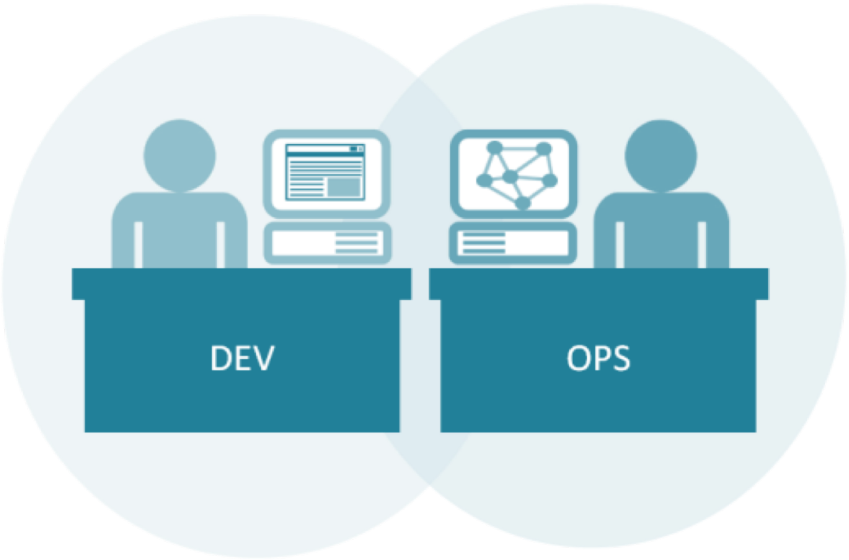


Microservices Engineering Boot Camp

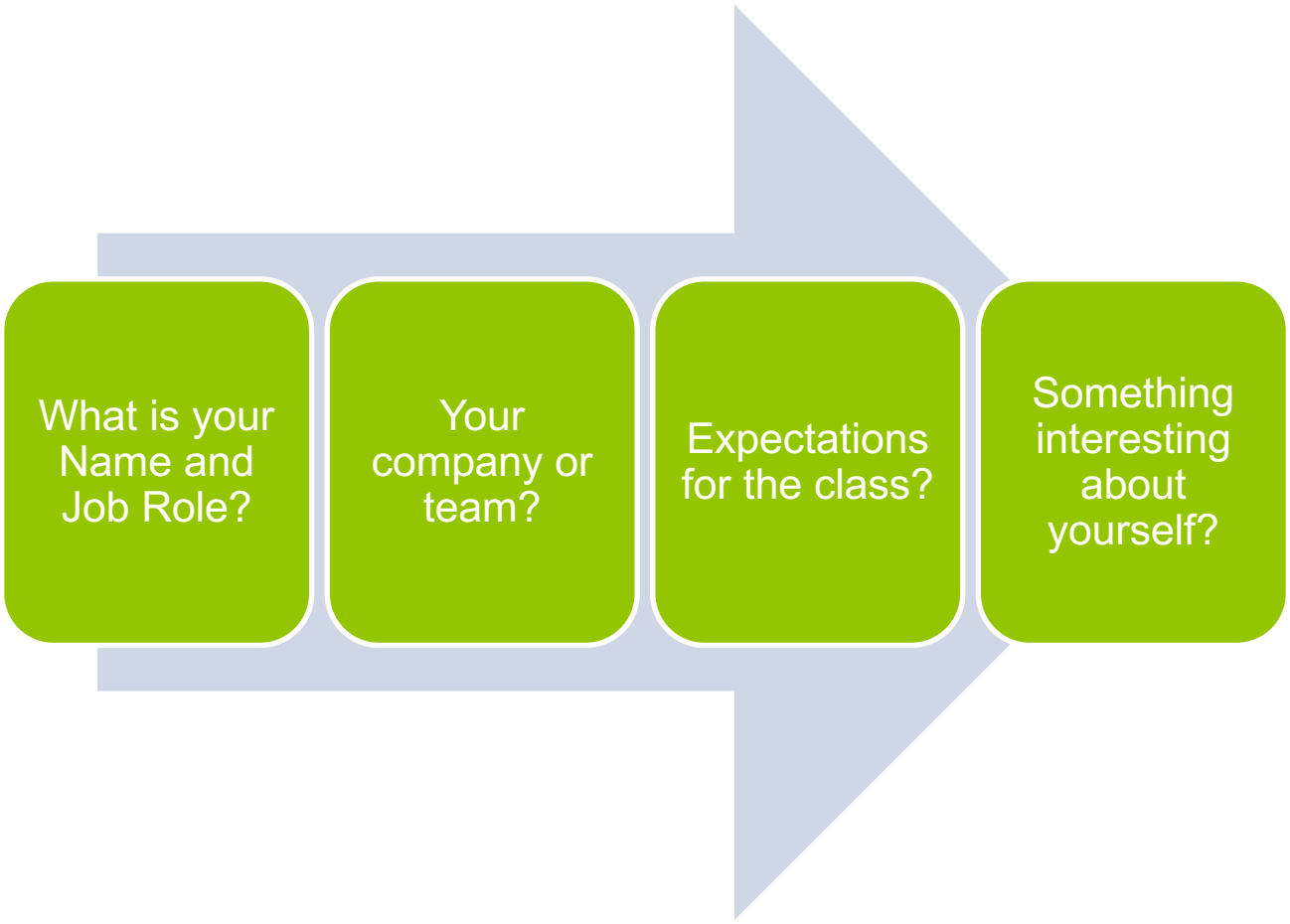
Welcome!

- Logistics (breaks, facilities, lunch, etc.)
- Rules of Engagement
- Introductions
- Let's Get Started!

Who is your instructor today?

A little about me…

Introductions



Core Objectives

- Understand how to adopt, plan or improve your transition to Microservices
- Navigate different tools for enabling Microservices and how to use them
- Map technical practices to the business strategy behind Microservices
- Get hands-on practice with tools which enable core Microservices architecture
- Build more DevOps practices through Microservices adoption
- Apply Microservices use cases to continuous integration, delivery and testing
- Understand how to refactor monolithic systems into more modular, component-based systems

What to expect from this class

- Flexibility
- Conversations
- Literacy and awareness on many of the principles, tools, and practices **surrounding Microservices culture, architecture and implementation.**
- An effort to focus on your own situations and challenges so you can act on what you learn.

How the course has been organized

Part I

- **Overview of Docker**
  - In this part, we will learn about Docker; practice Docker commands, build docker images and use docker-compose to run multiple containers at once

Part II

- **Microservices Concepts**
  - In this part, we will discuss about how Microservices helps in being more DevOps and important patterns. This will be a theoretical section and involves no hands-on

Part III

- **Dive Into Kubernetes**
  - In this part, we will learn about container orchestration tool (Kubernetes), to enable us running containers in multi-tenant and production environment

Part 1:
Docker Overview

Docker Basics

Docker Image

- The basis of a Docker container

Docker Container

- The standard unit in which the application service resides

Docker Engine

- Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider

Docker Registry

- On-premises registry for image storing and collaboration

Containers as a Solution

Container

- Packages up software binaries and dependencies
- Isolates software from each other
- Container is a standard format
- Easily portable across environment
- Allows ecosystem to develop around its standard

