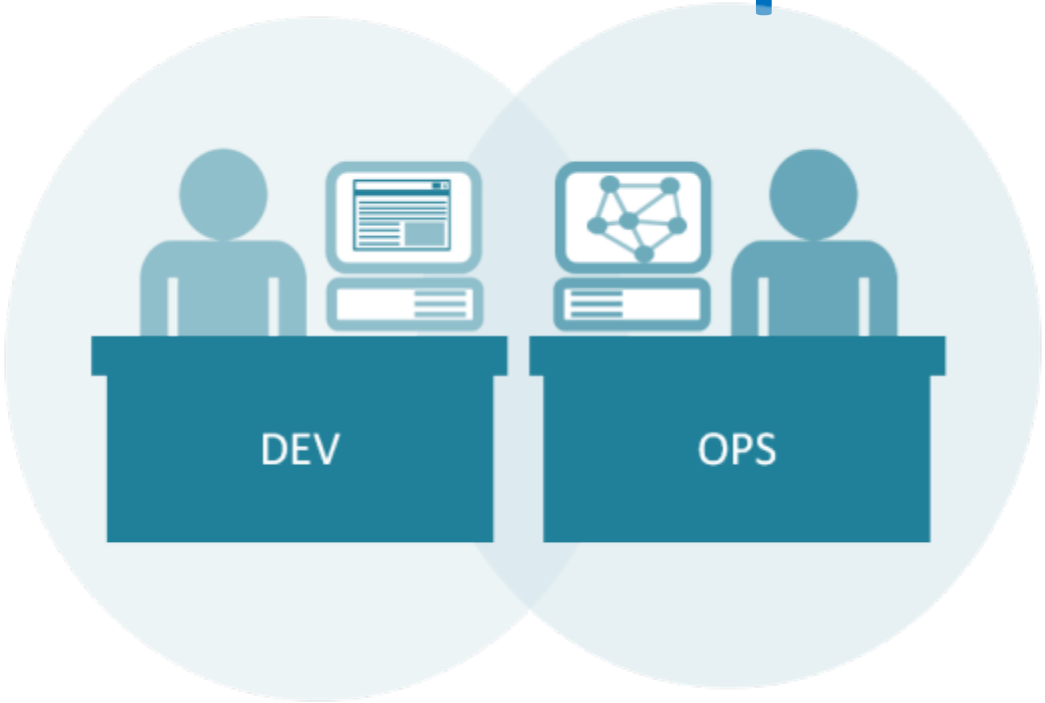
Docker
Container Implementation
Boot Camp



Welcome!

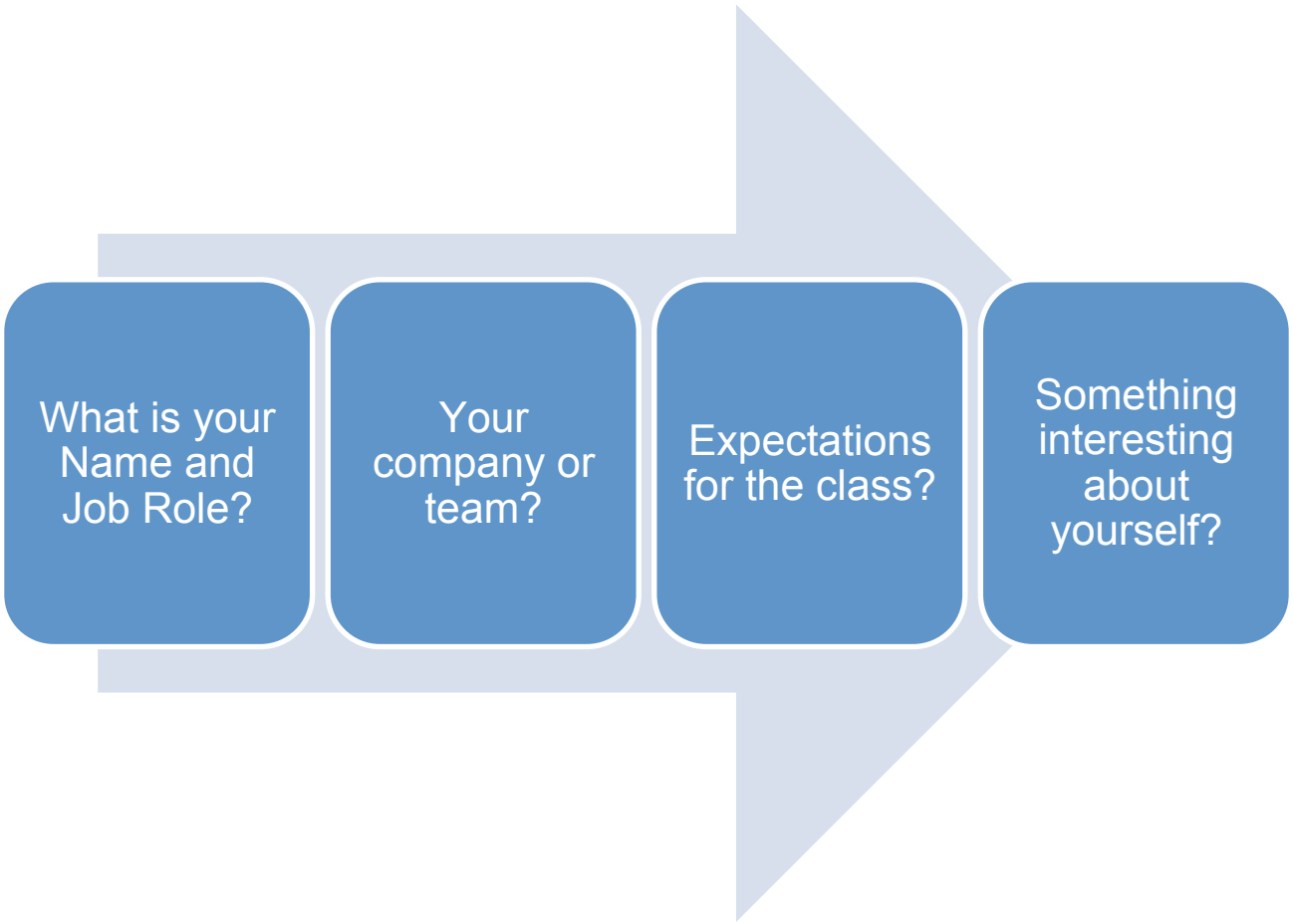
- Logistics (breaks, facilities, lunch, etc.)
- Rules of Engagement
- Introductions
- Let's Get Started!

Instructor Information

Who is your instructor?

A little about me…

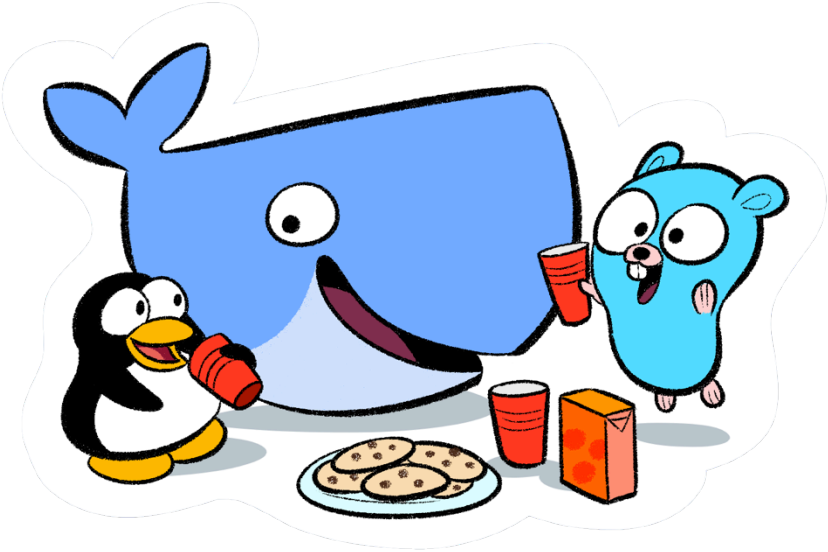
Introductions



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



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

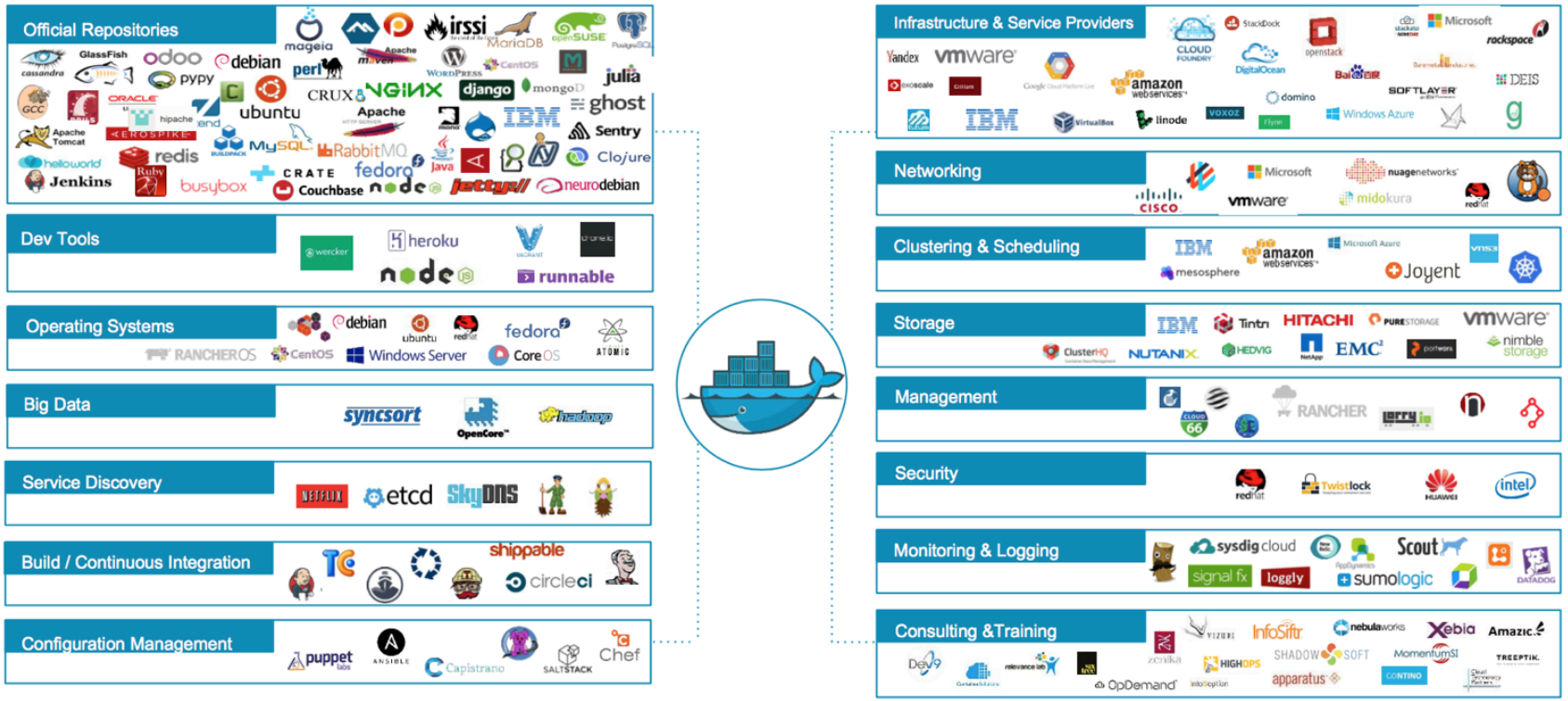
Containers as a Service provider

- 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

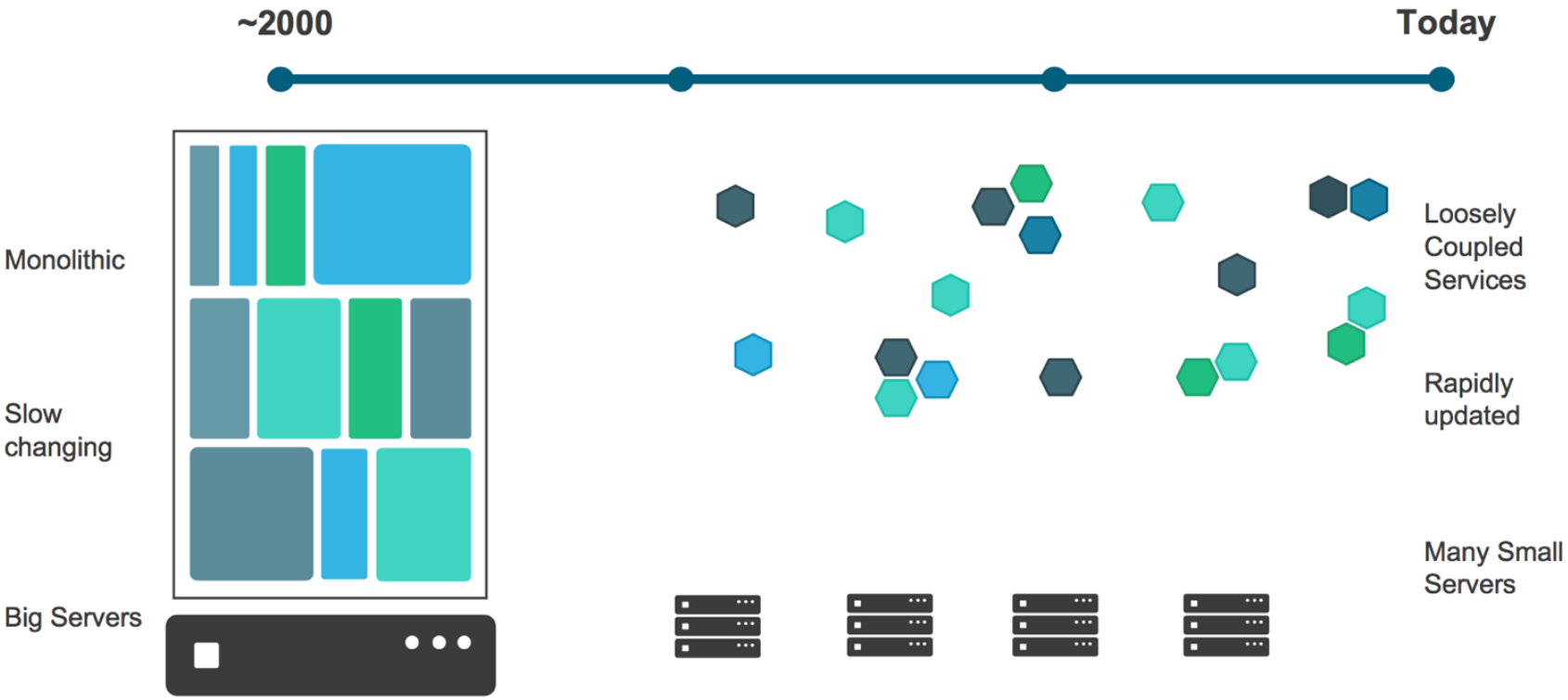
Docker Engine

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

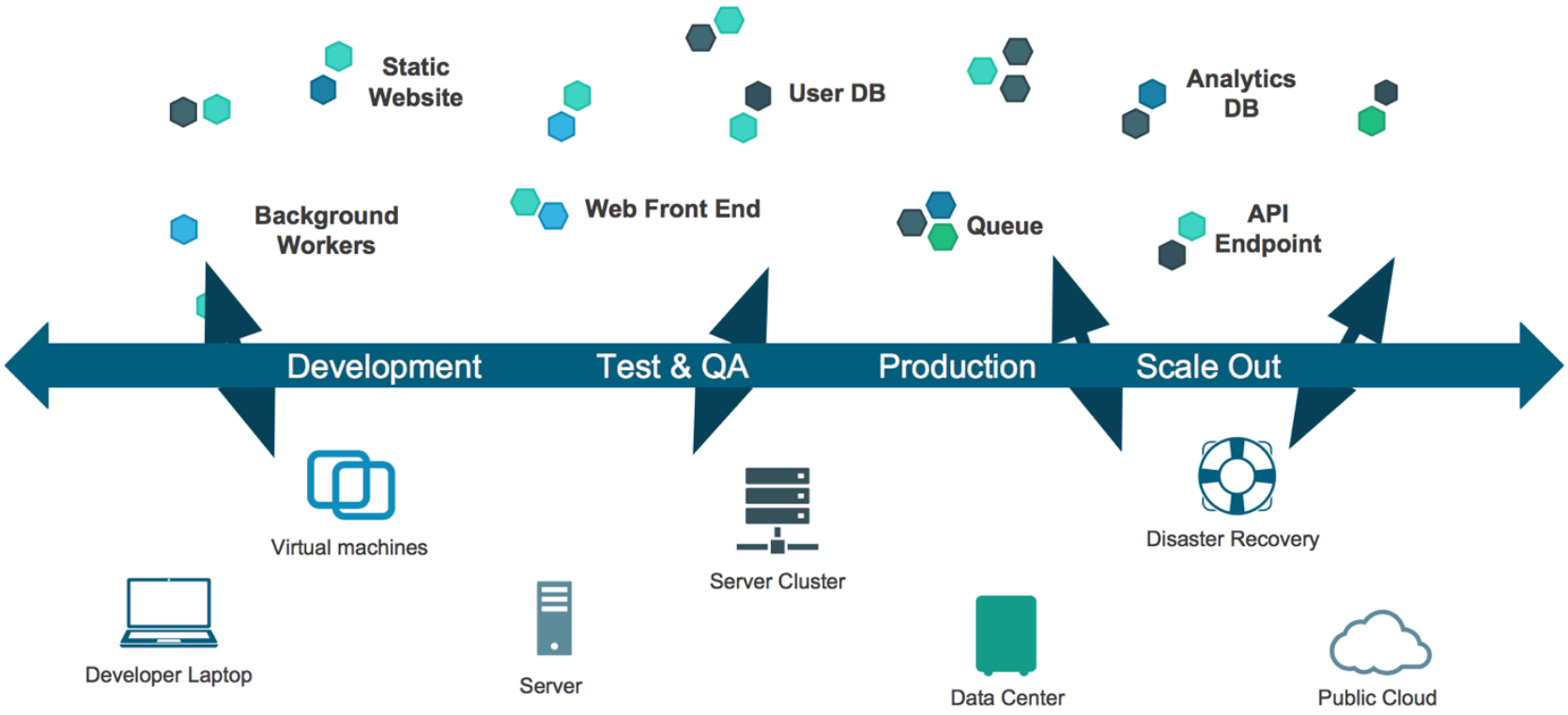
Docker Registry

- On-premises registry for image storing and collaboration

Application Evolution



Challenge: The Dependency Matrix
"It works on MY machine."

