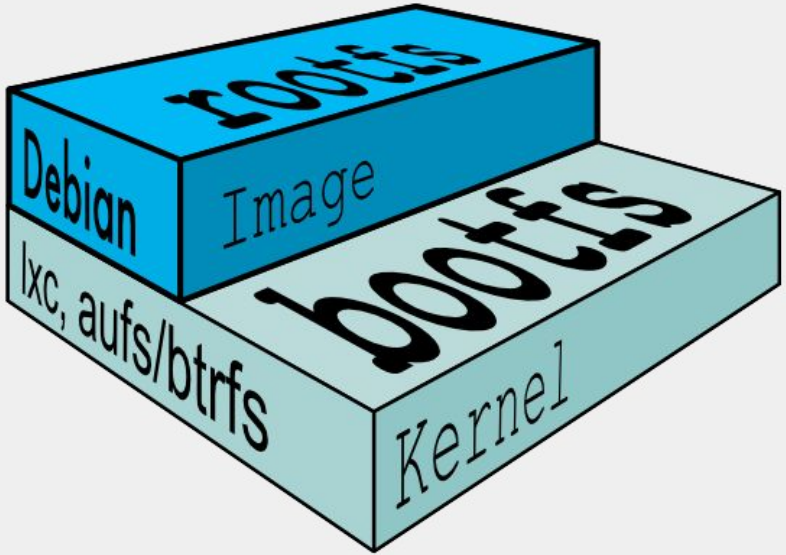


- A read-only template for creating containers
- The **build** component of docker
- Stored in registries
- Can be created by yourself distributed by others

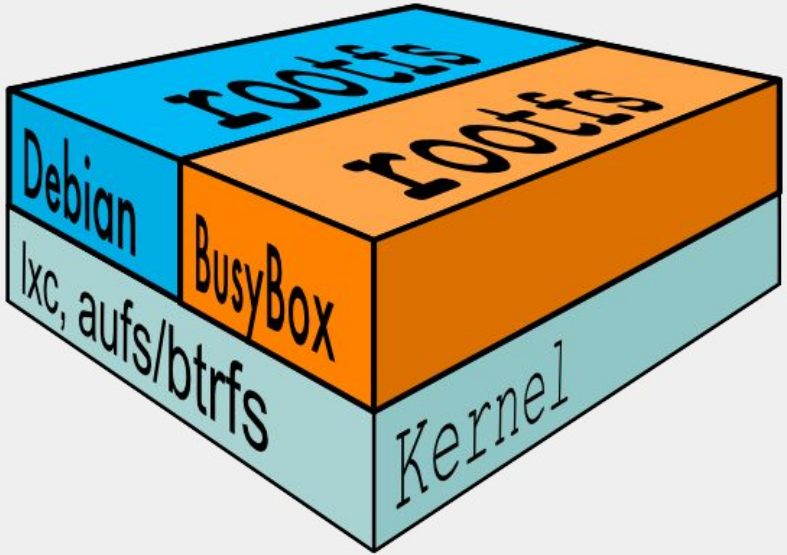
Images are layered read-only filesystems