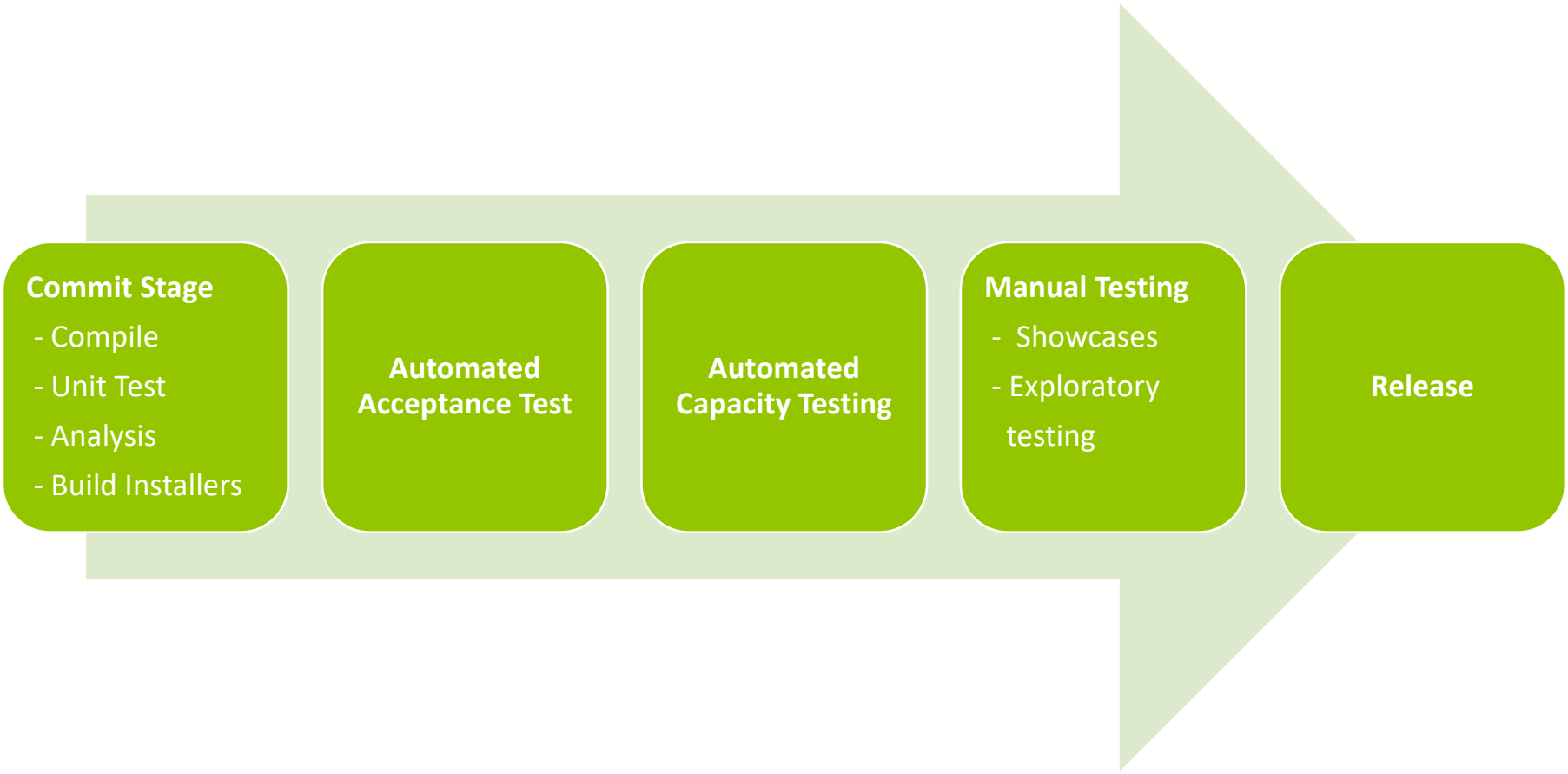
Developer Benefits

- Build once...(finally) run anywhere
  - A clean, safe, hygienic and portable runtime environment for your app.
  - No worries about missing dependencies, packages and other pain points during subsequent deployments.
  - Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for each app without worrying
  - Automate testing, integration, packaging...anything you can script
  - Reduce/eliminate concerns about compatibility on different platforms, either your own or your customers.
  - Cheap, zero-penalty containers to deploy services? A VM without the overhead of a VM? Instant replay and reset of image snapshots? That's the power of Docker

Operational Benefits

- Configure once...run anything
  - Make the entire lifecycle more efficient, consistent, and repeatable
  - Increase the quality of code produced by developers.
  - Eliminate inconsistencies between development, test, production, and customer environments
  - Support segregation of duties
  - Significantly improves the speed and reliability of continuous deployment and continuous integration systems
  - Because the containers are so lightweight, address significant performance, costs, deployment, and portability issues normally associated with VM

Where Containers are used?
- For automated testing & CICD



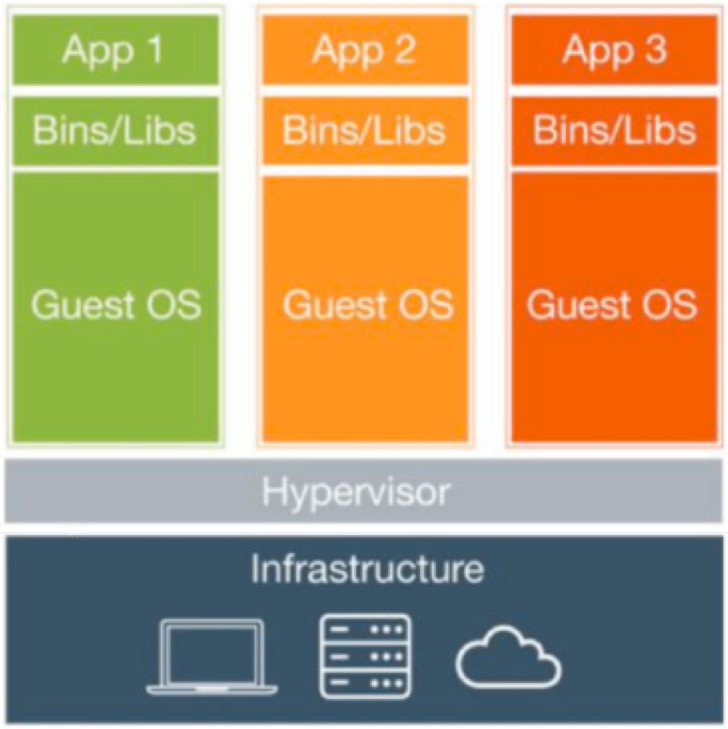
Where Containers are used?

- DevOps (Mostly in all types of Automation)
- Automated testing (Unit, acceptance and stress testing)
- CICD (if not tested on production-like environment, CICD will be risky)
- Microservices
- Less risky deployment architectures
- Chef kitchen test
- Not only used for testing but is used in production (along with container orchestration) in B2C and B2B organizations

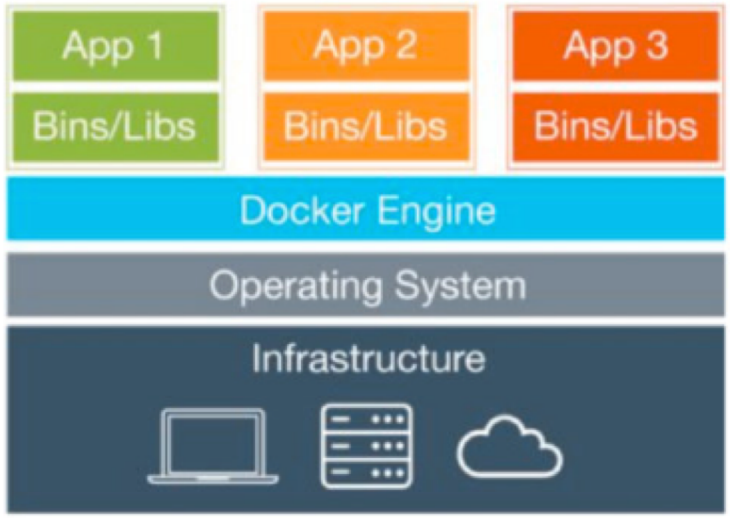
Advantage of using Containers

- Developers testing their code in Prod environments
- Lesser hand-offs and drastically lesser waiting time (to get new environment setup)
  - Reduce technical debt
- Overall system resilience - Service failure does not kill the application & may be invisible to users
- Scalability
  - Seamless replication
- High Availability
- Less risky patterns for rolling updates available
- Prevent server sprawl and Jenga infrastructure

VMs vs. Containers



Virtual Machines



Containers

VMs vs. Containers

Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and therefore more secure (maybe)	Process-level isolation and therefore less secure (maybe)

Installing Docker

- Docker cont**ainer** engine is built from and on top of the Linux kernel, so it needs Linux to run natively*
  - **Docker is increasingly being packaged with newer Linux distributions**
  - Docker is also **available o**n MacOS and Windows through the use of lightweight Linux VMs **- HyperV-based on Windows, HyperKit-based on MacOS**

*** Native Windows containers can also be used with Docker for Windows if you have very specific versions of Windows.**

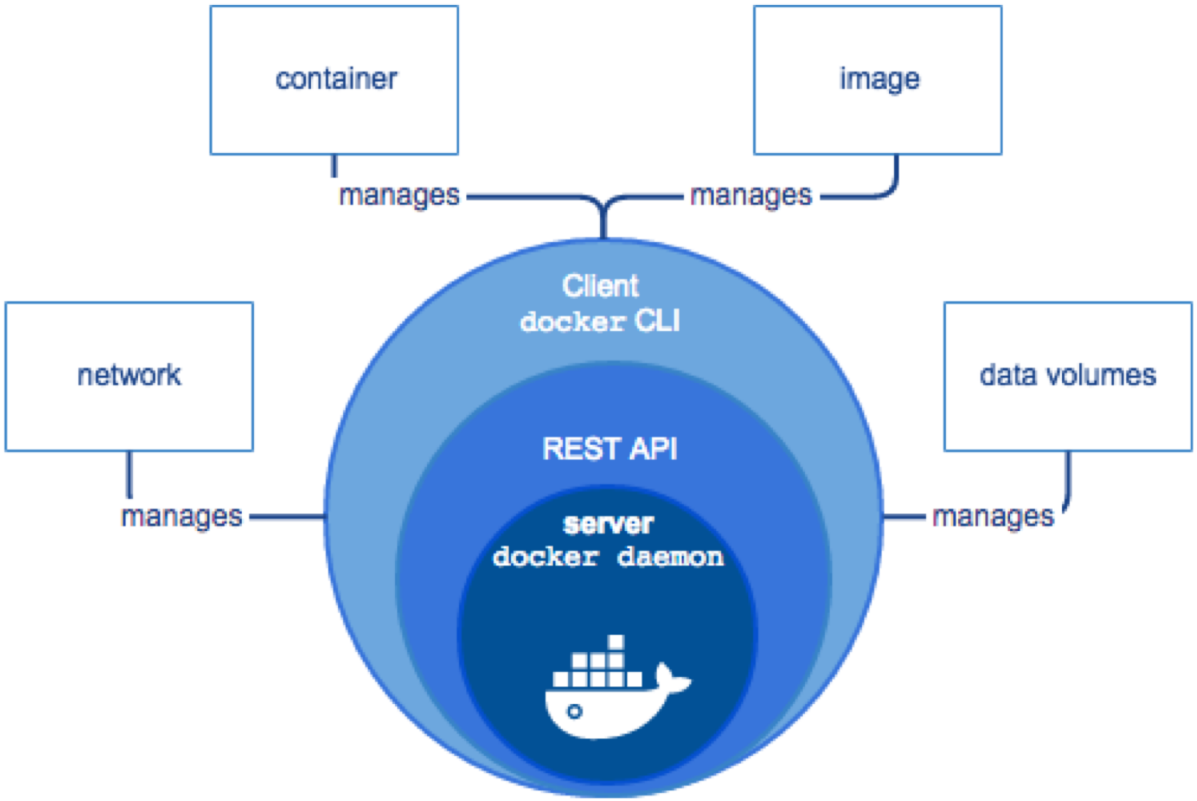
Installing Docker

```
$ sudo apt-get update

$ sudo apt-get -y install apt-transport-https ca-certificates curl

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

$ sudo add-apt-repository \
     "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
     $(lsb_release -cs) \
     stable"

$ sudo apt-get update

$ sudo apt-get -y install docker-ce

$ sudo docker run hello-world
```

