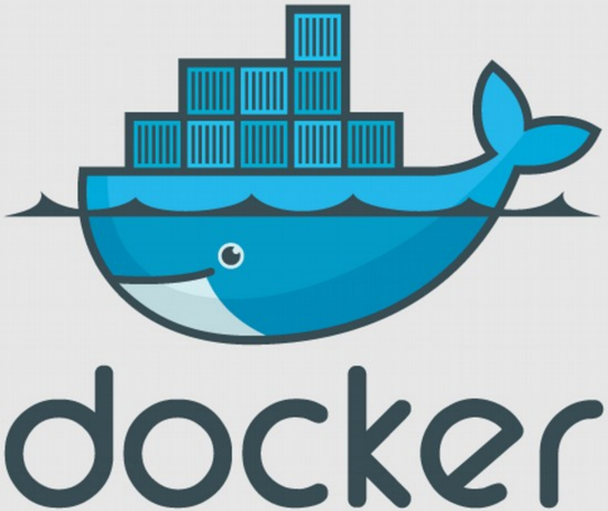


https://www.docker.io/

What is it?

- Docker is an open-source project to easily create lightweight, portable, self-sufficient containers from any application
- Makes fast running, portable **containers**

History

- Dotcloud, Inc creates PaaS service
- January 2013, work starts on docker internally
- March 2013, first public release
- Statistics: (moby project)
  - 56700 stars on github
  - 16400 forks
  - 1843 contributors
  - 38439 Commits
- Massive community interest
- Created by Solomon Hykes (French engineer ;)

EVER TRIED.
EVER FAILED.
NO MATTER.
TRY AGAIN.
FAIL AGAIN.
FAIL BETTER.

Samuel Beckett (1906-1989)

Why this hype?

- Solves an important problem
- Easy to use
- Efficient

Who uses Docker?

Companies using Docker

GILT

Baidu 百度

Yandex

yelp

rackspace
the #1 managed cloud company

ebay

Spotify

New Relic.

And many more…

Who uses Docker?

Docker PAAS Providers

Microsoft Azure

dotCloud

Google Cloud Platform

tutum

StackDock

amazon web services™

Quay.io

DigitalOcean

And many more…

Who uses Docker?

As an Infrastructure Tool along side

CHEF™

puppet labs

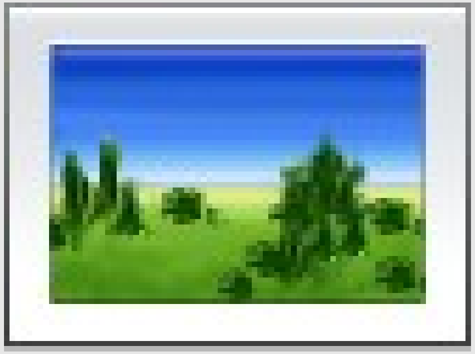
ANSIBLE

VAGRANT

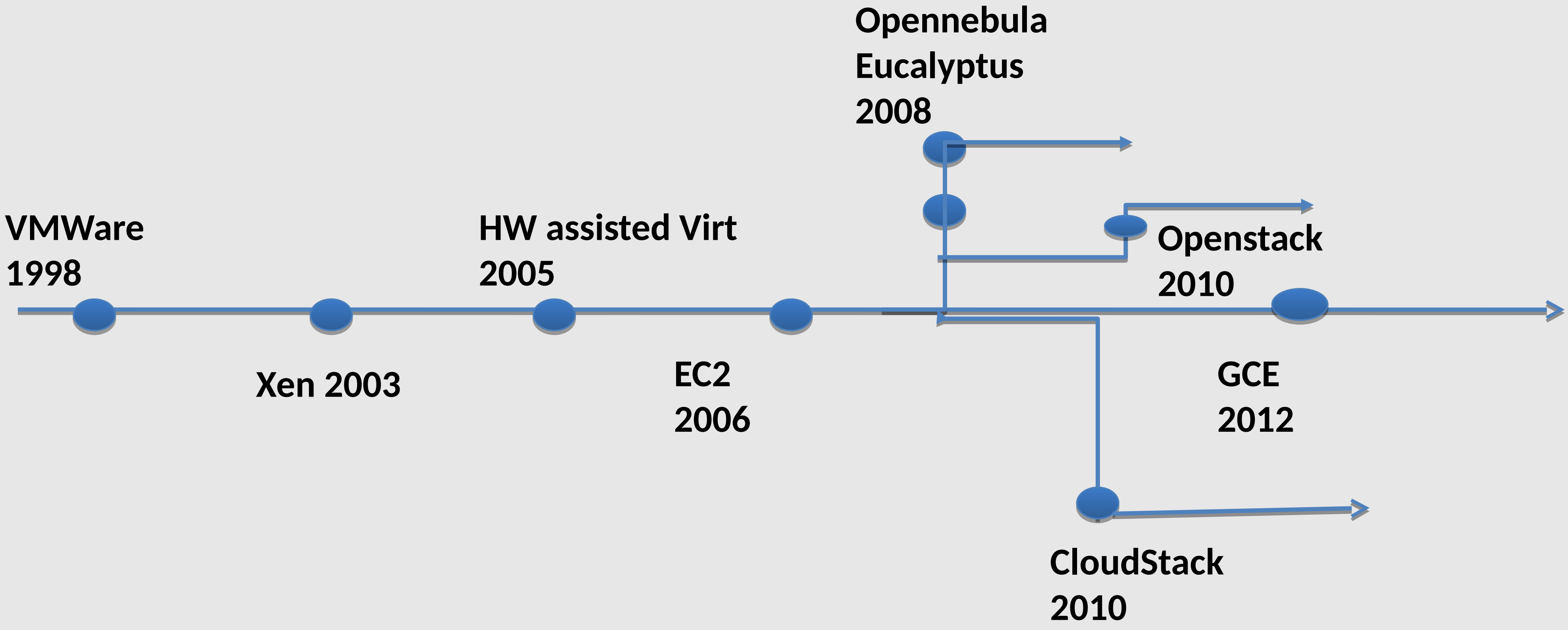
SALTSTACK

CFEngine®

Jenkins



IaaS History


Opennebula
Eucalyptus
2008
VMWare
1998
HW assisted Virt
2005
Openstack
2010
Xen 2003
EC2
2006
GCE
2012
CloudStack
2010

Goals

- Utility computing
- Elasticity of the infrastructure
- On-demand
- Pay as you go
- Multi-tenant
- Programmable access

So what…

Let’s assume this is solved.

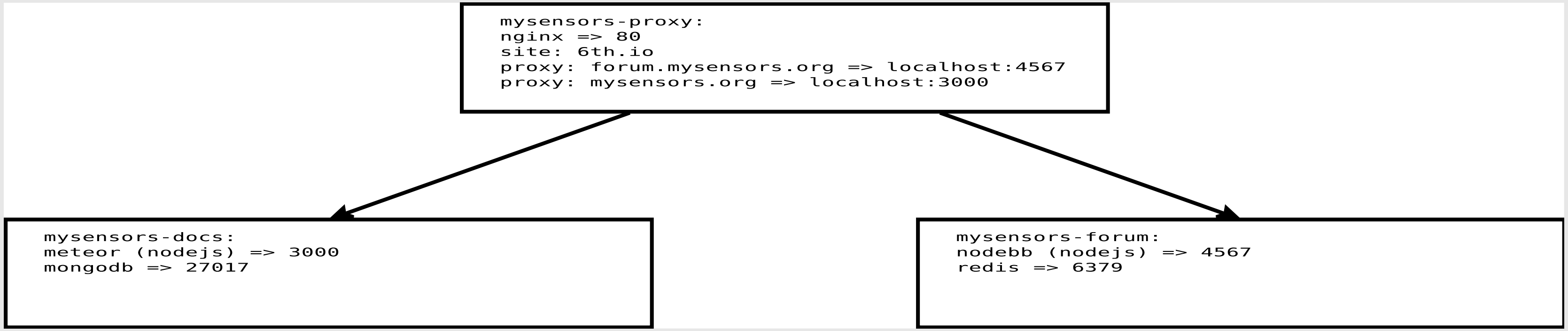
What is not solved:

- Application deployment
- Application scalability
- Application portability
- Application composability

Why docker?

Problem

- An application have many dependencies that needs to coexist
- Deployments with different demands:
  - Development/CI
  - Test
  - Production



```
mysensors-proxy:
nginx => 80
site: 6th.io
proxy: forum.mysensors.org => localhost:4567
proxy: mysensors.org => localhost:3000
```

```
mysensors-docs:
meteor (nodejs) => 3000
mongodb => 27017
```

```
mysensors-forum:
nodebb (nodejs) => 4567
redis => 6379
```

Application Deployment got complex

Application Stack

- Basic OS
- JVM
- Static web server
- Front-end platform
- Database layer
- Application code

X

Deployment environments

- Development VM
- QA Server
- Customer Data Center
- Public Cloud
- Contributors Laptop
- Production Servers
- Production Clusters

Problems

- Installing software
- Software versoning
- Configuration
- Testing

What is docker?

Analogy

- Transporting of goods before 1960

Solution: Shipping container

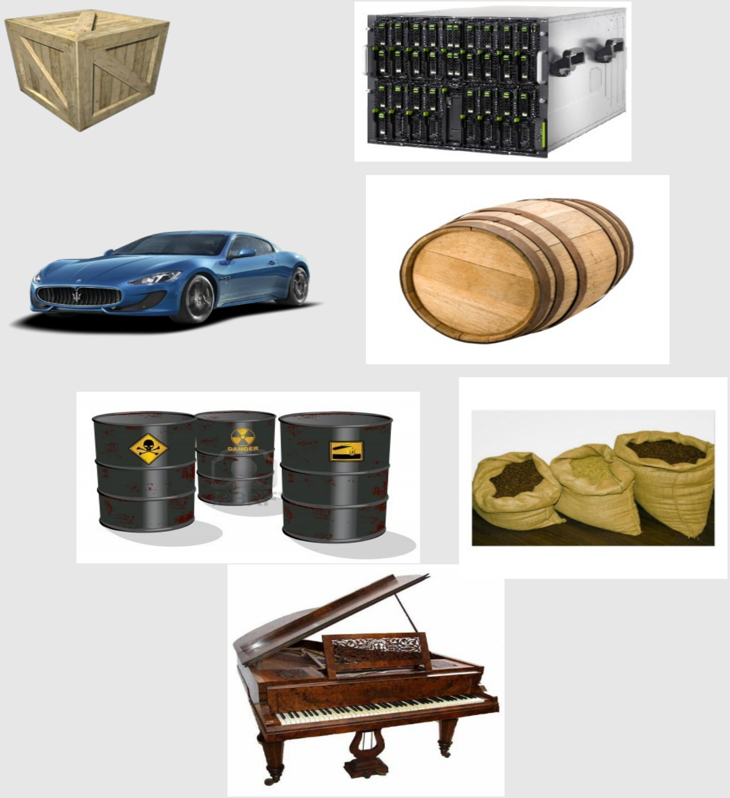
- Separation of concerns
  - User cares about packing the inside
  - Shipper cares about moving the container
- Standardized interface

What is docker?

Solution

- Containers

Pre-1960 shipping industry



x

Docker containers

Standardized interface for software container

Developer concerns
- Code
- Libraries
- Services
- Configuration
- Data

All servers look the same

Ops concerns
- Moving containers
- Starting/Stopping containers
- Logging
- Monitoring
- Network configuration

All containers look the same

Isolation

What is a Container?

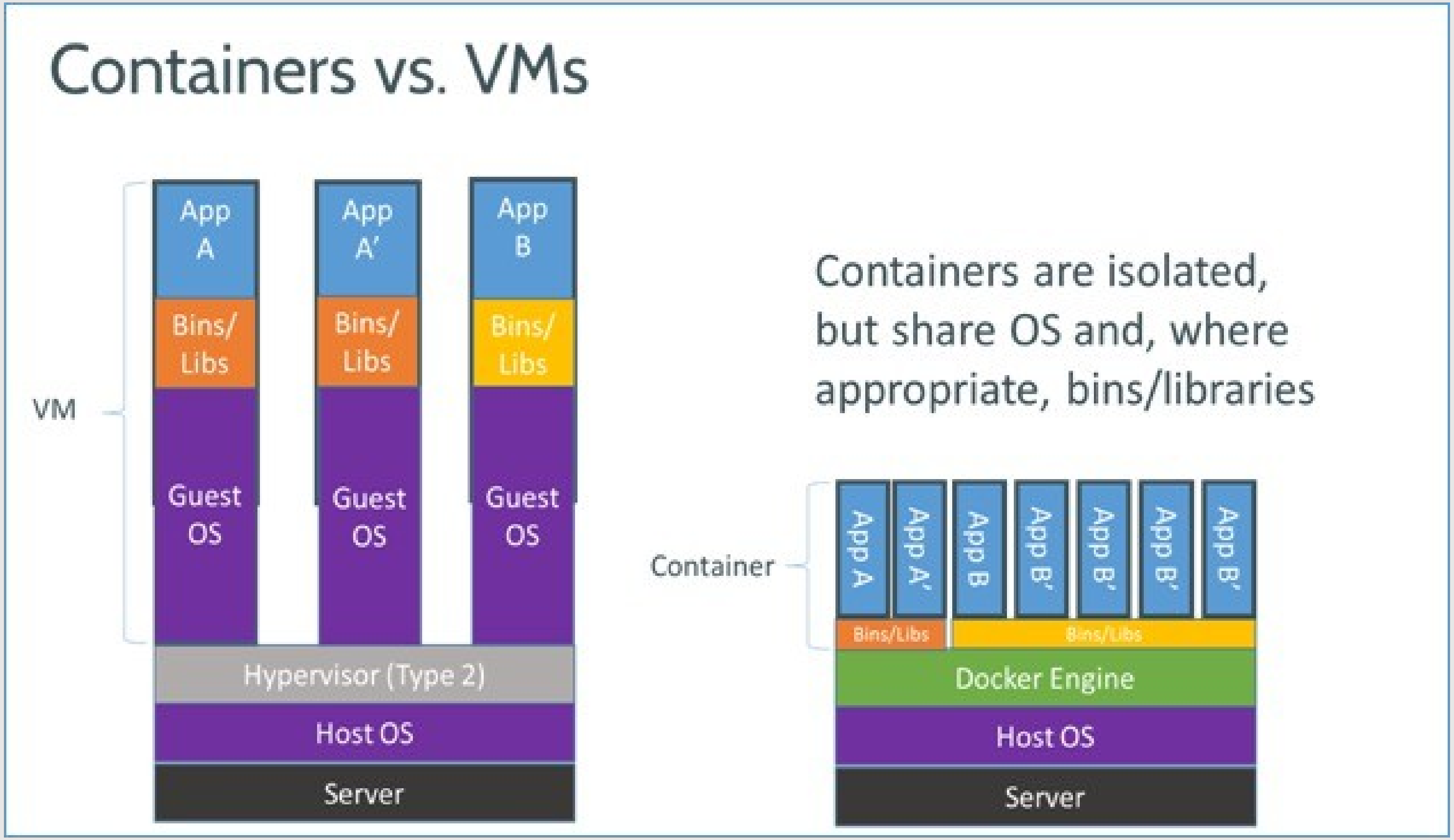
- **Operating system–level virtualization**
- Operating system–level virtualization is a server virtualization method where the kernel of an operating system allows for multiple isolated user-space instances, instead of just one
- Differs from traditional virtual machines

Containers vs. VMs

- Container
  - Shares the host OS and kernel
  - Zero boot time
  - Cannot run any OS (strictly Linux for Docker)
  - Little to no start-up or performance penalty. Technically native.

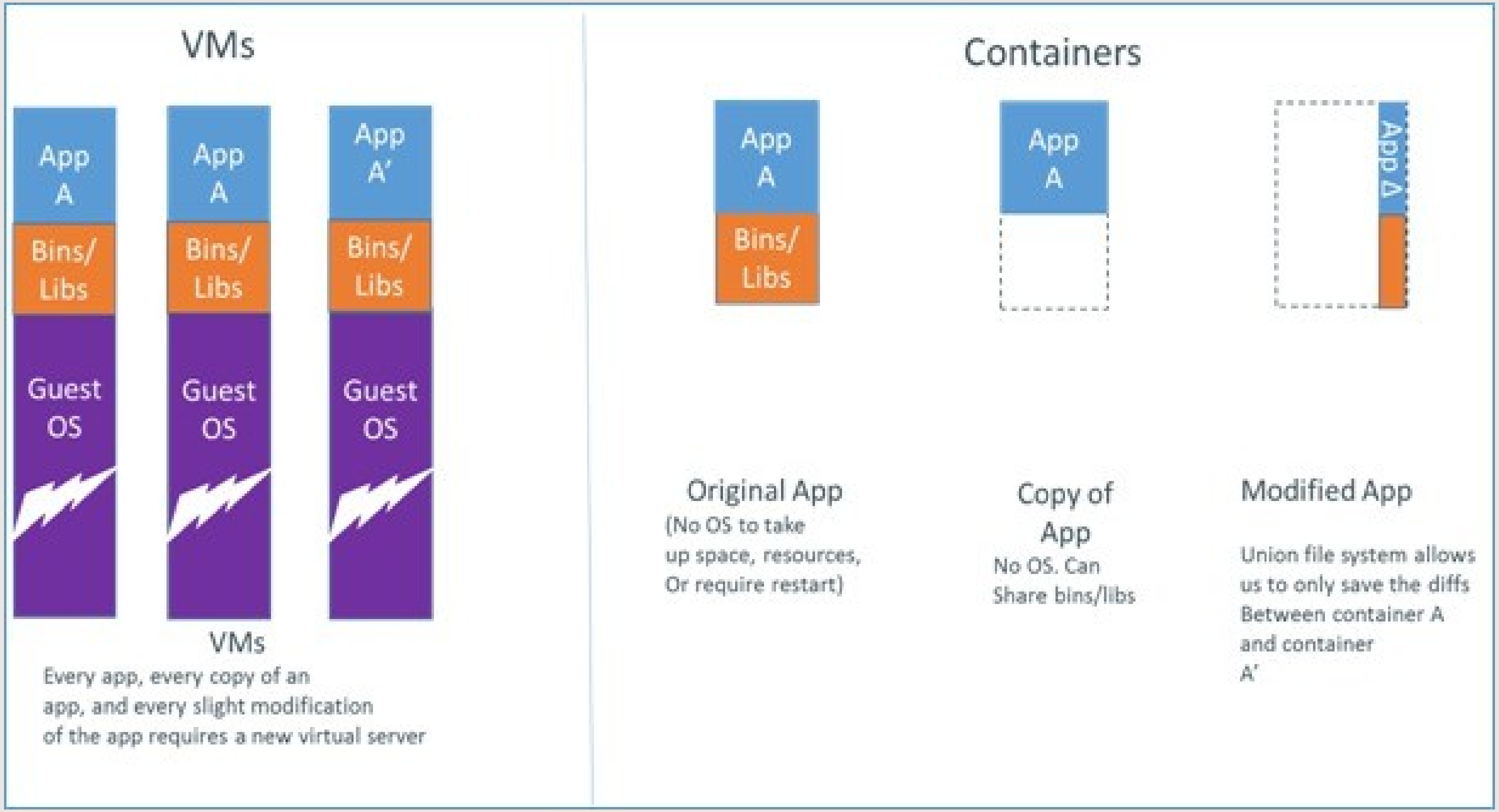
- Virtualization
  - Full OSes on top of a host OS via hypervisor
  - Full software stack
  - Each VM has it's own kernel
  - Full boot process for each VM (slow)
  - Can run any OS (Windows and BSD included)

