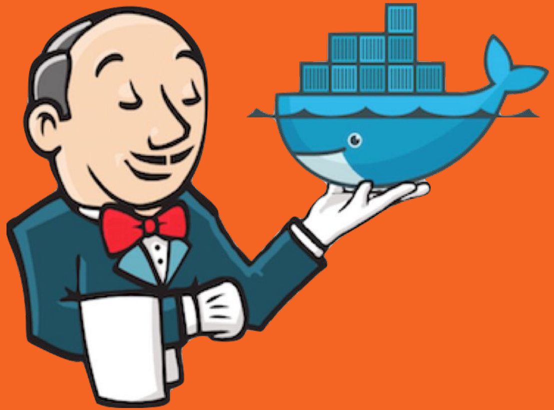
Continuous Delivery
with containers



Mike Long @meekrosoft

Part 1: 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 create your own containers
- How to share your containers
- How to create multi-container applications

Who is Mike Long?

- **Doer**: Embedded software, CoDe & DevOps
- **Trainer**: git, jenkins, docker, TDD
- **Speaker**: coming to a conference near you!
- **Manager**: Co-owner, CEO, Praqma Norway

Check in

- Who are you?
- What do you hope to learn?
- Have you used docker before?
- Have you used jenkins before?
- OS?

What the why now?

If docker is the answer, what is the question?

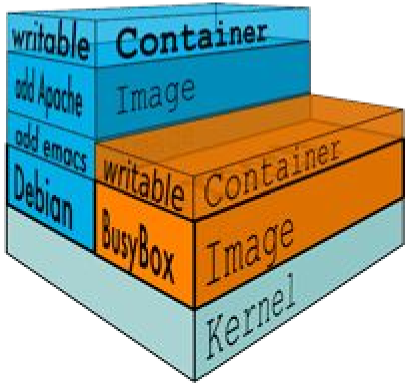
Docker is a platform

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

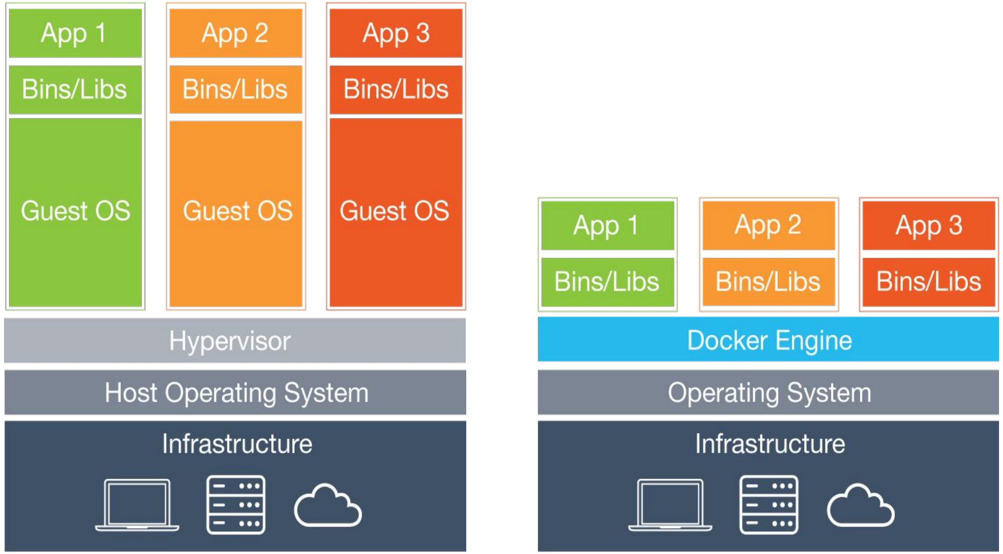
It provides **isolation**, so applications on the same host and stack can avoid dependency conflict

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

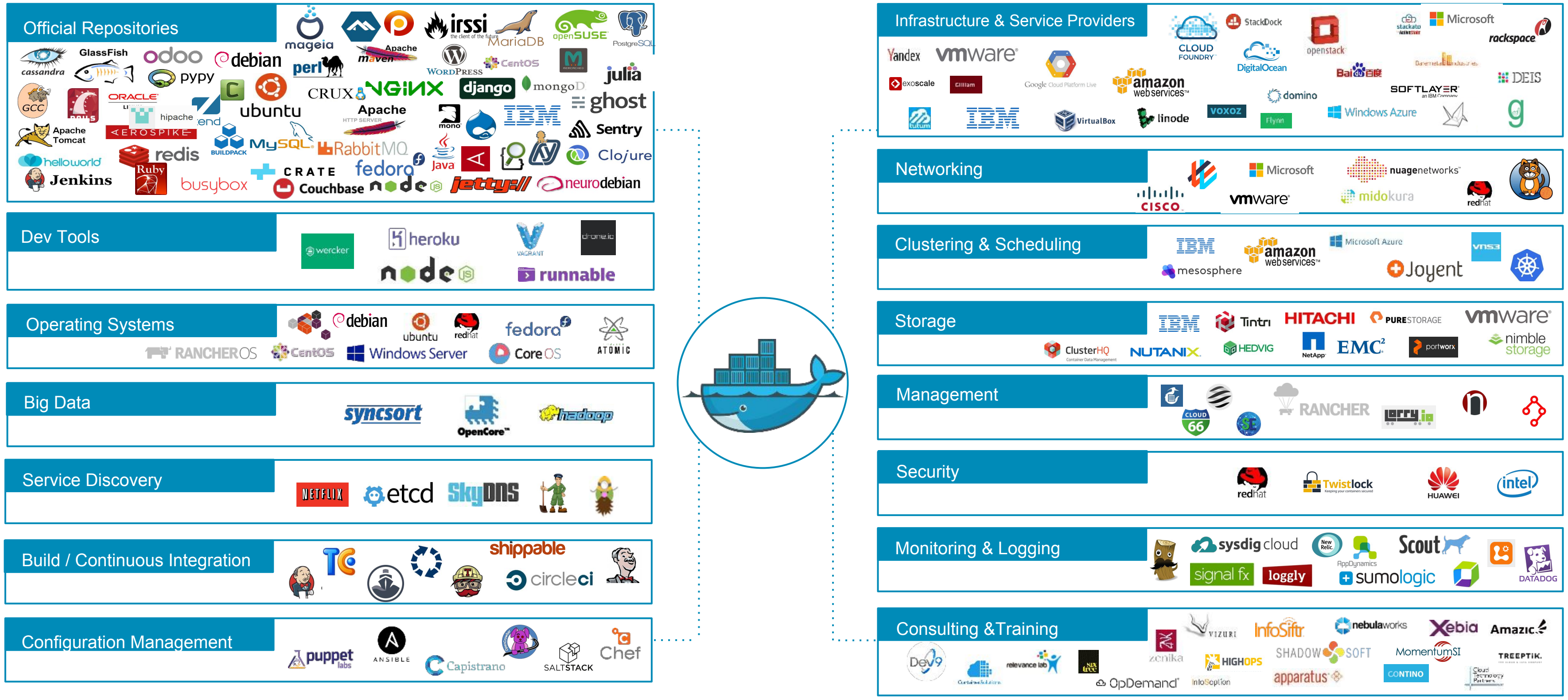
Resource Utilization

Better utilization, more portable, shared operating system



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

The Docker ecosystem



Official Repositories

Dev Tools

Operating Systems

Big Data

Service Discovery

Build / Continuous Integration

Configuration Management

Infrastructure & Service Providers

Networking

Clustering & Scheduling

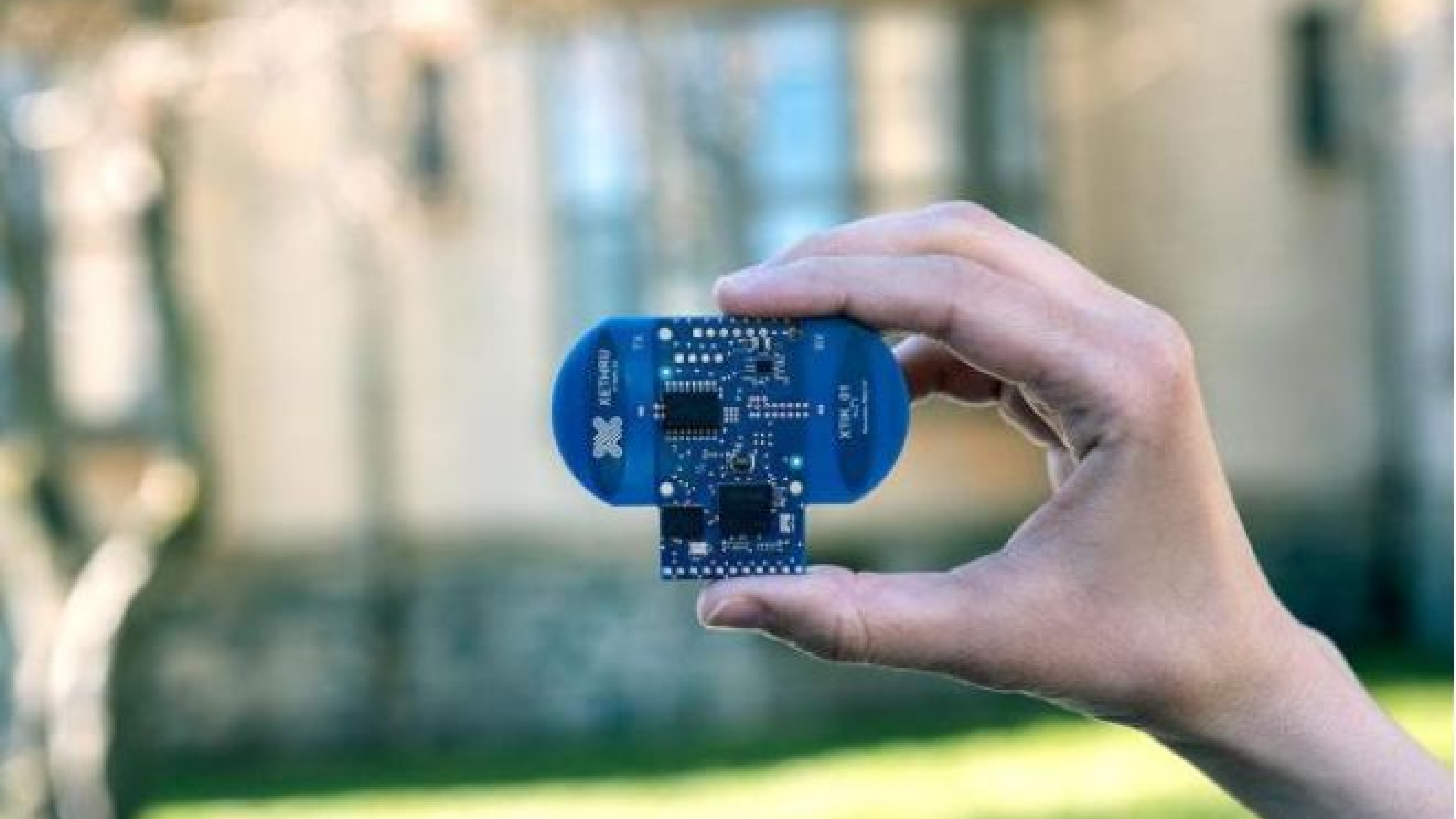
Storage

Management

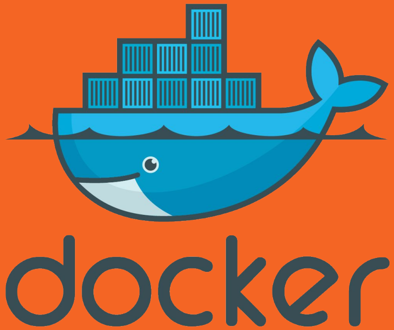
Security

Monitoring & Logging

Consulting &Training

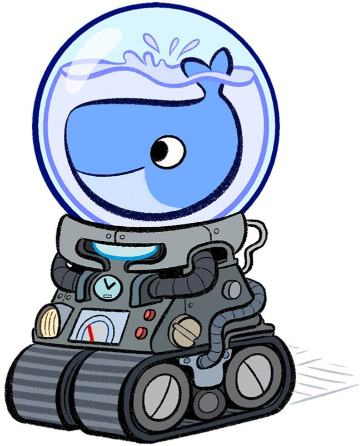


Install docker now!



http://docs.docker.com/

Sidetrack for those of us not on linux...



Docker toolbox is the simplest way to get started running containers on mac and windows systems

It uses virtualbox to create linux virtual machines for running containers

It can also be used to create docker environments on cloud providers such as amazon, google, and digitalocean

You will also need a working git

Are we there yet?

```
$   docker info
```

Are we there yet?

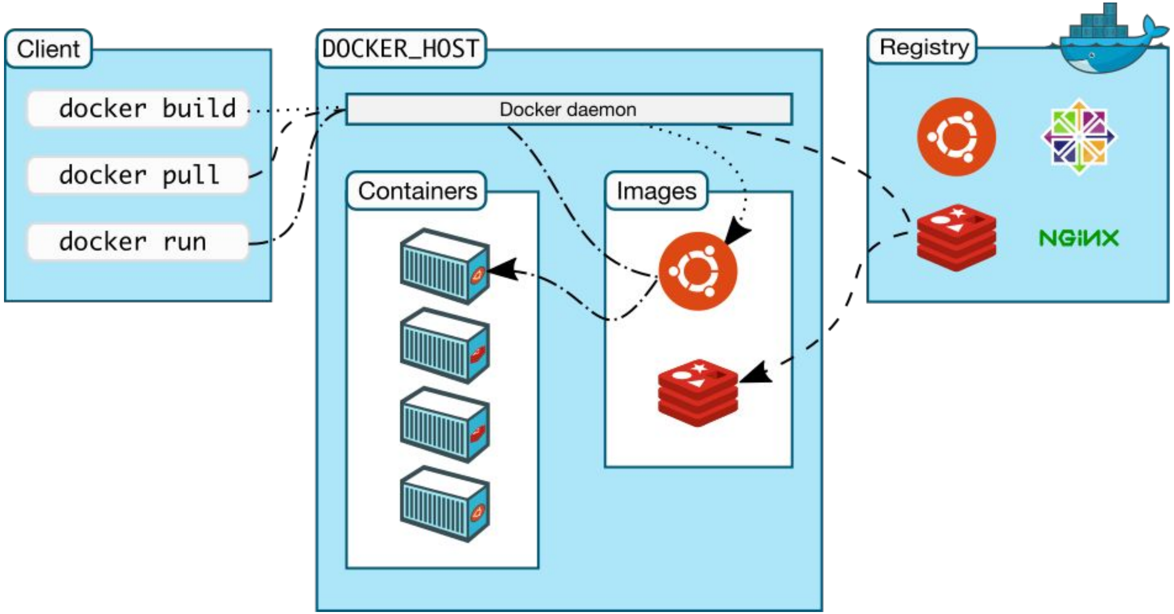
```
$   docker version
```

Let’s create some containers!

