PRAGMATIC DOCKER

A 15 MINUTES INTRODUCTION

Jens Fischer @jensfischerhh

WHAT IS DOCKER?

Virtualization on steroids!

- lightweight
- fast
- isolation
- predictability
- great ecosystem

WHERE TO GET IT?

LINUX (UBUNTU STYLE)

```
$ sudo apt-get update
$ sudo apt-get install docker.io
```

LINUX (YOLO STYLE)

```
$ curl -sSL https://get.docker.com/ | sh
```

MAC OS X

```
$ brew cask install virtualbox
$ brew install docker-machine
$ brew install docker
```

IS IT WORKING?

```
$ docker run -i -t --rm hello-world
```

Command	Meaning
`docker run`	create and run a new container
`-i`	run interactive
`-t`	allocate pseudo TTY
`--rm`	remove container after exiting
`hello-world`	docker image name

```
Unable to find image 'hello-world:latest' locally
Pulling repository docker.io/library/hello-world
af340544ed62: Download complete
535020c3e8ad: Download complete
Status: Downloaded newer image for hello-world:latest

Hello from Docker.
This message shows that your installation appears to be working correctl
```

IMPORTANT COMMANDS

Command	Description
`docker help`	show a list of all commands
`docker run <IMAGE>`	create and run a new container from <IMAGE>
`docker stop <CONTAINER>`	stop a running <CONTAINER>
`docker start <CONTAINER>`	restart a stopped <CONTAINER>
`docker logs <CONTAINER>`	show log output of <CONTAINER>
`docker ps`	list all running containers
`docker rm <CONTAINER>`	remove a <CONTAINER>
`docker images`	list all available images
`docker rmi <IMAGE>`	remove an <IMAGE>
`docker build <PATH>`	compile a new image named <PATH>

DOCKER LIFECYCLE

- edit `Dockerfile`
  - extend existing base image
  - add desired software
- create local image

  ```
  docker build -t epages/rnd-day .
  ```
- share local image in remote repository

  ```
  docker push epages/rnd-day
  ```
- download image from repository

  ```
  docker pull epages/rnd-day
  ```
- instantiate container from image and execute

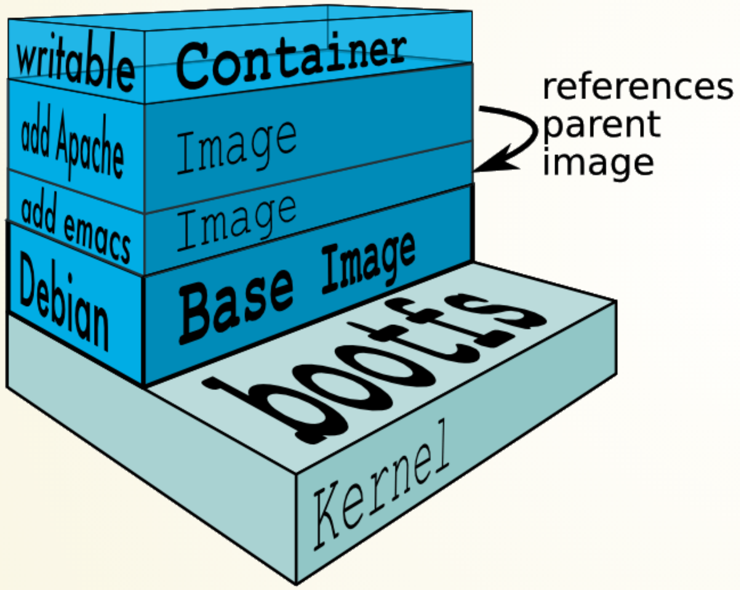
  ```
  docker run -d epages/rnd-day
  ```