Containers vs. VMs (Pretty Picture)



Pretty picture from https://www.docker.io/the_whole_story/

Containers vs. VMs (Another Pretty Picture)



Every app, every copy of an app, and every slight modification of the app requires a new virtual server

(No OS to take up space, resources, Or require restart)

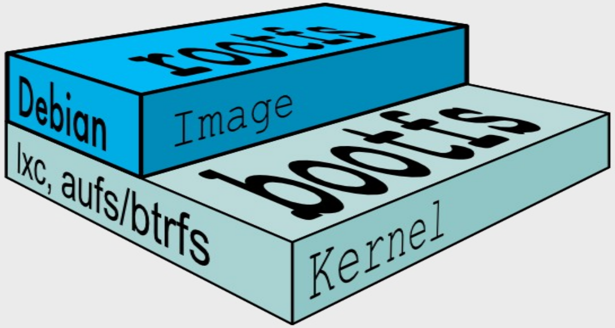
No OS. Can Share bins/libs

Union file system allows us to only save the diffs Between container A and container A'

Another pretty picture from https://www.docker.io/the_whole_story/

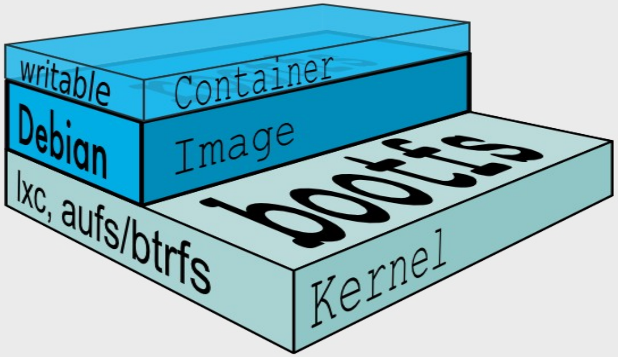
Docker glossary

- Image
  - Read-only template for a container
  - Includes all files required for application to run
  - Has additional metadata
    - Exposed network ports
    - Binary to start



Docker glossary

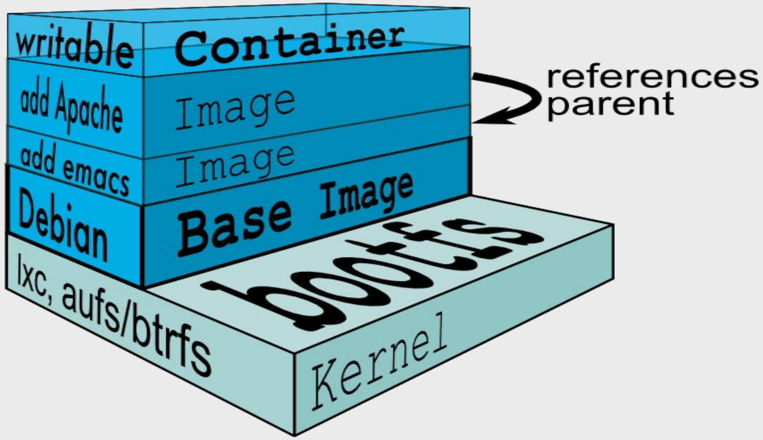
- Container
  - Running processes
  - Based on a particular image
  - Typically a single process
  - Isolated from host system
  - Cheap
  - Can write to filesystem
  - Commit creates new Image



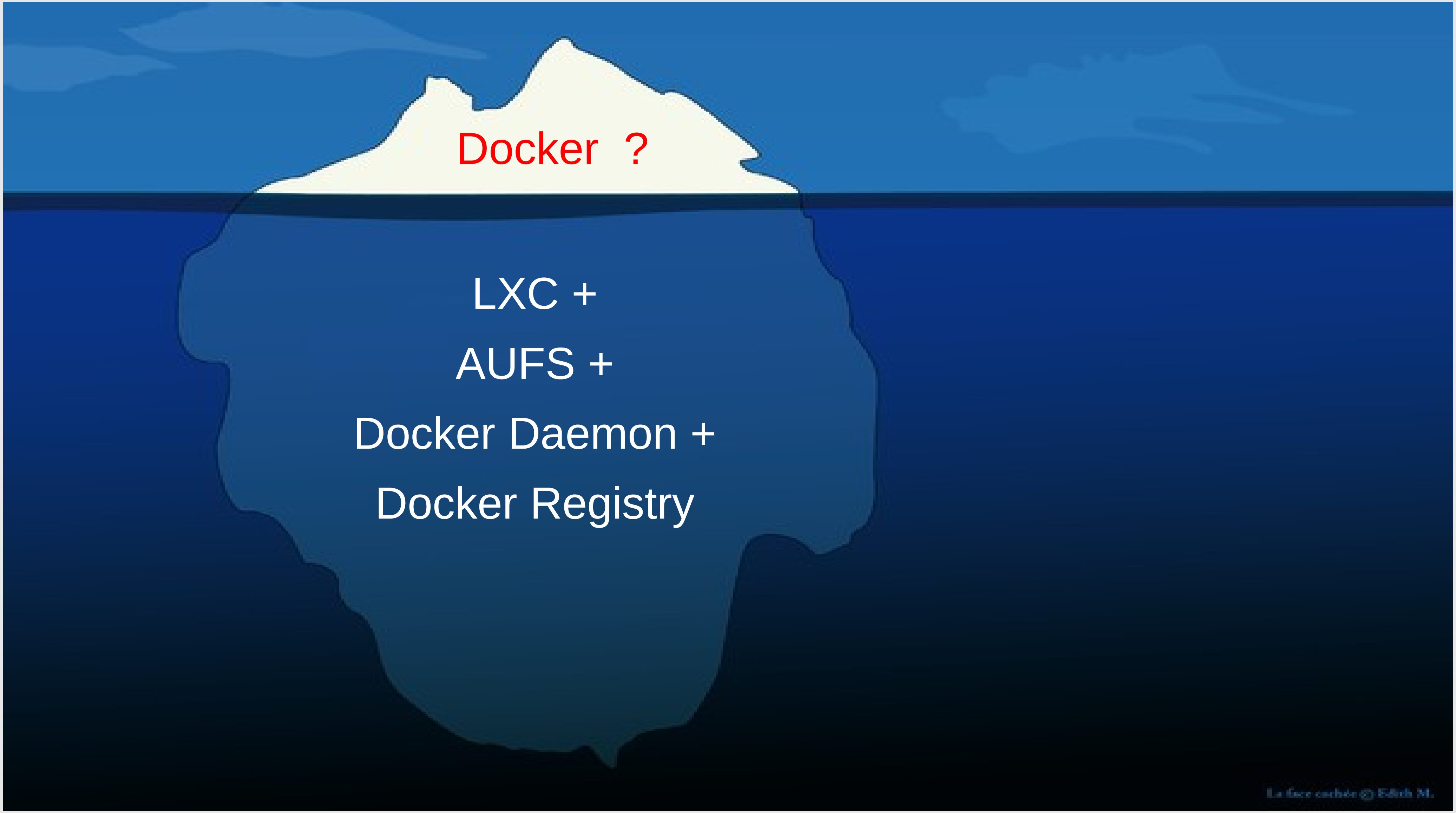
Docker glossary

Layers

- Images are based on a parent
- The layers stack on top
- Files in base layers are shared between Images
- Each commit creates a layer
- Base image has no parent



First Demo!



Docker ?

LXC +

AUFS +

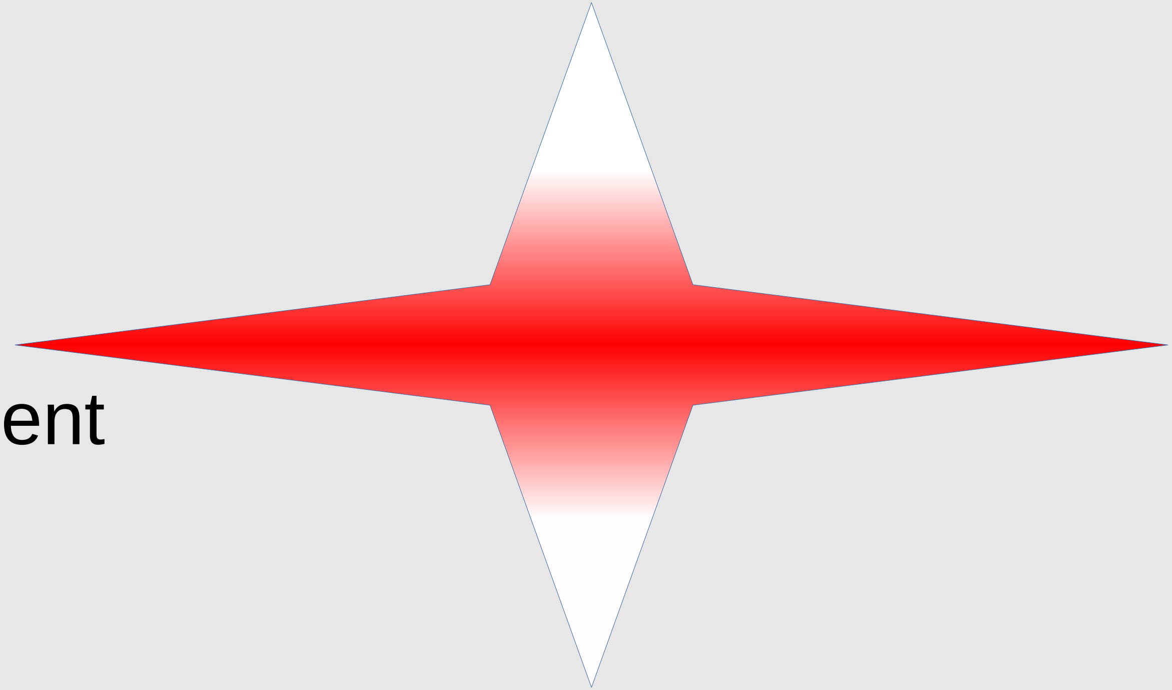
Docker Daemon +

Docker Registry

La face cachée © Edith M.

Containers are a userspace concept that takes advantage of several Kernel Subsystems

Key elements of Linux Containers



Process Isolation

Resource Management

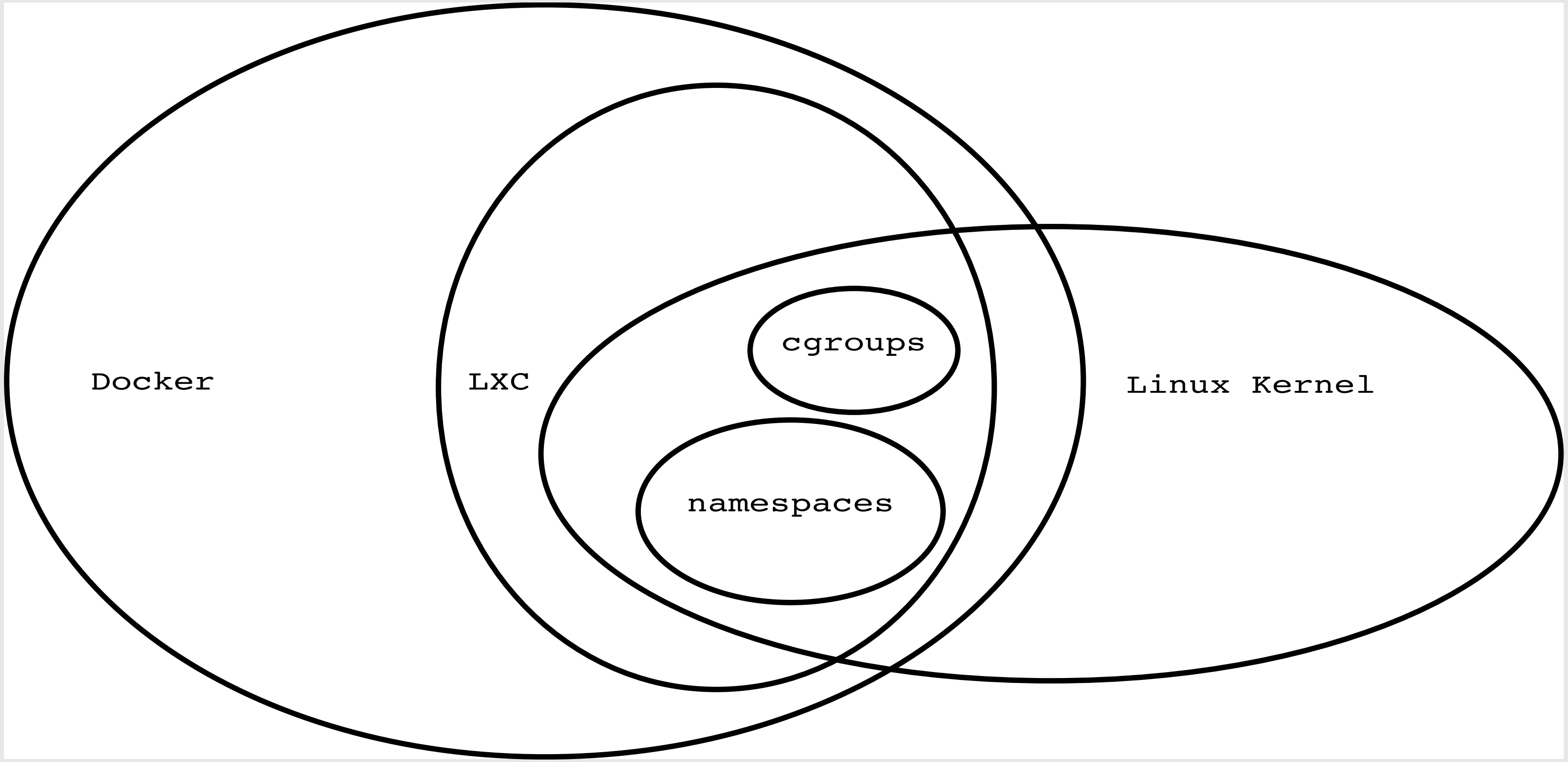
Security

Management

How does this work?

- cgroups
  - Group trees of processes
  - Allows control of resources for the group
- Namespaces
  - Isolate processes from host
  - Network, filesystem, pids, etc
- AuFS/DeviceMapper/Btrfs
  - CoW snapshoting of filesystem images

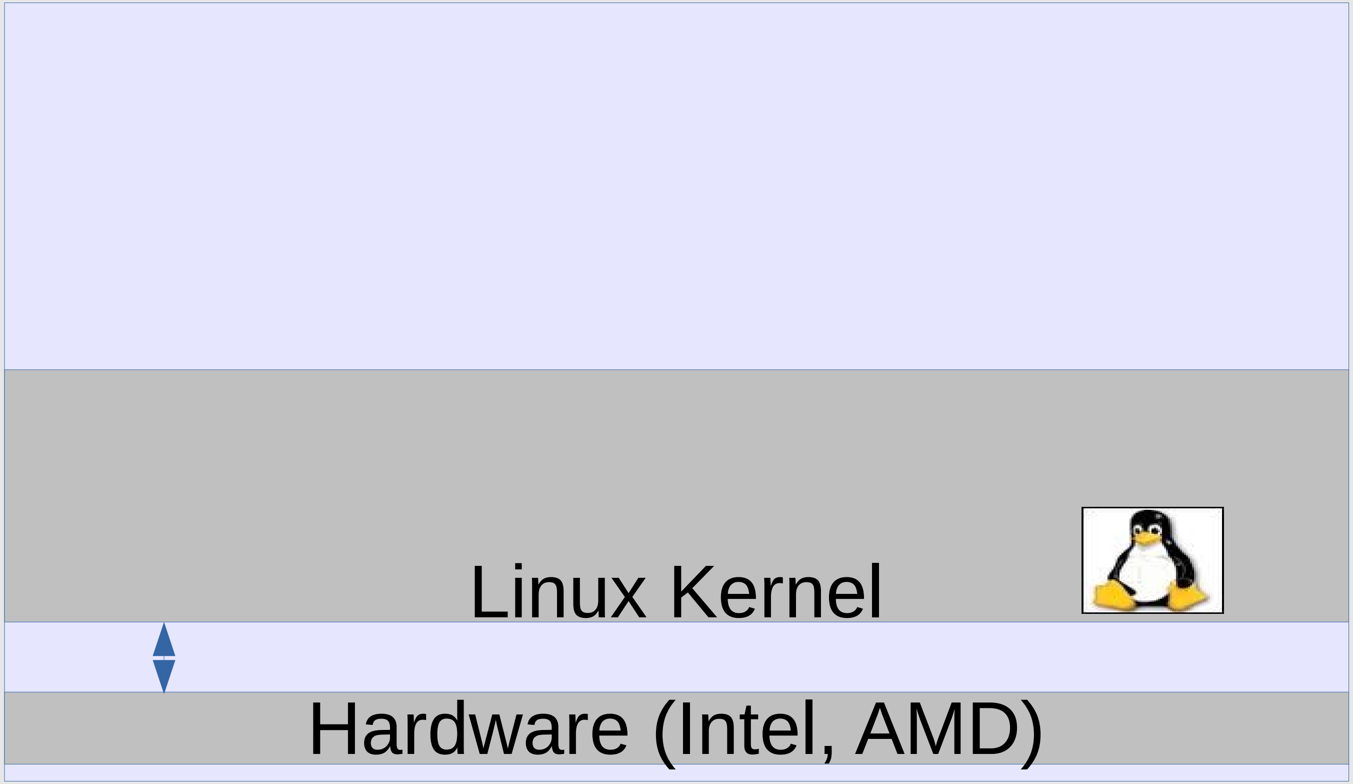
How does it work?