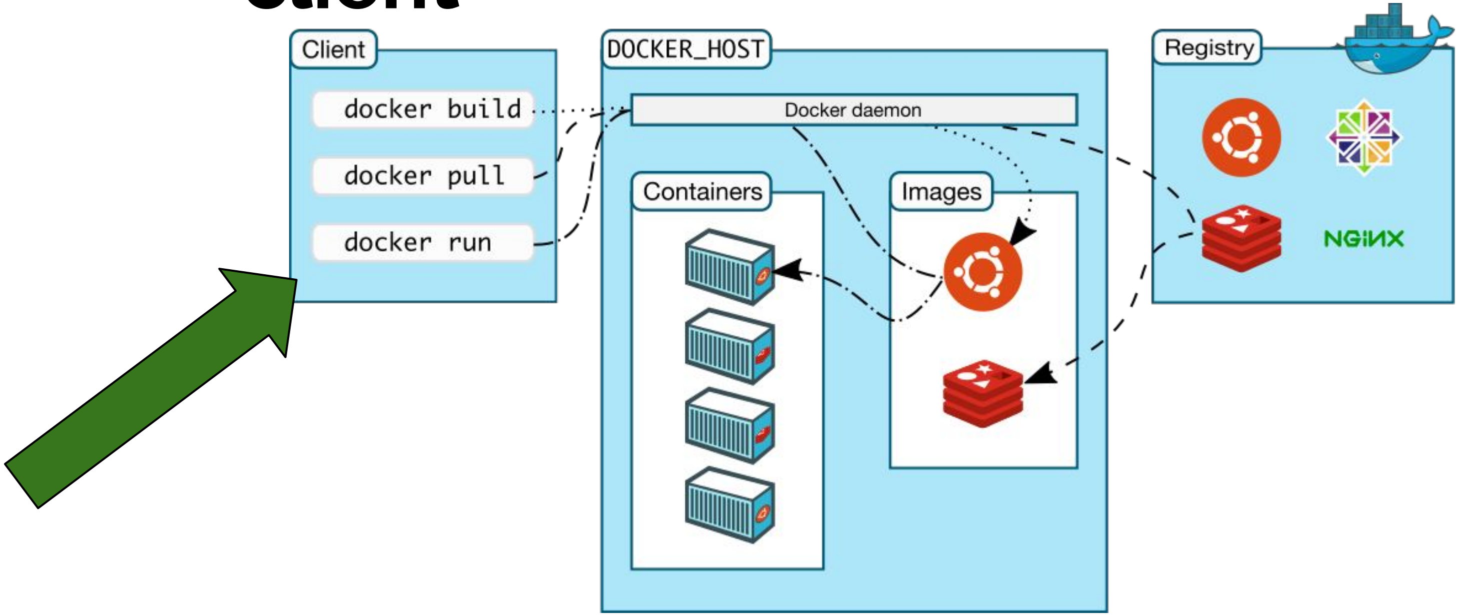
Hello, ACCU!

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

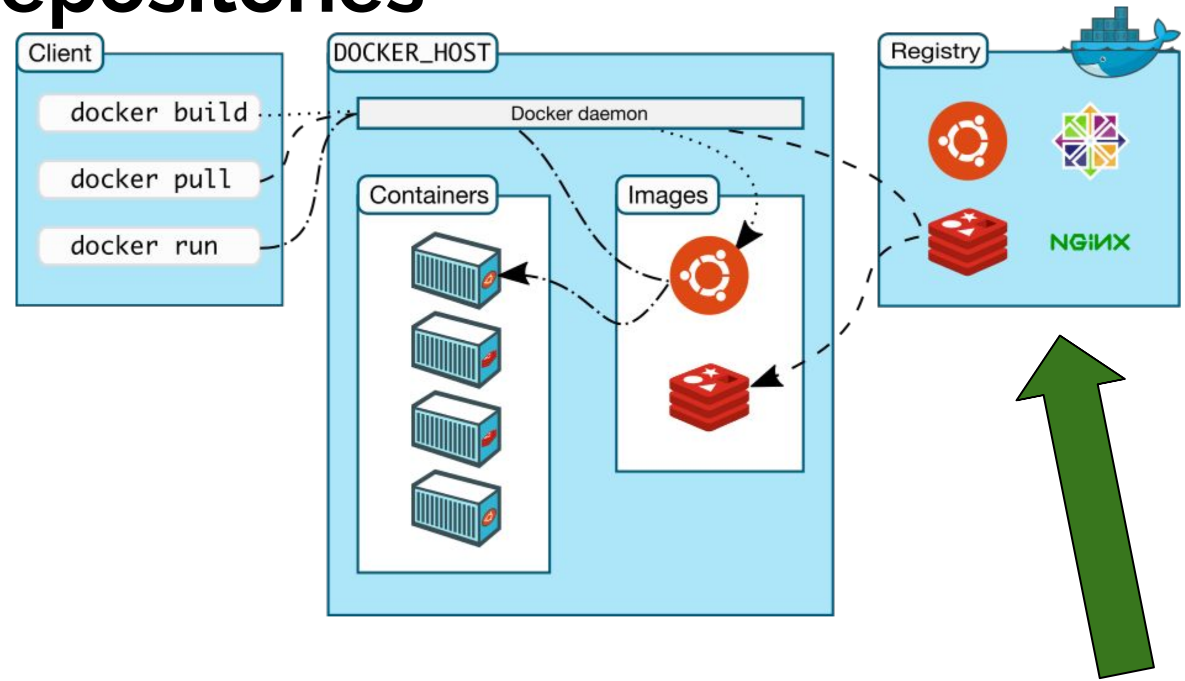
What just happened there then?



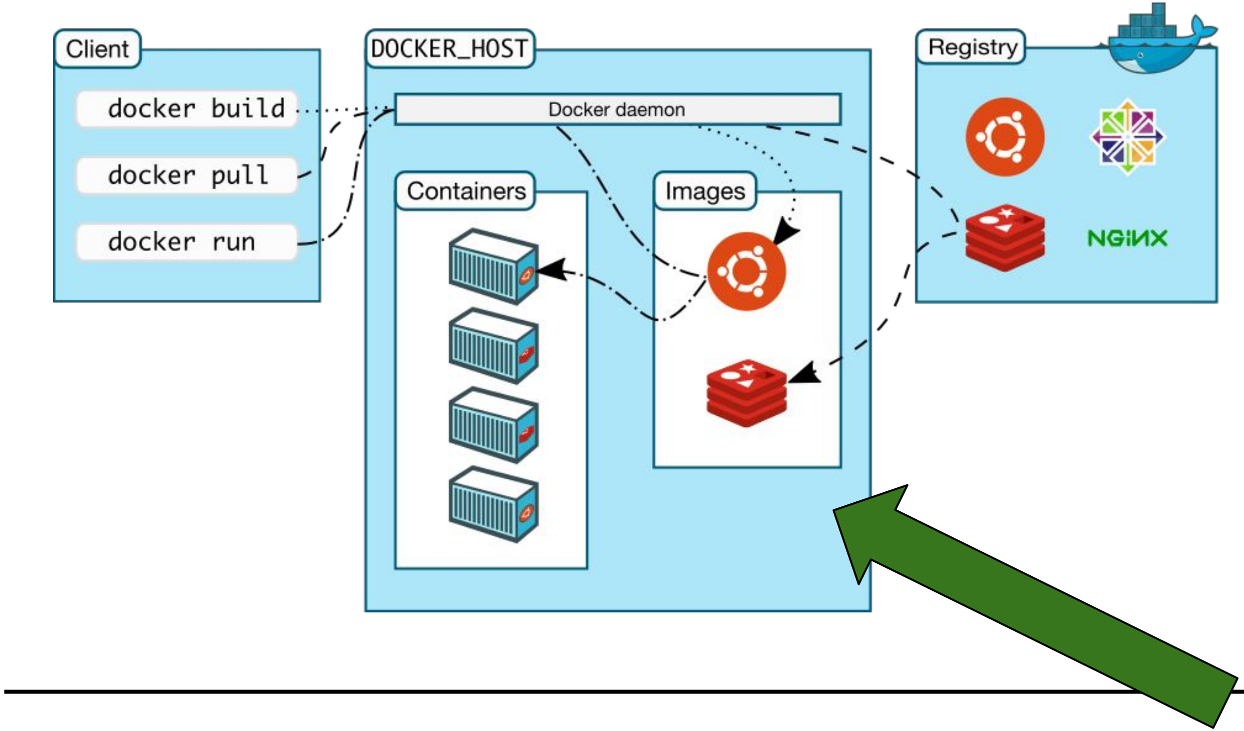
Commands are executed on the client



Images are pulled from repositories



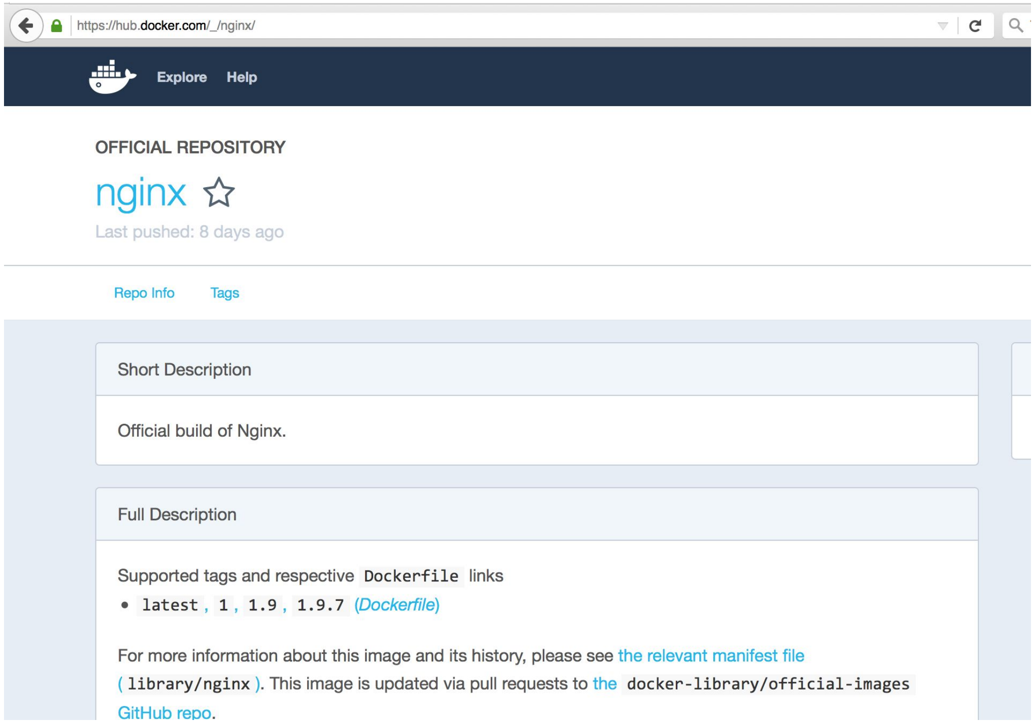
Containers are run from images



An container is...

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

Docker hub



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.

Find out what images you have

```
# docker images
```

Docker will attempt to use local image first
Will look to hub if not found

Image Tags

Images are specified by repository:tag

Default tag is latest

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

Let’s add something to our container

```
$ apt-get install vim
$ vim test.txt
$ exit
```

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

Look at our running containers

```
$ docker ps -a
```

List running containers
Use the -a flag to include stopped containers
Containers have ID's and Names

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*

Time for a web server!

