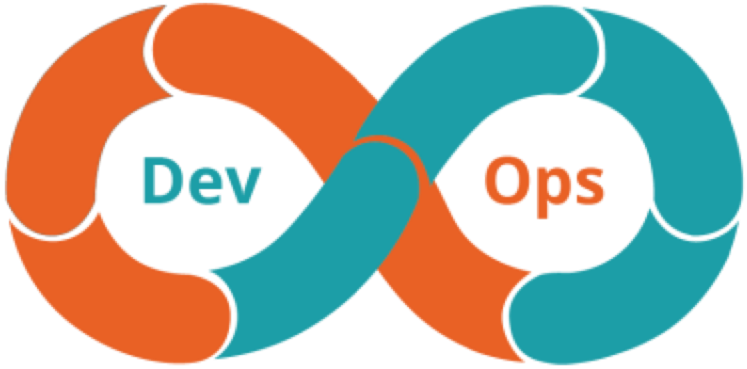
Docker Containerization Boot Camp



Welcome!

Logistics (breaks, facilities, lunch, etc.)

Rules of Engagement

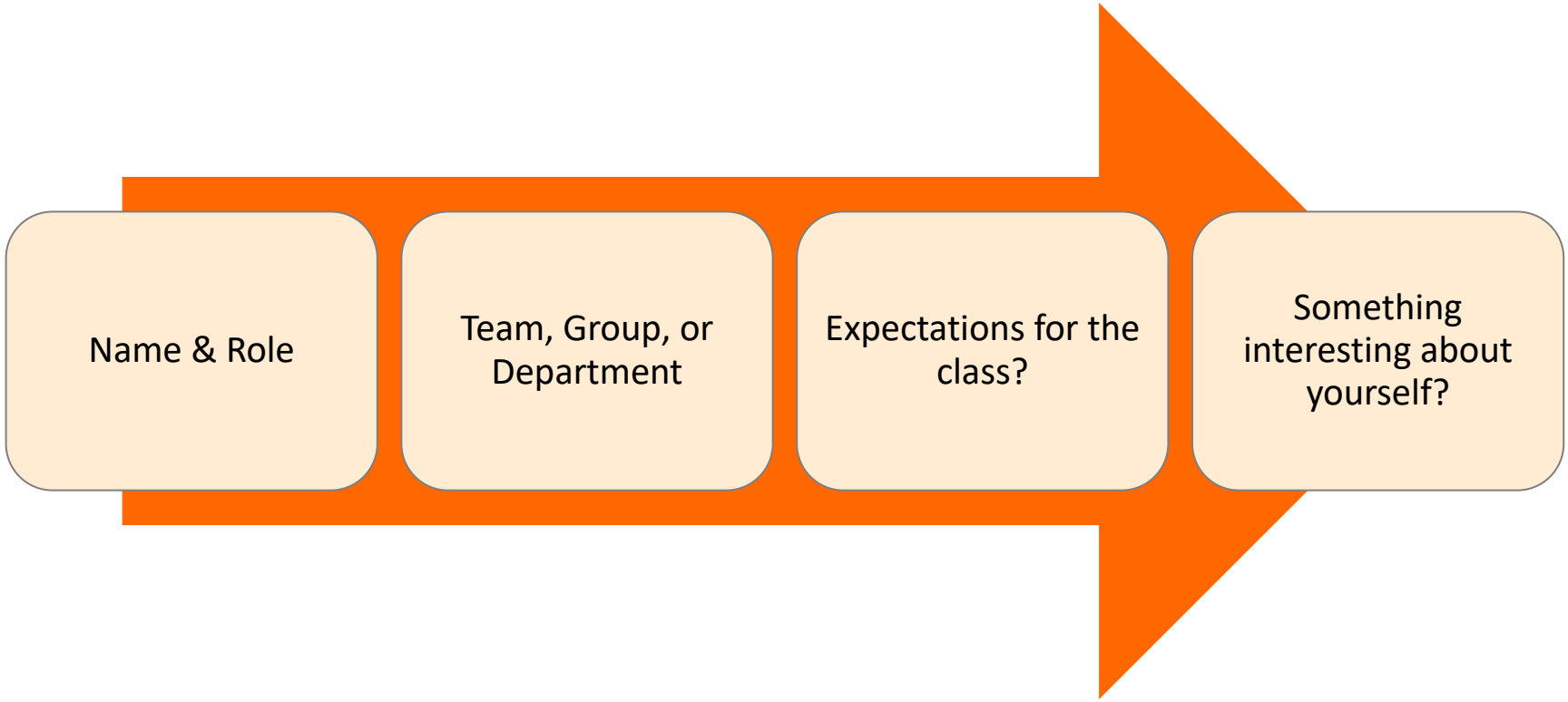
Introductions

Let’s Get Started!

Who is your instructor?

A little about me…

Introductions



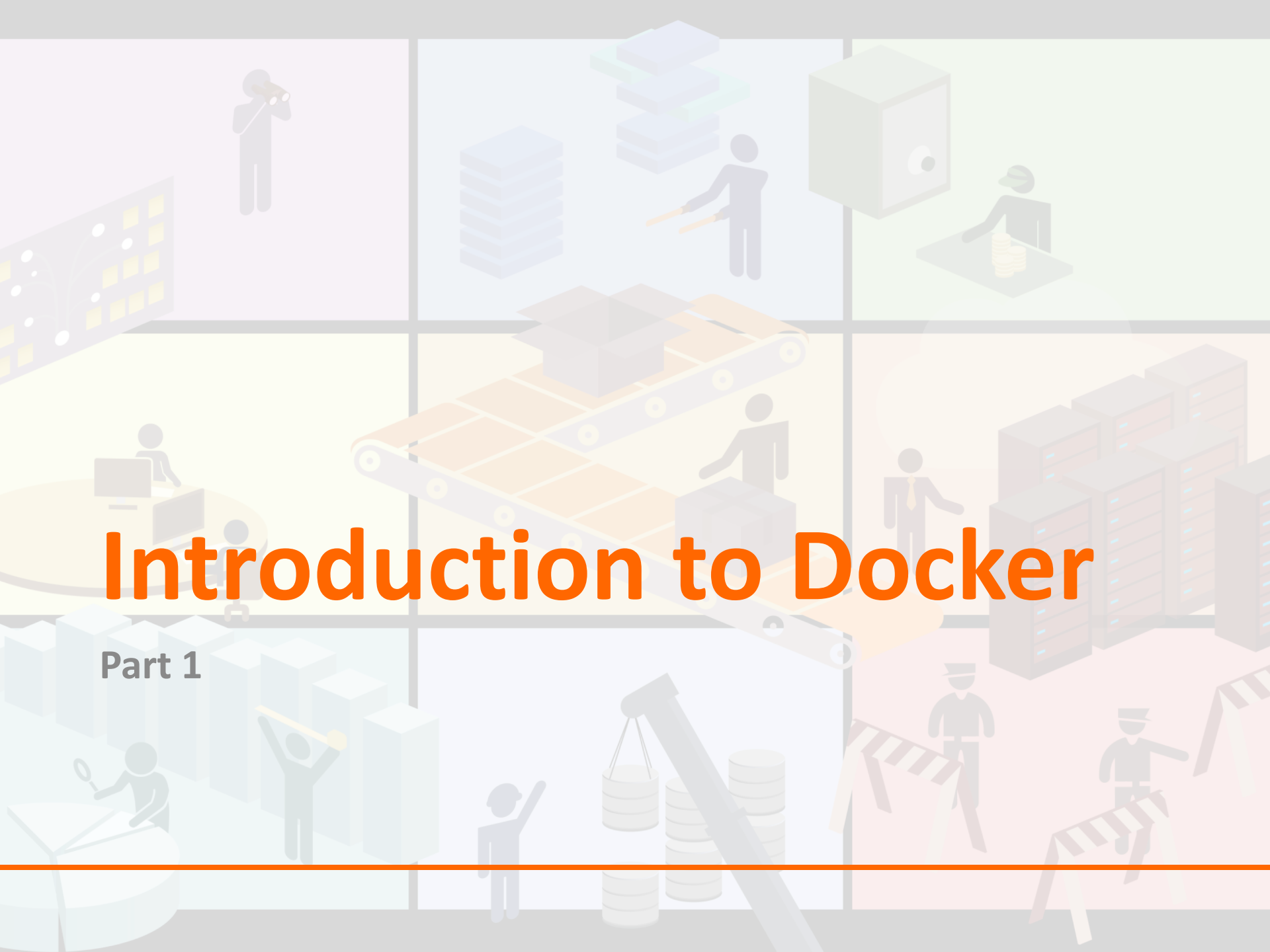
What to Expect from this Class

Flexibility

Conversations

Literacy and awareness on many of the principles, tools, and practices surrounding Docker as part of a DevOps architecture.

An effort to focus on your own situations and challenges so you can act on what you learn.



Introduction to Docker

Part 1

What is Docker?

The Docker Project

Open Source Project

- 2B+ Docker Image Downloads
- 2000+ contributors
- 40K+ GitHub stars
- 200K+ Dockerized apps
- 240 Meetups in 70 countries
- 95K Meetup members

Docker Inc

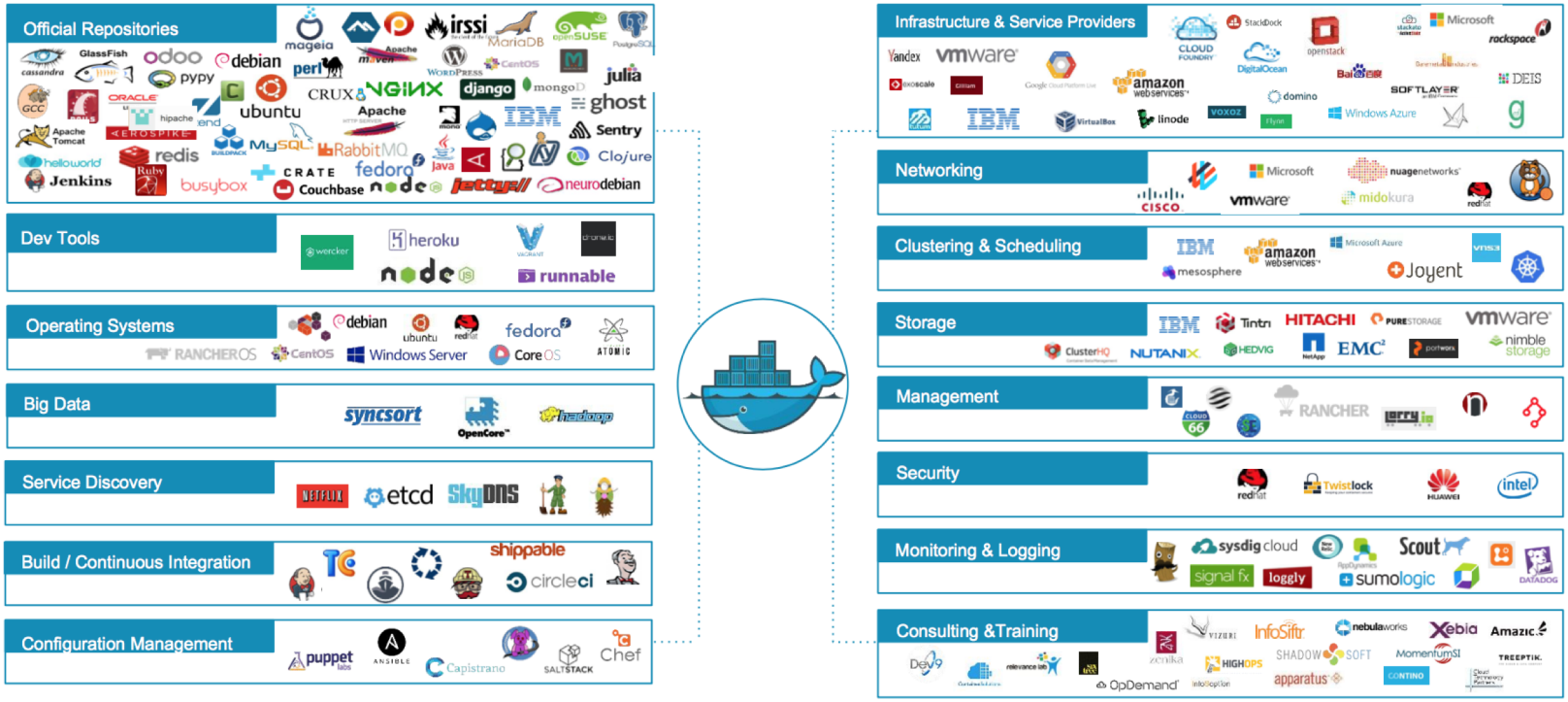
Containers as a Service provider

- Integrated platform for dev and IT
- Commercial technical support

Docker project sponsor

- Primary sponsor of Docker project
- Supports project maintainers

Docker Ecosystem



Docker Basics

Docker Image

- The basis of a Docker container

Docker Container

- The standard unit in which the application service resides

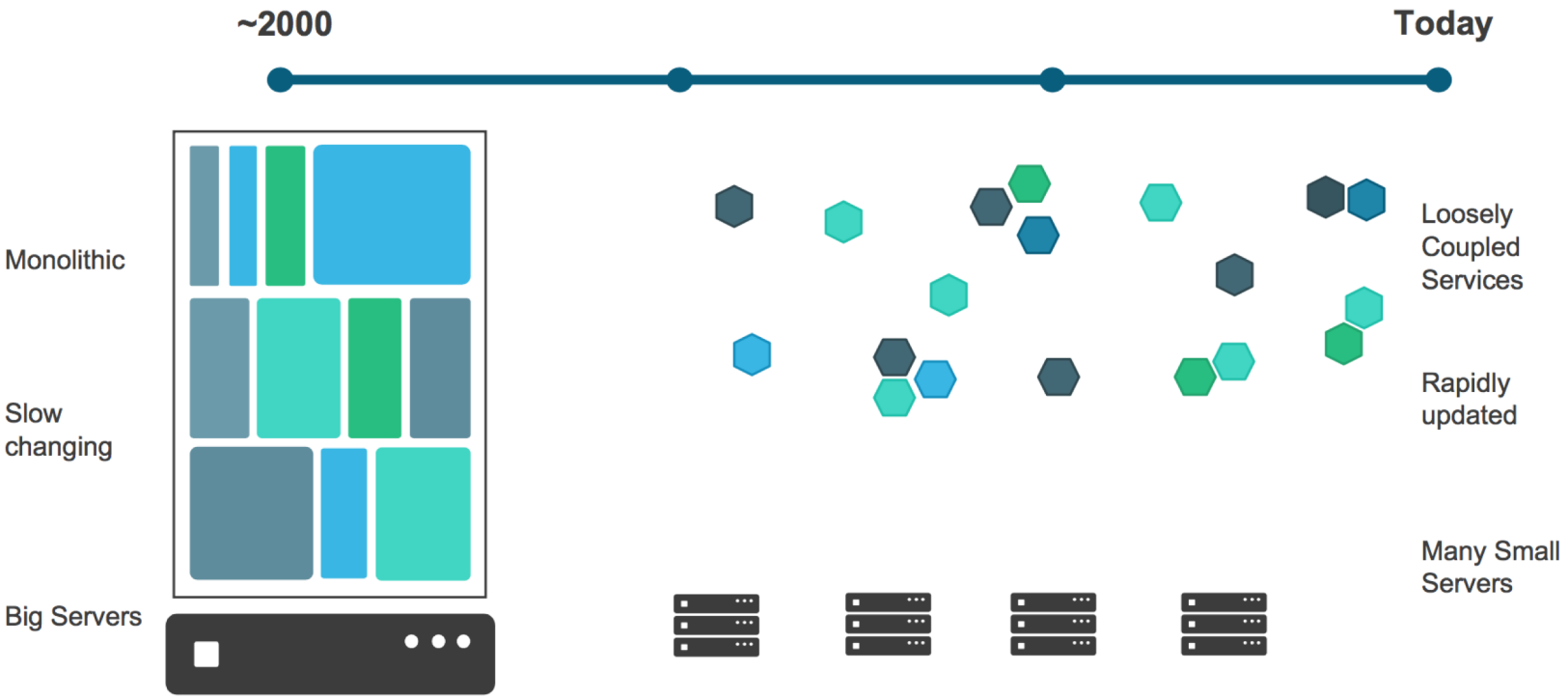
Docker Engine

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

Docker Registry

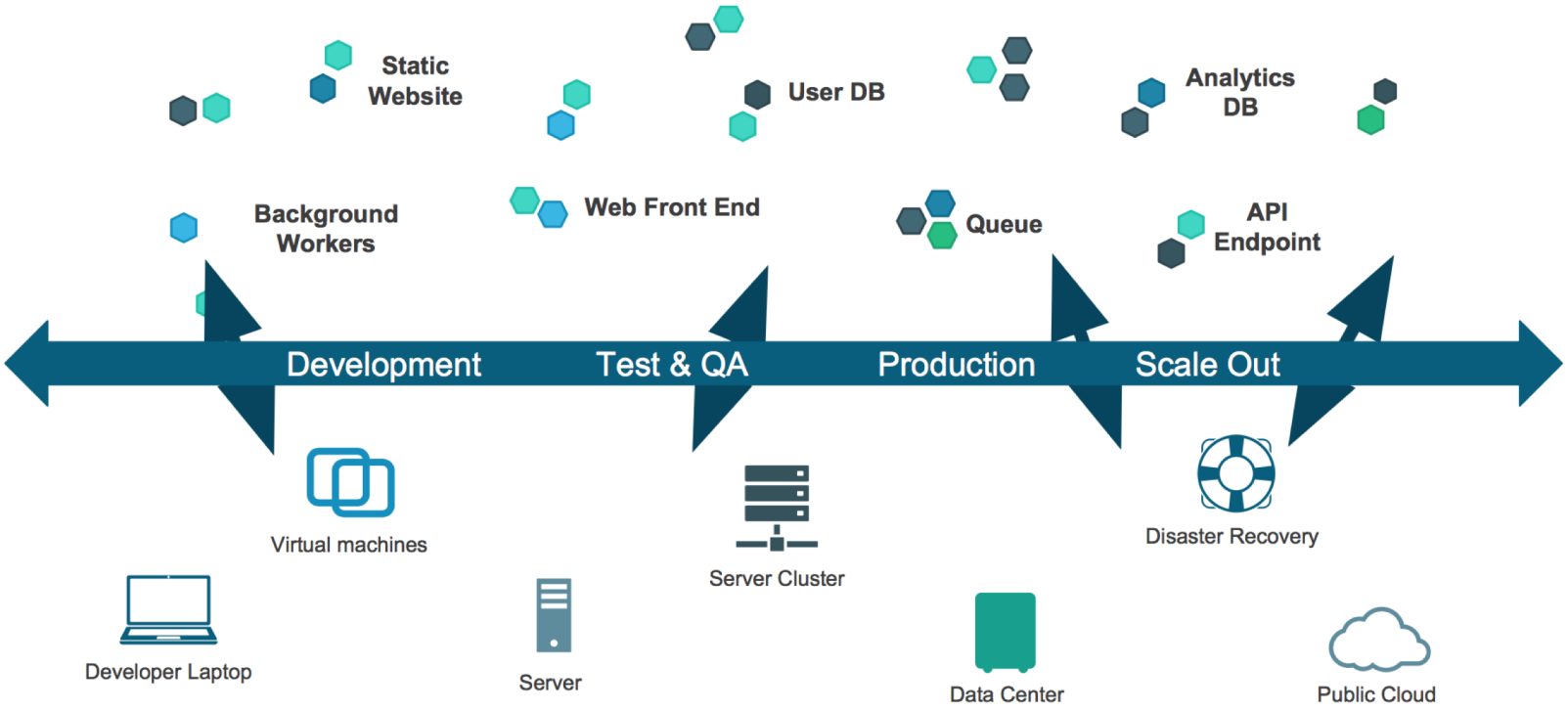
- On-premises registry for image storing and collaboration

Application Evolution



Challenge: The Dependency Matrix

“It works on MY machine.”



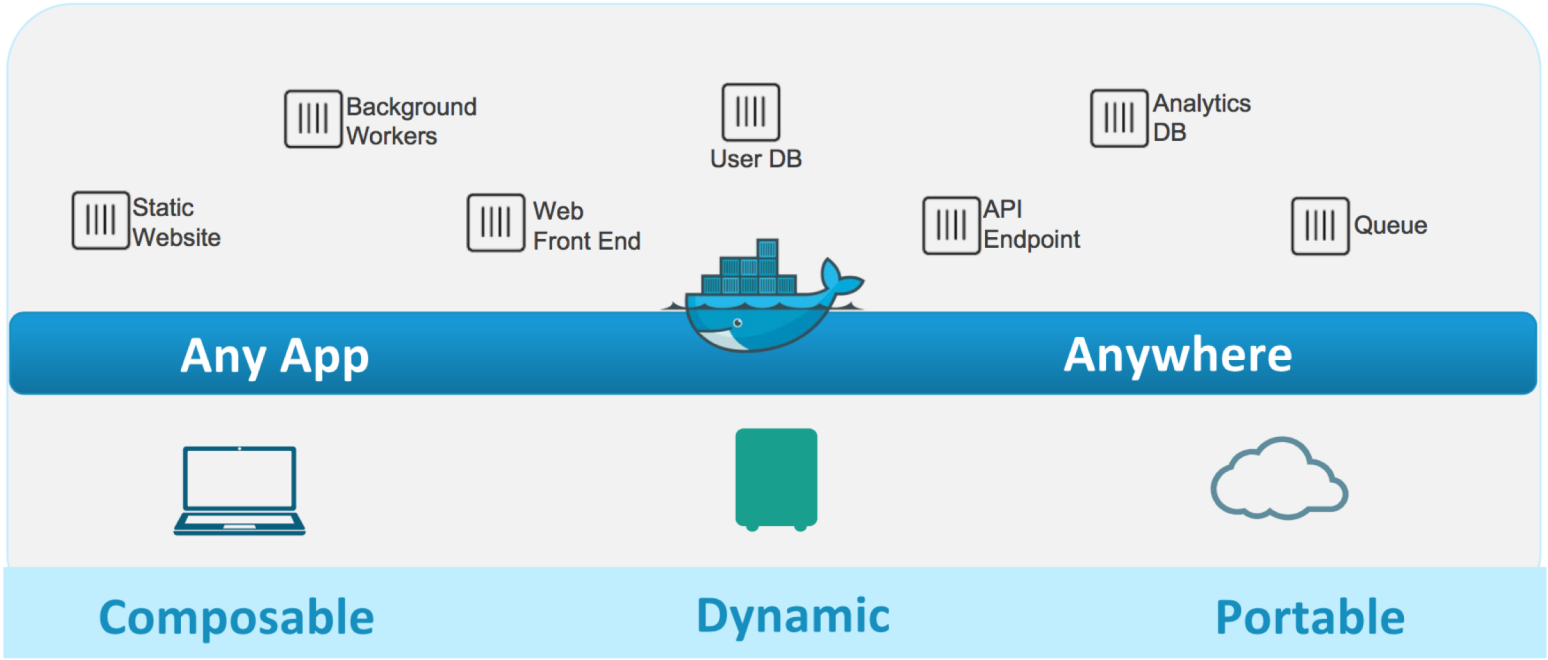
Notes

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

Containers as a Solution



Developer Benefits

Build once...(finally) run anywhere

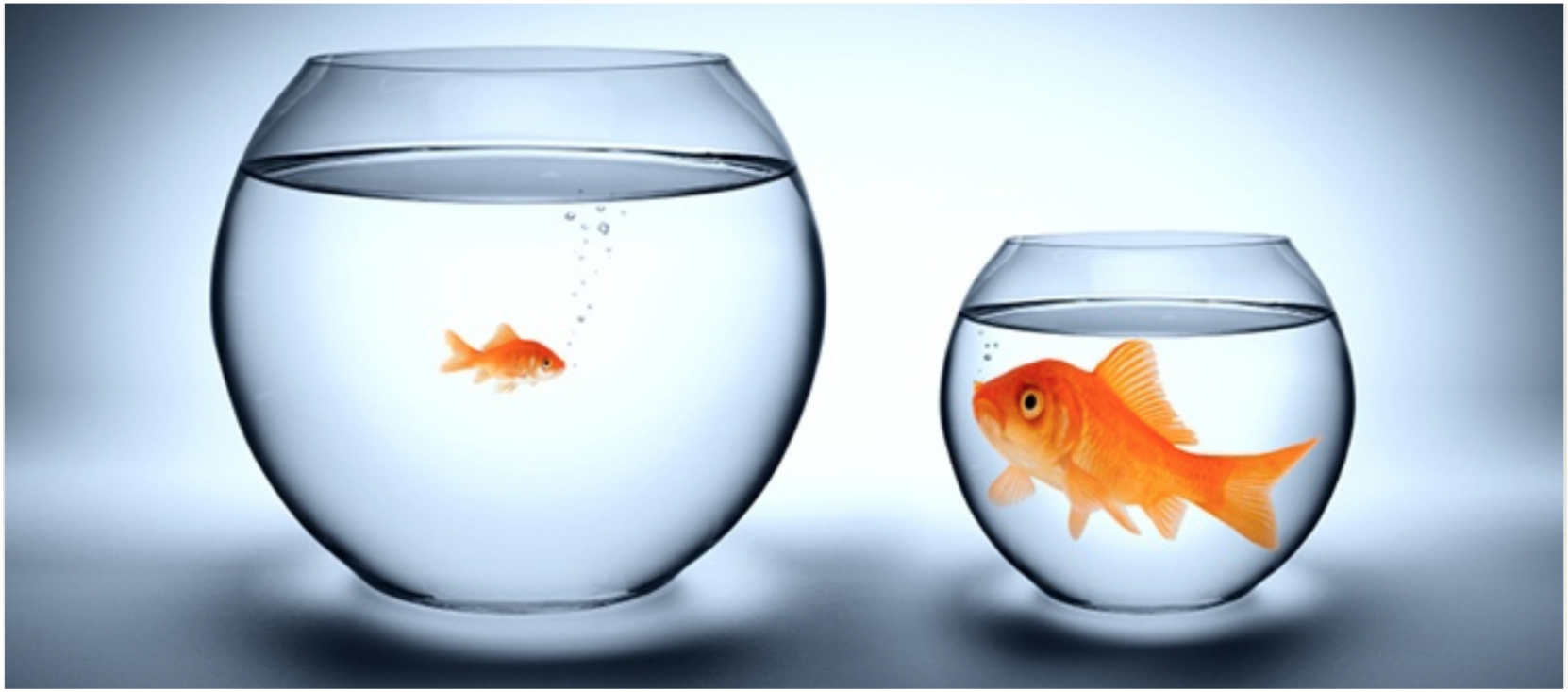
- **A clean, safe, hygienic and portable runtime environment for your app.**
- **No worries about missing dependencies, packages and other pain points during subsequent deployments.**
- **Run each app in its own isolated container, so you can run various versions of libraries and other dependencies for 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