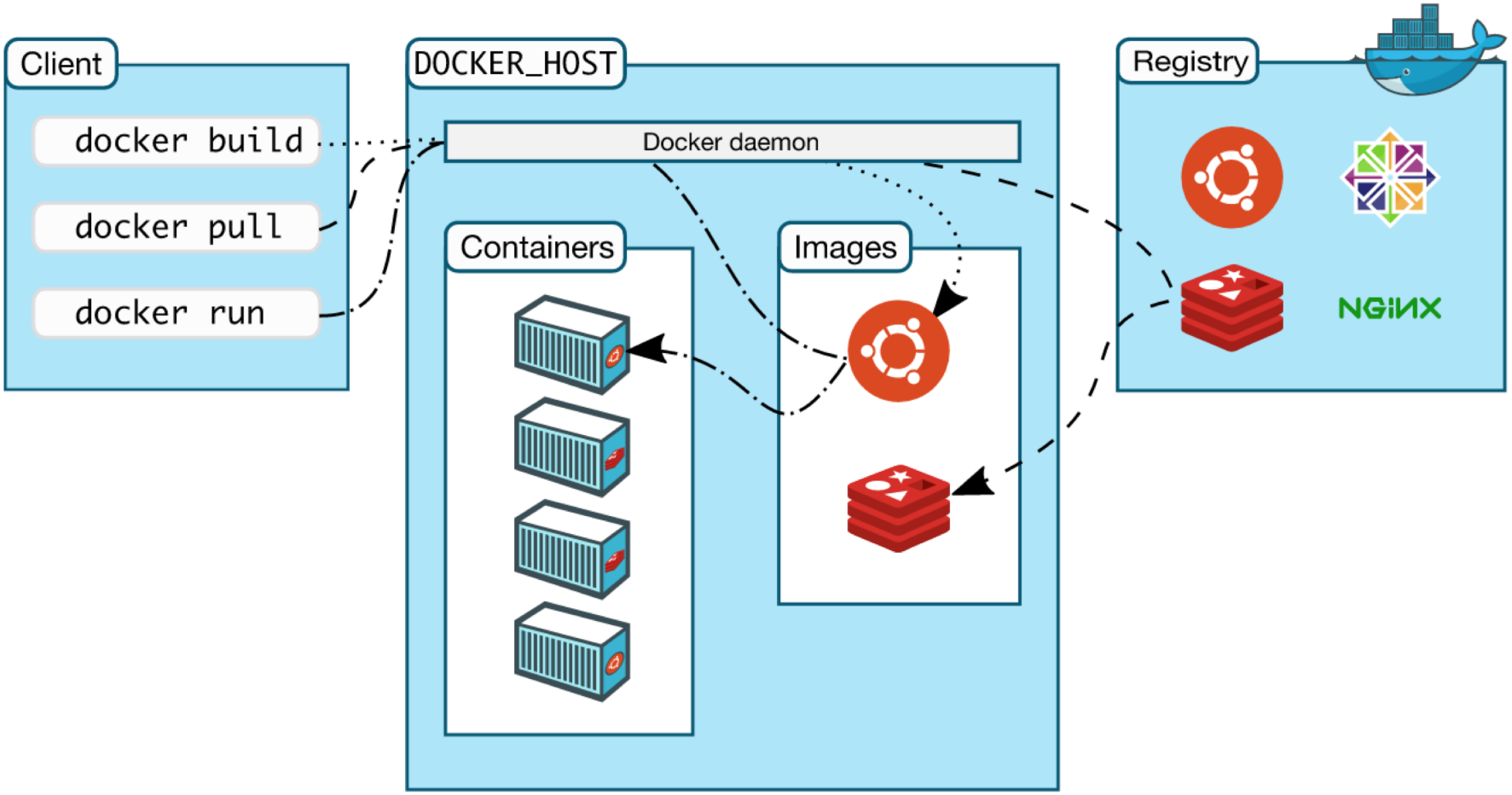
Installing Docker Script

- It's good to install Docker the "hard" way at least once so you **understand** what is involved.
  - There is a Docker-mai**ntained** community script that will install the latest release on your Linux machine (most popular flavors) available at https://get.docker.com/
  - Run 'curl -sSL https://get.docker.com/ | sh' in your terminal

What is ‘Docker Engine’?



How is Docker Architected?



Docker Webapp & Hello World

CLASSROOM WORK

20 minutes

Exercise 2.1 in Docker Labs
https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

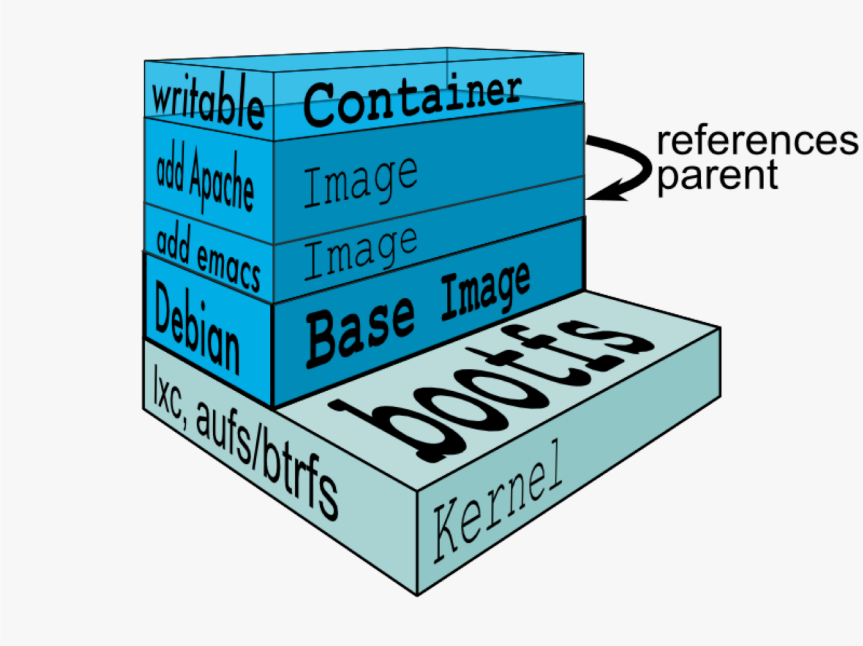
Do only exercise 2.1

DOCKER IMAGES

Docker Union File System

AuFS (AnotherUnionFS) is a multi-layered filesystem that implements union mount

- allows several filesystems or directories to be simultaneously mounted and visible through a single mount point
- appears to be one filesystem to the end user