LXC (Linux Containers)

- Capability is already built into the kernel
- Utilizes cgroups (control groups) to limit, account and isolate resource usage (CPU, memory, disk I/O, etc.).
- **Stuff runs isolated from the rest of the host OS**

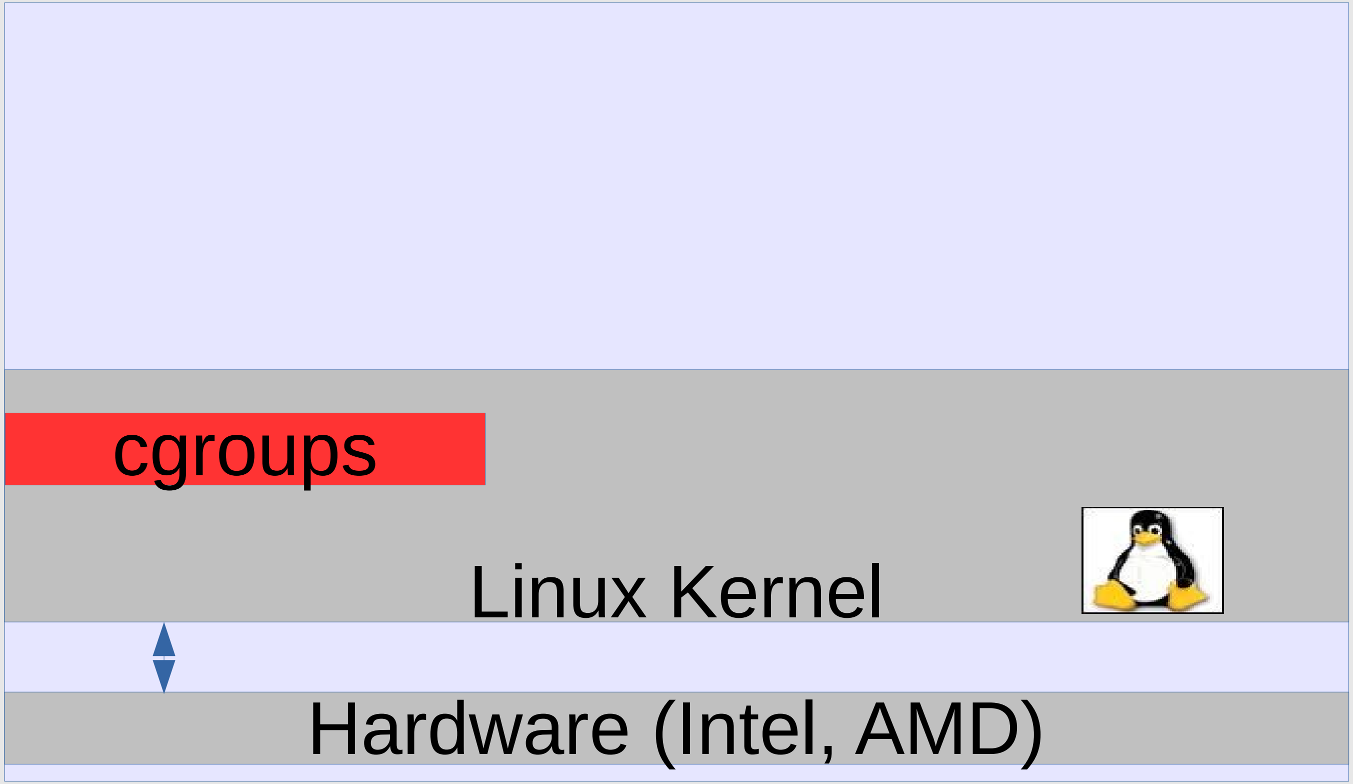
Linux Container Architecture



Linux Kernel

Hardware (Intel, AMD)

Linux Container Architecture

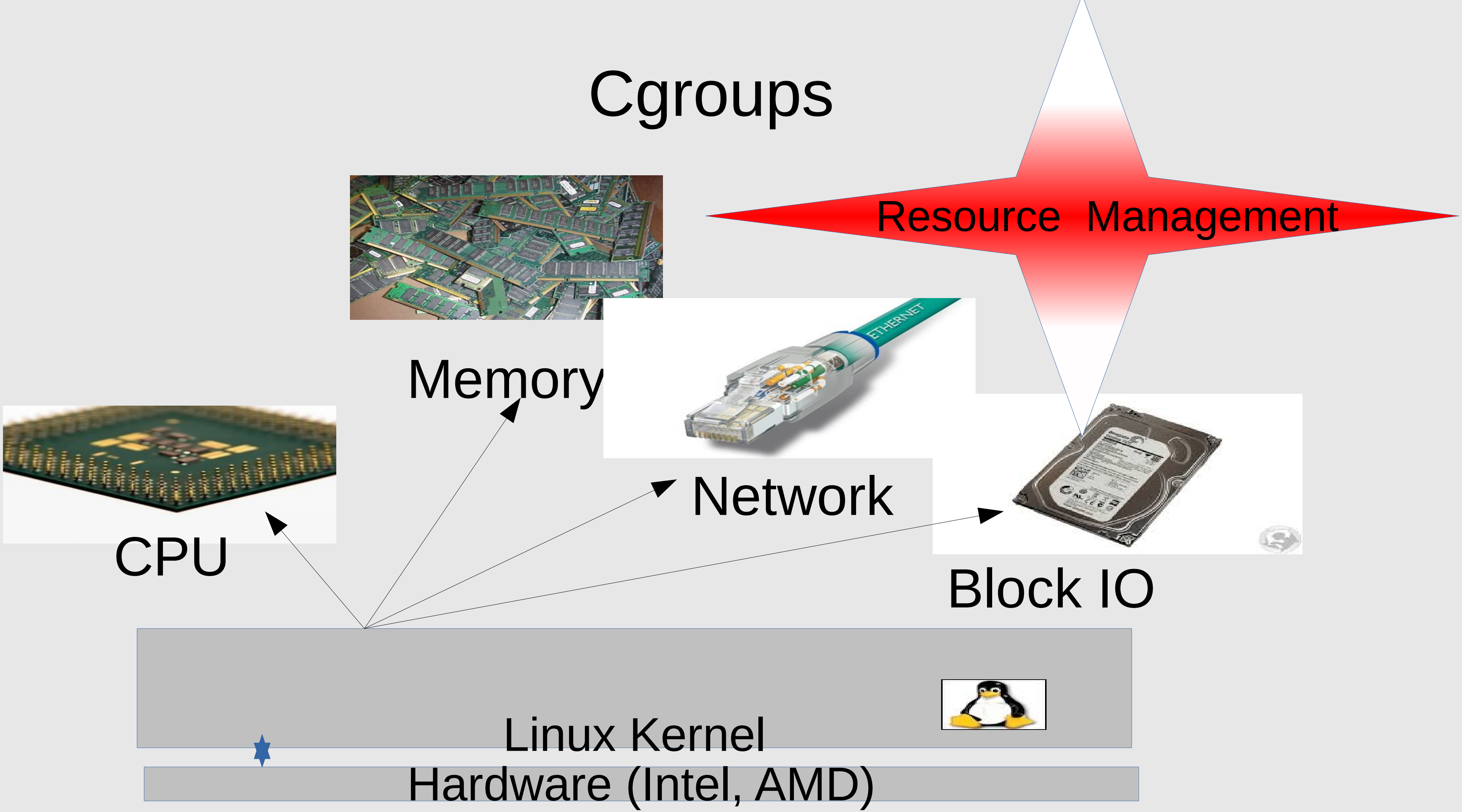


cgroups

Linux Kernel

Hardware (Intel, AMD)

Cgroups



Resource Management

Memory

CPU

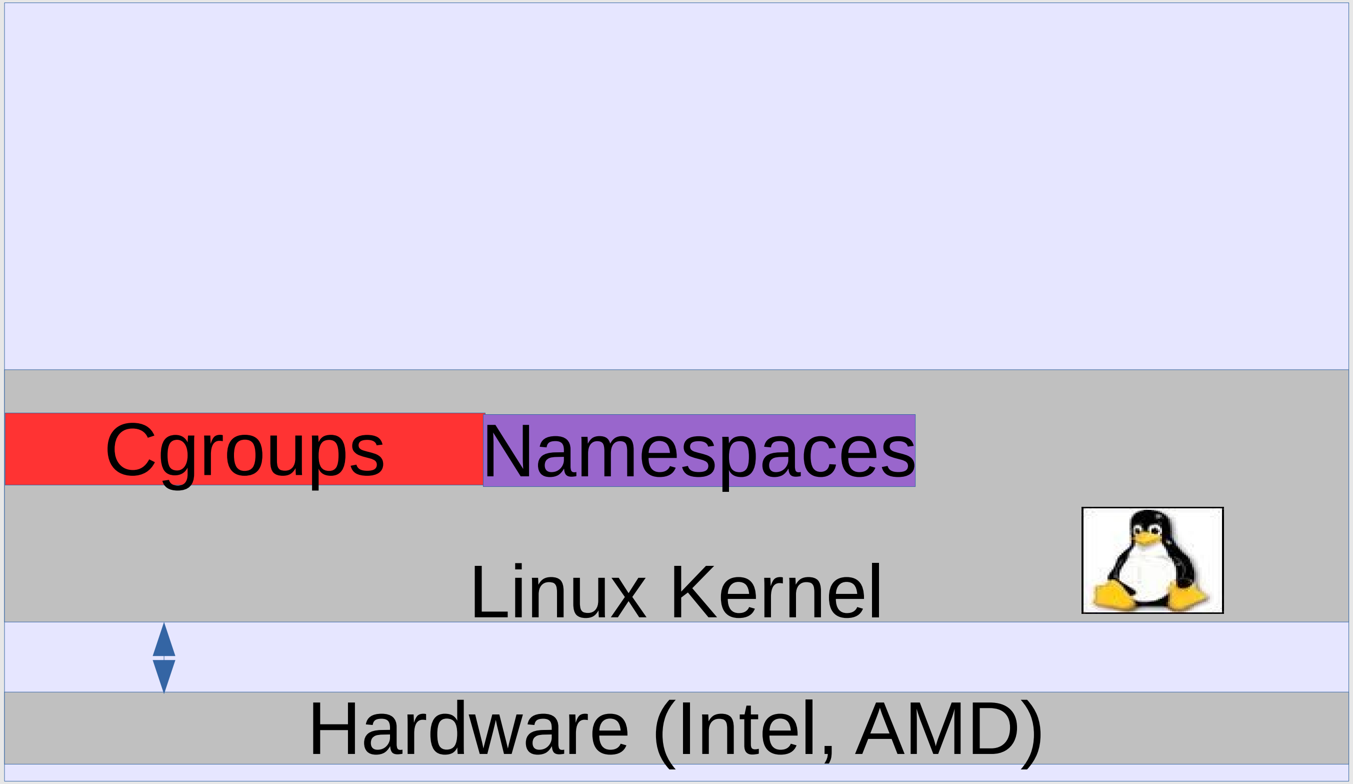
Network

Block IO

Linux Kernel

Hardware (Intel, AMD)

Linux Container Architecture



Cgroups

Namespaces

Linux Kernel

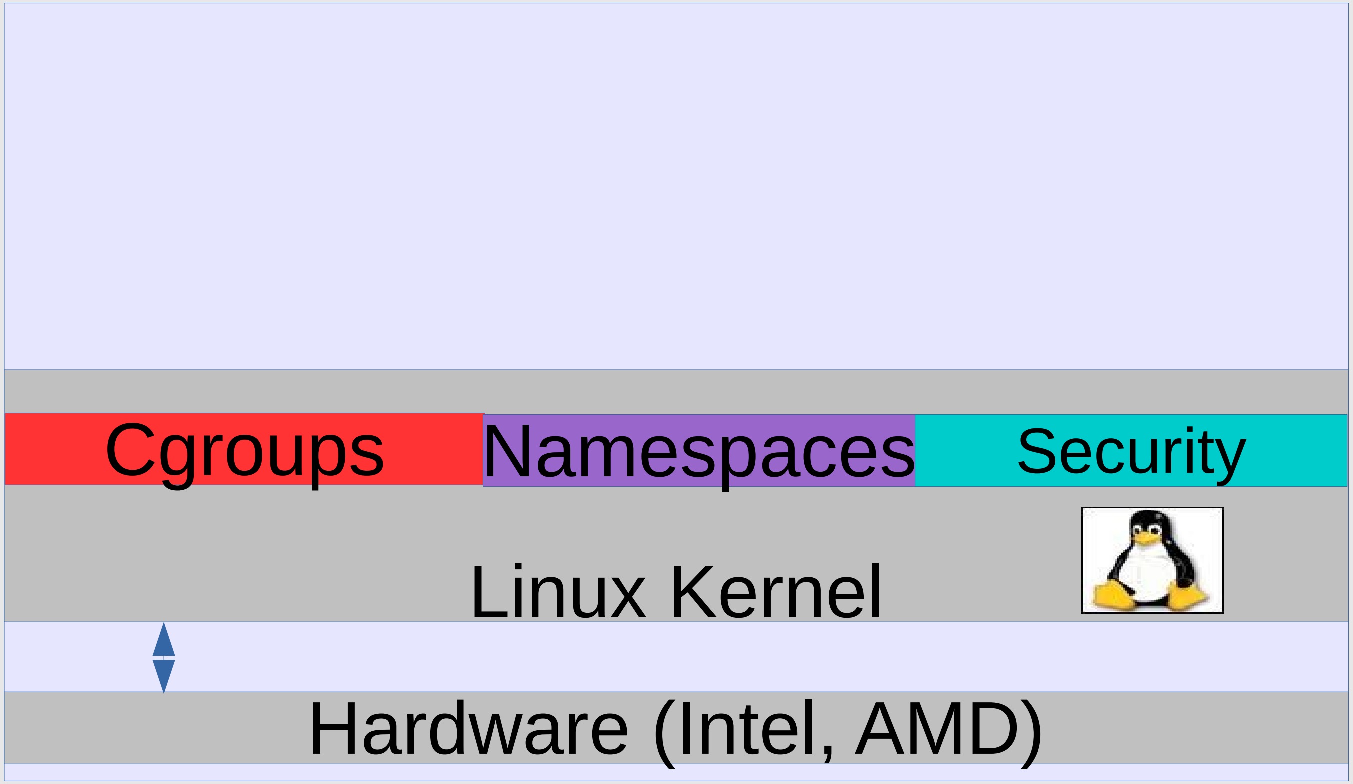
Hardware (Intel, AMD)

Namespaces

Process Isolation

- Isolate processes
  - Create a new environment with a
  - Subset of the resources
- Once set up, namespaces are transparent for processes
- Can be used in custom and complex scenarios
- Supported Namespaces
  - ipc, pid, mnt, net, uts
  - Future Red Hat Enterprise Linux 7: user

Linux Container Architecture



Cgroups

Namespaces

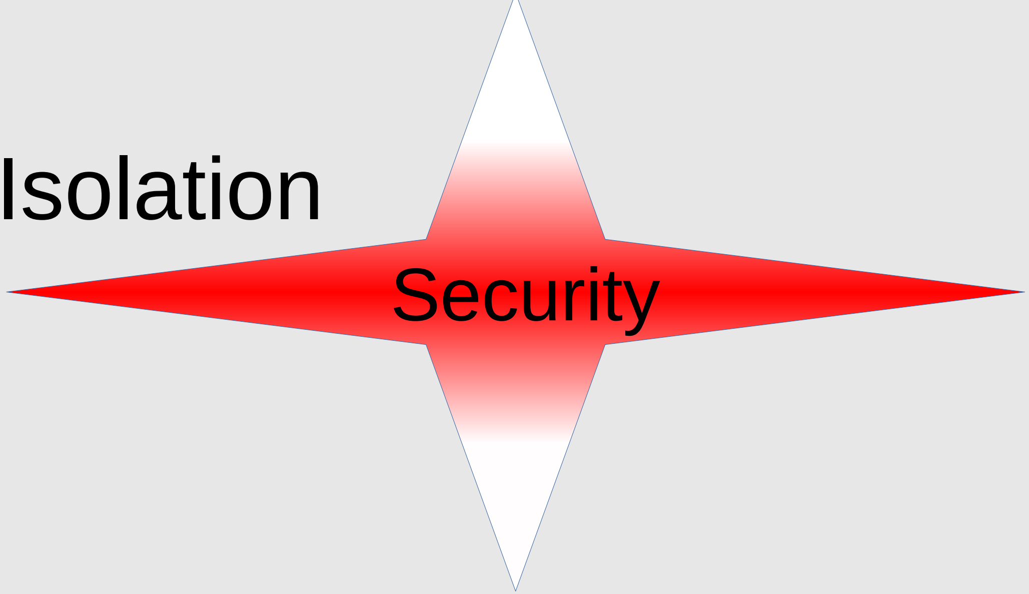
Security

Linux Kernel

Hardware (Intel, AMD)

Containers do NOT Contain!!!