DOCKERFILE

```
FROM debian
RUN apt-get install -y apache2
ADD httpd.conf /etc/apache2/httpd.conf
VOLUME ["/var/www", "/var/log/apache2"]
EXPOSE 80 443
ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

Command	Description
`FROM`	extend from base image
`RUN`	execute command and store results
`ADD`	include local content
`VOLUME`	create mount points
`EXPOSE`	listen on network ports
`ENTRYPOINT`	Command for executing container

LAYERED FILESYSTEM



- read-write layer on top
- read-only layer below
- `ADD` creates a new layer
- `ADD` gets cached
- transfer only layer diffs

NETWORKING

- container has internal network
- manually map exposed ports to host

```
$ docker run -p <HOST_PORT>:<CONTAINER_PORT>
```

- dynamically map all exposed ports to host

```
$ docker run -P
```

- link containers using names

```
$ docker run -d --name db my/database
$ docker run -d -P --link db:db my/webapp
```

PERSISTENT DATA

- persist changes in running container to new image

```
$ docker commit <CONTAINER>
```

- create named data volume container

```
$ docker run --name mysql-data -v /var/lib/mysql busybox tr
```

- use named data volume container

```
$ docker run -d --volumes-from mysql-data -p 3306:3306 mysq
```

BEST PRACTICES

- only one process per container
- log to STDOUT
- don't fix a running container
  - delete container
  - fix `Dockerfile`
  - create new container
- minimize layers
  - understand caching
  - use multi-line arguments for `RUN`
- don't `apt-get upgrade`

ECOSYSTEM

- CoreOS: container OS
- Project Atomic: container OS
- Docker Machine: container provisioning
- Docker Compose: container orchestration
- Kubernetes: container orchestration
- Docker Swarm: container clustering
- Apache Mesos: container clustering
- ClusterHQ: container storage
- Weave: container networking
- Deis: container PaaS

DOCKER
ALL THE THINGS

STEP #1: WORDPRESS ON MYSQL

1. use official MySQL image

```
docker run \
  --detach=true \
  --name=db \
  --env="MYSQL_ROOT_PASSWORD=root"
  --env="MYSQL_DATABASE=wordpress"
  mysql
```

2. use official Wordpress image

```
docker run \
  --detach=true \
  --name=web \
  --link db:mysql \
  --publish 80:80 \
  wordpress
```

STEP #2: TURBOPRESS CUSTOMIZATION

1. use official MySQL image
2. prepare `Dockerfile`
3. build customization

```
docker build \
  --no-cache=true \
  -t "rnd/turbopress:latest" \
  .
```

4. use local image

```
docker run \
  --detach=true \
  --name=web \
  --link db:mysql \
  --publish 80:80 \
  rnd/turbopress
```

STEP #3: ORCHESTRATED CONTAINERS

1. prepare `docker-compose.yml`
2. build customization

```
docker-compose build --no-cache
```

3. run orchestrated containers

```
docker-compose up -d
```

STEP #4: SWITCH TO MARIADB

1. prepare `docker-compose.yml`
2. build customization

```
docker-compose build --no-cache
```

3. run orchestrated containers

```
docker-compose up -d
```

4. prepare `manage.yml`
5. backup/restore data

```
docker-compose -f manage.yml run --rm bac
docker-compose -f manage.yml run --rm res
```

STEP #5: LOAD-BALANCE SCALED CONTAINERS

1. prepare `docker-compose.yml`
2. build customization

   ```
   docker-compose build --no-cache
   ```

3. run single web container

   ```
   docker-compose up -d web
   ```

4. scale web container

   ```
   docker-compose scale web=10
   ```

5. run load-balancer

   ```
   docker-compose up -d lb
   ```

See original blog post by @eyenx

STEP #6: CACHE LOAD-BALANCED RESPONSES

1. prepare `docker-compose.yml`
2. build customization

```
docker-compose build --no-cache
```

3. run single web container

```
docker-compose up -d web
```

4. scale web container

```
docker-compose scale web=10
```

5. run cache in front of load-balancer

```
docker-compose up -d cache
```