Docker Containerization Boot Camp

Classroom Manual

63000M_7.0_2017

Other courses in ASPE's Techtown Training technology innovation curriculum

We offer learning solutions for some of the most critical skills needs in the enterprise technology ecosystem. We can help your organization achieve great things with innovative skills for software delivery, IT operations, security and data engineering.

For more information on our classes, visit:

www.techtowntraining.com

DEVOPS TRANSFORMATION

- Introduction to DevOps
- DevOps Implementation Boot Camp
- Agile Boot Camp

LEAN LEADERSHIP

- DevOps for Executives
- Lean IT and Technology Management

CONFIGURATION MANAGEMENT

- Ansible Configuration Management Workshop
- Chef Configuration Management Workshop
- Puppet Configuration Management Workshop

CONTAINERIZATION

- Docker Containerization Boot Camp
- Kubernetes Container Cluster Management

DATA ENGINEERING & ANALYTICS

- Data Analysis Boot Camp
- Data Life Cycle Management
- Big Data Boot Camp (Hands-On Hadoop Administration)
- Introduction to R
- Predictive Data Science: Foundations Boot Camp

SOFTWARE ENGINEERING

- Continuous Integration Workshop
- Continuous Delivery Workshop
- Test Automation Boot Camp
- Test Driven Development
- Agile Driven DevOps
- Hands-On Agile Engineering

SECURITY

- Practical Information Security Boot Camp
- Fundamentals of Secure Software Engineering
- CISM Certification Exam Boot Camp

Seasoned trainers guide you through today's real-world technology tools and send you back to work with immediately useful skills.

There's nothing like having unfettered, in-person access to an expert who can teach you new skills and answer your real-world questions. When you choose Techtown Training for your training needs, you find yourself in the same room with some of the best technology pros in the business. There's nothing academic about our training. Every class we teach is designed to equip you with immediately useful skills you can take back to work and begin using right away.

The value of classroom training from Techtown Training includes:

- In-person access to an engaging real-world expert who answers your questions in person
- The ability to tie subject matter to your own situation, so you can go back to work ready to use it
- Labs, exercises and demonstrations that give you actual practice with what you learn
- Our 100% guarantee that you will get what you need from our training

At Techtown Training, it's our experts that set us apart. These subject matter experts are dedicated to transferring their skills to you in the classroom so you can begin to leverage today's technologies, tools and processes in your own workplace. Our promise to you is to provide seasoned consultants with real-world experience who walk you through the most complex technology problems in a way that transforms them into sources of competitive advantage.

We offer our hands-on training over the internet, delivered by a live expert.

You can attend many of our industry-leading training courses from anywhere in the world. We utilize Zoom Video Conferencing to connect you with one of our expert instructors in an online environment to give you the same experience you would get in the classroom.

Live Online Training from Techtown Training provides everything a traditional classroom would.

- **Live expert instruction:**
  The instructor is live and can answer any of your questions in real time. You can communicate with the instructor both electronically and verbally in our live online training, interacting just as you would in the classroom.

- **Courseware:**
  Our live online training courses use the same courseware as our classroom courses. Prior to the course date, you will receive all the materials each student would get in a normal class.

- **Labs:**
  The critical component...we conduct hands-on group activities in breakout sessions utilizing the power of the Zoom platform. You'll get the same benefits of group exercises that you would in a classroom setting.

- **Peer Interaction:**
  Even in this live online training environment you will gain valuable knowledge interacting with your peers, in class, in our breakout sessions, and during course lectures.

Our experts can deliver private onsite training to your teams on any Techtown Training topic. Work through issues in private, and use your own environments for labs and case studies.

An onsite training option brings our experts to you — on your schedule, at your location. It also allows us to plan your training in advance and tailor classes according to your needs. You get your particular issues solved, with learning targeted to your unique team environment. The entire team benefits immediately, and the cost per student is less than any other option on the market.

How onsite training works:

Every Techtown Training course is custom tailored to address the unique complexities you face. You start by talking through your top challenges with our training team. We then bring in a Techtown Training subject matter expert matching your needs, and together we set training objectives.

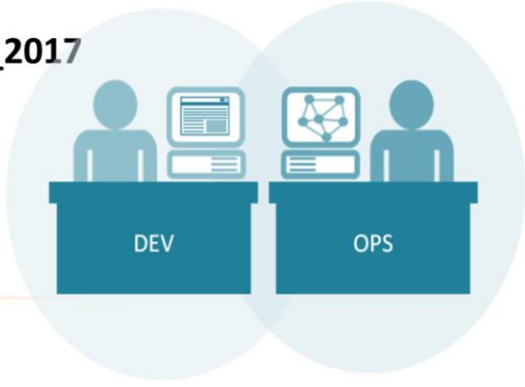
1. Choose your topic or course
2. Have a free consulting call with a Techtown Training advisor
3. Receive a simple, all-inclusive price quote and itinerary
4. Work with the Subject Matter Expert teaching your class to tailor the content to your needs
5. Select dates, times and location that work best for you and your team

Key benefits of onsite training:

- **Custom Learning:** Eliminate uncertainty by working directly with our Subject Matter Expert to tailor your onsite training to your specific needs. Pay only for the skills, tools and techniques your team needs to succeed.
- **Flexibility:** You decide when and where your training is delivered. Choose between half days, full days, mornings, nights, weekends or even have your team attend remotely. Your needs are the priority.
- **Cost Savings:** Onsite training is designed specifically for group training. Because onsite training typically happens at your location, there is no facility cost incurred. You also save by not having travel or lodging costs associated with your employees traveling for training.
- **Confidentiality:** Private, onsite training allows sensitive and confidential information to be discussed freely, since only your team is in attendance. NDAs for our instructors are routine, so the instructor is yours. Ask the tough questions you need to solve your specific issues.

Docker Container Implementation

63000M-7.0_2017



TECHTOWN training

NOTES:

NOTES:

Who is your instructor?