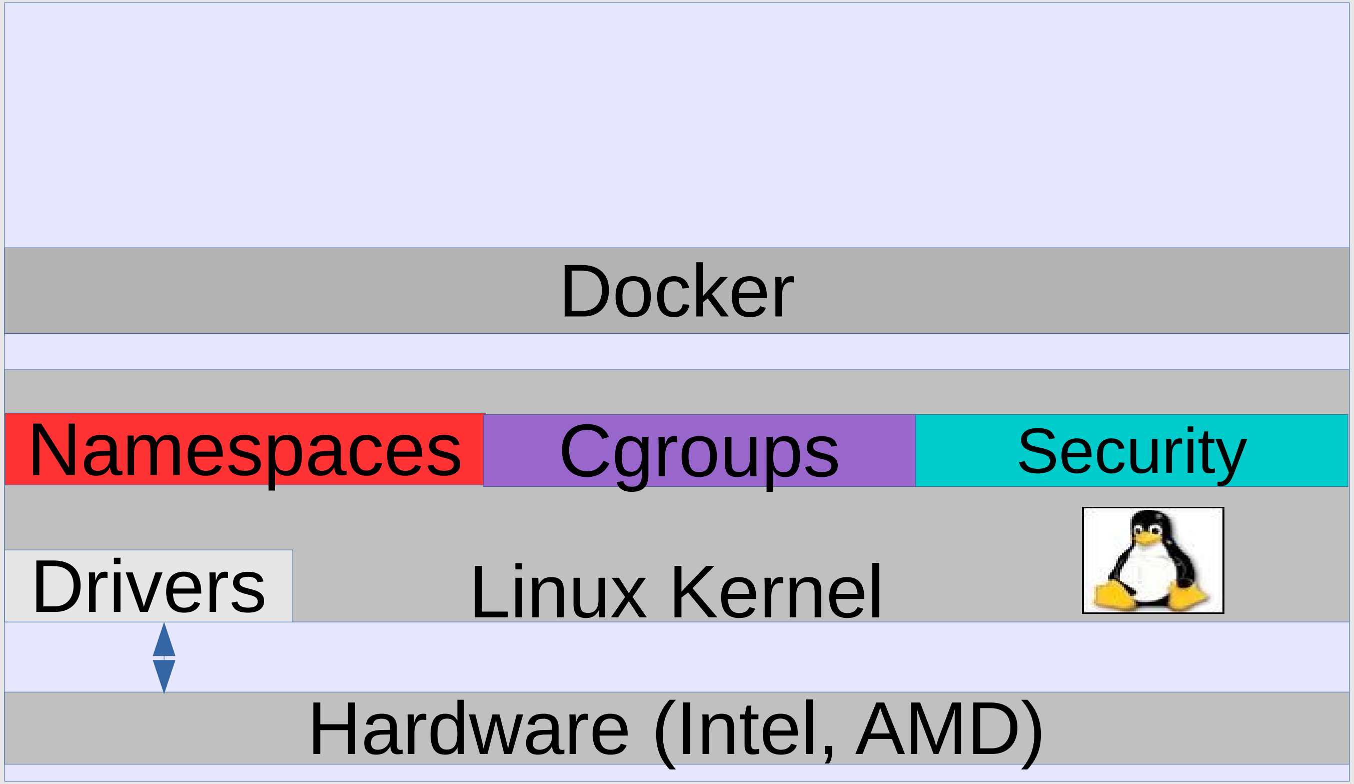
Security Isolation



Security

- Linux Containerization not complete
  - Not everything in Linux is namespaced
- SELinux sVirt
  - Container tooling uses sVirt
    - Type Enforcement
    - MCS Separation
- Capabilities
- Future User Namespaces

Linux Container Architecture



Docker

Namespaces

Cgroups

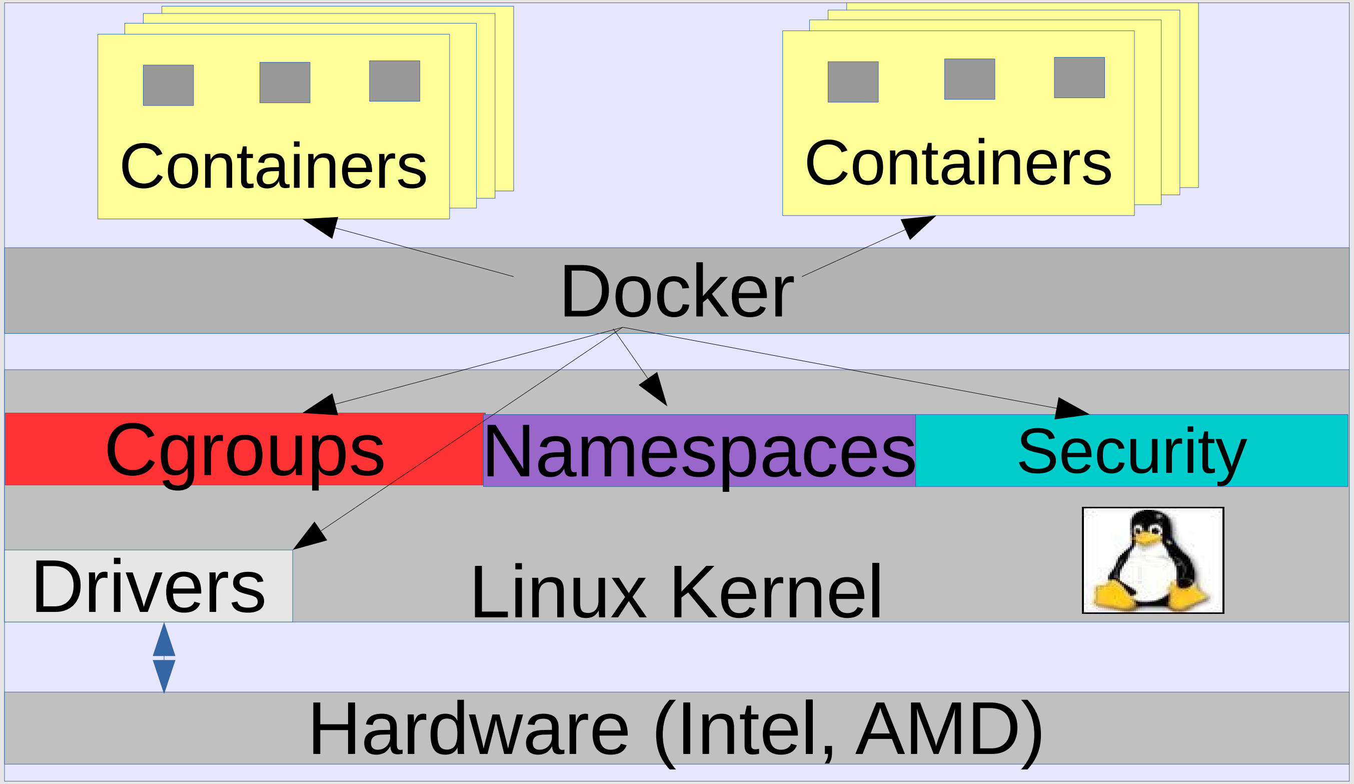
Security

Drivers

Linux Kernel

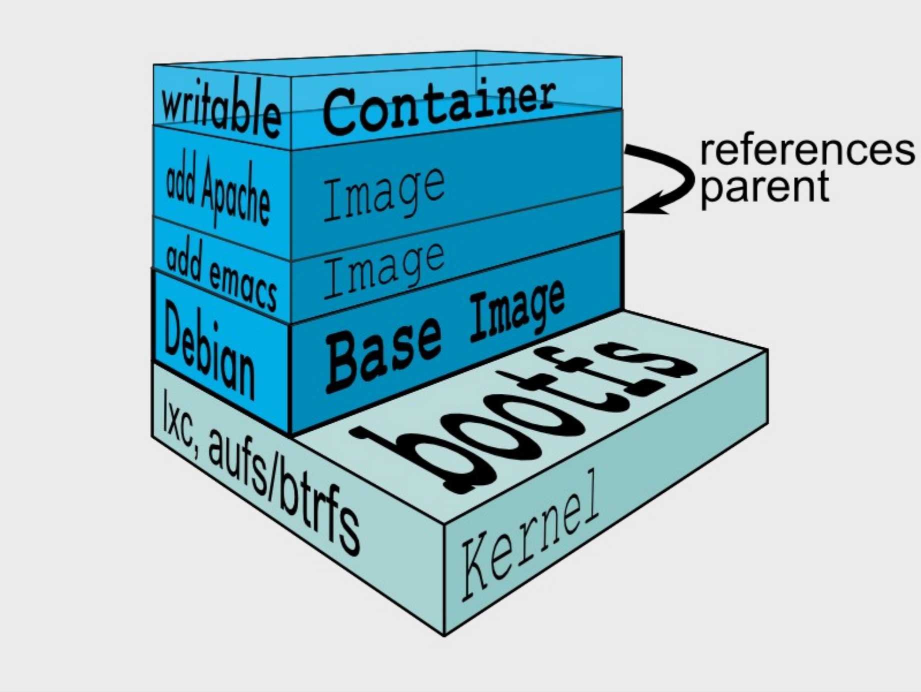
Hardware (Intel, AMD)

Linux Container Architecture



AUFS (AnotherUnionFS)

- Union file systems
  - Allows several file-systems to be mounted at one time, appearing to be one file-system.
- Docker uses it to create a layered file system.



Pretty picture from http://docs.docker.io/en/latest/terms/layer/

Docker Networking

- Docker uses a standard network bridge called docker0
- docker0 gets assigned an unused IP range
- Containers get bonded to docker0
- docker0's IP is the gateway for containers

Docker Daemon

- Software layer that allow for easy creation and management of Docker containers (glorified LXC instances).

Example Docker Work Flow (CentOS + SSH)

```
docker pull centos

docker run -t centos yum -y install openssh-server

docker commit <image id> centos/ssh

docker run -d -p 2222:22 -t centos/22 /usr/sbin/sshd
```

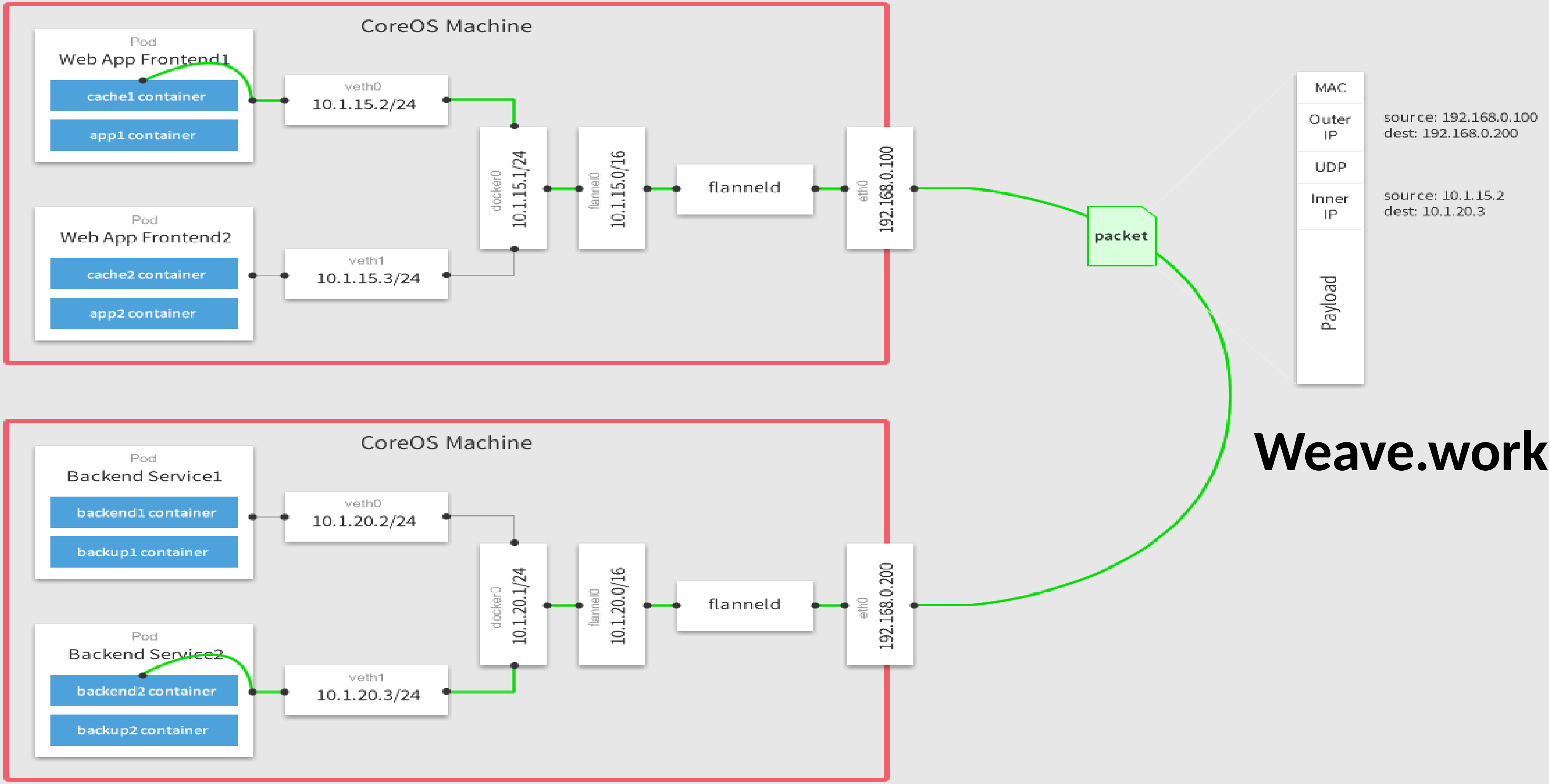
Host/Container Communication

- Ports can be exposed and mapped to the host with “docker -p”
  - `docker run -p 80:80 -t <image> <cmd>`

Container to Container Communication

- Handled with container linking
  - `docker run -d -p 6666 --name parent <image>`
  - `docker run --link parent:child -i -t  <image>`
- Linked container will have environment variables like:
  - `CHILD_PORT_6666_TCP_ADDR=172.17.0.2`
  - `CHILD_PORT=tcp://172.17.0.2`

Multi-Host networking



Weave.works

Docker Registry

- Software that runs the docker index at https://index.docker.io/
- Base images for things like CentOS, Ubuntu, Fedora, etc are pulled from the public registry.
- Local registries can be created and submitted to.

Docker Registry = App store

```
$ docker push barais/application

$ docker pull barais/application

$ docker run -d barais/application
```

Docker Files

- Allow for easy recreation of an application anywhere Docker can be ran. Example:

```
# use the ubuntu base image provided by dotCloud
FROM ubuntu

# make sure the package repository is up to date
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update

# install memcached
RUN apt-get install -y memcached
```

Dockerfile

FROM dockerfile/node   Base Image

RUN apt-get update –qq Instructions

RUN mkdir /my/app while building image

ADD . /my/app

CMD [“node”,”web”] What Command to run

Dockerfile

```
//Build an Image
$> docker build –t “rohitghatol/node” .
//Run an Image
$>docker run –d –p 80:3000 rohitghatol/node
//Push to Docker Hub
$>docker push rohitghatol/node          //developer
$>docker pull rohitghatol/node          //operations
```

Uses for Docker

- Testing on multiple versions of multiple distros without the pitfalls of standard virtualiztion.
- Running newer or older versions of applications not in your host OSes repos.
- Creating an applications that can be easily rebuilt and reused on any distro, anywhere.

More Topics

- Working with the index (pull/push/login/search)
- Using a private registry
- Using volumes for data
- Naming containers
- Using links between containers
- Creating base images

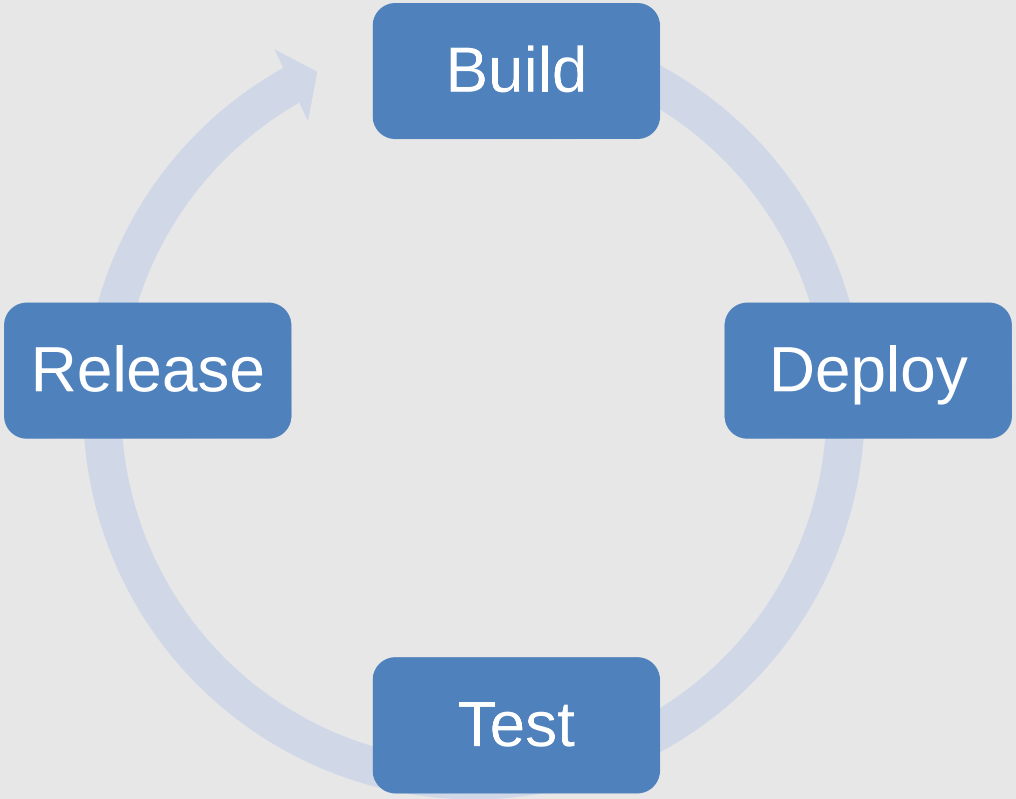
Summary

Advantages:

- Portable configuration
- Reuse images other people have built
- Lightweight, fast
- Easy to scale up
- ”Build once run everywhere”