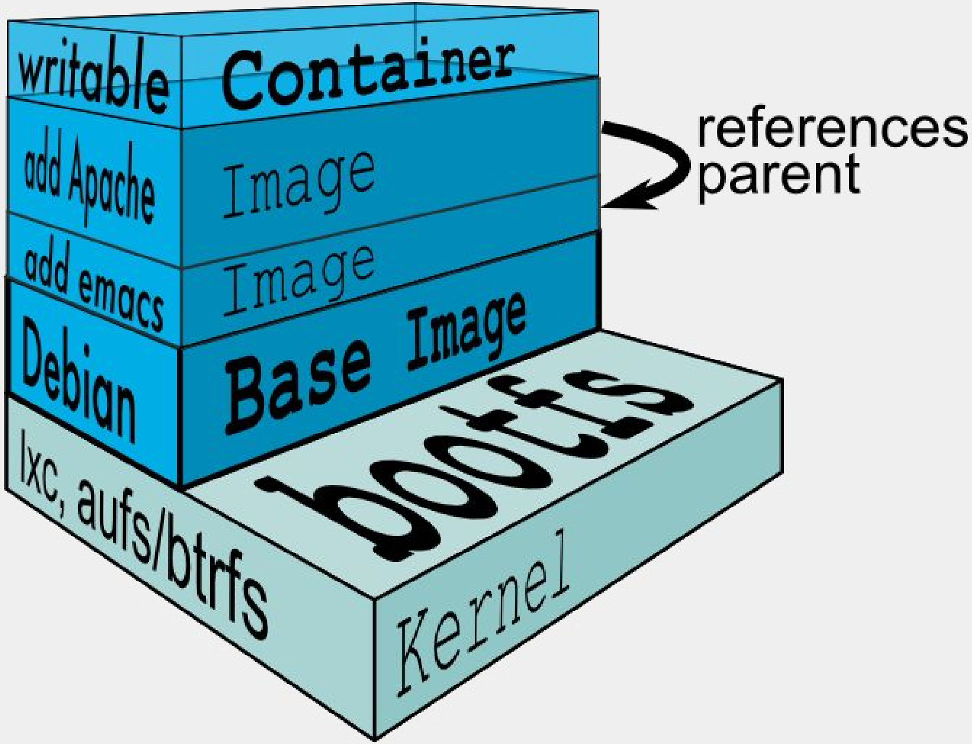
```
$ docker run -d -P nginx
```

Use `docker-machine default` *to get the VM IP*

Use `docker ps` *to get the nginx port mapping*

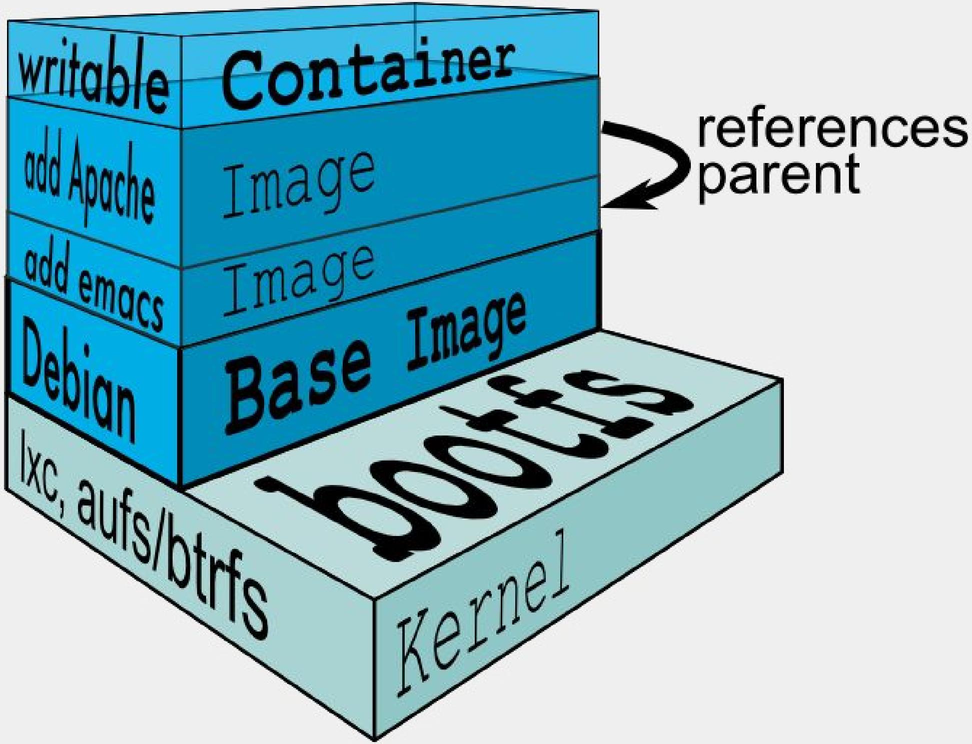
Images

An image is...

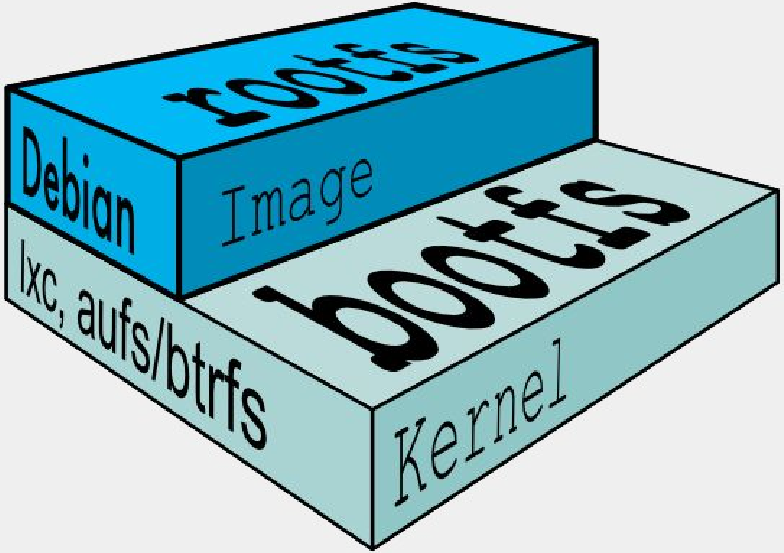


- A read-only template for creating containers
- The **build** component of docker
- Stored in registries
- Can be created by yourself distributed by others

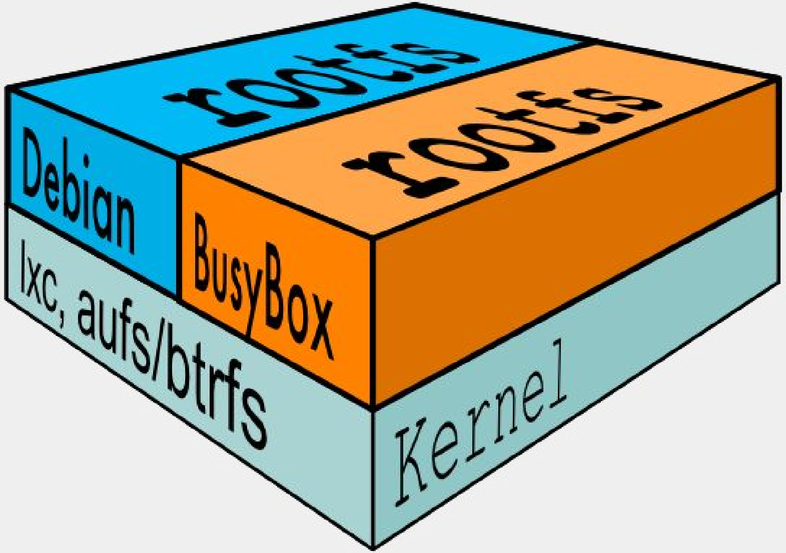
Images are layered read-only filesystems



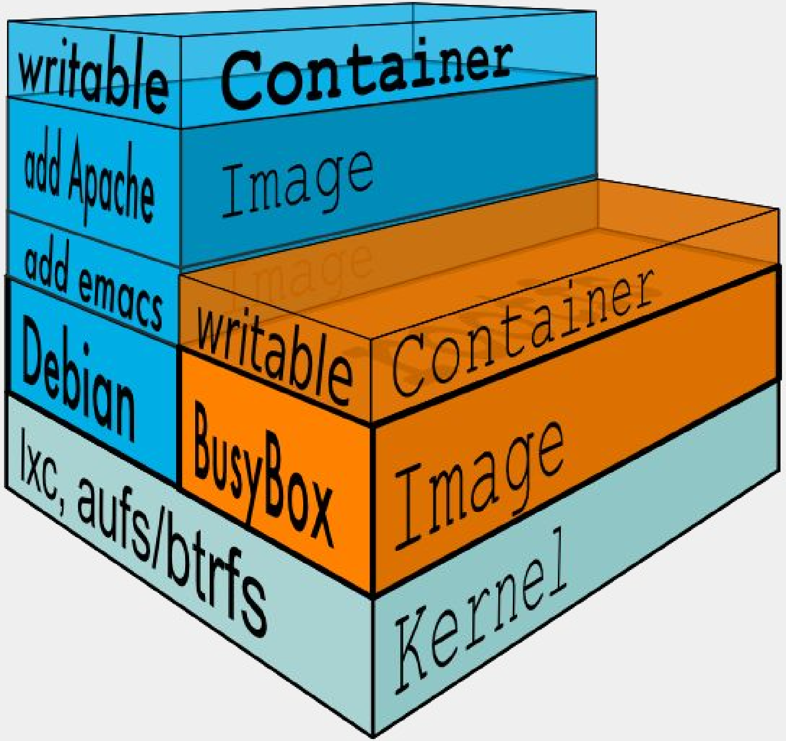
Images have base layers



Multiple root file systems per host are normal



When an image is run, a writable layer is added



Downloading an image with pull

```
$ docker pull busybox
```

Let’s make an image

Docker commit saves changes in a container as a new image

```
$ docker commit 234d3ea32 meekrosoft/myapp:1.0
```

Let’s run our new image

```
$ docker run -ti meekrosoft/myapp:1.0 bash
root@2343245:/# curl 127.0.0.1
```

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**

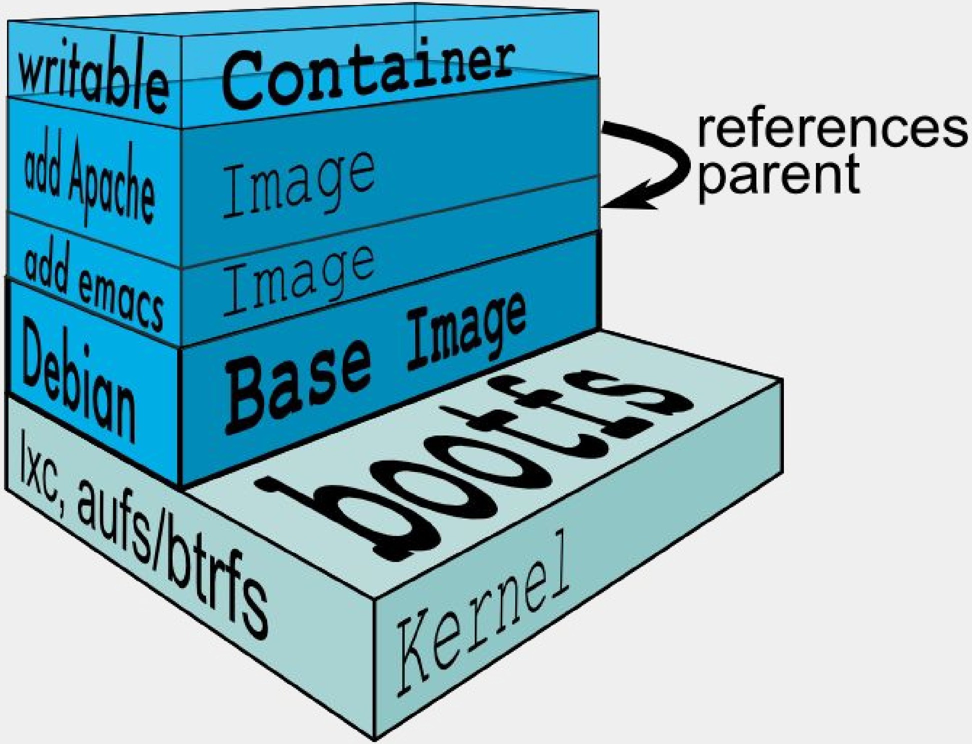
More **effective** than using `commit`

- 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 install curl
RUN apt-get install vim
```

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 meekrosoft/myapp:1.1 .
```

The build command takes a build context on the filesystem

-f flag can be used to specify a different location for the Dockerfile

Go ahead and make your image

The CMD instruction

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

Can only be defined once
Can be overridden at run time

Run your new image with and without a command

The ENTRYPOINT instruction

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

Cannot be overridden at run time
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]
```

Can only remove stopped containers

Deleting images

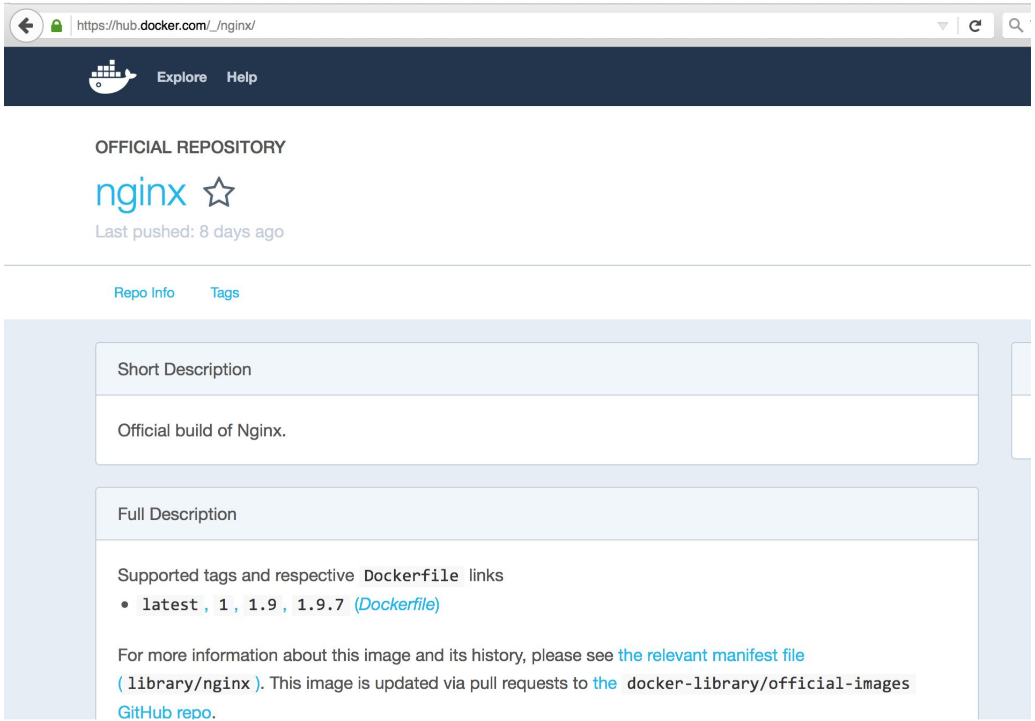
```
$ docker rmi meekrosoft/curl:1.0
```

Wipe em all out

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

Sharing containers

Let’s add our repository on hub



OFFICIAL REPOSITORY

nginx ☆

Last pushed: 8 days ago

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.

Make a tag that matches our repository on hub