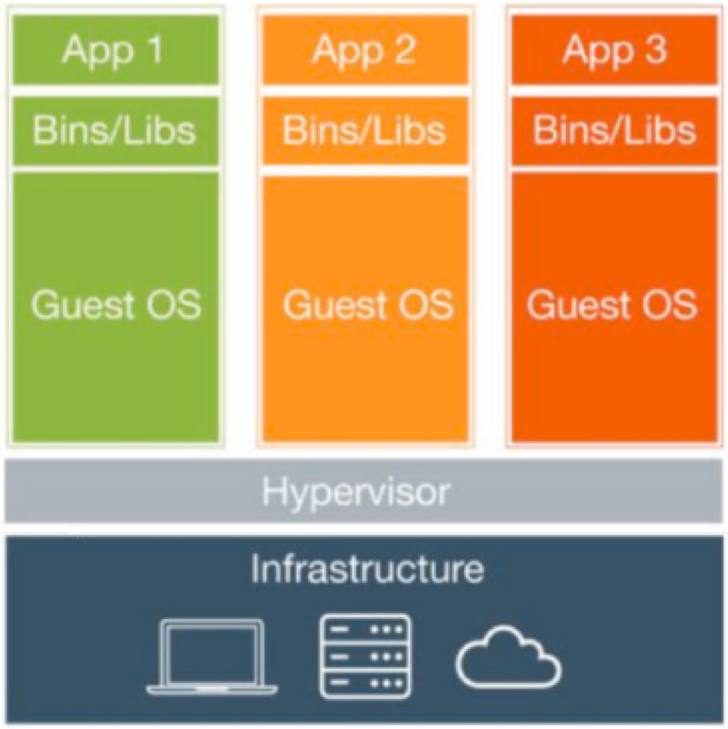
Configure once...run anything

- **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**

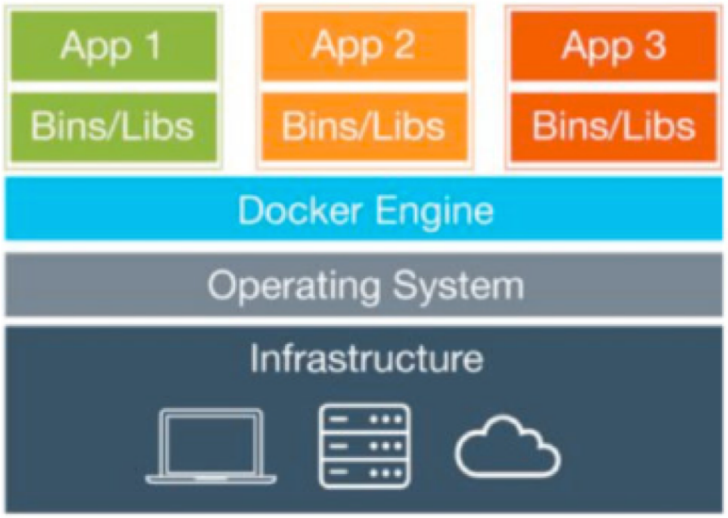
VMs vs. Containers



VMs vs. Containers



Virtual Machines



Containers

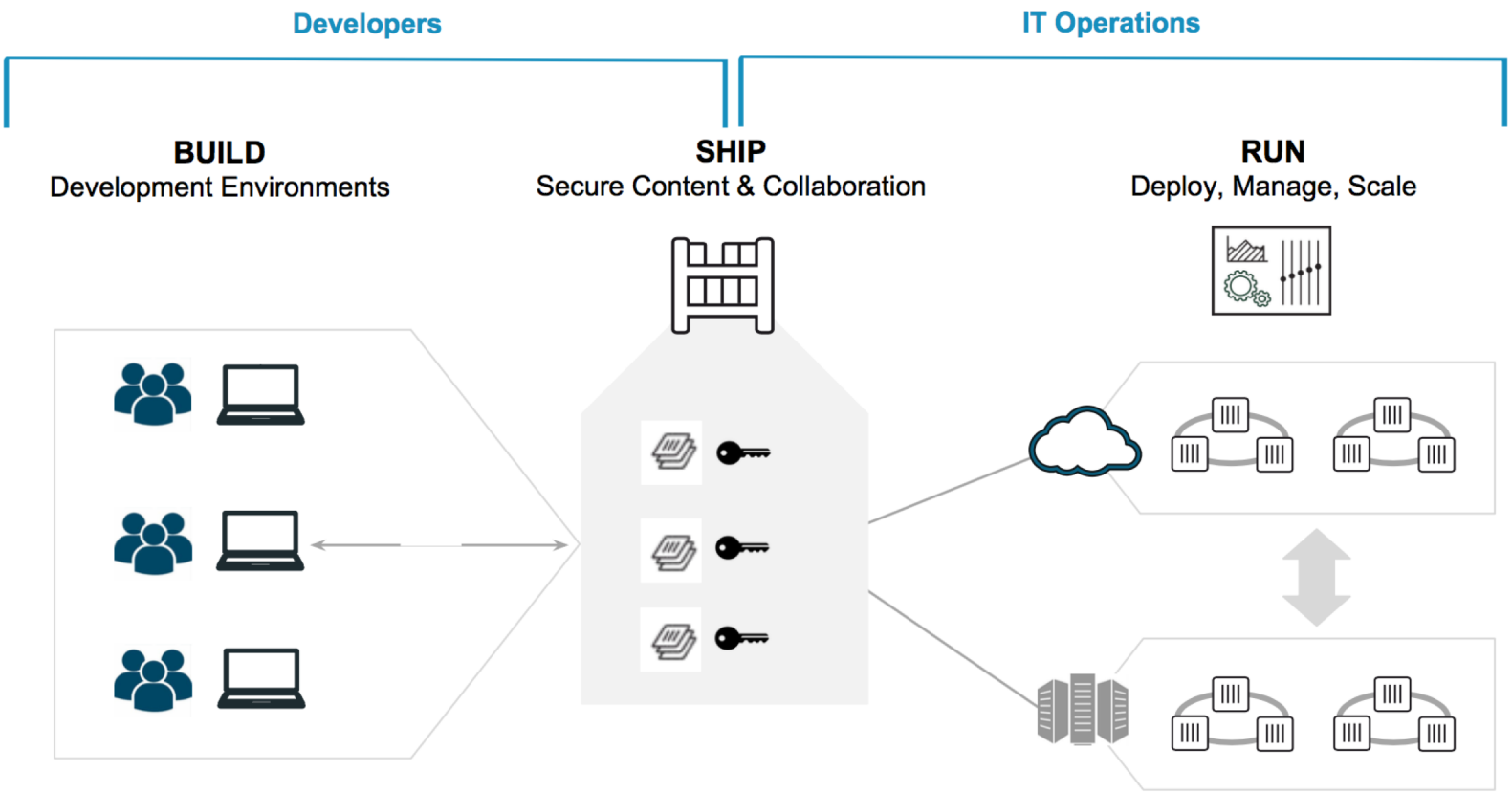
VMs vs. Containers

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)

VMs vs. Containers

Simulates a physical machine
Provides a local file system
Can be accessed over a network
Full and independent guest operating system
Virtualized device drivers
Strong resource and memory management
Huge memory foot-print
Needs a hypervisor

Dev/Ops Container Workflow



Container Benefits

Docker CE vs. Docker EE

	COMMUNITY EDITION	ENTERPRISE EDITION BASIC	ENTERPRISE EDITION STANDARD	ENTERPRISE EDITION ADVANCED
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Docker Certified - Infrastructure, Plugins and ISV Containers		✓	✓	✓
Image Management (private registry, caching)	Cloud hosted repos		✓	✓
Docker Datacenter - Integrated container app management			✓	✓
Docker Datacenter - Multi-tenancy with RBAC, LDAP/AD support			✓	✓
Integrated secrets mgmt, image signing policy			✓	✓
Image security scanning	Preview			✓
Support	Community	Biz Day or 24x7	Biz Day or 24x7	Biz Day or 24x7
Pricing	Free	$750/node/year	$1,500/node/year	$2,000/node/year

Installing Docker

- **Docker container 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 on 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.**

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/docker_installation.md

Prerequisite: Connect to Classroom VM

CLASSROOM WORK

- **$ ssh student@FQDN**
  - **student password = DockerStudentPW0**

FQDN = The fully qualified domain name of the machine assigned to you by your instructor. This will commonly be in the form docker#.domain.com (where # is your student number as assigned to you by your instructor).

Prerequisite: Add & Update Repos

CLASSROOM WORK

Also at
https://docs.docker.com/engine/installation/

```
$ 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
```

Install Docker-CE

CLASSROOM WORK

Also at
https://docs.docker.com/engine/installation/

```
$ sudo apt-get -y install docker-ce

$ sudo docker run hello-world
```

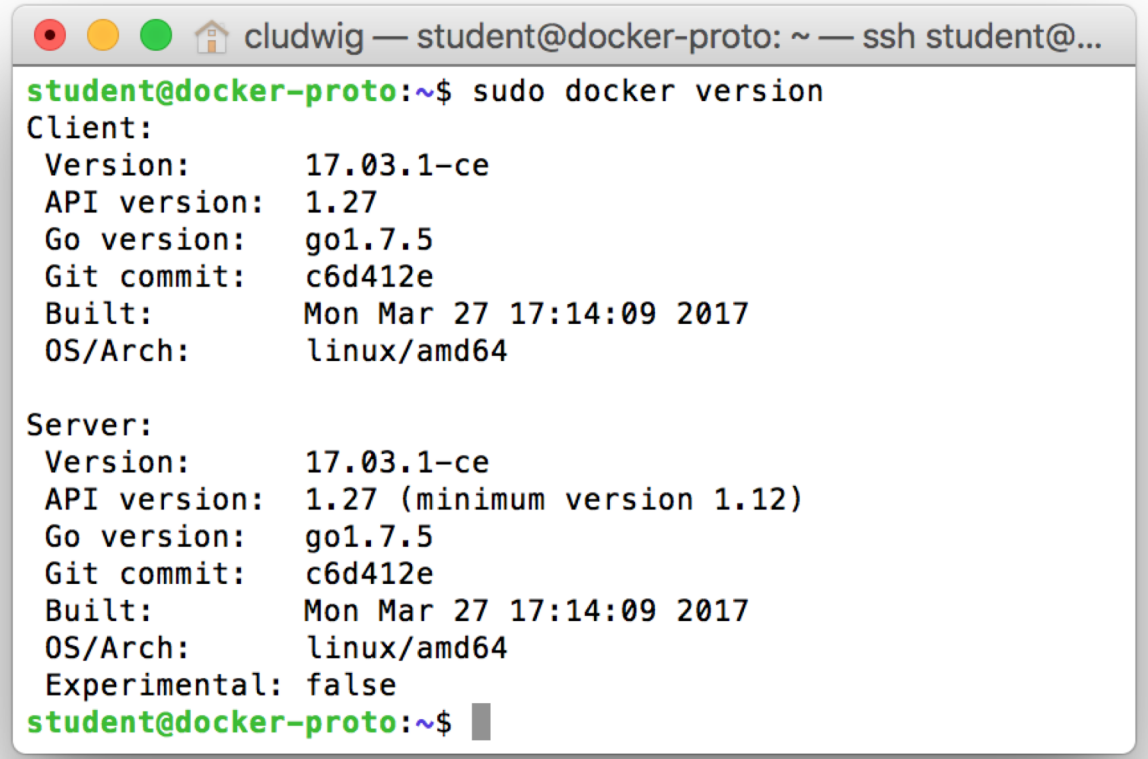
Installing Docker Script

It's good to install Docker the "hard" way at least once so you understand what is involved.

- **There is a Docker-maintained 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**
- **The script will also install a Union File System driver and verify Docker engine functionality**

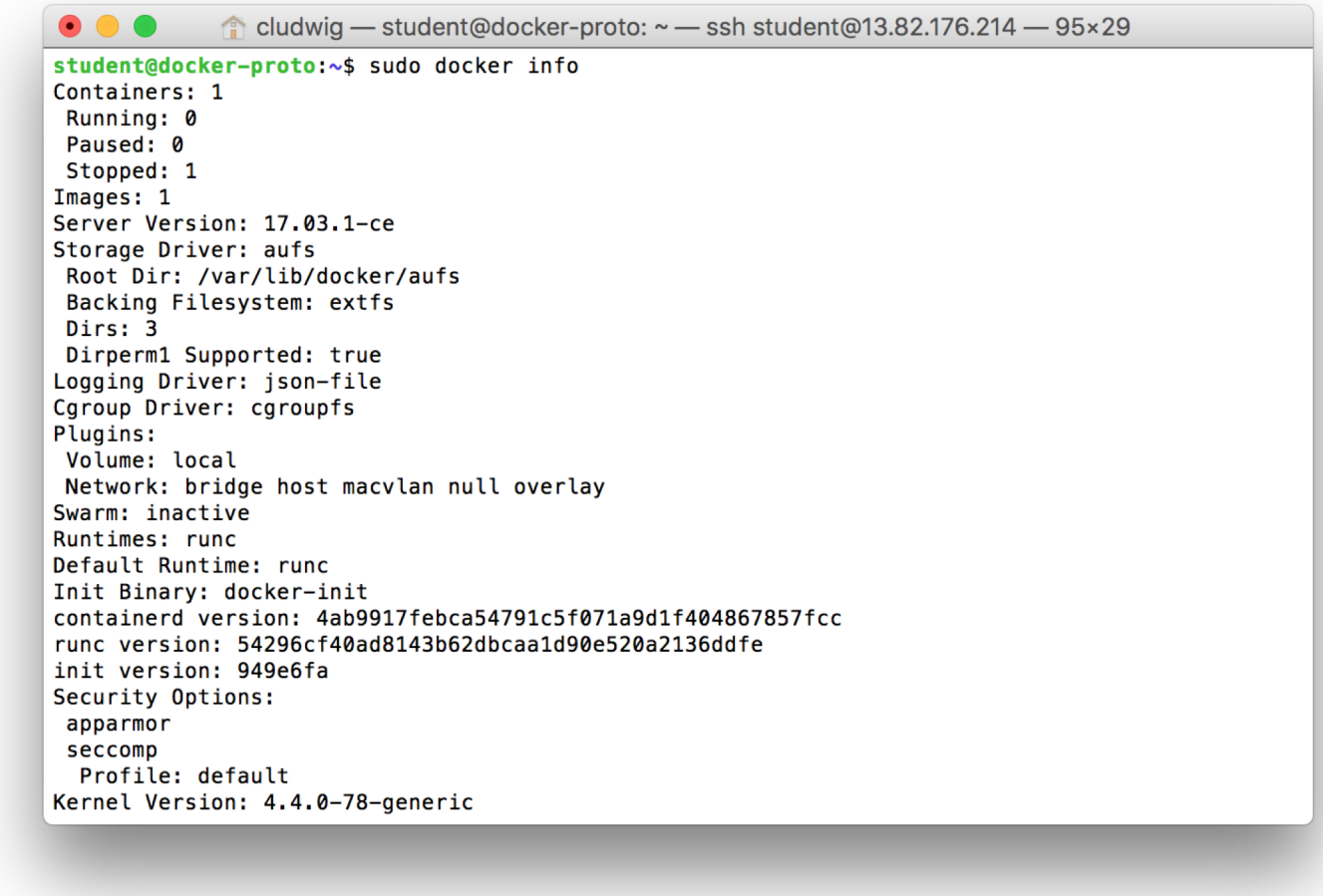
Docker Basics

$ sudo docker version



Docker Basics

$ sudo docker info



Docker Basics

- Standard command format:
  - docker [NOUN] verb
- $ sudo docker --help #displays help and management subcommands (nouns)
- $ sudo docker NOUN --help
- There are reasonable defaults and short forms of option flags for many things. i.e.:
  - “docker pull” assumes “image”
  - “docker stop” assumes “container”
  - can use “img” instead of “image”
  - can use “con” instead of “container”, etc.

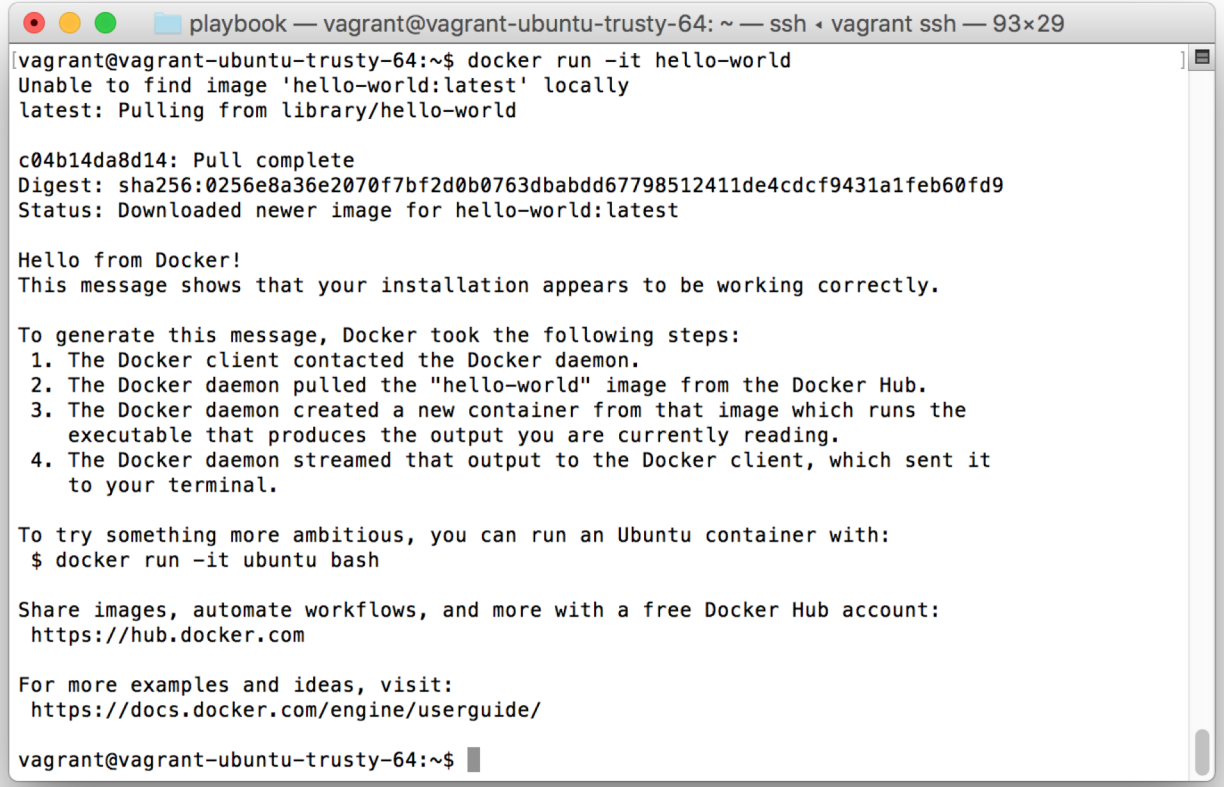
Docker Basics

Determine the status of the Docker service

- $ sudo service docker status

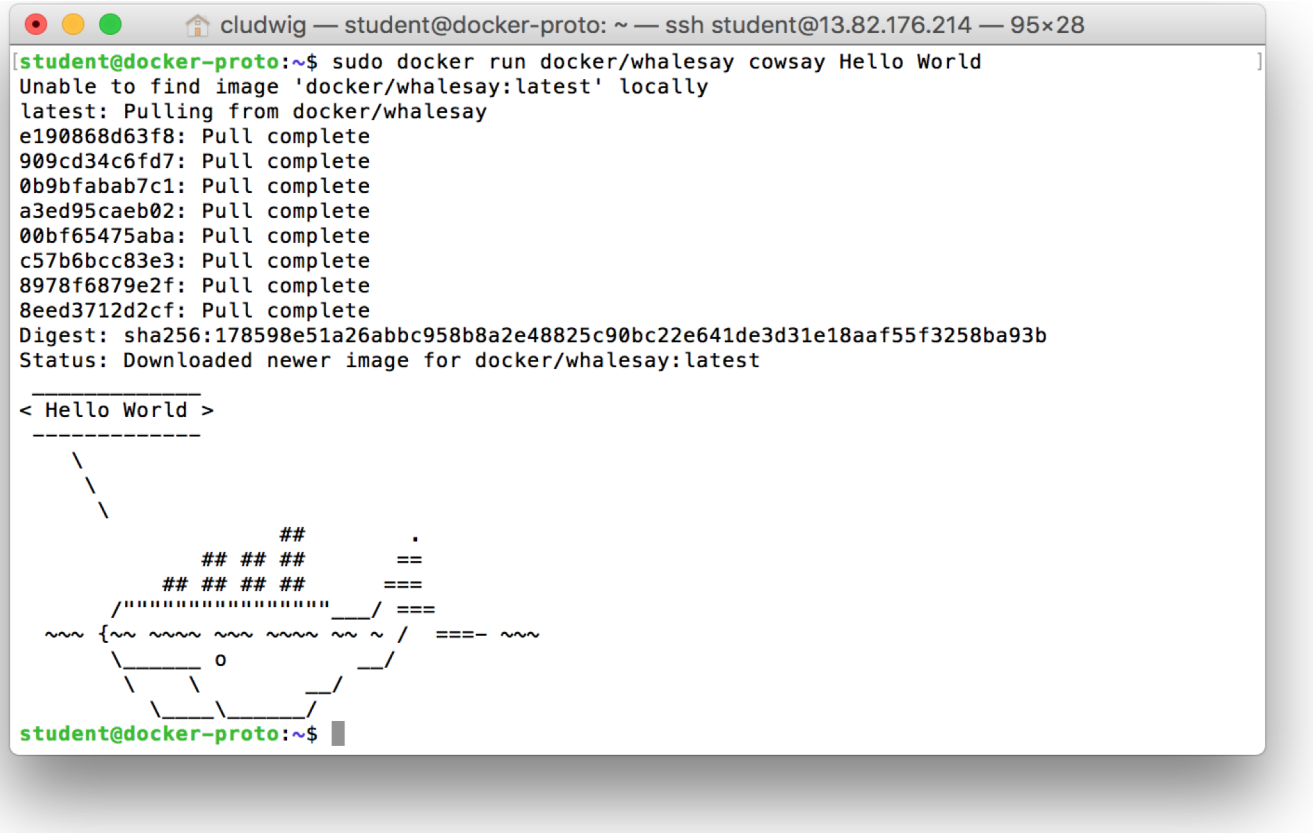
```
ubuntu@ubuntu-xenial:~/docker$ sudo service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2016-11-29 14:12:54 UTC; 2h 47min ago
     Docs: https://docs.docker.com
 Main PID: 1204 (dockerd)
    Tasks: 22
   Memory: 82.9M
      CPU: 1min 31.242s
   CGroup: /system.slice/docker.service
           ├─1204 /usr/bin/dockerd -H fd://
           └─1212 docker-containerd -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --shim docker-containerd-shim --metrics-interval=0 --start-ti

Nov 29 16:29:33 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:29:33.147992827Z" level=info msg="Layer sha256:6fc2e4ad81348c14fd2d7e3880b5db2bbc2f699a5cdcf18b
Nov 29 16:29:36 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:29:36.424426912Z" level=info msg="Layer sha256:a21398f45083710727f6f382cf232d5d50d4dcd9589d6ff5
Nov 29 16:30:09 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:30:09.788820380Z" level=info msg="Layer sha256:9359188bd45e8c8b98fcd0f0297ba87b4f5ed64e9dc8fba7
Nov 29 16:30:09 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:30:09.913582571Z" level=info msg="Layer sha256:9359188bd45e8c8b98fcd0f0297ba87b4f5ed64e9dc8fba7
Nov 29 16:30:28 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:30:28.557526233Z" level=info msg="Layer sha256:be3b076c597ddebb0e264d732927a160c2e91c78516c84a1
Nov 29 16:30:50 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:30:50.018714100Z" level=info msg="Layer sha256:47e85df6c4115d2974ab00ef823c215917dae0a9011f4262
Nov 29 16:30:56 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:30:56.620202459Z" level=info msg="Layer sha256:ce06475a45c5830b905647755ce0324b8c313502bf66ea0c
Nov 29 16:31:14 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:31:14.829527550Z" level=info msg="Layer sha256:2b9628b0eba7b5a870fd354986be4d15e605d51c8f9f7f2a
Nov 29 16:31:15 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:31:15.102270791Z" level=info msg="Layer sha256:c8f15147c2663dc3d68f177fffc0934bb2c04e25eddc6007
Nov 29 16:32:33 ubuntu-xenial dockerd[1204]: time="2016-11-29T16:32:33.801824138Z" level=info msg="Layer sha256:72deb7ae364e53981d0d1befdf7d9c5b15208f93cd8e169b
lines 1-22/22 (END)
```