DOCKER
AND
CONTINUOUS DELIVERY

Continuous Delivery



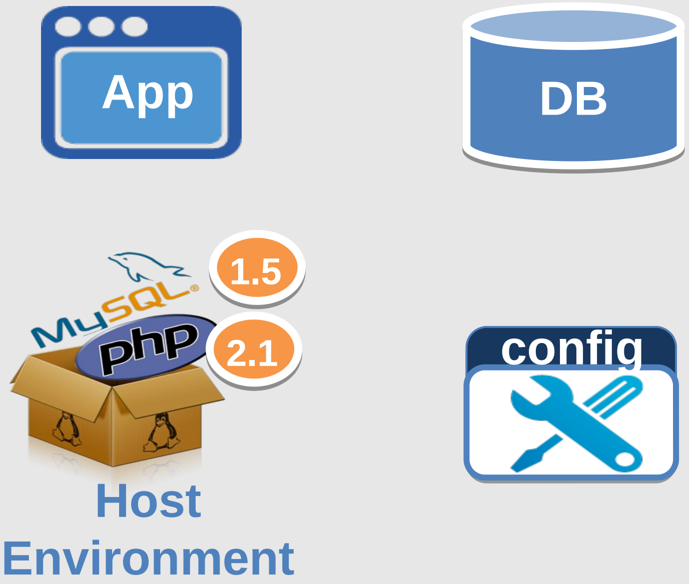
Build

Deploy

Test

Release

The Components



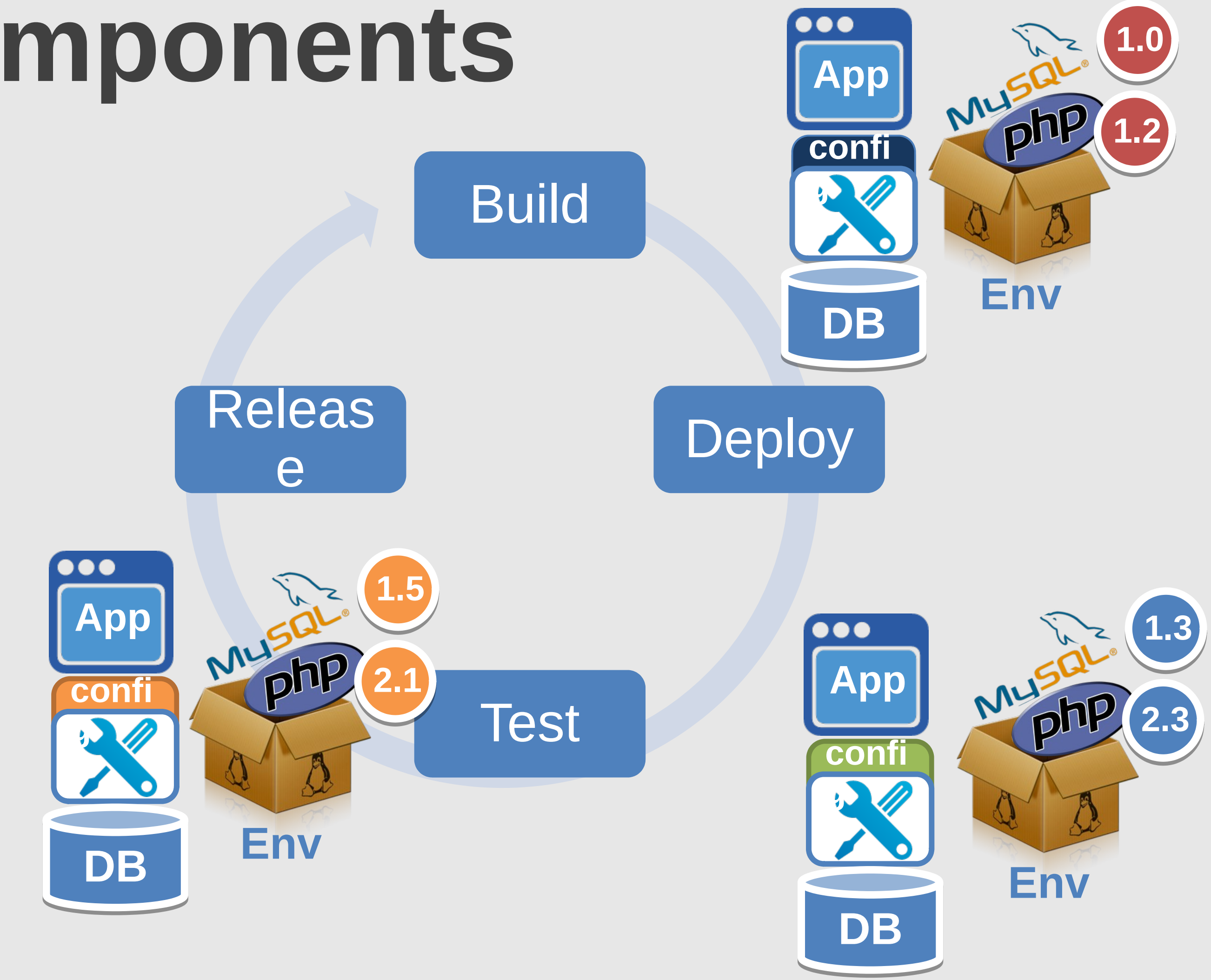
App

DB

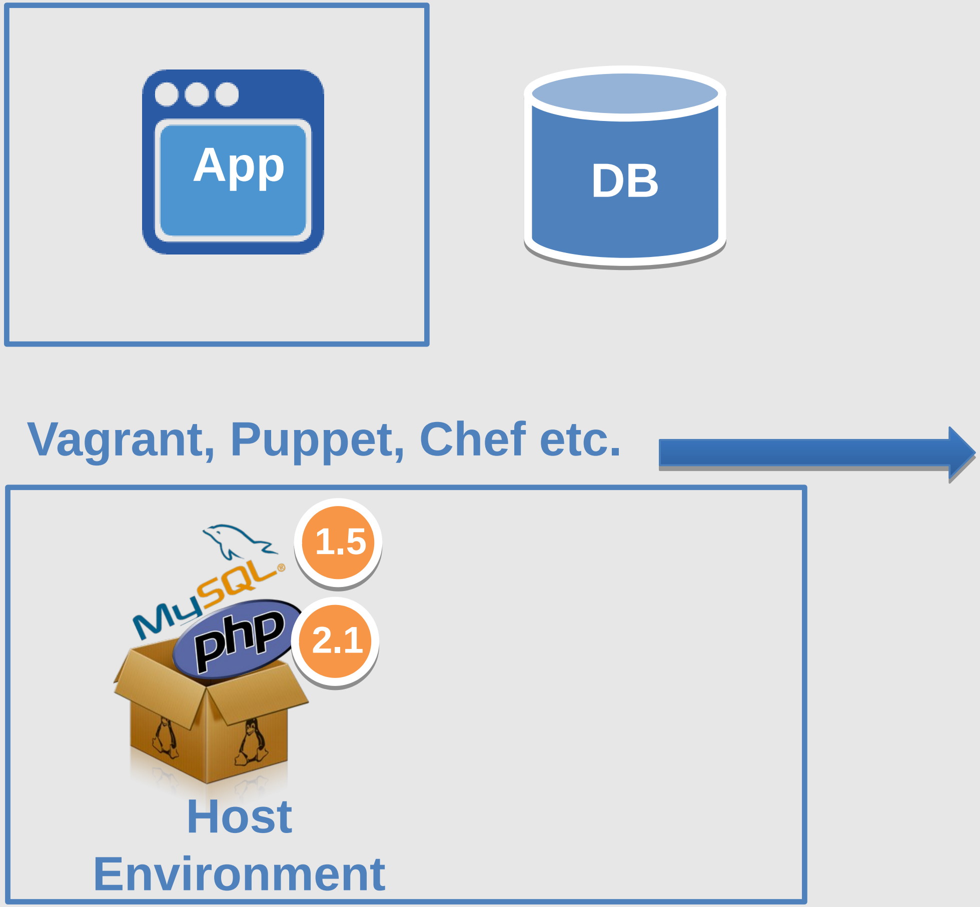
Host Environment

config

The Components



The Components



Jenkins, Bamboo, etc

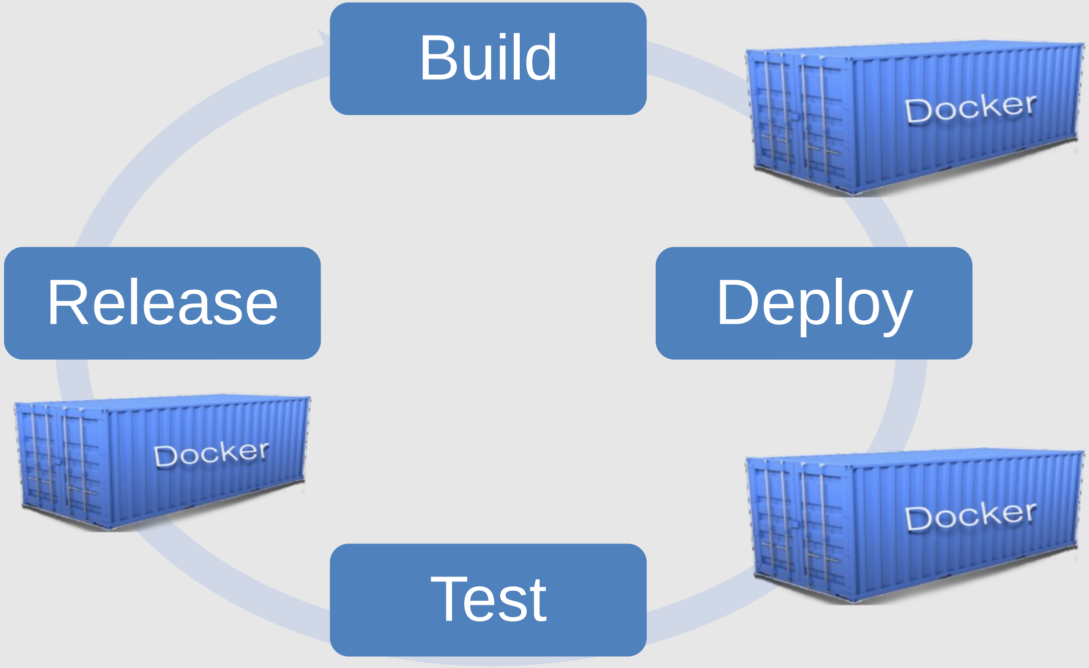
Vagrant, Puppet, Chef etc.

Host Environment

Virtual Machines, Instructions, Commands, Etc.

CONTINUOUS DELIVERY THE NEXT STEP…

Containers

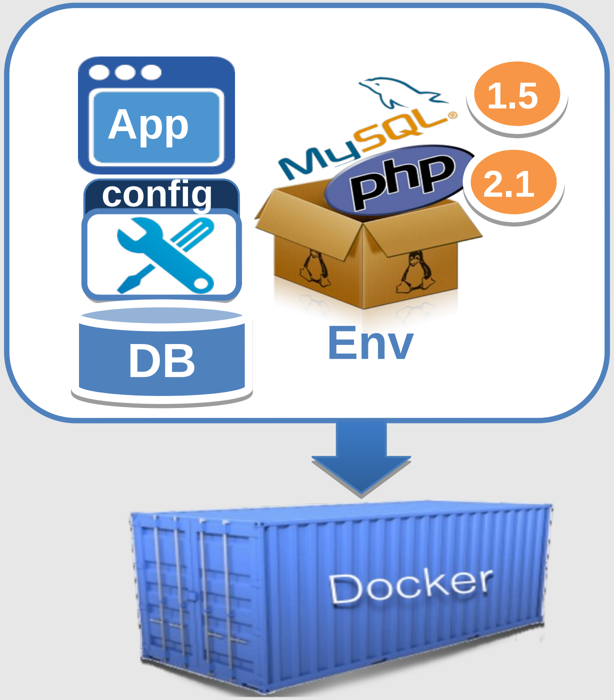


Build

Deploy

Test

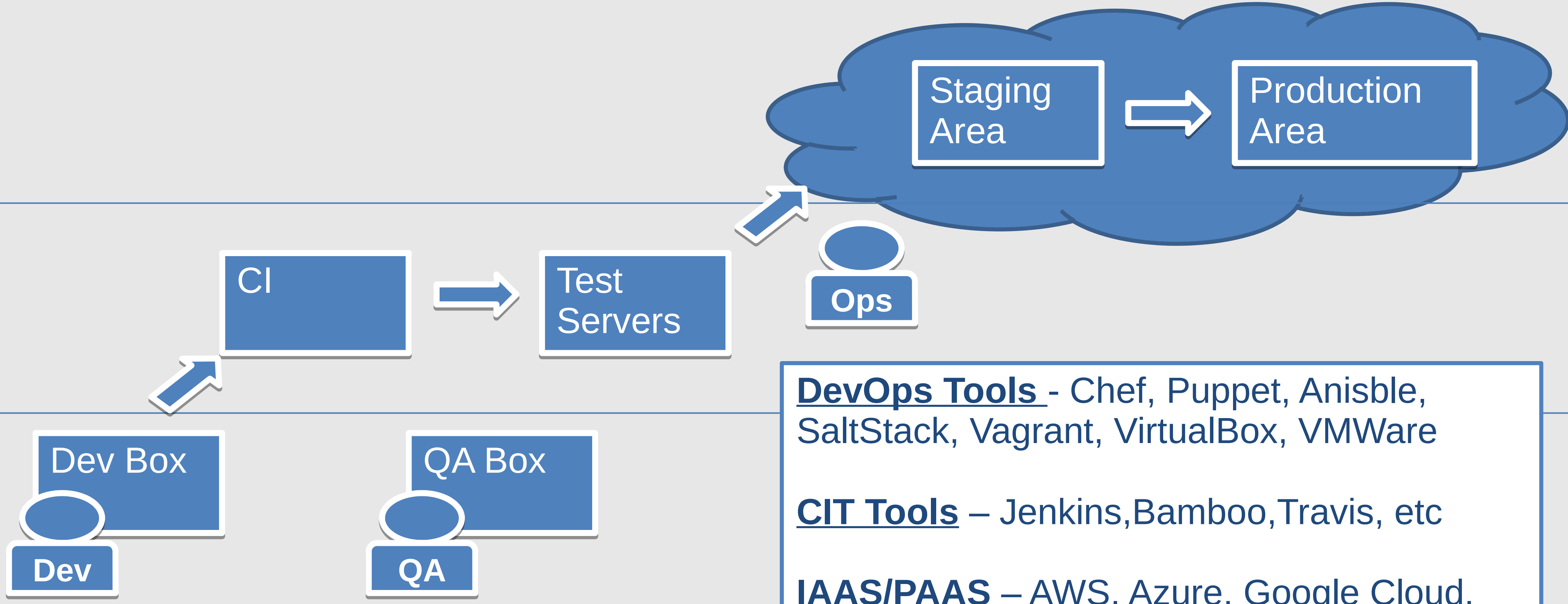
Release



LANDSCAPE

How companies are deploying SAAS today?