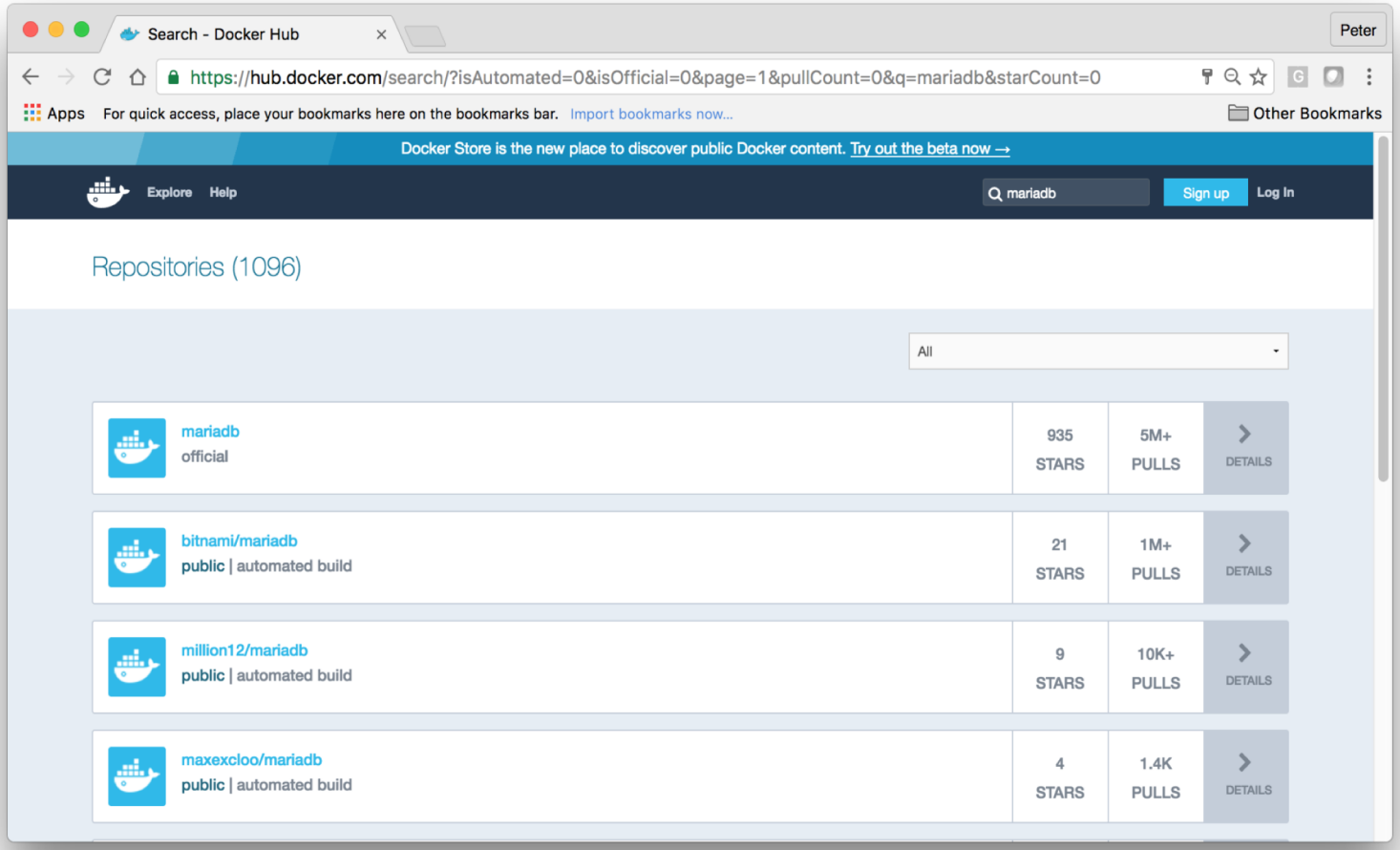
Example Docker Layers

Docker Layer – Each Docker image is composed of a series of layers. Docker uses **Union File Systems** to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

Docker Hub

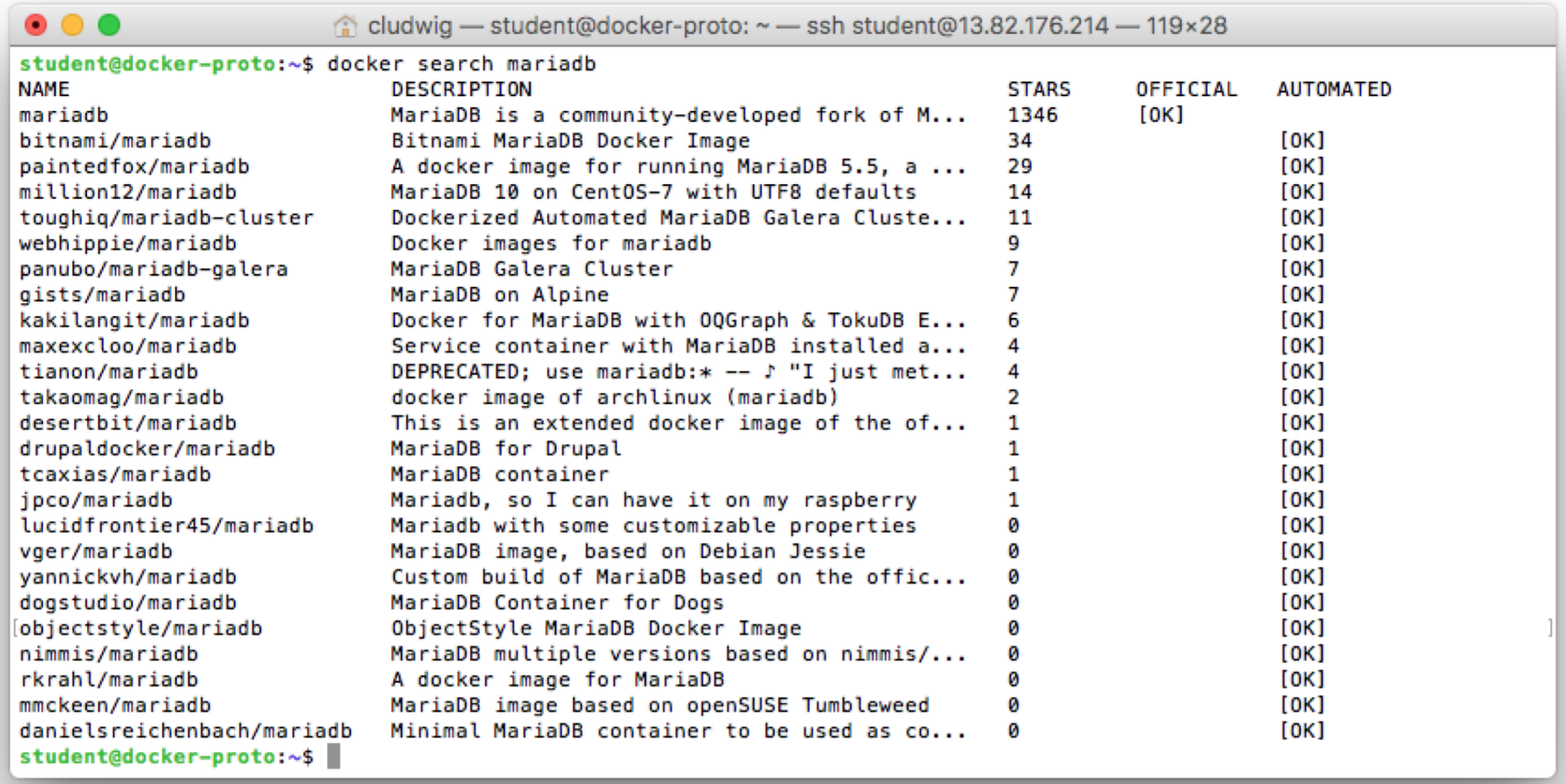
- Docker's first-party cloud-based registry
  - Hosts a broad repository of Docker images
  - Contains nearly any popular open source technology including MariaDB, Jenkins, Cloudera, etc
  - Contains 'Official images' from vendors such as Canonical, Oracle, Red Hat, etc
  - Provides one free private Docker repo, more for a paid subscription

Docker Hub


Search - Docker Hub
Peter
https://hub.docker.com/search/?isAutomated=0&isOfficial=0&page=1&pullCount=0&q=mariadb&starCount=0
Apps
For quick access, place your bookmarks here on the bookmarks bar.
Import bookmarks now...
Other Bookmarks
Docker Store is the new place to discover public Docker content. Try out the beta now →
Explore
Help
mariadb
Sign up
Log In
Repositories (1096)
All
mariadb
official
935 STARS
5M+ PULLS
DETAILS
bitnami/mariadb
public | automated build
21 STARS
1M+ PULLS
DETAILS
million12/mariadb
public | automated build
9 STARS
10K+ PULLS
DETAILS
maxexcloo/mariadb
public | automated build
4 STARS
1.4K PULLS
DETAILS

$ Few Basic Commands

$ docker search [searchterm]



$ docker image commands

```
ubuntu@ip-10-0-0-117:~/code/multistage$ docker image --help

Usage:  docker image COMMAND

Manage images

Options:
      --help   Print usage

Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE

Run 'docker image COMMAND --help' for more information on a command.
```

$ docker container commands

```
ubuntu@ip-10-0-0-117:~/code/multistage$ docker container --help

Usage:  docker container COMMAND

Manage containers

Options:
      --help   Print usage

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
```

$ docker image pull

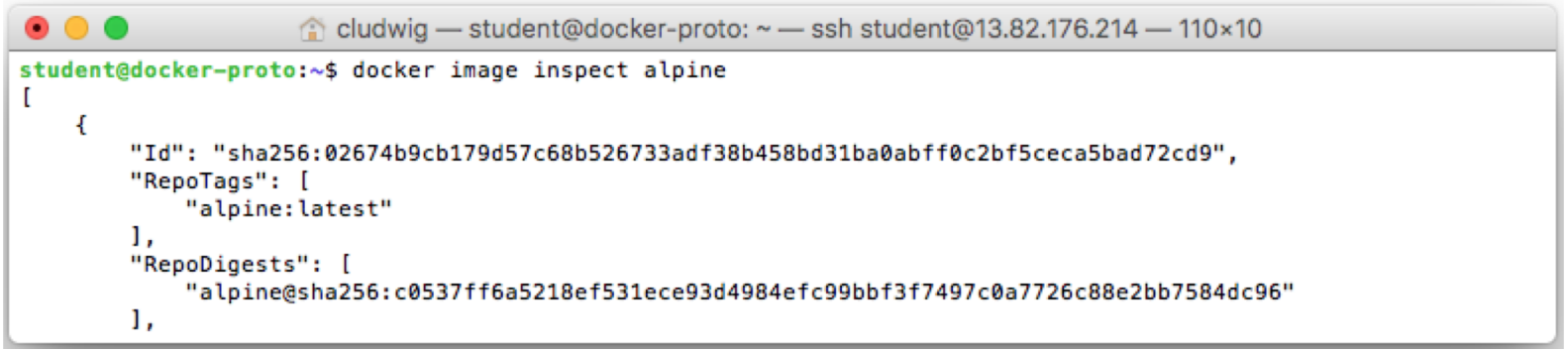
```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 119×6
[student@docker-proto:~$ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
cfc728c1c558: Pull complete
Digest: sha256:c0537ff6a5218ef531ece93d4984efc99bbf3f7497c0a7726c88e2bb7584dc96
Status: Downloaded newer image for alpine:latest
```