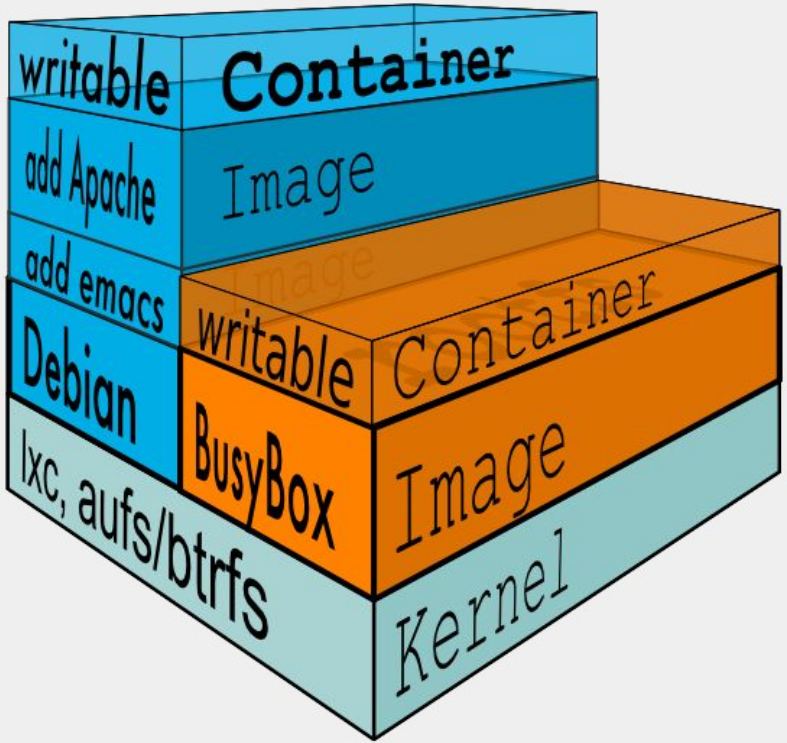
Images have base layers



Multiple root file systems per host are normal



When an image is run, a writable layer is added



Let’s make an image

Docker commit saves changes in a container as a new image

```
$ docker commit 234d3ea32 simple:1.0
```

Let’s run our new image

```
$ docker run -it simple:1.0 bash

root@2343245:/# cat test.txt
root@2343245:/# nano test.txt
```

The Dockerfile

The Dockerfile

A **Dockerfile** is a configuration file that allows us to specify instructions on how to build an image

It enables **configuration as code**

More **effective** than using `commit`

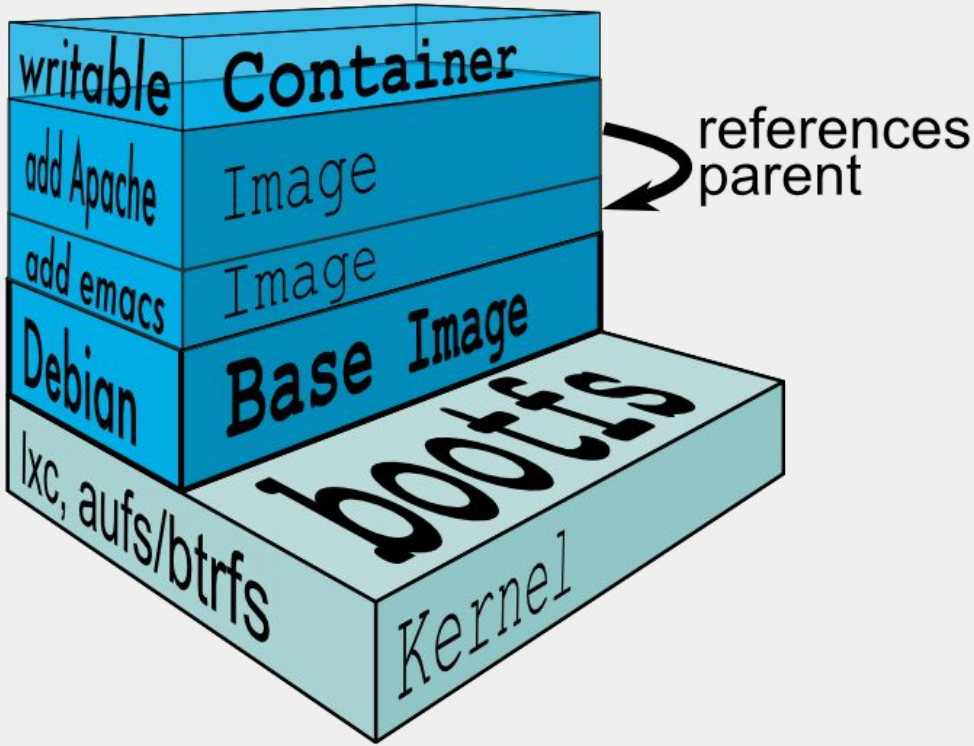
- Share the configuration rather than image
- Supports continuous integration
- Easier to review
- Easier to update

Dockerfile instructions

```
# Dockerfile for myapp
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y curl
RUN apt-get install -y vim
```

The default name for the file is **`Dockerfile`**

Run instructions are executed in the top writable layer



Aggregating RUN instructions to reduce layers

```
# Dockerfile for myapp
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y \
  curl \
  vim
```

Building an image from a Dockerfile

```
$ docker build -t simple:1.1 .
```

The build command takes a build context on the filesystem
`-f` flag can be used to specify a different Dockerfile location

The CMD instruction

```
# Dockerfile for myapp
FROM ubuntu:14.04
RUN apt-get install -y vim
CMD [“PING”, “127.0.0.1”, “-c”, “10”]
```