A little about me…

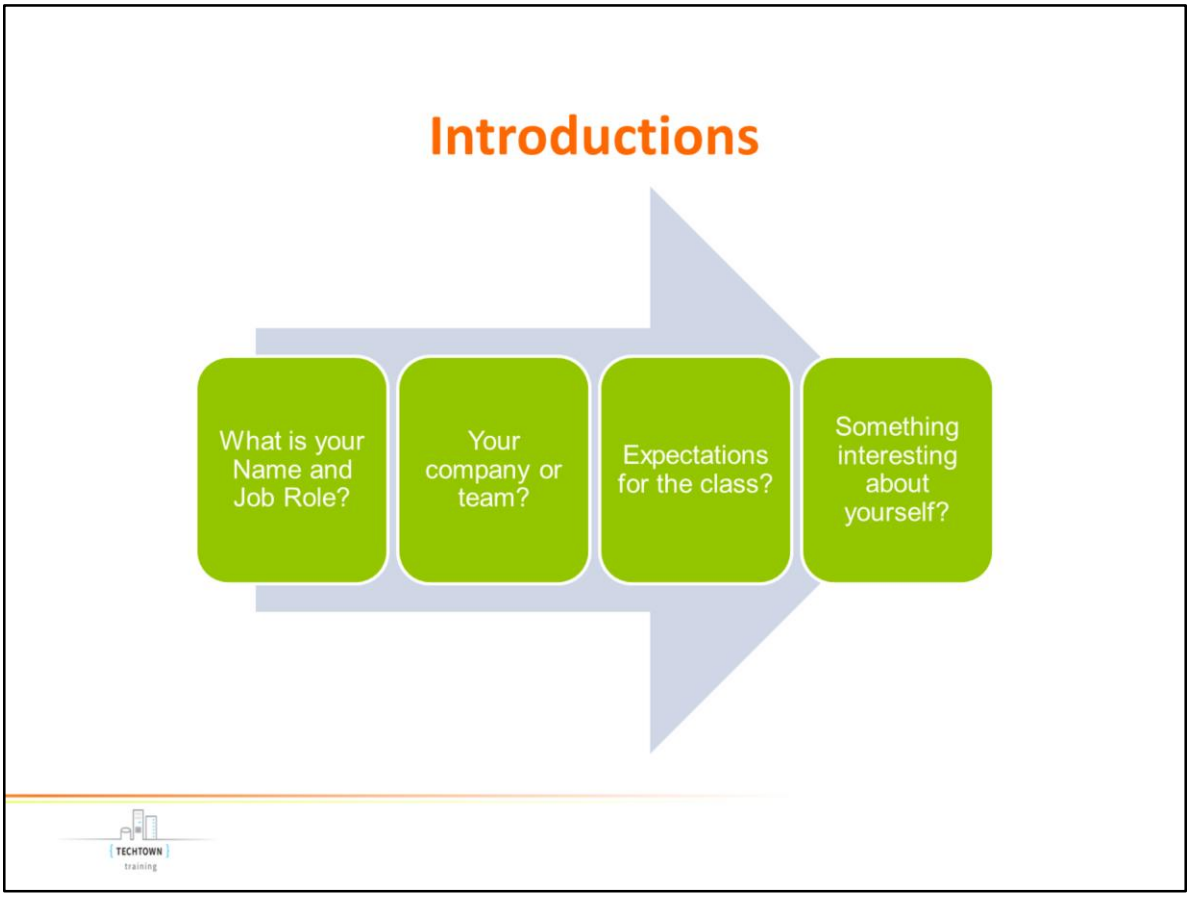
Instructor Information

Who is your instructor?

A little about me…

TECHTOWN
training

NOTES:



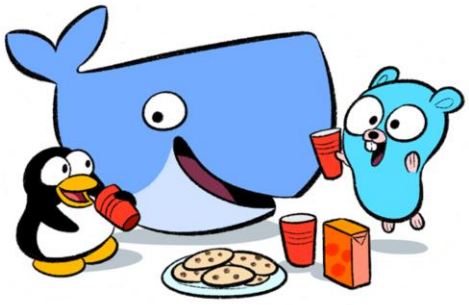
NOTES:

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.

TECHTOWN training

NOTES:



Introduction to Docker

Part 1: DevOps Docker

TECHTOWN training

NOTES:

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

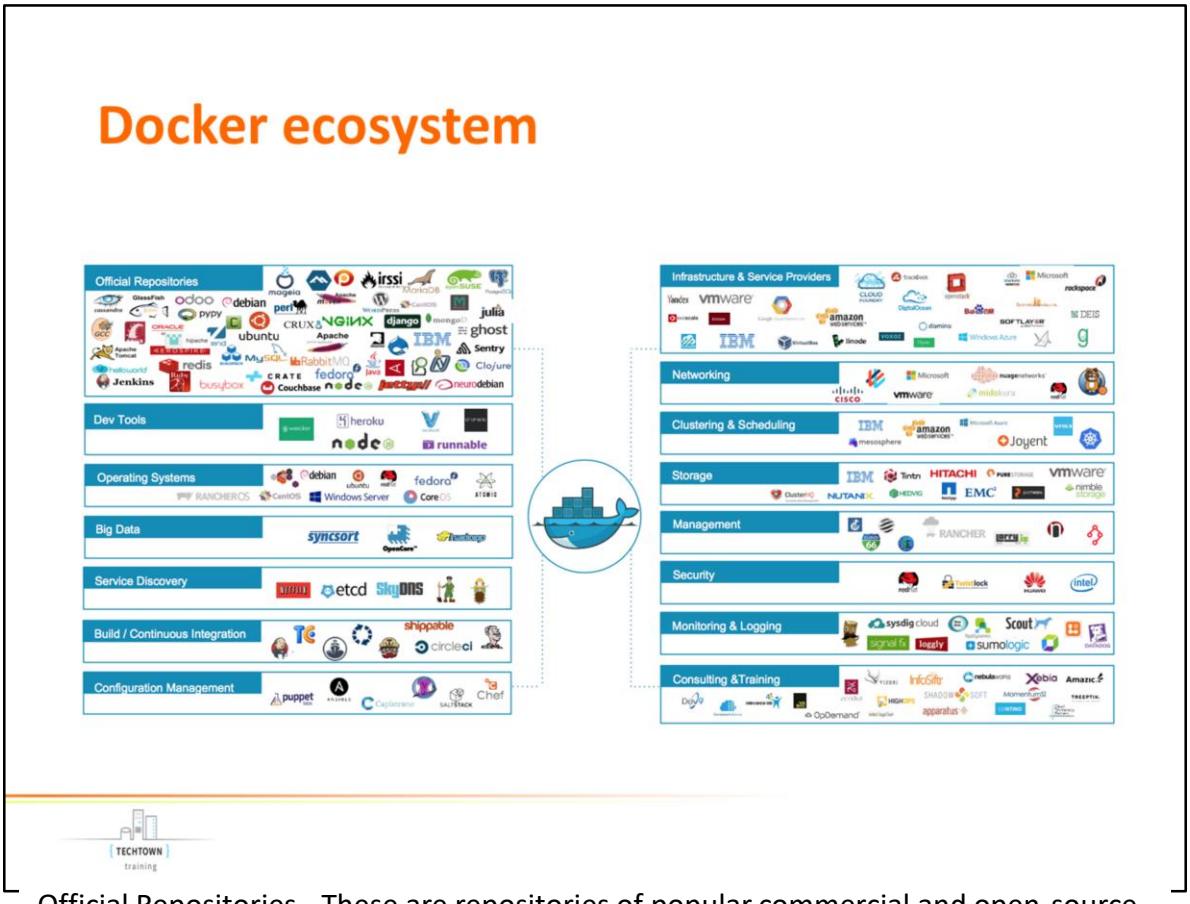
Containers as a Service provider

- Integrated platform for dev and IT
- Commercial technical support

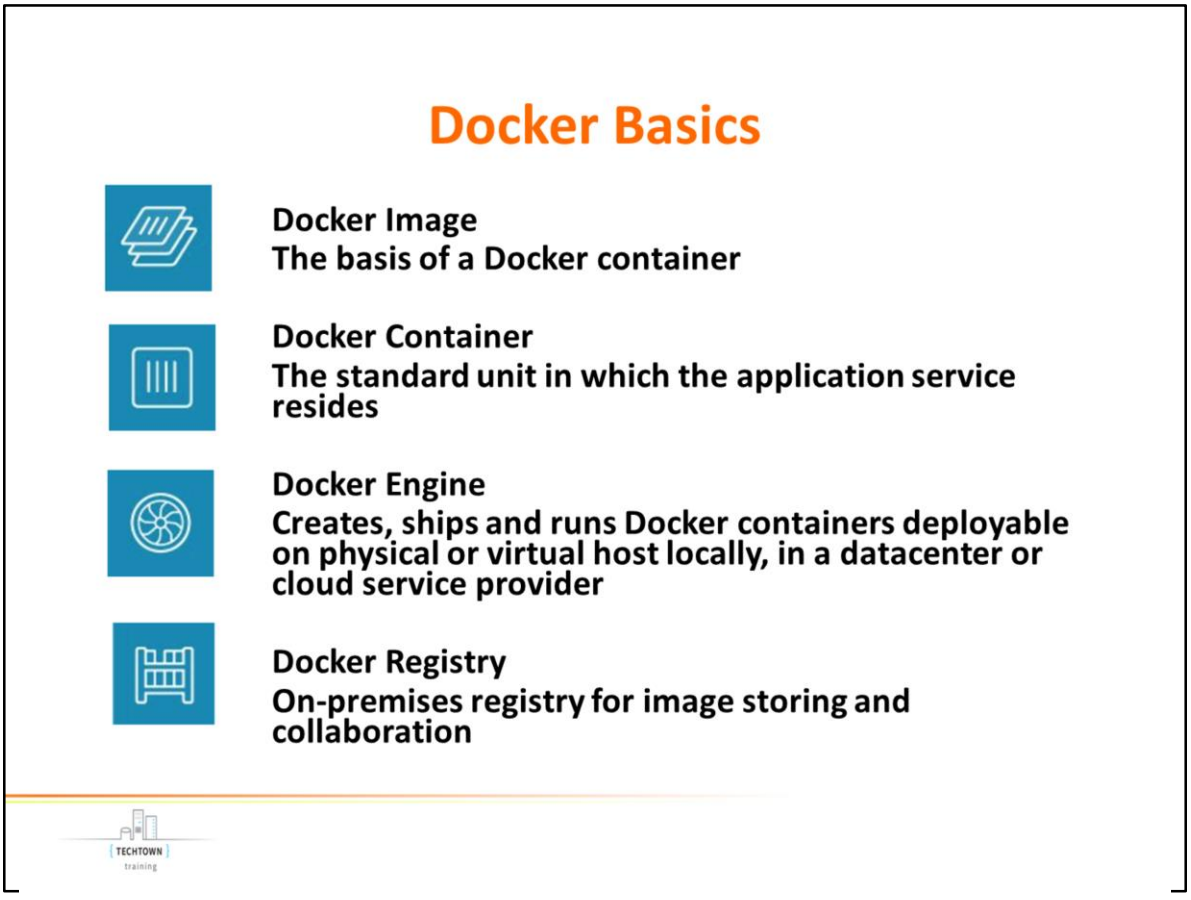
Docker project sponsor

- Primary sponsor of Docker project
- Supports project maintainers

NOTES:

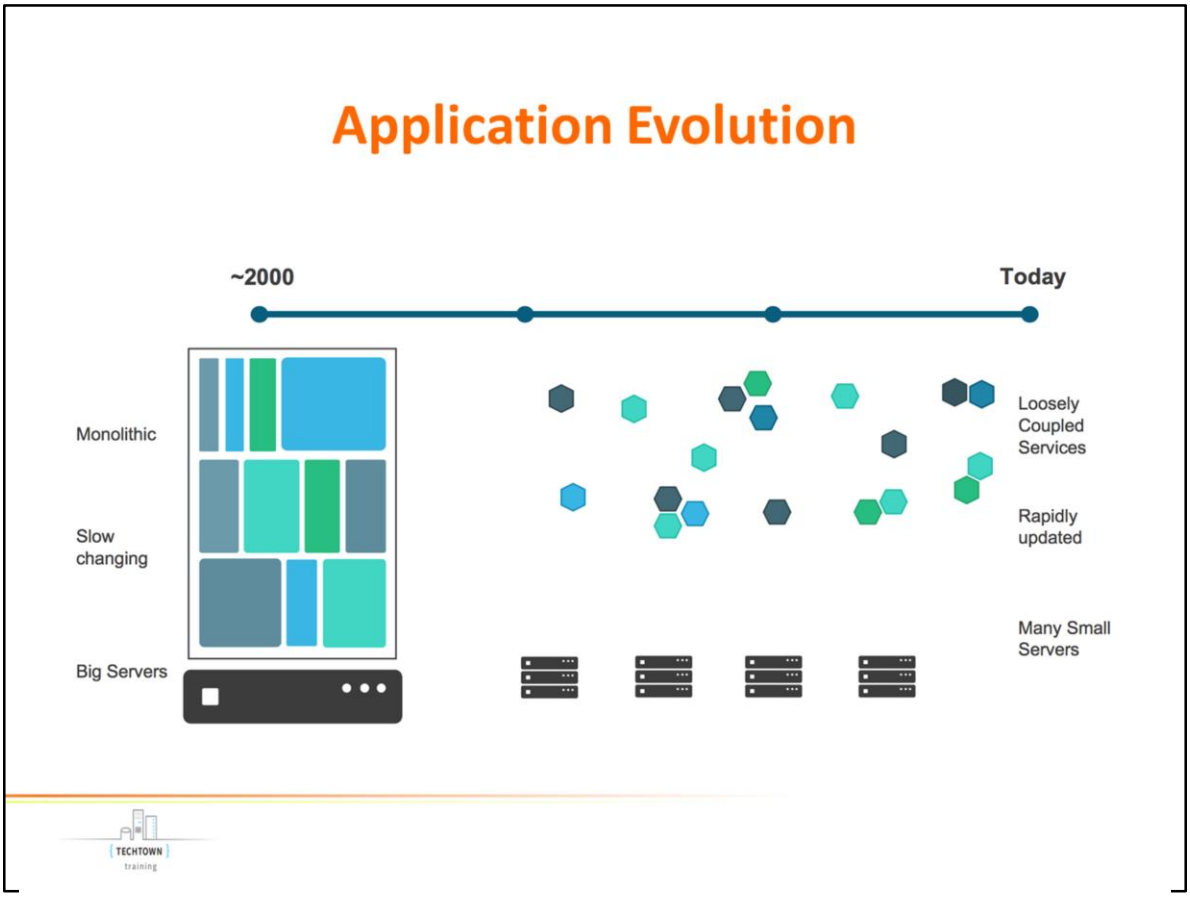


Official Repositories - These are repositories of popular commercial and open-source software and systems that are maintained by their “Official” providers. These undergo special vetting, static analysis for malware, and are usually among the best-documented container images.



A Docker image is a read-only template with instructions for creating a Docker container. For example, an image might contain an Ubuntu operating system with Apache web server and your web application installed. You can build or update images from scratch or download and use images created by others. An image may be based on, or may extend, one or more other images. A docker image is described in text file called a Dockerfile, which has a simple, well-defined syntax.

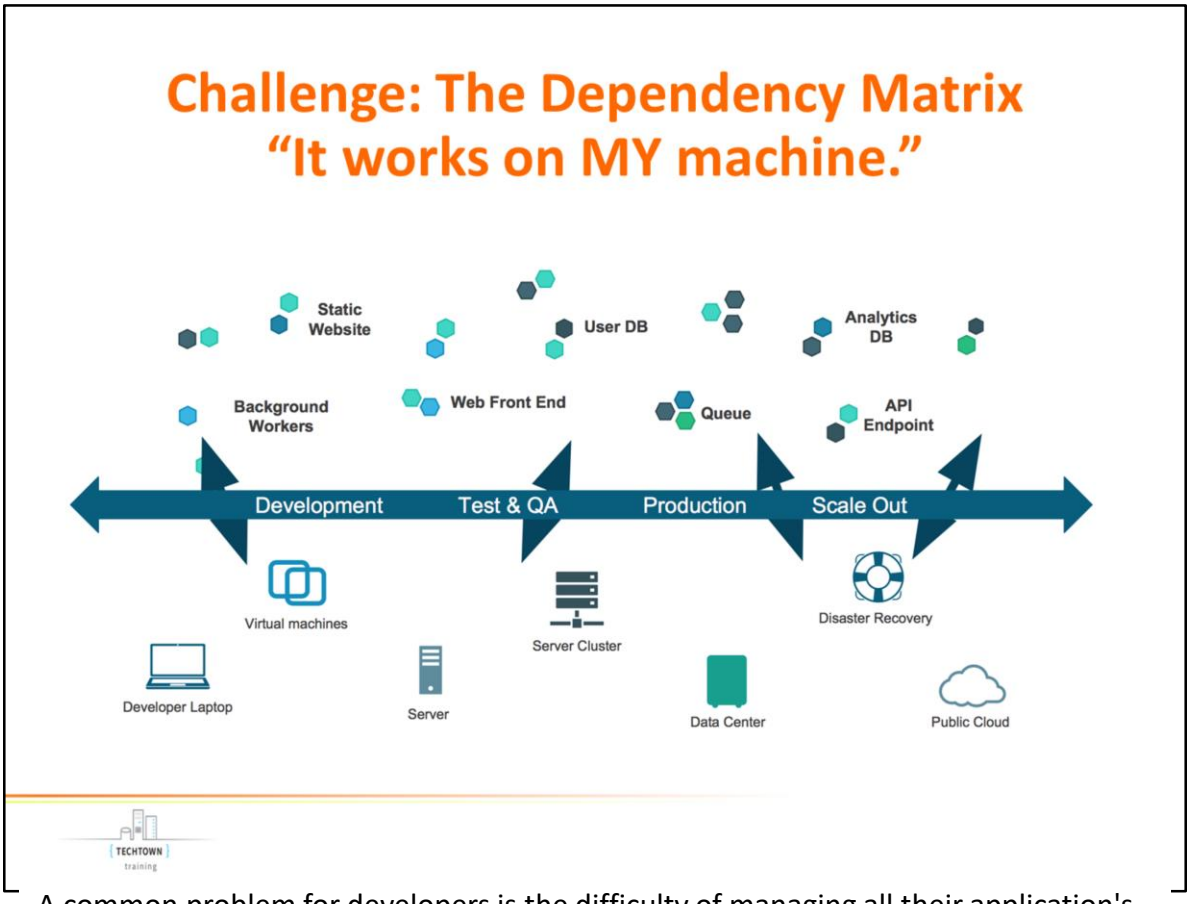
A Docker container is a runnable instance of a Docker image. You can run, start, stop, move, or delete a container using Docker API or CLI commands. When you run a container, you can provide configuration metadata such as networking information or environment variables. Each container is an isolated and secure application platform, but can be given access to resources running in a different host or container, as well as persistent storage or databases.



Management of software has changed radically in this time too:

We used to have big up-front specifications for “projects” that represented yearly releases, and deadlines were frequently missed.

Now we have Agile development… “projects” are mostly gone and instead we have ever-changing “products” with a much-tighter release schedule… sometimes multiple times per day.