Best Practice: Choose the smallest base images possible

Debian – 123 MB
Alpine – 5 MB

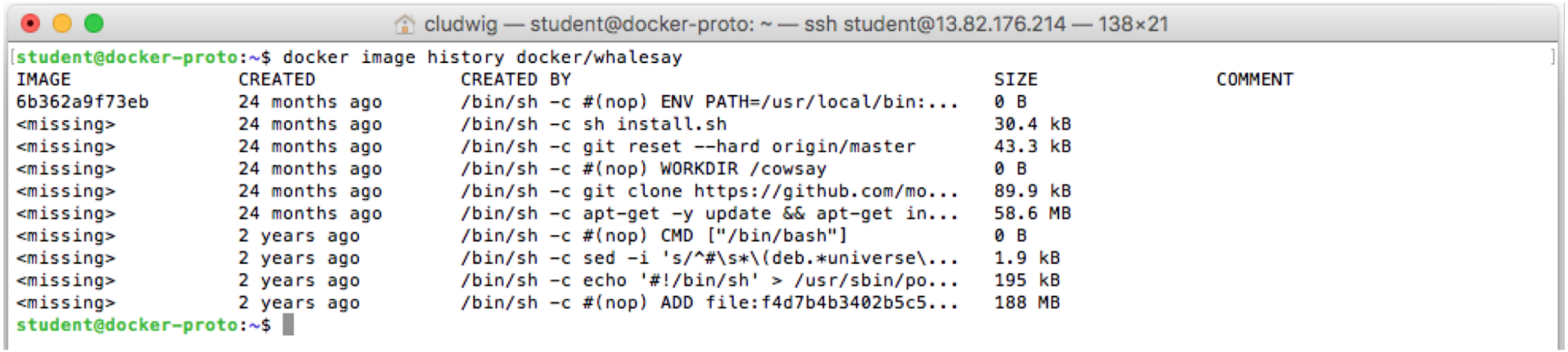
```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 119×7
[student@docker-proto:~$ docker image list
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
alpine                      latest              02674b9cb179        11 days ago         3.99 MB
hello-world                 latest              48b5124b2768        4 months ago        1.84 kB
dockersamples/static-site   latest              f589ccde7957        14 months ago       191 MB
docker/whalesay             latest              6b362a9f73eb        24 months ago       247 MB
student@docker-proto:~$
```

$ docker inspect [image|container]



- **ID** The unique identifier for the container
- **State** This stanza has various status flags and the process id for the container. Using the ExitCode from this State element a graceful shutdown or recovery process could be initiated. The following format will return just the ExitCode of the most recently run container:
- docker inspect -f '{{.State.ExitCode}}' $(docker ps -lq)
- **Image** The image this container is running.
- **NetworkSettings** The network environment for the container and therefore for the application(s) within the image.
- **LogPath** The system path to this container's log file.
- **RestartCount** Keeps track of the number of times the container has been restarted. This value is the key value used when defining a container's restart policy..
- **Name** The user defined name for the container.
- **Volumes** Defines the volume mapping between the host system and the container.
- **HostConfig** Key configurations for how the container will interact with the host system. These could take CPU and memory limits, networking values, or device driver paths.
- **Config** The runtime configuration options set when the docker run command was executed. Part of this configuration is another "Image" value. This image {{.Config.Image}} is the tagged image which may be different than the image listed in {{.Image}}

$ docker image history docker/whalesay



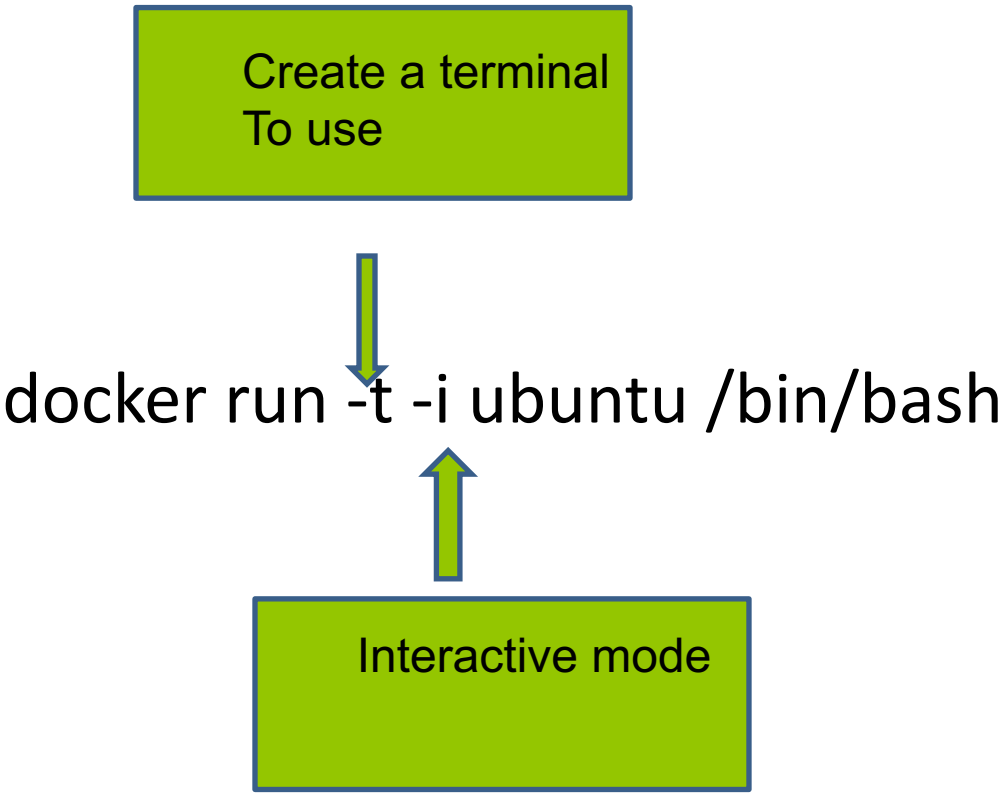
Use history to view layers in an image

Docker Interactive Mode

Best Practice: Run containers in interactive mode while developing docker images

```
ubuntu@instance-41:~/flask-app$ sudo docker run -t -i ubuntu /bin/bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
6bbedd9b76a4: Pull complete
fc19d60a83f1: Pull complete
de413bb911fd: Pull complete
2879a7ad3144: Pull complete
668604fde02e: Pull complete
Digest: sha256:2d44ae143feeb36f4c898d32ed2ab2dffeb3a573d2d8928646dfc9cb7deb1315
Status: Downloaded newer image for ubuntu:latest
root@a4b1111d9d0e:/#
```

Docker Interactive Mode



Docker Interactive Mode

Attach to a running container and shell instance

docker attach some_container #by Name

Attach to a running container by starting a new shell instance

docker exec -it some_container /bin/bash #by Name

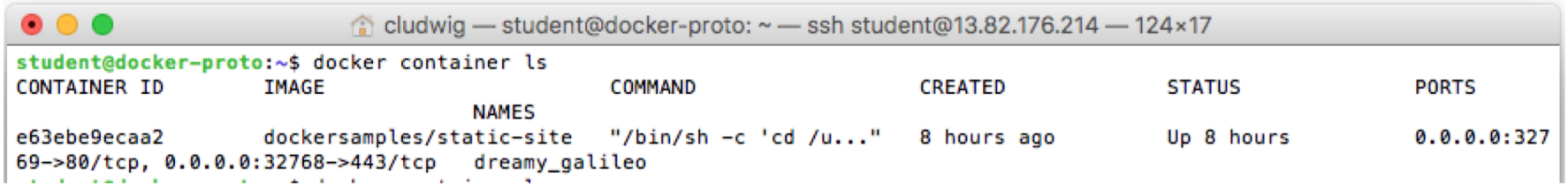
Docker Interactive Mode

Quick Tip: Be aware that alpine and other distros may have a different entry point (sh vs bash)

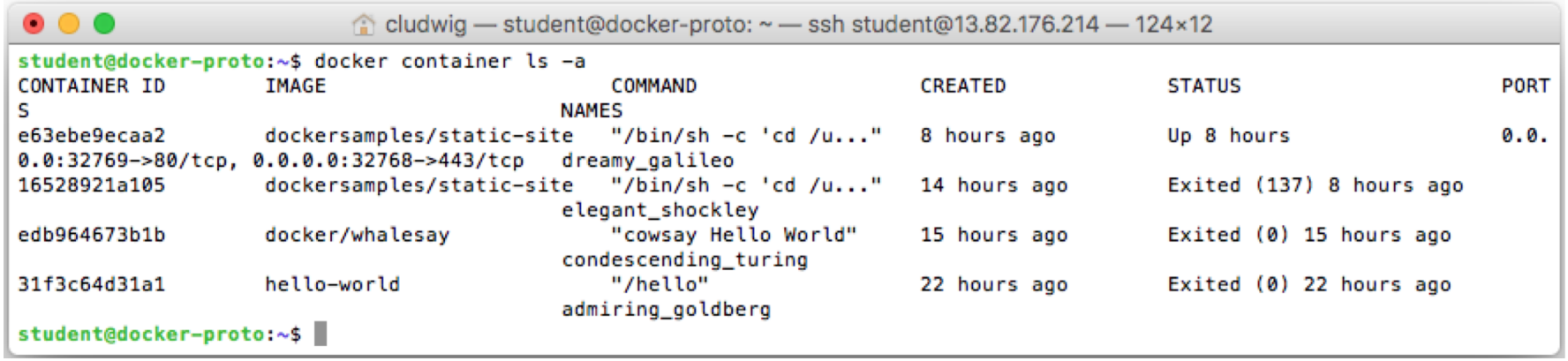
```
docker run -it alpine /bin/sh
```

See all containers

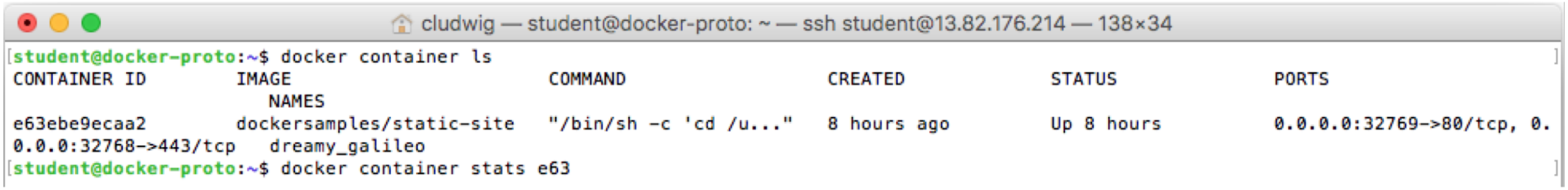
- **$** docker **container ls** #show running