```
$ docker tag meekrosoft/myapp:1.0 meekrosoft/mycurl:1.0
```

Push to hub

```
$ docker push meekrosoft/mycurl: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
```

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

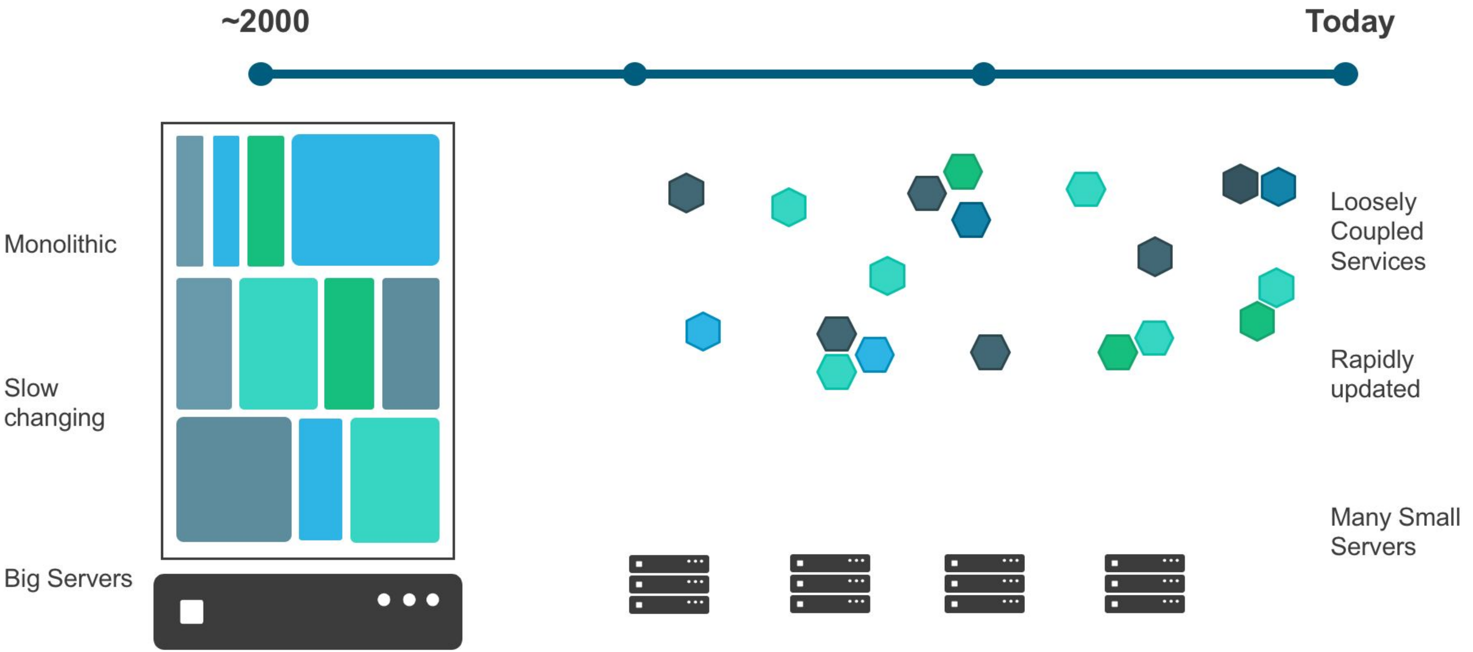
Containers should be ephemeral

Avoid mounting directories from the host in production

Data containers are recommended

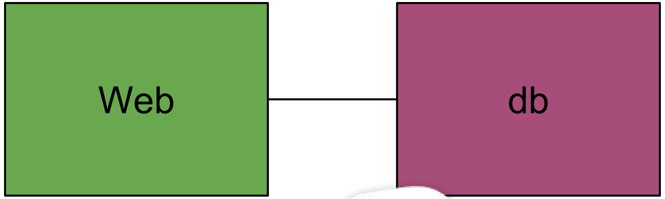
Docker compose

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

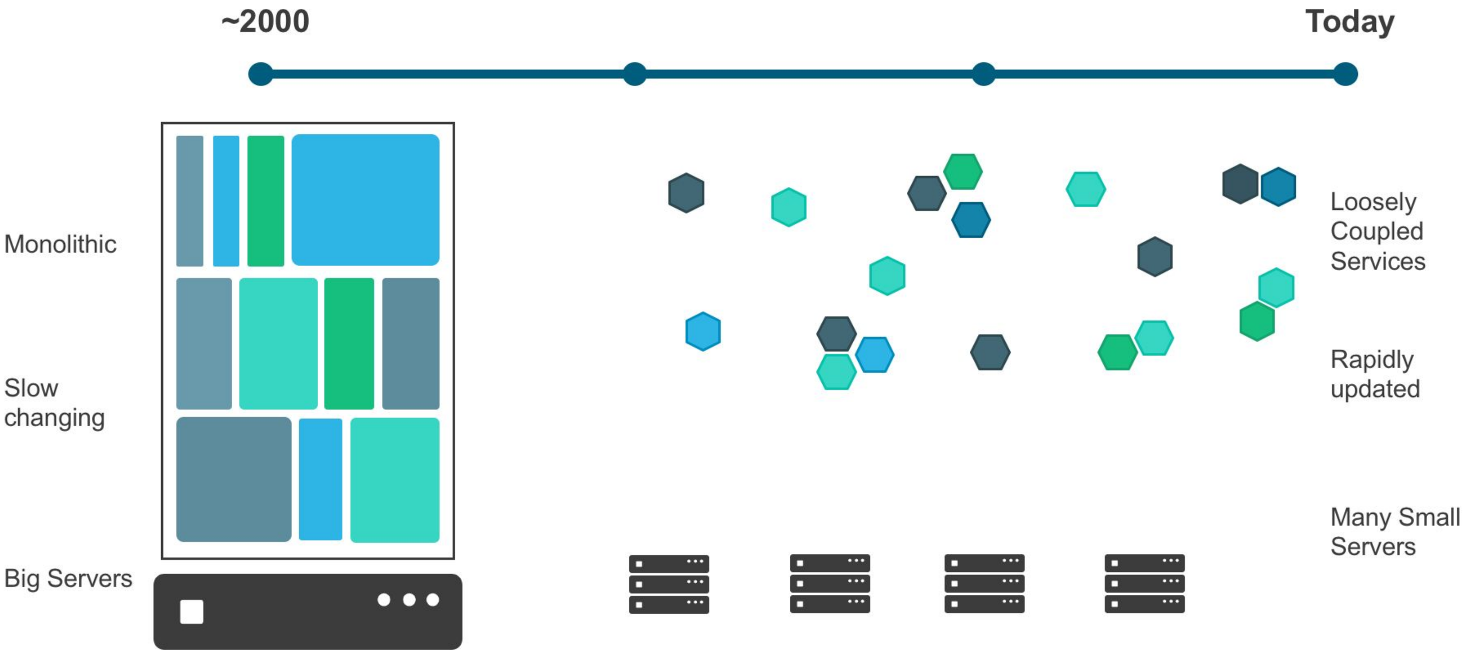
```
$ docker-compose up
$ docker-compose -d up
$ docker ps
$ docker-compose ps
$ docker-compose start <service name>
$ docker-compose stop <service name>
$ docker-compose rm <-v> <service name>
```

Using docker-compose continued...

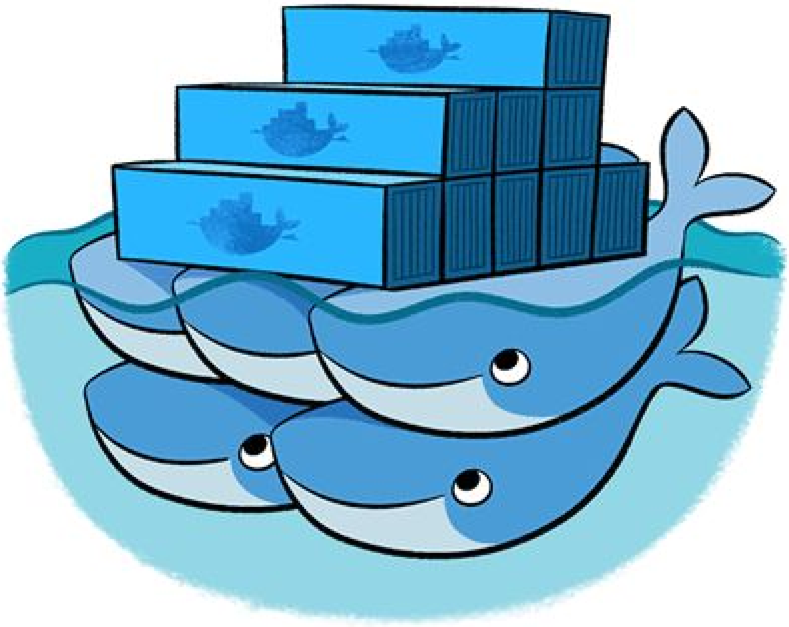
```
$ docker-compose logs
$ docker-compose scale
$ docker-compose -f compose-net.yml --x-
networking up -d
```

Multi-host applications

Transforming the Application Landscape



Using docker-swarm to create multi-host apps



Cluster technology for containers

Integrated networking and volumes

High availability options

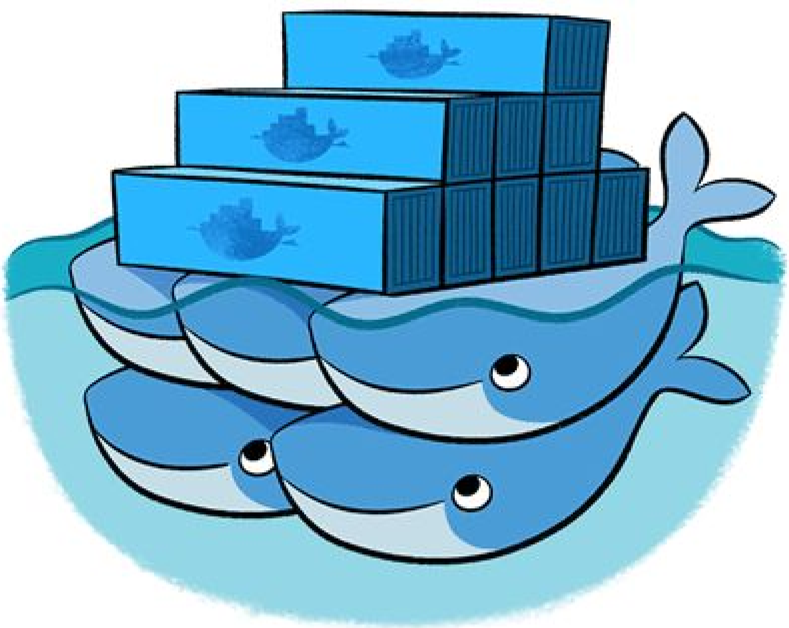
Pluggable schedulers and node discovery

Set up a docker-swarm using docker-machine

https://docs.docker.com/swarm/install-w-machine/