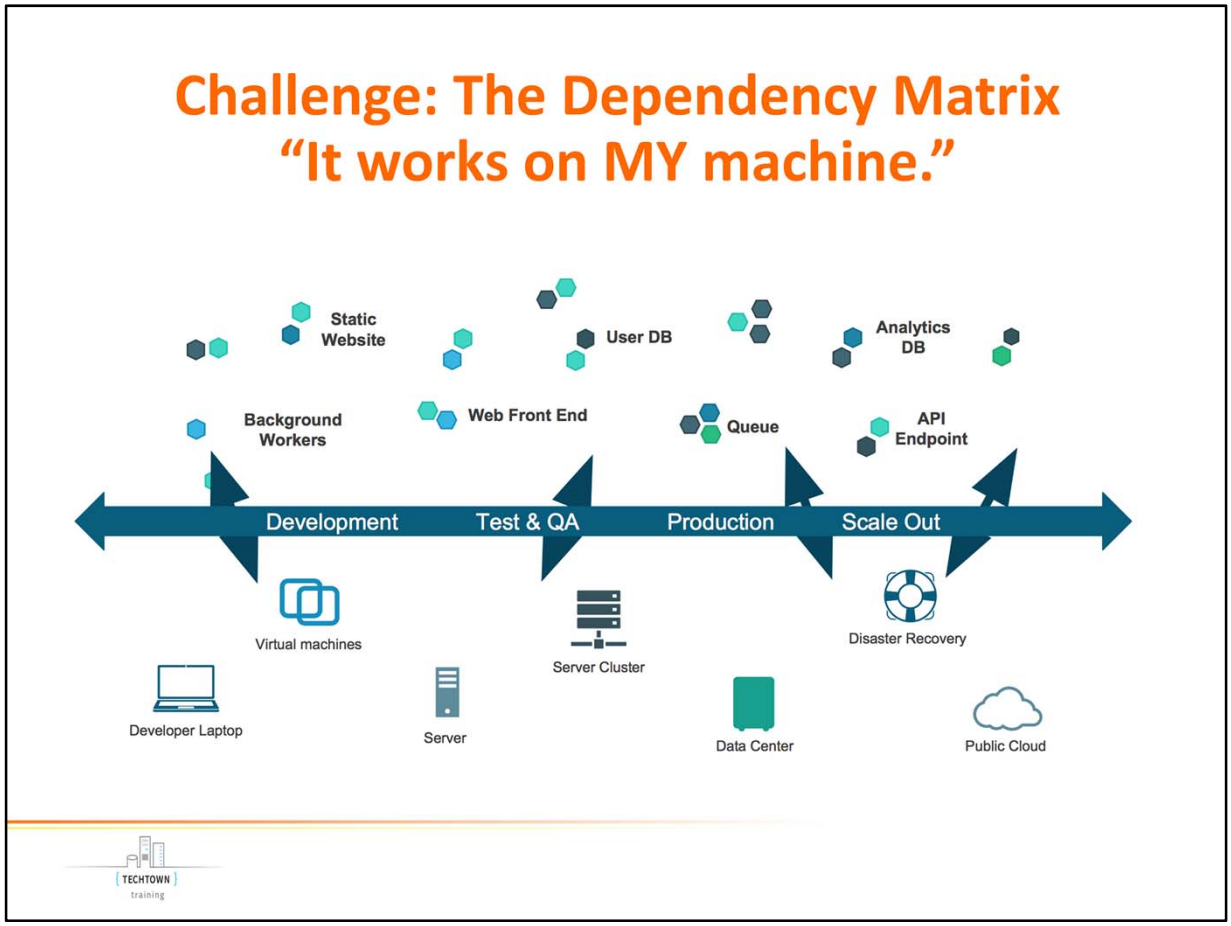
A common problem for developers is the difficulty of managing all their application's dependencies in a simple and automated way.
This is usually difficult for several reasons:

Cross-platform dependencies. Modern applications often depend on a combination of system libraries and binaries, language-specific packages, framework-specific modules, internal components developed for another project, etc. These dependencies live in different "worlds" and require different tools - these tools typically don't work well with each other, requiring awkward custom integrations.

Conflicting dependencies. Different applications may depend on different versions of the same dependency. Packaging tools handle these situations with various degrees of ease - but they all handle them in different and incompatible ways, which again forces the developer to do extra work.

Custom dependencies. A developer may need to prepare a custom version of their application's dependency. Some packaging systems can handle custom versions of a dependency, others can't - and all of them handle it
differently.

(Continued on next page.)



Docker solves the problem of dependency hell by giving the developer a simple way to express all their application's dependencies in one place, while streamlining the process of assembling them. If this makes you think of XKCD 927, don't worry. Docker doesn't replace your favorite packaging systems. It simply orchestrates their use in a simple and repeatable way. How does it do that? With layers.

Docker defines a build as running a sequence of Unix commands, one after the other, in the same container. Build commands modify the contents of the container (usually by installing new files on the filesystem), the next command modifies it some more, etc. Since each build command inherits the result of the previous commands, the order in which the commands are executed expresses dependencies.

Here's a typical Docker build process:

FROM ubuntu:12.04 RUN apt-get update && apt-get install -y python python-pip curl RUN curl -sSL https://github.com/shykes/helloflask/archive/master.tar.gz | tar -xzv RUN cd helloflask-master && pip install -r requirements.txt

Note that Docker doesn't care how dependencies are built - as long as they can be built by running a Unix command in a container.

Reference: https://github.com/docker/docker

Containers as a Solution

Container

Packages up software binaries and dependencies

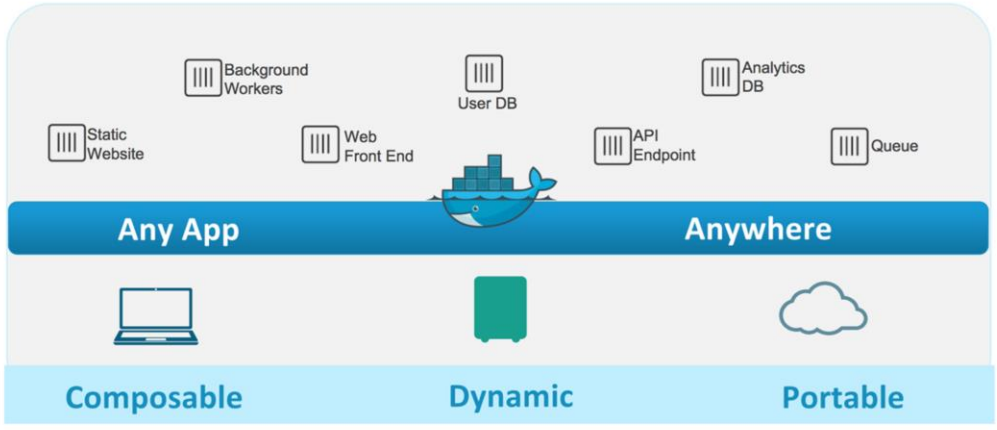
Isolates software from each other

- **Container is a standard format**
- **Easily portable across environment**
- **Allows ecosystem to develop around its standard**