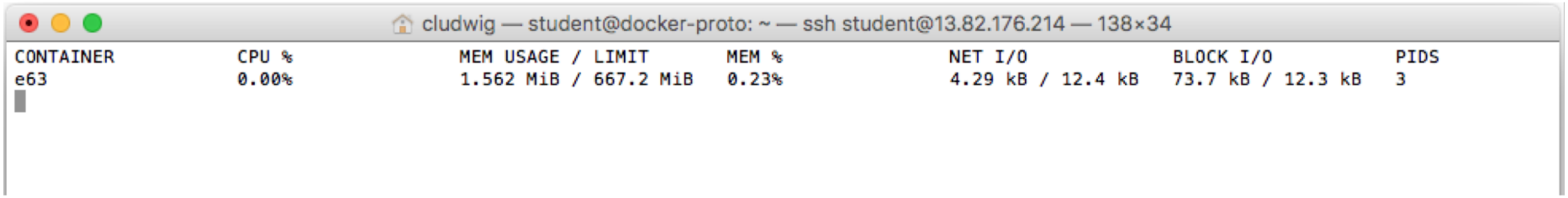


- $ docker container ls -a #show all containers (including running, stopped, exited, etc.)



$docker container stats





- Provides the CPU, memory, network, and other monitoring KPI's of a docker container.
- Multiple containers can be profiled as well
- All containers can be profiled by leaving out the container id(s)

Removing Docker Containers & Images

#!/bin/bash

*# **Remove** all stopped containers*

docker container prune [-f]

*# **Remove** all unused images*

docker image prune [-f] [-a]

-f option skips confirmation.

-a option (for images) removes ALL unused images, not just those that were left behind when a container was removed (dangling).

Best Practice: Prevent image inflation and regularly clean up images

Docker Working with Images and Containers

CLASSROOM WORK 2.1

20 minutes

Practice Docker image and container commands

- https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/docker_image_container.md

DOCKER FILES

Dockerfile

Dockerfile format

```
#Comment
INSTRUCTION arguments
```

Command	Description
FROM	The base image to use in the build. This is mandatory and must be the first command in the file
MAINTAINER	An optional value for the maintainer of the script
RUN	Executes a command and save the result as a new layer This executes during build time
CMD	The command that runs when the container starts Doesn't execute during built time whereas RUN executes during build time
ENV	Sets an environment variable in the new container
COPY	Copies a file from source to destination in the file system of the container
ADD	Remote URL support and Copies a file from the host system onto the container
ENTRYPOINT	Primary command to run whenever the image is executed
EXPOSE	Opens a port for linked containers
ONBUILD	A command that is triggered when the image in the Dcokerfile is used as a base for another image It behaves as if a RUN instruction is inserted immediately after the FROM instruction of the downstream Dockerfile
USER	Sets the default user within the container
VOLUME	Creates a shared volume that can be shared among containers or by the host machine
WORKDIR	Set the default working directory for the container for other instructions like RUN, CMD, COPY, ADD, ENTRYPOINT
LABEL	Add metadata to the image; LABEL key=value

Dockerfile

FROM command

The first instruction in a Dockerfile is a FROM command, to specify the Image from which you are building from.

FROM <image>
FROM <Image>:<tag>

FROM ubuntu:14.04

Dockerfile

MAINTAINER command

This lets you know who to consult (or blame ☺) for any dockerfile issues

MAINTAINER Santa Claus “santa@abc.org”

ENV command

Used to provide environment variables for containers.

ENV <key> <value>
ENV PATH $PATH:/usr/foo

Dockerfile

RUN command

Executes commands and commits the results in a new layer.

RUN <command>
RUN apt-get update

CMD command

Can only be one in a dockerfile. The CMD instruction provides defaults to an executing container and can be overridden with arguments to docker run.

CMD [“cowsay”]

Dockerfile

ADD/COPY command

Places files onto a file system. ADD performs the same as copy, but also allows the <src> to be a URL. It will also unpack recognized compression formats.

COPY <src> <dest>
ADD <src> <dest>

EXPOSE command

Informs Docker that the container listens on a network port at runtime. Ports of a container are still not accessible to the host unless –p flag is passed to the Docker run command when the container is invoked.
EXPOSE 80

Dockerfile

Best Practices for writing Dockerfiles:

- Use .dockerignore to exclude any unnecessary files and improve Docker’s performance
- Run a single process per **container** to simplify scale and reuse
- Minimize layers per container
- Sharing base images is a common among development teams
- When possible, base from tiny images such as Alpine Linux
- For production: specify version of the base image (do not use latest)
- **For production: use image sha digest rather than image name**

Dockerfile

- Dockerfile build syntax

docker build .
docker build -t <**namespace**/image name:**versiontag**> /path
docker build -t myimage/kafka .

***** Be aware that when you run 'docker build .', all files in the current folder get uploaded to docker engine. This may be undesirable, and use of .dockerignore will assist with this situation.***

- Then run the newly created image

Docker run –t myname/jenkins

Dockerfile Exercises

CLASSROOM WORK 3.2

20 minutes

Exercise 2.2 & 2.3 in Docker Labs
https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

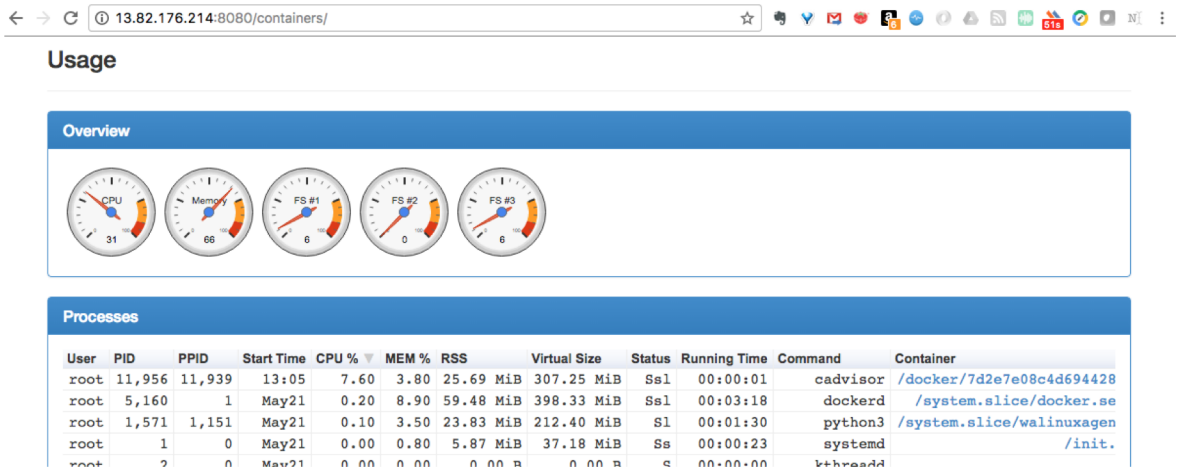
NB: You can skip the second part of step 2.3.4 (“Push Your Image”) unless you want to first create yourself an account at hub.docker.com.

