Introduction to Docker

@taylodl
software architect, enterprise architecture


MAERSK
TOMAS
MAERSK
MAERSK
MAERSK
MAERSK
45
SWIFT
FedEx

Java
JEE

RAILS

php

node
JS

MySQL

django

Cassandra

Apache
Derby

TOMCAT

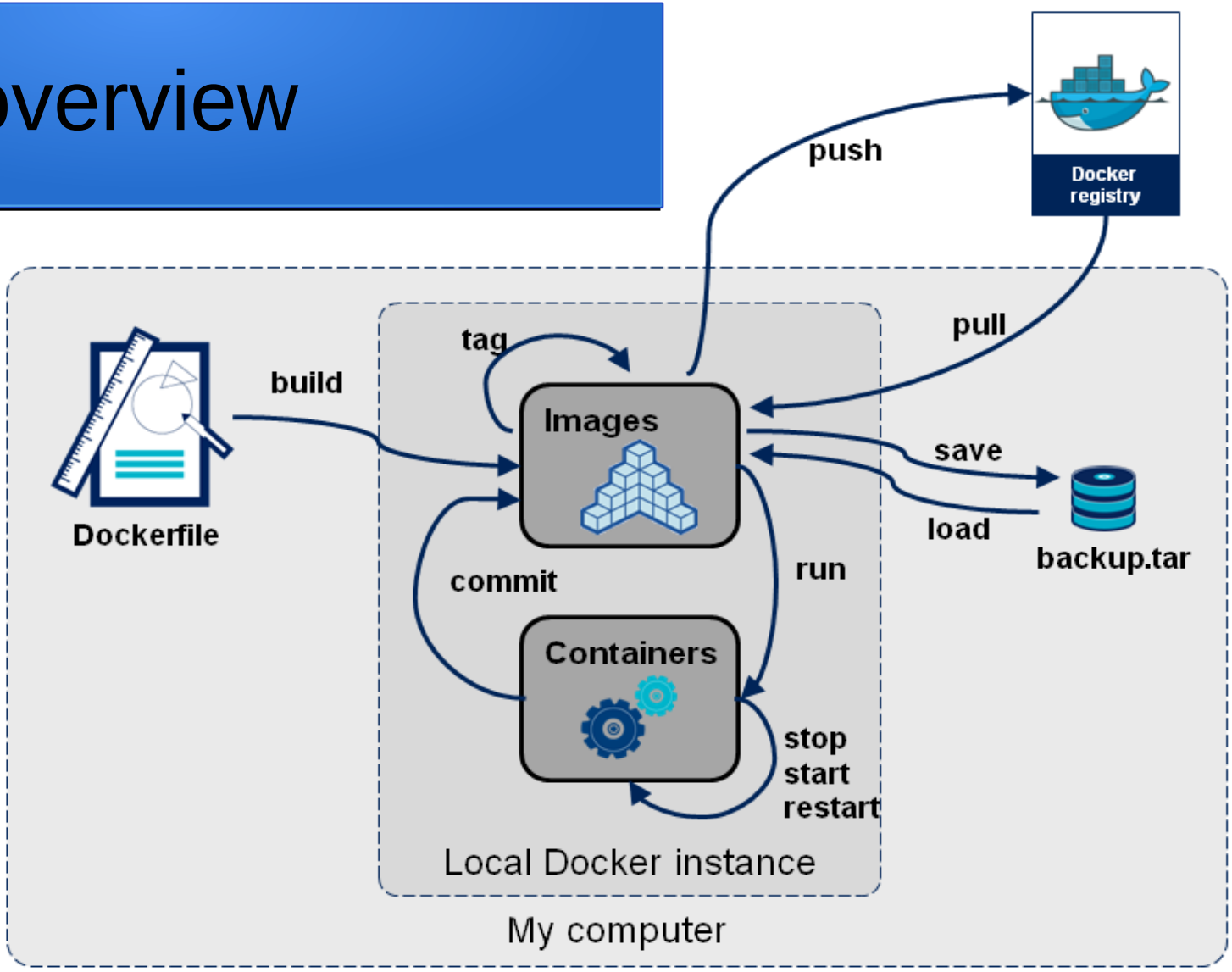
ORACLE

NGINX


amazon
web services
Microsoft Azure
rackspace
the open cloud company
Google
app engine
bluelock
CLOUDFORMS
by Red Hat Cloud
dotCloud
vmware
Joyent
verizon
CenturyLink
IBM Cloud
ORACLE
CLOUD
terremark
CloudSigma
Engine Yard
lunacloud
Jelastic

Is the solution!

Docker overview



Running busybox

```
docker run busybox cat /etc/os-release
```

What are images?