TECHTOWN training

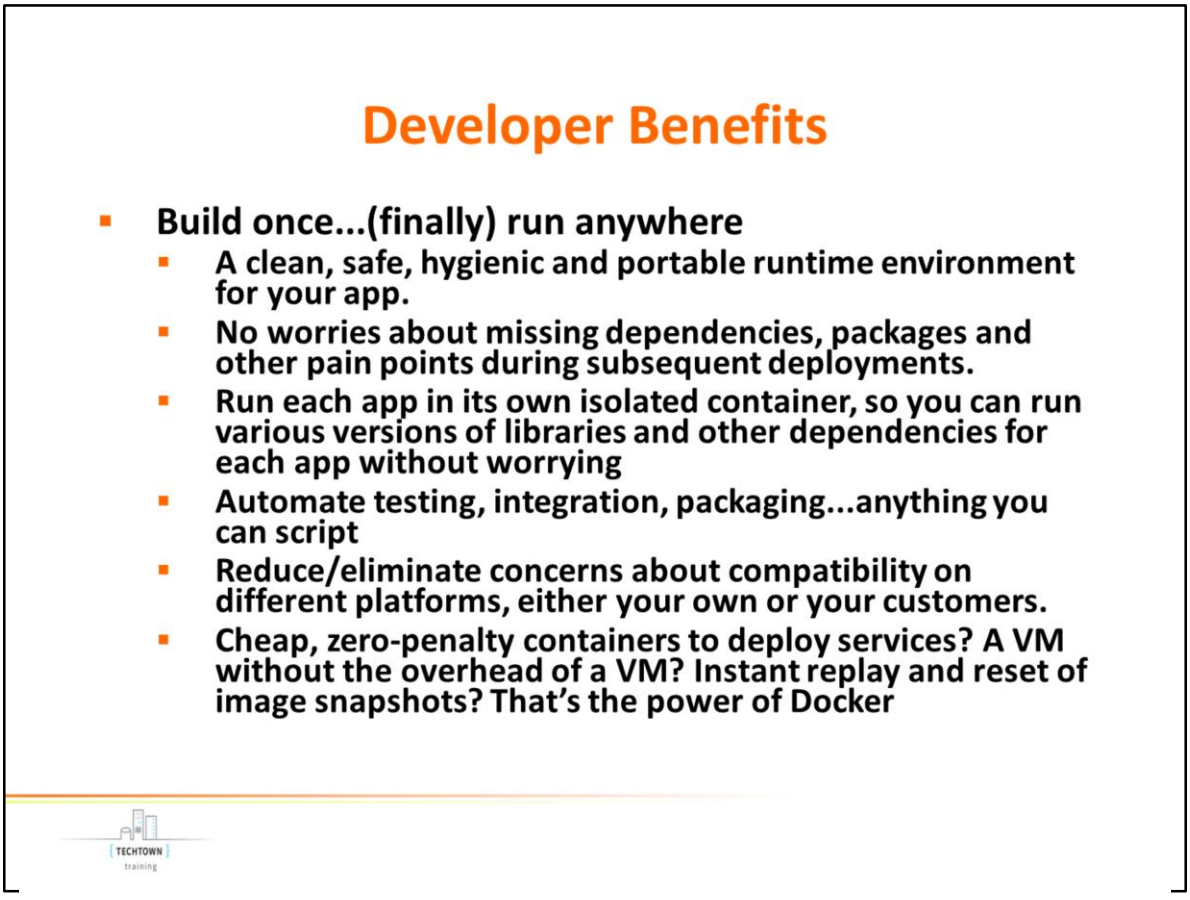
NOTES:

Containers as a Solution



TECHTOWN training

NOTES:



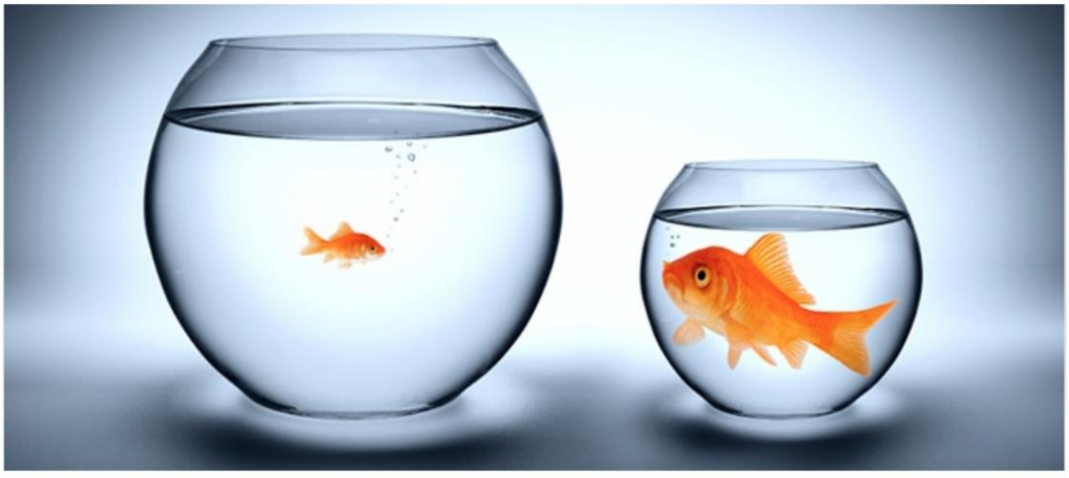
One big driver for the initial adoption of Docker has been startups in an effort to drive down their Cloud Bill. There is lots of anecdotal evidence of startups cutting their cloud bill in half by using Docker to increase utilization. Instead of having a single VM for a database for example, putting a database, mid tier and front end container all in the same VM thus using less resources overall.