Docker Hello World(s)

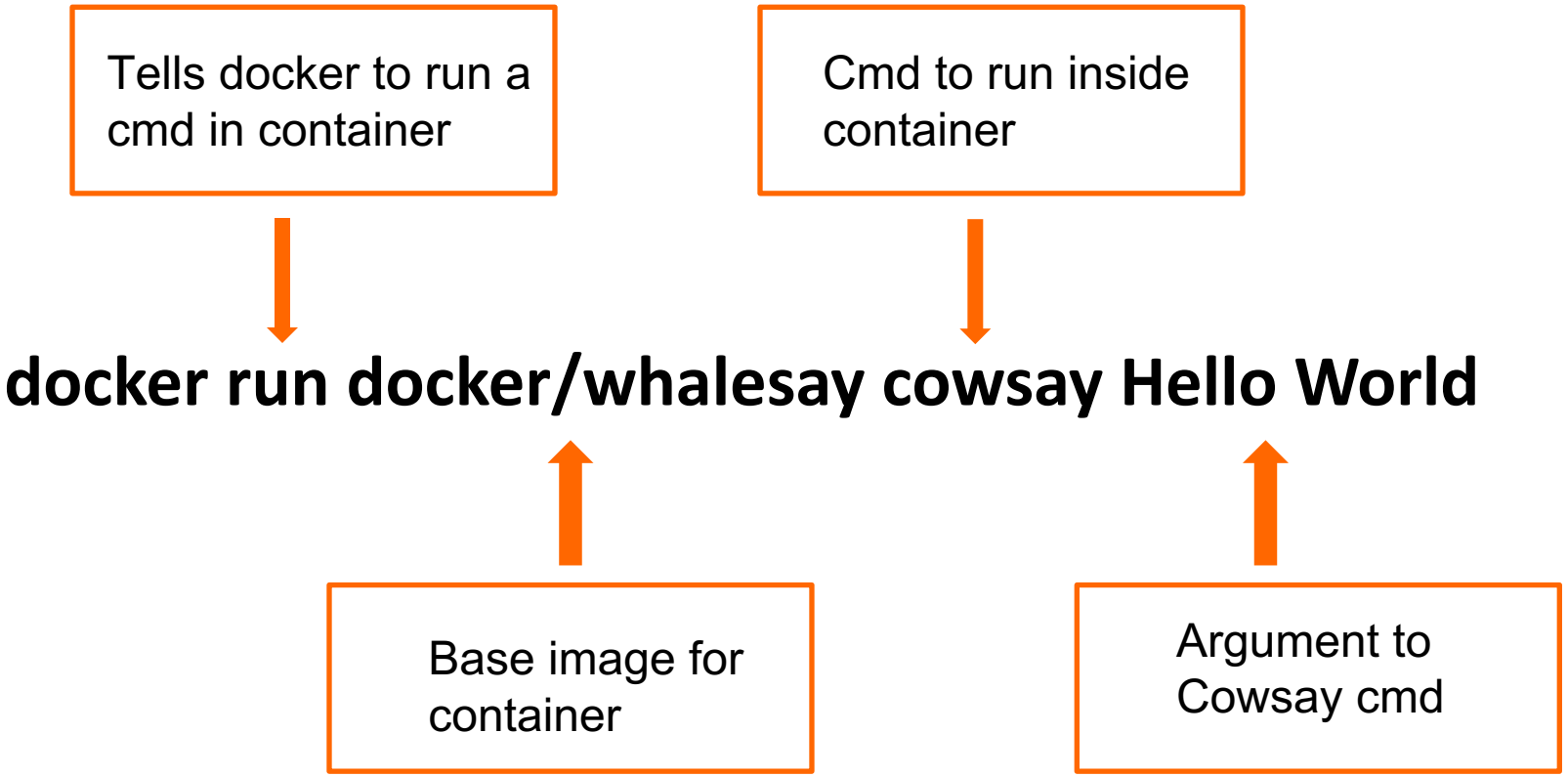


Docker Hello World(s)

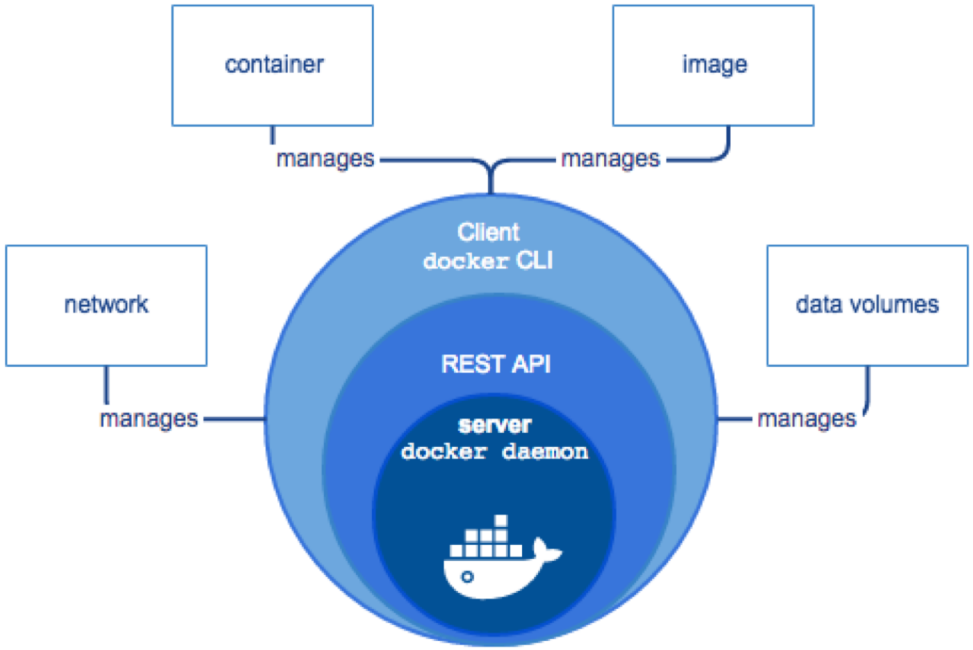
$ sudo docker run docker/whalesay cowsay Hello World



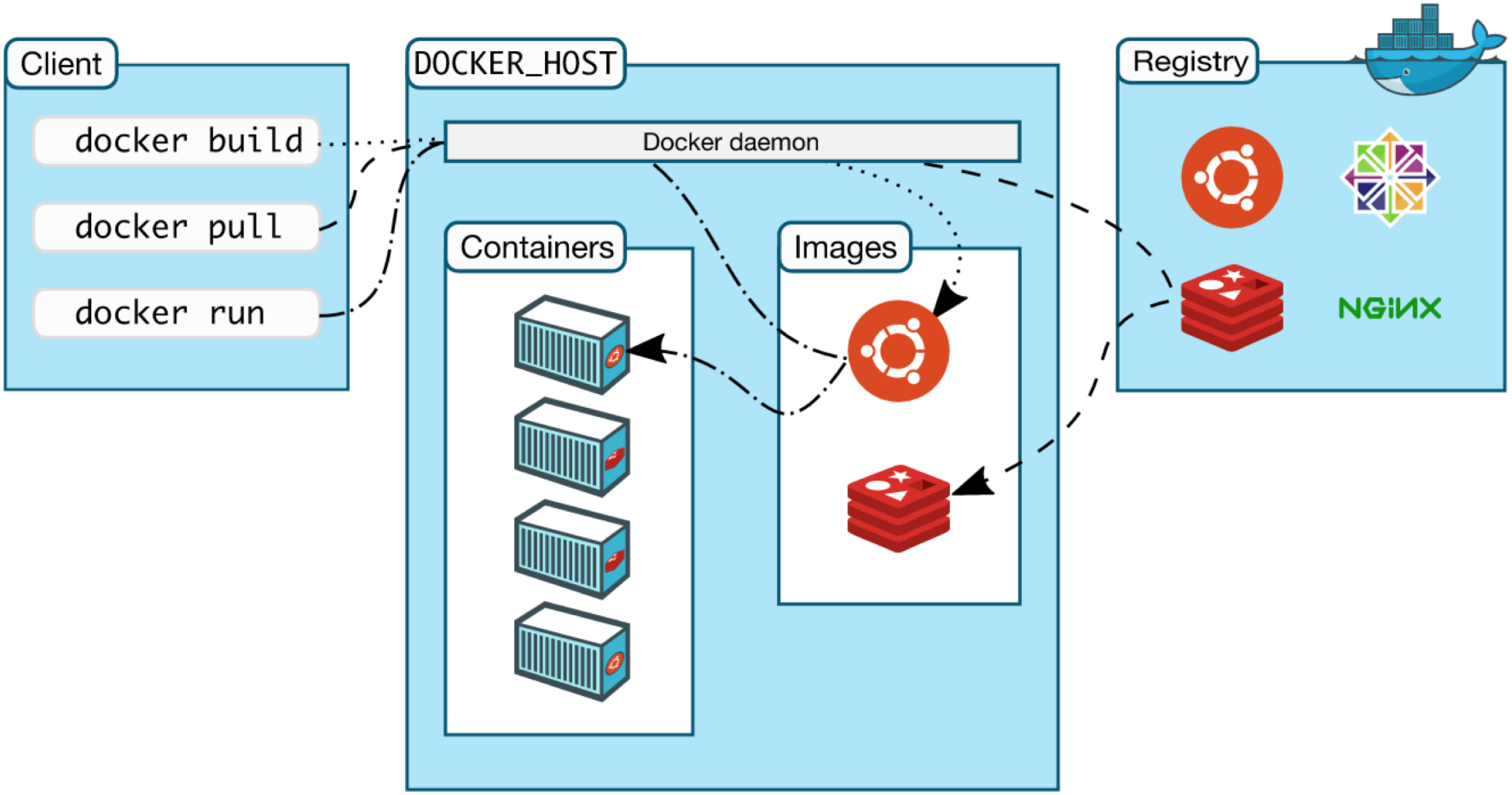
Docker Hello World(s)



What is ‘Docker Engine’?



How is Docker Architected?



Docker is Built on Foundational Tech

- **cgroup** and **namespacing** capabilities of the Linux kernel
- **Libcontainer Library Specification**
  - **(namespaces, filesystem, resources, security, etc)**
- **Go** programming language
  - (written in Go)
- **Docker Image** Specification
  - (for container image management)

No ‘sudo’ Configuration

Run docker without sudo

- https://docs.docker.com/engine/installation/linux/linux-postinstall/

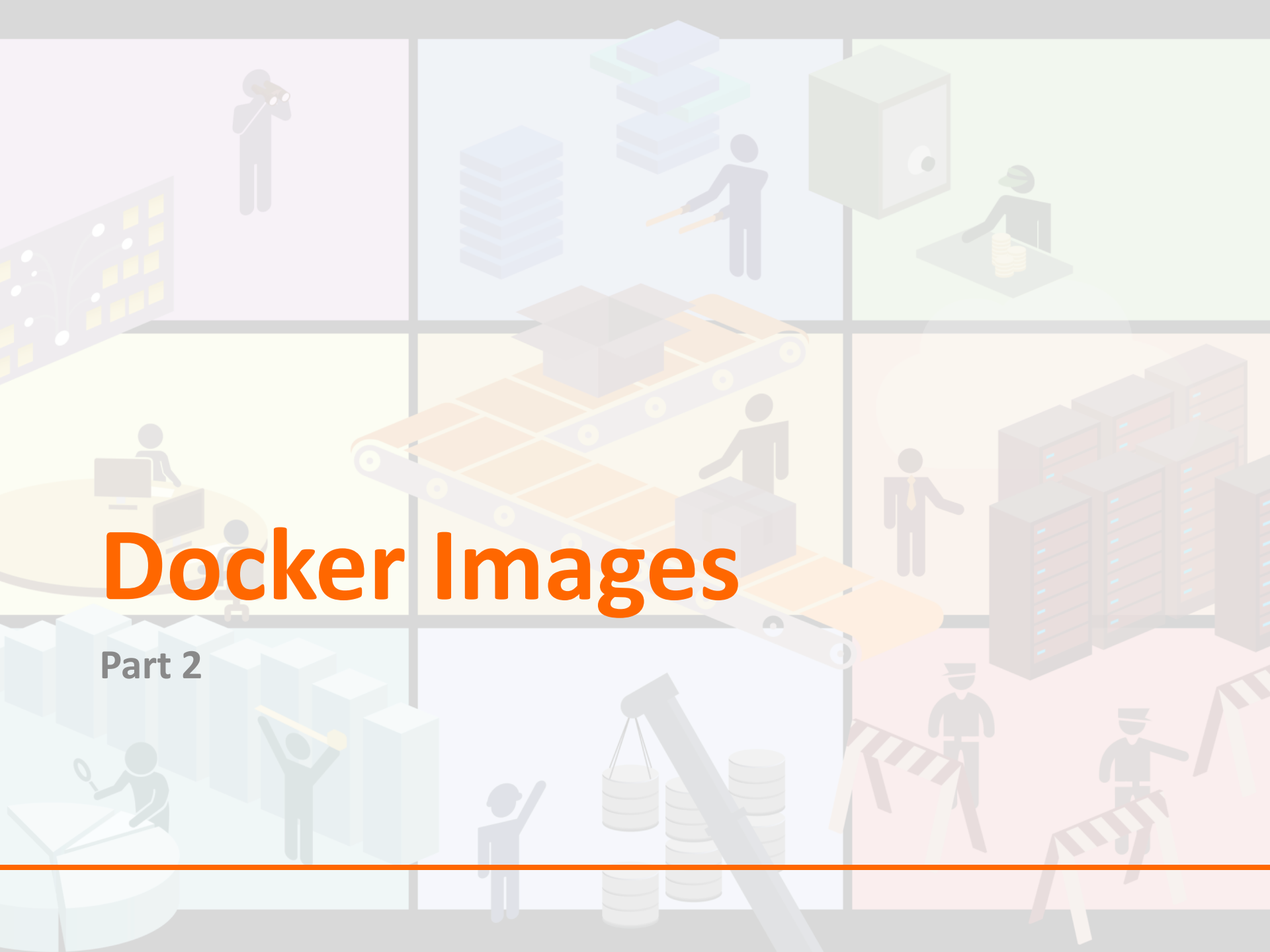
1. **Create Docker group:**
   - **$ sudo groupadd docker**
2. **Add your user to docker group:**
   - **$ sudo usermod -aG docker $USER**
3. **Log out and log back in:**
   - **$ exit # then log back in**

Docker Webapp & Hello World

CLASSROOM WORK

- **Exercise 2.1 in Docker Labs**

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise2.1-webapps.md



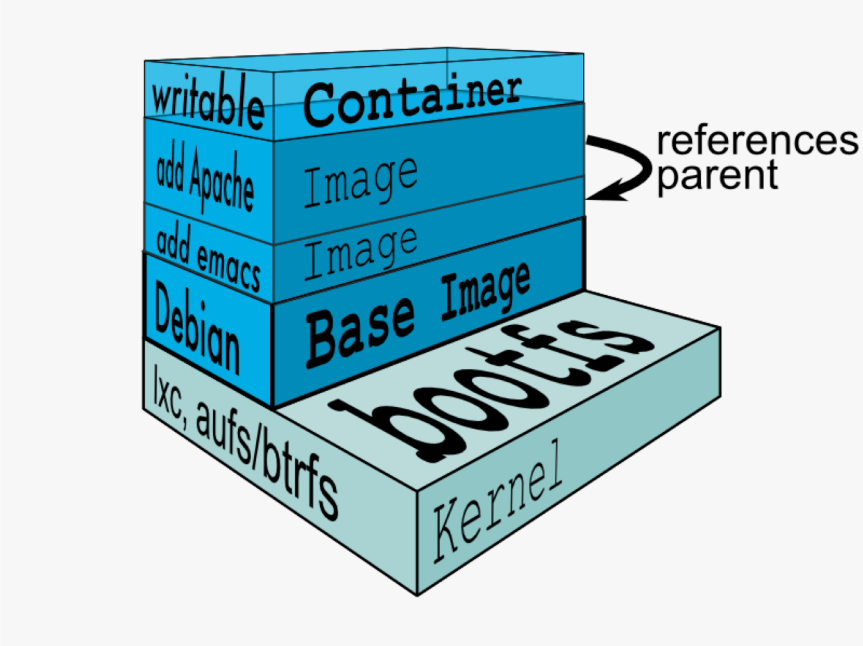
Docker Images

Part 2

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**

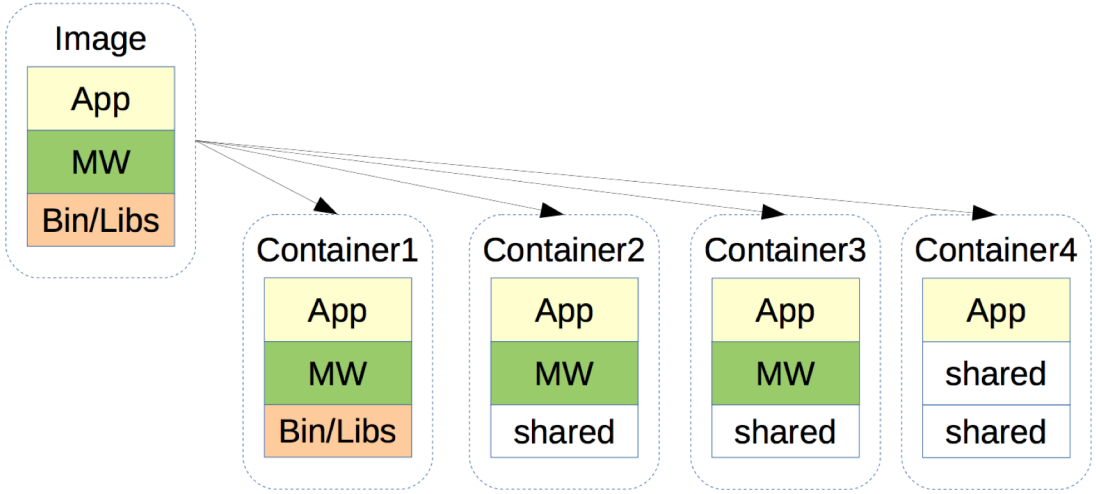


Example Docker Layers

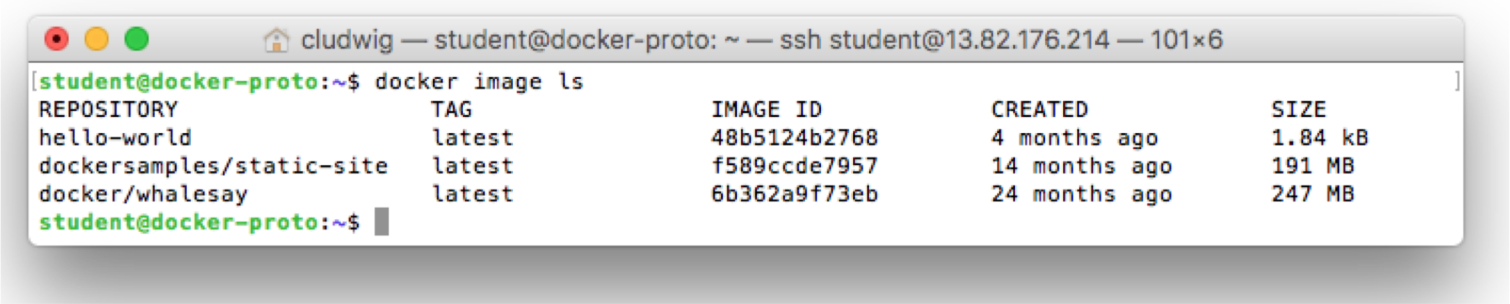
Docker Layer – Each Docker image is composed of a series of layers. Docker uses Union File Systems to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

Benefits of Union File Systems

- Files are shared across containers
- Storage and memory spaces are saved
- Faster deployment of containers



$ docker image ls

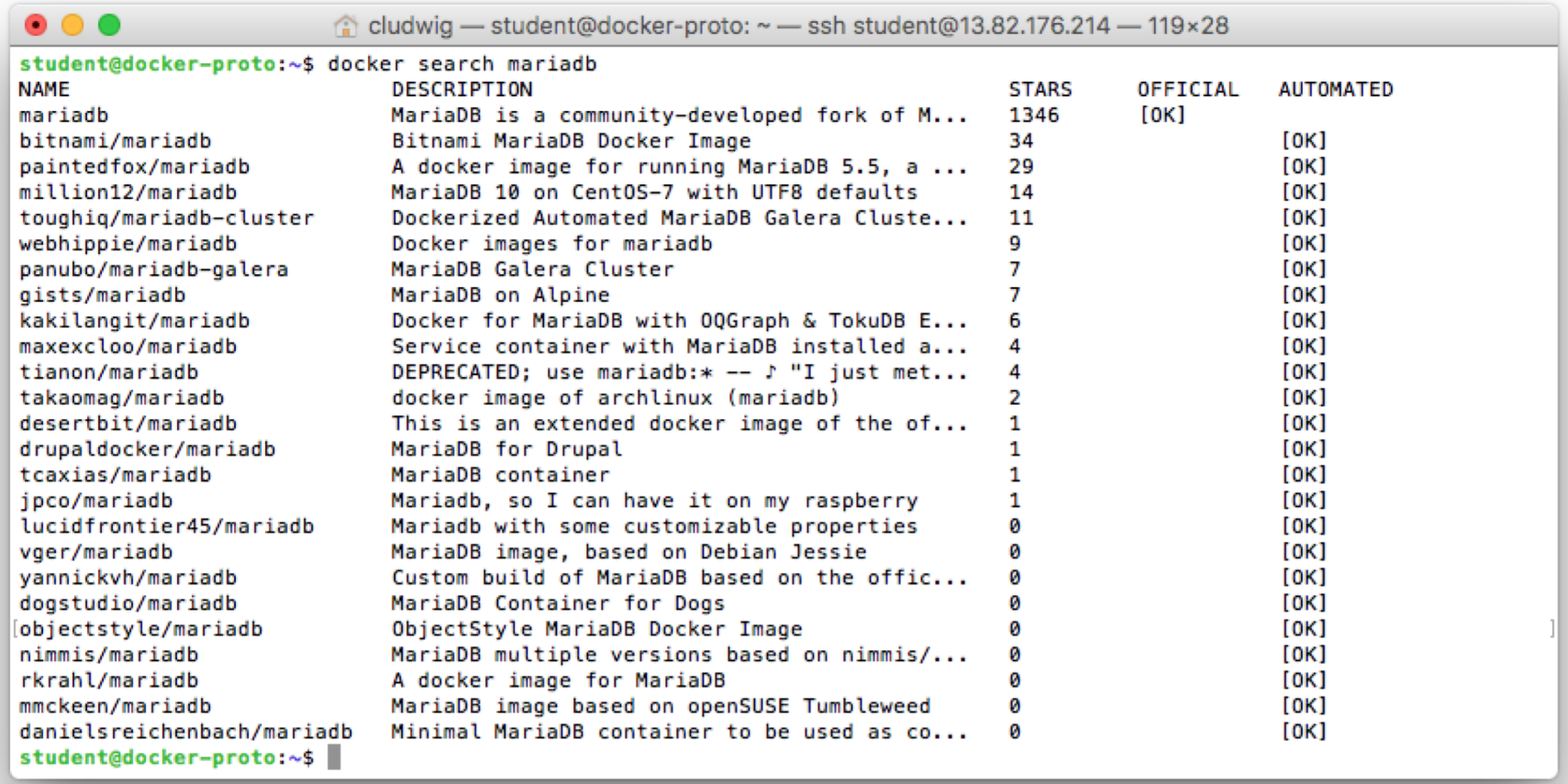


Docker Hub

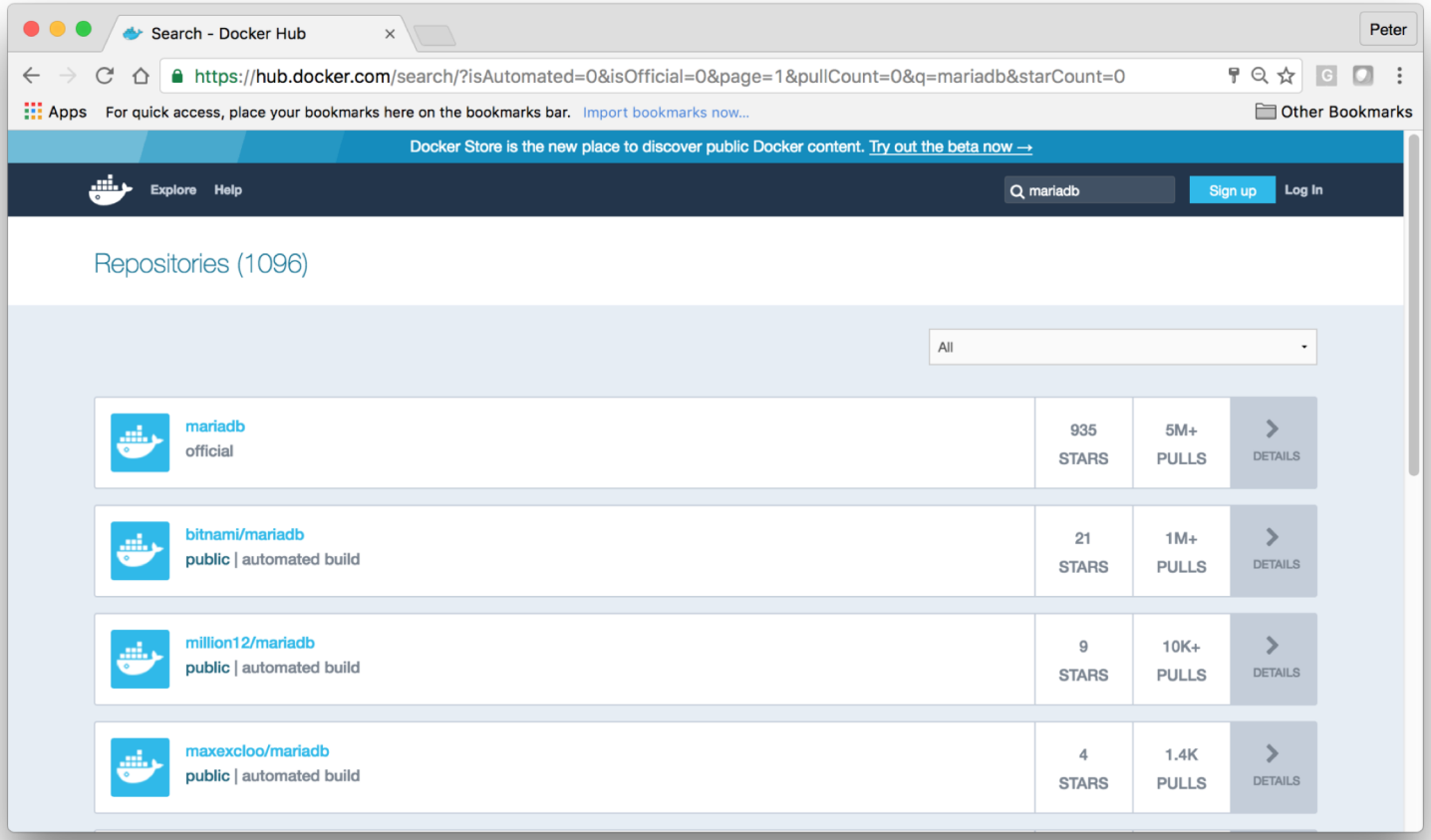
Docker's first-party cloud-based registry