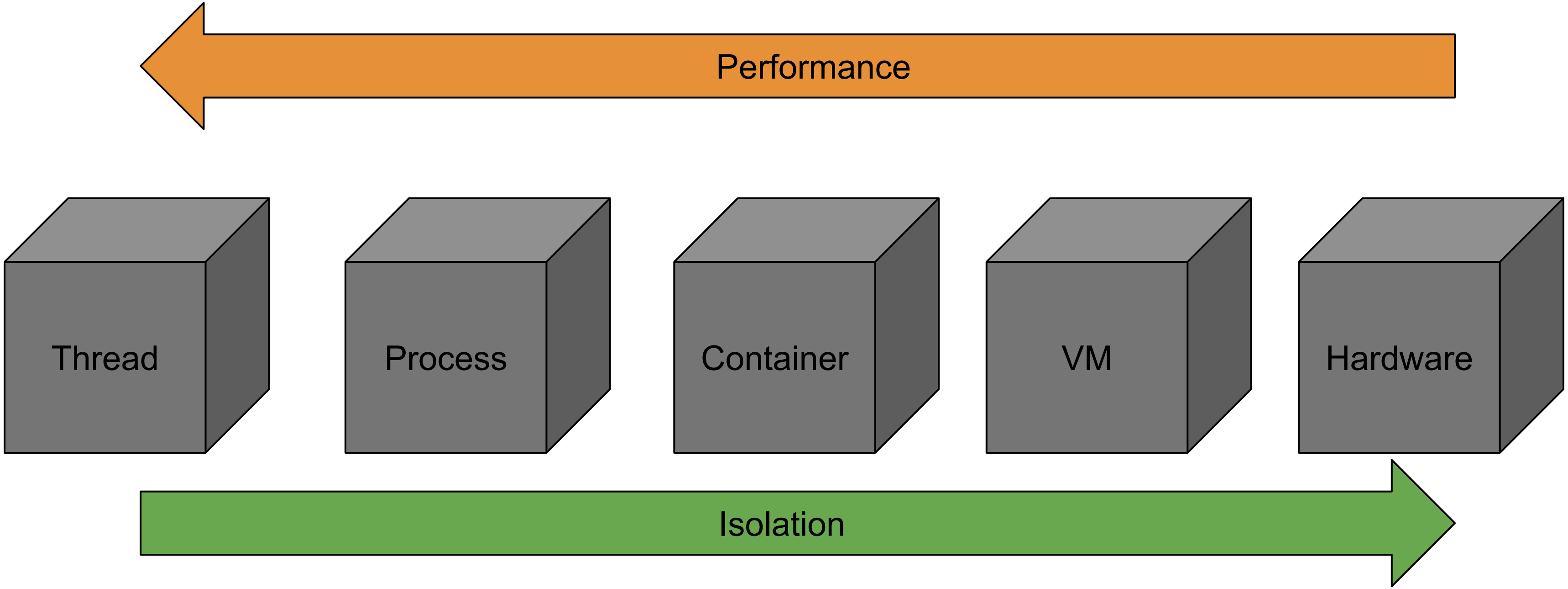
```
$ eval $(docker-machine env --swarm swarm-master)
$ docker ps -a
```

A tour of swarm



https://docs.docker.com/swarm/install-w-machine/
https://gist.github.com/meekrosoft/f4f345331aaee2c917c44e78699c29ef

Where are we now?


Performance
Thread
Process
Container
VM
Hardware
Isolation

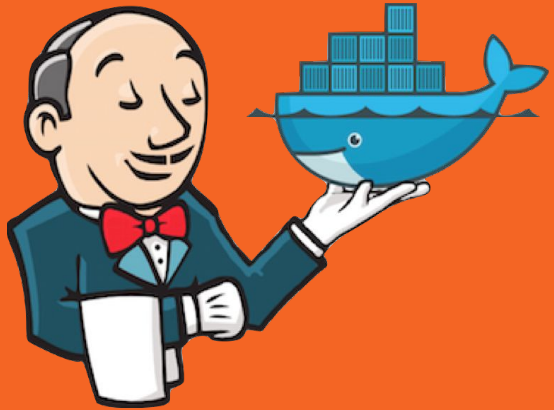
A brief tour of Docker

By the end of this workshop you will understand:

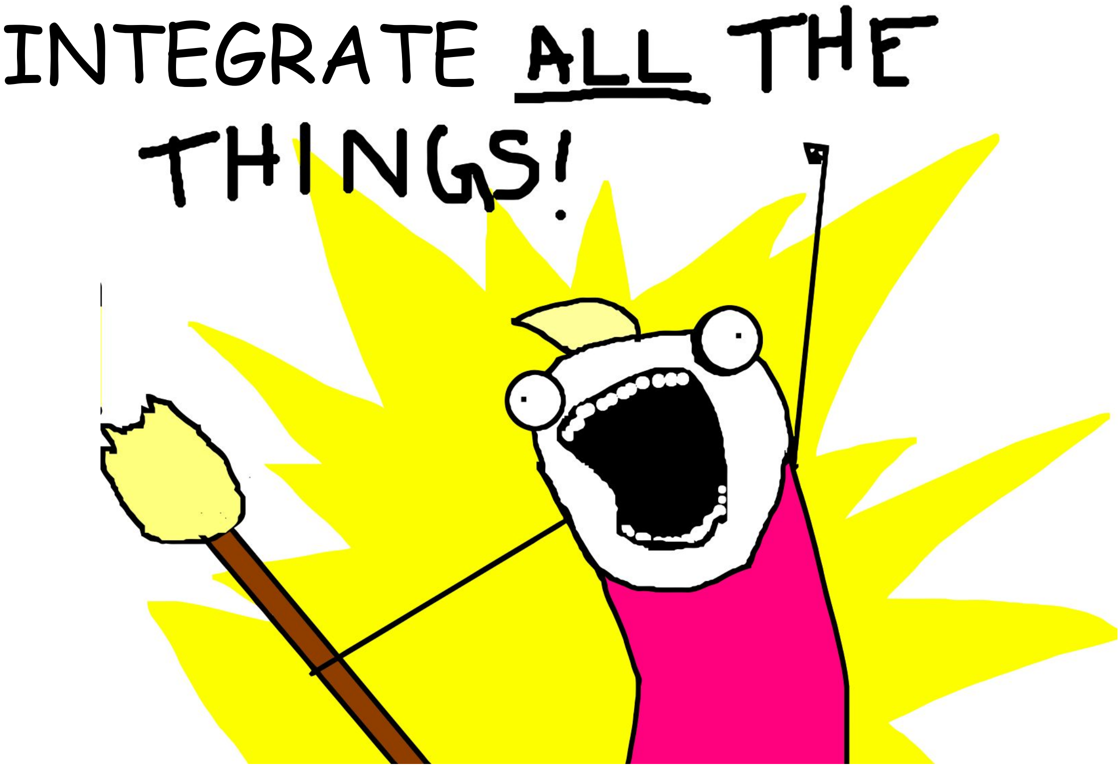
- What is a container and why you may want one
- How to create your own containers
- How to share your containers
- How to create multi-container applications
- How to create multi-host applications

Continuous Delivery

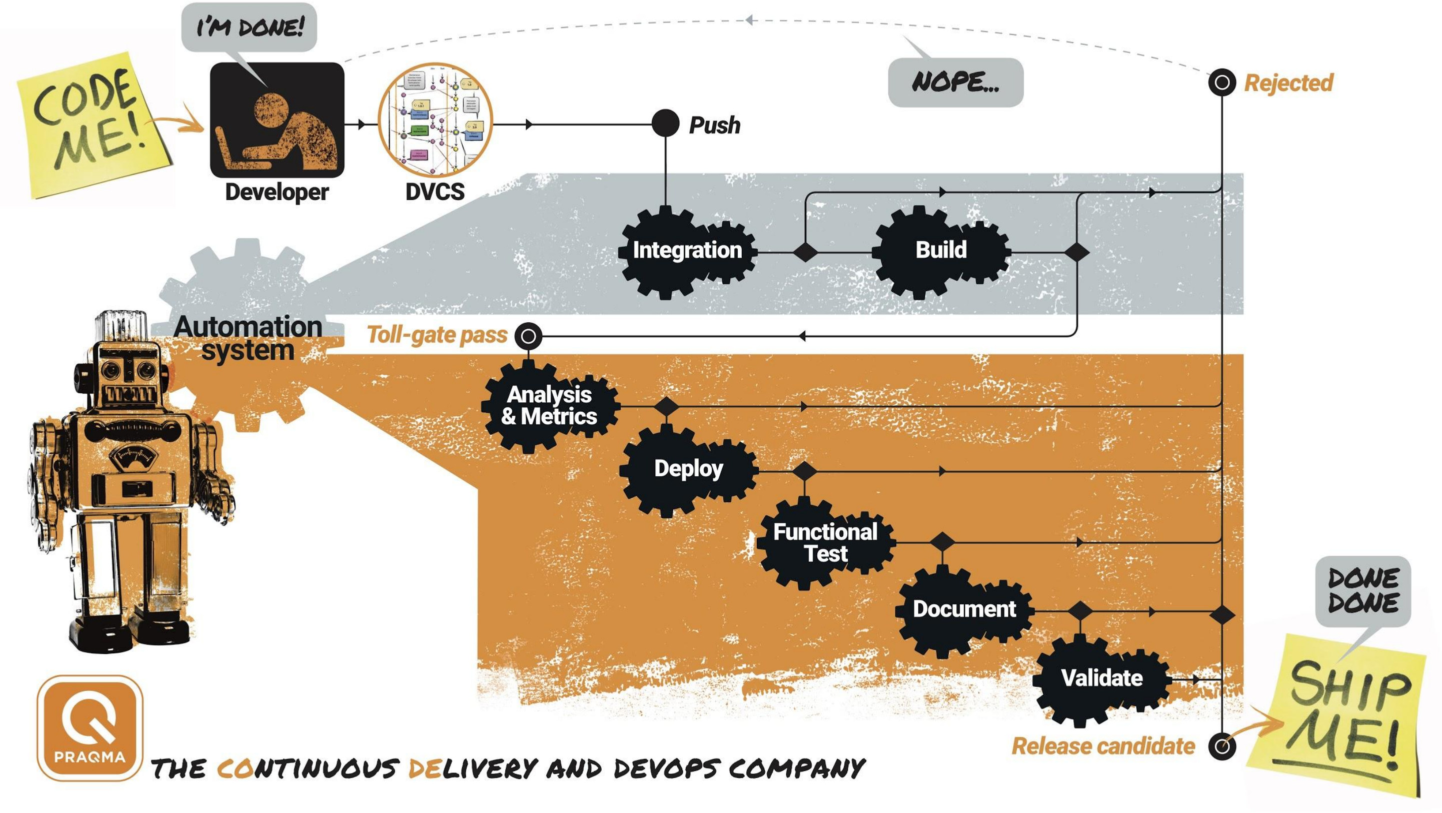
with containers



Mike Long


INTEGRATE ALL THE
THINGS!

CONTINUOUS DELIVERY STORYLINE



THE CONTINUOUS DELIVERY AND DEVOPS COMPANY

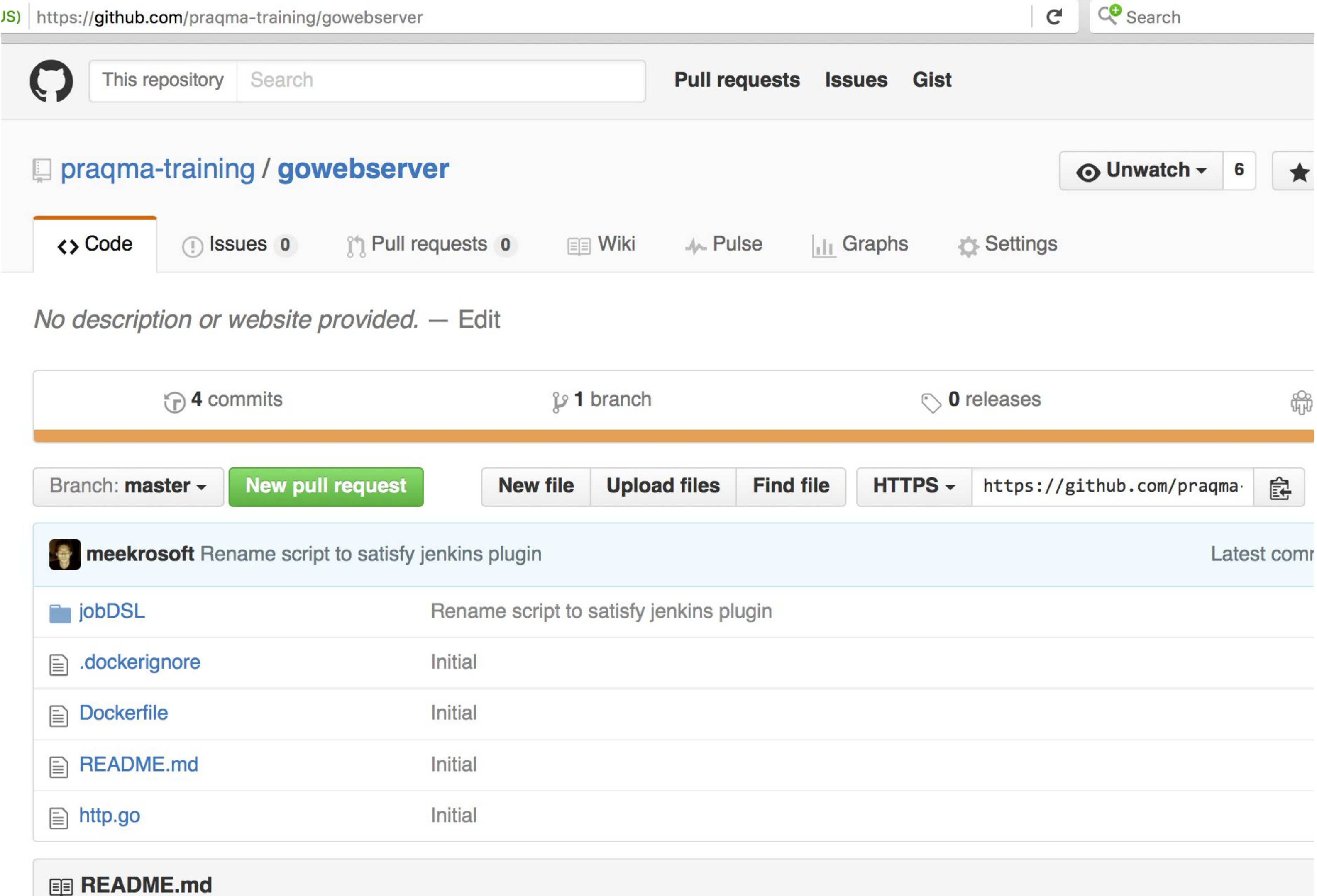
@meekrosoft

The trip

main ports of call

- Running your CI/CD infrastructure in Docker
- Build pipeline “as code”
- ‘Gonas’ and the whale

Our app



praqma-training / gowebserver

No description or website provided. — Edit

meekrosoft Rename script to satisfy jenkins plugin

jobDSL	Rename script to satisfy jenkins plugin
.dockerignore	Initial
Dockerfile	Initial
README.md	Initial
http.go	Initial

Let’s Code!