Comprised of:

1. OS services and shared libraries
2. Application runtime environments
3. Environment variable settings
4. IP port settings
5. Your application

Images & containers

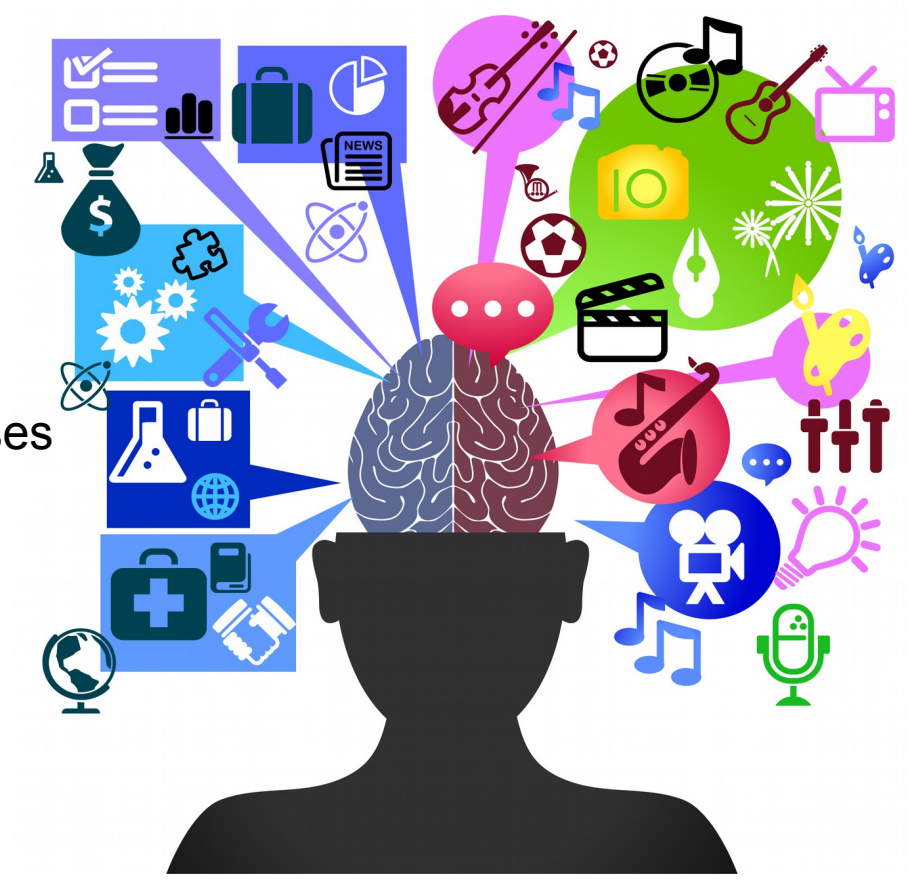
Images → Classes
Containers → Objects

- Images are created using `docker build`
- Containers are instantiated by running images via `docker run`
- Images are stored in repositories, much like the source code for classes
- Images can inherit from one another
- Containers are listed using `docker ps`
- Local images are listed using `docker images`

Creating Docker Images

Left Brain Strategy

- use dockerfile
- repeatable process
- source control mgmt
- preferred by enterprises



Right Brain Strategy

- use existing image
- apt-get
- export
- commit container
- exploratory work

docker

Dockerfile

Common Commands

FROM	name of base image
ADD(*)	add files (or tarballs) to the image
ENV(*)	add an environment variable
EXPOSE(*)	IP port to be exposed
CMD(+)	command to be run when not specified via docker run

Less Common Commands

RUN(*)	run a command, for example apt-get
WORKDIR(*)	working directory for RUN command
USER(*)	user for RUN or CMD
COPY(*)	like add but less powerful, can't handle URLs or tarballs
ENTRYPOINT(+)	defaults to /bin/sh -c
VOLUME(*)	specify volume mount point in image

Building the image

docker build -t <image name:version> .

Right brained way

Essentially is manual processing of Dockerfile:

- Clear extraneous containers `docker rm `docker ps -aqf exited=0``
- Run image `docker run -i -t <image name:version> /bin/bash`
- You'll be running bash as root inside a newly-created container
- Make changes
  - `mkdir, groupadd, useradd, usermod, chown, chgrp, chmod, apt-get, export, wget`
- Exit
- `docker commit `docker ps -aqf exited=0` [image name:version]`

IP port forwarding

taylodl/wls-mydomain image exposes port 7001, the WebLogic admin console port□□

```
docker run -p 7001:7001 taylodl/wls-mydomain
docker run -p 8001:7001 taylodl/wls-mydomain
docker run -p 9001:7001 taylodl/wls-mydomain
```

We can run multiple instances of WebLogic on our machine without colliding IP ports and without having to reconfigure each instance of WebLogic.

Volume mapping

```
docker run
    -p 7001:7001
    -v /home/developer/docker/wls-autodeploy
       :/appl/oracle/middleware/wls/wls12120/user_projects/domains/
        mydomain/autodeploy
taylodl/wls-mydomain
```

We can make the contents of local directories available in the container. In this case we're making a directory containing JEE EAR files available so they'll be deployed upon starting the taylodl/wls-mydomain image.

More to explore

Topics we haven't covered:

- Coordinating multiple Docker containers
- Private repositories
- Debugging images

Questions