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

NOTES:

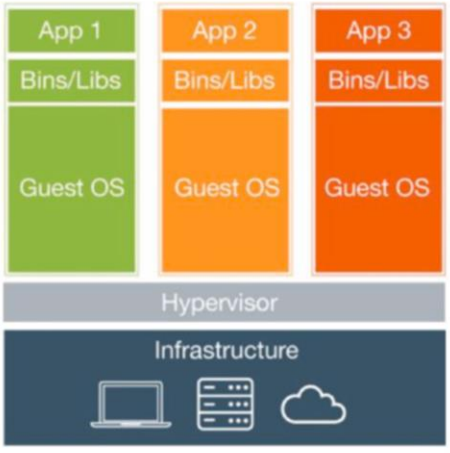
VMs vs. Containers



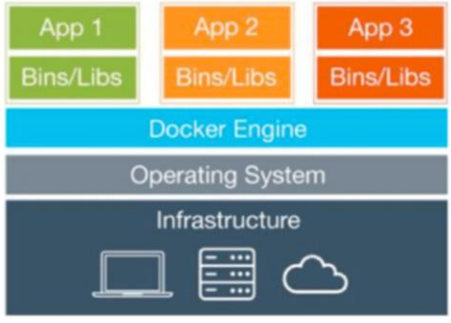
A badly kept secret on cloud is the over-provisioning of VMs for workloads. The average CPU utilization on the average data center VM is between 6-12% and this is part of the reason for a very healthy ecosystem of cloud consultants.
Containers are more lightweight and are typically filled with higher density workloads.

https://gigaom.com/2013/11/30/the-sorry-state-of-server-utilization-and-the-impending-post-hypervisor-era/

VMs vs. Containers



Virtual Machines



Containers

TECHTOWN training

NOTES:

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)

Size: VMs are very large which makes them impractical to store and transfer.

Performance: running VMs consumes significant CPU and memory, which makes them impractical in many scenarios, for example local development of multi-tier applications, and large-scale deployment of cpu and memory-intensive applications on large numbers of machines.

Portability: competing VM environments don't play well with each other. Although conversion tools do exist, they are limited and add even more overhead.

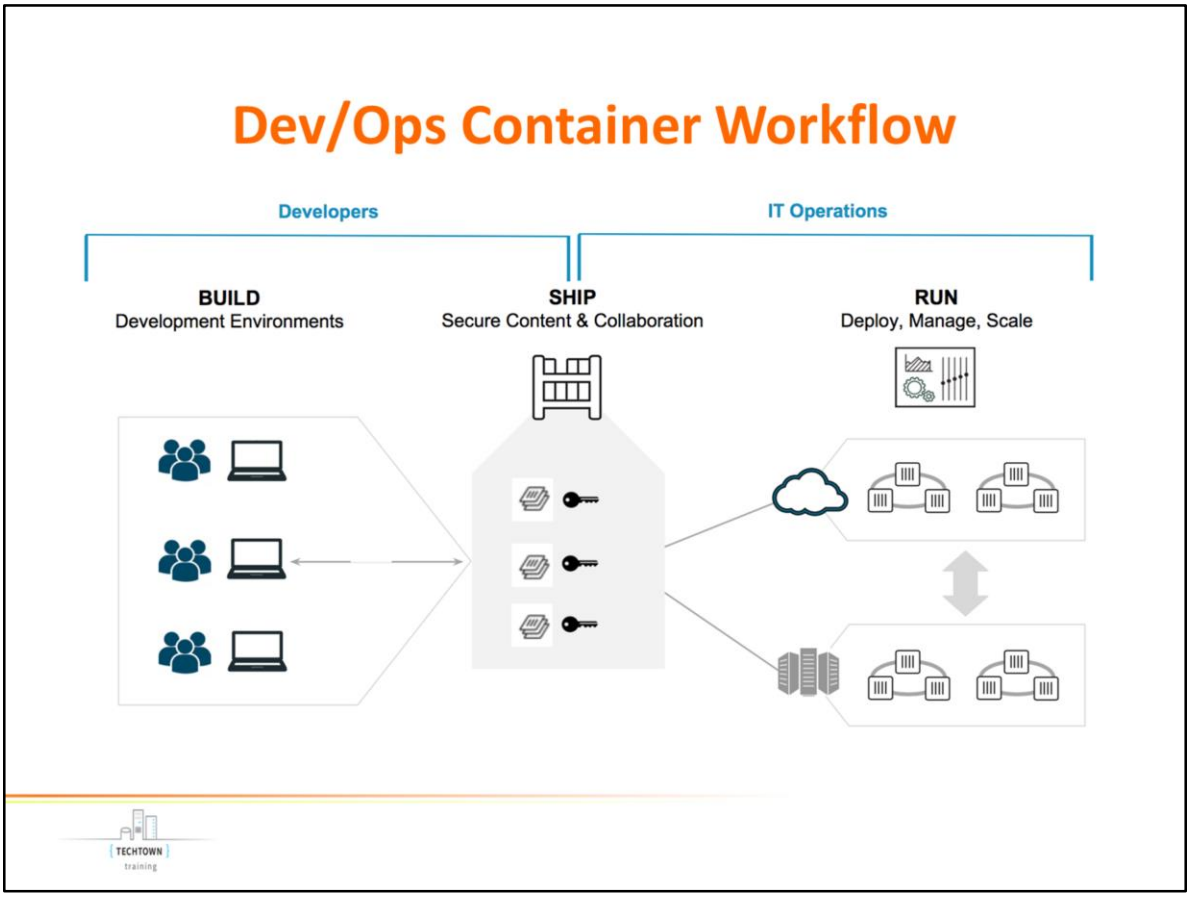
Hardware-centric: VMs were designed with machine operators in mind, not software developers. As a result, they offer very limited tooling for what developers need most: building, testing and running their software. For example, VMs offer no facilities for application versioning, monitoring, configuration, logging or service discovery.