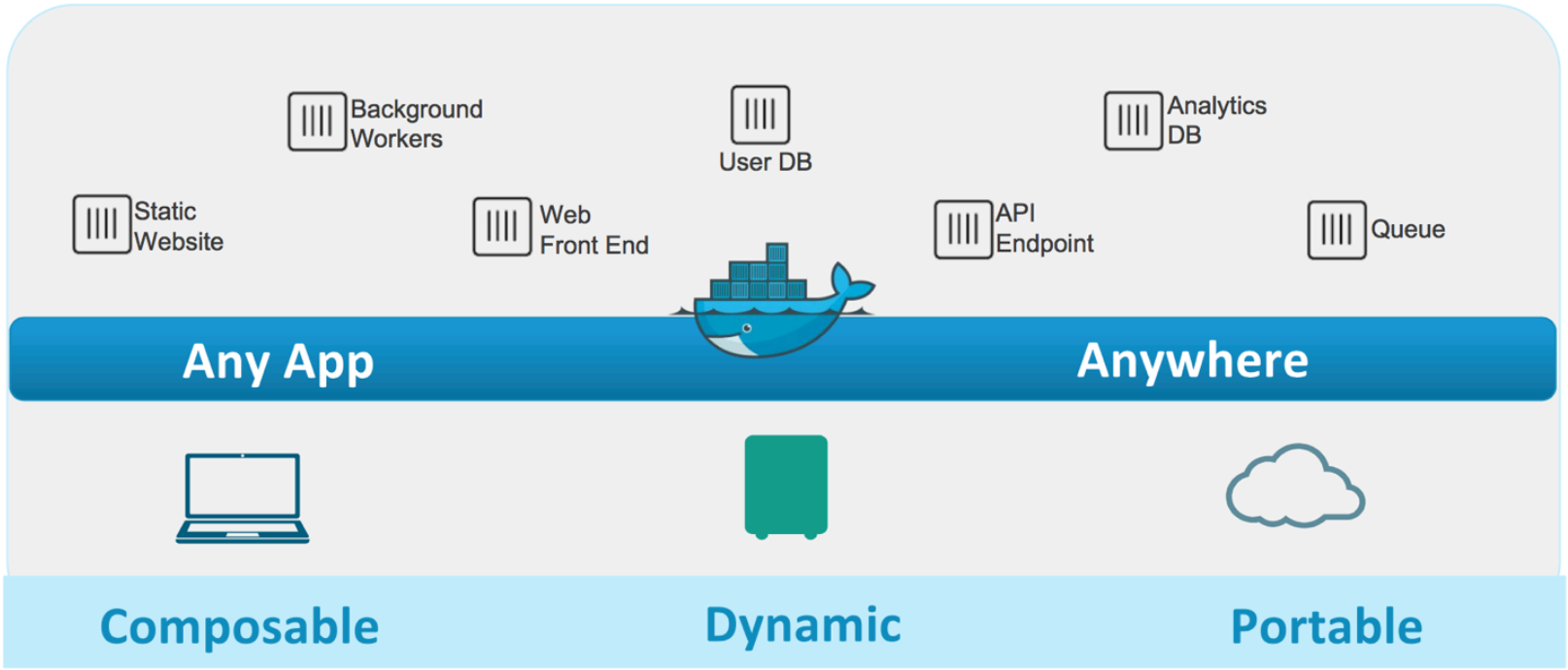


Containers as a Solution

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



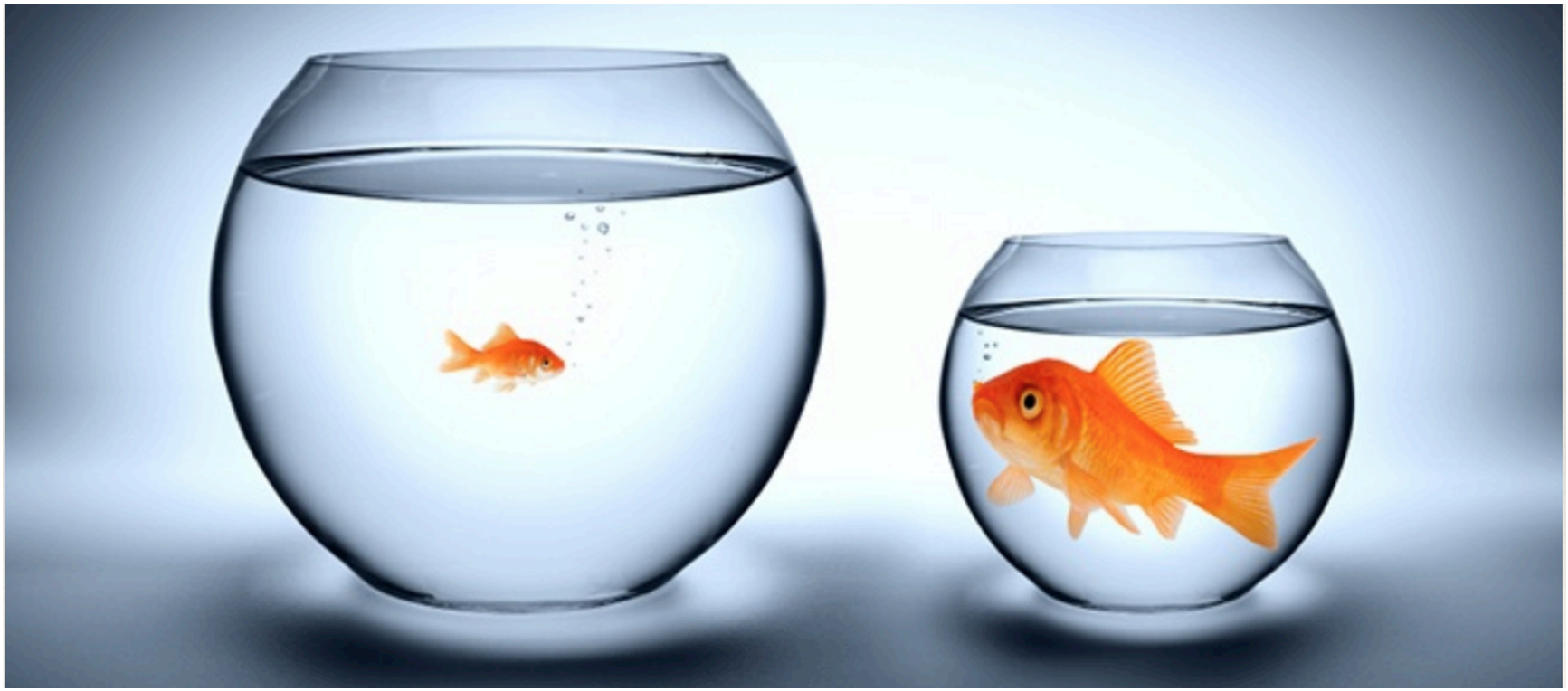
Developer Benefits

- 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

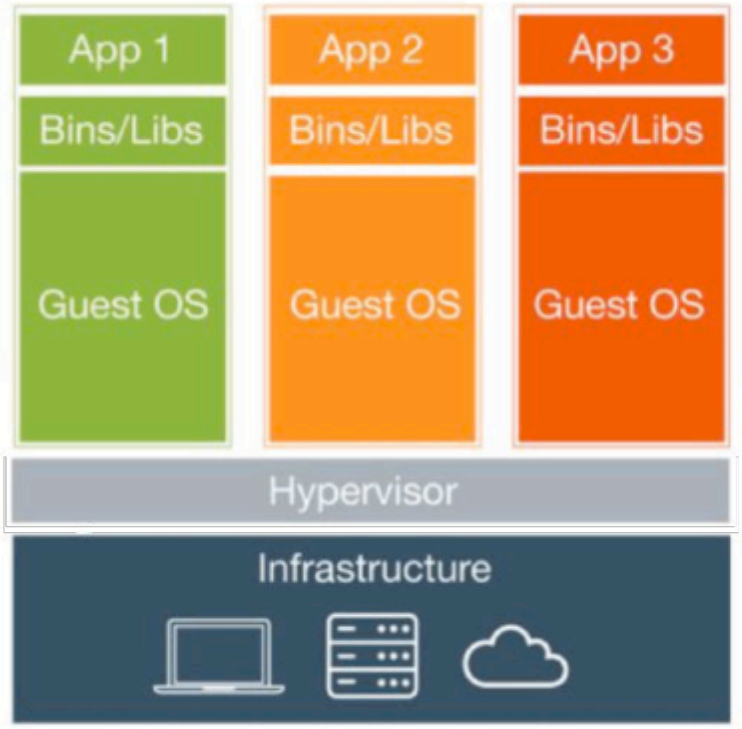
Operational Benefits

- 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

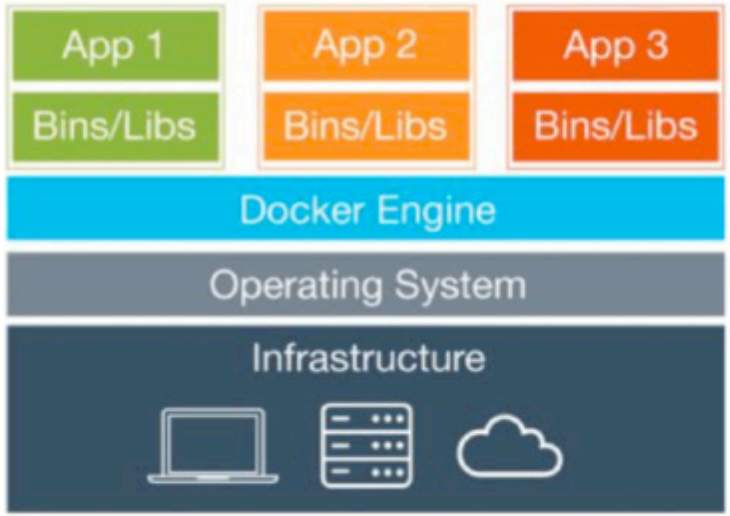
VMs vs. Containers



VMs vs. Containers



Virtual Machines



Containers

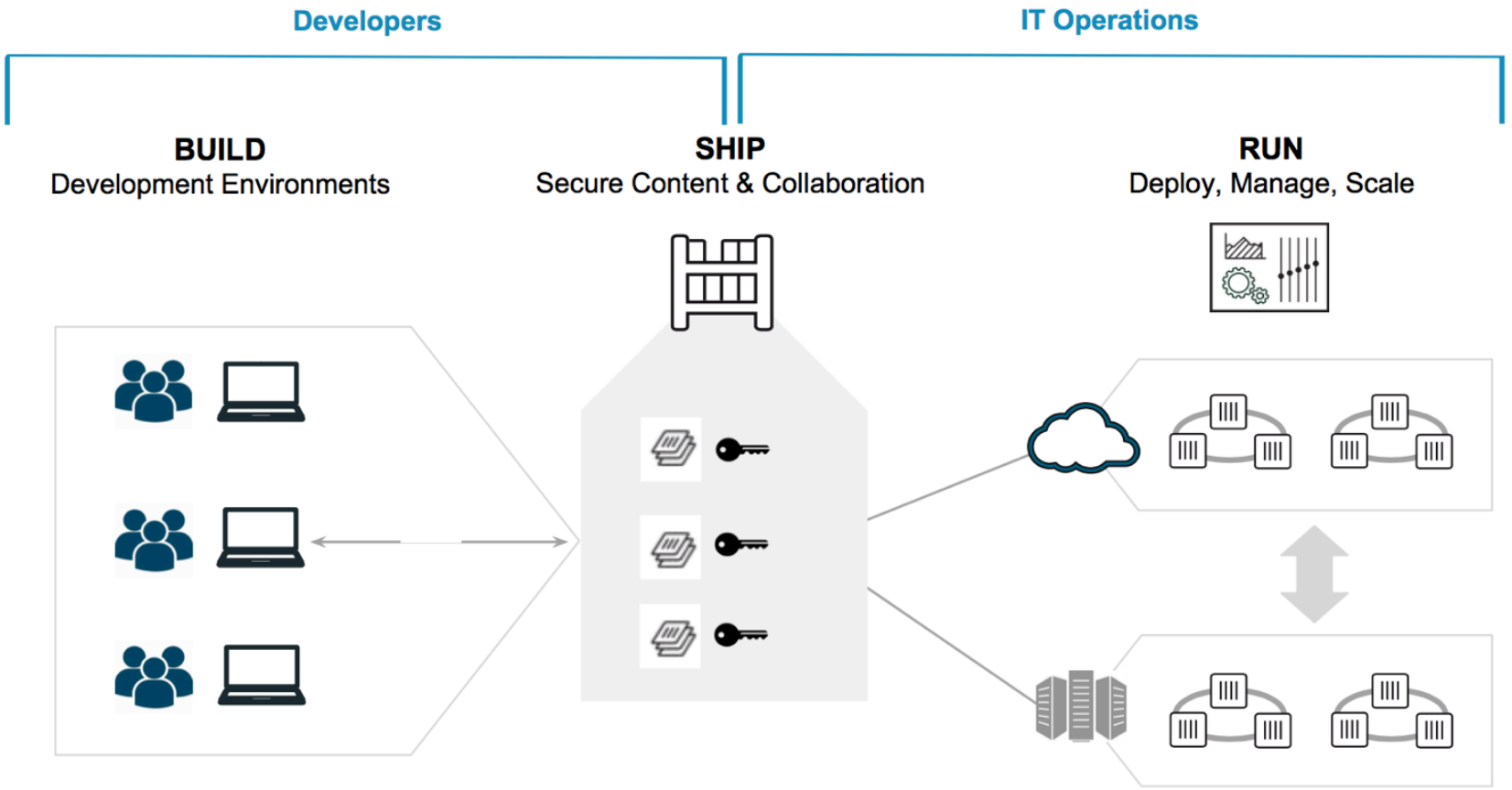
VMs vs. Containers

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 (independent software vendors) 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.

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

Done already in AWS instances

```
$ 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
```

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

Install Docker-CE

CLASSROOM WORK

Done already in AWS instances

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

$ sudo docker run hello-world
```

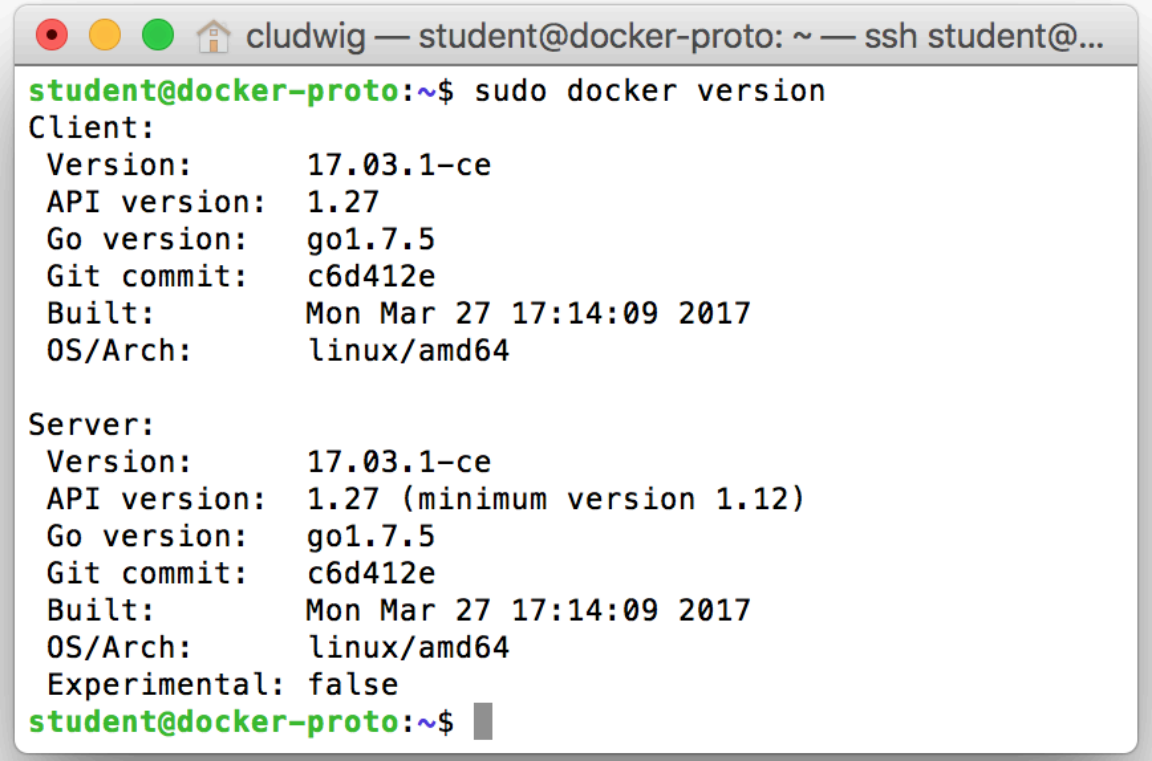
Also at
https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/#install-using-the-repository

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

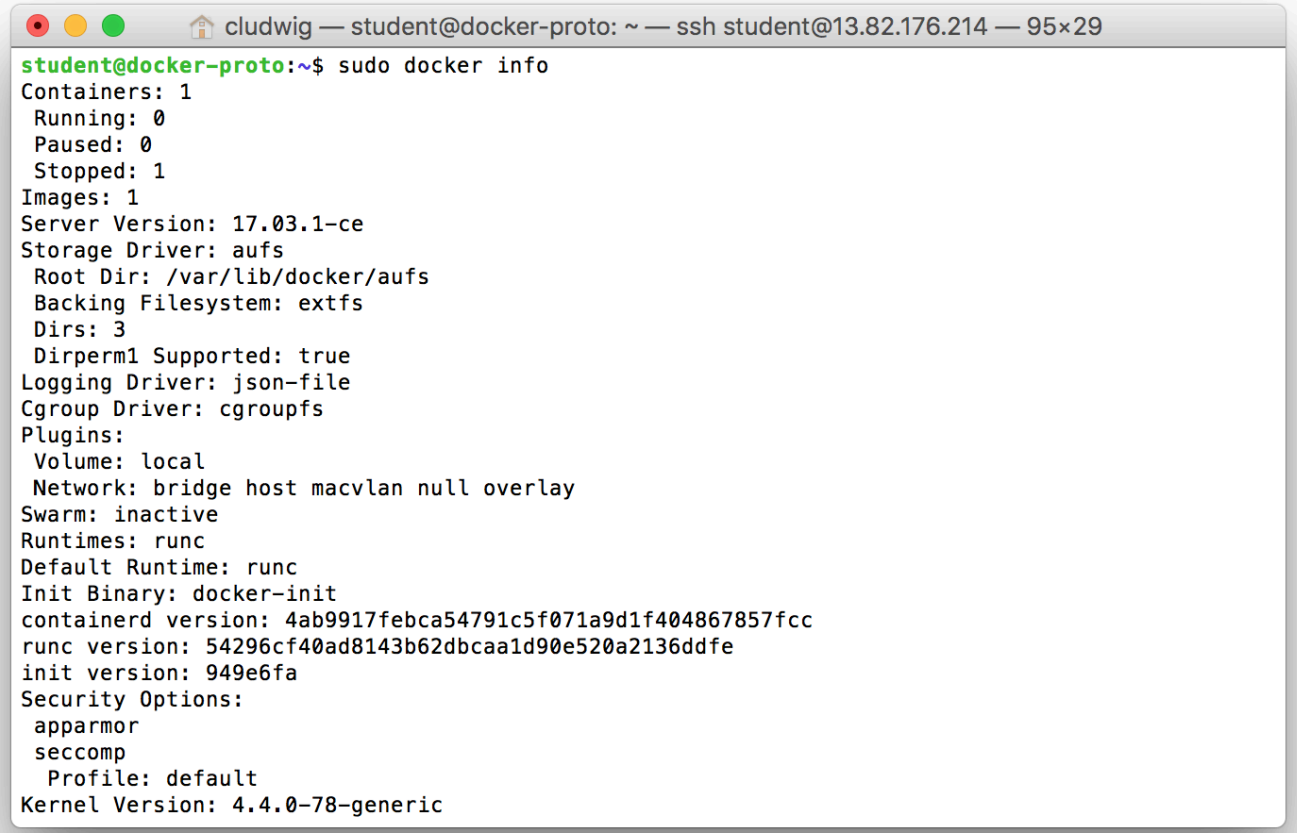
Docker Basics

$ 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.

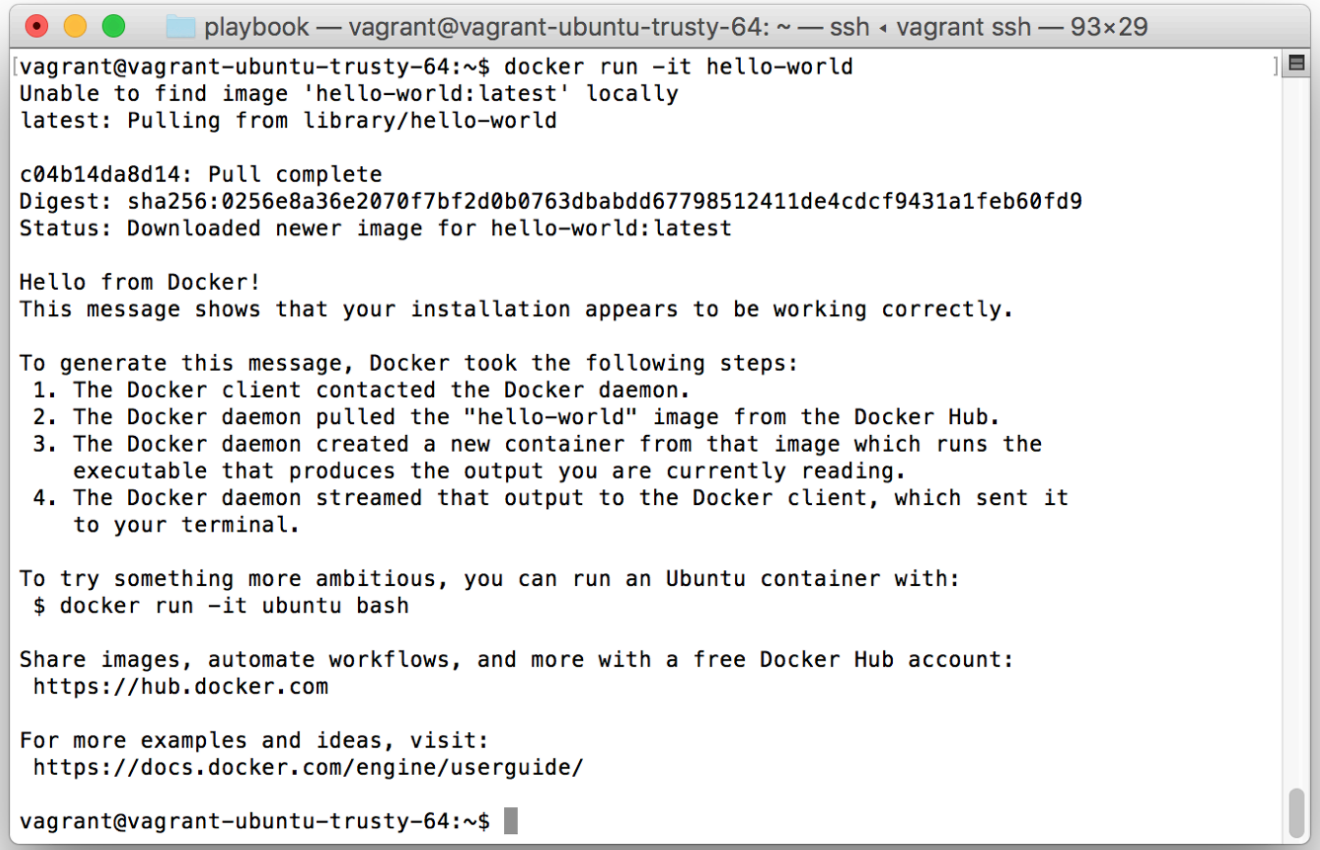
Docker Basics

Determine the status of the Docker service

- $ sudo service docker status

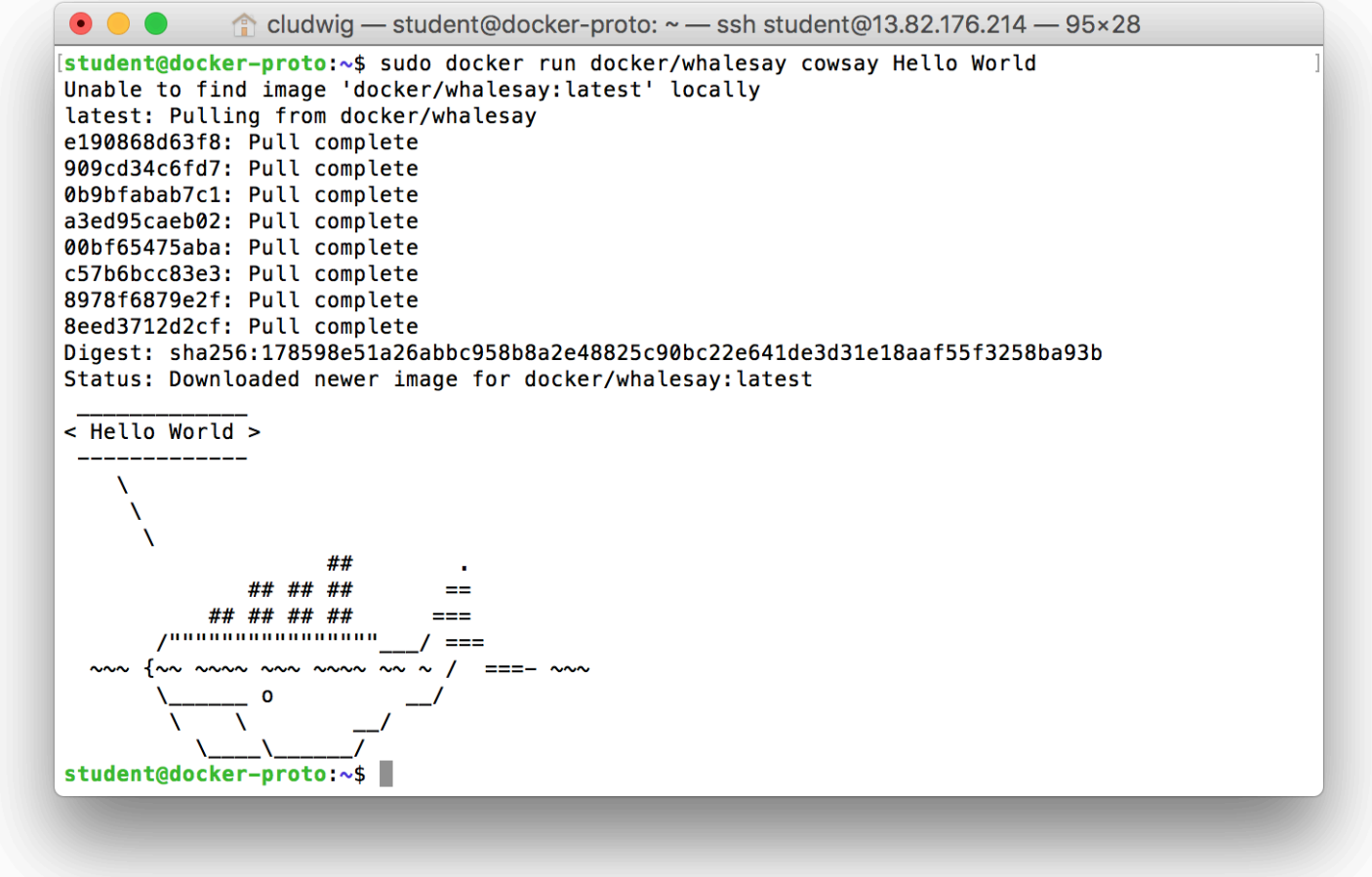
```
ubuntu@ubuntu-xenial:~/docker$ sudo service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2016-11-29 14:12:54 UTC; 2h 47min ago
     Docs: https://docs.docker.com
 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)