Can only be defined once
Can be overridden at run time

The ENTRYPOINT instruction

```
# Dockerfile for myapp
FROM ubuntu:14.04
...
ENTRYPOINT [“PING”]
```

Can have a CMD in addition

Other notable Dockerfile commands

```
# Dockerfile for myapp
EXPOSE 80
ENV JAVA_HOME /usr/bin/java
COPY index.html /var/www
ADD robots.txt /var/www
```

Dockerfile best practices

Containers should be ephemeral

Use a `.dockerignore` file to exclude unnecessary files from the build context

Avoid including unnecessary packages and dependencies

Run only one process per container

Minimize the number of layers

Use the build cache to your advantage

Managing Containers

Other notable commands

```
$ docker run -d nginx
$ docker stop [CONTAINER_ID]
$ docker start [CONTAINER_ID]
```

Getting terminal access to a container

```
$ docker exec -it [CONTAINER_ID] bash
```

Removing containers

```
$ docker rm [CONTAINER_ID]
```

Will only remove stopped containers

Deleting images

```
$ docker rmi simple:1.0
```

Wipe em all out

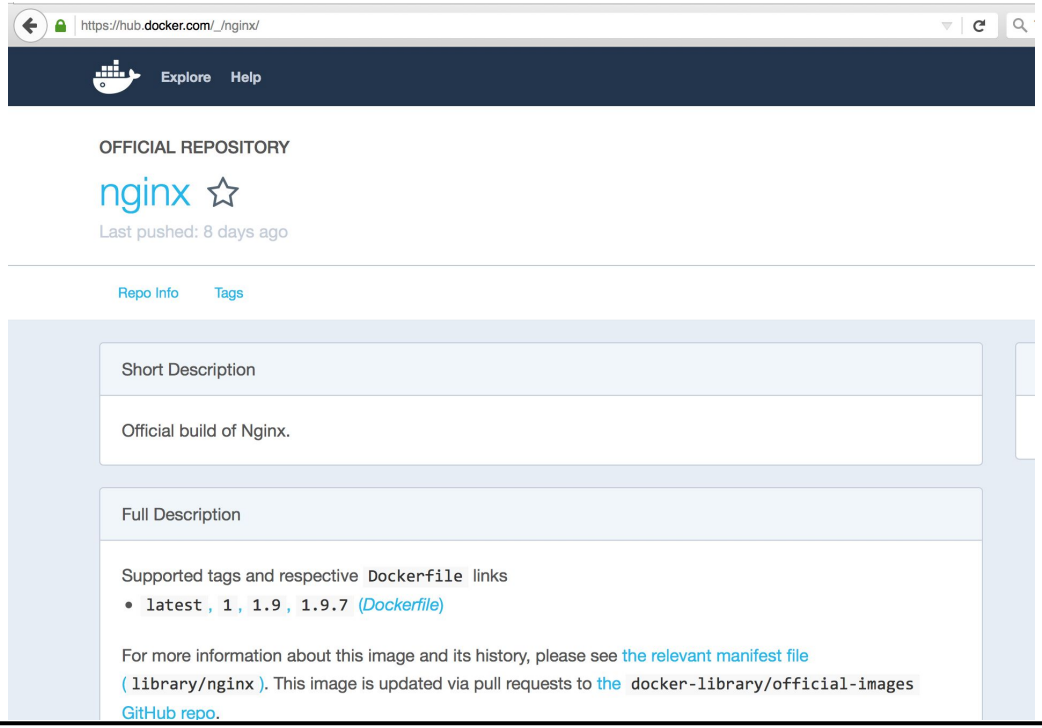
```
$ docker rm -f $(docker ps -a -q)
```

Kitematic

- GUI for Docker
- Start and stop containers
- Connects to Docker Hub to access your images
- Runs on Mac OSX and Windows
- Support for auto update

Sharing containers

Let's add our repository on hub



Make a tag that matches our repository on hub

```
$ docker tag simple:1.0 jkrag/idademo:1.0
```

Push to hub

```
$ docker push jkrag/idademo:1.0
```

Docker volumes

A volume is a directory in a container used for persistence

- **Survive beyond the lifetime of a container**
- **Can be mapped to a host folder**
- **Can be shared amongst containers**

A volume is a directory in a container used for persistence

```
$ docker run -d -P -v /tmp/myapp/html/:/www/website
nginx
$ docker exec -ti [ID] bash
$ ls /var/www/html
```