Landscape



DevOps Tools - Chef, Puppet, Anisble, SaltStack, Vagrant, VirtualBox, VMWare

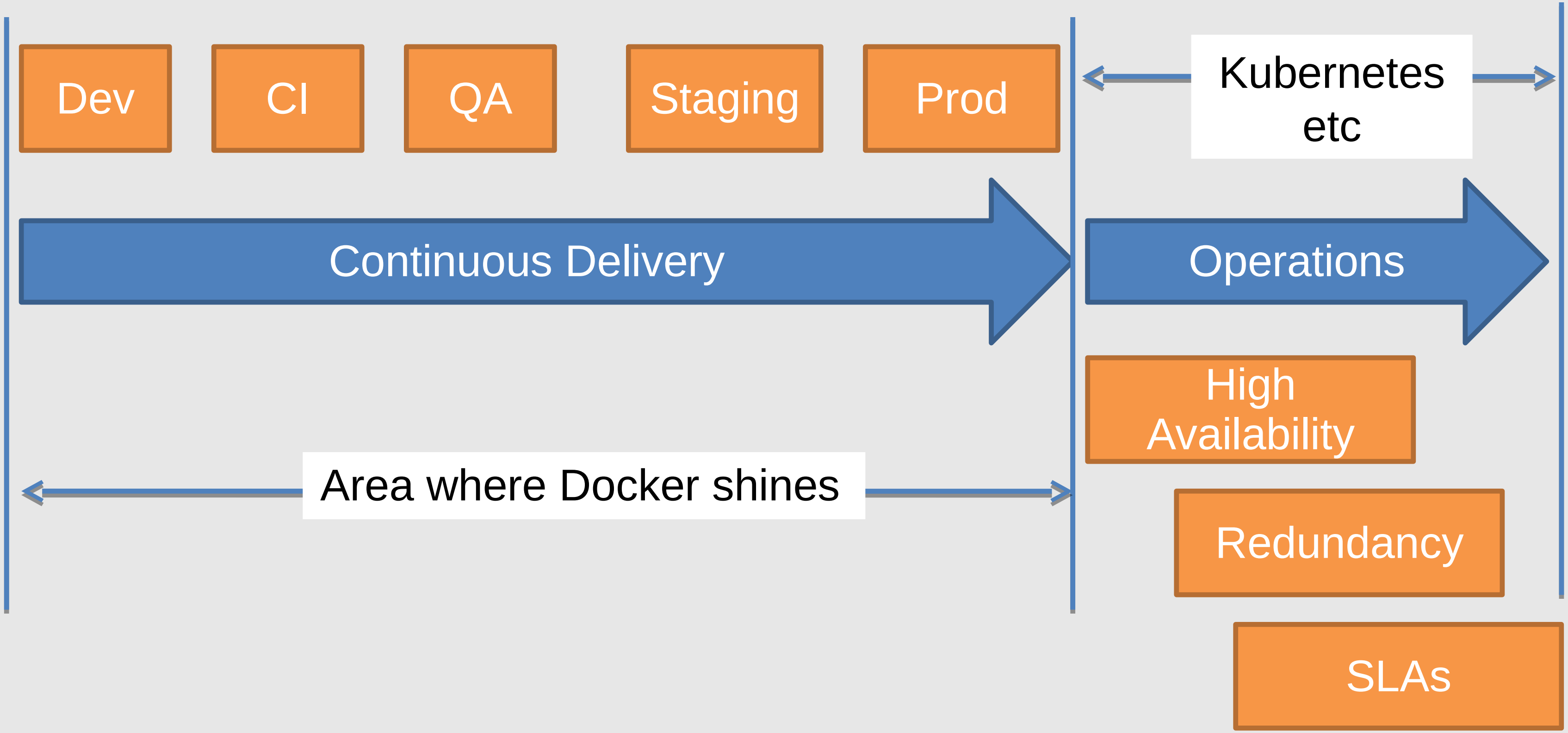
CIT Tools – Jenkins,Bamboo,Travis, etc

IAAS/PAAS – AWS, Azure, Google Cloud, Digital Ocean,Heroku etc

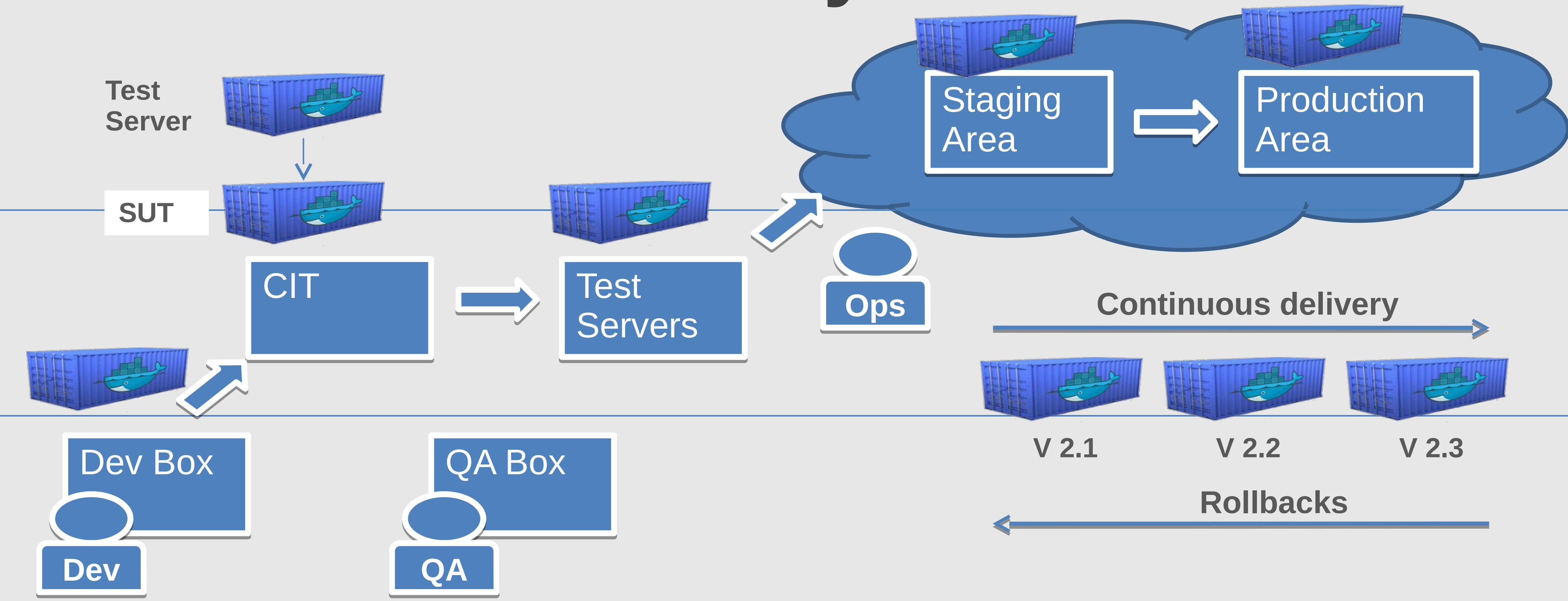
DOCKER USE CASES

CONTINUOUS DELIVERY

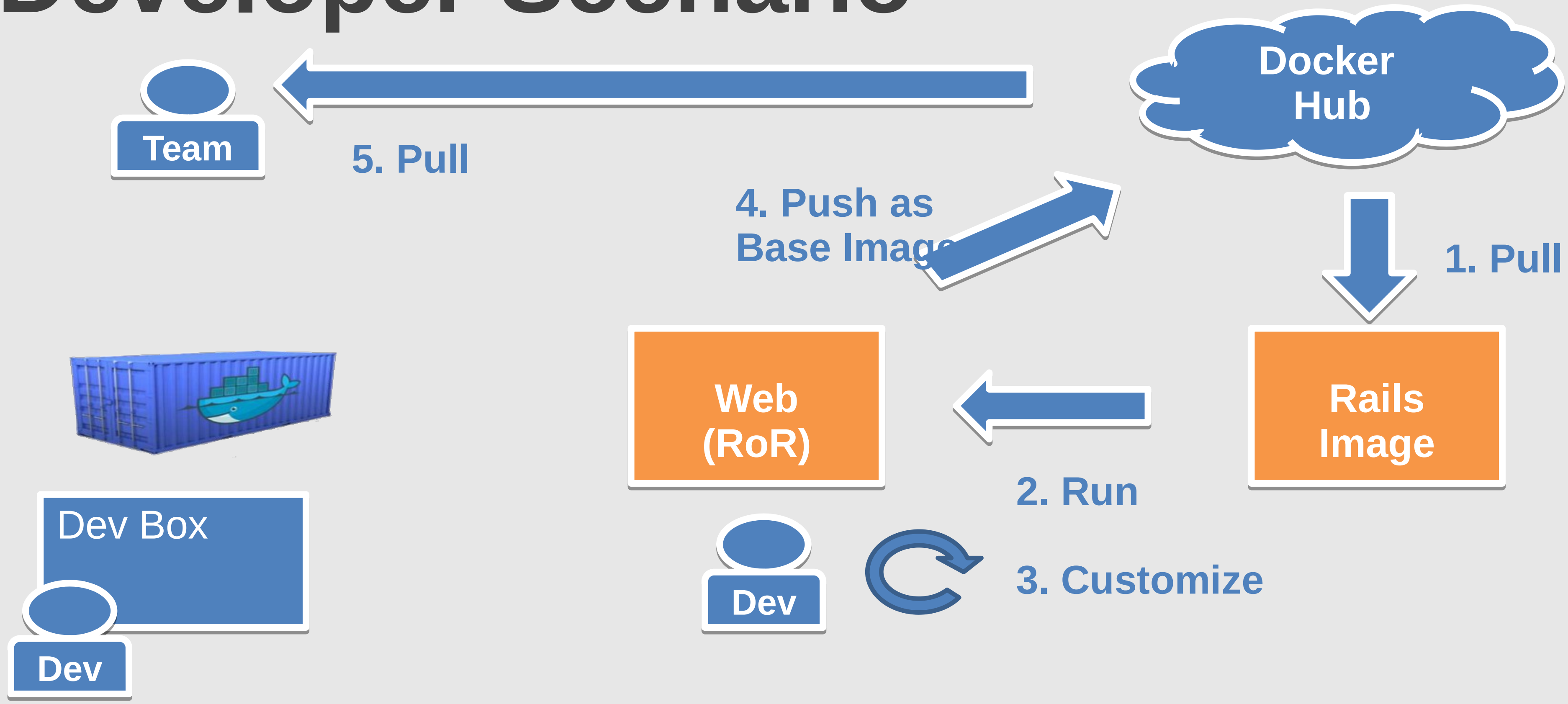
CONTINUOUS DELIVERY



Continuous Delivery Use case



Developer Scenario



Docker Hub

1. Pull

Rails Image

2. Run

Web (RoR)

Dev

3. Customize

4. Push as Base Image

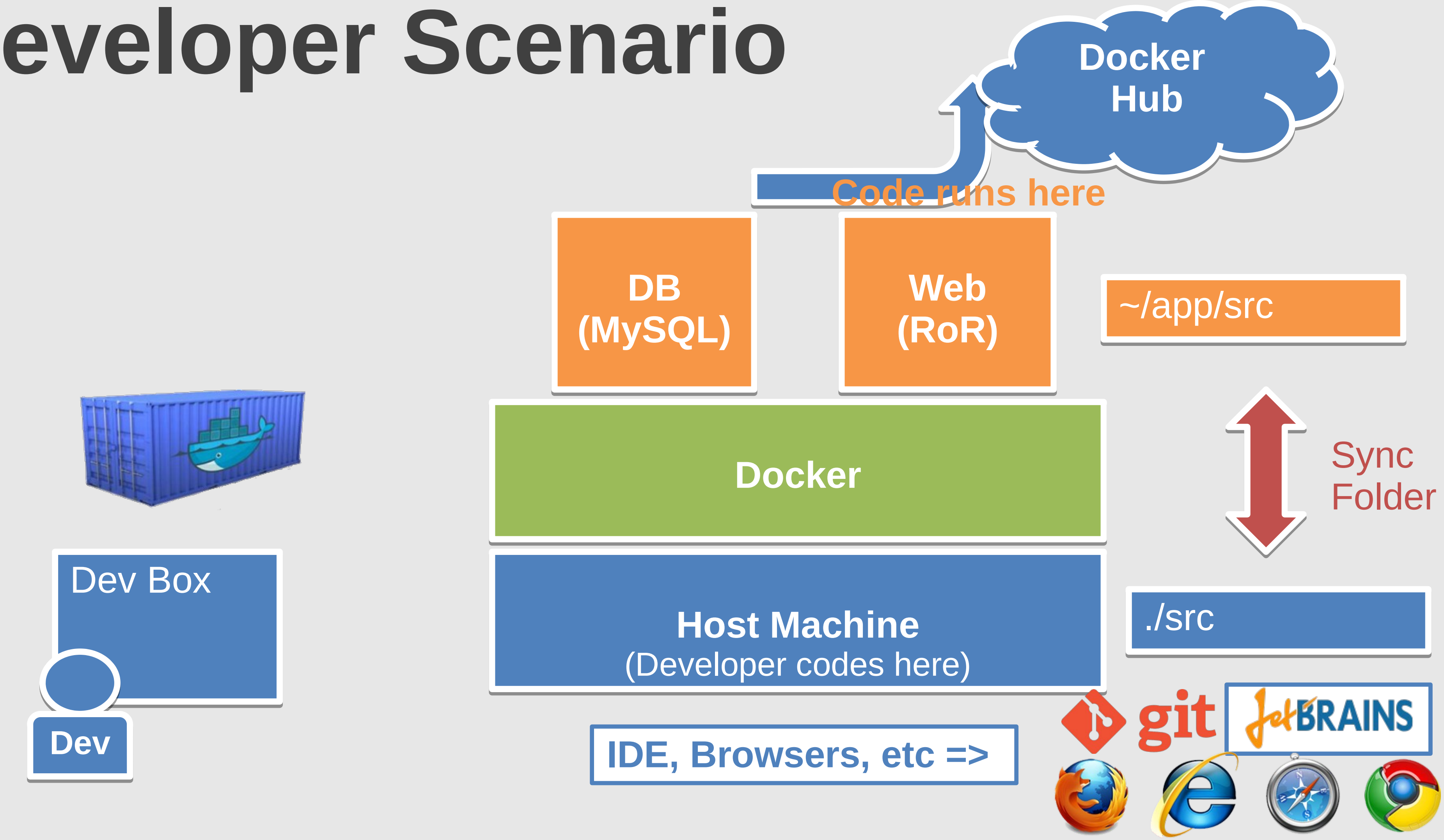
5. Pull

Team

Dev Box

Dev

Developer Scenario



Docker Hub

Code runs here

DB (MySQL)

Web (RoR)

~/app/src

Docker

Sync Folder

Host Machine
(Developer codes here)

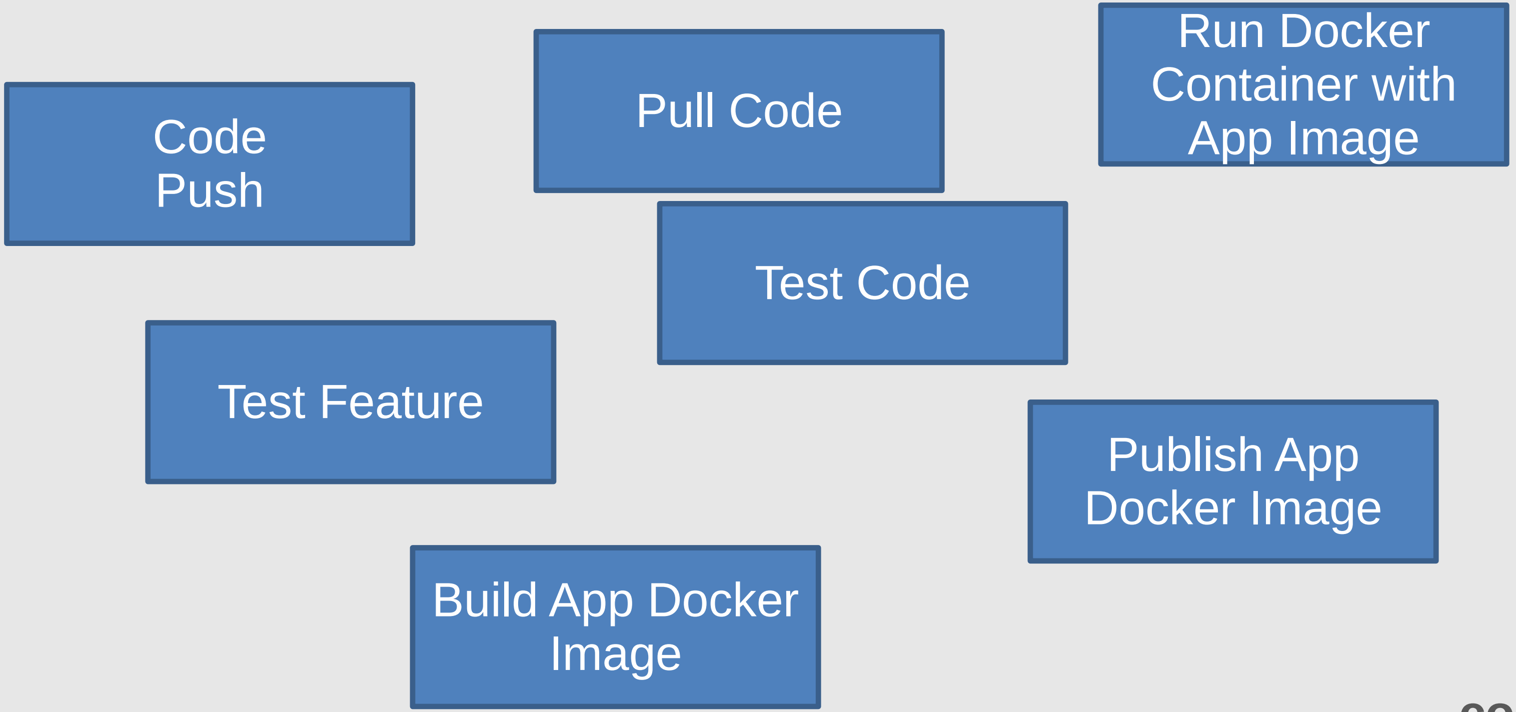
./src

IDE, Browsers, etc =>

Dev Box

Dev

CI Scenarios



Code Push

Pull Code

Run Docker Container with App Image

Test Code

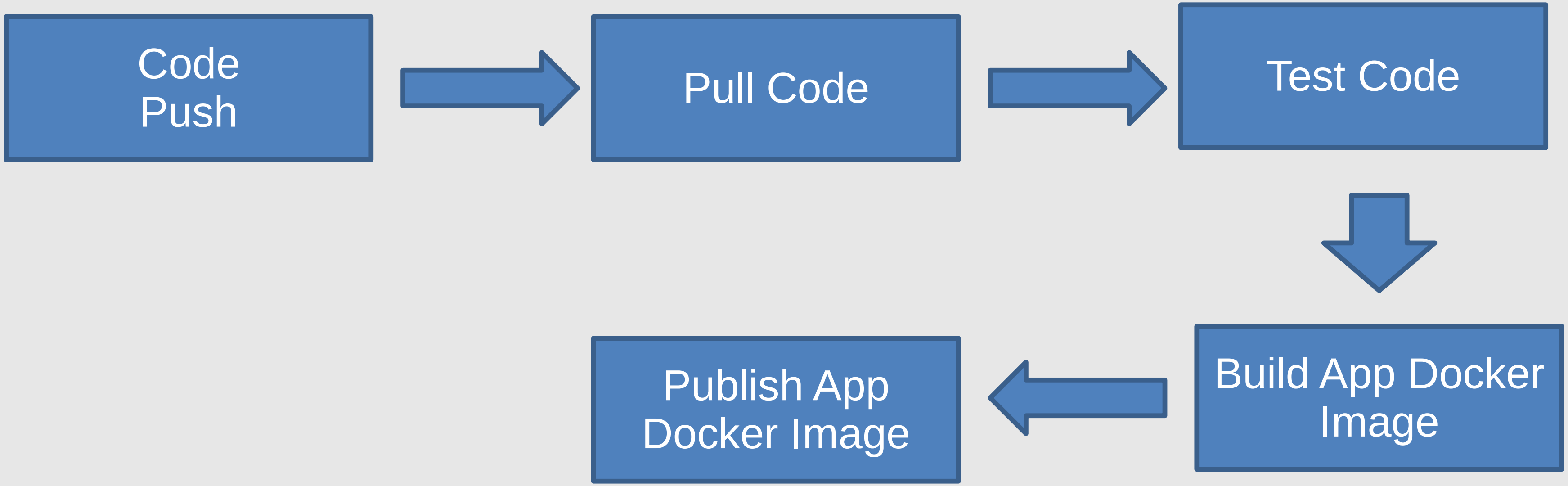
Test Feature

Publish App Docker Image

Build App Docker Image

Numerous combinations…

CI Scenarios – Option 1



Code Push

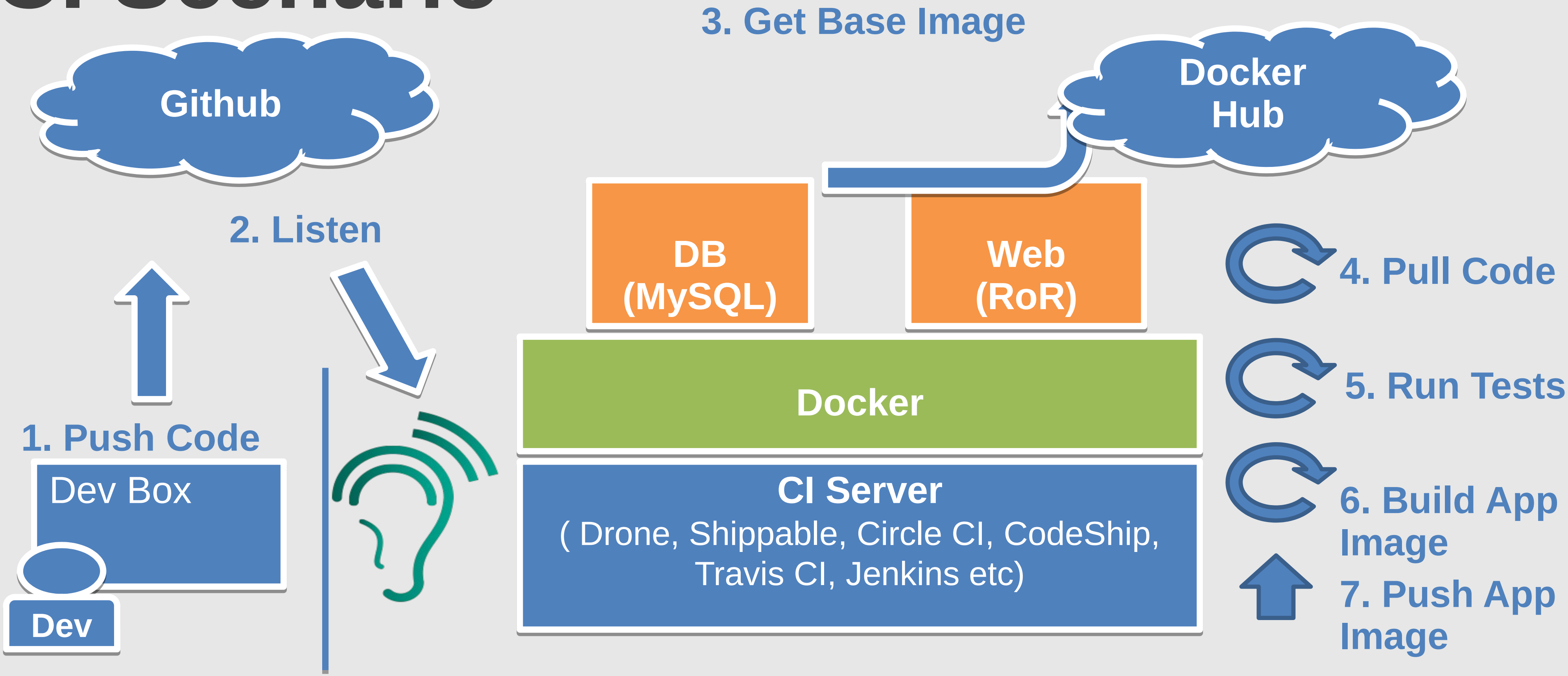
Pull Code

Test Code

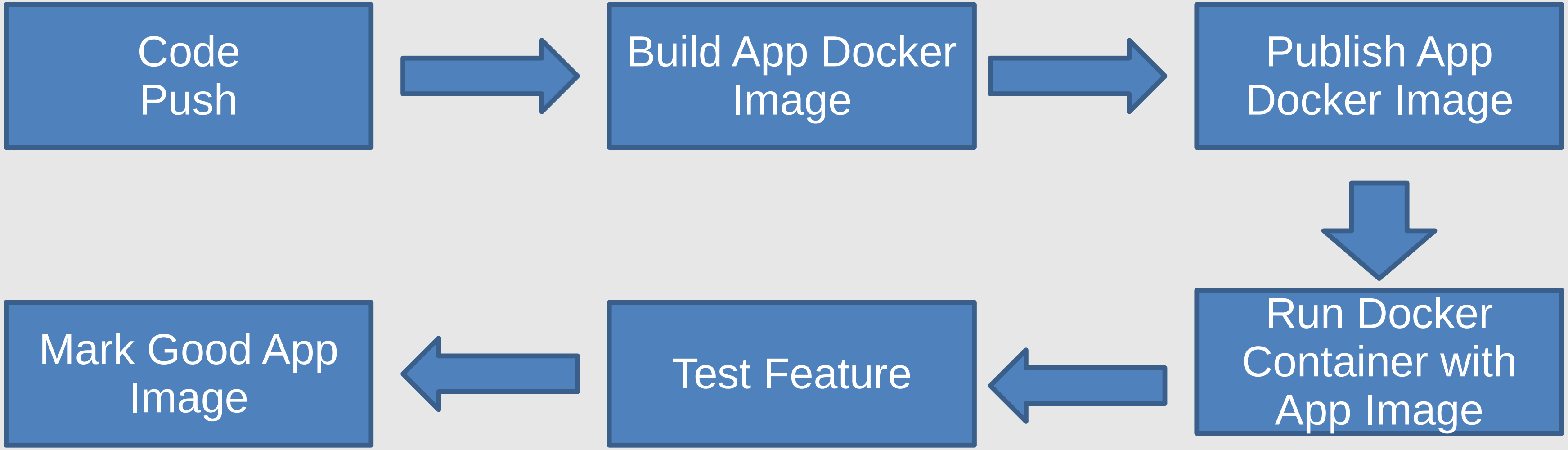
Build App Docker Image

Publish App Docker Image

CI Scenario


3. Get Base Image
Github
Docker Hub
2. Listen
DB (MySQL)
Web (RoR)
4. Pull Code
5. Run Tests
Docker
1. Push Code
Dev Box
CI Server
(Drone, Shippable, Circle CI, CodeShip, Travis CI, Jenkins etc)
6. Build App Image
7. Push App Image
Dev