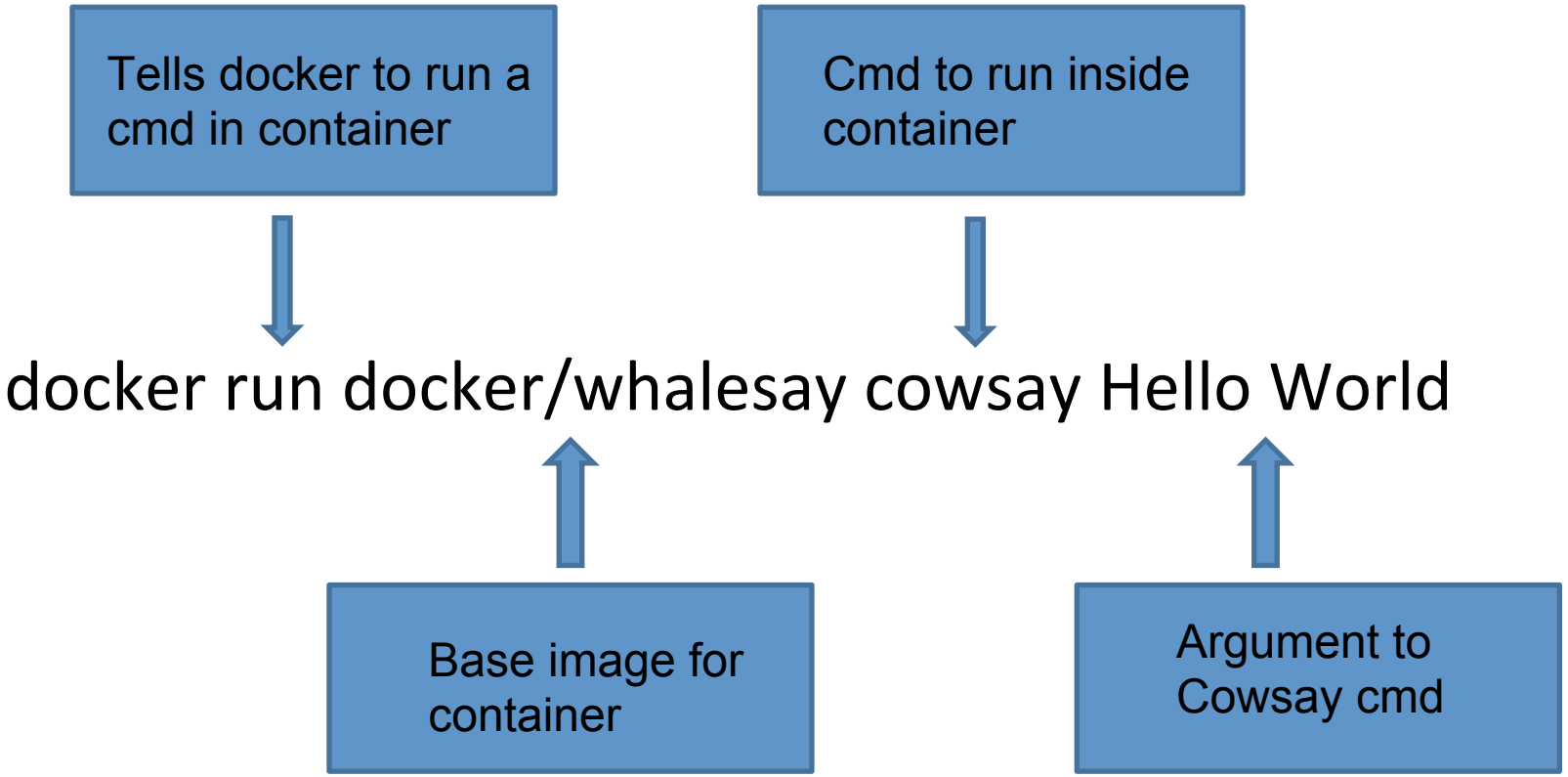


Docker Hello World(s)

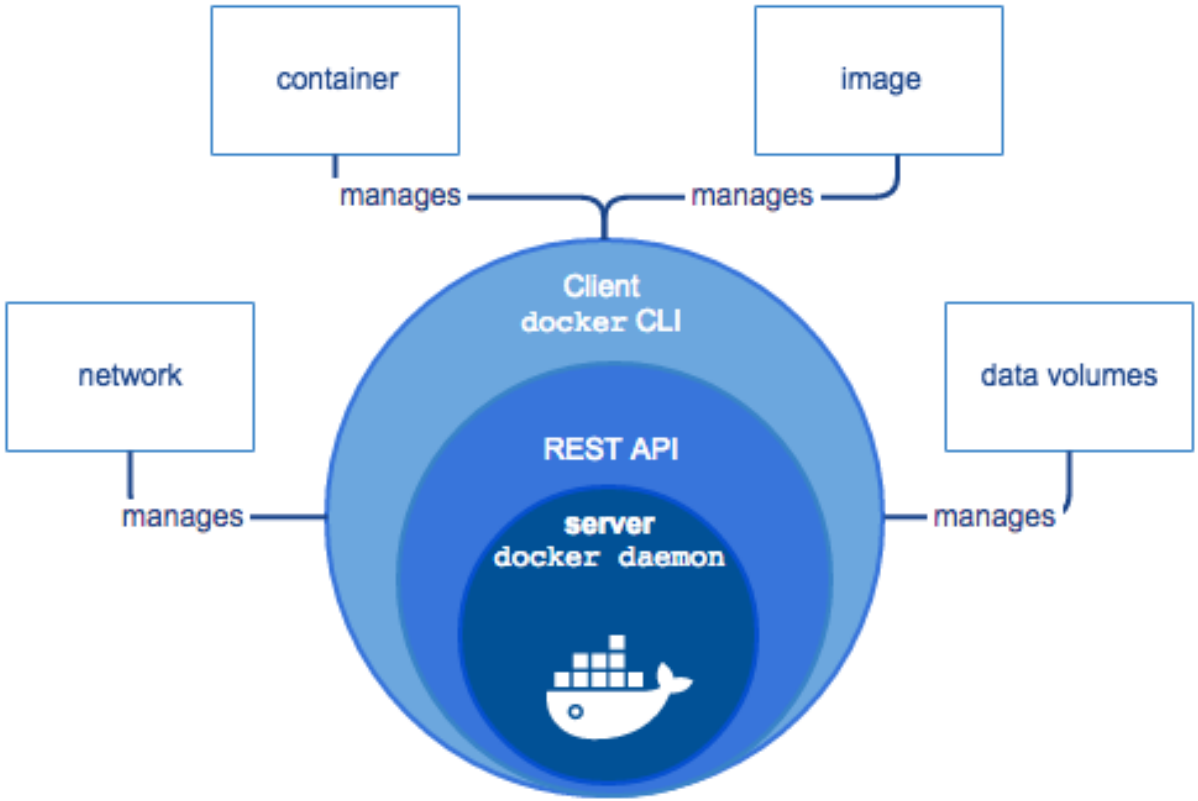
$ sudo docker run docker/whalesay cowsay Hello World



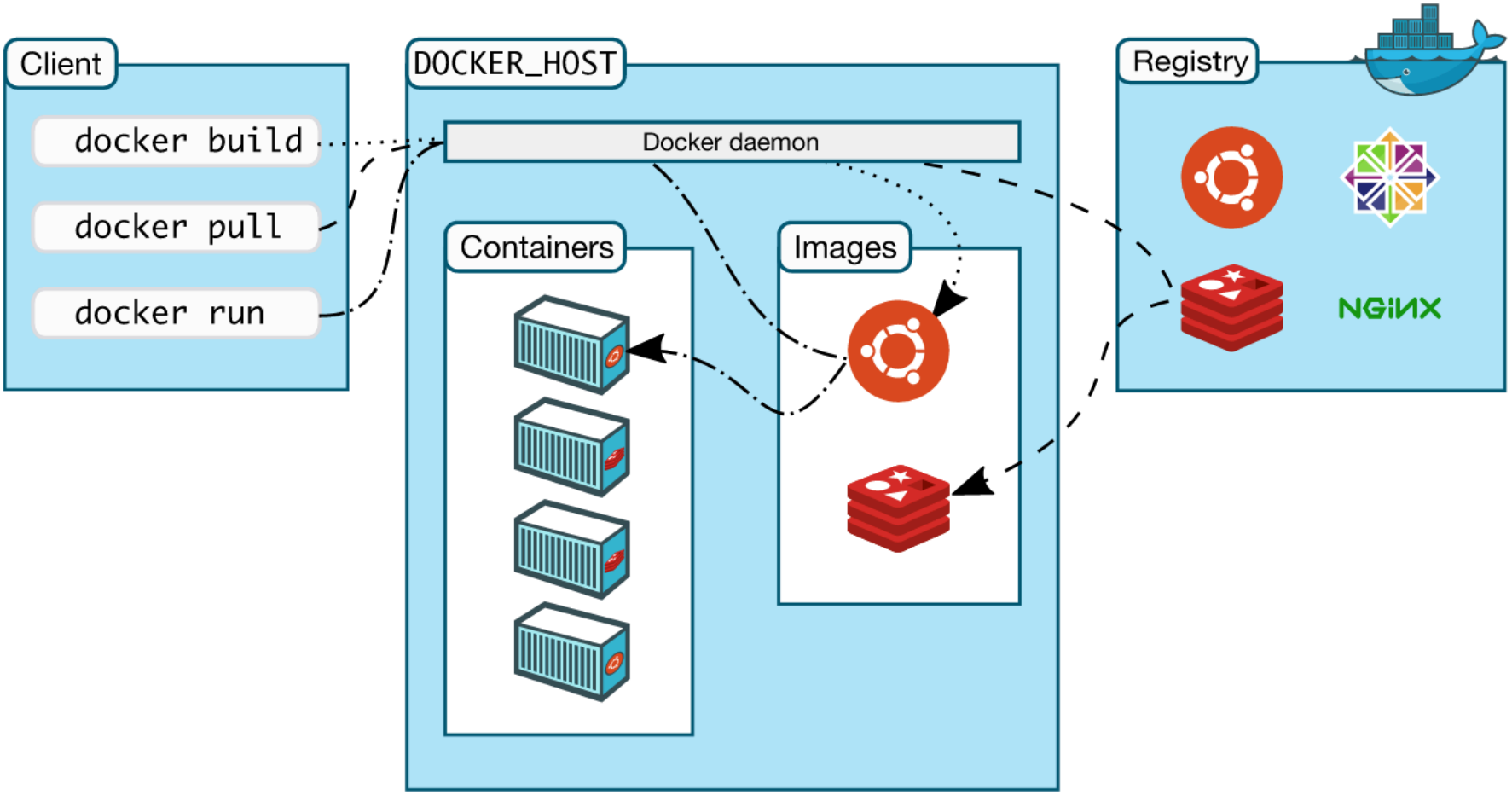
Docker Hello World(s)



What is ‘Docker Engine’?



How is Docker Architected?



No ‘sudo’ Configuration

Done already in AWS instances

- 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

20 minutes

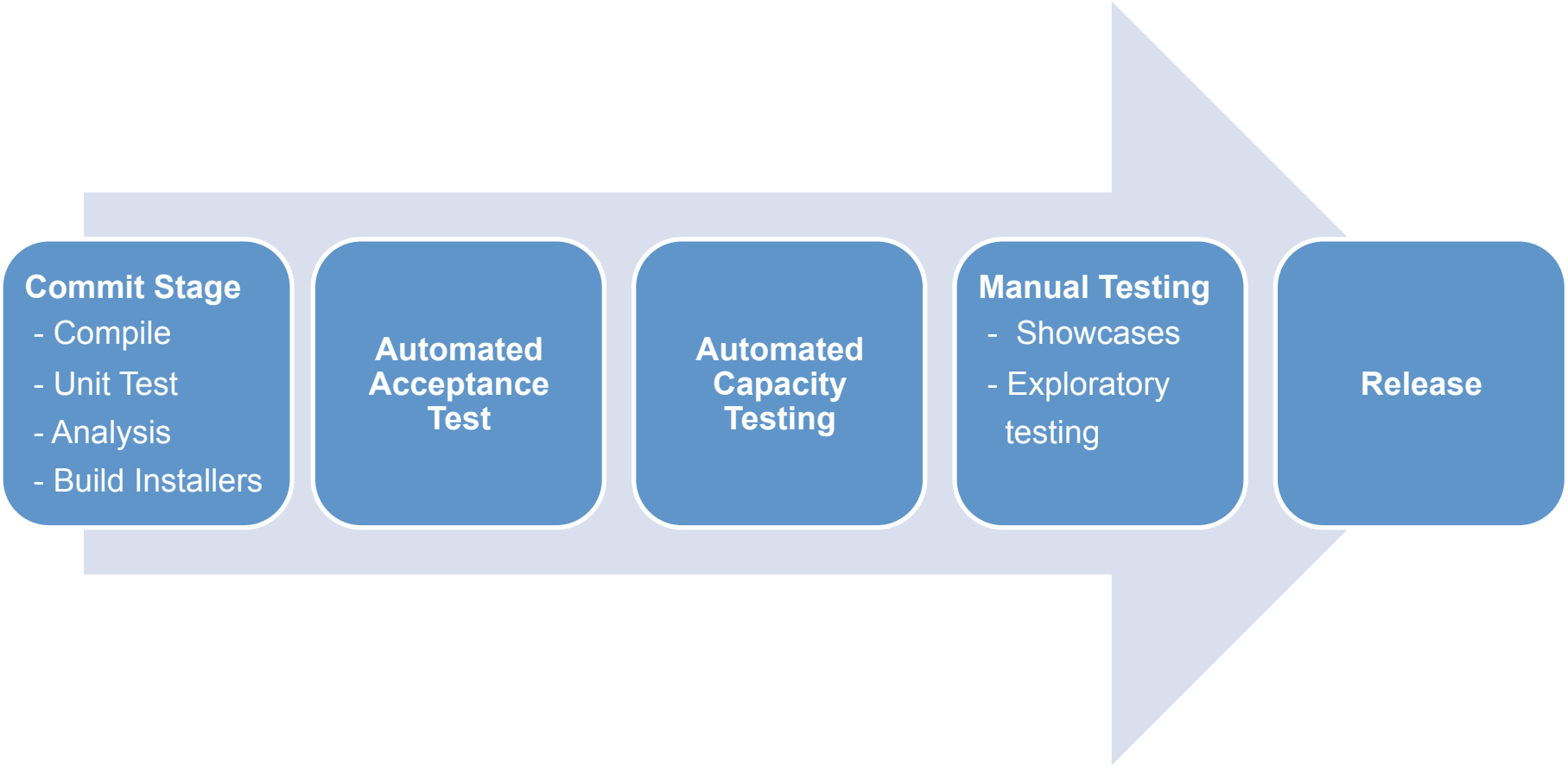
Exercise 2.1 in Docker Labs
https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

Do only exercise 2.1

So far !!

- Basic difference between VMs and Containers
- Installing Docker
- Images:- Hello World, whalesay, static-website in action

Where Containers are used?
- For automated testing & CICD



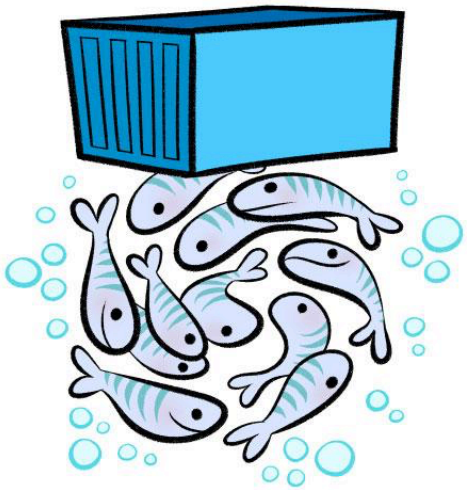
Where Containers are used?

- 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

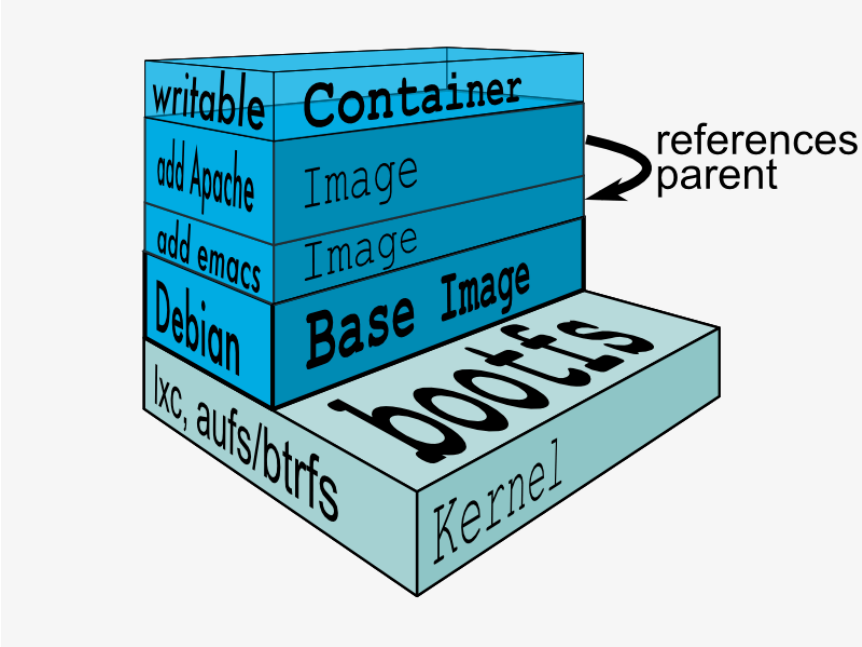
Docker Images



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

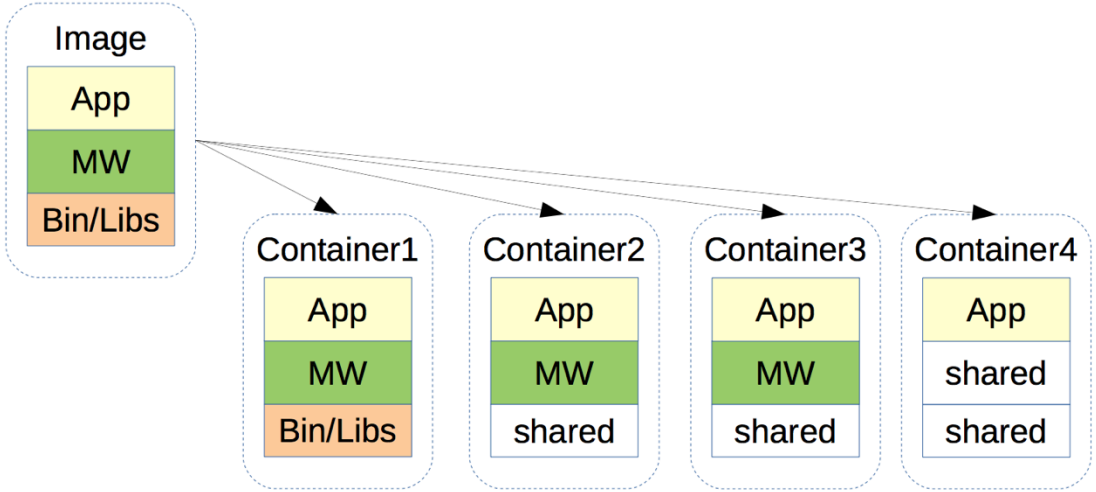


Example Docker Layers

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.

Benefits of Union File Systems

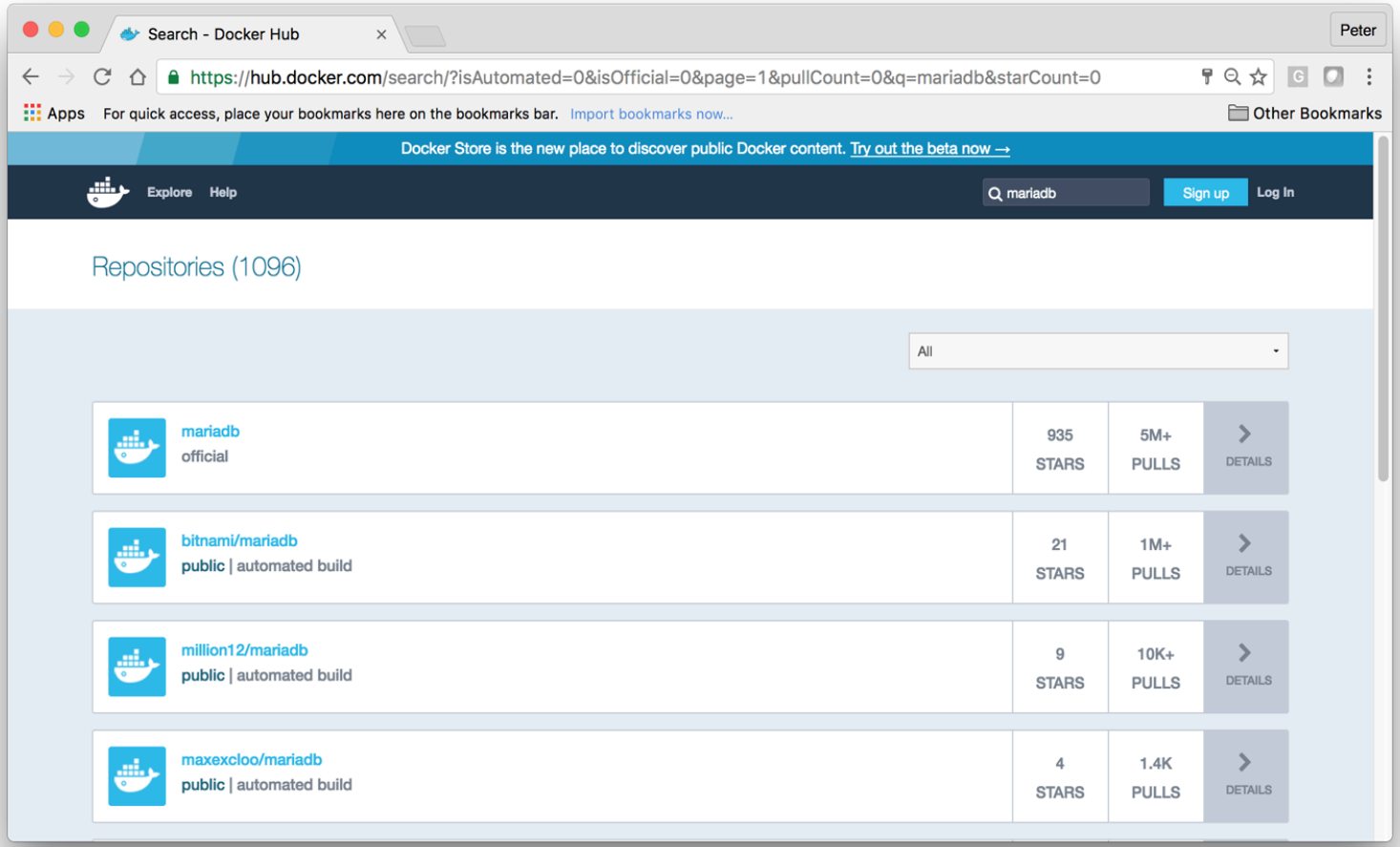
- Files are shared across containers
- Storage and memory spaces are saved
- Faster deployment of containers