Docker volume command

- `docker volume create`
- `docker volume ls`
- `docker volume inspect`
- `docker volume rm`

You can also add volumes in the Dockerfile

```
# create a volume
VOLUME /myvol

# multiple volumes
VOLUME /myvol1 /logs

# json syntax
VOLUME [“myvol1”,”myvol2”]
```

Volume best practices

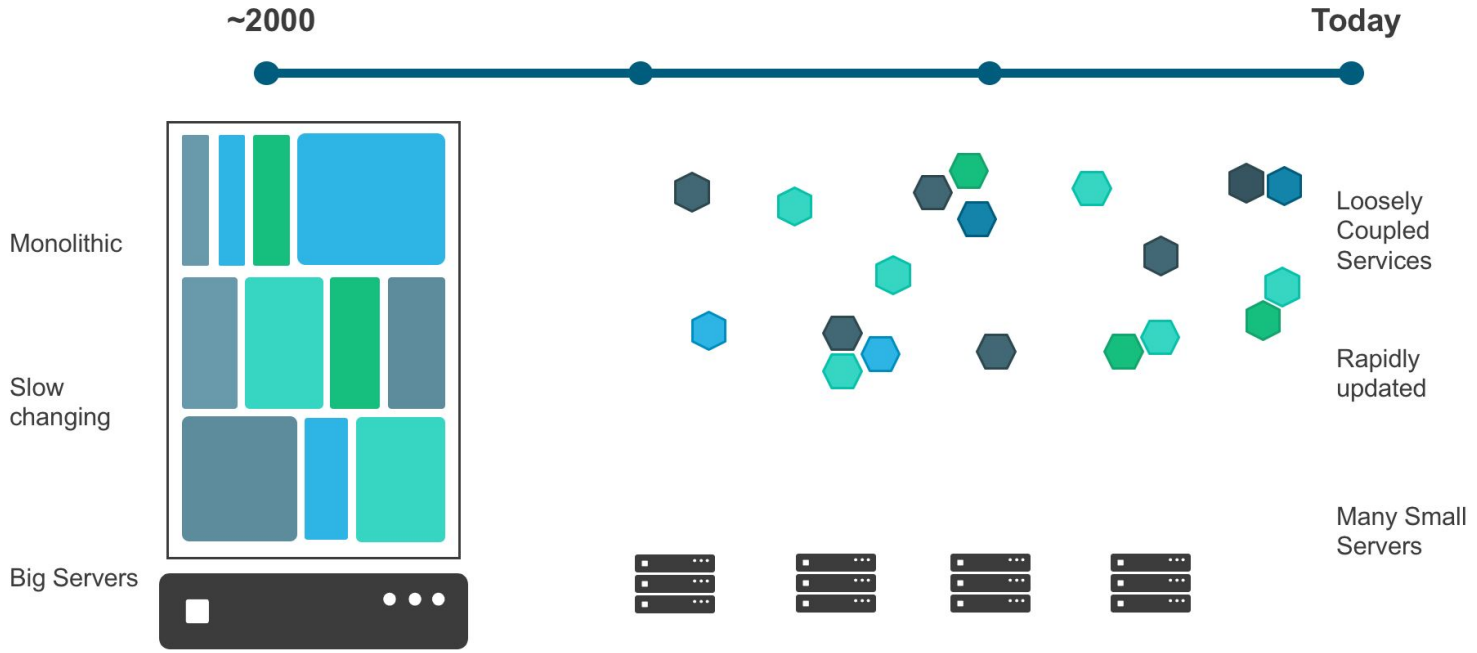
Containers should be ephemeral

Avoid mounting directories from the host in production

Data containers are recommended

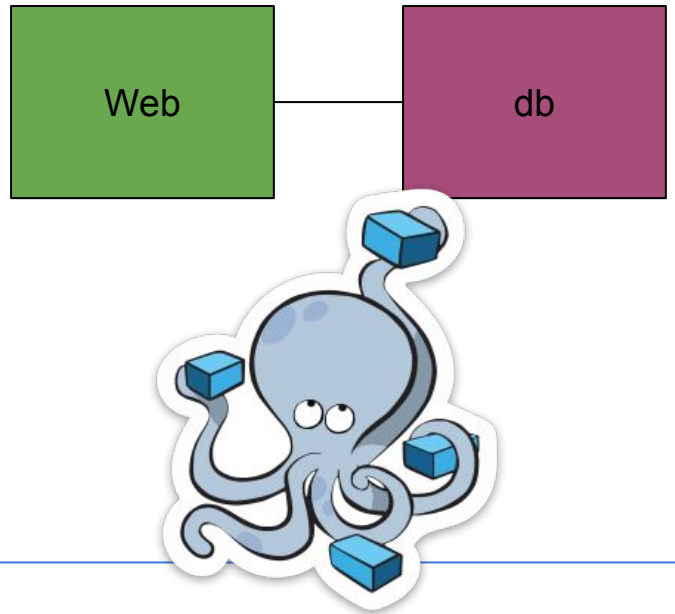
Docker compose

Transforming the Application Landscape