CI Scenarios – Option 2



Code Push

Build App Docker Image

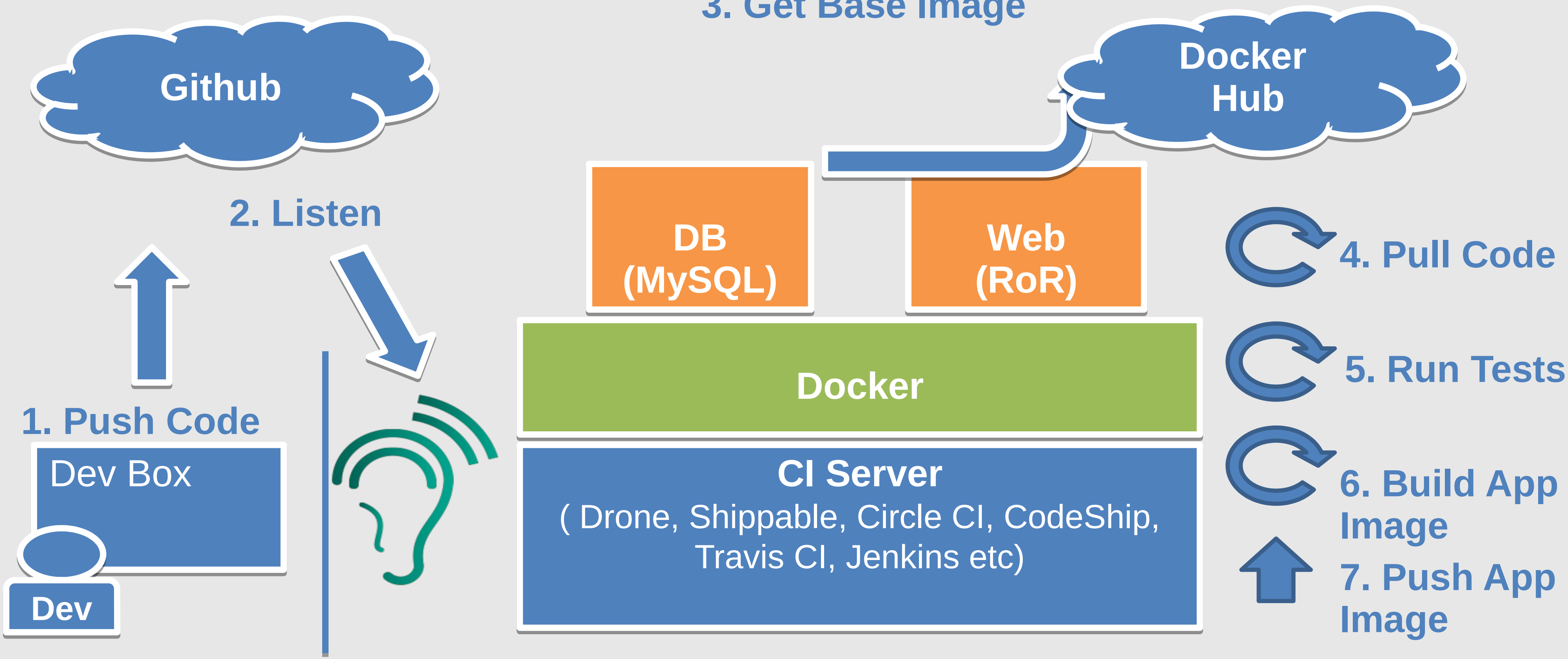
Publish App Docker Image

Run Docker Container with App Image

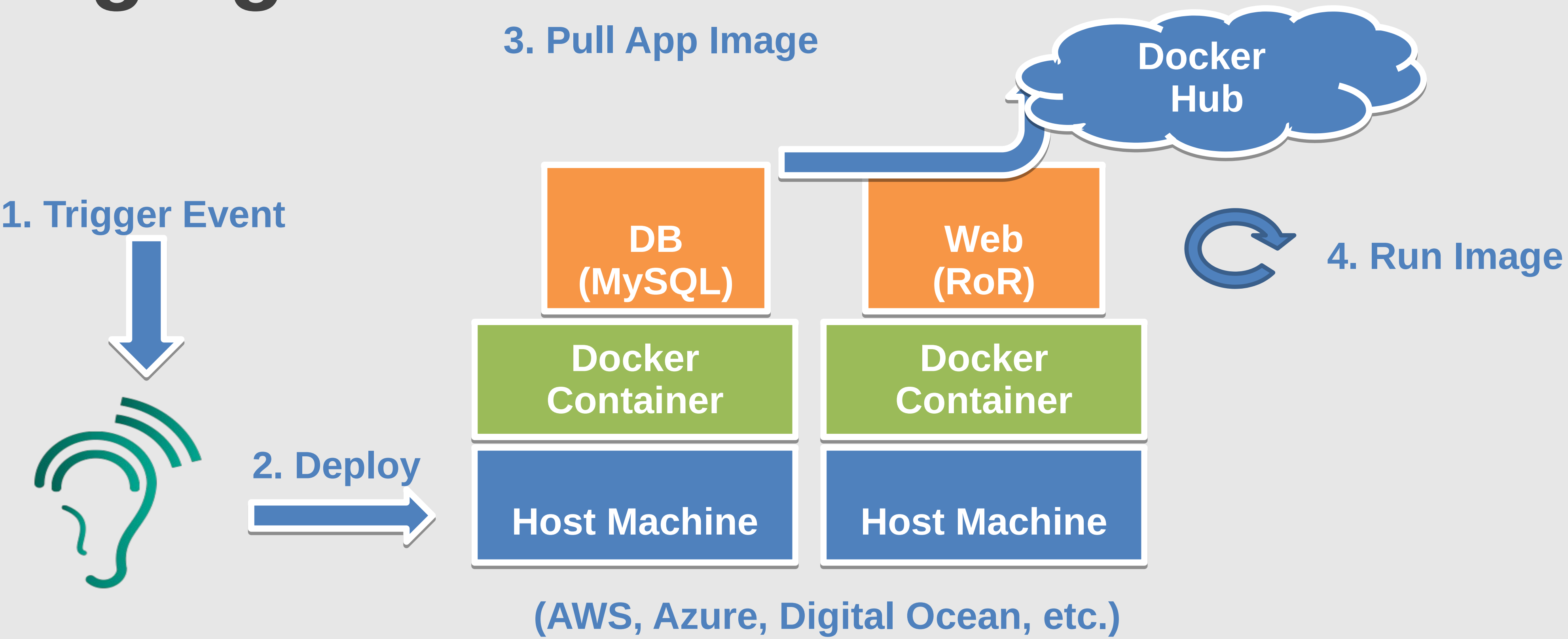
Test Feature

Mark Good App Image

CI Scenario

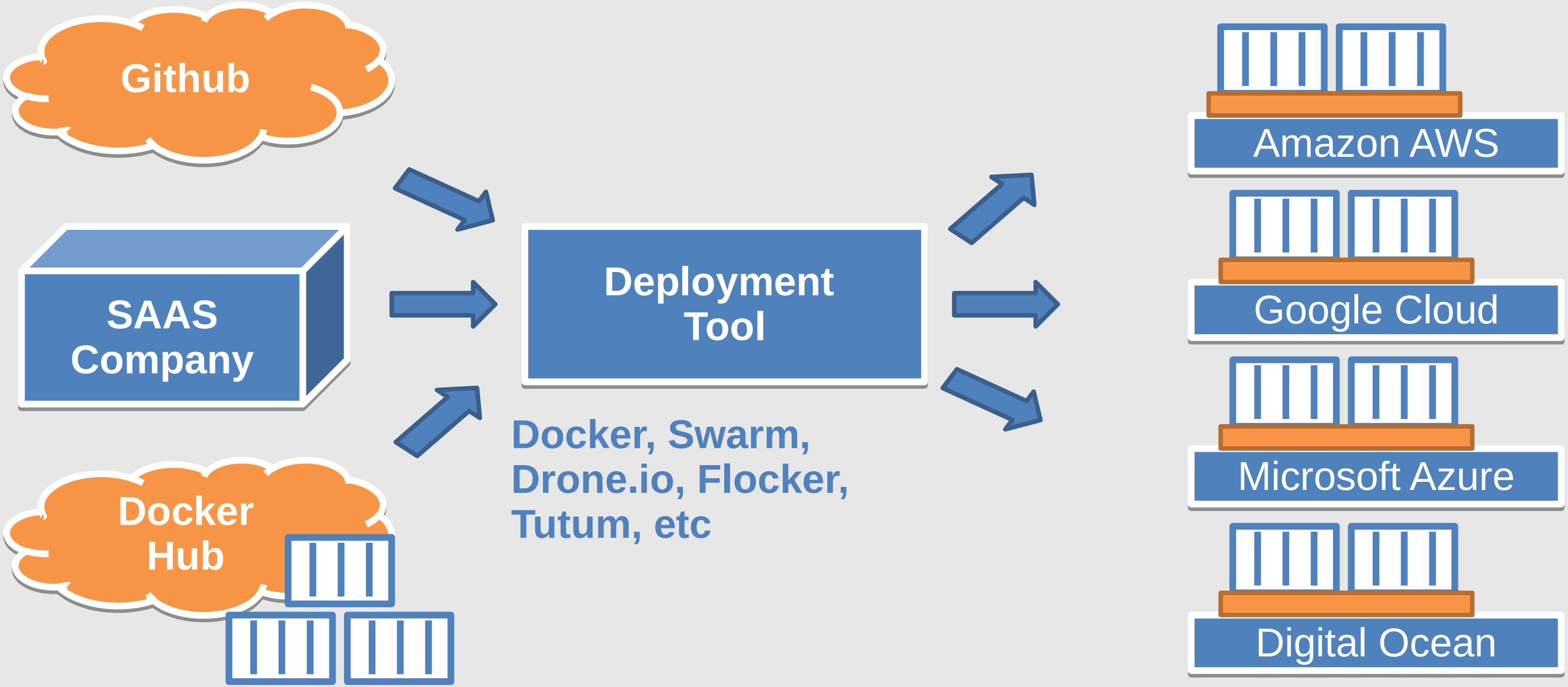

3. Get Base Image
Github
Docker Hub
2. Listen
DB (MySQL)
Web (RoR)
4. Pull Code
Docker
5. Run Tests
1. Push Code
Dev Box
CI Server
(Drone, Shippable, Circle CI, CodeShip, Travis CI, Jenkins etc)
6. Build App Image
7. Push App Image
Dev

Staging/Prod Scenario



CLOUD PORTABILITY

Cloud Portability Use case



Docker, Swarm, Drone.io, Flocker, Tutum, etc

MEAN STACK

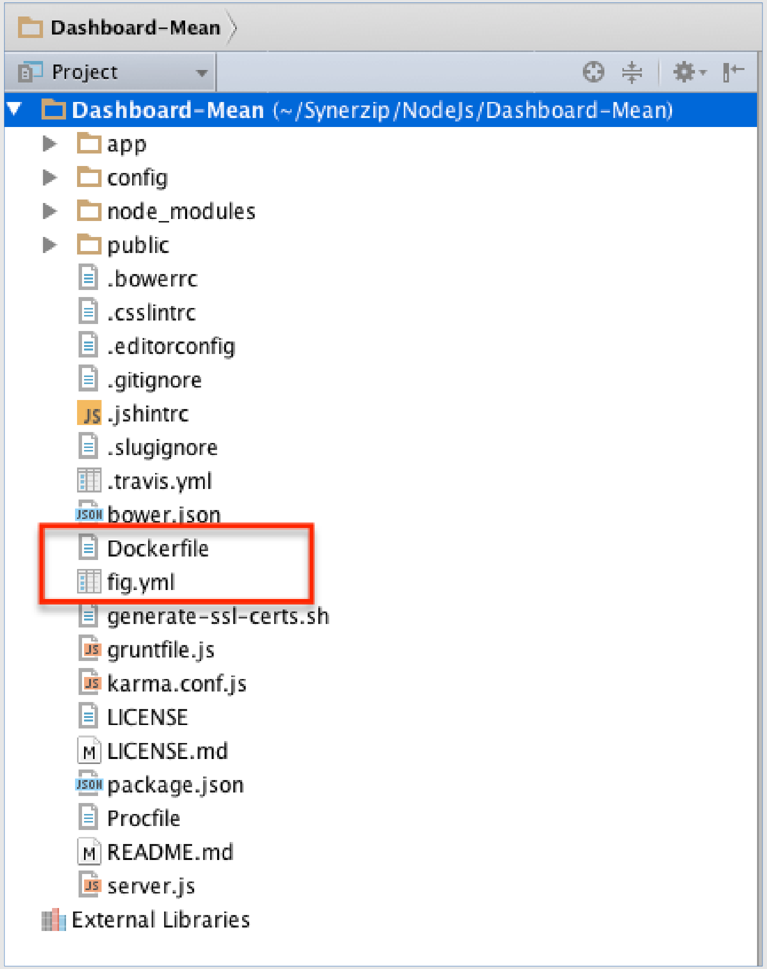
MeanStack Use case

Open-Source Full-Stack Solution for MEAN Applications

MeanStack Use case

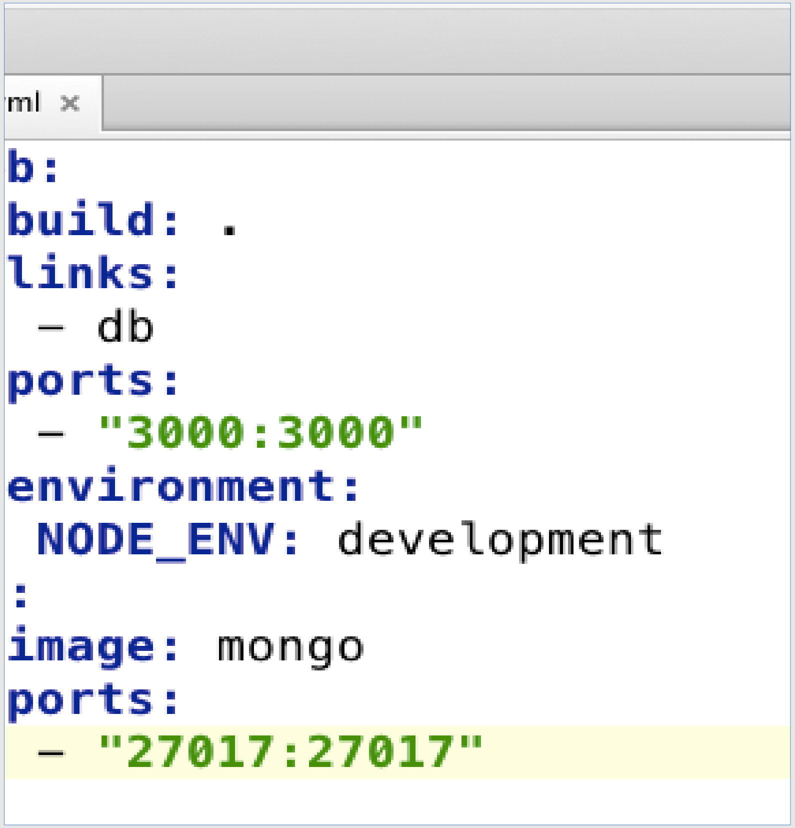
- Mean.js provides
  - Code generator to generate Mean App
  - Mean.js apps typically have
    - Node Js Server
    - Mongo DB database
  - Provides Dockerfile and fig.yml to run the app in Docker Containers
    - One Docker container for Node Js Server
    - One Docker container for Mongo DB Database