- 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 search [searchterm]



Docker Hub Equivalent



$ docker image pull [imageName]

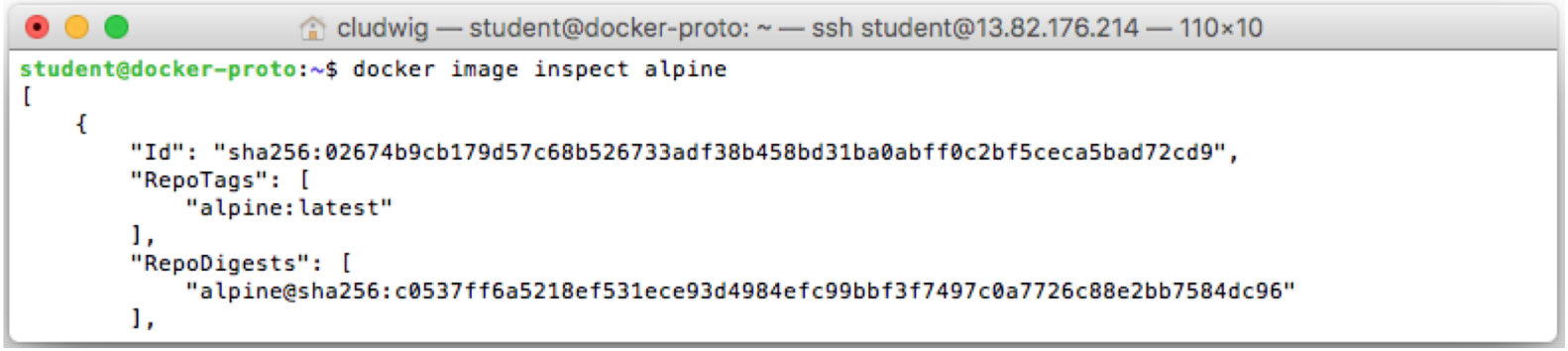
```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 119×6
[student@docker-proto:~$ docker image pull alpine
Using default tag: latest
latest: Pulling from library/alpine
cfc728c1c558: Pull complete
Digest: sha256:c0537ff6a5218ef531ece93d4984efc99bbf3f7497c0a7726c88e2bb7584dc96
Status: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

Best Practice: Choose the smallest base images possible

Debian – 123 MB
Alpine – 5 MB

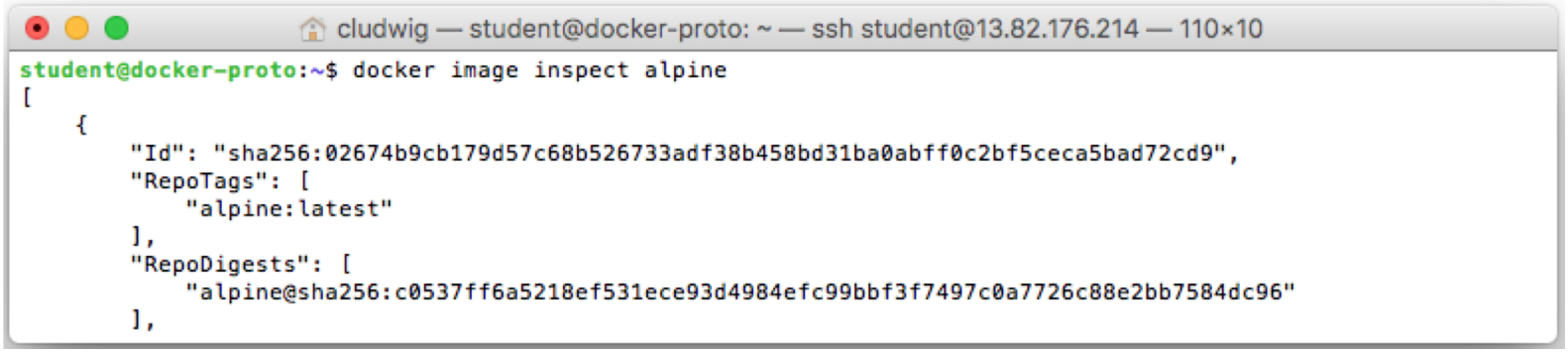
```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 119×7
[student@docker-proto:~$ docker image list
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
alpine                      latest              02674b9cb179        11 days ago         3.99 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 image inspect [imagename]



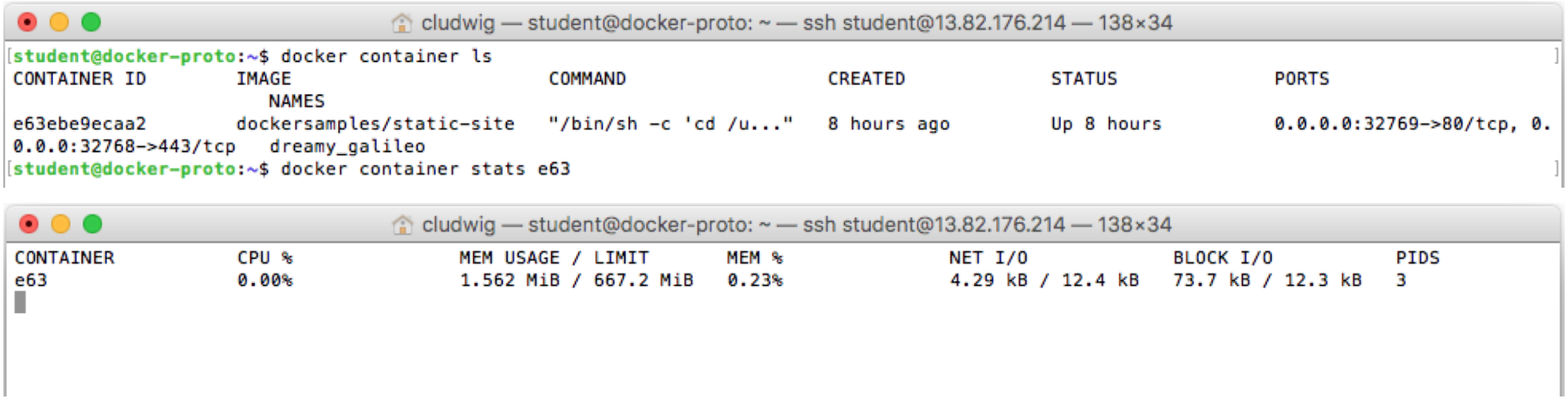
- ID The unique identifier for the container
- State This stanza has various status flags and the process id for the container. Using the ExitCode from this State element a graceful shutdown or recovery process could be initiated. The following format will return just the ExitCode of the most recently 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.

$ docker image inspect [imagename] (continued)



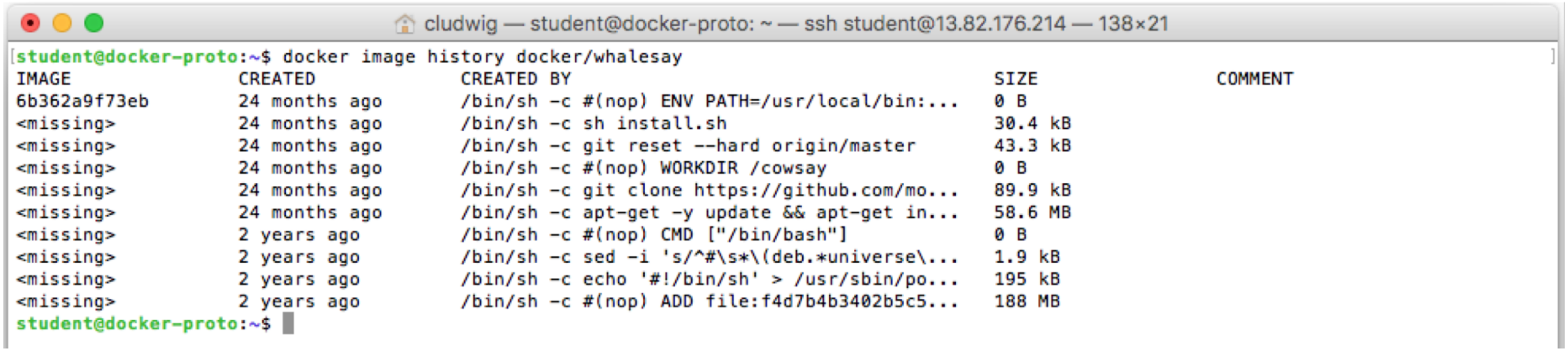
- LogPath The system path to this container's log file.
- RestartCount Keeps track of the number of times the container has been restarted. This value is the key value used when defining a container's restart policy..
- Name The user defined name for the container.
- Volumes Defines the volume mapping 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 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)**

$ docker image history docker/whalesay



Use history to view layers in an image

Removing Docker Containers & Images

#!/bin/bash

*# **Remove** all stopped containers*

docker container prune [-f]

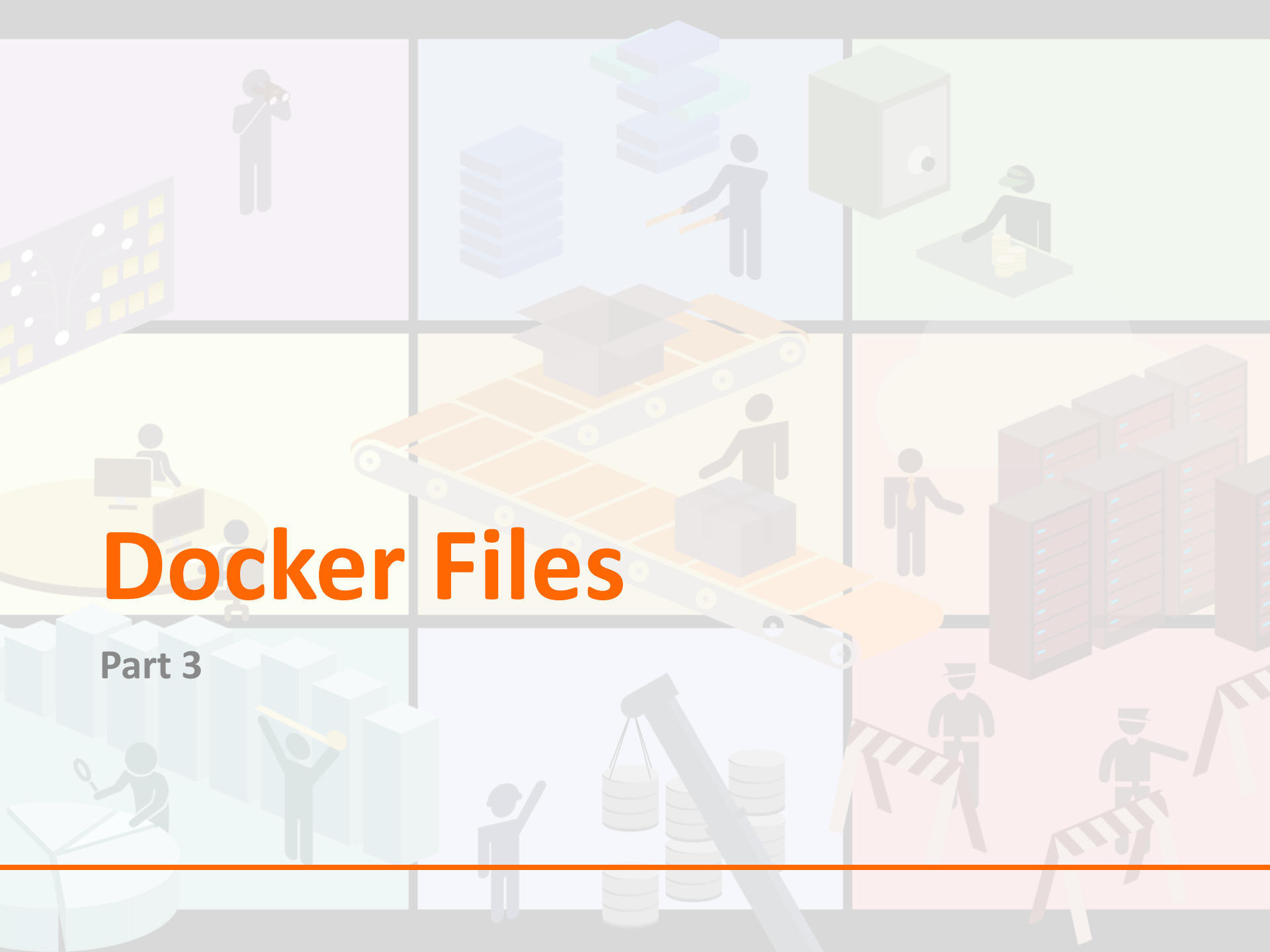
*# **Remove** all unused images*

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).

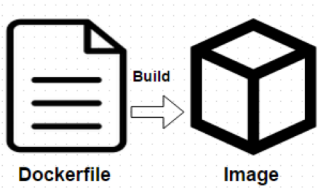
Best Practice: Prevent image inflation and regularly clean up images



Docker Files

Part 3

Dockerfile




```
FROM ubuntu:14.04

# install cowsay, and move the "default.cow" out of the way so we can overwrite it with "docker.cow"
RUN apt-get update && apt-get install -y cowsay --no-install-recommends && rm -rf /var/lib/apt/lists/* \
    && mv /usr/share/cowsay/cows/default.cow /usr/share/cowsay/cows/orig-default.cow

# "cowsay" installs to /usr/games
ENV PATH $PATH:/usr/games

COPY docker.cow /usr/share/cowsay/cows/
RUN ln -sv /usr/share/cowsay/cows/docker.cow /usr/share/cowsay/cows/default.cow

CMD ["cowsay"]
```

- Docker images are built, layer by layer, from a base image
- Images are composed of layers, each with a simple command such as
  - Run a shell command
  - Add a file or directory
  - Create an environment variable
  - Modify permissions
  - Execute process or script

Dockerfile

Dockerfile format

```
#Comment
INSTRUCTION arguments
```

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@northpole.org

ENV command

Used to provide environment variables for containers.

ENV <key> <value>
ENV AWS-KEY 123ABC12345ABCDEF

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.

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 pattern among development teams**
- **When possible, base from tiny images such as Alpine Linux**
- **Specify version in FROM statements when container is destined for production**

Dockerfile

- **Dockerfile build syntax**

  docker build .

  docker build -t <namespace/image name:versiontag> /path

  docker build -t myname/jenkins .

- **Then run the newly created image**

  docker run –t myname/jenkins

Dockerfile Exercises

CLASSROOM WORK

Exercise 2.2 & 2.3 in Docker Labs
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise2.1-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.

Dockerfile Optimization

3.1 CLASSROOM WORK

This exercise covers optimizing dockerfile to reduce the number of layers

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise3.1-dockerfile-optimization.md

Multi-Stage Build

- New as of Docker 17.05
- Allows “ship artifacts, not build environments”
- Ideal for languages like Go, where binaries are built in a heavyweight environment, but can ship the binary in a lightweight container.

```
# first stage does the building
# for UX purposes, I'm naming this stage `build-stage`

FROM golang:1.8 as build-stage
WORKDIR /go/src/github.com/codeship/go-hello-world
COPY hello-world.go .
RUN go build -o hello-world .

# starting second stage
FROM alpine:latest

# copy the binary from the `build-stage`
COPY --from=build-stage /go/src/github.com/codeship/go-hello-world/hello-world /bin

CMD hello-world
```

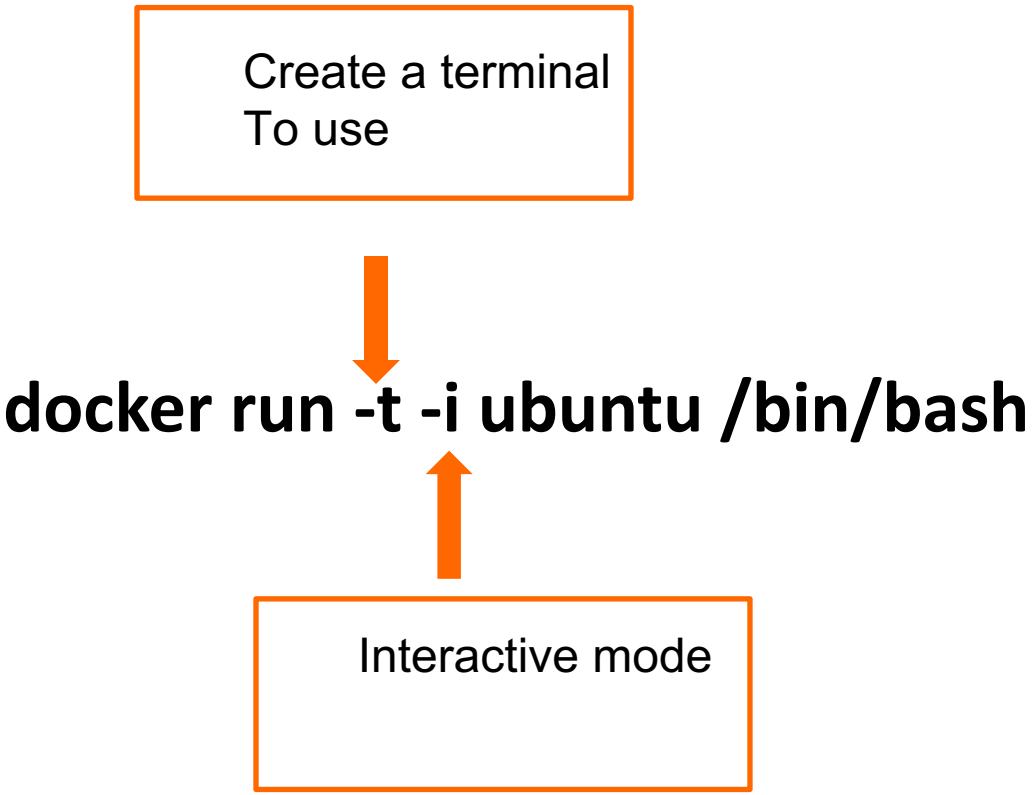
```
$ docker images

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
single-stage        latest              58328409dbf7        2 minutes ago       704MB
multi-stage         latest              9af3c2a2bf40        23 minutes ago      5.54MB
```

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

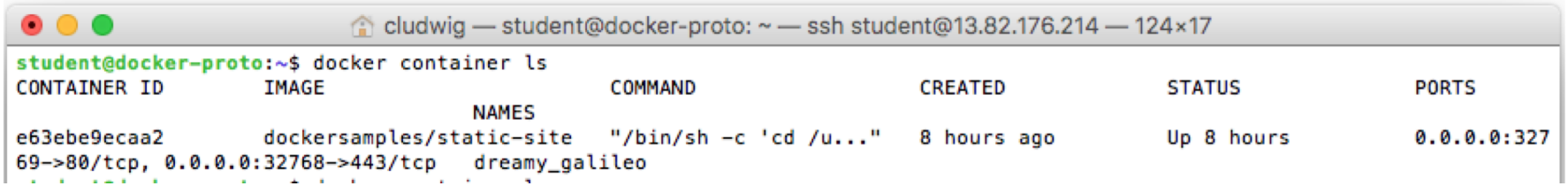
Docker Interactive Mode

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

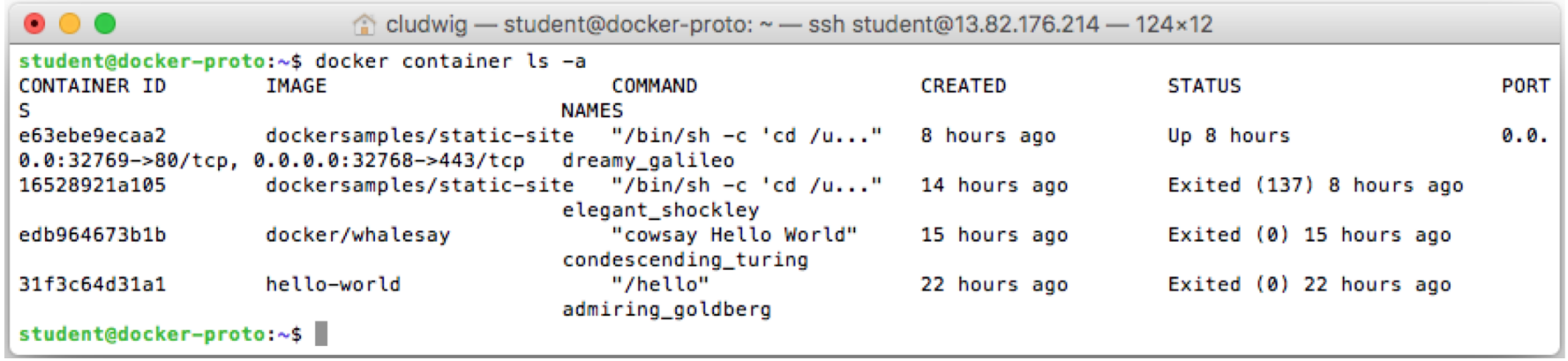
docker run -it alpine /bin/sh

See All Containers

- $ docker container ls #show running containers



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



Docker Hub/Ghost

CLASSROOM WORK

Assignment - Standup a ghost web server and connect to it with your browser

Docker Hub/MariaDB

CLASSROOM WORK

Assignment - Standup a mariaDB database and connect to it's command line prompt.