Docker Hub

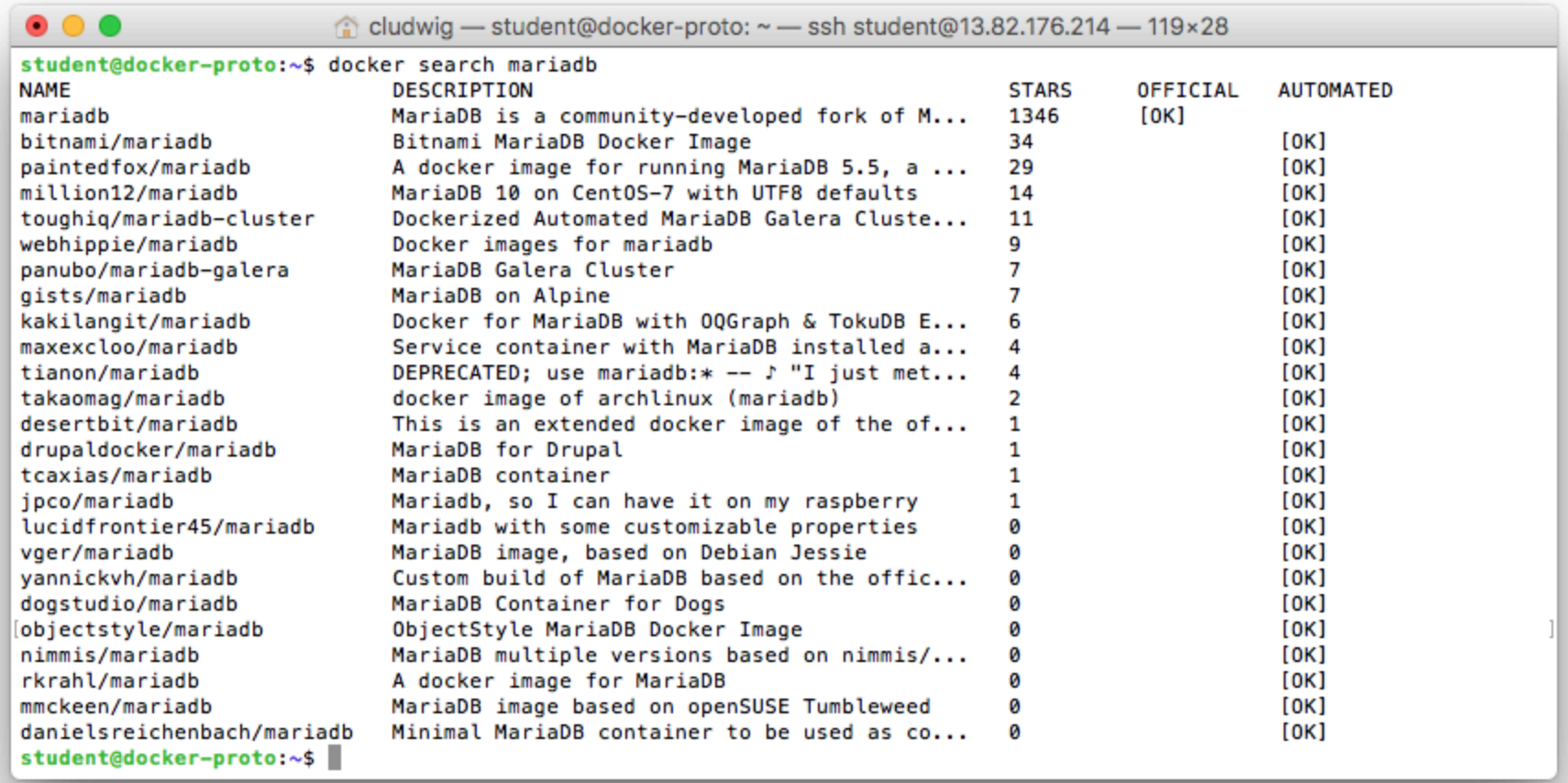
- Docker's first-party cloud-based registry
  - Hosts a broad repository of Docker images
  - Contains nearly any popular open source technology including MariaDB, Jenkins, Cloudera, etc
  - Contains 'Official images' from vendors such as Canonical, Oracle, Red Hat, etc
  - Provides one free private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



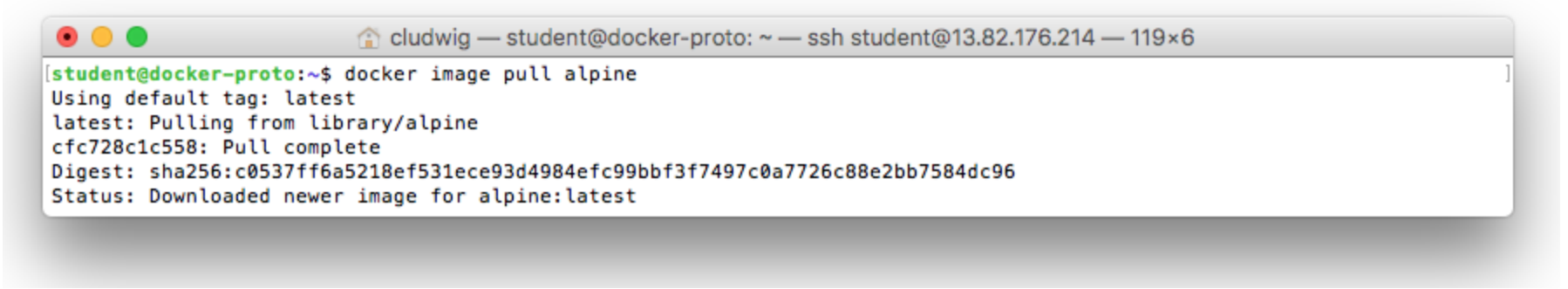
Docker image commands

```
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

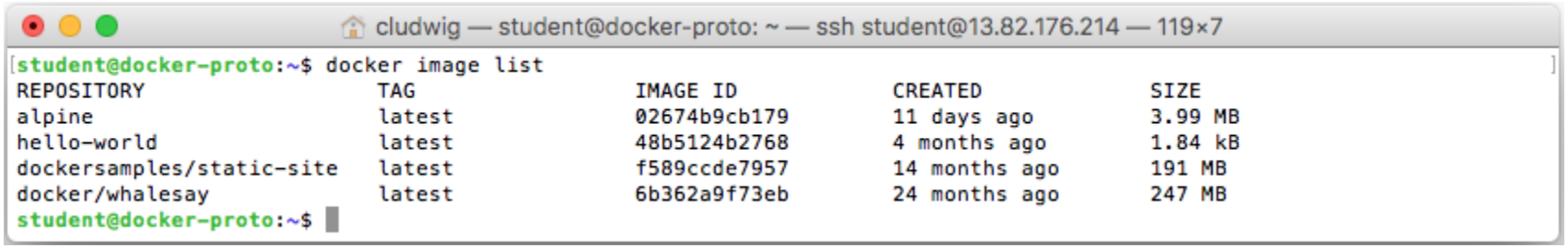
```
ubuntu@ip-10-0-0-117:~/code/multistage$ docker container --help

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 [imageName]

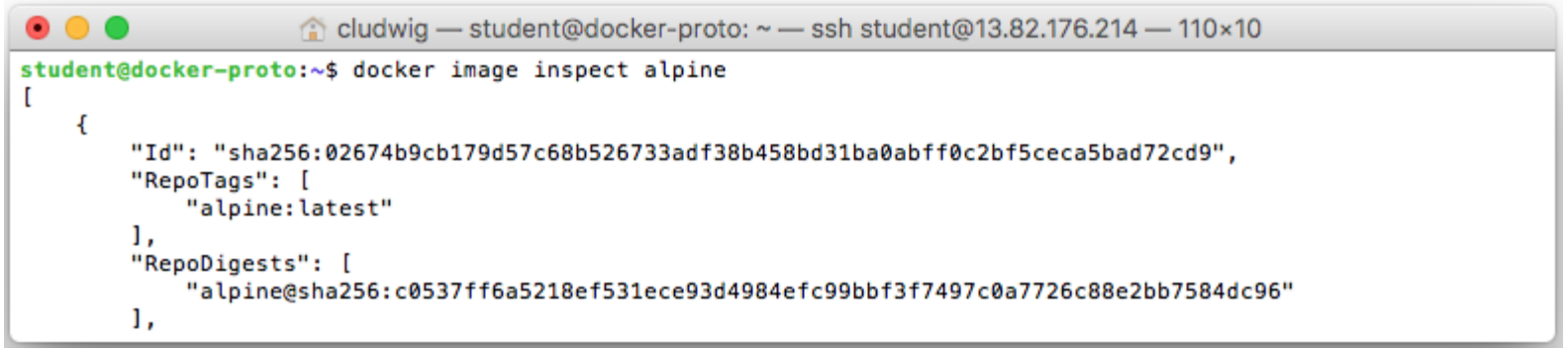


Best Practice: Choose the smallest base images possible

Debian – 123 MB
Alpine – 5 MB

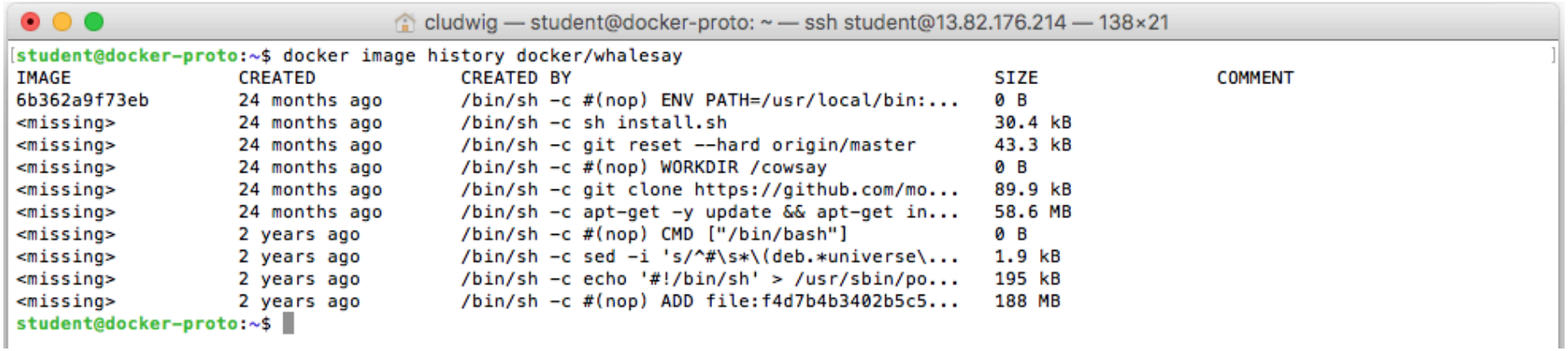


$ docker inspect [image|container]



- **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



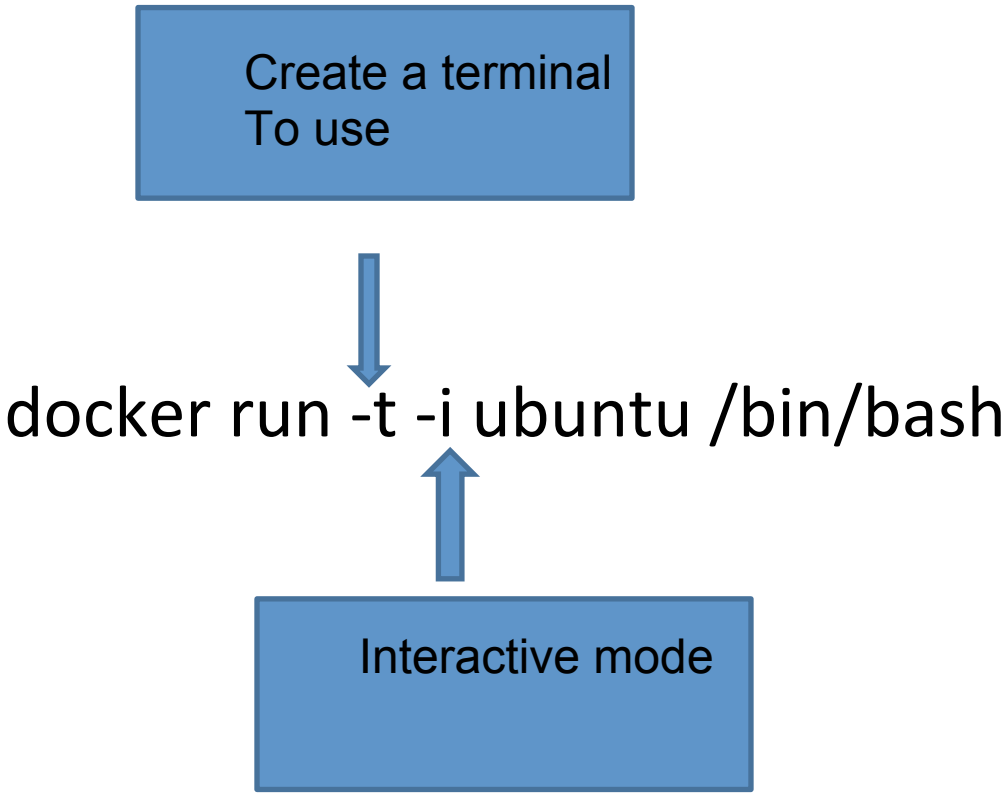
Use history to view layers in an image

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

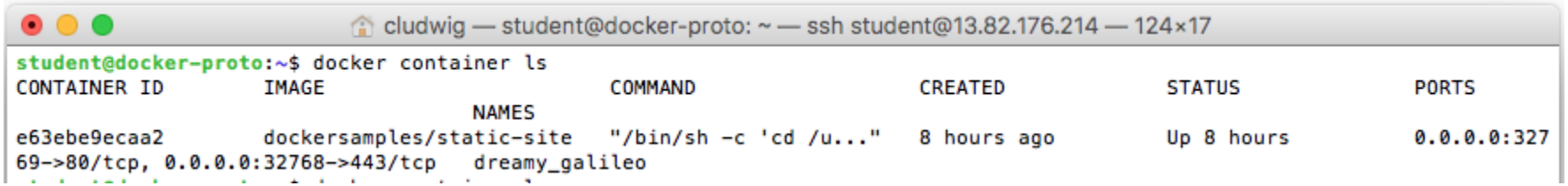
Docker Interactive Mode

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

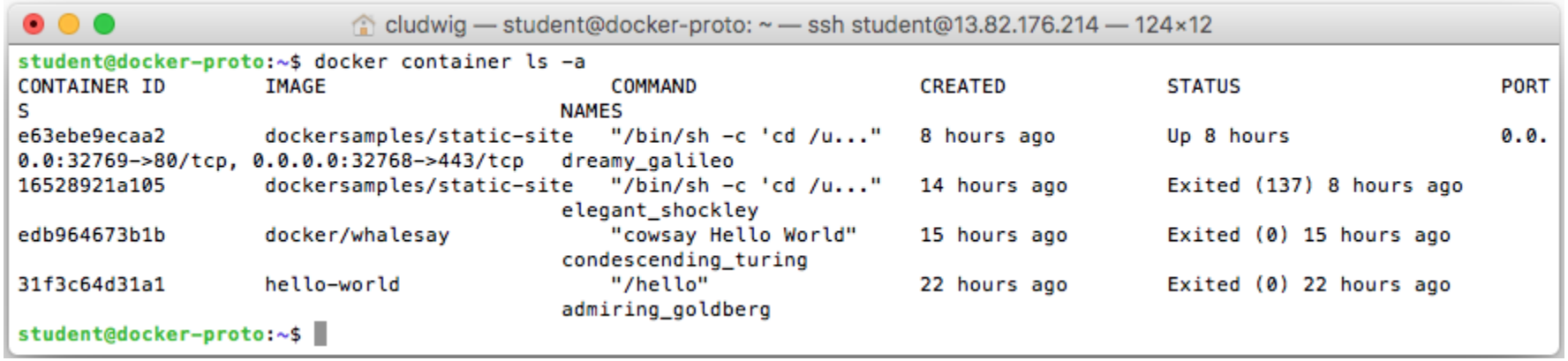
```
docker run -it alpine /bin/sh
```

See all containers

- $ docker container ls #show running



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



$docker container stats

```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 138×34
[student@docker-proto:~$ docker container ls
CONTAINER ID        IMAGE                      COMMAND                  CREATED             STATUS              PORTS
                    NAMES
e63ebe9ecaa2        dockersamples/static-site  "/bin/sh -c 'cd /u..."   8 hours ago         Up 8 hours          0.0.0.0:32769->80/tcp, 0.
0.0.0:32768->443/tcp   dreamy_galileo
[student@docker-proto:~$ docker container stats e63
```

```
cludwig — student@docker-proto: ~ — ssh student@13.82.176.214 — 138×34
CONTAINER           CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
e63                 0.00%               1.562 MiB / 667.2 MiB   0.23%               4.29 kB / 12.4 kB   73.7 kB / 12.3 kB   3
```

- 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

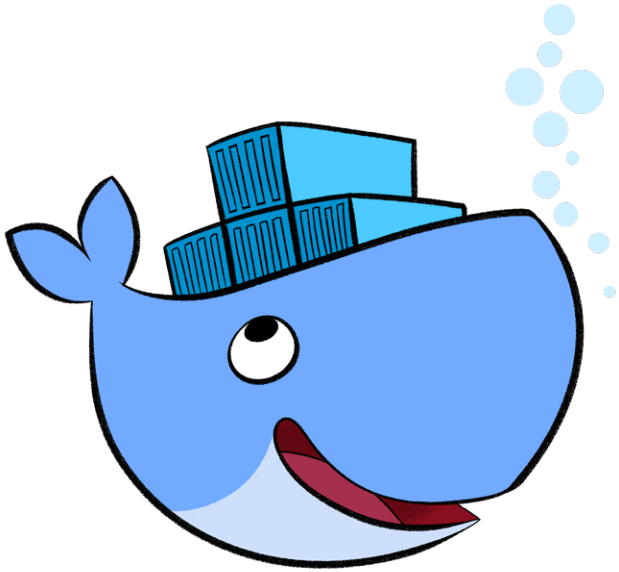
45 minutes

Practice Docker image and container commands

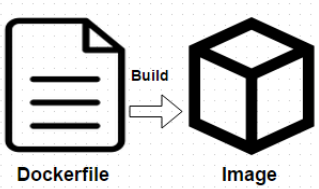
- https://github.com/shekhar2010us/microservices_monolithic_docker/blob/master/docker_image_container.md

**** Day 1:**
Exercise 2.1
4 – 5 pm ET

Docker Files



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
```

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

Docker Create your first image

CLASSROOM WORK 3.1

45 minutes

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

** Day 2:
Exercise 3.1, 3.2, 3.3
11 am – 1 pm ET

Dockerfile Exercises

CLASSROOM WORK 3.2

30 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.

Dockerfile Optimization

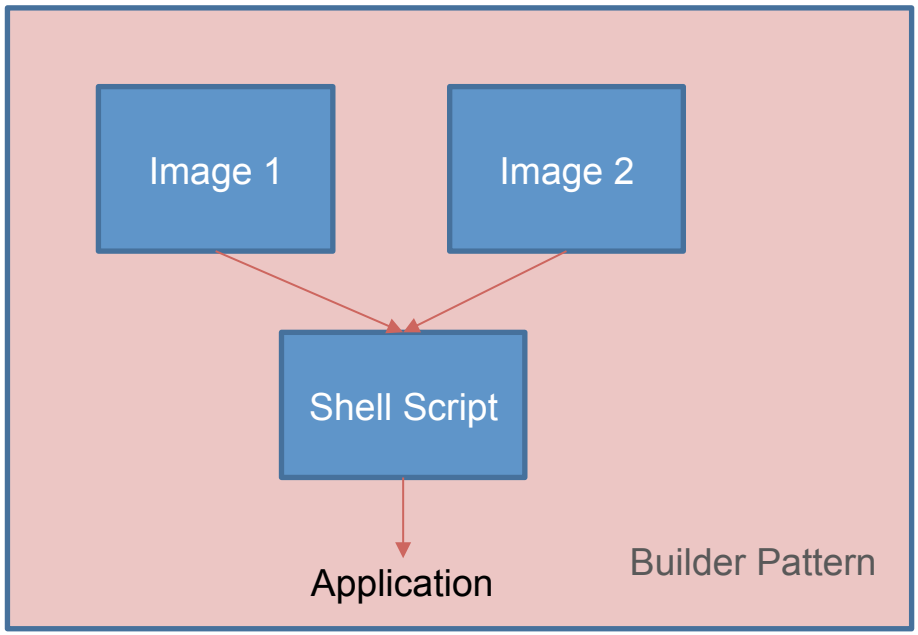
CLASSROOM WORK 3.3

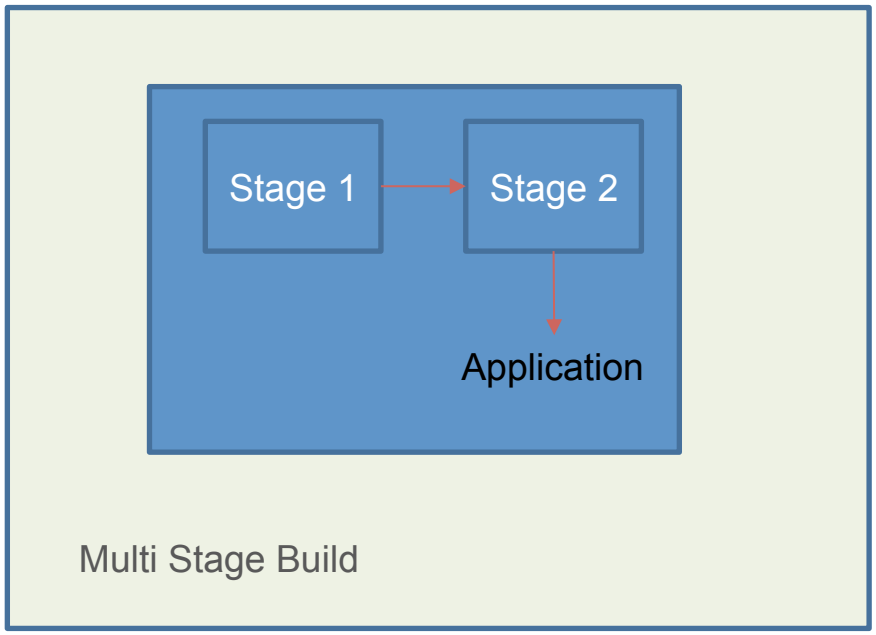
10 minutes

This exercise covers optimizing dockerfile to reduce the number of layers

https://github.com/docker/dceu_tutorials/blob/master/11-infrastructure-as-code.md

Multi-stage Build





- **Builder Pattern:** Common practice to maintain two dockerfiles. One for development and other for production that contains application and only things needed to run the application

This is not ideal !!!!

- **Multi-Stage Build**: One stage to another

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
```

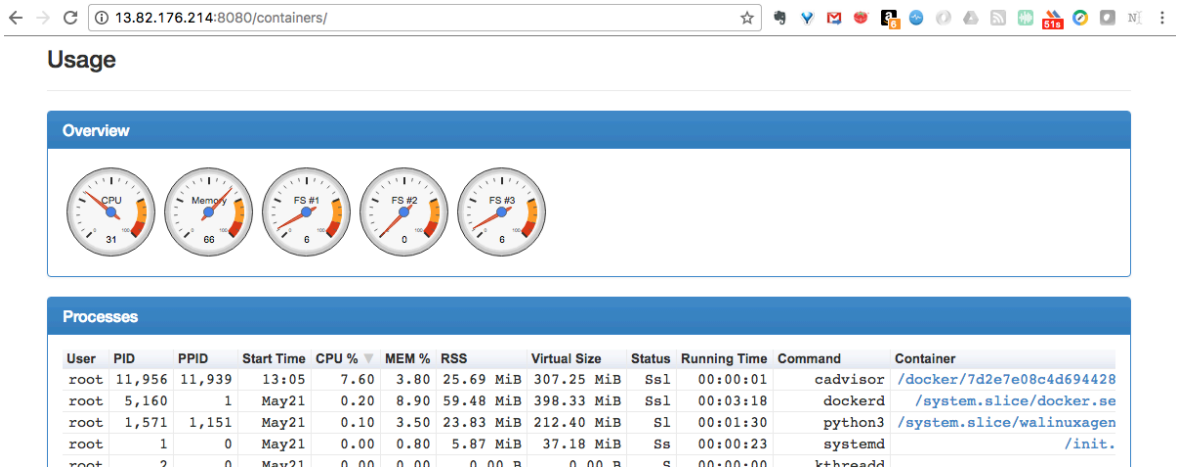
```
$ docker images

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

Docker 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

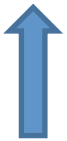
yourdockerhost:8080



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

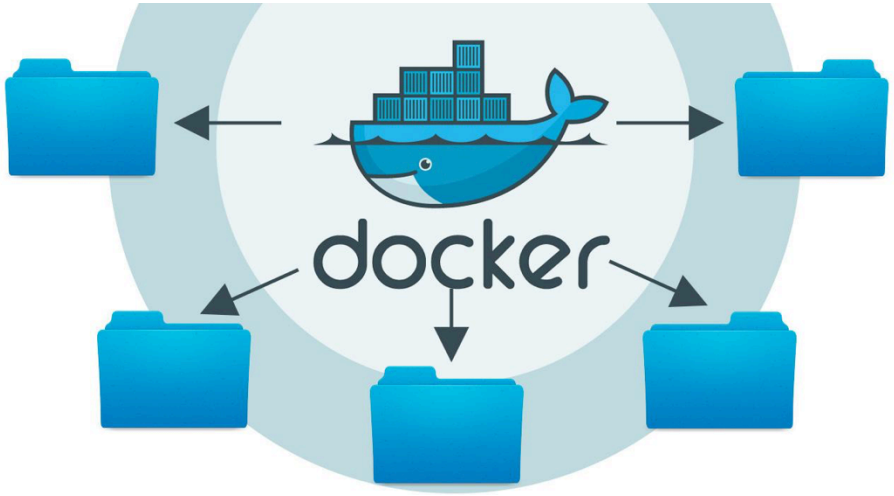
- Search Registry
  - https://docs.docker.com/registry/spec/api/#listing-repositories

```
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 Volumes



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 -it --name test1 –v data2:/data ubuntu
- You can share volumes from one container to another using **volumes-from** tag
- 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 -it --name test1 -v /home/ubuntu/src:/data ubuntu

This command mounts the host directory, /home/ubuntu/src, into the container at /data.