```
$ git clone https://github.com/praqma-training/gowebserver
```

Create a virtual machine for running the CI system (optional)

```
$ docker-machine create --driver virtualbox code
```

Let’s go for a spin!

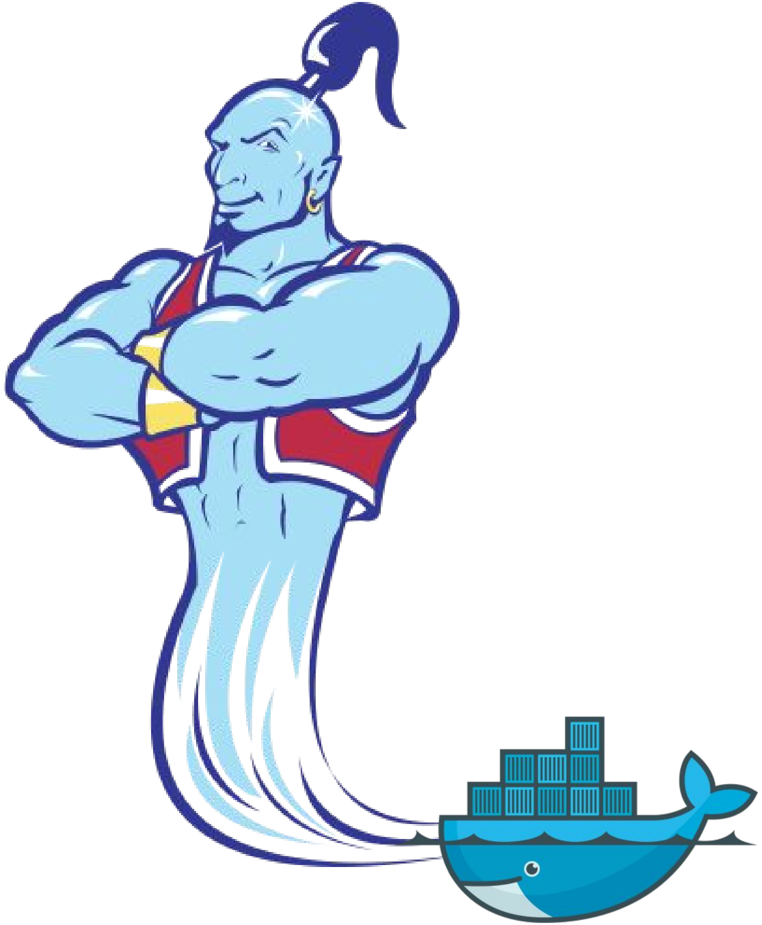
```
$ docker build -t myapp .
$ docker run -d -p 8000:8000 --name myapp myapp:latest
$ curl $(docker-machine ip code):8000
```

CoDe Infrastructure

Servers as Pets

Servers as Pets -> Cattle

Servers as Pets -> Cattle -> Phoenix



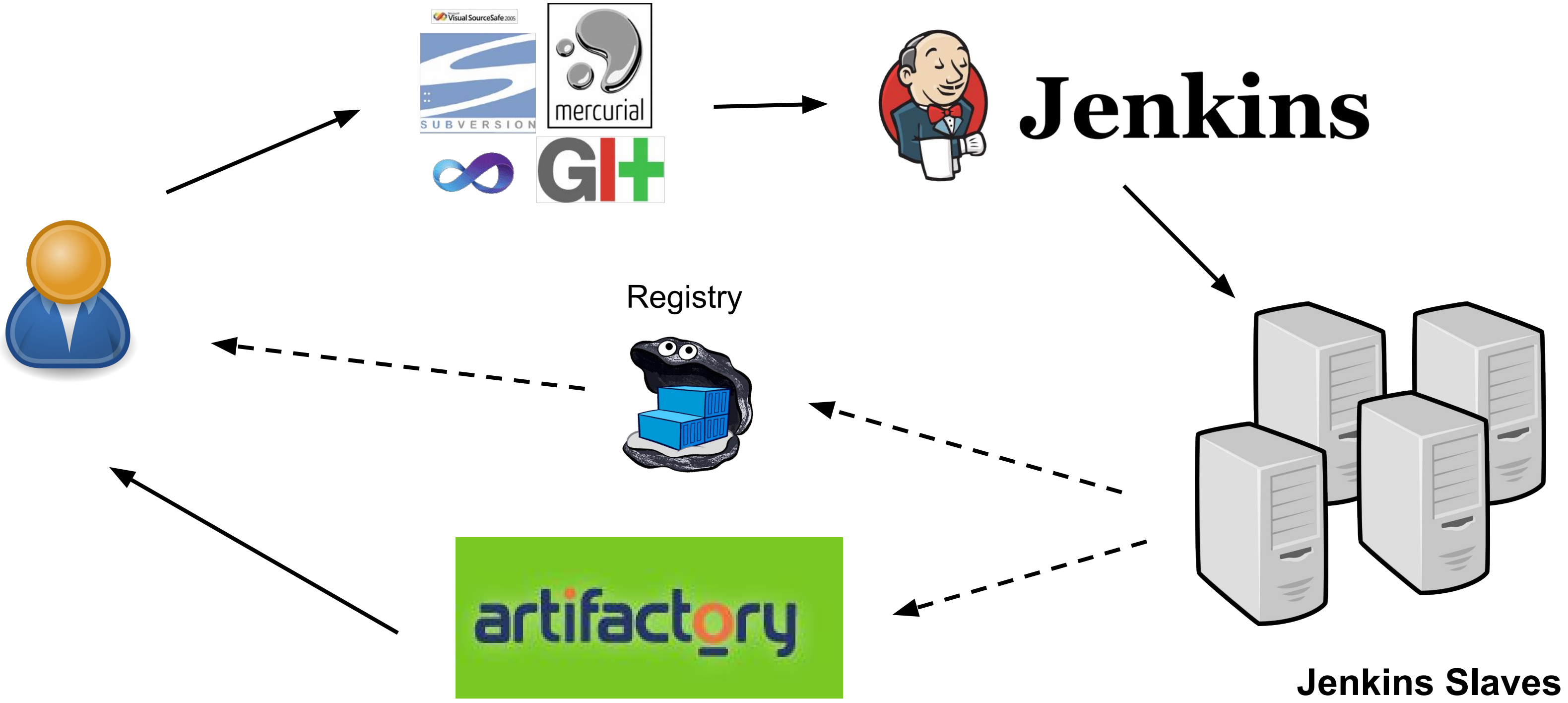
Servers as Pets -> Cattle -> Phoenix -> Genie

Components

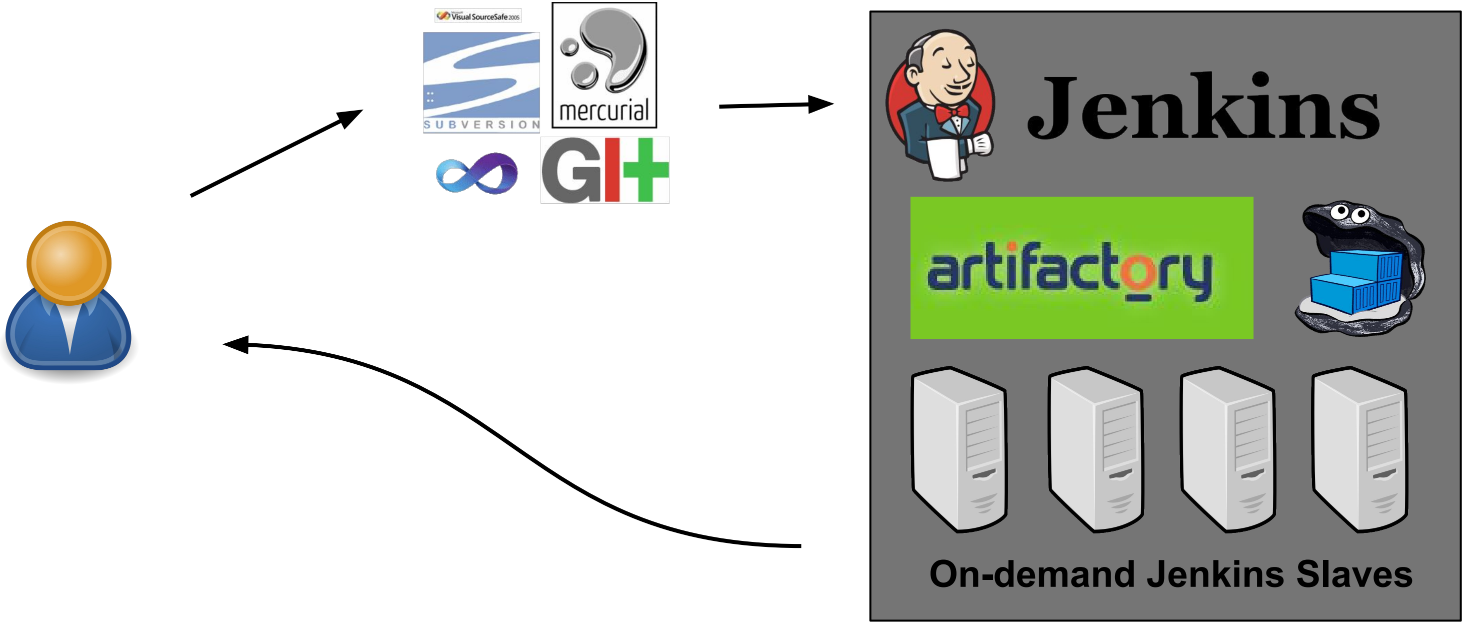
All as Docker containers

- Jenkins master
- Jenkins slaves
- Artifactory
- Docker Registry

The flow


Visual SourceSafe 2005
SUBVERSION
mercurial
GIT
Jenkins
Registry
Jenkins Slaves
artifactory

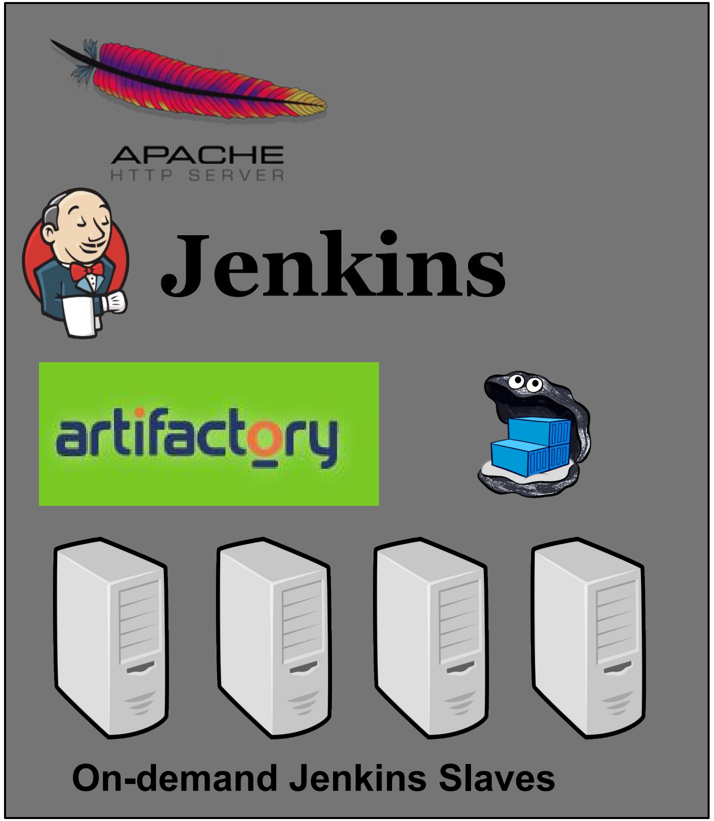
The flow ... with Docker



Docker Host

Docker-Compose