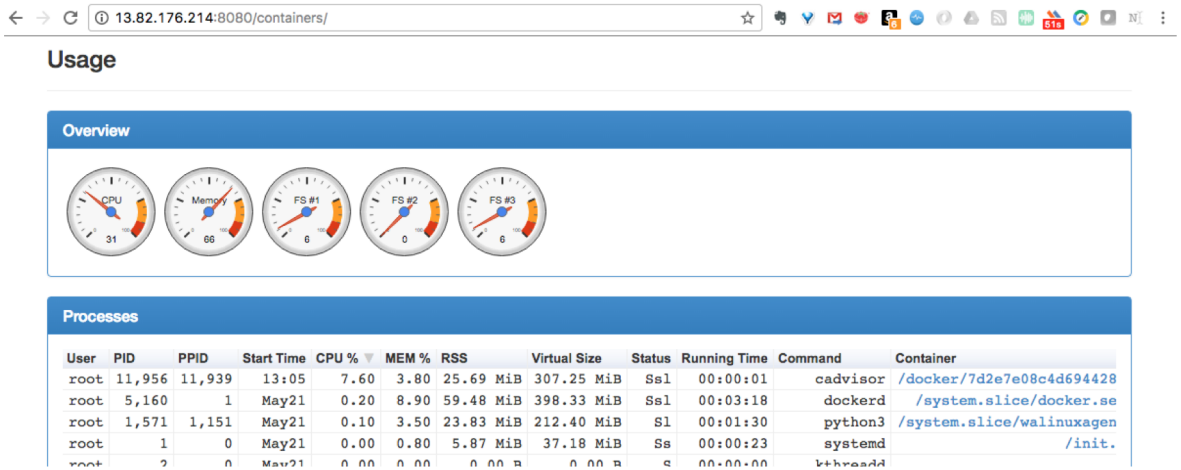
Bonus: Stand up a NOSQL database and connect to it (Cassandra, MongoDB, etc)

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=cadvisor google/cadvisor:latest

yourdockerhost:8080



Running a Private Docker Registry

- docker run -d -p5000:5000 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

- **Push a local image to a local registry**
  - **Docker push localhost:5000/<id>**
  - **https://github.com/docker/docker.github.io/blob/master/registry/deploying.md**

```
[ubuntu@instance-41:~/app$ sudo docker push localhost:5000/plamar
The push refers to a repository [localhost:5000/plamar]
88c7ceeb5d69: Pushed
fa69d2e74ebd: Pushed
4b80ed184035: Pushed
90b244cdf6f8: Pushed
e8c1de3c7027: Pushed
011b303988d2: Pushed
latest: digest: sha256:4dd3b61159aeeda583f480c542da468a68c63633d7e8a51e5645609701814e72 size: 1572
```

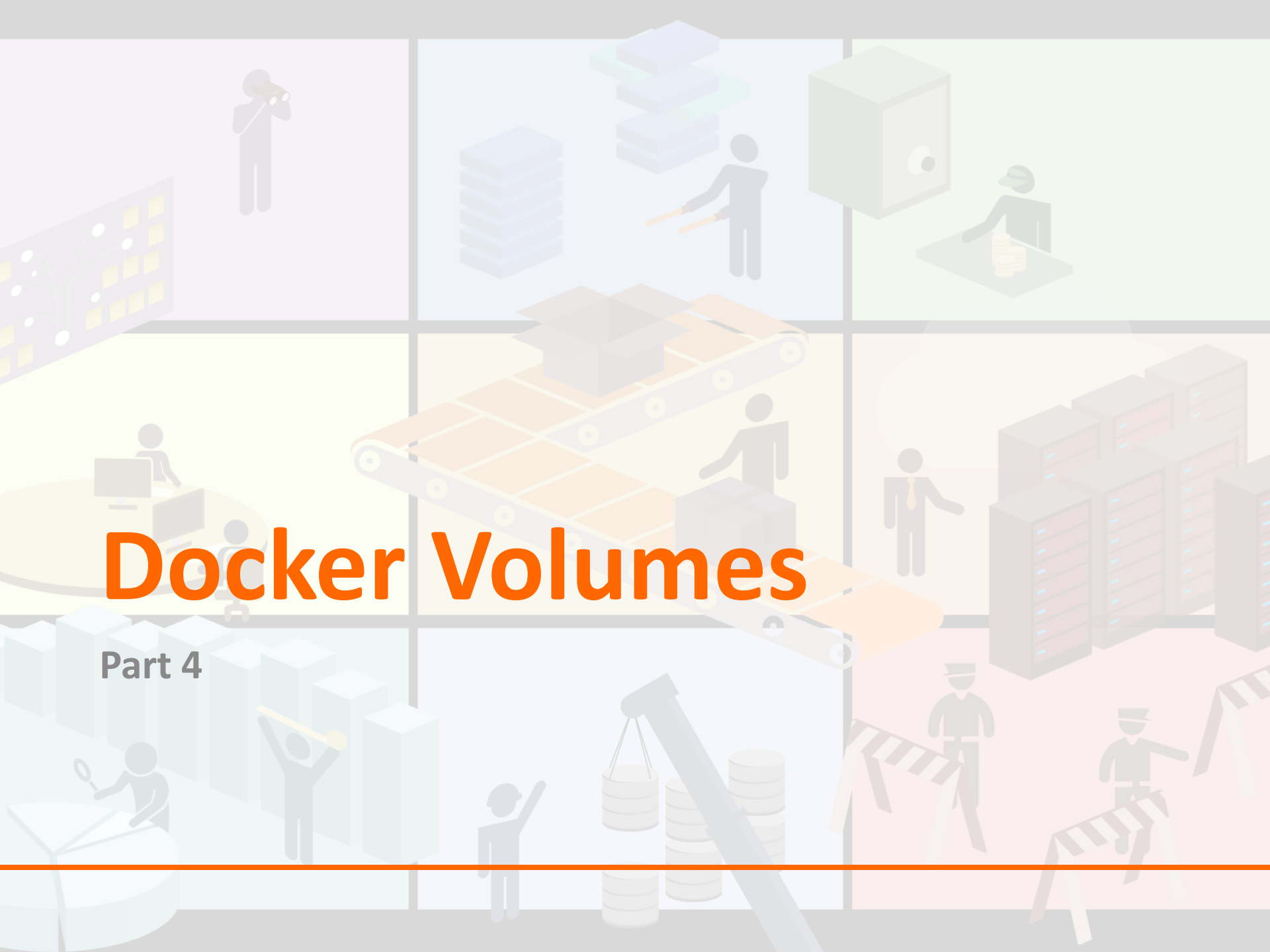
Docker Registry

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

```
ubuntu@instance-41:~/app$ curl -X GET http://localhost:5000/v2/_catalog
{"repositories":["plamar"]}
```

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

```
ubuntu@instance-41:~/app$ curl -X GET http://localhost:5000/v2/plamar/tags/list
{"name":"plamar","tags":["latest"]}
```



Docker Volumes

Part 4

Data Volumes

A *data volume* is a specially-designated directory within one or more containers that bypasses the *Union File System*. Data volumes provide several useful features for persistent or shared data:

- **Volumes are initialized when a container is created. If the container's base image contains data at the specified mount point, that existing data is copied into the new volume upon volume initialization. (Note that this does not apply when mounting a host directory.)**
- **Data volumes can be shared and reused among containers.**
- **Changes to a data volume are made directly.**
- **Changes to a data volume will not be included when you update an image.**
- **Data volumes persist even if the container itself is deleted.**

Data volumes are designed to persist data, independent of the container's lifecycle. Docker therefore *never* automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

Adding a Data Volume

- You can add a data volume to a container using the -v flag with the docker create and docker run command. You can use the -v multiple times to mount multiple data volumes. Now, mount a single volume in your web application container.

  - docker run -d -P --name web -v /webapp training/webapp python app.py

- This will create a new volume inside a container at /webapp.

- In addition to creating a volume using the -v flag you can also mount a directory from your Docker engine's host into a container

  - docker run -d -P --name web -v /src/webapp:/webapp training/webapp python app.py

- This command mounts the host directory, /src/webapp, into the container at /webapp.

Volume Persistence

Docker –v flag can mount the volume to a specific directory

```
ubuntu@instance-41:~$ docker run -v /volumetesting --name="persistdata" ubuntu /bin/sh -c "echo testing persistence w
ith volumes > /volumetesting/textfile.txt"

ubuntu@instance-41:~$ docker run --volumes-from=persistdata ubuntu /bin/sh -c "cat /volumetesting/textfile.txt"
testing persistence with volumes
```

Docker Volume Commands

Command	Description
docker volume create	Create a volume
docker volume ls	List volumes
docker volume inspect	Detailed information on volume
docker volume rm	Remove volume

Working with Volumes

```
ubuntu@instance-41:~$ docker volume create --name myvolume
myvolume
ubuntu@instance-41:~$ docker volume ls
DRIVER              VOLUME NAME
local               030f652ad7ea6766f3d17a421233896b78296cf89d9bc816c69e6084df1aa24d
local               94ed6b916617b932638619fcbdbfc77d843175a57c520d02d3162f5750916409
local               myvolume
ubuntu@instance-41:~$ docker volume inspect myvolume
[
    {
        "Name": "myvolume",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/myvolume/_data",
        "Labels": {},
        "Scope": "local"
    }
]
ubuntu@instance-41:~$ docker volume rm myvolume
myvolume
```

Dangling Volumes

- If container is deleted with a volume attached, the volume remains. Sometimes removing all such ‘dangling’ volumes is desired
  - $ docker volume prune

Dangling Volumes

- **Before**

```
ubuntu@instance-41:~$ docker volume ls
DRIVER              VOLUME NAME
local               030f652ad7ea6766f3d17a421233896b78296cf89d9bc816c69e6084df1aa24d
local               1e16af70af2319707350ce672c0af925c010561a5f75f83393674b5f20df7074
local               94ed6b916617b932638619fcbdbfc77d843175a57c520d02d3162f5750916409
```

- **After**

```
ubuntu@instance-41:~$ docker volume prune
030f652ad7ea6766f3d17a421233896b78296cf89d9bc816c69e6084df1aa24d
1e16af70af2319707350ce672c0af925c010561a5f75f83393674b5f20df7074
94ed6b916617b932638619fcbdbfc77d843175a57c520d02d3162f5750916409
ubuntu@instance-41:~$ docker volume ls
DRIVER              VOLUME NAME
```

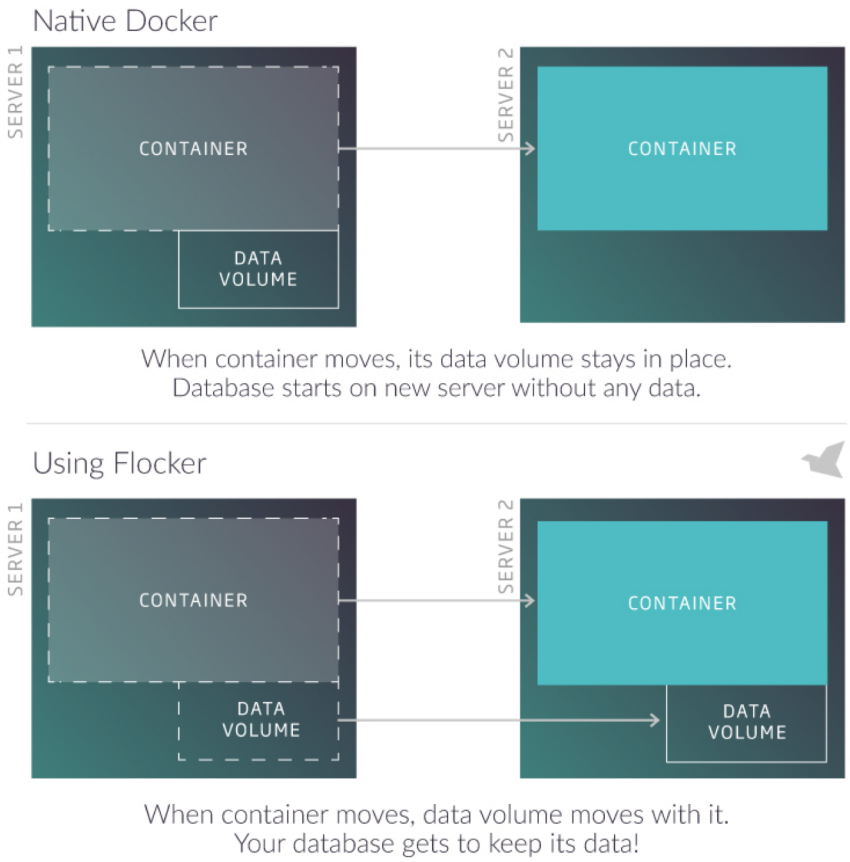
Docker Volumes Exercise

4.1 CLASSROOM WORK

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise4.1-docker-volumes.md

Docker Volume Plugin - Flocker

https://clusterhq.com/flocker/



Docker Cheat Sheet

docker Cheat Sheet

REBELLABS by ZEROTURNAROUND

Glossary

Layer - a set of read-only files to provision the system

Image - a read-only layer that is the base of your container. Might have a parent image

Container - a runnable instance of the image

Registry / Hub - central place where images live

Docker machine - a VM to run Docker containers (Linux does this natively)

Docker compose - a utility to run multiple containers as a system

Useful one-liners

Download an image
```
docker pull image_name
```

Start and stop the container
```
docker [start|stop] container_name
```

Create and start container, run command
```
docker run -ti --name container_name
    image_name command
```

Create and start container, run command, destroy container
```
docker run --rm -ti image_name command
```

Example filesystem and port mappings
```
docker run -it --rm -p 8080:8080 -v
/path/to/agent.jar:/agent.jar -e
JAVA_OPTS="-javaagent:/agent.jar"
tomcat:8.0.29-jre8
```

Docker cleanup commands

Kill all running containers
```
docker kill $(docker ps -q)
```

Delete dangling images
```
docker rmi $(docker images -q -f
    dangling=true)
```

Remove all stopped containers
```
docker rm $(docker ps -a -q)
```

Docker machine commands

Use docker-machine to run the containers

Start a machine
```
docker-machine start machine_name
```

Configure docker to use a specific machine
```
eval "$(docker-machine env machine_name)"
```

Docker compose syntax

docker-compose.yml file example
```
version: "2"
services:
web:
  container_name: "web"
  image: java:8 # image name
  # command to run
  command: java -jar /app/app.jar
  ports: # map ports to the host
    - "4567:4567"
  volumes: # map filesystem to the host
    - ./myapp.jar:/app/app.jar
mongo: # container name
  image: mongo # image name
```

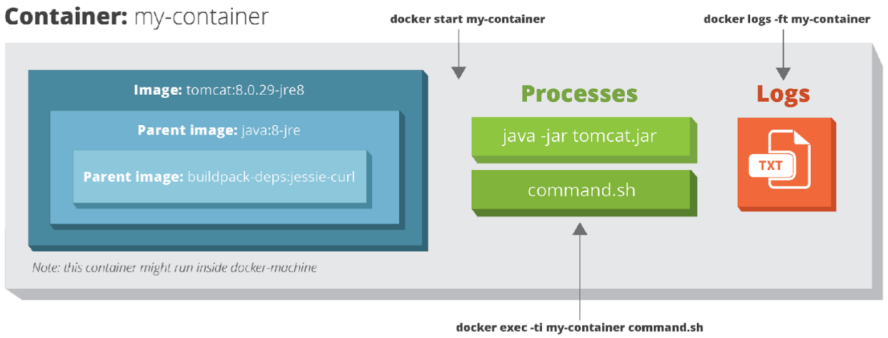
Create and start containers
```
docker-compose up
```

Interacting with a container

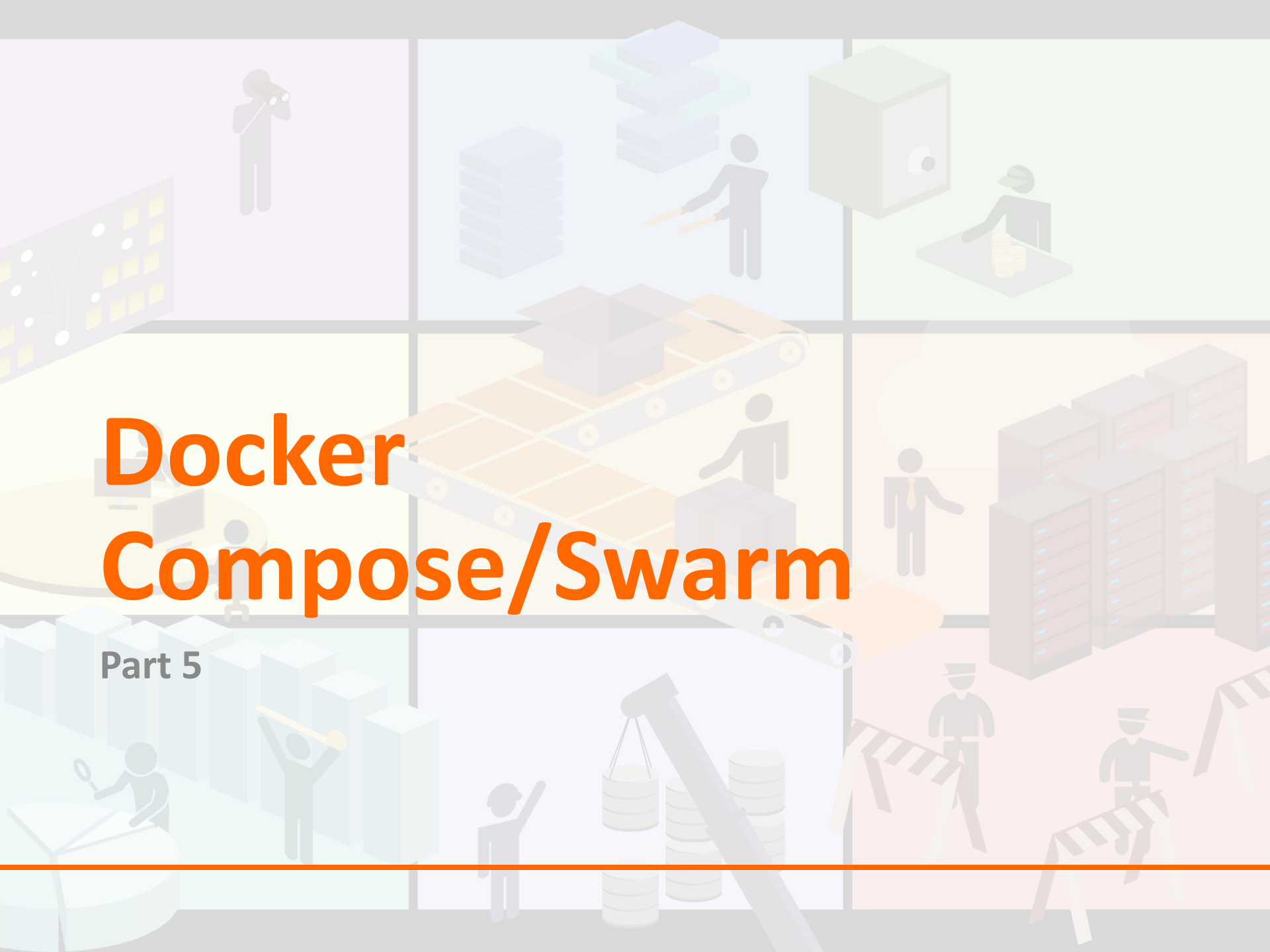
Run a command in the container
```
docker exec -ti container_name command.sh
```

Follow the container logs
```
docker logs -ft container_name
```

Save a running container as an image
```
docker commit -m "commit message" -a "author"
    container_name username/image_name:tag
```



BROUGHT TO YOU BY JRebel



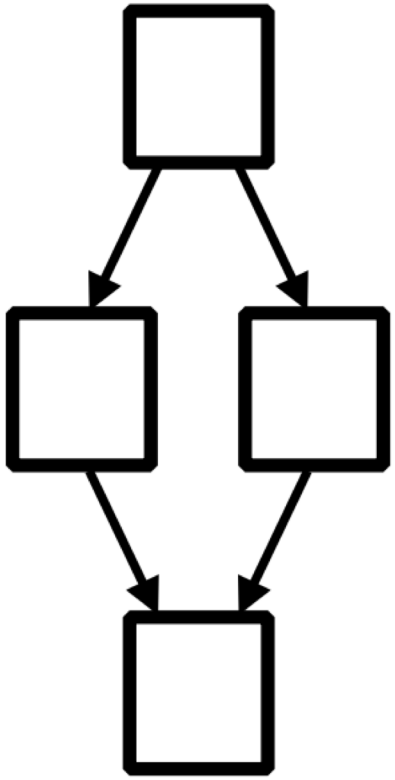
Docker Compose/Swarm

Part 5

Launching Multiple Containers is Clumsy

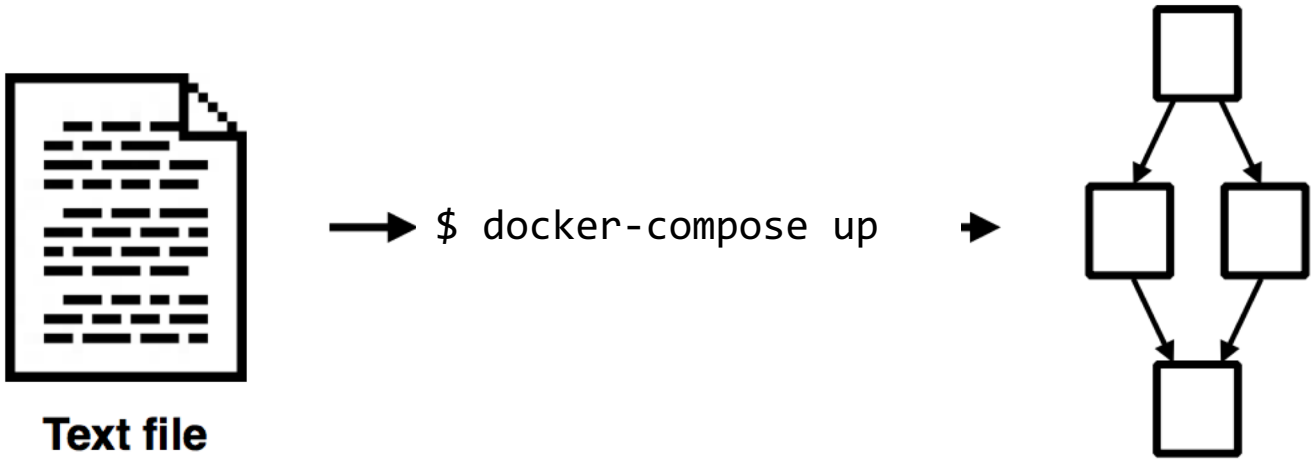
```
$ docker pull redis:latest

$ 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
```

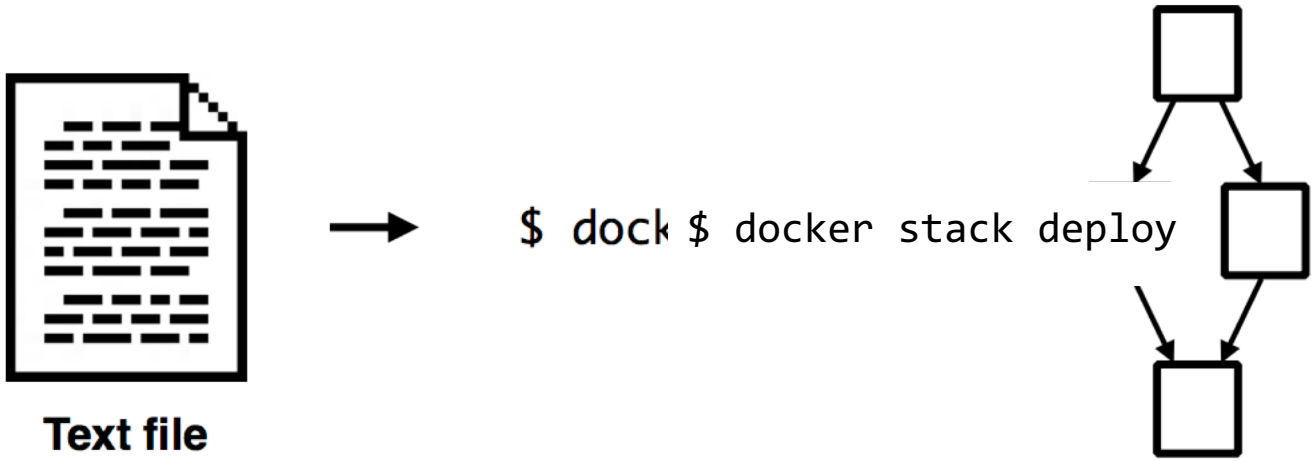
Launching Multiple Containers is Clumsy



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

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

Docker Compose or Swarm Launches ap





Docker Compose Commands

Command	Description
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

5.1 CLASSROOM WORK

First, install Compose:

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

```
$ curl -L https://github.com/docker/compose/releases/download/1.23.2/docker-compose-
`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
$ chmod +x /usr/local/bin/docker-compose
```

Then, Compose "Getting Started" Lab:

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise5.1-docker-compose.md

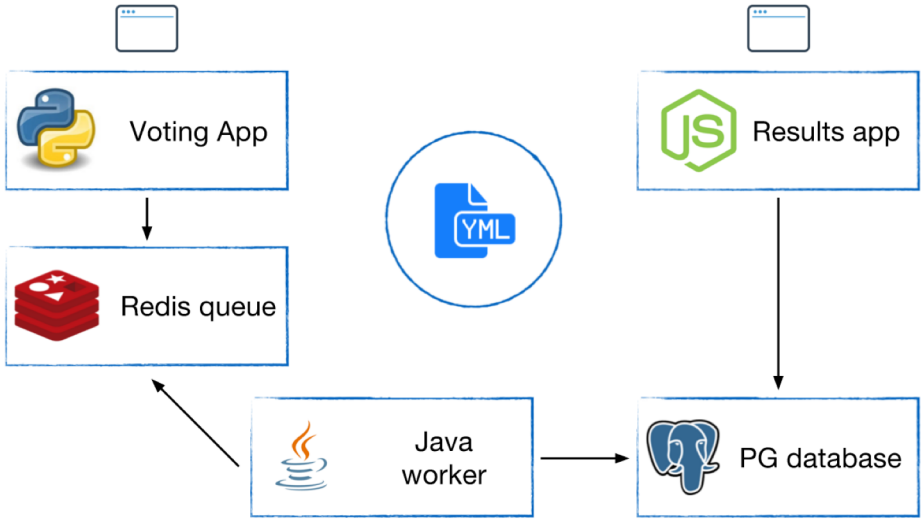
Docker Swarm Commands

Command	Description
docker swarm init	Create a swarm
docker stack deploy	Deploy the swarm
docker stack services	Show swarm services
docker stack rm	Remove stack

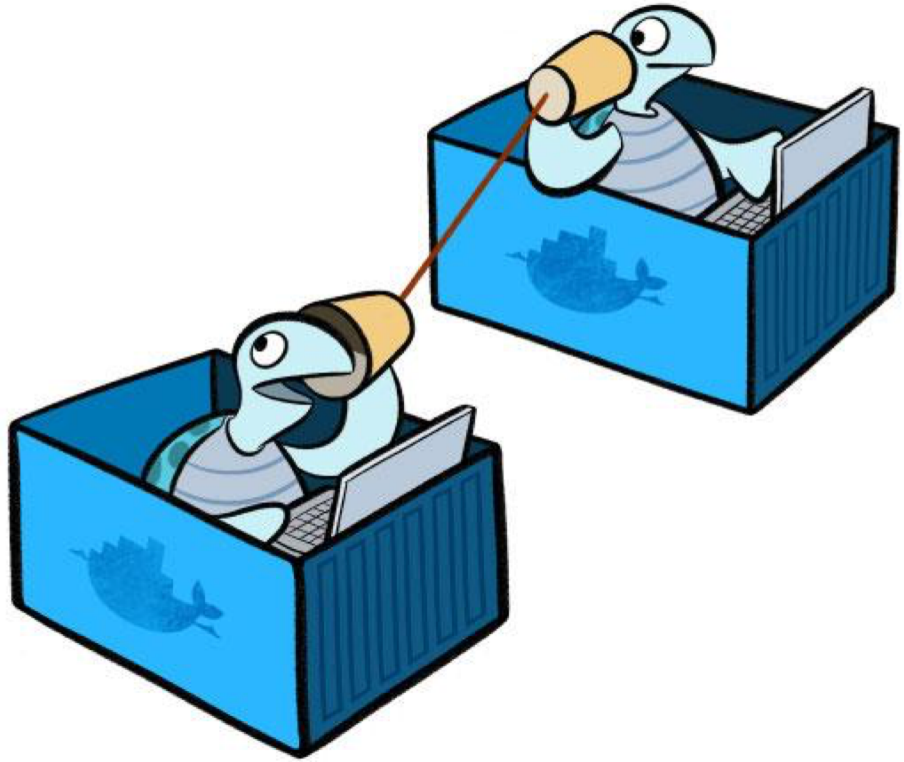
Docker Swarm to Deploy a Cluster

5.2 CLASSROOM WORK

Exercise 3.0 in Docker Labs
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise5.2-voting-app.md