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

CLASSROOM WORK

30 minutes

https://goo.gl/gR6XTl

(https://github.com/docker/dceu_tutorials/blob/master/07-volumes.md)

Docker “--volumes-from”

volumes-from

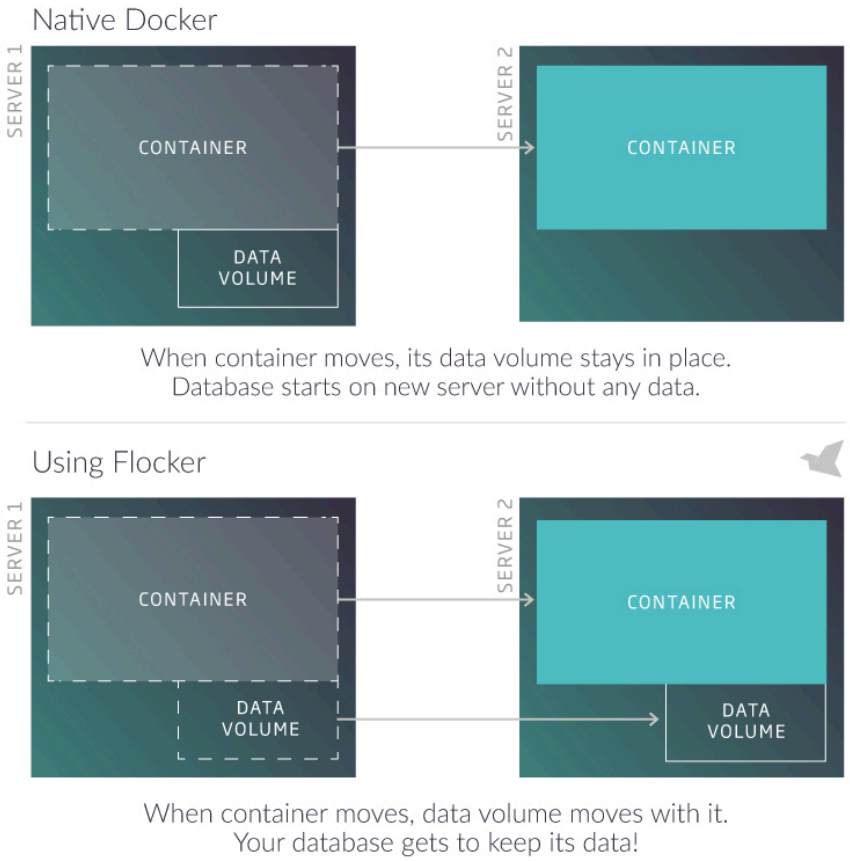
- docker volume create --name backup
- docker volume create --name logs
- docker run -it --name master -v backup:/backup -v logs:/logs ubuntu bash
- create sample files in backup & logs folders in container
- You can stop the container, but do not remove it
- Create another container using master volumes
  - docker run -it --name slave1 --volumes-from master ubuntu bash
- check the 2 folders ; exit

Remote Volumes

https://container42.com/2014/03/29/docker-quicktip-4-remote-volumes/

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
```

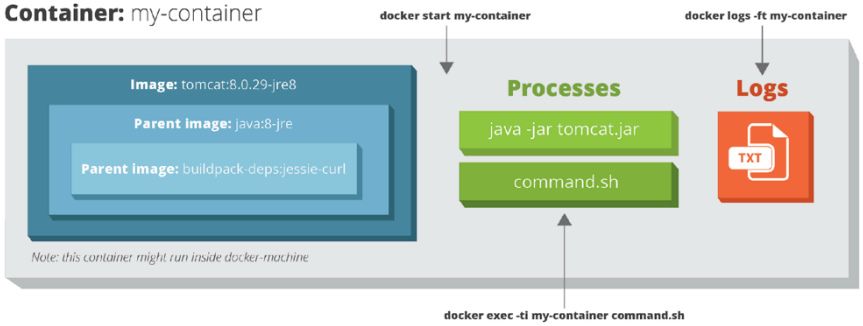
Create and start containers
```
docker-compose up
```

Interacting with a container

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

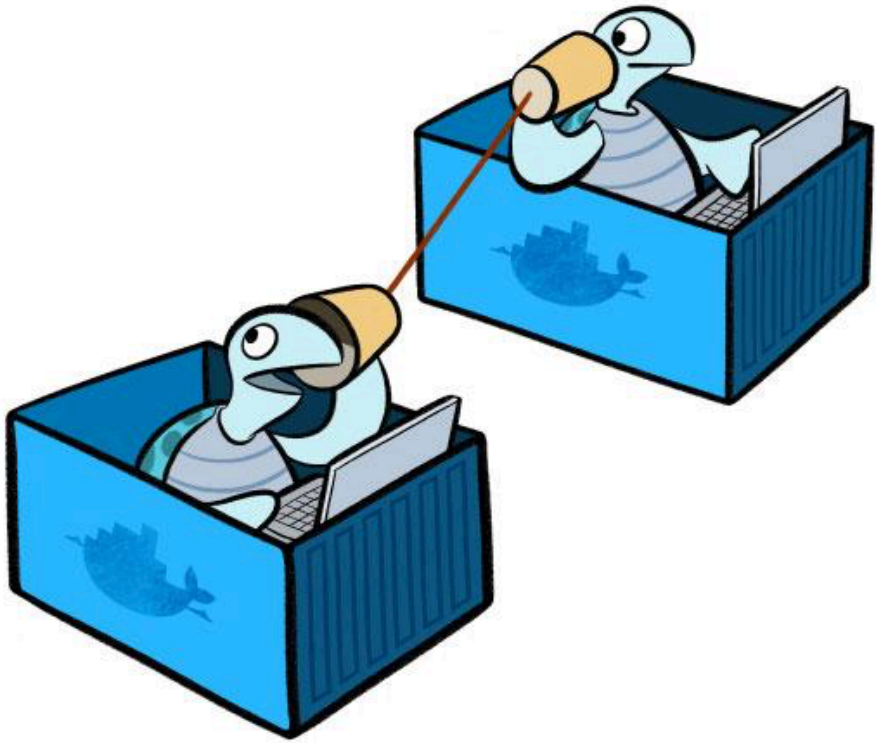
Follow the container logs
```
docker logs -ft container_name
```

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



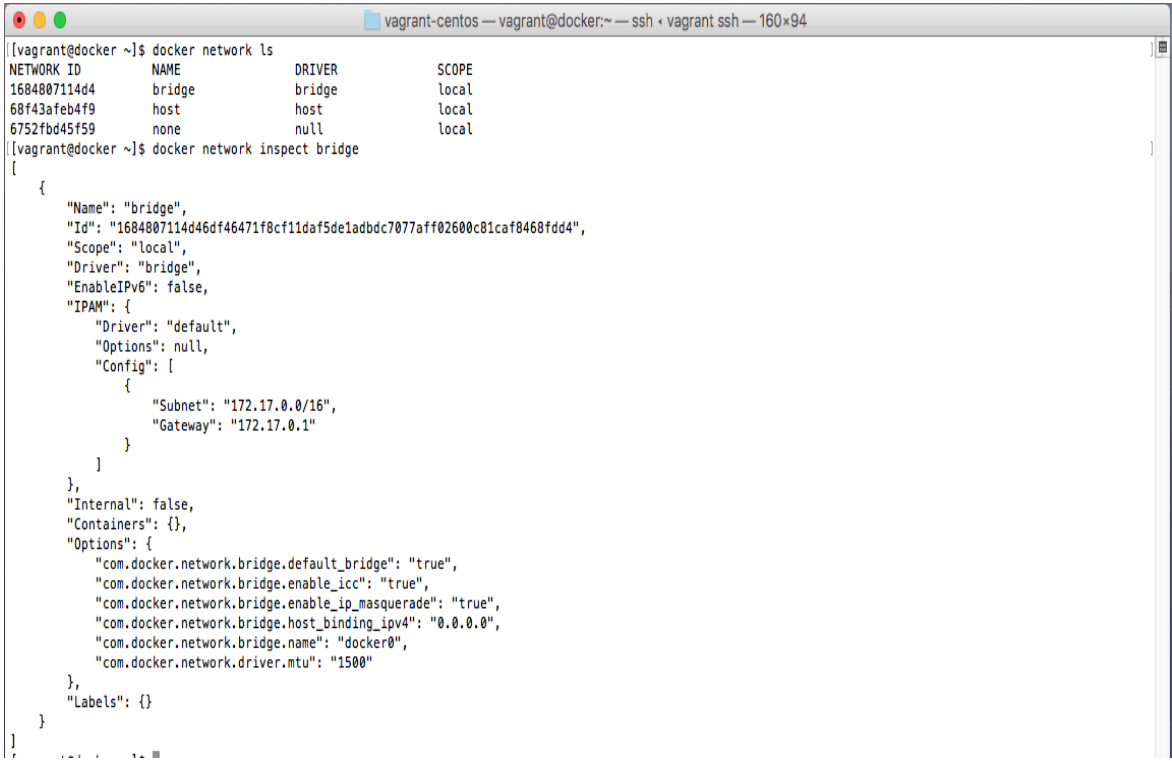
BROUGHT TO YOU BY JRebel

Docker Networking

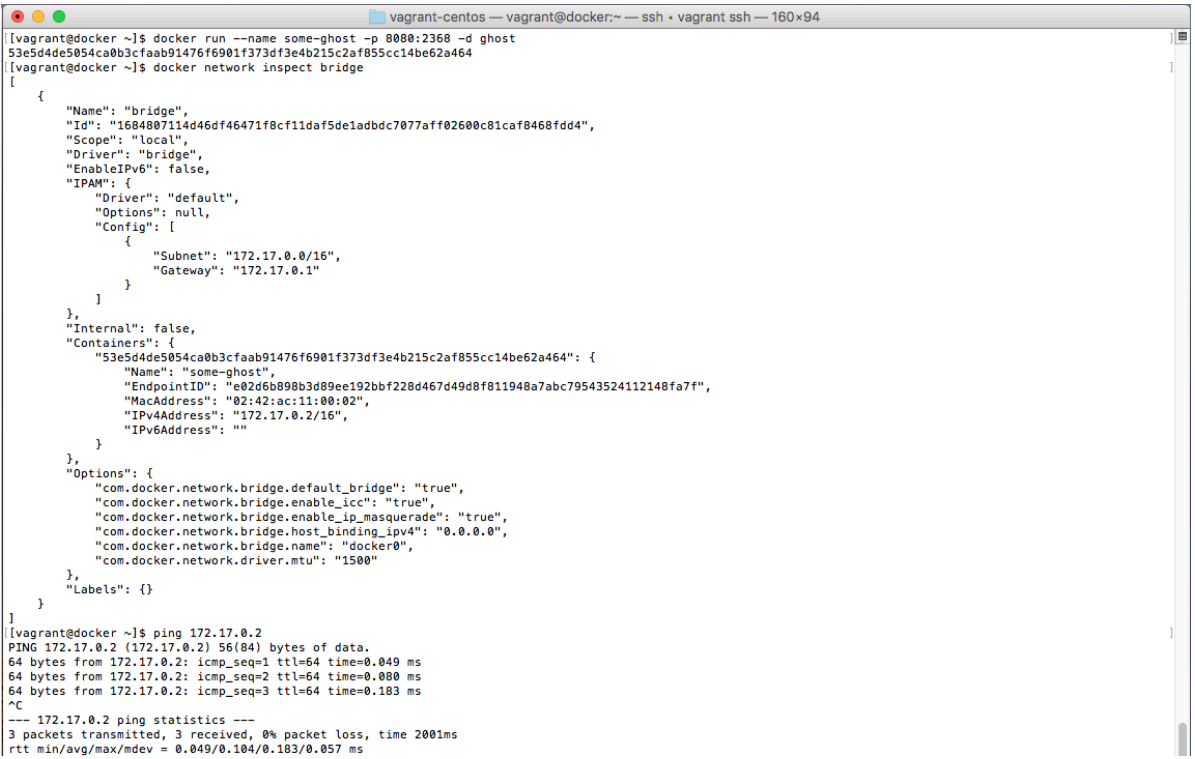


Docker Network Defaults

- 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

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:

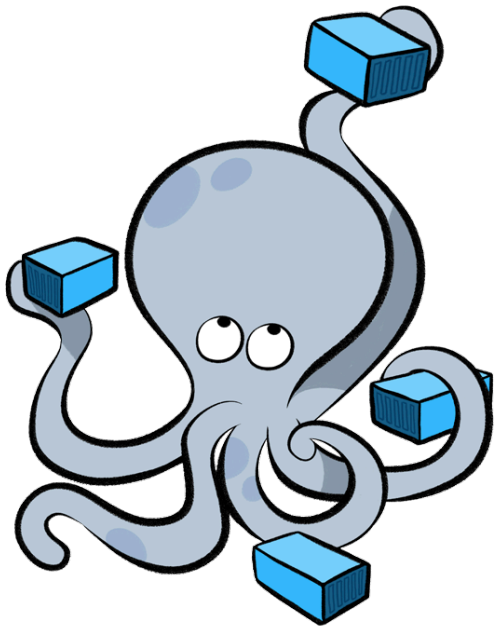
```
$ docker network create --opt encrypted --driver
overlay my-multi-host-network
```

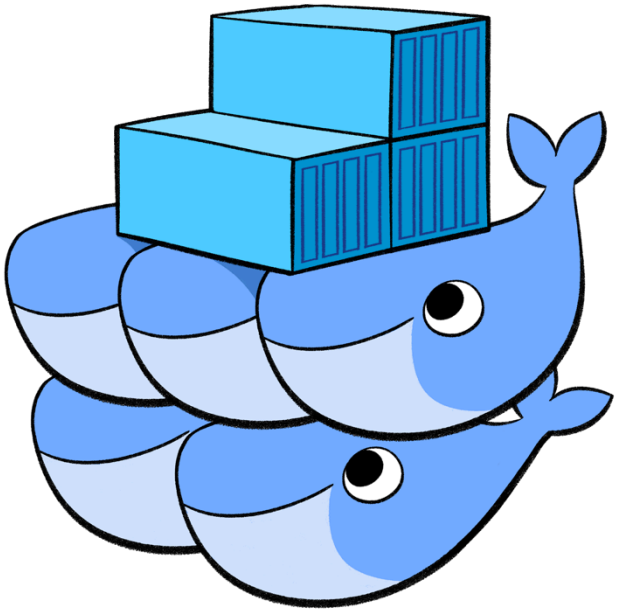
Docker Networks Exercise

- Run a redis container with default network
- Run a ubuntu container with default network
- Install redis-tools in host and check if you can connect to redis container and add some data
- Install redis-tools in ubuntu container and check if you can connect to redis container. (optional: create an image with redis-tools and ubuntu)
- Create a bridge network (bridgea)
- Run a ubuntu container with bridgea network and the new image, check if you can connect to redis – probably not
- Connect this new ubuntu container with “bridge” network, and check again if you can connect to the redis container

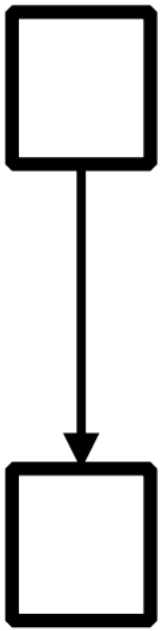
** keep inspecting network and container after every step

Docker Compose/Swarm



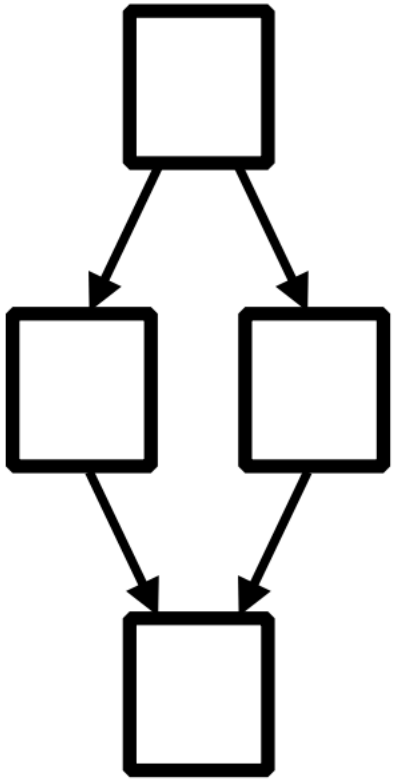


Launching multiple containers is clumsy



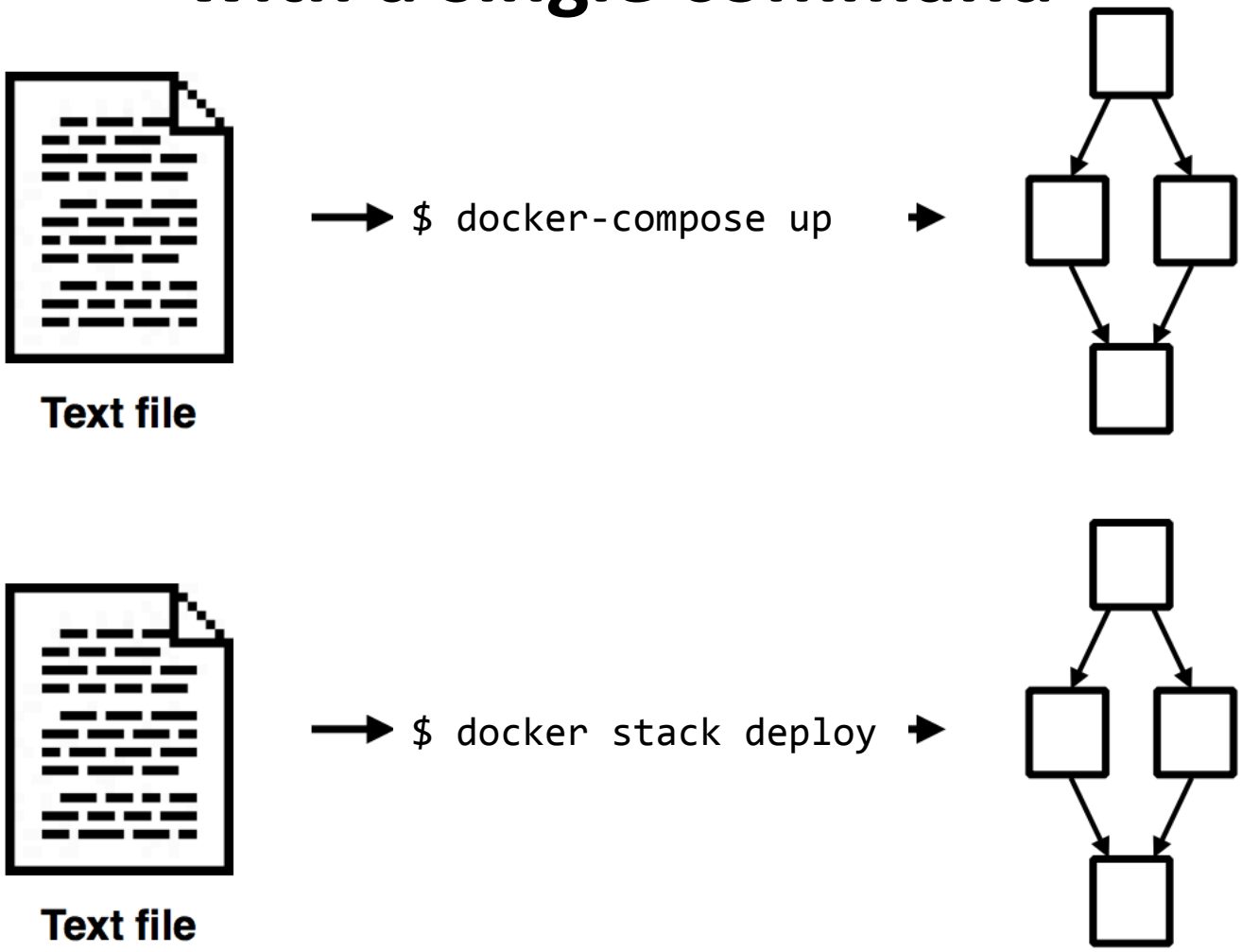
```
$ docker pull redis:latest

$ 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

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

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/

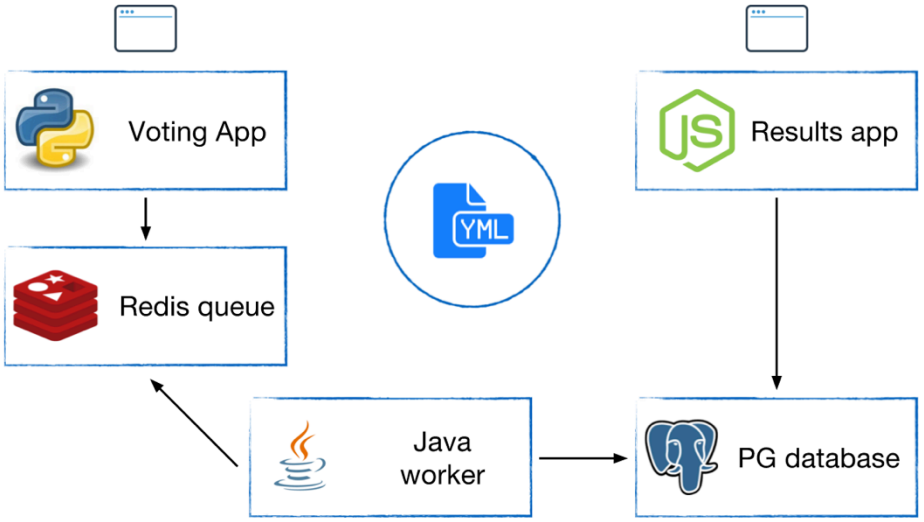
Docker Swarm commands

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

CLASSROOM WORK

Exercise 3.0 in Docker Labs
https://github.com/docker/labs/blob/master/beginner/chapters/votingapp.md