Docker Monitoring

https://github.com/google/cadvisor -
Monitor the system from inside the Docker container!

sudo docker run --volume=/:/rootfs:ro --volume=/var/run:/var/run:rw --volume=/sys:/sys:ro --volume=/var/lib/docker/:/var/lib/docker:ro --publish=8080:8080 --detach=true --name=cadvisor1 google/cadvisor:latest

yourdockerhost:8080



Running a Private Docker Registry

- docker run -d –p 5000:5000 registry

```
ubuntu@instance-41:~/app$ sudo docker pull registry
Using default tag: latest
latest: Pulling from library/registry
3690ec4760f9: Already exists
930045f1e8fb: Pull complete
feeaa90cbdbc: Pull complete
61f85310d350: Pull complete
b6082c239858: Pull complete
Digest: sha256:1152291c7f93a4ea2ddc95e46d142c31e743b6dd70e194af9e6ebe530f782c17
Status: Downloaded newer image for registry:latest
ubuntu@instance-41:~/app$ sudo docker run -d -p5000:5000 registry
64730d7011f71d463d480982a765624b4f353e2f2c7003779281cf453c8b2178
```

Push/Pull enabled
from private registry

Docker Registry

- Tag an image
  - **docker tag ubuntu localhost:5000/myubuntu:0.1**
- Push the tagged image to registry
  - docker push **localhost:5000/myubuntu:0.1**
- **Verify the pushed image**
  - **docker rmi –f ubuntu**
  - **docker rmi –f localhost:5000/myubuntu:0.1**
  - **docker pull localhost:5000/myubuntu:0.1**
  - **Run a container from the above image**

Docker Registry Search

- Search Registry
  - https://docs.docker.com/registry/spec/api/#listing-repositories

```
curl -XGET http://localhost:5000/v2/_catalog
```

- List Image Tags
  - https://docs.docker.com/registry/spec/api/#listing-image-tags

```
curl -XGET http://localhost:5000/v2/myubuntu/tags/list
```

Docker Create your first image

CLASSROOM WORK 3.1

25 minutes

https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/docker_create_image.md

Docker Monitoring/Registry Exercise

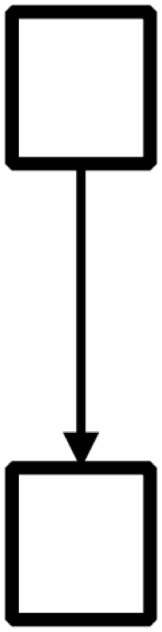
CLASSROOM WORK

15 minutes

https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/docker_monitor_registry.md

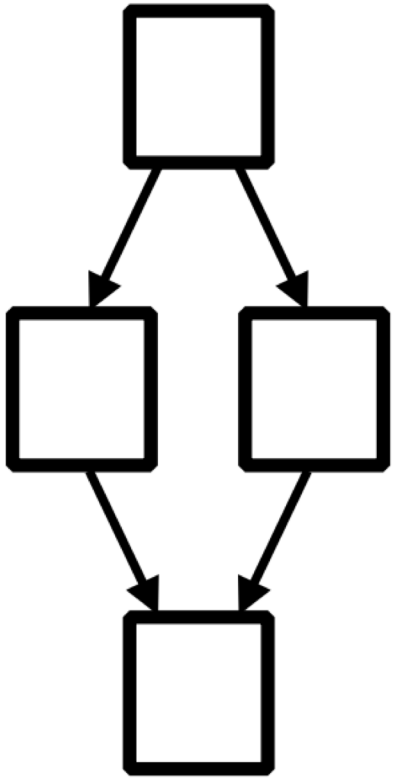
DOCKER COMPOSE

Launching multiple containers is clumsy



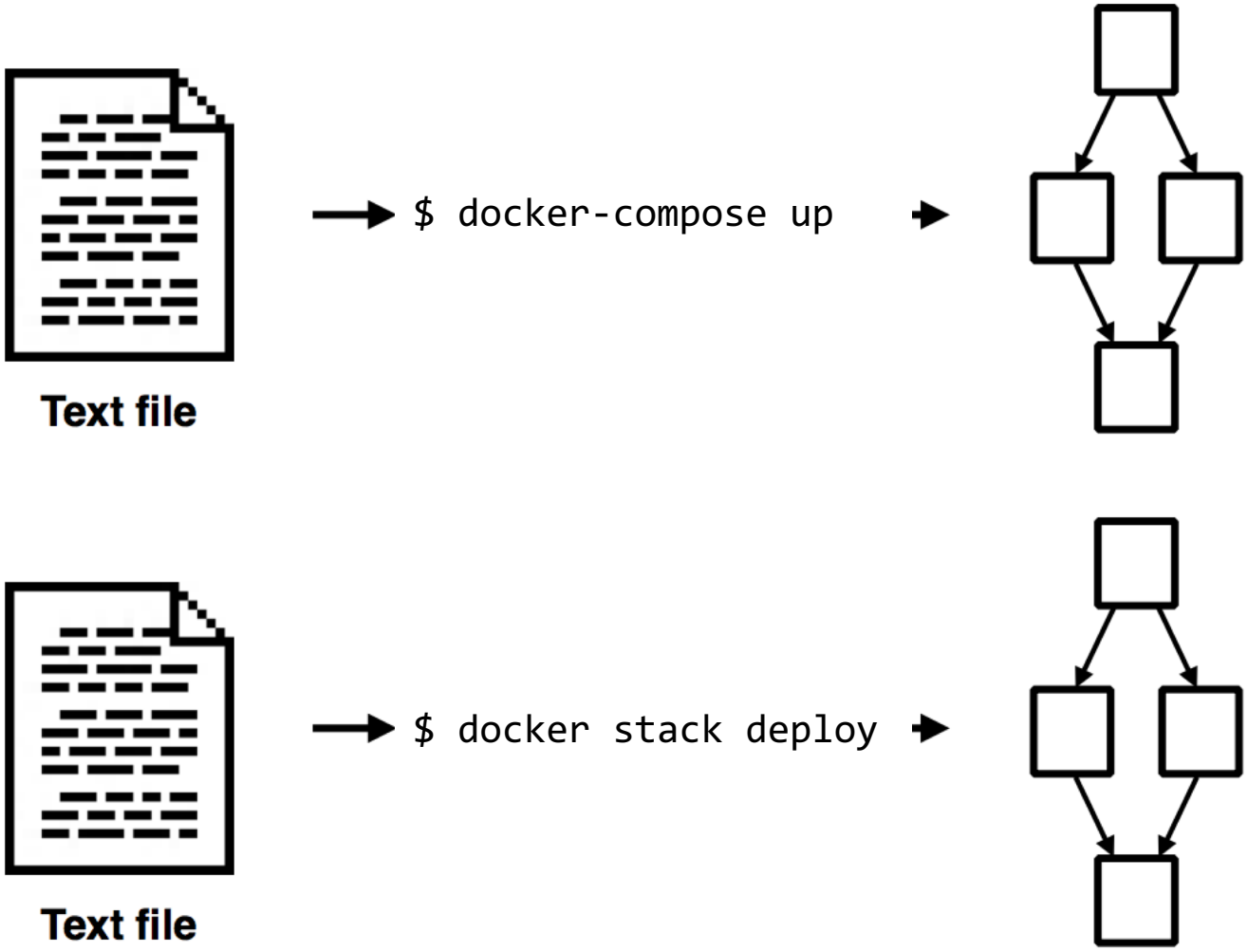
```
$ docker pull redis:latest

$ docker build -t web .

$ docker run -d --name=db redis:latest redis-server --
appendonly yes

$ docker run -d --name=web --link db:db -p
5000:5000 -v `pwd`:/code web python app.py
```

Launching multiple containers is clumsy



```
$ docker pull ...
$ docker pull ...
$ docker build ...
$ docker build ...

$ docker run ...
$ docker run ...
$ docker run ...
$ docker run ...
```

Docker Compose or Swarm launches app with a single command



Docker Compose commands

docker-compose up	(Re)build services
docker-compose kill	Kill the containers
docker-compose logs	Show the logs of the containers
docker-compose down	Stop and remove images, containers, volumes and networks
docker-compose rm	Remove stopped containers

Docker Compose to deploy a cluster

CLASSROOM WORK

First, install Compose:

https://github.com/docker/compose/releases

- sudo curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
- sudo chmod +x /usr/local/bin/docker-compose
- docker-compose version

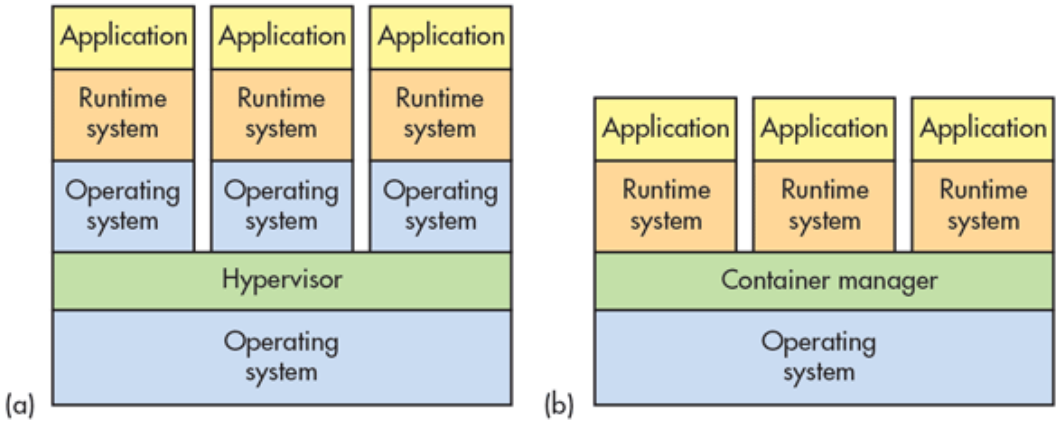
Then, Compose “Getting Started” Lab:

https://docs.docker.com/compose/gettingstarted/

Challenges of using Docker in a single machine

Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify Devops practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption