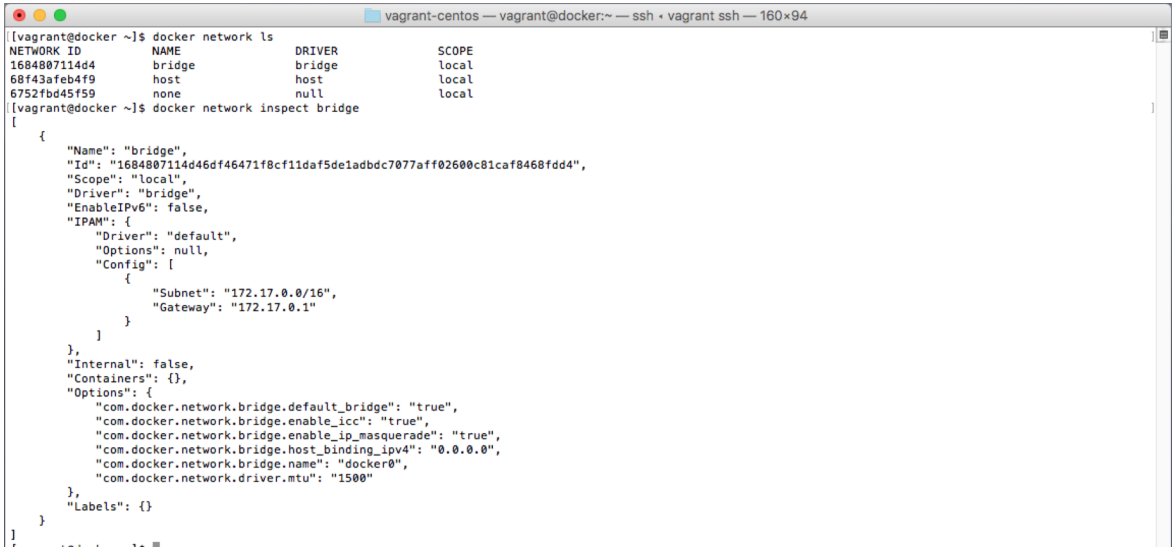


Docker Networking

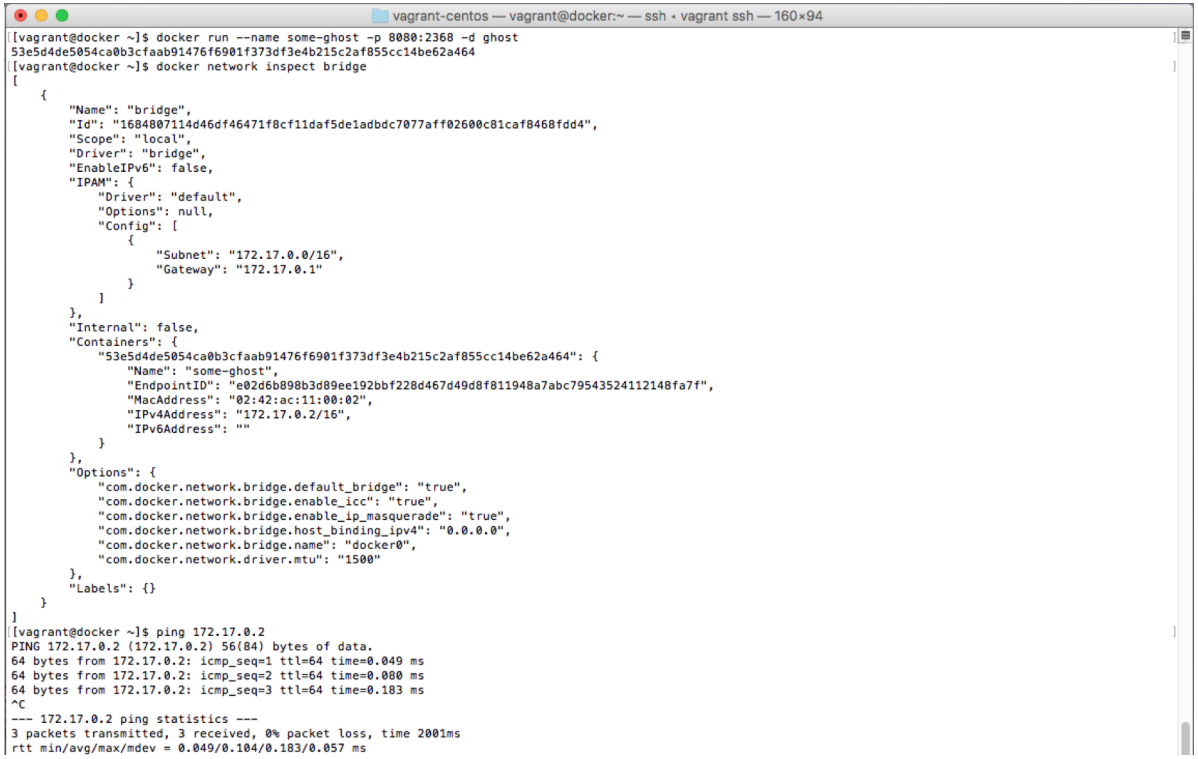


Docker Network Defaults

- Three networks are created by default (bridge, host & none)
  - docker network ls
- We can easily inspect the bridge network
  - docker network inspect bridge
- We see that a 172.17.0.0/16 address space is allocated. Created Containers are given an IP in this space by default.



Docker Network Defaults



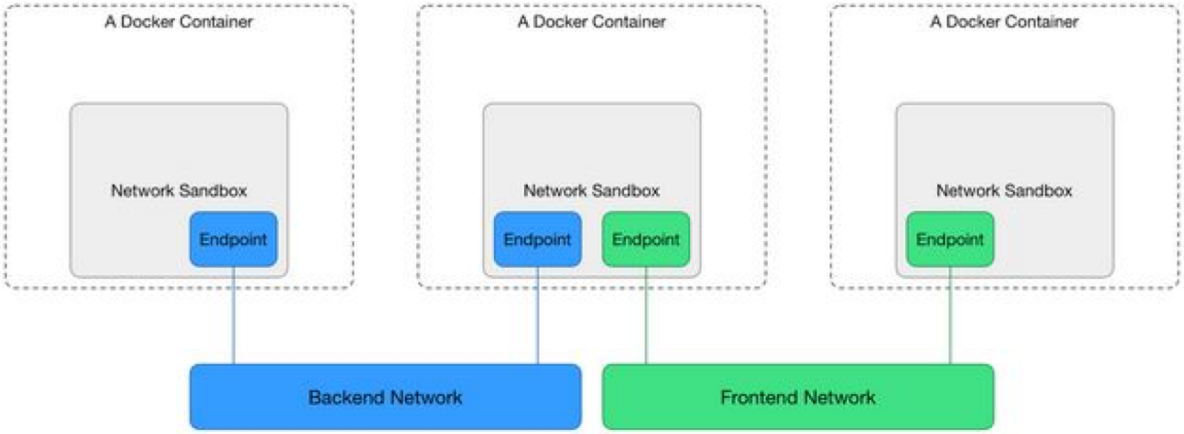
- Created container is attached to bridge network by default
  - Can be attached to user created networks as well with –network flag
- Container is assigned 172.17.0.2 ip in the bridge network and appears in the docker inspect command
- The container can then be pinged at 172.17.0.2

Docker Network Commands

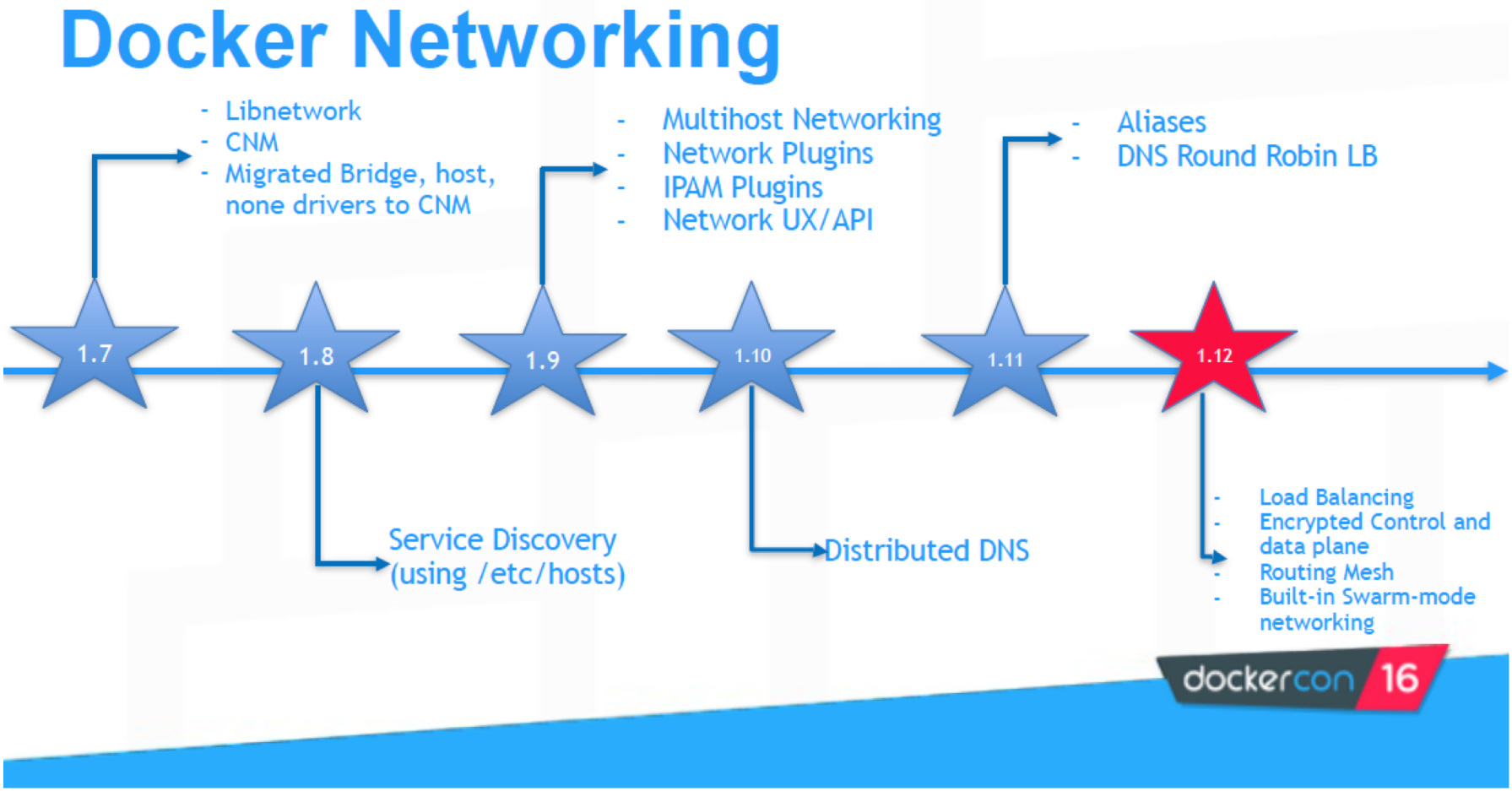
Command	Description
docker network create	Create a network
docker network ls	List networks
docker network inspect	Detailed information on network
docker network rm	Remove network
docker network disconnect	Disconnect container from network
docker network connect	Connect container to network

Isolated User Networks

- The bridge network is useful in a development environment
- Production environments usually require more security and isolated or more customized networking solutions.
- Docker network (libnetwork) and the swarm mode overlay network type provide the building blocks to do so.



Docker Network Releases



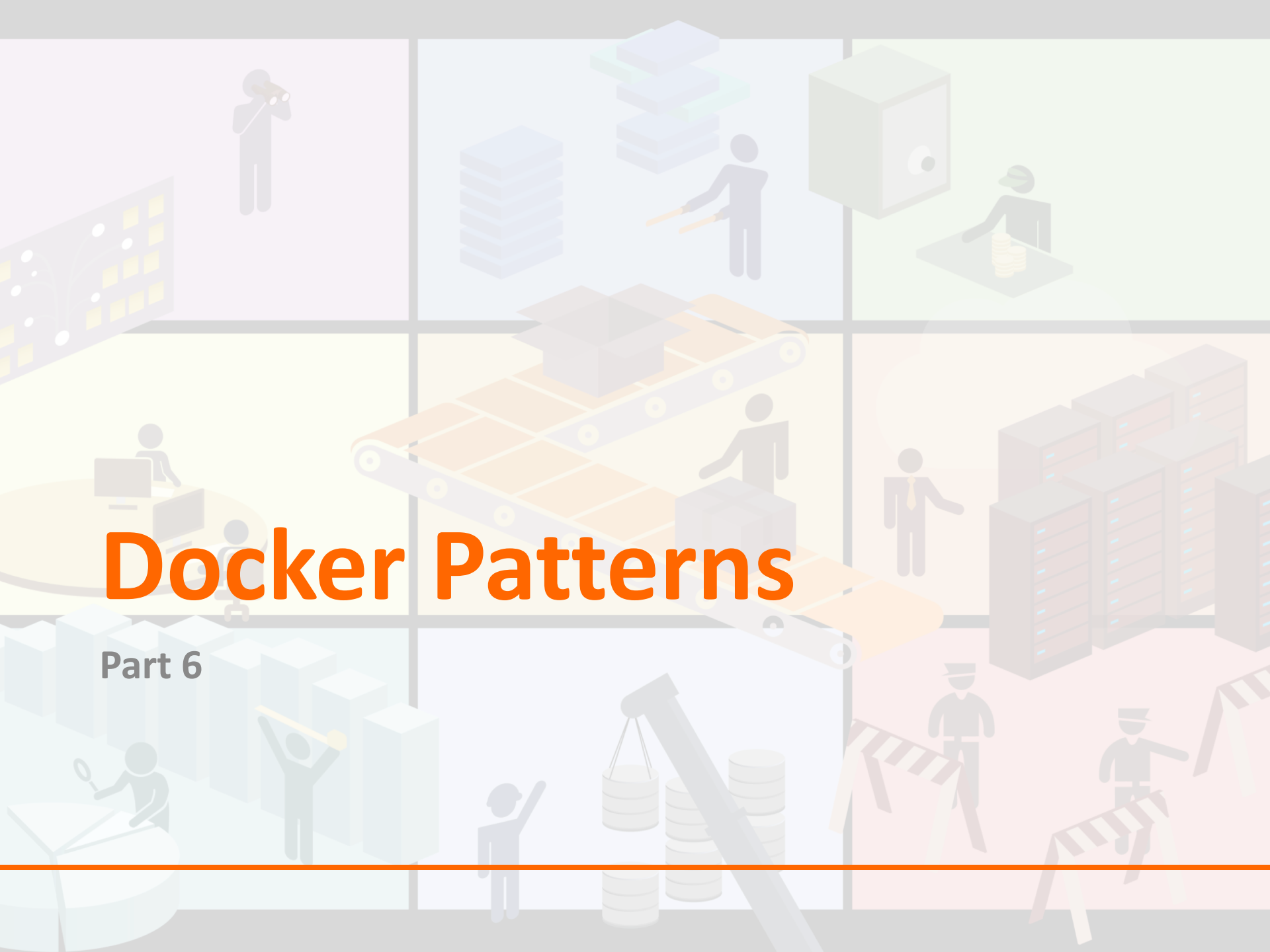
Docker Network Releases: New in 1.13+

- Add --attachable network support to enable docker run to work in swarm-mode overlay network
- Add support for host port PublishMode in services using the --publish option in docker service create

Docker Overlay Networks

- Overlay networking for Docker Engine swarm mode comes secure out of the box. By default all nodes encrypt and authenticate network information they exchange using the AES algorithm in GCM mode. Manager nodes in the swarm rotate the key used to encrypt gossip data every 12 hours.
- You can also encrypt data exchanged between containers on different nodes on the overlay network. When you enable overlay encryption, Docker creates IPSEC tunnels between all relevant* nodes attached to the overlay network. These tunnels also use the AES algorithm in GCM mode and manager nodes automatically rotate the keys every 12 hours.
- To enable encryption, when you create an overlay network pass the --opt encrypted flag:

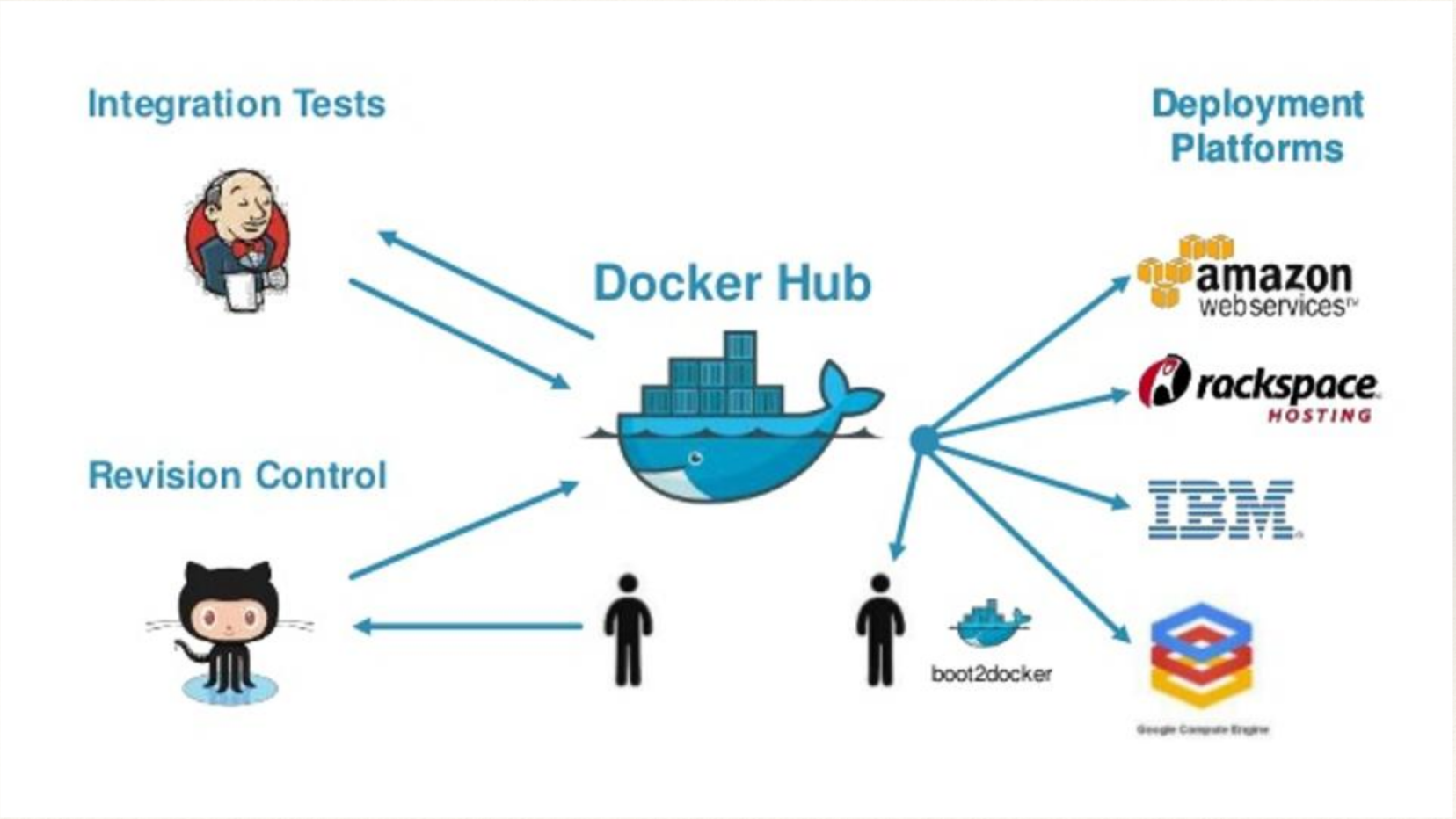
  $ docker network create --opt encrypted --driver overlay my-multi-host-network