Using docker-compose to create multi-container apps

```
web:
  build: .
  ports:
   - "5000:5000"
  volumes:
   - .:/code
  links:
   - redis
redis:
  image: redis
```

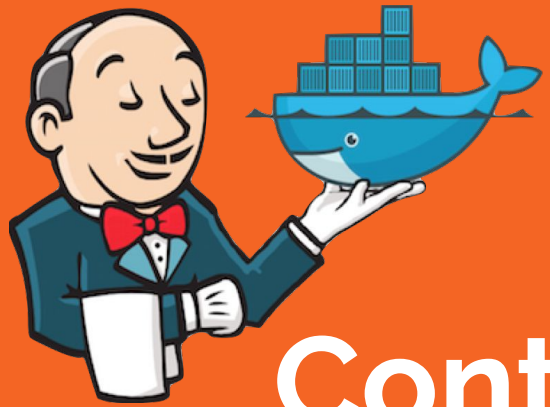


Using docker-compose

```
$ docker-compose up
$ docker-compose -d up
$ docker ps
$ docker-compose ps
$ docker-compose start <service name>
$ docker-compose stop <service name>
$ docker-compose rm <-v> <service name>
```

Using docker-compose continued...

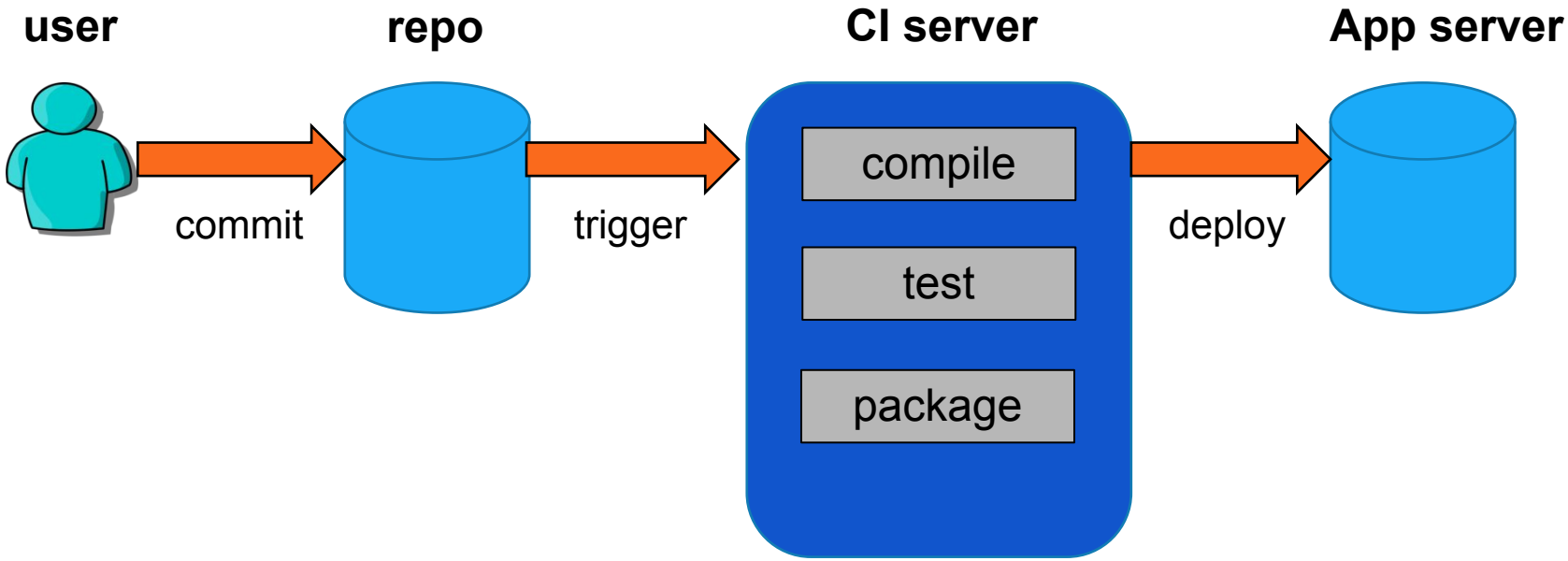
```
$ docker-compose logs
$ docker-compose scale
```