```
apache:
  build: apache/
  ports:
   - "80:80"
  links:
   - jenkins
   - artifactory
   - registry
jenkins:
  build: jenkins/
  ports:
   - "50000:50000"
  volumes:
   - /opt/containers/jenkins_home:/var/jenkins_home
  command: --prefix=/jenkins
artifactory:
  image: mattgruter/artifactory
  volumes:
   - /opt/containers/artifactory/data:/artifactory/data
   - /opt/containers/artifactory/logs:/artifactory/logs
   - /opt/containers/artifactory/backup:/artifactory/backup
  environment:
   - JAVA_OPTS='-Djsse.enableSNIExtension=false'
registry:
  image: registry:2
  volumes:
   - /opt/containers/registry:/var/lib/registry
```

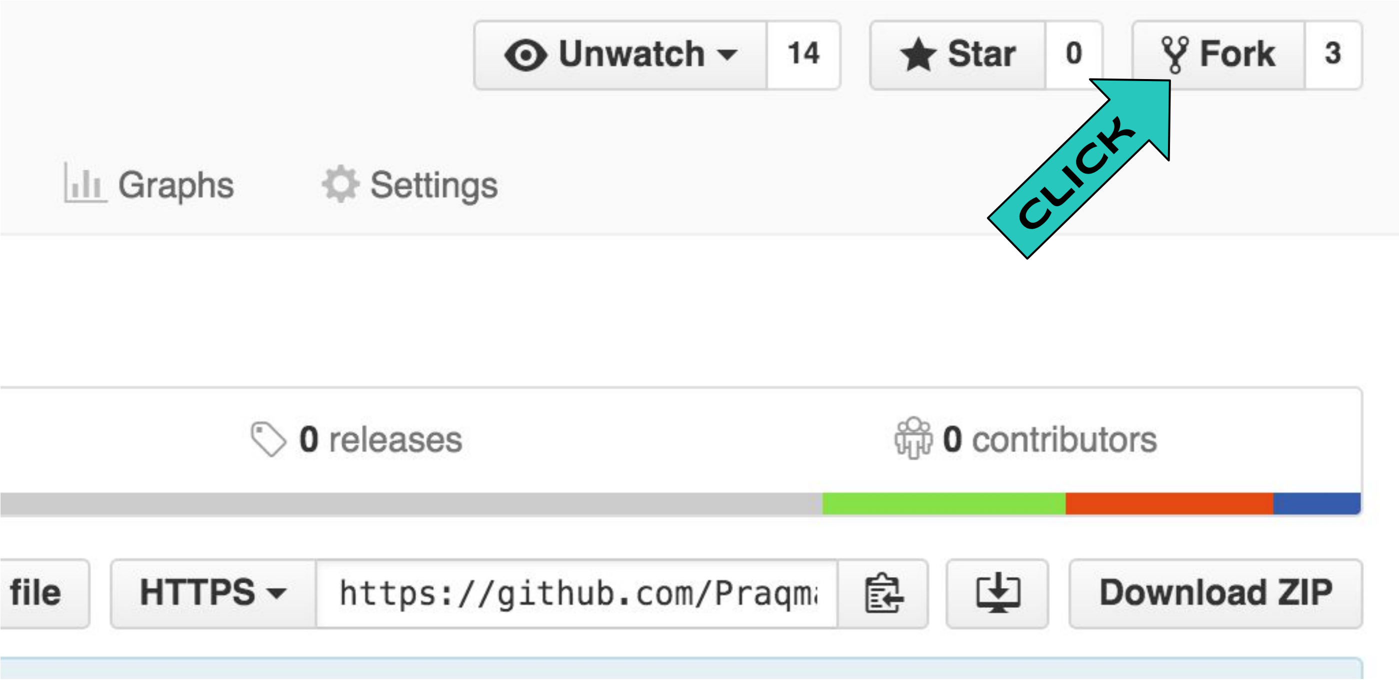


On-demand Jenkins Slaves

Getting your host ready

- Fork our github repository to your local github account. (https://github.com/praqma-training/code-infra)
- Clone your repo to your docker host:
  - $ git clone https://github.com/<YOUR USER>/code-infra.git

Fork Github repo



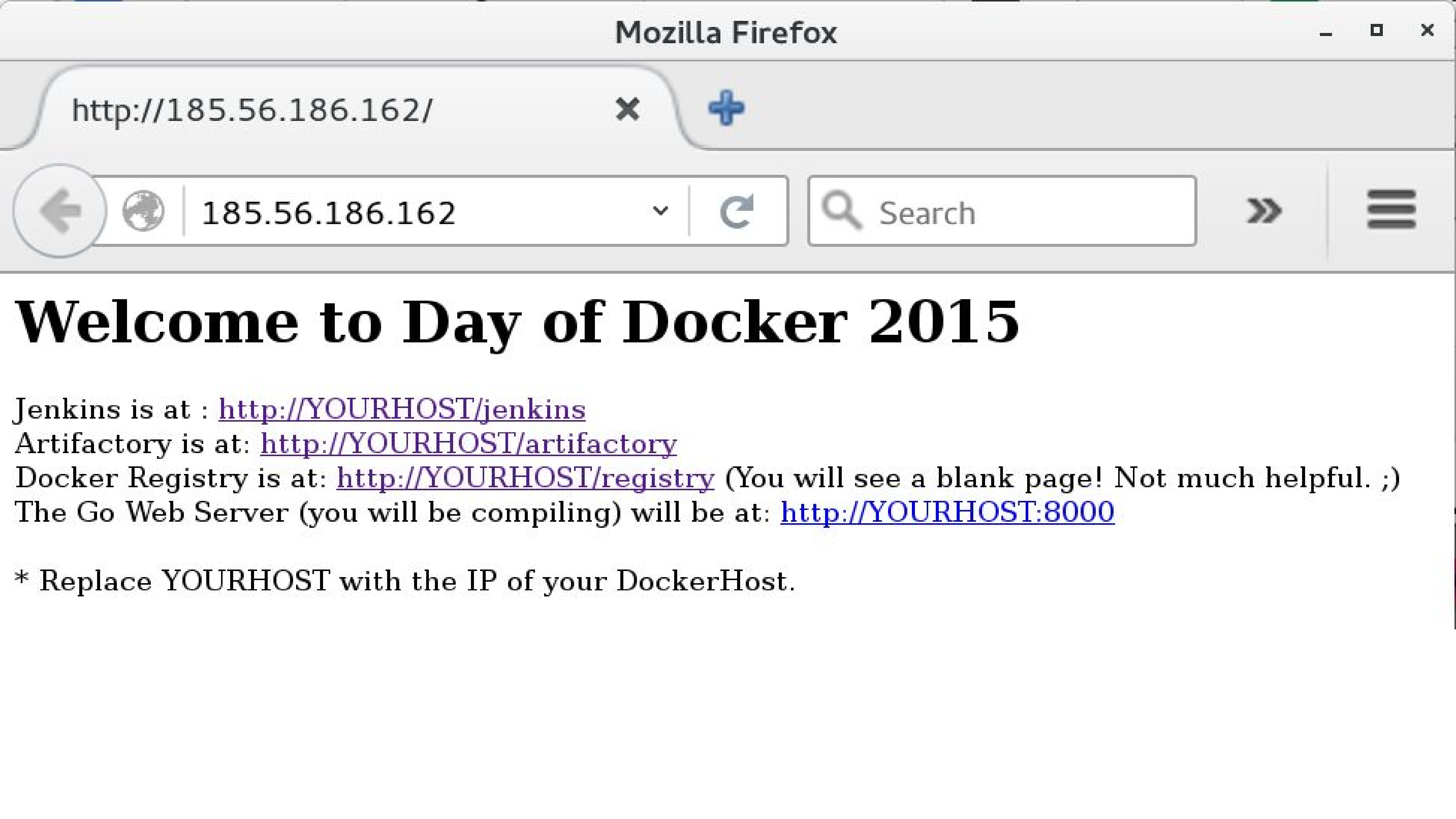
Getting your host ready

```
$ docker-machine create code --driver virtualbox
$ eval $(docker-machine env code)
```

Follow the instructions to add the directory structure

Getting your host ready

- And then get it up and running:
  - $ cd code-infra/containers
  - $ docker-compose build
  - $ docker-compose up



Welcome to Day of Docker 2015

Jenkins is at : http://YOURHOST/jenkins
Artifactory is at: http://YOURHOST/artifactory
Docker Registry is at: http://YOURHOST/registry (You will see a blank page! Not much helpful. ;)
The Go Web Server (you will be compiling) will be at: http://YOURHOST:8000

* Replace YOURHOST with the IP of your DockerHost.

5 min demo!

‘Gonah’ and the whale

The ‘Gonah’ (simplified) pipeline

Build

+ ‘Unit’ test

Functional test

+ Deploy to test

Release

+ Deploy to production

JobDSL

Jenkins pipelines *as code*

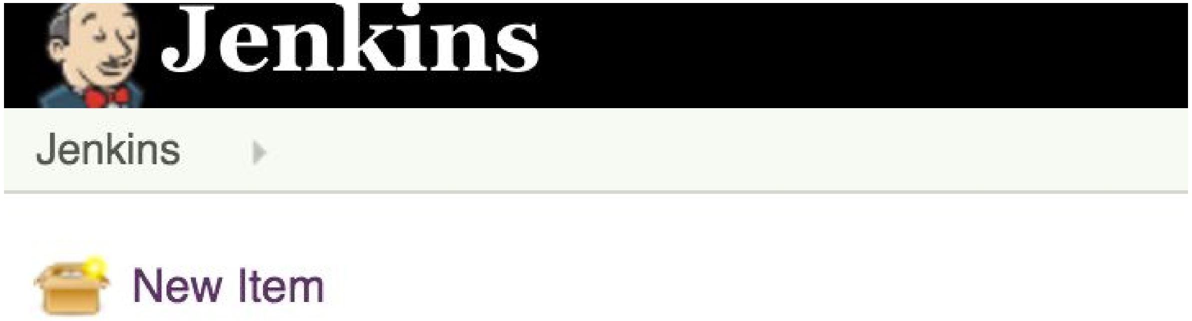
A groovy DSL for creating Jenkins jobs and views

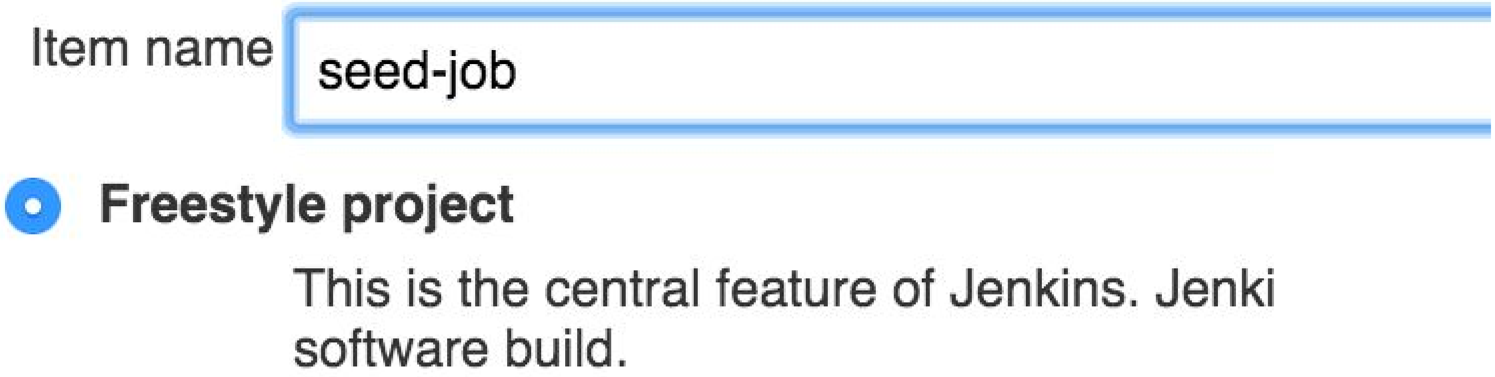
```
job('tag-version'){
 scm {
   git {
     remote {
       name('origin')
       url(
      'https://github.com/drbosse/dayofdocker15.git'
      )
     }
     branch('master')
   }
 }
 triggers {
   scm('* * * * *')
 }
 steps {
   shell('''echo "Hello Dockeristas"''')
 }
}
```

Exercise - generate the pipeline

- Step 1 - Create a seed job
- Step 2 - run the provided JobDSL
- Step 3 - *there is no step 3*
- Step 4 - *profit*

Step 1 - Create seed job



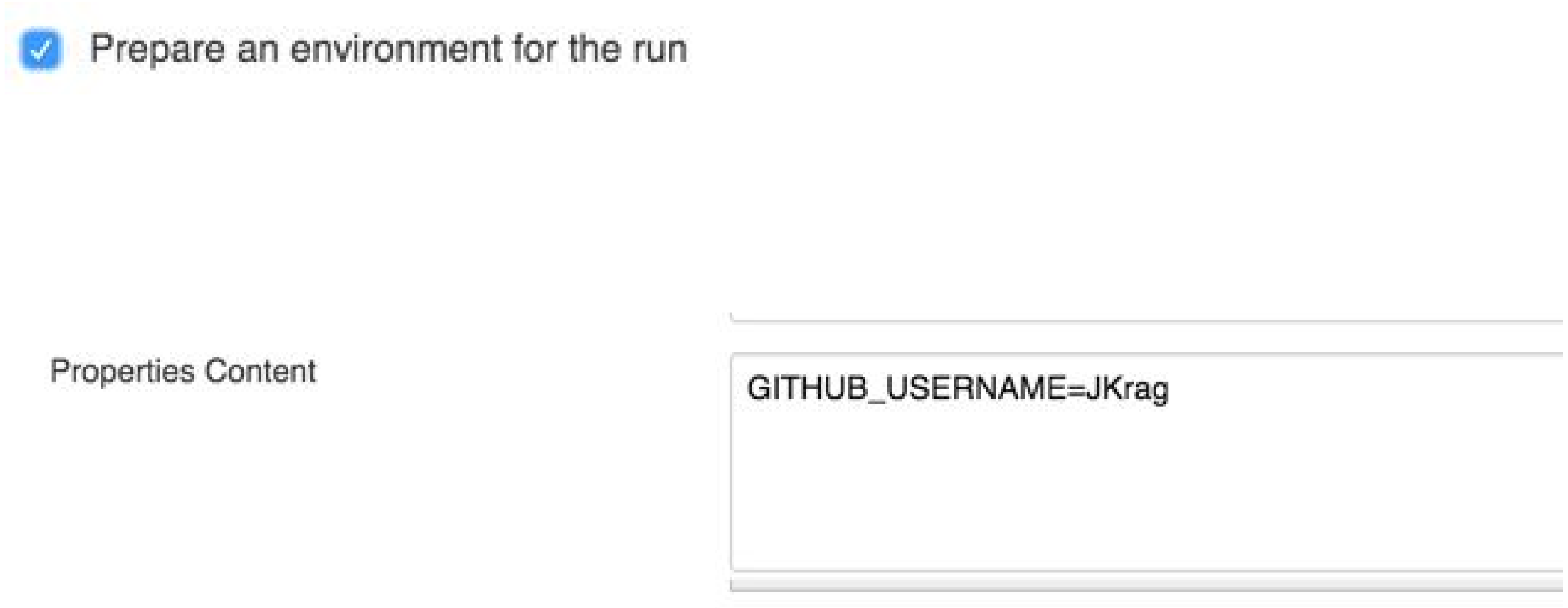


Item name: seed-job

● **Freestyle project**

This is the central feature of Jenkins. Jenki
software build.

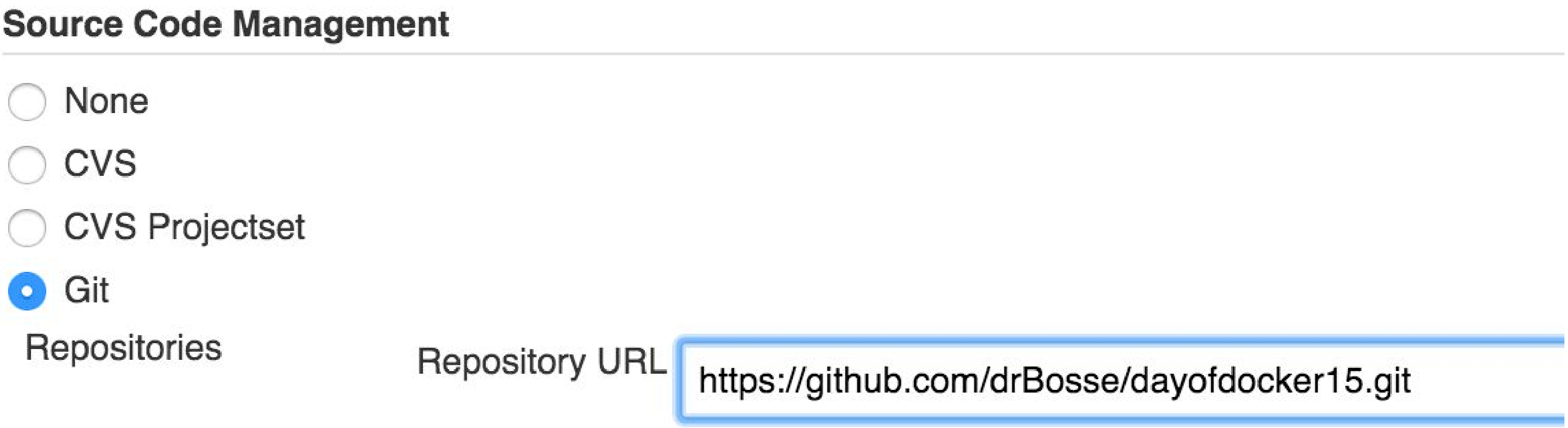
… add parameter for GITHUB_USERNAME



☑ Prepare an environment for the run

Properties Content