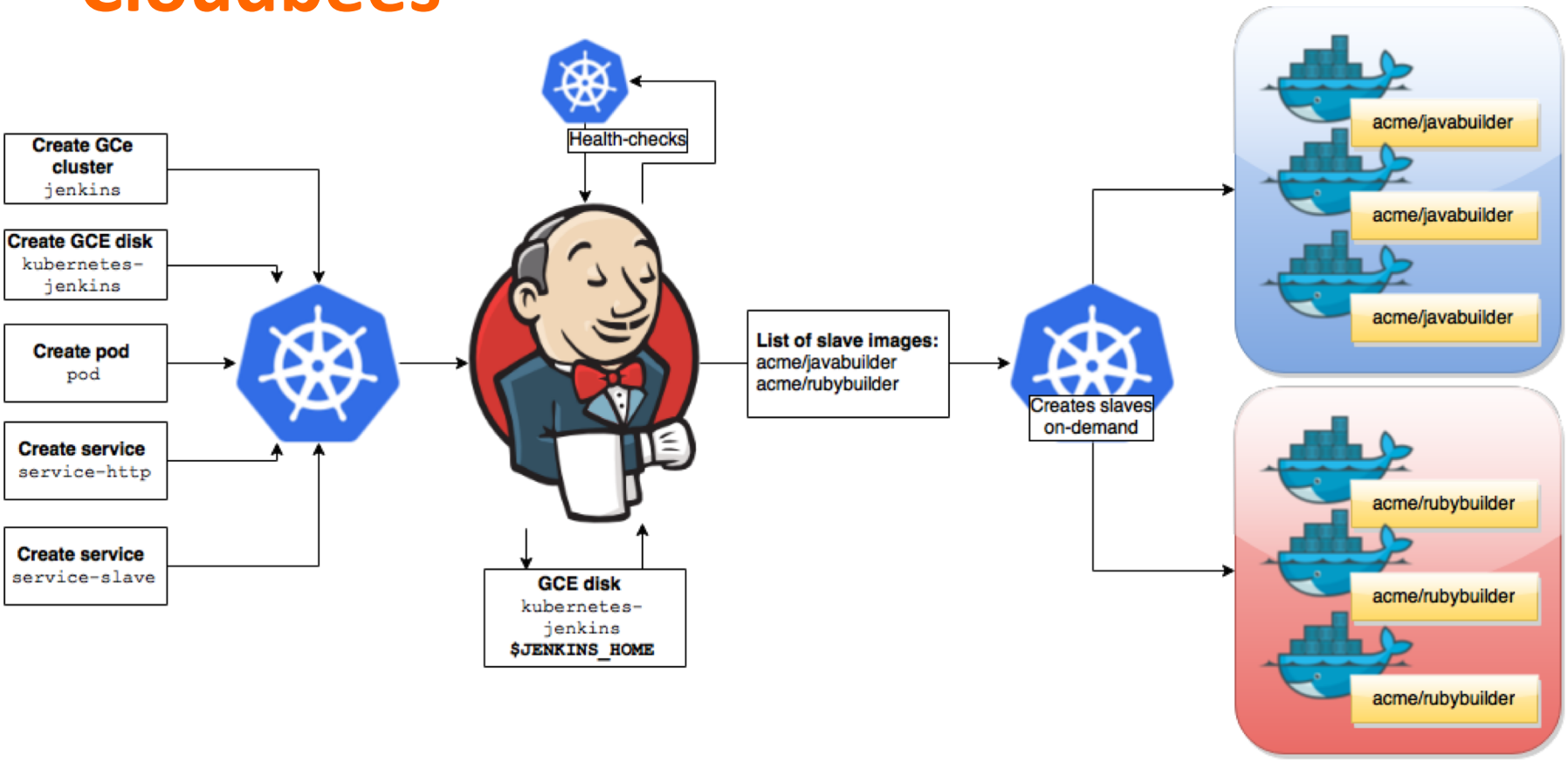
Docker Patterns

Part 6

Docker Patterns

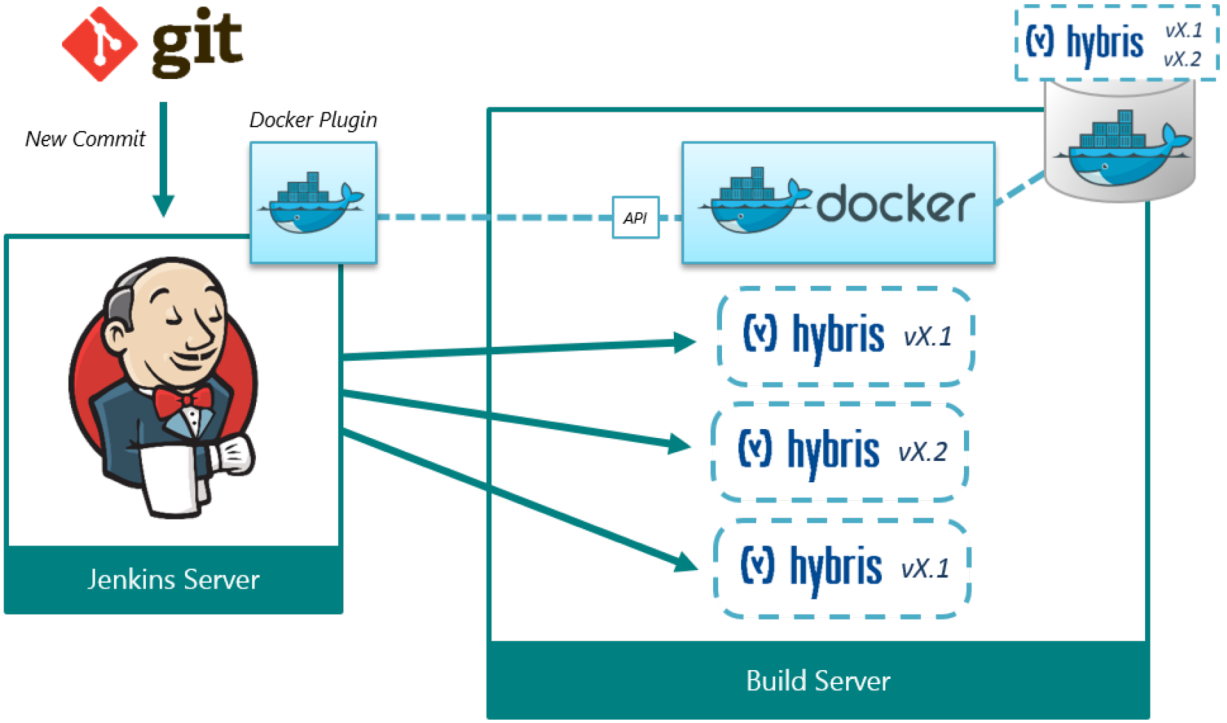


Jenkins Continuous Integration- Cloudbees



Link to Blog: Clustering Jenkins
http://bit.ly/2a092Xo

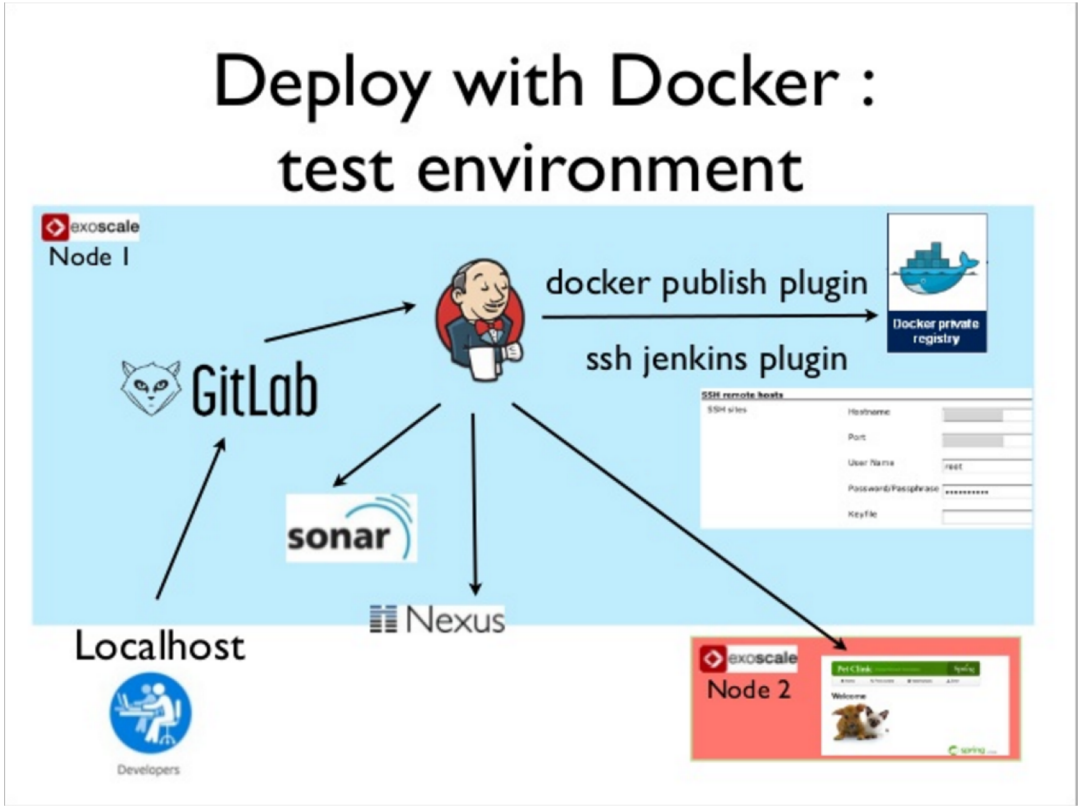
According to Michael Duke



Link to Blog: Docker as Jenkins Build Solution

http://bit.ly/2a57Za8

According to Julia Mateo



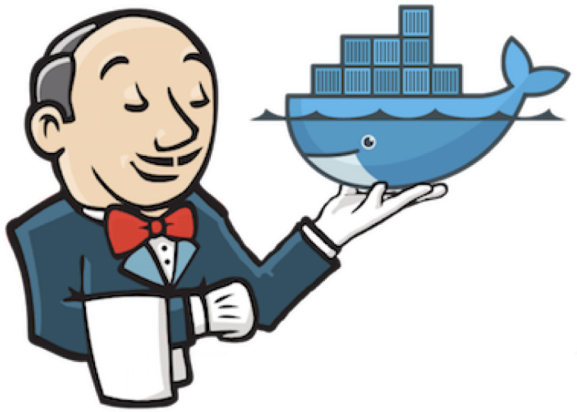
Link to Presentation: Continuous Delivery

http://bit.ly/2aHZj9o

Docker and Jenkins

Docker and Jenkins, a popular open source continuous integration tool, is a fairly common pattern, enough so that it is worth going into an example.

- Spin up Jenkins in a Docker Container
- Enable Jenkins to create other containers on the Docker Host computer



How to run a ‘Host’ Docker command inside a ‘Client’ Docker container?

Mount a Docker socket

- Issuing a Docker run command with the Docker socket will enable you to manage host containers from within the client container
- ‘docker run –v /var/run/docker.sock:/var/run/docker.sock’

Docker and Jenkins, Part 1

6.1 CLASSROOM WORK

Part 1, Jenkins Exercise in Docker Tutorials

This assignment is to set up a dockerfile that uses an official Jenkins image from Docker Hub, build an image with that image as a starting point, run a Jenkins container using the new image, and successfully get a Docker Hello world container running on the host but orchestrated from Jenkins in the container.

This exercise will help you practice Docker volume syntax as well as configure a client container (our Jenkins container) such that it can make calls to its host Docker engine.

Estimated time to complete: 20-30 min

Solution:
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise6.1-jenkins-part1.md

Sharing Persistent Volumes

```
ubuntu@ubuntu-xenial:~/docker-workshop/jenkins$ docker create -v /var/jenkins_home --name jenkinstore library/jenkins
eef810bfbad2247678db46187c0dec05d86a93397f27b40aff30640957977e15
ubuntu@ubuntu-xenial:~/docker-workshop/jenkins$ docker volume ls
DRIVER              VOLUME NAME
local               e99cc1a13396614cd1d6eaf0b44be7544c6af0560a4ca93c8605e3bcaf0a7161


ubuntu@ubuntu-xenial:~/docker-workshop/jenkins$ docker volume inspect e99cc1a13396614cd1d6eaf0b44be7544c6af0560a4ca93c8605e3bcaf0a7161
[
    {
        "Name": "e99cc1a13396614cd1d6eaf0b44be7544c6af0560a4ca93c8605e3bcaf0a7161",
        "Driver": "local",
        "Mountpoint": "/var/lib/docker/volumes/e99cc1a13396614cd1d6eaf0b44be7544c6af0560a4ca93c8605e3bcaf0a7161/_data",
        "Labels": null,
        "Scope": "local"
    }
]


ubuntu@ubuntu-xenial:~/docker-workshop/jenkins$ docker run --volumes-from jenkinstore --name jenkin1 -p 8080:8080 -p 50000:50000  library/jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
Nov 23, 2016 4:58:27 PM Main deleteWinstoneTempContents


ubuntu@ubuntu-xenial:~/docker-workshop/jenkins$ docker run --volumes-from jenkinstore --name jenkin2 -p 8080:8080 -p 50000:50000  library/jenkins
Running from: /usr/share/jenkins/jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
Nov 23, 2016 5:03:06 PM Main deleteWinstoneTempContents
WARNING: Failed to delete the temporary Winstone file /tmp/winstone/jenkins.war
```

Docker and Jenkins, Part 2

6.2 CLASSROOM WORK

Part 2, Shared Volumes Exercise in Docker workshop

The second assignment is to successfully share a persistent storage volume between two different Jenkins containers. Simulate the scenario of restoring your Jenkins state after a failure of the first container using the storage volume.

This exercise of sharing volumes could be extended to some simple open source databases as well.

Estimated time to complete: 15-25 min

Solution:
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise6.2-jenkins-part2.md

Docker and Jenkins, Part 3

6.3 CLASSROOM WORK

Part 3, Set up a Continuous Integration example.

Get a three-stage continuous integration pipeline working within your Jenkins container. This may involve installing additional plugins to your Jenkins instance (automate within your dockerfile) and adjusting the permissions on the Docker run command.

Estimated time to complete: 20-30 min

Solution:
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise6.3-jenkins-part3.md

Docker and Jenkins, Part 4

CLASSROOM WORK

Part 4, Call Continuous Integration Container from Docker Compose

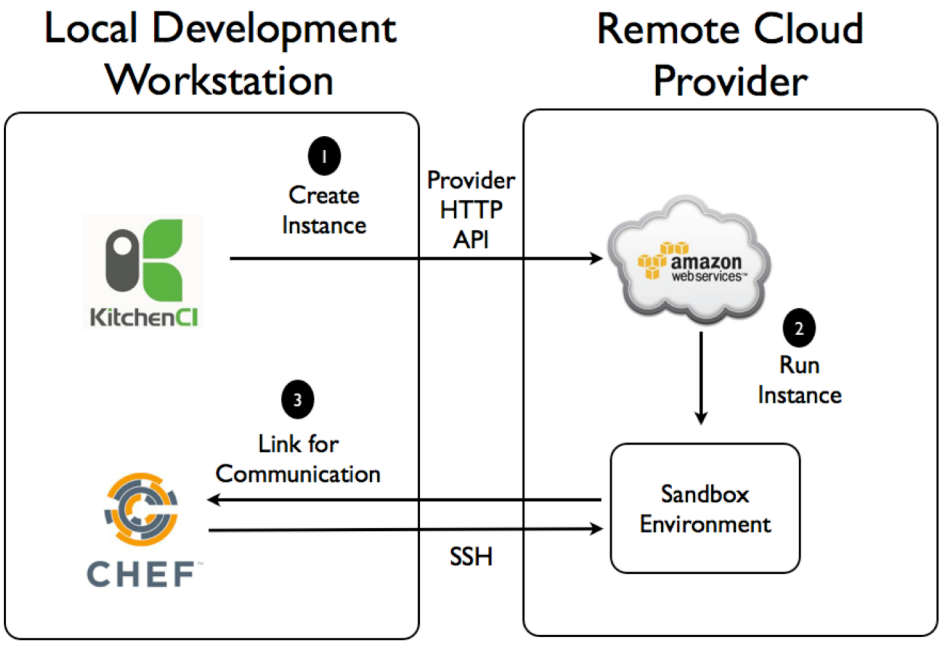
Take the working code from Part 3 and summarize it with a call to Docker Compose. Docker Compose is a much more convenient interface and scales better when multiple containers must be invoked.

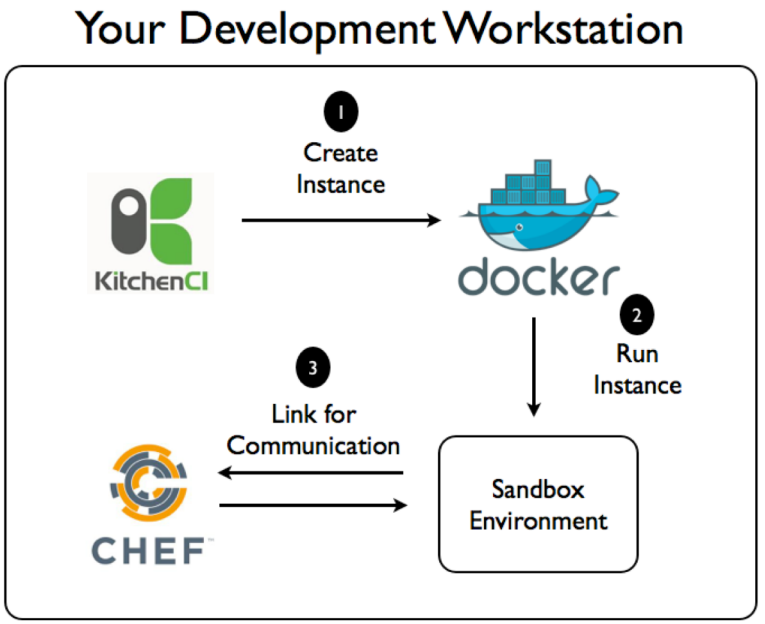
Bonus: Configure Jenkins container to skip setup workflow

Estimated time to complete: 15-25 min

Solution:
https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise6.4-jenkins-part4.md