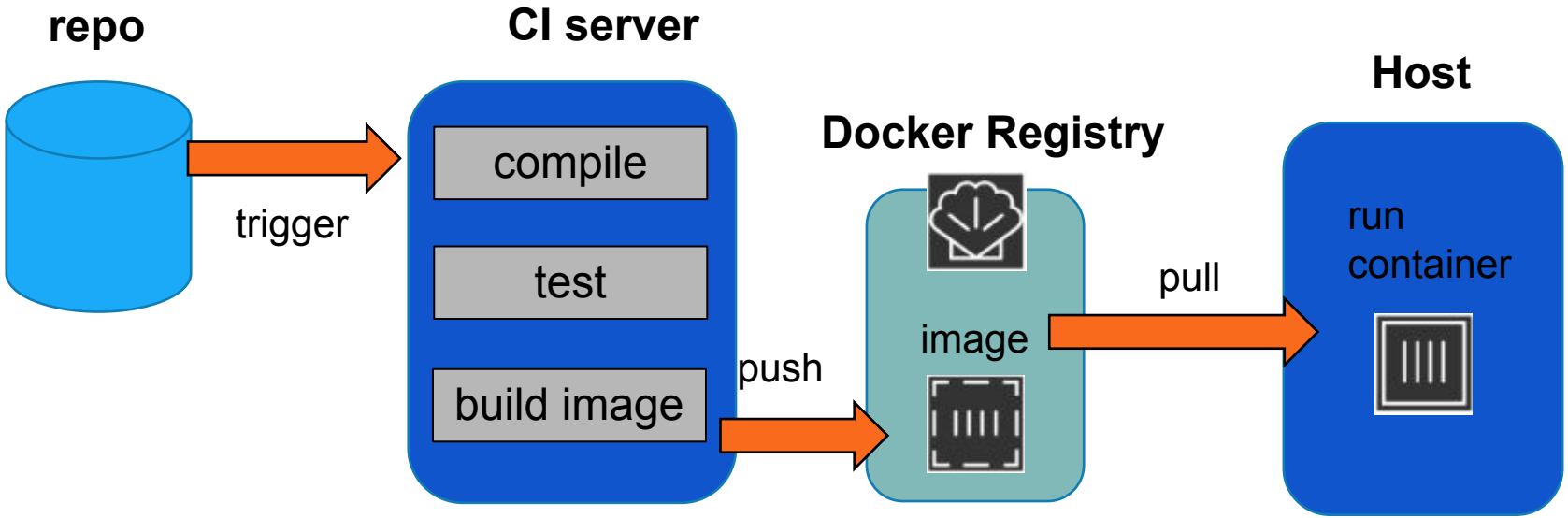
Continuous Delivery

with Docker

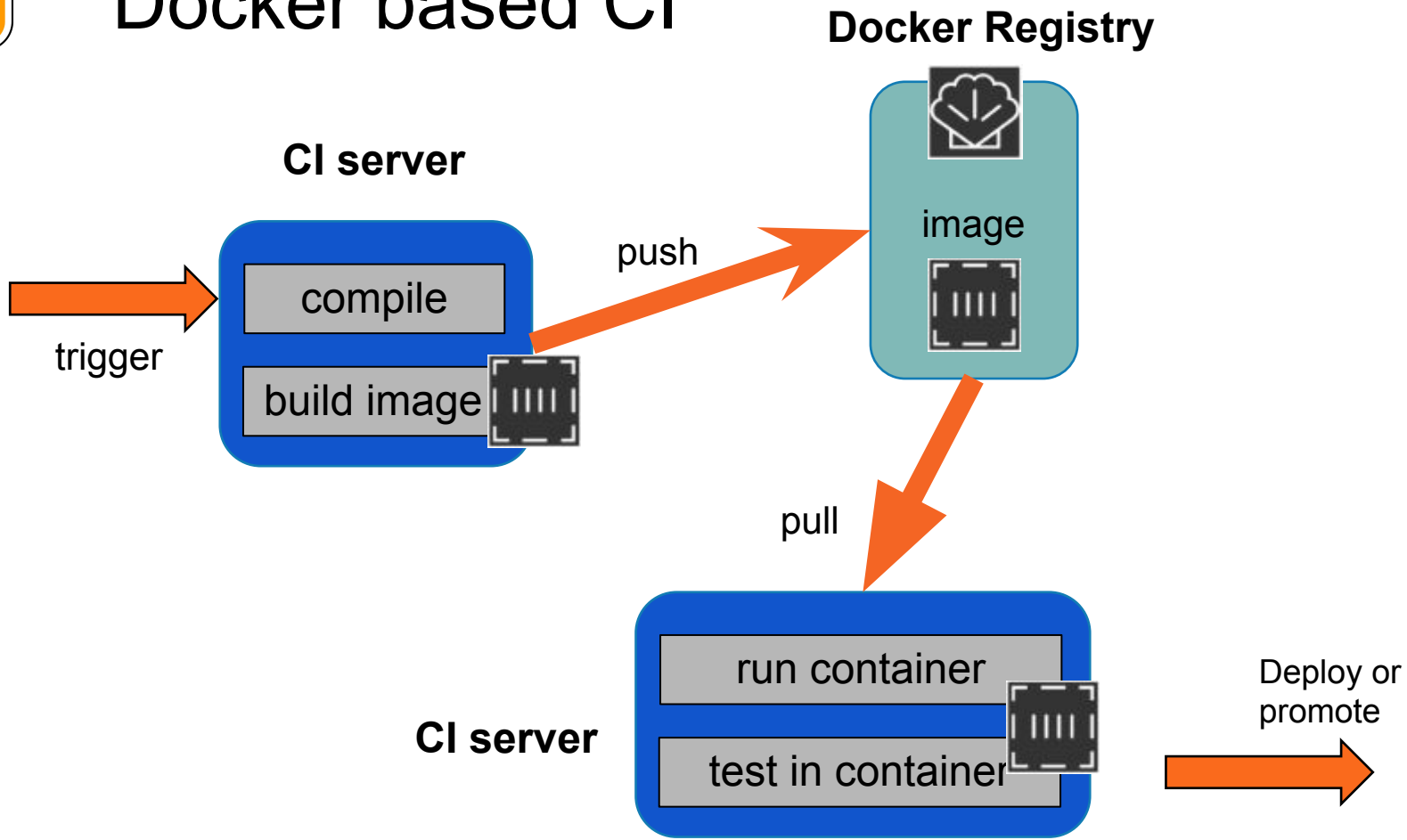
Traditional CI server



Docker based CI

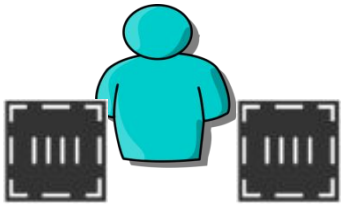


Docker based CI

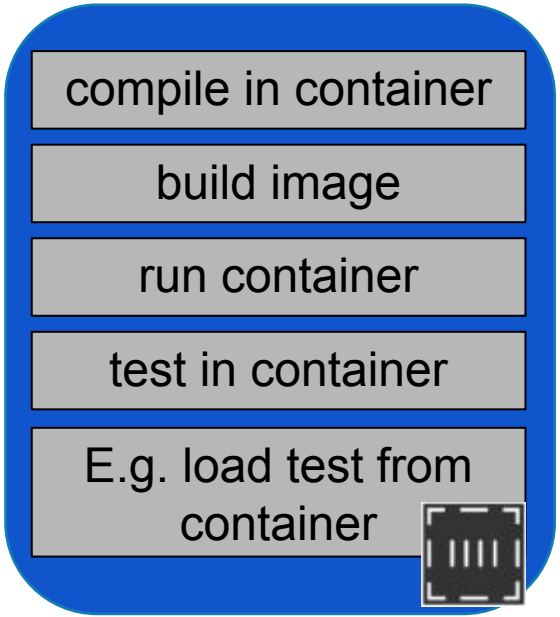


Docker based CI

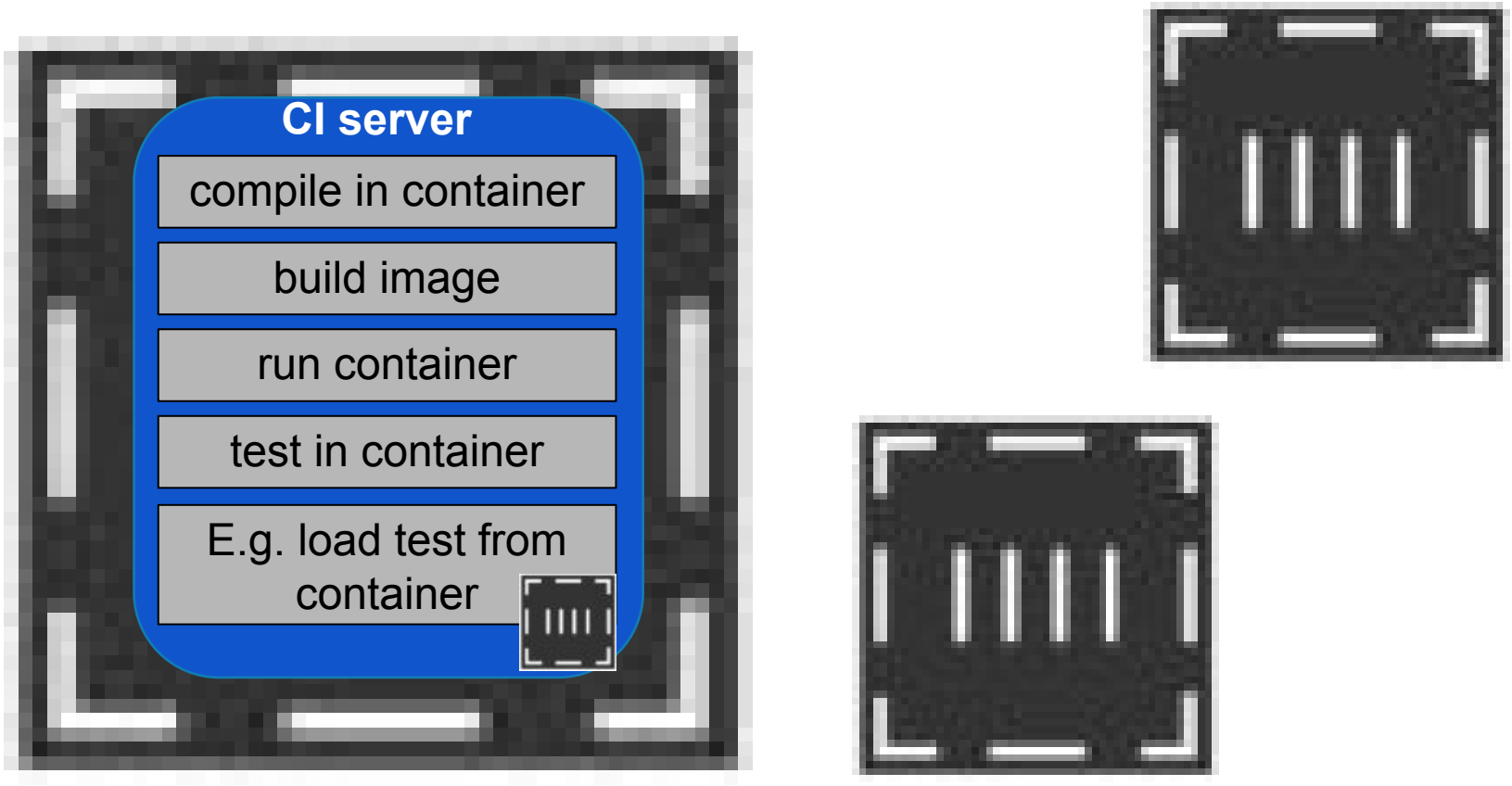
dev



CI server



Docker based CI environment



Advantages for CoDe

Need different sets of compilers or versions

Isolation and reproducibility of complex build environments

No conflicts between different tools

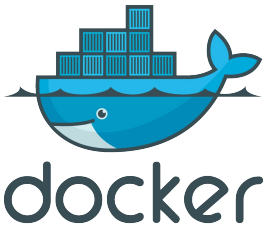
Rather than keeping a physical or virtual machine around you can just keep the Dockerfile that built a release

Good for having twenty years of accountability

Docker Hub's auto build

Where are we now?

A brief tour of Docker



We have now covered:

- What is a container and why you may want one
- How to run and manage containers
- How to create your own containers
- How to share your containers
- How to create multi-container applications
- How Docker supports Continuous Delivery

Q & A