```
GITHUB_USERNAME=JKrag
```



Source Code Management

○ None

○ CVS

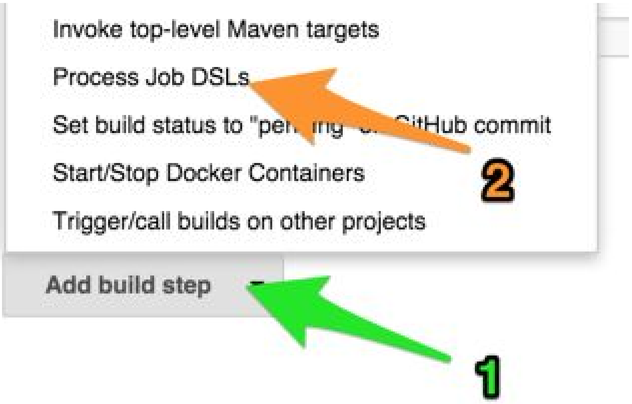
○ CVS Projectset

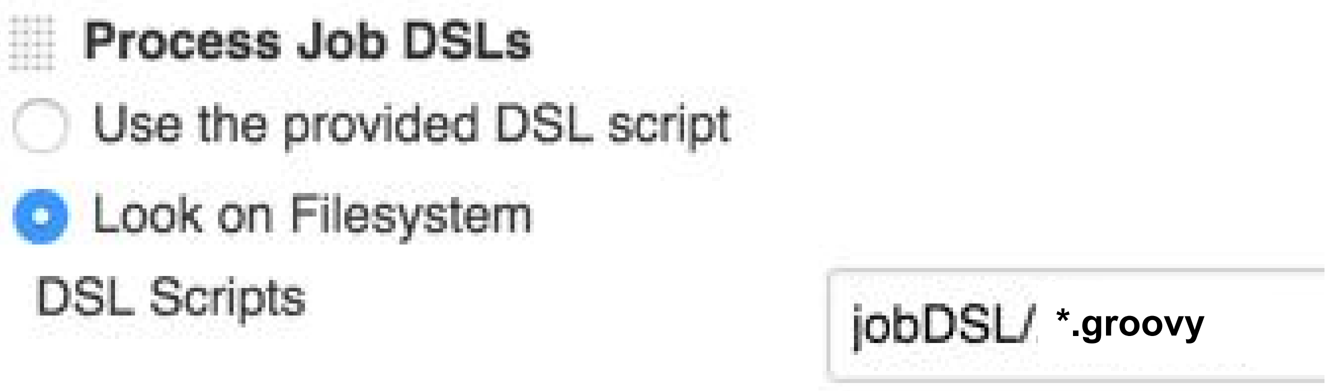
● Git

Repositories

Repository URL: https://github.com/drBosse/dayofdocker15.git

Then:


Invoke top-level Maven targets
Process Job DSLs
Set build status to "pending" on GitHub commit
Start/Stop Docker Containers
Trigger/call builds on other projects
2
Add build step
1



Process Job DSLs

○ Use the provided DSL script

● Look on Filesystem

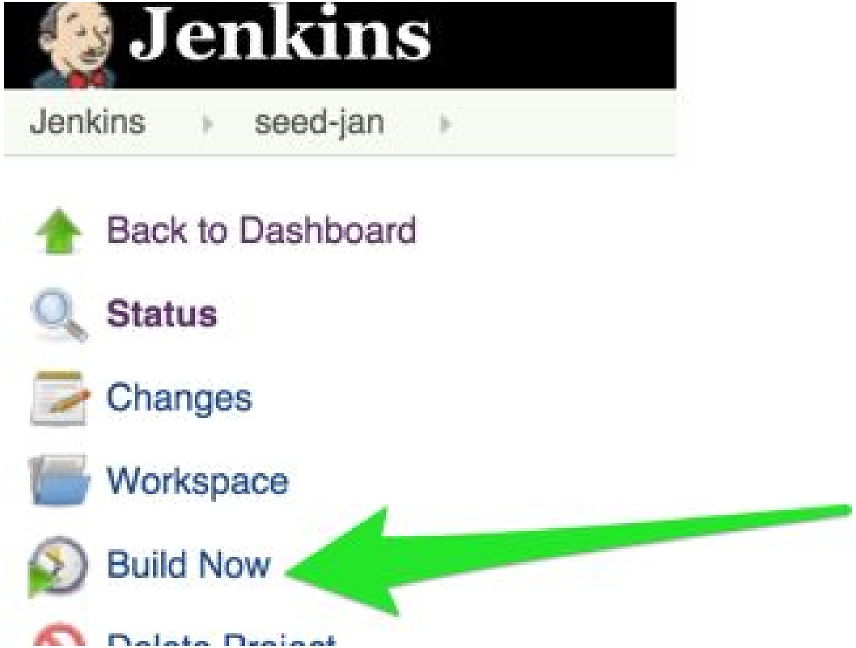
DSL Scripts: jobDSL/ *.groovy

```
jobDSL/*.groovy
```

Not quite so fast....

```
$ cd code-infra/containers/siege
$ docker build -t siege-engine .
$
```

Save and run



Jenkins
Jenkins
seed-jan
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project

Congrats

You should now have:

- 3 jobs:
  - server-build
  - server-test
  - server-release
- A ‘View’ (tab)
- A Build pipeline view

A note on versioning

The simplified story:

- Semantic versioning
- Controlled by developer
- In `version.txt` file
  - Pulled from repo in build phase
  - Passed through all the way to release phase
  - Used to tag release version of docker image

Build phase

Where we are going to build "the Docker way" and test our running web server

Build phase - simplified

```
docker build -t drbosse/http-app:snapshot .
```

- *build Gonah and tag with snapshot*

```
docker run -d --name testing-app -p 8001:8000 drbosse/http-app:snapshot
```

- *run Gonah on test port*

```
docker run --rm siege-engine -g http://<ip-of-http-app-container>:8000/
```

- *Use Dockerized **siege** to test that the server responds*

If all is OK, we trigger the test phase, and pass the image id.

NOTE: Look at the full version in your own “build-browser” job

Docker ONBUILD

The `ONBUILD` instruction adds to the image a *trigger* instruction to be executed at a later time, when the image is used as the base for another build.

Go-lang ONBUILD image

```
FROM golang:1.3

RUN mkdir -p /go/src/app
WORKDIR /go/src/app

# this will ideally be built by the ONBUILD below ;)
CMD ["go-wrapper", "run"]

ONBUILD COPY . /go/src/app
ONBUILD RUN go-wrapper download
ONBUILD RUN go-wrapper install
```

Building from ONBUILD

Dockerfile:

```
FROM golang:1.3-onbuild
```

Just build it...

That’s essentially it

Although we could also …

- Add `.dockerignore`
  - e.g. Dockerfile, README
- Add more to Dockerfile
  - maintainer
  - expose ports
  - CMD to run web server

The functional test

Where we spin up the Gonah server and check that it works

Testing phase

“Deploy to test”

```
docker run -d --name testing-app -p 8000:8000  $IMAGEID
```

- *Run test version on port 8000*

“Run functional test”

```
docker run --rm siege-engine http://<ip-of-http-app-container>:8000/
```

- *Load test with siege engine*
- *If availability is OK, then tag image as stable*

```
docker tag $IMAGEID drbosse/http-app:stable
```

- *and for fun, we plot some of the output from siege*
- *If everything is OK, we call trigger a release*

The release

Where we spin up the Gonah server and check that it works

Release phase

Tag with version nr. and 'latest'

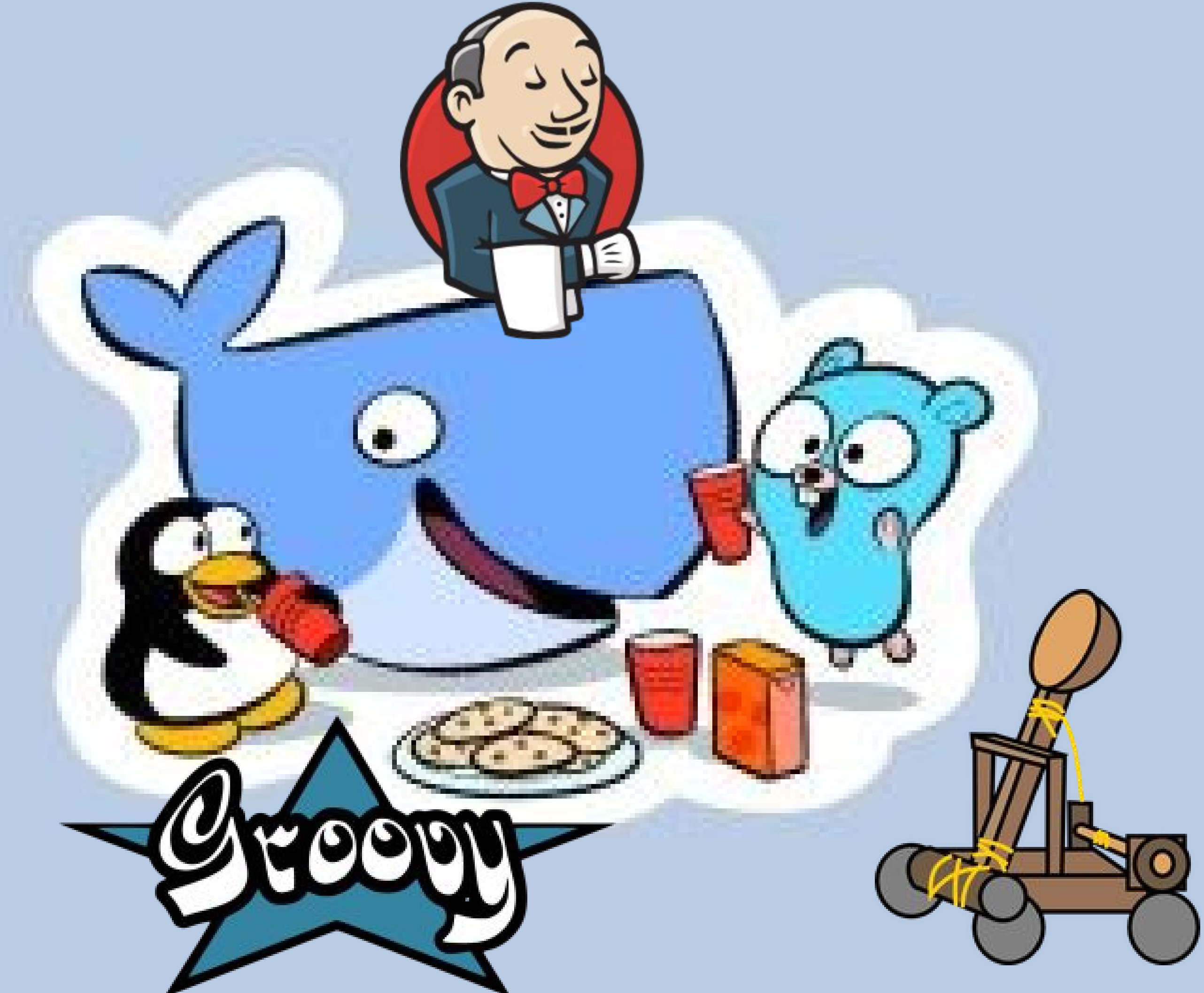
```
docker tag -f drbosse/http-app:stable drbosse/http-app:latest
docker tag -f drbosse/http-app:stable drbosse/http-app:$VERSION
```

Deploy to "production"

```
docker run -d --name deploy-app -p 81:8000 drbosse/http-app:latest
```

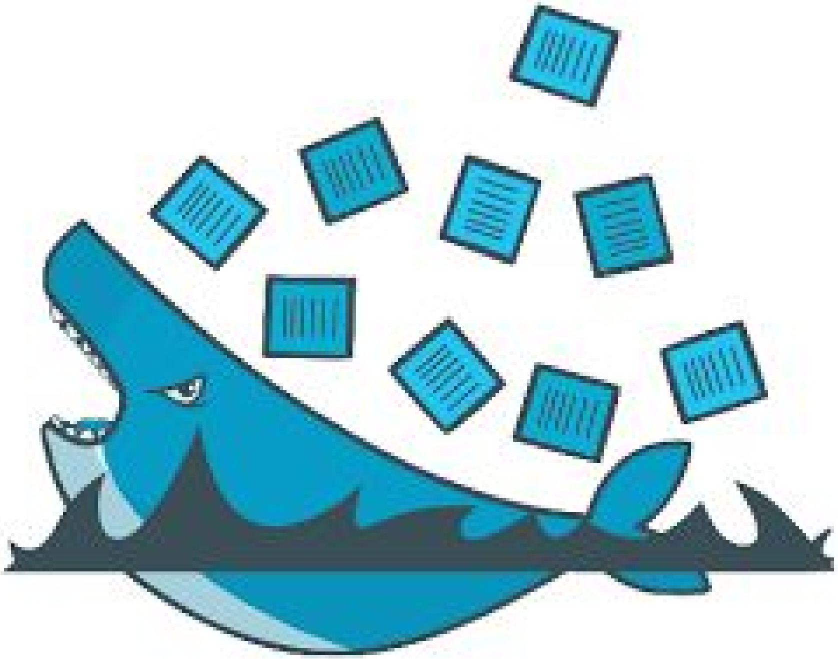
- *Run prod on port 81 to avoid conflict with existing Apache*

- ***If everything is OK, we are just happy***



Groovy

… else …



It’s whales all the way down...

Going “all in” with docker in your Continuous Delivery setup

Extra credit...

Choose one:

- Put your `gowebserver` behind a HAProxy and scale with interlock
- Add a SonarQube to the `code-infra` setup
- Change `code-infra` to use data containers
- Run your `code-infra` on swarm