Docker Security

Security For Docker Images

- Secure Registry/Mirror Access
- Getting trustworthy images
  - trusted sources
    - docker hub / docker store
      - official images
      - pull by digest
    - private registry
- Docker Content Security
  - Docker Notary
    - Digitally signed images
      - "only signed content in production"

Docker Content Trust

CLASSROOM WORK

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

https://github.com/docker/labs/blob/master/security/trust/README.md

(Skip Step 5)

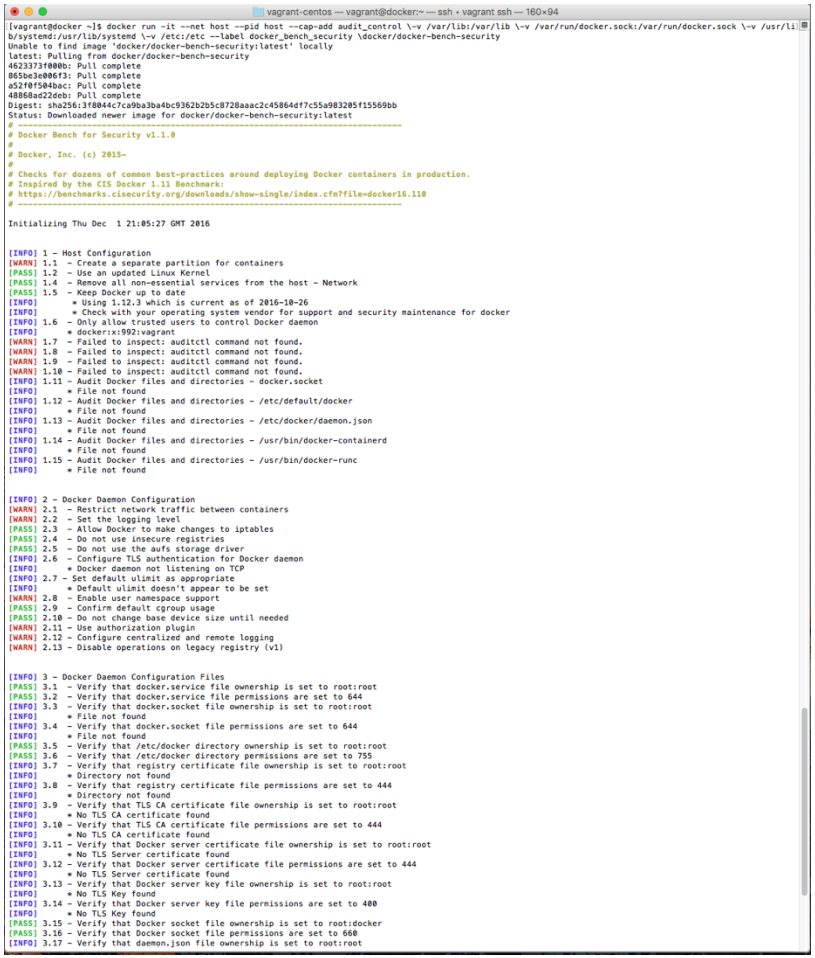
Docker Bench Security

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
```



10-15 Minutes

Docker Bench Results Discussion

https://www.nearform.com/blog/securing-docker-containers-on-aws/
https://www.cisecurity.org/cis-benchmarks/

(Some) Docker Security Best Practices

- 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.
  - Mitigate 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 inspect --format='{{.LogPath}}' <containerid> => push to ELK for monitoring

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

https://github.com/docker/labs/blob/master/security/scanning/README.md

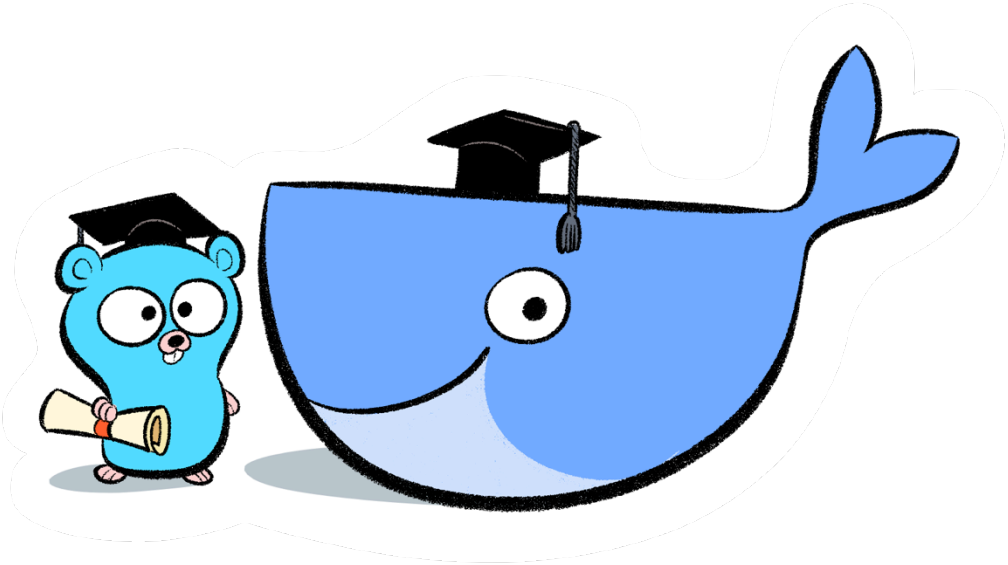
Clair by CoreOS

- 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

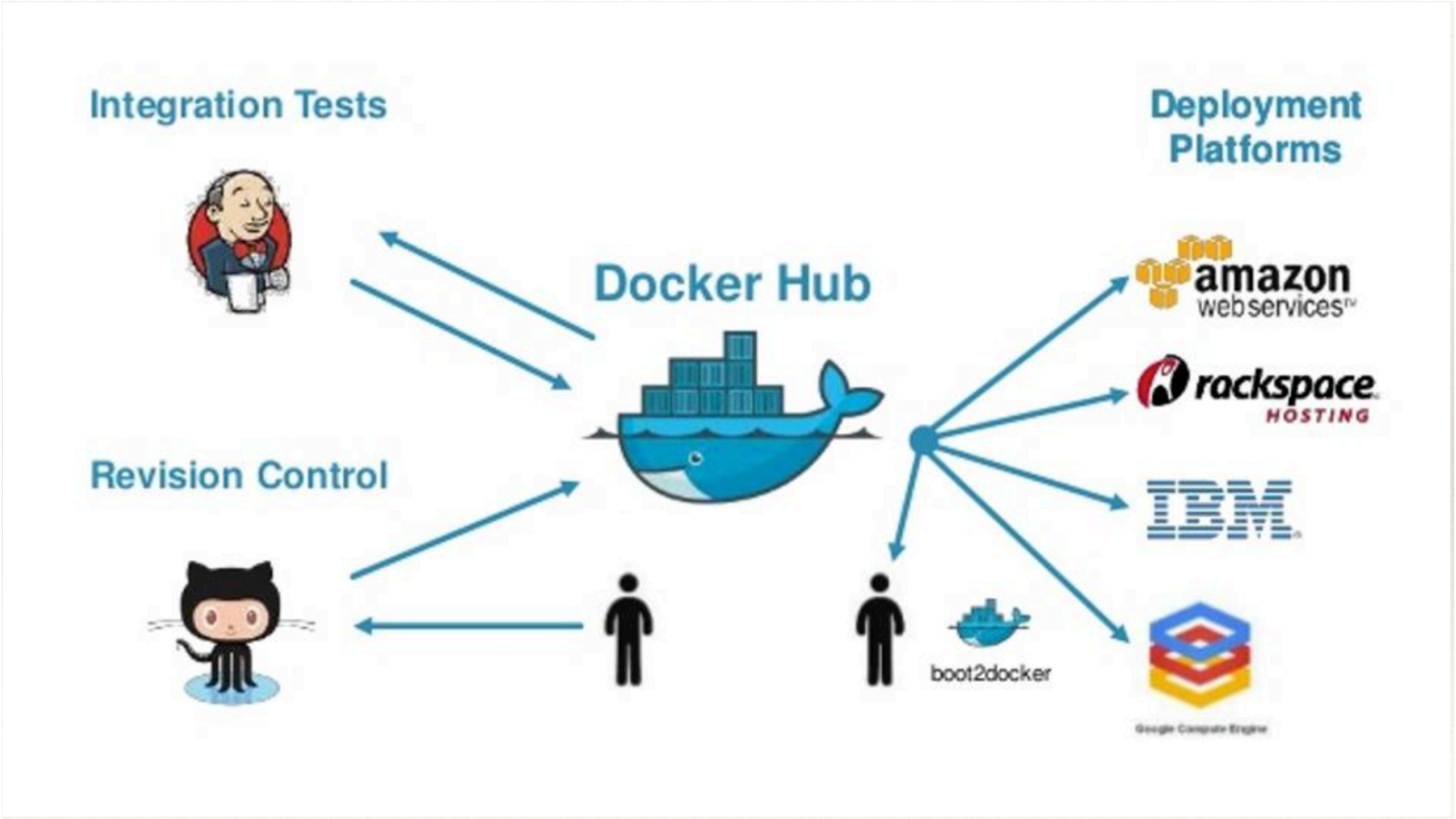
Docker Security

- 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

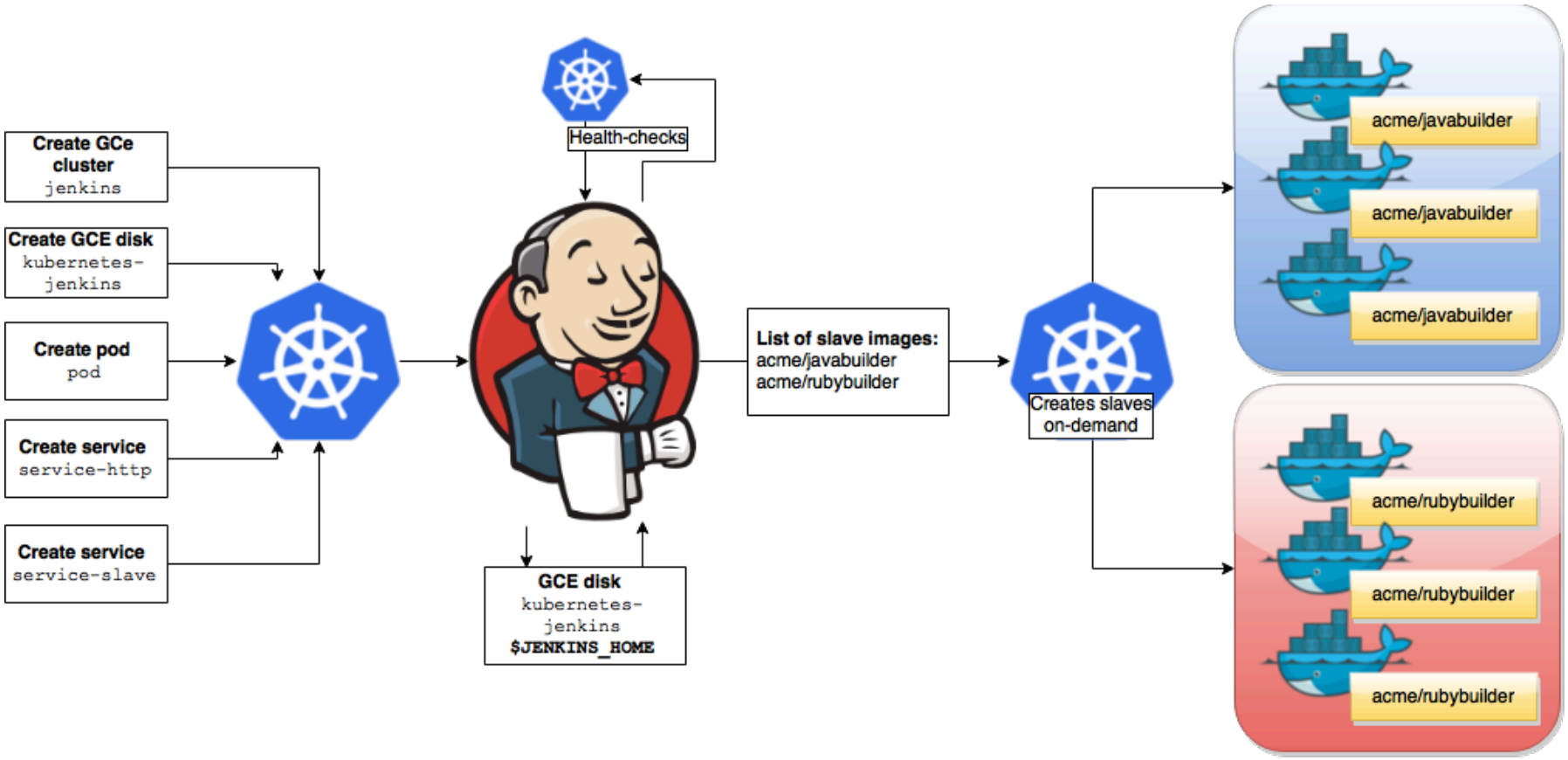
Docker Patterns



Docker Patterns

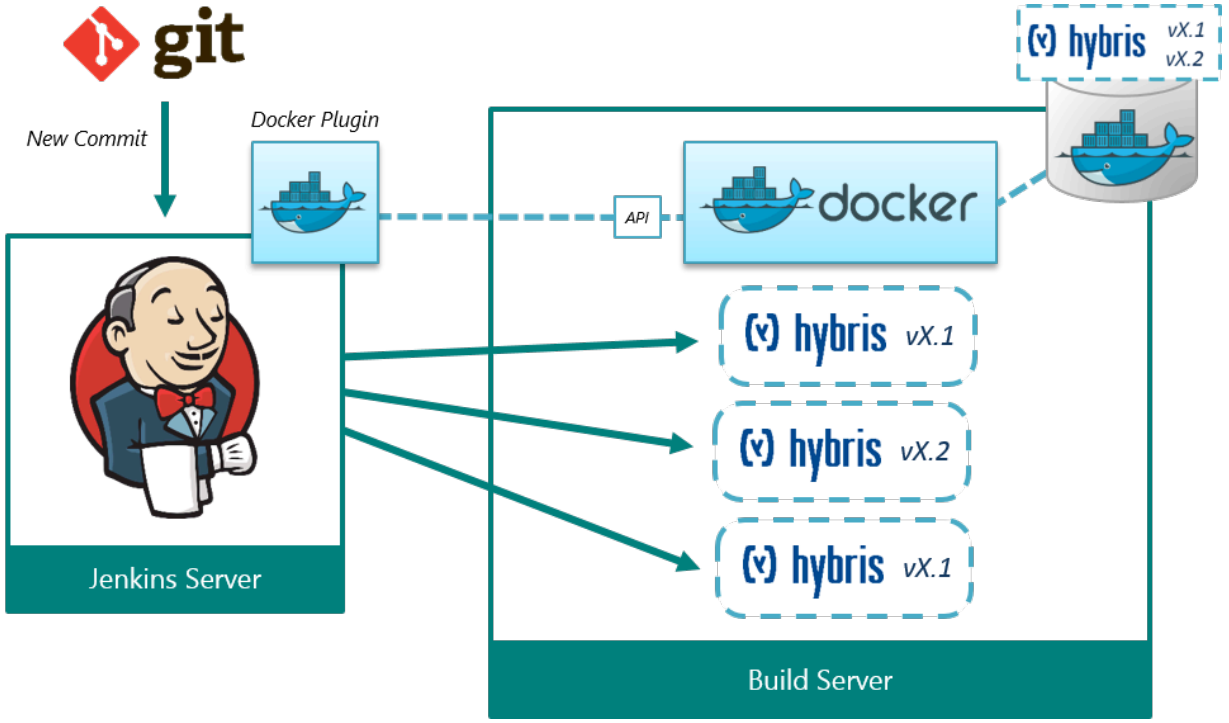


Jenkins Continuous Integration-Cloudbees



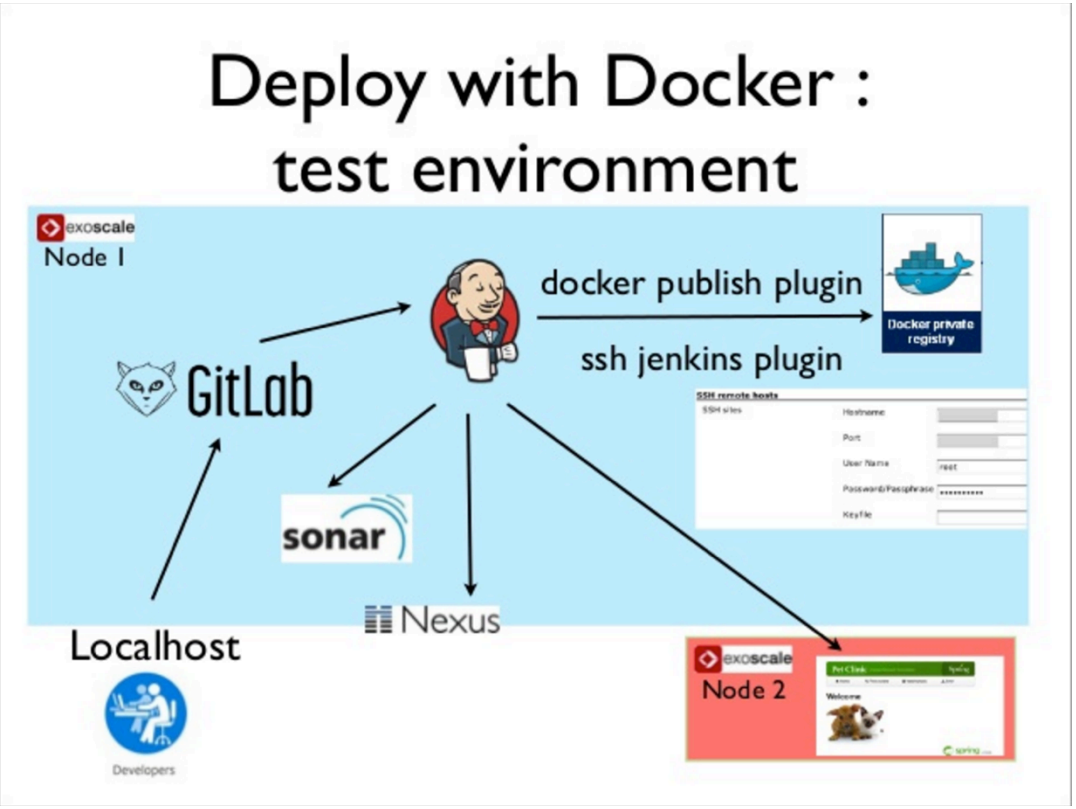
Link to Blog: Clustering Jenkins
http://bit.ly/2a092Xo

According to Michael Duke



Link to Blog: Docker as Jenkins Build Solution
http://bit.ly/2a57Za8

According to Julia Mateo

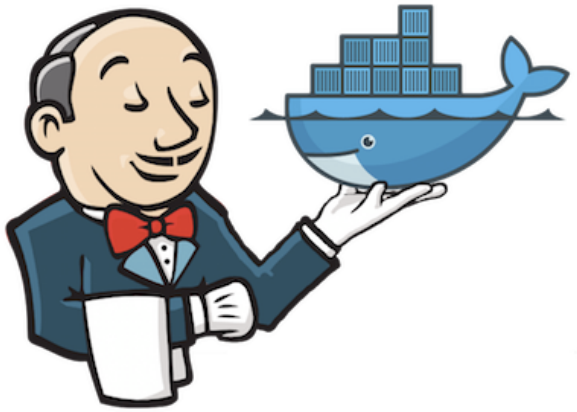


Link to Presentation: Continuous Delivery
http://bit.ly/2aHZj9o

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

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/papaludwig/docker-tutorials/tree/master/jenkins

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

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/papaludwig/docker-tutorials/tree/master/sharedvolume

Docker and Jenkins, Part 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/papaludwig/docker-tutorials/tree/master/ciexample

Docker and Jenkins, Part 4

CLASSROOM WORK

Part 4, Call Continuous Integration Container from Docker Compose

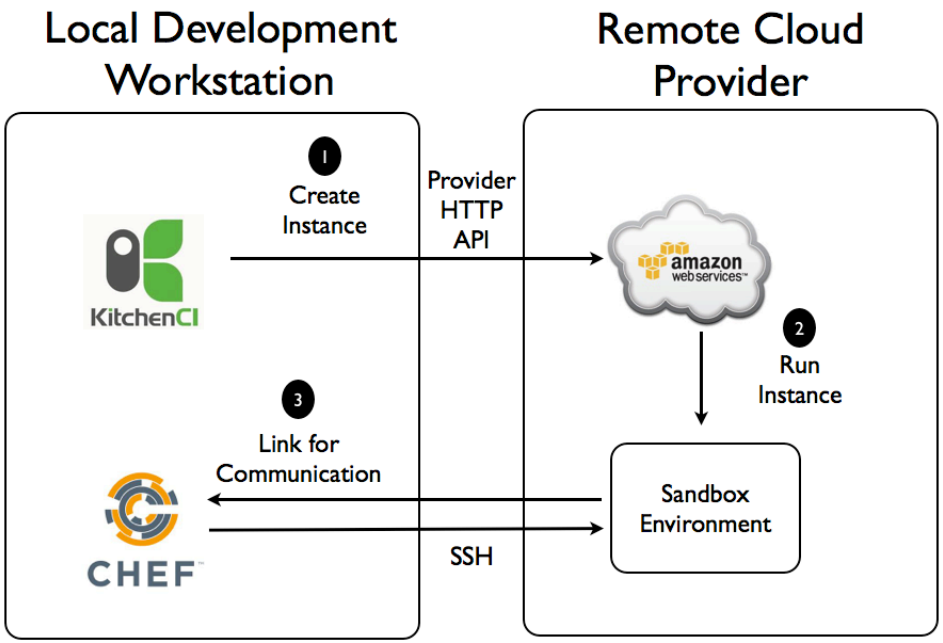
Take the working code from Part 3 and summarize it with a call to Docker Compose. Docker Compose is a much more convenient interface and scales better when multiple containers must be invoked.

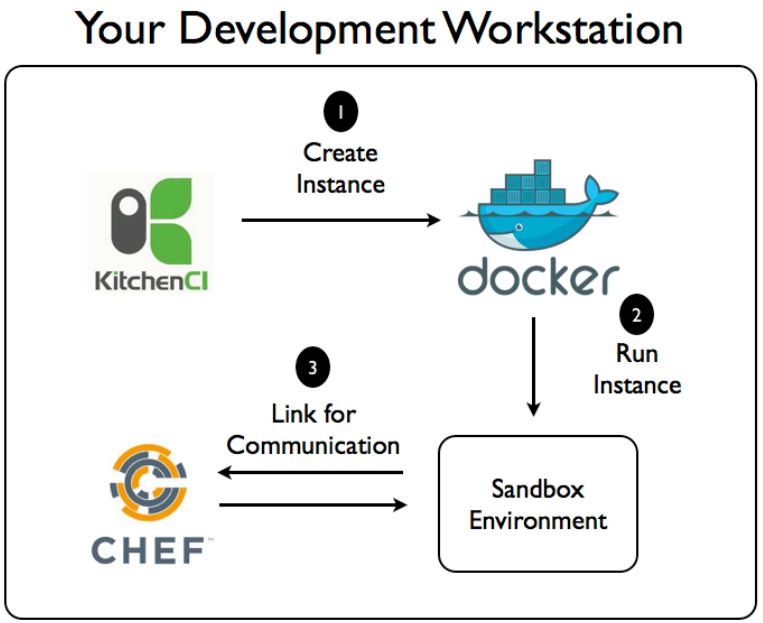
Bonus: Configure Jenkins container to skip setup workflow

Estimated time to complete: 15-25 min

Solution:
https://github.com/papaludwig/docker-tutorials/tree/master/compose

According to Mischa Taylor

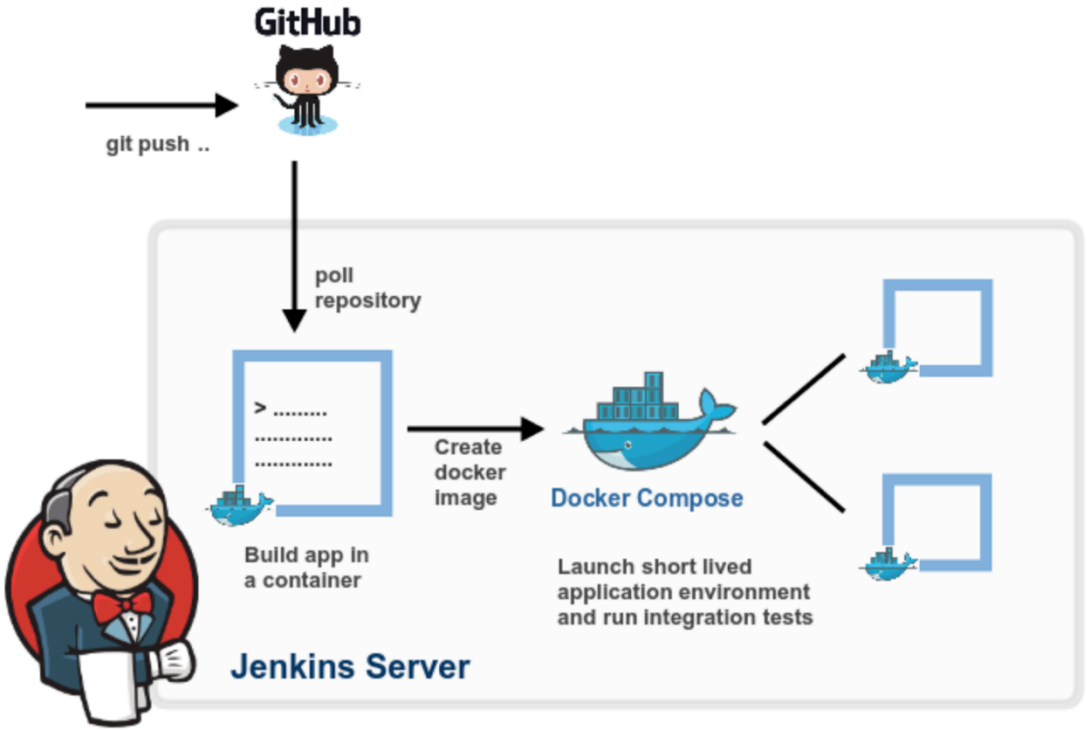




Link to Blog: Survey of Test Kitchen Providers
http://bit.ly/2a56kS2

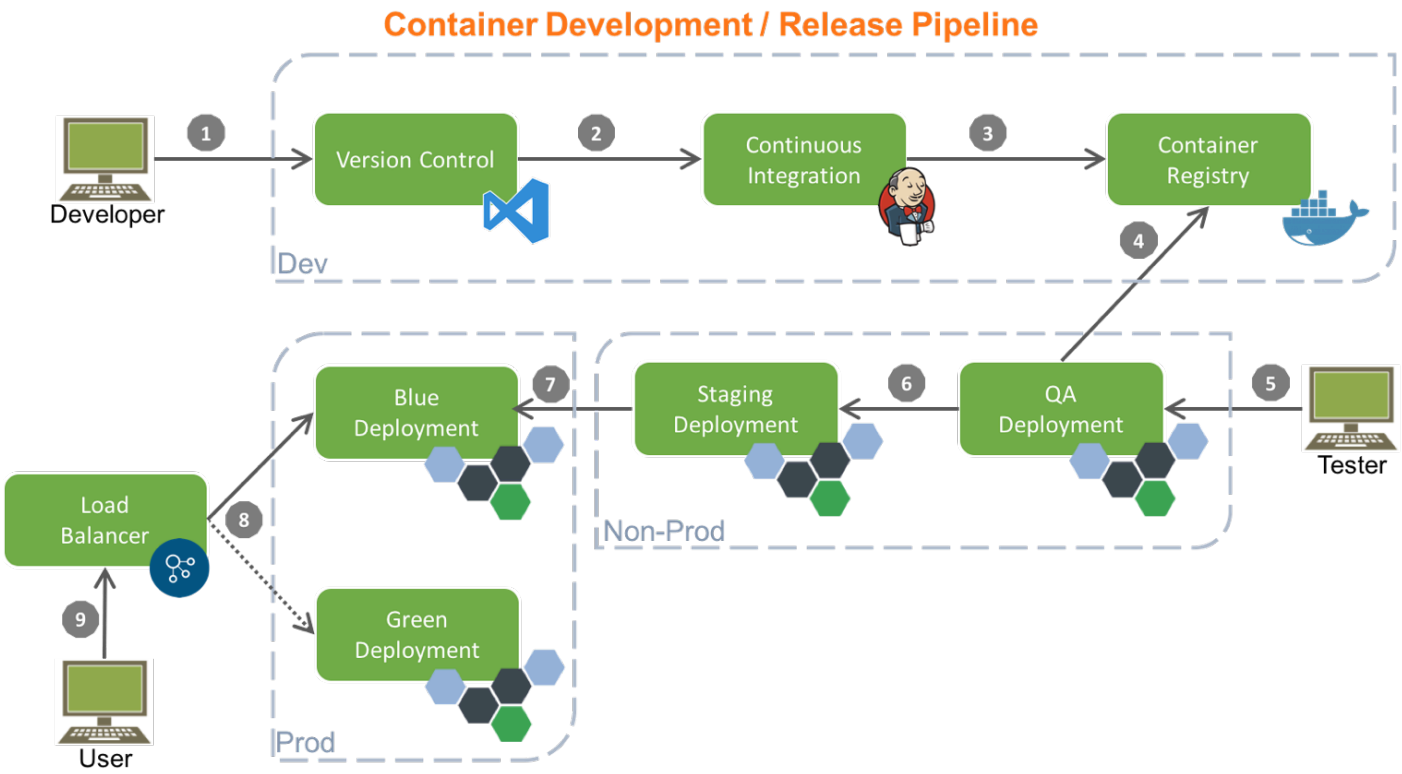
According to Rancher

RANCHER



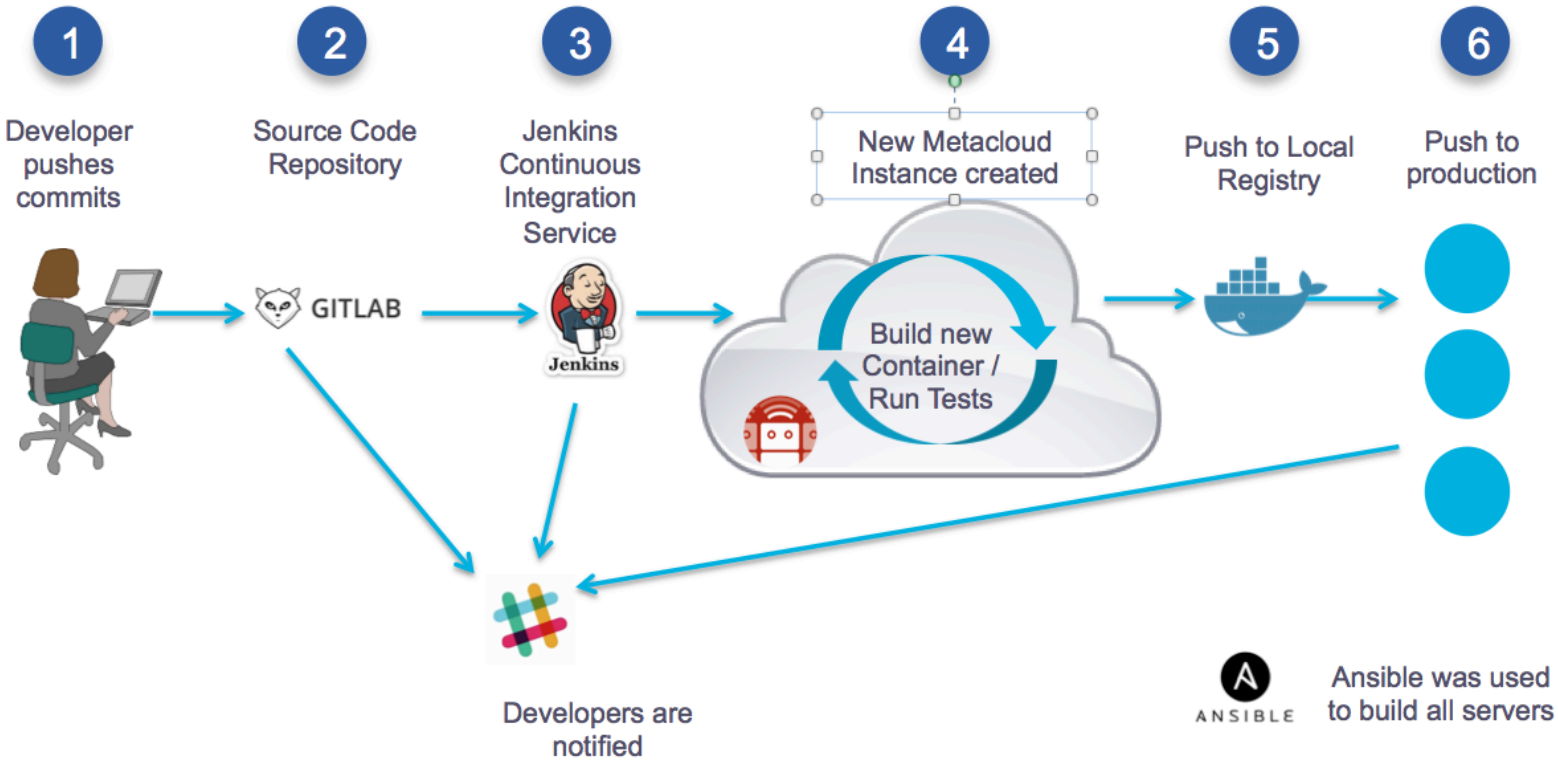
Link to Blog: Docker-based Build Pipelines
http://bit.ly/2aHtu0K

According to Robert Greiner



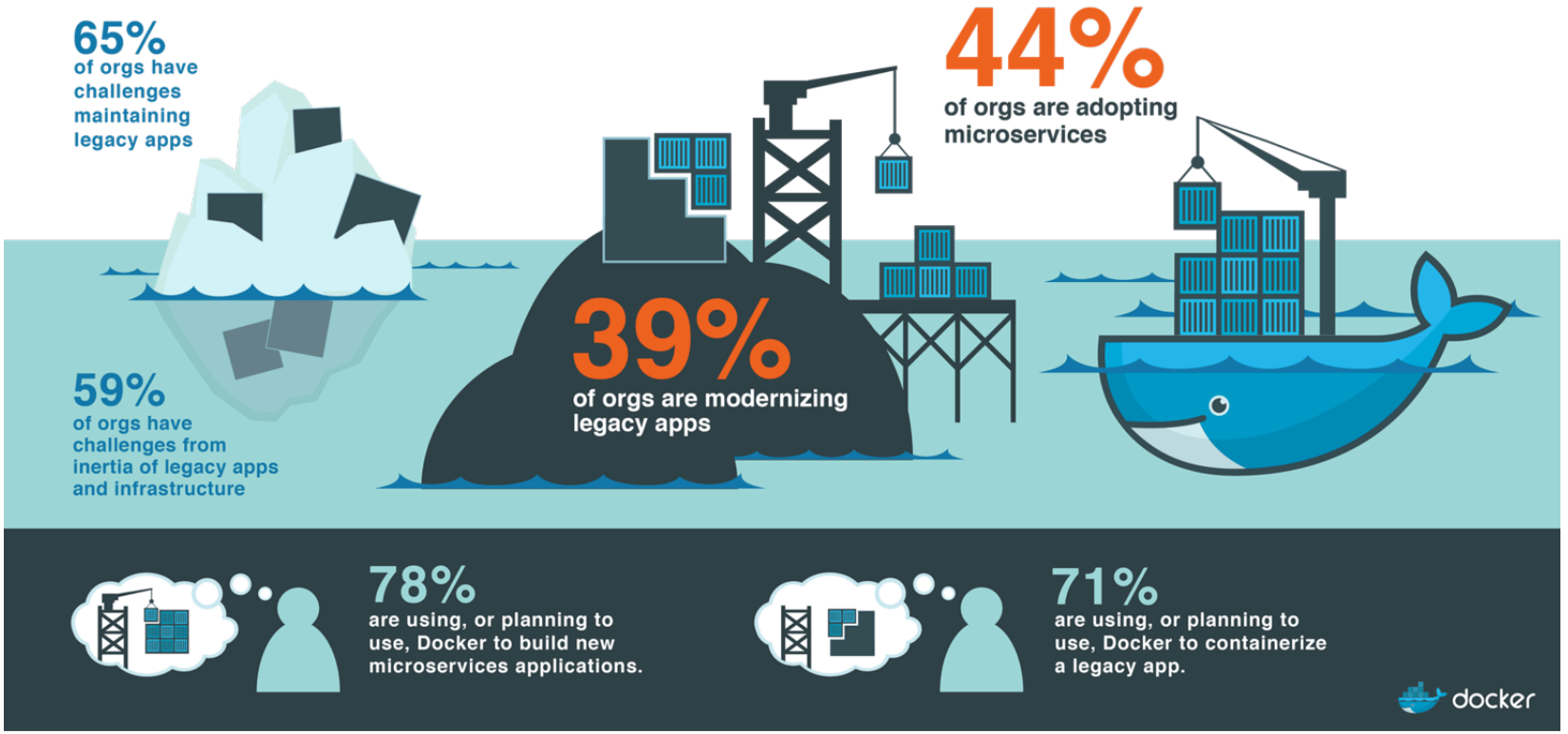
Link to Blog: Continuous Integration with Docker
http://bit.ly/2aeA1io

According to Vallard



Link to Blog: Continuous Delivery of a Simple Web Application
http://bit.ly/2asS8G3

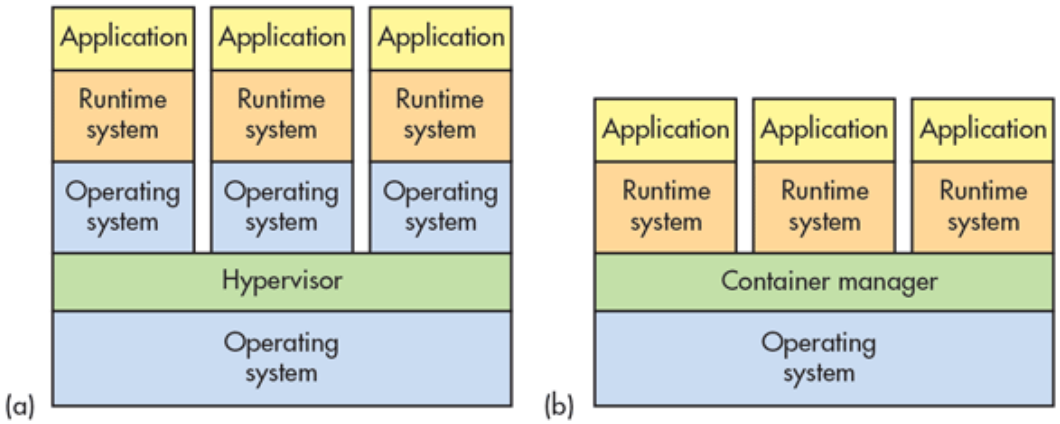
Docker Adoption & Usage Data



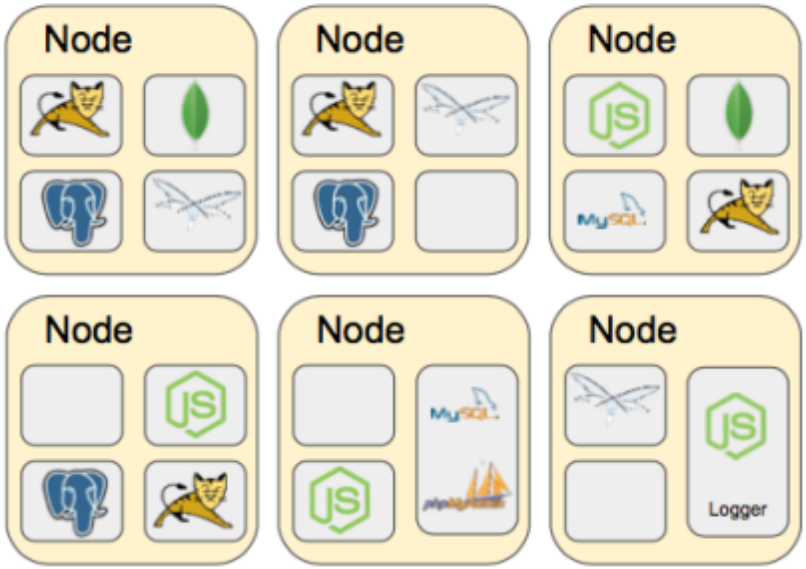
Kubernetes

Container Advantages

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



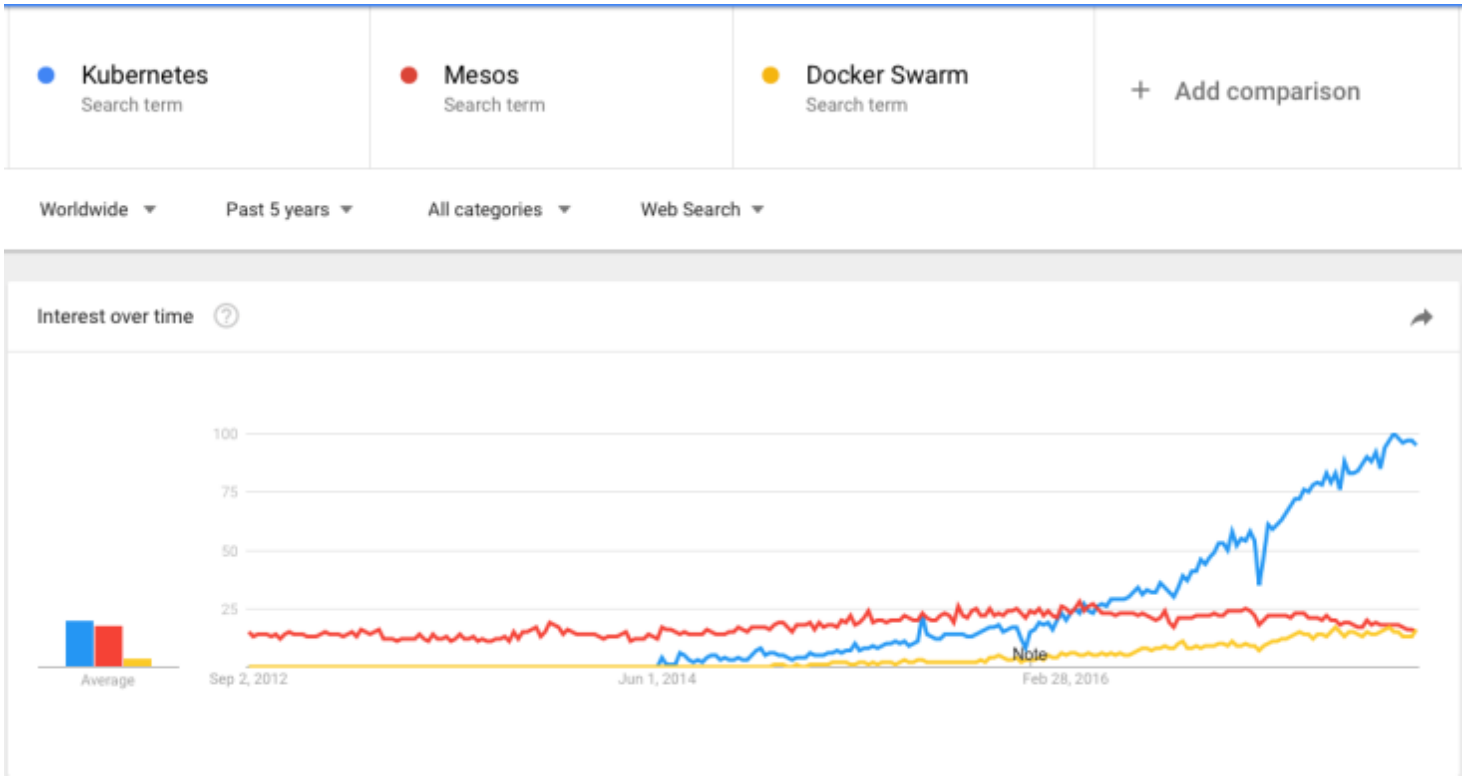
Challenges with multiple containers



- 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?

Container Orchestration Tools