According to Mischa Taylor



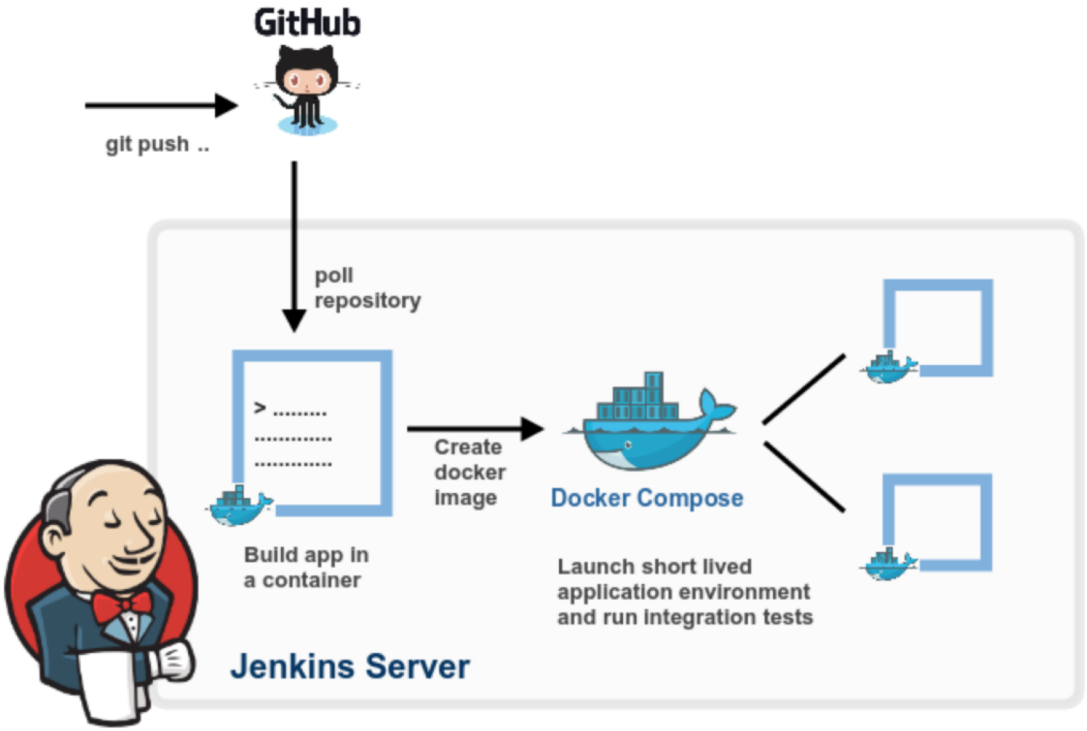


Link to Blog: Survey of Test Kitchen Providers

http://bit.ly/2a56kS2

According to Rancher

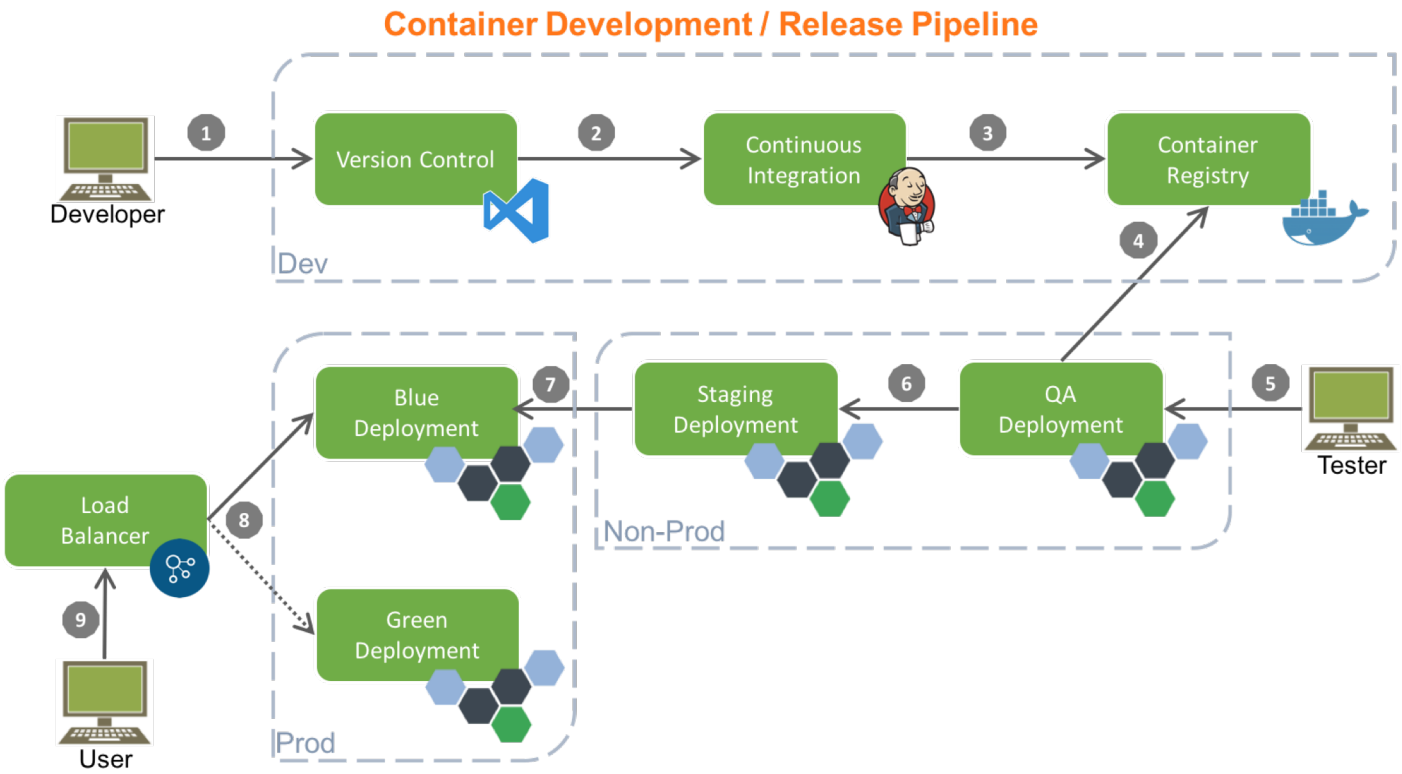
RANCHER



Link to Blog: Docker-based Build Pipelines

http://bit.ly/2aHtu0K

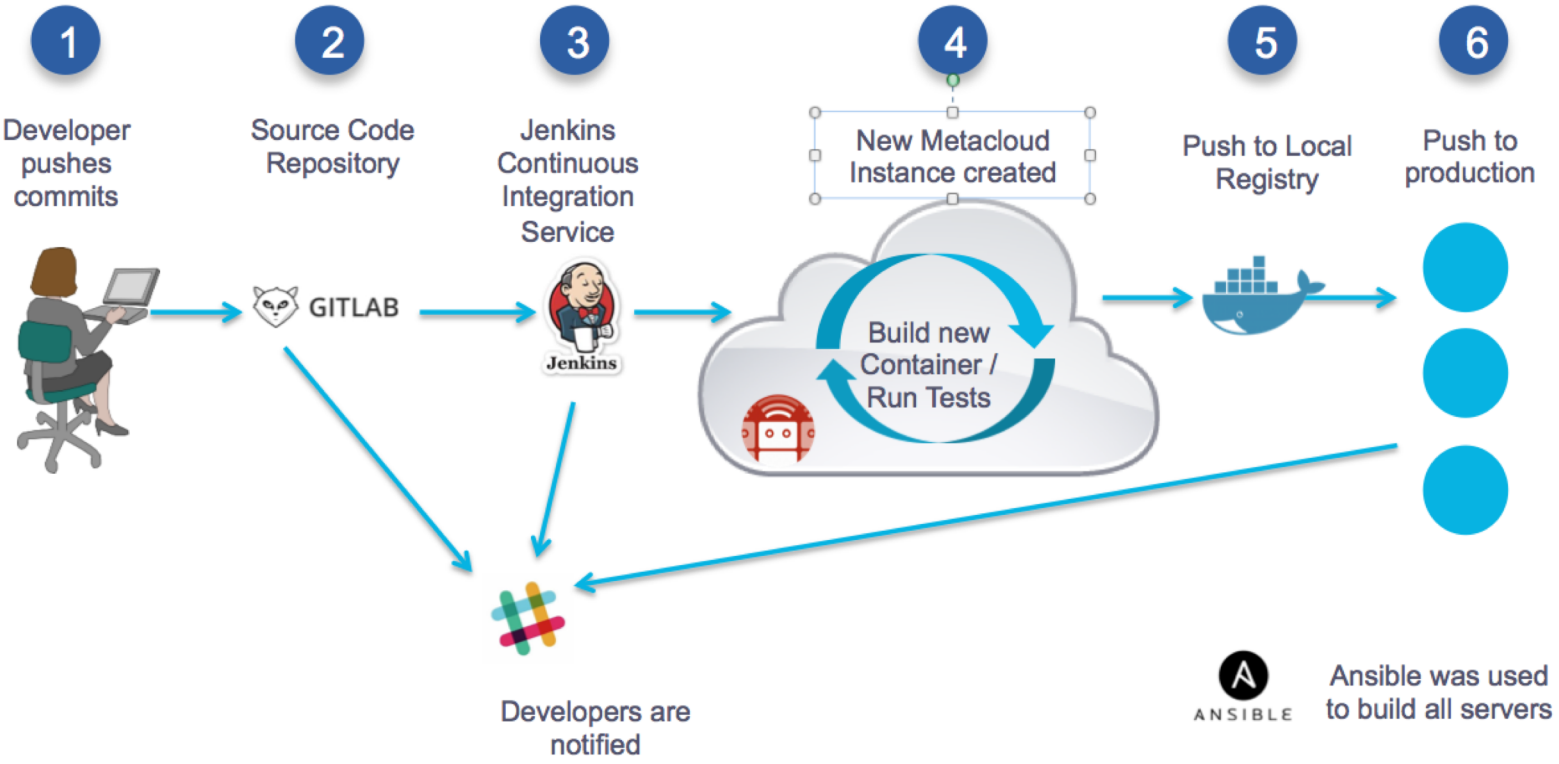
According to Robert Greiner



Link to Blog: Continuous Integration with Docker

http://bit.ly/2aeA1io

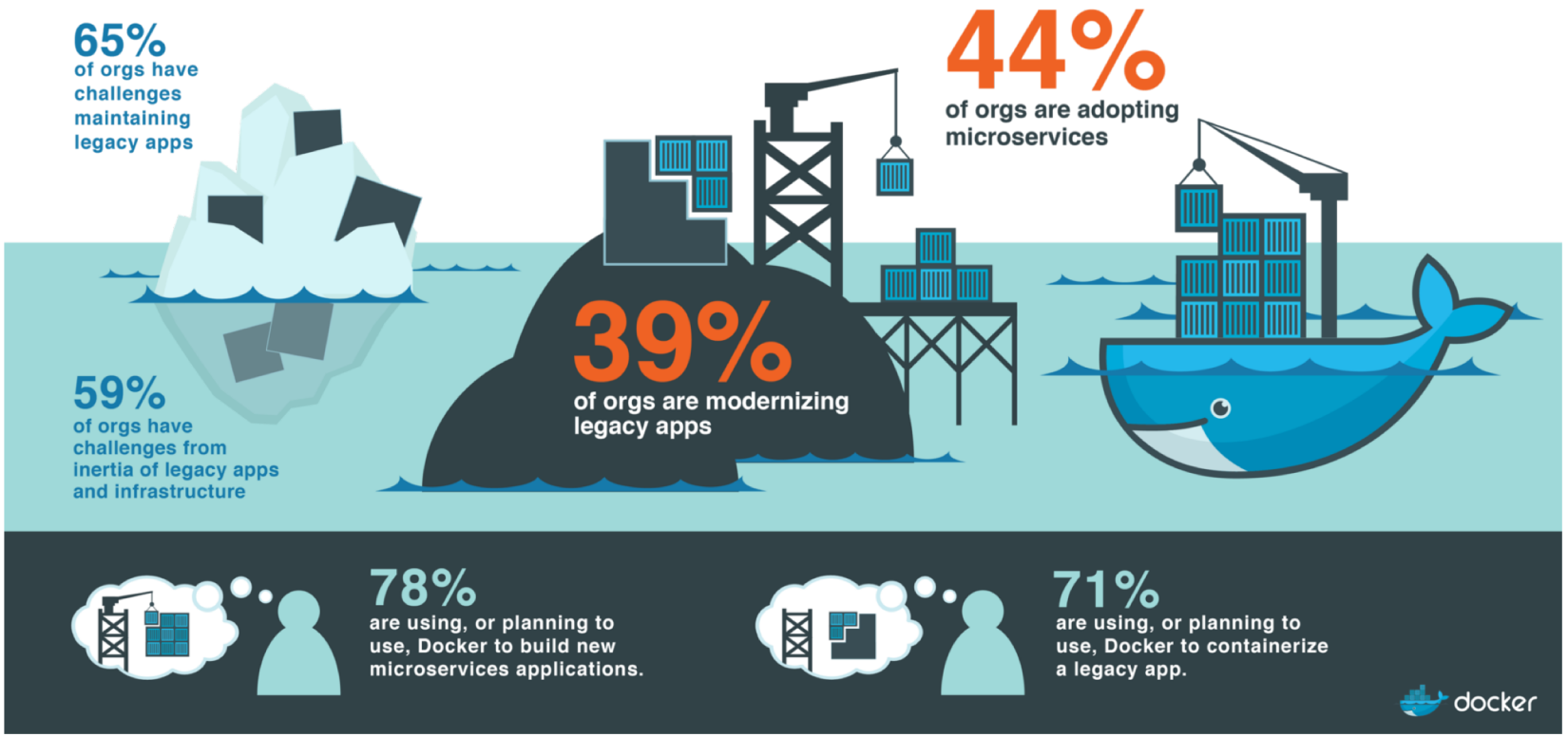
According to Vallard



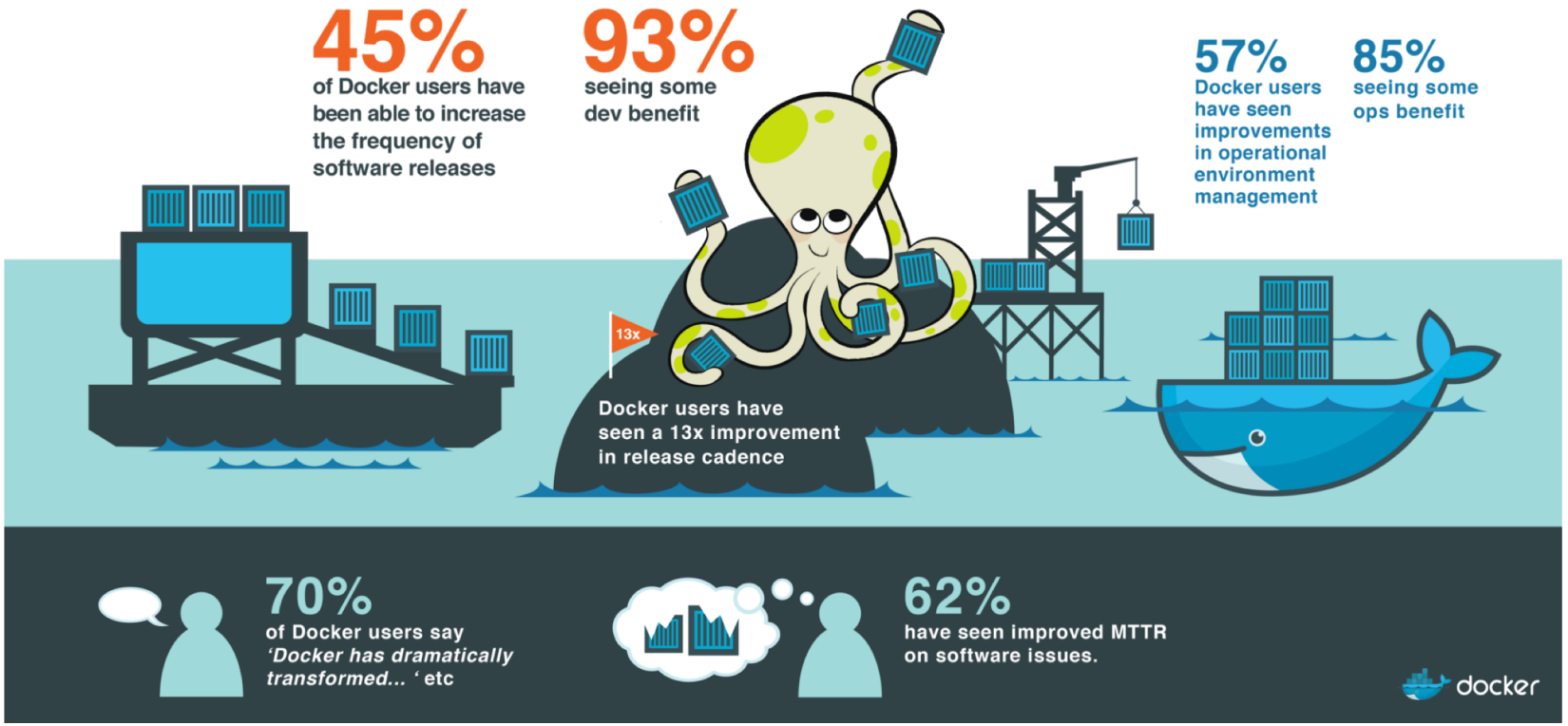
Link to Blog: Continuous Delivery of a Simple Web Application

http://bit.ly/2asS8G3

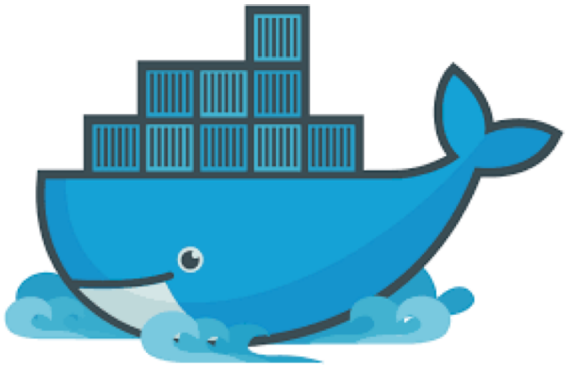
Docker Adoption & Usage Data



Docker Adoption & Usage Data

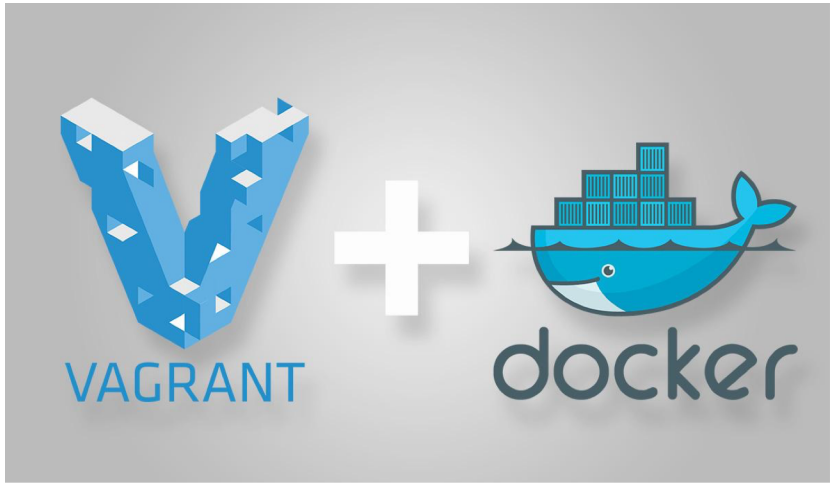


Local Docker Development



MacOS & Windows - The Old-Old Way Vagrant Recipe for Local Docker Dev

1. Install Vagrant
   https://www.vagrantup.com/
2. Install VirtualBox
   https://www.virtualbox.org/wiki/Downloads
3. Git clone
   https://github.com/russmckendrick/monitoring-docker.git
4. Enter vagrant-centos dir
   - cd vagrant-centos
5. Launch Centos VM (with docker installed)
   - vagrant up
6. Connect to Centos VM
   - vagrant ssh
7. Verify success!
   - docker run hello-world

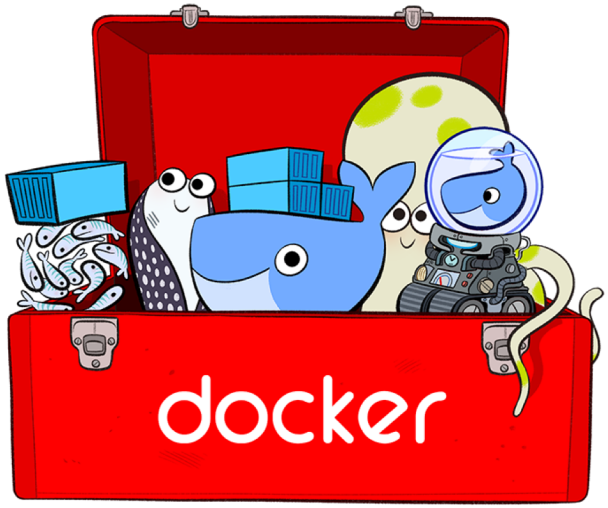


MacOS & Windows - The Less-Old Way
Use Docker Toolbox / Machine Locally

- Ideal for older Mac and Windows systems that do not meet the requirements of Docker for Mac or Docker for Windows.

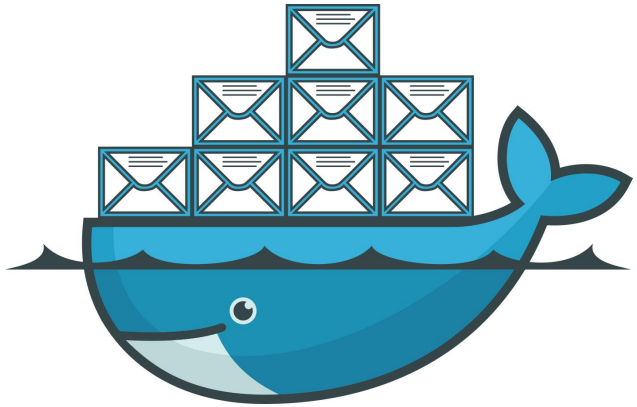
1. Install Docker Toolbox https://docs.docker.com/toolbox/overview/

- This one step gets you:
  - Native Docker CLI client
  - VirtualBox
  - Docker Machine (which can bring up a local Linux VM using VirtualBox)
  - Docker Compose
  - Kitematic (Docker GUI)
  - Docker QuickStart shell



MacOS & Windows - The New Way
Docker for Mac / Docker for Windows

- Simple one-step install
- Docker for Mac requires Mac OS 10.11 or newer running on 2010 or newer Mac hardware
- Docker for Windows requires 64bit Windows 10 Pro Enterprise and Education and Microsoft Hyper-V
- Both tools provide the full Docker environment as well as a convenient GUI for managing volumes, utilization, and networking
- Under the hood, it's docker-machine

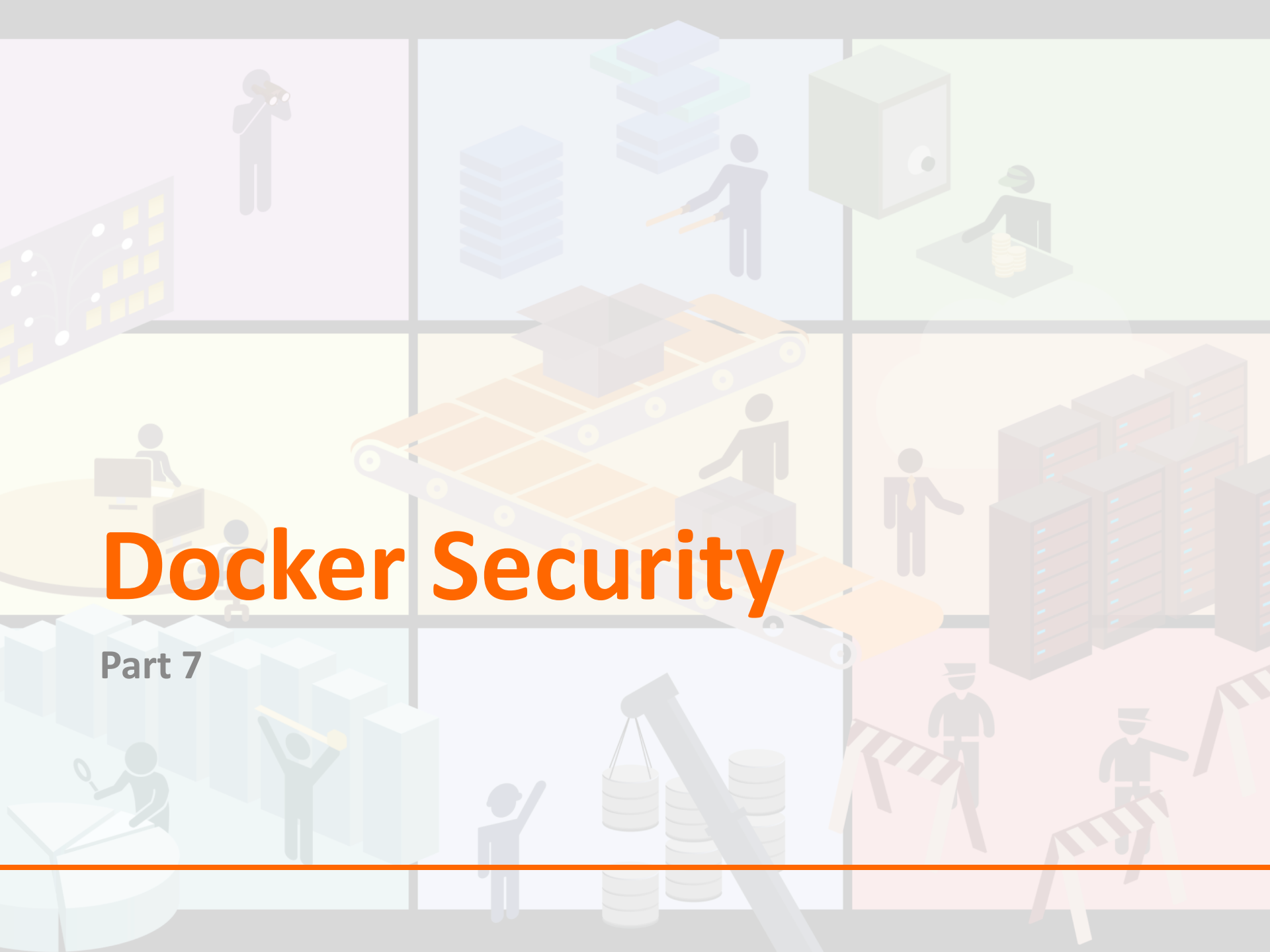


Docker Machine - Docker Engine Provisioning

- Machine lets you create Docker hosts on your computer, on cloud providers, and inside your own data center. It creates servers, installs Docker on them, then configures the Docker client to talk to them. It also can provision Swarm clusters.
- $ docker-machine create \
  -d virtualbox default
- $ docker-machine create \
  -d digitalocean \
  --digitalocean-access-token xxxxx \
  docker-sandbox
- $ docker-machine create -d amazonec2 \ --amazonec2-ac
  AKI******* \ --amazonec2-secret-key 8T93C******* \
  aws-sandbox
- Local and Cloud drivers available

New in Docker for Windows / Docker 1.13 Native Windows Containers

- Certain very new versions of “Windows 10 Pro Enterprise and Education” and “Windows Server 2016” now have Windows kernels that support containerization and can be used to run native Windows Containers that run Windows applications in a container environment
- Requires Docker for Windows, and supports shared-kernel (“Windows Server Containers”) and VM-based (“Hyper-V Containers”)



Docker Security

Part 7

Security For Docker Images

- **Secure Registry/Mirror Access**
- **Getting trustworthy images**
  - **Trusted sources**
    - docker hub / docker store
      - official images
      - pull by digest
    - private registry
  - **Building secure**
- **Docker Content Security**
  - **Docker Notary**
    - Digitally signed images
      - "only signed content in production"
    - Yubico Keys can be used for code signing

Docker Content Trust

7.1 CLASSROOM WORK

In this exercise, we'll learn how to enable Docker Content Trust and sign images.

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise7.1-docker-security.md

(Step 3.4 requires an active Docker Hub account… easy to acquire with just an email address.)

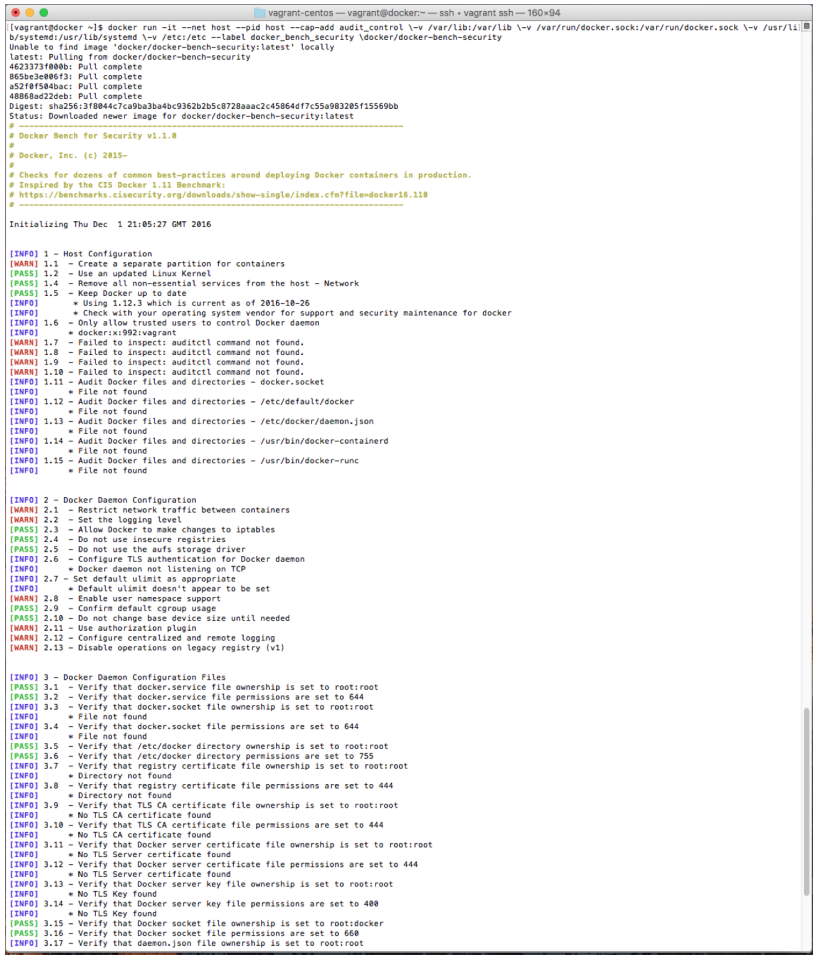
Docker Bench Security

Docker Bench is an automated script that checks for dozens of best-practices around deploying Docker in production

https://github.com/docker/docker-bench-security

Run with the following command (available at the github page above)

```
docker run -it --net host --pid host --cap-add
audit_control \
 -v /var/lib:/var/lib \
 -v /var/run/docker.sock:/var/run/docker.sock \
 -v /usr/lib/systemd:/usr/lib/systemd \
 -v /etc:/etc --label docker_bench_security \
 docker/docker-bench-security
```



Docker Bench Results Discussion

Review Docker Security white paper:

https://d3oypxn00j2a10.cloudfront.net/assets/img/Docker%20Security/WP_Intro_to_container_security_03.20.2015.pdf

Points to consider:

- **Security is one of the most difficult areas of IT for which to prescribe solutions**
- **Security is often misunderstood or de-prioritized by the business**
- **Which is the strongest recommendation from the Docker Bench Results list? Most surprising?**

Discussion

Discuss

(Some) Docker Security Best Practices

- Access to Docker host = full access to all running containers and any new ones
- Docker containers can attach to volumes in read only mode with :ro option
  - Docker run –d –v /some/volume:ro jenkins
- Start Docker containers with the -u flag so that they run as an ordinary user instead of root. Consider dropping SUID support entirely in production containers
- Mitigate DoS by limiting CPU, RAM, Sockets that each container can consume
- Use secure computing (seccomp) to block system calls at kernel level. Use strace to determine kernel calls made, then create a profile file in json format and start the container using that profile file
  - docker run --rm -it --security-opt seccomp=custom_profile.json custom_app
- Log to stdout / stderr - so you can see with docker logs and docker can move to syslogs (for capture/rotate by ELK, etc.)

Docker Security

Stay up to date with the discussion at Docker's user group, forum, github issues, and IRC channel

- https://groups.google.com/forum/#!forum/docker-dev
- https://forums.docker.com/
- https://github.com/docker/docker/issues
- https://docs.docker.com/opensource/get-help/
- **IRC #docker & #docker-dev**

Docker Security Scanning (formerly Project Nautilus)

- Docker Security Scanning conducts binary level scanning of your images before they are deployed, provides a detailed bill of materials (BOM) that lists out all the layers and components, continuously monitors for new vulnerabilities, and provides notifications when new vulnerabilities are found.
- Available for private Docker Hub repositories (potentially free)
- Available as a paid service to Docker Cloud customers, maintainers of “Official” repositories on Docker Hub, and Docker EE Advanced

Clair by CoreOS

- Robust free and open source image security scanning tool
- https://github.com/coreos/clair
- Static analysis of images for known vulnerabilities
- Integrates into CI pipelines

Swarm Mode Security

Swarm Mode was designed with security in mind

- **Keys required to join a swarm**
  - Key rotation features built-in
- **All node communications (management channel) are encrypted.**
  - Certificates required for node communications
  - Certificate rotation features built-in
- **Overlay network management channel communications encrypted by default**
  - Overlay network data may be easily encrypted by setting flag at network creation time
  - Key and certificate rotation features built-in

Swarm Security Basics

7.2 CLASSROOM WORK

In this exercise, we'll play with some of Swarm's built-in service node management security features.

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise7.2-docker-swarm-security.md

Secrets Management

- ENV variables - NOT recommended
- General-purpose Key/Value Pair solutions
  - Vault
  - Keywhiz
- Embedded in orchestration
  - Docker 1.13+ - recommended
  - Kubernetes secrets - NOT recommended*
- Custom solutions
  - Make sure you care as much as Docker et al.

Swarm Secrets Management

7.3 CLASSROOM WORK

In this exercise, we'll use Docker's secrets objects to share a secret across nodes in a swarm.

1. **Make sure your host is not participating in a swarm**
   a. **docker swarm leave**
2. **Create (and join) a new swarm:**
   a. **mkdir ~/secrets**
   b. **cd ~/secrets**
   c. **docker swarm init**

https://github.com/techtown-training/docker-bootcamp/blob/master/exercise/exercise7.3-docker-secrets.md

What's New in Docker 1.13+

https://blog.docker.com/2017/01/whats-new-in-docker-1-13/

- Compose files for docker swarm deployment
- Improved CLI backwards compatibility
- Clean-up commands
- CLI restructured - object->action
- Monitoring improvements for services
- --squash flag (flatten all newly built layers)
- Docker for AWS and Azure out of beta
- Secrets objects out of beta

Announced at DockerCon US 2017

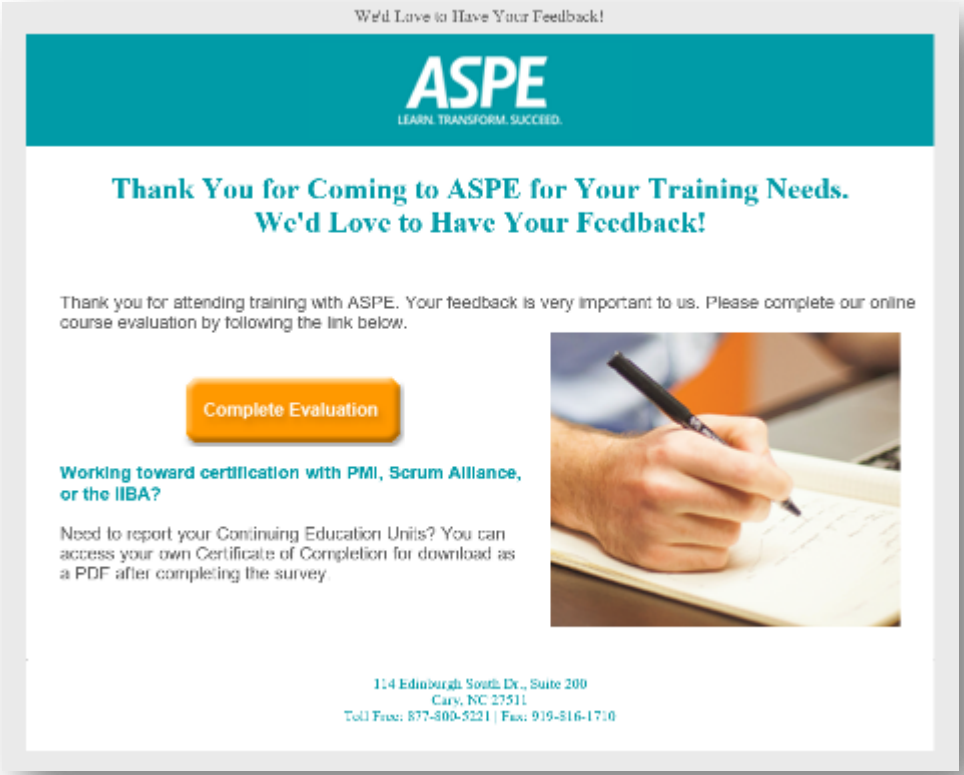
- Multi-stage Builds
- Docker Google Cloud Platform beta available
- LinuxKit available
  - Open source tool for building your own extraordinarily tiny and needs-specific Linux OS distributions
- Image2Docker (for Native Windows Container dev)
  - Analyzes Windows VM files (VHD, VHDK, WIM and VMDK) and suggests dockerfile commands to replicate installed components
- Oracle announces it will release their products into Docker Store
- Persistent storage solutions abounded - EMC, StorageOS, Nimble*

Docker is So Much Fun!

- **So easy to get started**
  - **Got an open-source or commercial service or application that you're considering? Docker makes ~~playing with~~ researching it as easy as:**
    - $ docker run coolnewtoy:latest
- **So easy to scale:**
  - **docker service scale webfrontend=5**
- **Some more fun:**
  - **https://github.com/docker/dockercraft**
    - https://www.youtube.com/watch?v=eZDlJgJf55o

Feedback Email

- If you haven't already, you will receive an email similar to the one here asking for your feedback on the course.
- This Evaluation will ask your feedback on many aspects of your training, including the course materials, instructor facilitation and labs where applicable.
- We appreciate you completing this survey to help us continue to better our content and deliveries in order to serve you better.



Questions