MeanStack Use case



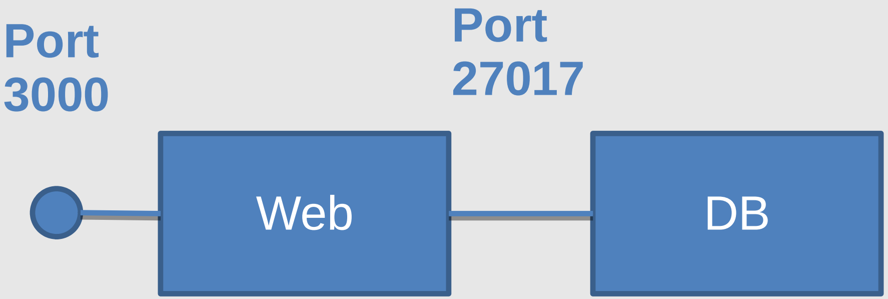
```
▼ Dashboard-Mean (~/Synerzip/NodeJs/Dashboard-Mean)
   ▶ app
   ▶ config
   ▶ node_modules
   ▶ public
     .bowerrc
     .csslintrc
     .editorconfig
     .gitignore
     .jshintrc
     .slugignore
     .travis.yml
     bower.json
     Dockerfile
     fig.yml
     generate-ssl-certs.sh
     gruntfile.js
     karma.conf.js
     LICENSE
     LICENSE.md
     package.json
     Procfile
     README.md
     server.js
  External Libraries
```

MeanStack Use case



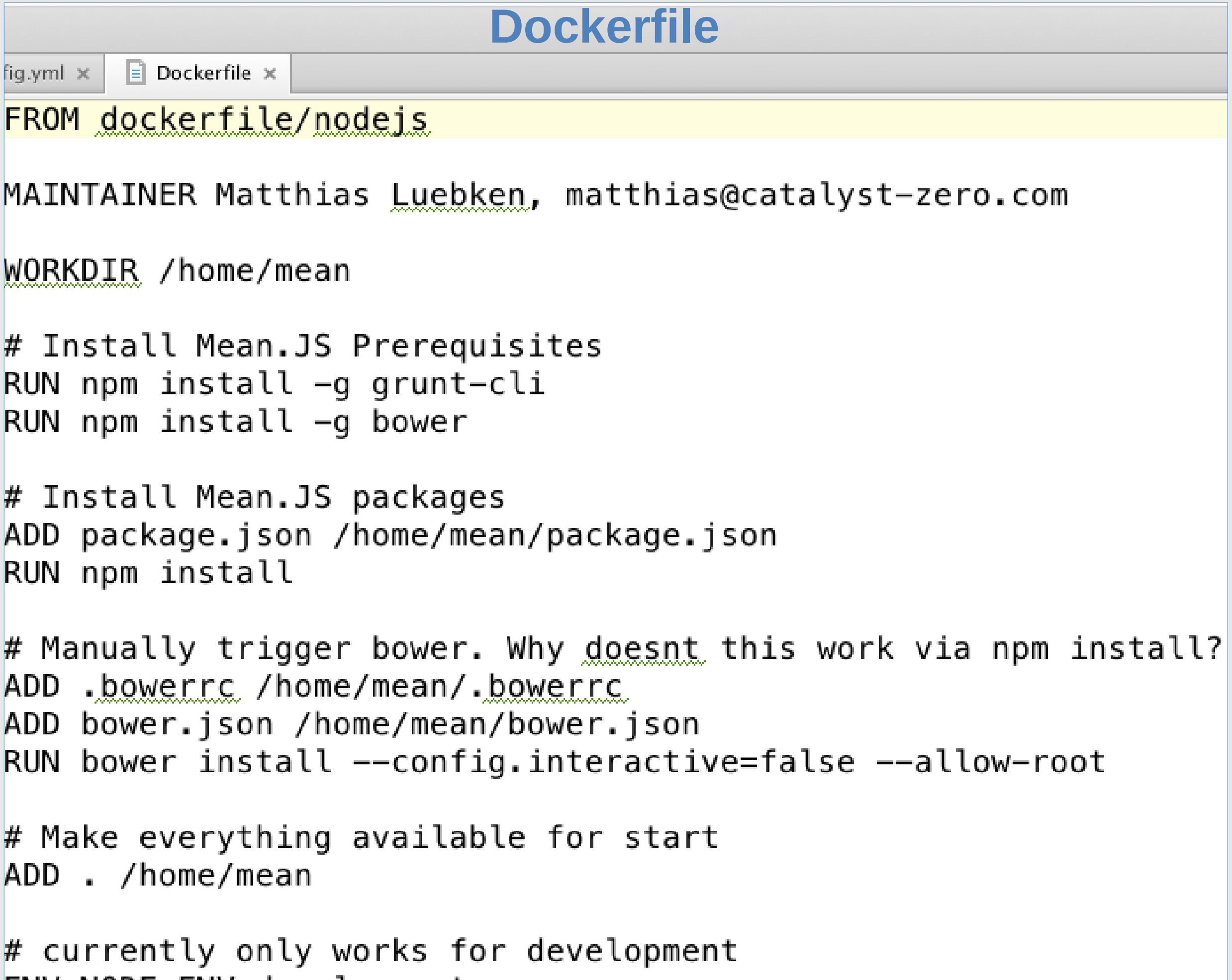
```
b:
build: .
links:
 - db
ports:
 - "3000:3000"
environment:
 NODE_ENV: development
:
image: mongo
ports:
 - "27017:27017"
```

fig.yml



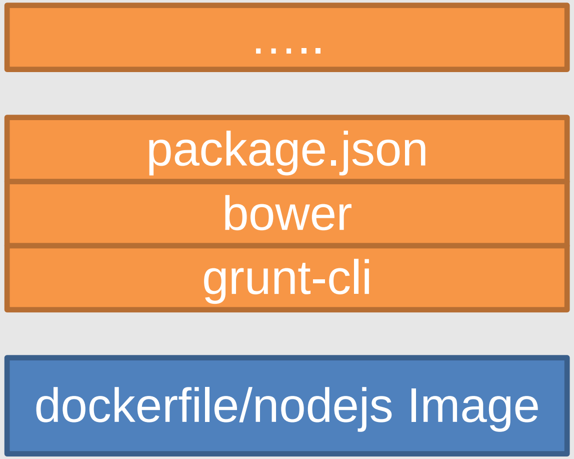
Docker Containers

MeanStack Use case



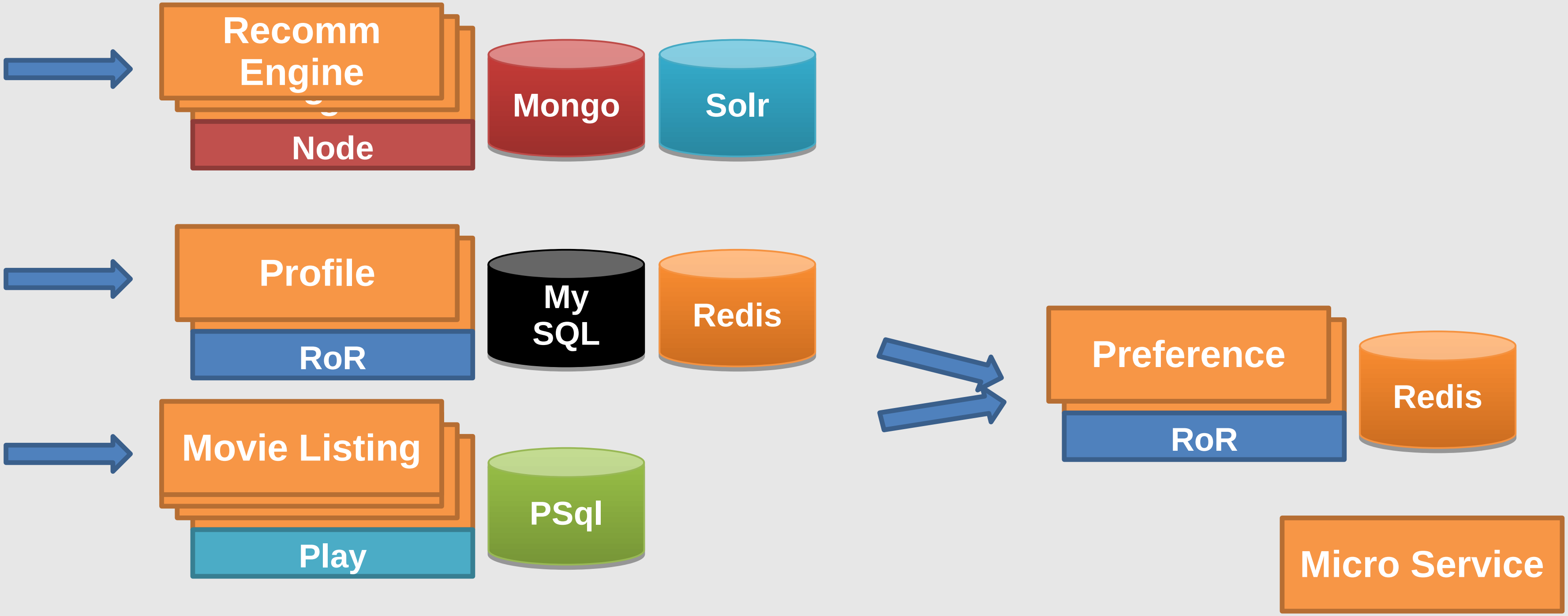
Dockerfile

```
FROM dockerfile/nodejs

MAINTAINER Matthias Luebken, matthias@catalyst-zero.com

WORKDIR /home/mean

# Install Mean.JS Prerequisites
RUN npm install -g grunt-cli
RUN npm install -g bower

# Install Mean.JS packages
ADD package.json /home/mean/package.json
RUN npm install

# Manually trigger bower. Why doesnt this work via npm install?
ADD .bowerrc /home/mean/.bowerrc
ADD bower.json /home/mean/bower.json
RUN bower install --config.interactive=false --allow-root

# Make everything available for start
ADD . /home/mean

# currently only works for development
ENV NODE_ENV development
```



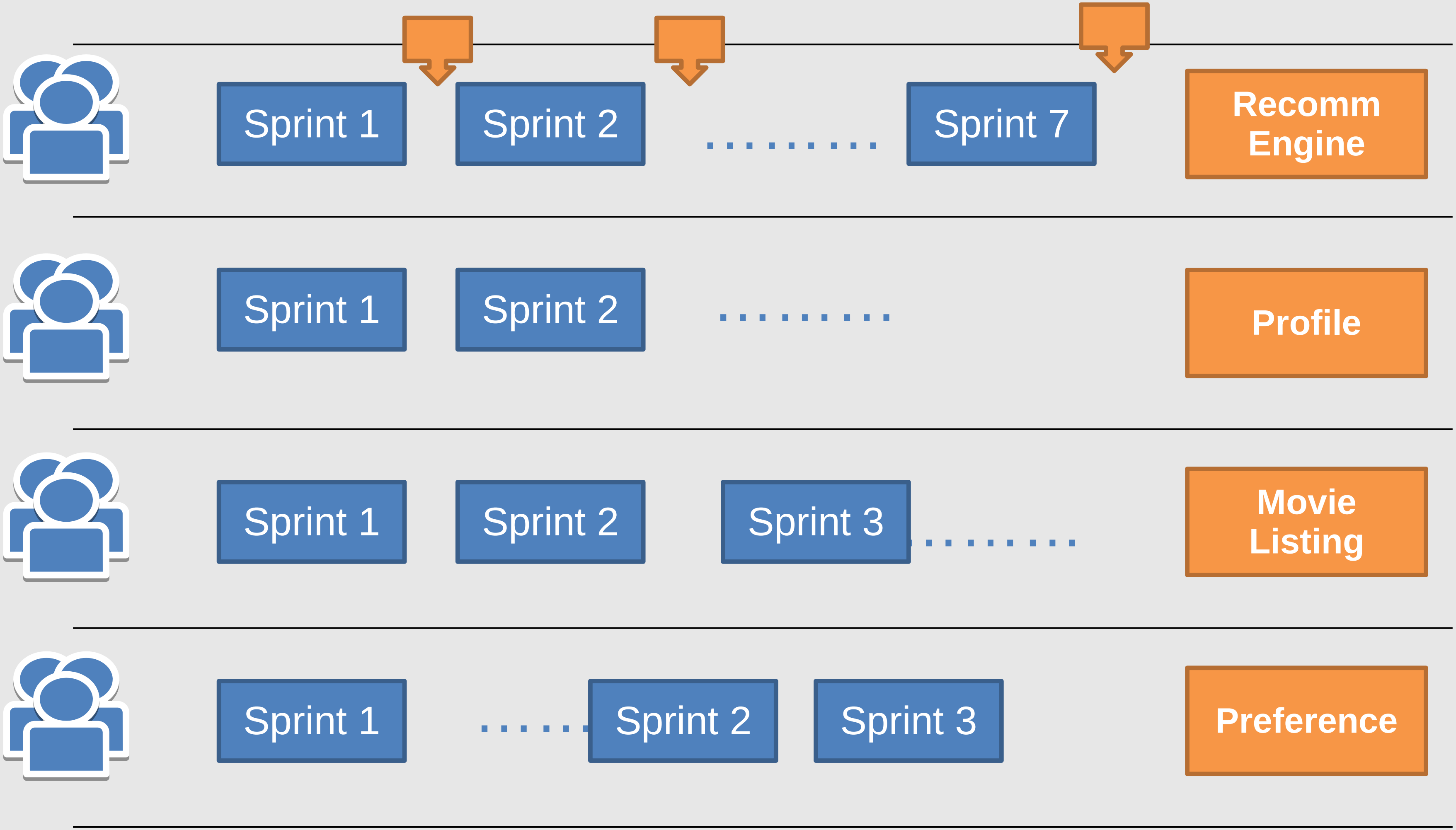
MICRO SERVICES

Micro Services Use case



Micro Services Use case

Deploy



Sprint 1 Sprint 2 .... Sprint 7

Recomm Engine

Sprint 1 Sprint 2 ....

Profile

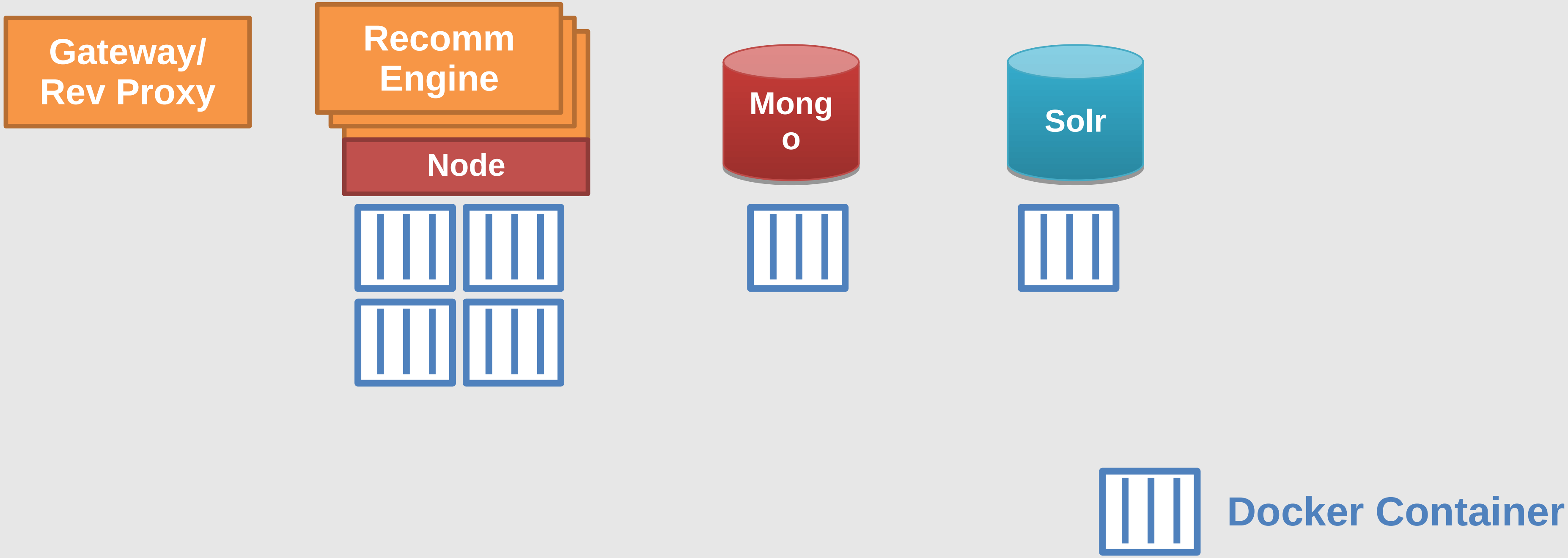
Sprint 1 Sprint 2 Sprint 3 ....

Movie Listing

Sprint 1 ...... Sprint 2 Sprint 3

Preference

Micro Services Use case



Micro Services Use case

- Micro services are hard to run
- Needs strong DevOps process
- Docker helps by
  - Defining container/micro service as unit
  - Shipping one micro service as one container
  - More containers = more scale
  - By improving Dev – Operations relationships

Micro Services Use case

- What else is needed?
  - Scheduling
  - High Availability
  - Service Discovery
  - Etc.

kubernetes by Google

By Invite

Giant Swarm

FUTURE OF DOCKER

NEW DOCKER PRODUCTS

New Docker Products

- Docker Machine
- Docker Swarm
- Docker Compose

Docker Machine

- Machine makes it really easy to create Docker hosts on local hypervisors and cloud providers.
- It creates servers, installs Docker on them, then configures the Docker client to talk to them.

Docker Machine

- Machine makes it really easy to create Docker hosts on local hypervisors and cloud providers.
- It creates servers, installs Docker on them, then configures the Docker client to talk to them.

Docker Swarm

- Swarm is a simple tool which controls a cluster of Docker hosts and exposes it as a single "virtual" host.
- Swarm uses the standard Docker API as its frontend, which means any tool which speaks Docker can control swarm transparently.

Docker Swarm

- Swarm is a simple tool which controls a cluster of Docker hosts and exposes it as a single "virtual" host.
- Swarm uses the standard Docker API as its frontend, which means any tool which speaks Docker can control swarm transparently.

Docker Compose

- An orchestration tool for Docker
- Defines
  - Which Docker containers are to be run
  - How they are connected
  - What ports they expose
  - All in single file
  - Initial design based on Fig.sh
-

What is Docker Compose?

- Define and run multi-container applications
- Specify images and configuration in a simple YAML file:

```
docker-compose.yml
```

- One command to get it all running:

```
$ docker-compose up
```

What is Docker Compose?

`docker-compose up:`

- Builds images from Dockerfiles
- Pulls images from registries
- Creates and starts containers
- Streams their logs

What is Docker Compose?

Make your development environments:

- Repeatable
- Isolated
- Fast

What’s new in 1.3.0?

- Performance and stability improvements
- Lots more config option support
- New feature (experimental!): **Smart Recreate**

  Only recreate containers whose configuration has been changed

  ```
  $ docker-compose up --x-smart-recreate
  ```

  Will eventually be the default behaviour

IAAS/PAAS ADOPTION

IAAS/PAAS Adoption

- Amazon ECS
  - Container service
  - Supports tasks configuration
- Google Cloud
  - Based on Kubernetes
- Microsoft Azure

Thank You

https://www.docker.io/