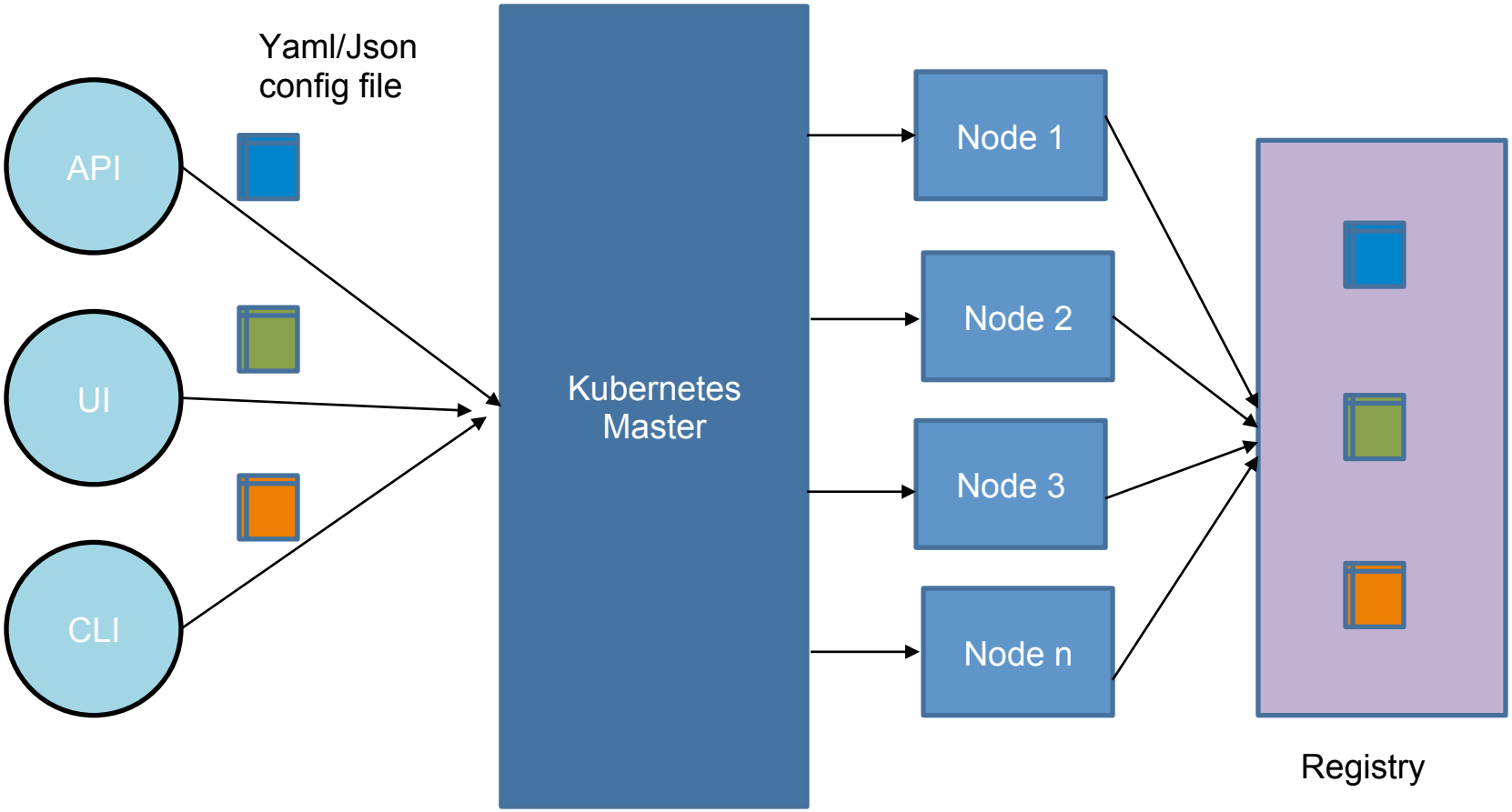
- The three most popular are: Kubernetes, Docker Swam & Mesos
- Kubernetes has become the unofficial standard



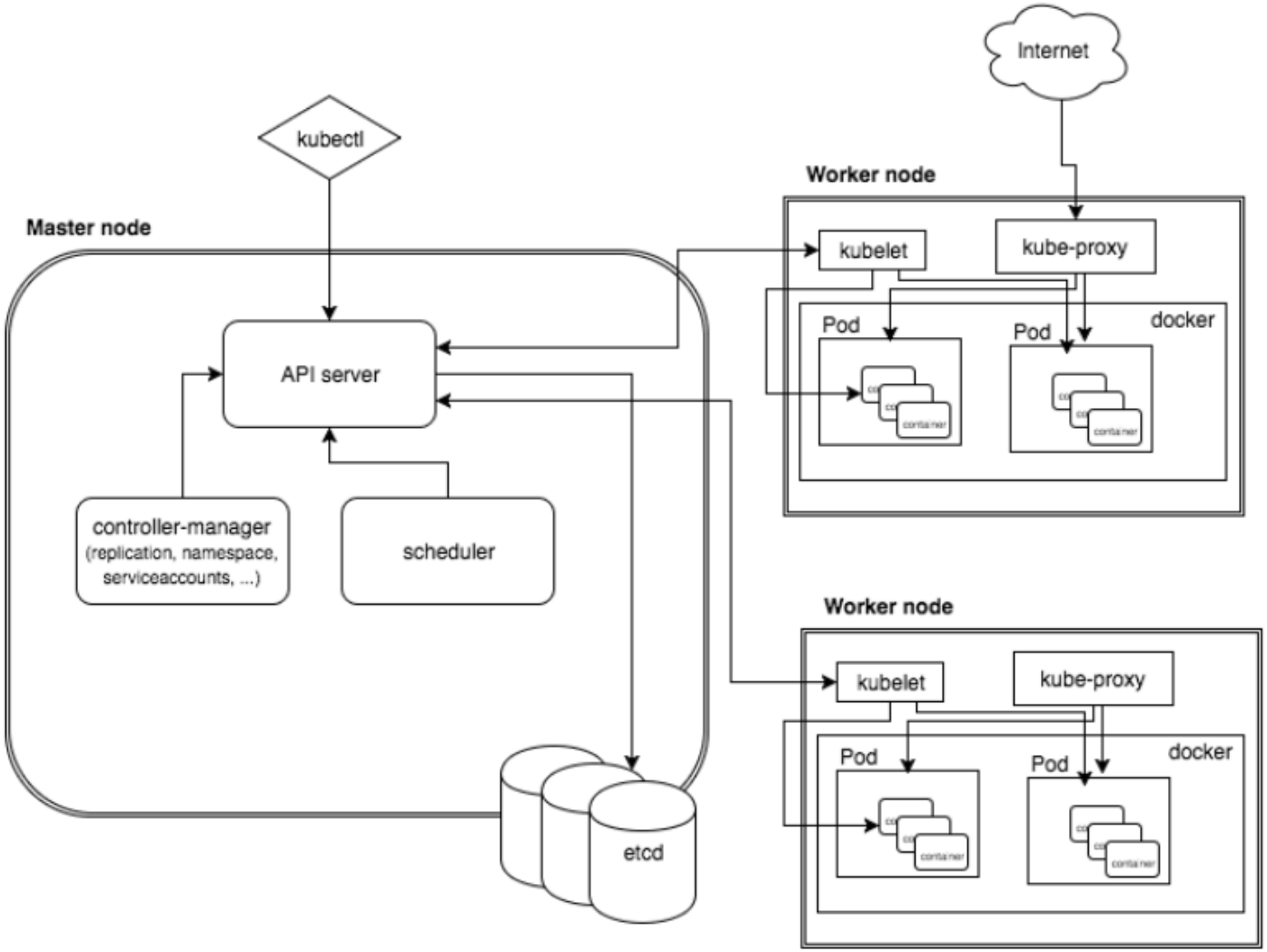
What is Kubernetes?

- Kubernetes is inspired from an internal Google project called Borg
- Open source project managed by the Linux Foundation
- Unified API for deploying web applications, batch jobs, and databases
- Decouples applications from machines through containers
- Declarative approach to deploying applications
- Automates application configuration through service discovery
- Maintains and tracks the global view of the cluster
- APIs for deployment workflows
  - –Rolling updates, canary deploys, and blue-green deployments

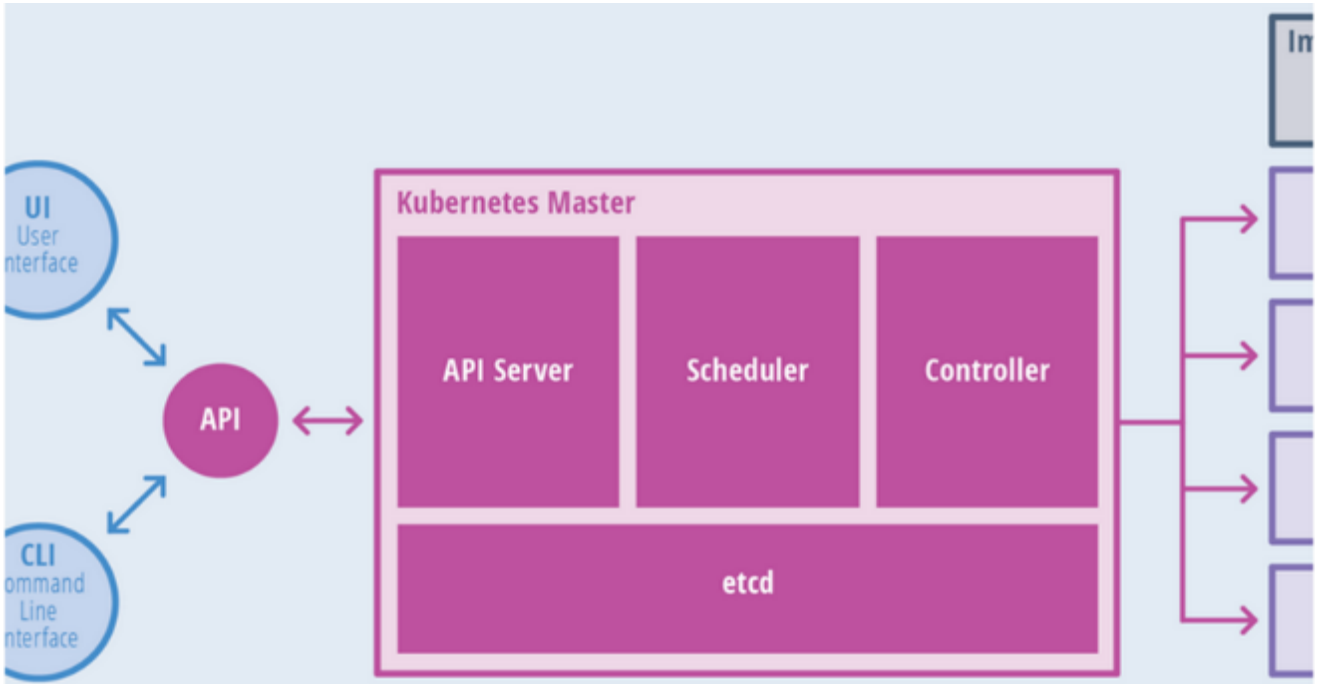
Kubernetes Architecture



K8s Architecture - detailed



Kubernetes Master



Components of master:

- API Server
- Scheduler
- Controller
- etcd

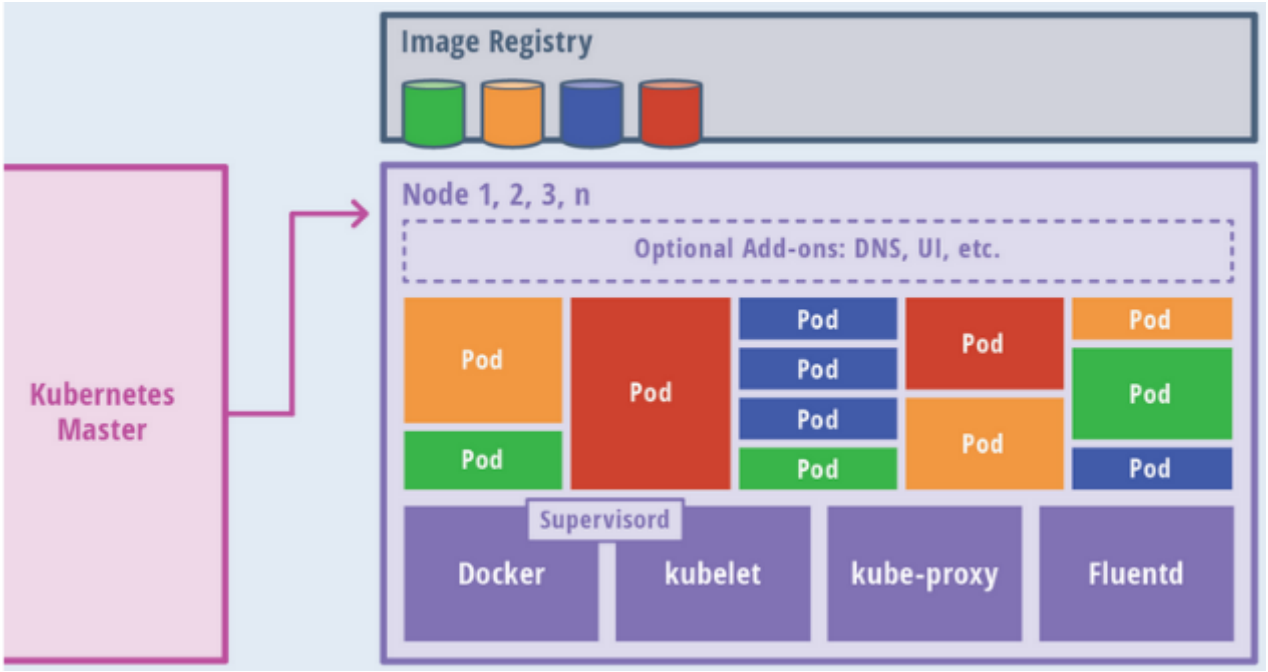
Kubernetes Master

- The **API server** is the entry points for all the REST commands used to control the cluster. It processes REST requests, validates them, and executes the bound business logic. The result state has to be persisted in the “etcd” component.
- **Etcd** is an open source, distributed key-value database; it acts as a single source of truth (SSOT) for all components of the Kubernetes cluster. Masters query etcd to retrieve various parameters of the state of the nodes, pods and containers. Etcd is considered a metadata service in Kubernetes.

Kubernetes Master

- **Controller Manager** is responsible for most of the collectors that regulate the state of the cluster. In general, a controller can be considered a daemon that runs in nonterminating loop and is responsible for collecting and sending information to the API server. It works toward getting the shared state of cluster and then making changes to bring the current status of the server to the desired state. The key controllers are **replication controller**, **endpoint controller**, **namespace controller**, and **service account controller**. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.
- **Scheduler** is one of the key components of Kubernetes master. It is responsible for distributing the workload, tracking resource utilization on cluster nodes and selecting the nodes for the workloads to run. In other words, this is the mechanism responsible for allocating pods to available nodes.

Kubernetes Nodes



Components of a node:

- kubelet
- Kube-proxy
- Docker
- Fluentd

Kubernetes Node - Kubelet

- **Kubelet** Service on each node is responsible for relaying information to and from the control plane service. It interacts with etcd store to read configuration details and to write values **via api-server**. It communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

Kubernetes Node - Kubeproxy

- Kubernetes **Proxy** Service is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.

Kubernetes Installed in AWS

Kubernetes installed in the machine (Just 1 master and is configured to use master to deploy pods)

35.182.246.51

kubectl get nodes --all-namespaces -o wide

kubectl get services --all-namespaces -o wide

kubectl get pods --all-namespaces -o wide

kubectl get deployment --all-namespaces -o wide

Kubernetes: Pods

- The smallest unit that can be scheduled to be deployed through K8s is called a *pod*.
- Homogeneous group of containers.
- This group of containers would share storage, Linux namespaces, cgroups, IP addresses. These are co-located, hence share resources and are always scheduled together.
- Pods are not intended to live long. They are created, destroyed and re-created on demand, based on the state of the server and the service itself.

- Containers in a pod share
  - IP address and port space
  - Filesystem
  - Storage (Volumnes)
  - Labels
  - Secrets