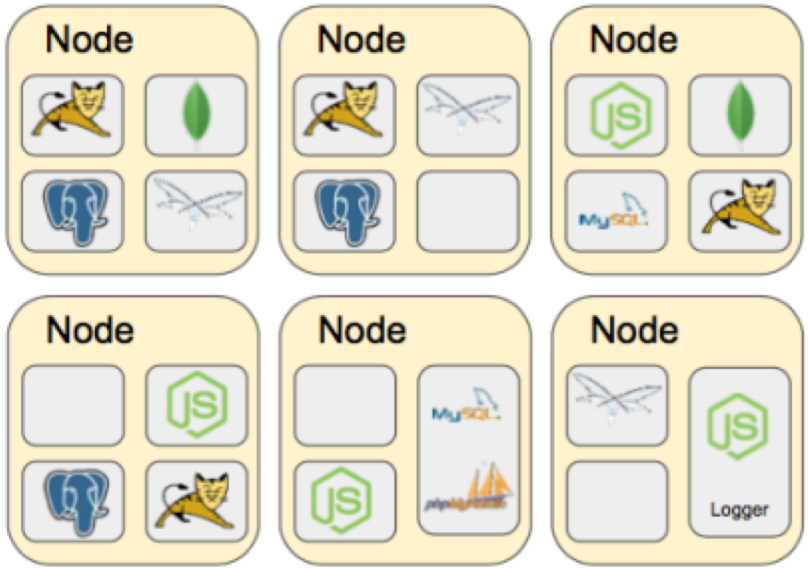
The Need for Container Orchestration

- Containers are becoming the standard unit of deployment
- Each container image has
  - Code
  - Binaries
  - Configuration
  - Libraries
  - Frameworks
  - Runtime
- Developers and Operators love containers

The Need for Container Orchestration

- Docker has solved the problem of packaging, deploying and running containerized applications
- Docker has 3 key components
  - Container Engine
  - Registry
  - Tools (like docker-cli, docker-compose, docker-machine, etc.)
- Docker is great for managing a few containers running on a fewer machines
- Production applications deal with dozens of containers running on hundreds of machines

Challenges with multiple containers



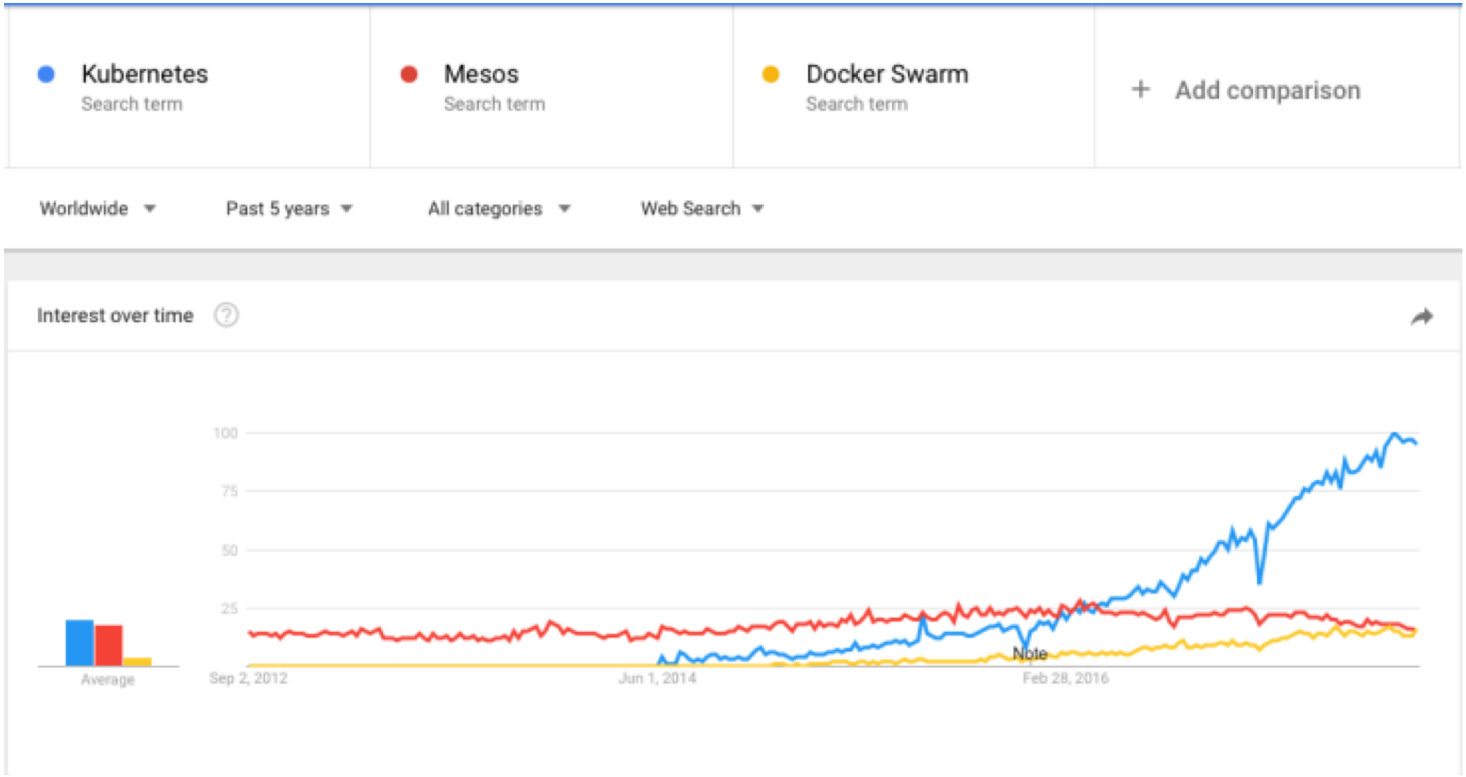
- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?

The Need for Container Orchestration

- The unit of deployment is changing from a machine to a container
- Infrastructure has become immutable
- Emphasis on treating the datacenter as a large server (cluster)
- Tools are evolving to manage the new datacenter infrastructure
  - Docker Swarm
  - Kubernetes
  - Mesosphere DC/OS
- Manage the lifecycle of containerized applications running in production
- Automate the distribution of applications

Container Orchestration Tools

- The three most popular are: Kubernetes, Docker Swam & Mesos
- Kubernetes has become the unofficial standard

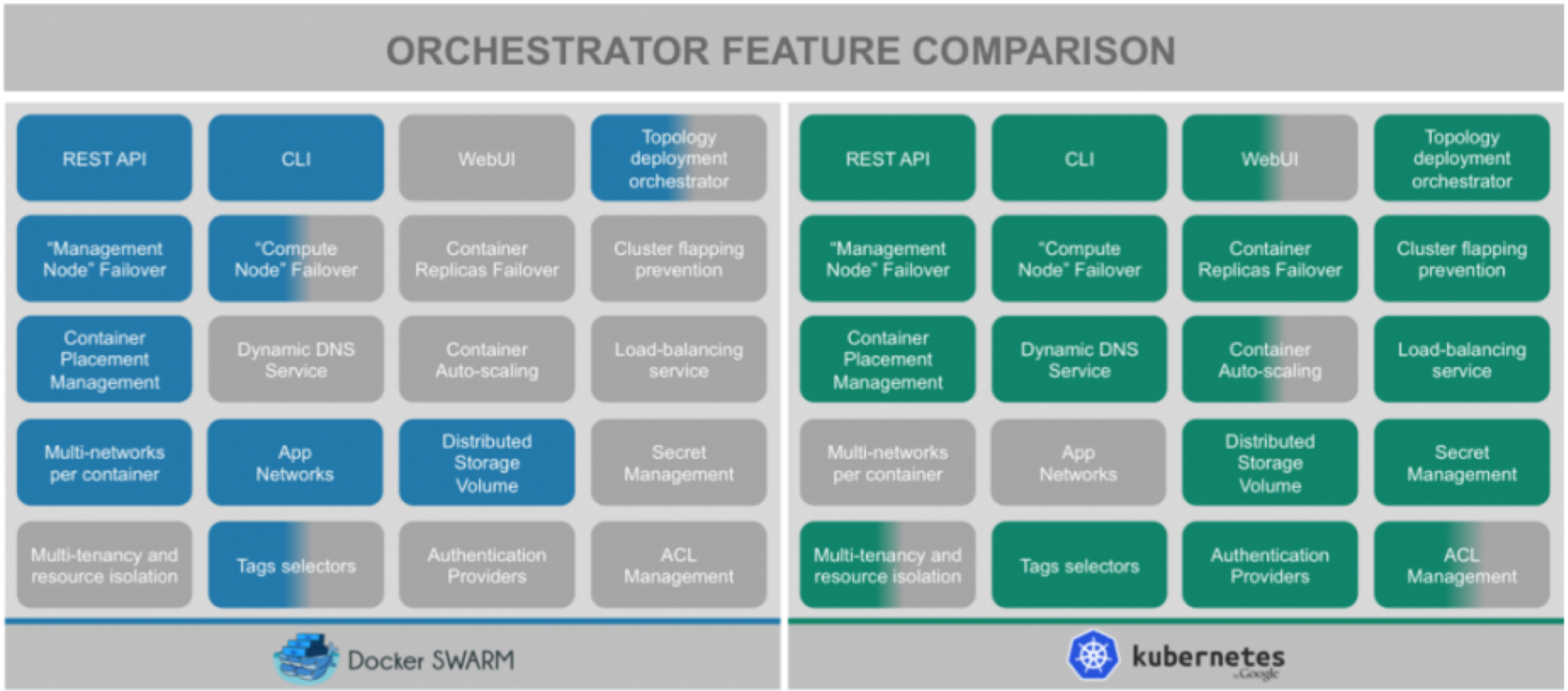


Kubernetes Stats

- Top 100 project by stars
- 22k LinkedIn professionals
- Largest container management ecosystem

Github		
54k+ Commits	280+ Releases	1365+ Contributors
Top 100 Forked Github Project	Top 2 Starred Go Project	Top 0.01% Starred Github Project

Feature Comparison (March 2016)



End of Docker Overview – Part I