```
root@ip-172-31-0-112:~/code# cat pod.yml
# The example Pod utilizing the profile loaded by the sample daemon.
apiVersion: v1
kind: Pod
metadata:
  name: nginx-apparmor
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80

root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl create -f pod.yml
pod/nginx-apparmor created
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-apparmor           1/1       Running   0          8s
redis-7869f8966-sq8xm    1/1       Running   0          3m
root@ip-172-31-0-112:~/code# _
```

Note: This is just creation of the pod. In order to expose it, a service must be created. In order for it to be respawned, a deployment or replication control must be created.

Kubernetes: Deployment

Abstraction that use Replication controller (Controller manager of kubernetes master) to manage replica-sets of pods

- If object {pod} is used, and it dies, it will not start again
- If object {deployment} is used to start pods, if the pod dies, deployment use replica-set to start the pods again to make sure desired number of pods are always alive
- Try deleting a pod (deployed using deployment) and check if it comes back?

```
root@ip-172-31-0-112:~/code# cat dep.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80

root@ip-172-31-0-112:~/code# kubectl create -f dep.yml
deployment.apps/nginx-deployment created
root@ip-172-31-0-112:~/code# kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            1           5s
redis              1         1         1            1           15m
root@ip-172-31-0-112:~/code# _
```

```
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
nginx-apparmor                      1/1       Running   0          13m
nginx-deployment-67594d6bf6-5p4w2   1/1       Running   0          1m
nginx-deployment-67594d6bf6-fdqx2   1/1       Running   0          1m
nginx-deployment-67594d6bf6-sx62p   1/1       Running   0          1m
redis-7869f8966-sq8xm               1/1       Running   0          16m
root@ip-172-31-0-112:~/code# _
```

Kubernetes: Services

Abstraction regarding a set of pods which enables load balancing, traffic exposure, load balancing and service discovery.

- Abstraction that allows pods to die and replicate in Kubernetes without affecting your application.
- Enable loose coupling between different Pods.
- Provides a stable virtual IP and port
- Services allow Pods to receive traffic.
  - Each Pod has a unique IP but those IP addresses are not exposed outside the Pod without a service.

Kubernetes: Types of Services

- **Cluster IP**:- Expose the service on a cluster-internal IP. Using this makes the service only reachable from within the cluster.
- **NodePort** :- Expose the service on each Node's IP at a static port. You will be able to contact the NodePort Service, from outside the cluster by <NodeIP>:<NodePort>
- **LoadBalancer** :- Expose the service externally using load balancer

```
[root@ip-172-31-0-112:~/code# cat userservice.yml
apiVersion: v1
kind: Service
metadata:
  name: userservice
spec:
  type: NodePort
  ports:
  - name: "user-service"
    port: 8080
    targetPort: 8080
    nodePort: 31001
  selector:
    app: userservice
```

Using mount host path

https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/k8s_yaml/5apicomposerservice-deployment.yml

Kubernetes: Persistent Volumes

- PV: Storage in the cluster that has been provisioned by an administrator (static or dynamic) and is independent of any individual pod that uses the PV.

Following volumes are supported by Kubernetes:

GCEPersistentDisk ; AWSElasticBlockStore

AzureFile ; AzureDisk

FC (Fibre Channel) ; FlexVolume

Flocker ; NFS

iSCSI ; RBD (Ceph Block Device)

CephFS ; Cinder (OpenStack block storage)

Glusterfs ; VsphereVolume

Portworx Volumes ; ScaleIO Volumes

StorageOS

Kubernetes: Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Kubernetes: Persistent Volume Claims

- PVC: Request for a storage by a user, it is similar to pod and it can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/k8s_yaml/userservice-pc.yml

https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/k8s_yaml/3userservice-deployment.yml

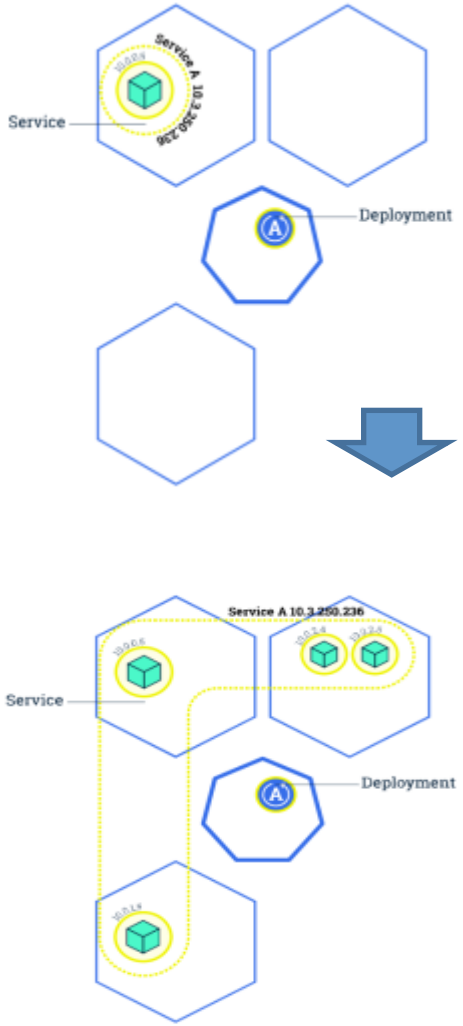
CronJob Example

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

Kubernetes: Scaling

Changing the number of resources to meet a desired state

- Accomplished by adjusting the number of replicas in a deployment
- Accommodates both scaling up and scaling down to a minimum of 0
- Traffic is automatically sent to newly created instances through the service load balancer
- Can be used to enable rolling updates without downtime

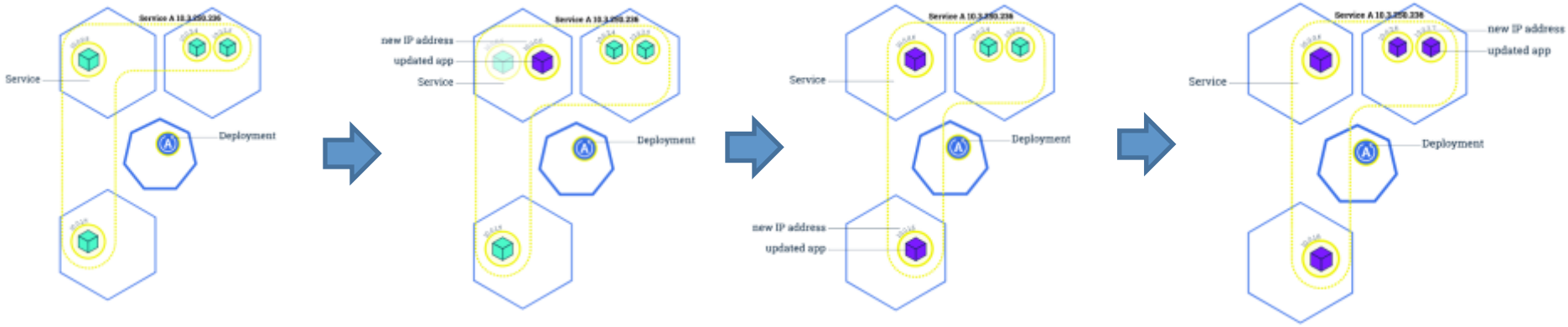


```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              1         1         1            1           8s
nginx-deployment   3         3         3            3           17m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl scale deployments/nginx --replicas=3
deployment.extensions/nginx scaled
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              3         3         3            3           34s
nginx-deployment   3         3         3            3           18m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
nginx-6f858d4d45-b4cv2              1/1       Running   0          42s
nginx-6f858d4d45-g42mg              1/1       Running   0          16s
nginx-6f858d4d45-l69jv              1/1       Running   0          16s
nginx-apparmor                      1/1       Running   0          30m
nginx-deployment-67594d6bf6-7v5lk   1/1       Running   0          12m
nginx-deployment-67594d6bf6-fdqx2   1/1       Running   0          18m
nginx-deployment-67594d6bf6-sx62p   1/1       Running   0          18m
redis-7869f8966-sq8xm               1/1       Running   0          33m
```

Kubernetes: Rolling Update

Updates the pods in a serial manner will load balancing traffic to available instances

- If deployment is exposed publicly, the service will load-balance traffic to only active and available pods
- Used to enable zero downtime updates
- Allows greater application update frequency
- Can also be leveraged to rollback to previous versions



```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx:1.12.1 --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code# kubectl set image deployments/nginx nginx=nginx:latest
deployment.extensions/nginx image updated
root@ip-172-31-0-112:~/code# kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-6c486b77db (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  43s   deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  ScalingReplicaSet  11s   deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  10s   deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
```

```
root@ip-172-31-0-112:~/code# kubectl rollout undo deployments/nginx
deployment.extensions/nginx
root@ip-172-31-0-112:~/code# kubectl describe deployments/nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=3
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:1.12.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-7f9bc86464 (1/1 replicas created)
Events:
  Type    Reason             Age              From                   Message
  ----    ------             ----             ----                   -------
  Normal  ScalingReplicaSet  1m               deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  1m               deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
  Normal  ScalingReplicaSet  18s (x2 over 2m) deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  DeploymentRollback 18s              deployment-controller  Rolled back deployment "nginx" to revision 1
  Normal  ScalingReplicaSet  17s              deployment-controller  Scaled down replica set nginx-6c486b77db to 0
```

Note: Rollbacks can also be also enacted with zero-downtime
kubectl rollout status deployment nginx
kubectl rollout history deployment nginx
kubectl rollout history deployment/nginx --revision=3
kubectl rollout undo deployment/nginx --to-revision <num>

Java Example

- Install Kubernetes on bare metal

https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/kubernetes_setup.md

- Java application – Monolithic with Docker

https://github.com/shekhar2010us/microservices_monolithic_docker

- Java application – Microservices with Kubernetes

https://github.com/shekhar2010us/microservices_kubernetes_docker

eval(this.class);

- Please fill out our online course evaluation, which gives you immediate access to your certificate of completion.
- All responses are confidential even though an email address is requested.

1. Go to: **www.metricsthatmatter.com/ASPE**
2. From the drop down menu choose **Docker Containerization Boot Camp, site, instructor name, class start date**
3. Enter email address, answer the questions, provide desired written feedback, and click submit.

THANK YOU !!!