Reference: https://github.com/docker/docker

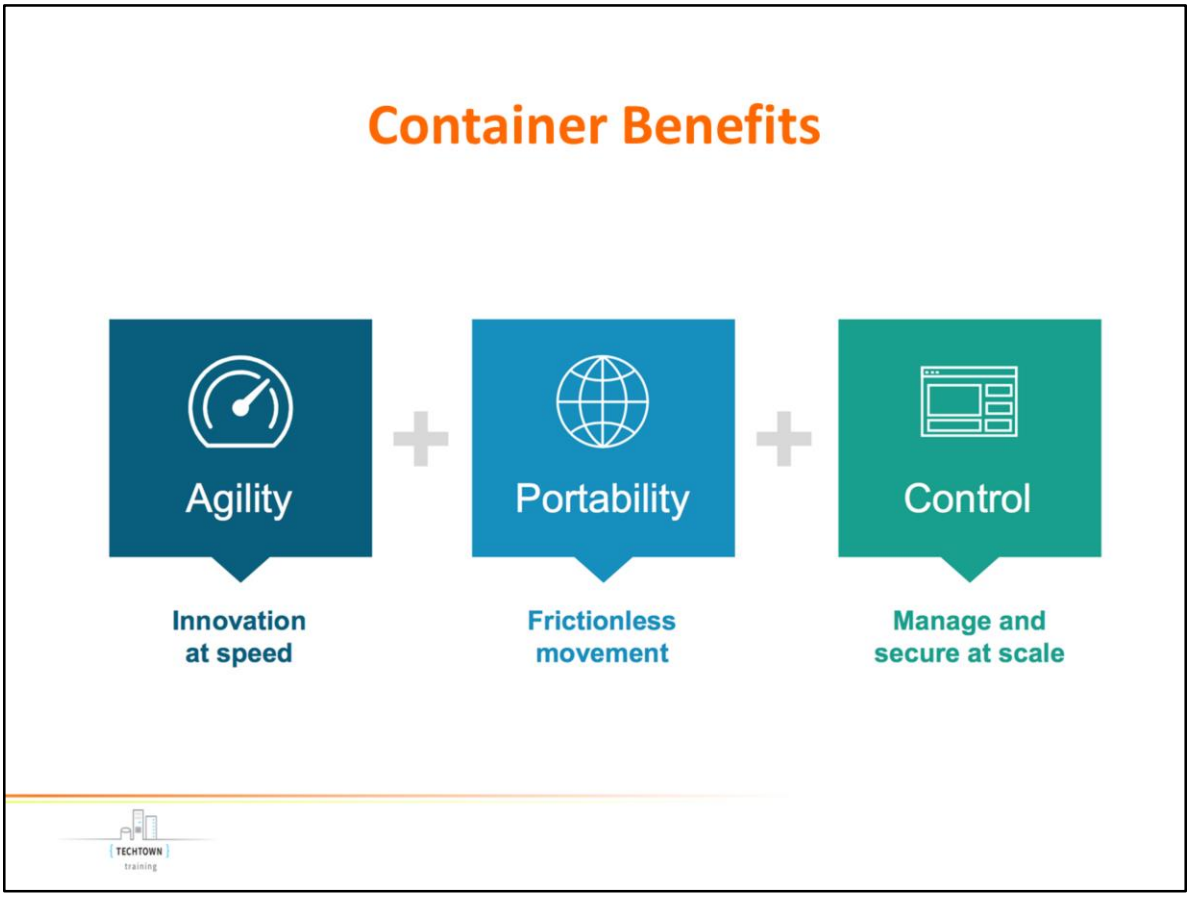
VMs vs. Containers

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

The first three layers (black items) are those that are essentially shared by both VM and Container approaches.
The balance of the list describes elements (mostly cons) specific to true Virtual Machines.



NOTES:



NOTES:

Docker CE vs. Docker EE

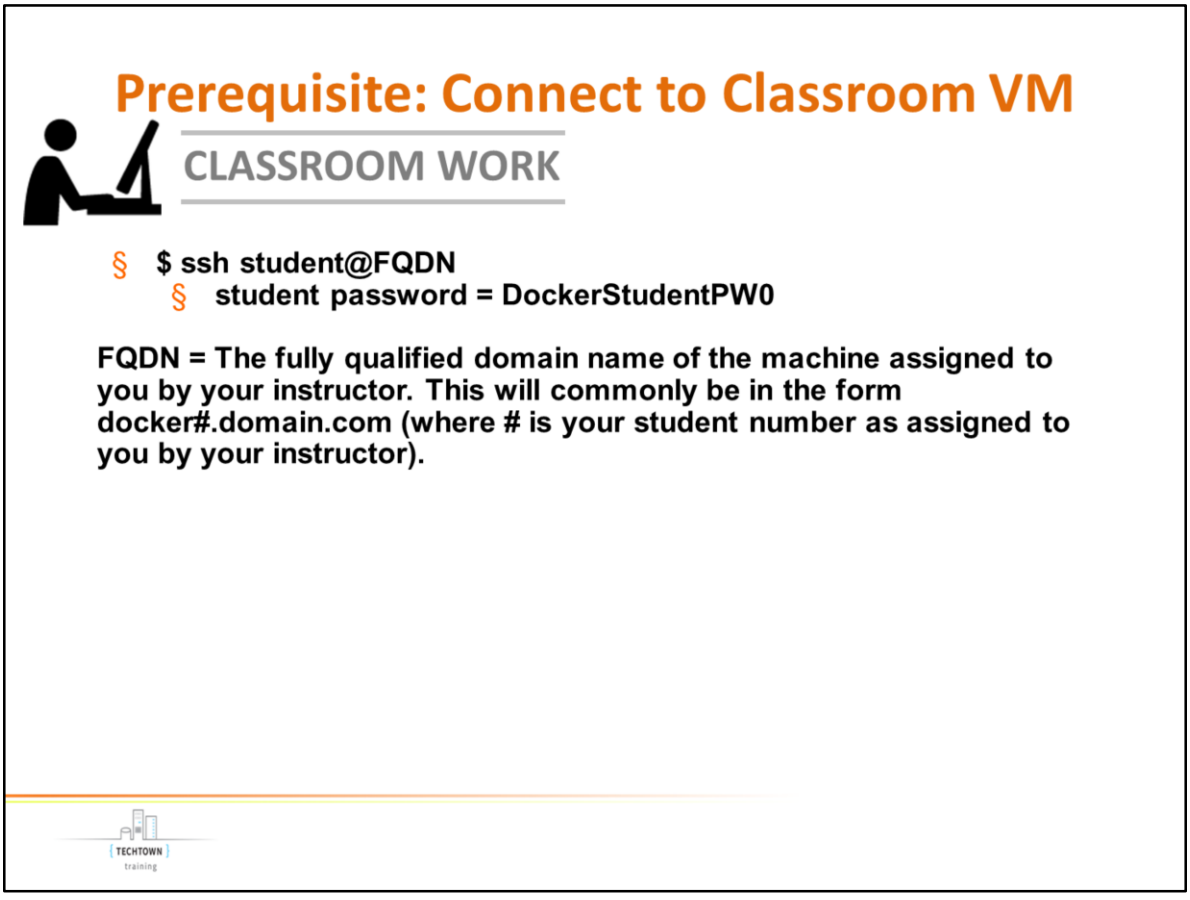
	COMMUNITY EDITION	ENTERPRISE EDITION BASIC	ENTERPRISE EDITION STANDARD	ENTERPRISE EDITION ADVANCED
Container engine and built in orchestration, networking, security	✓	✓	✓	✓
Docker Certified - Infrastructure, Plugins and ISV Containers		✓	✓	✓
Image Management (private registry, caching)	Cloud hosted repos		✓	✓
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

We will be using Community Edition exclusively for our in-class work, but it's valuable to know the differences.

Installing Docker

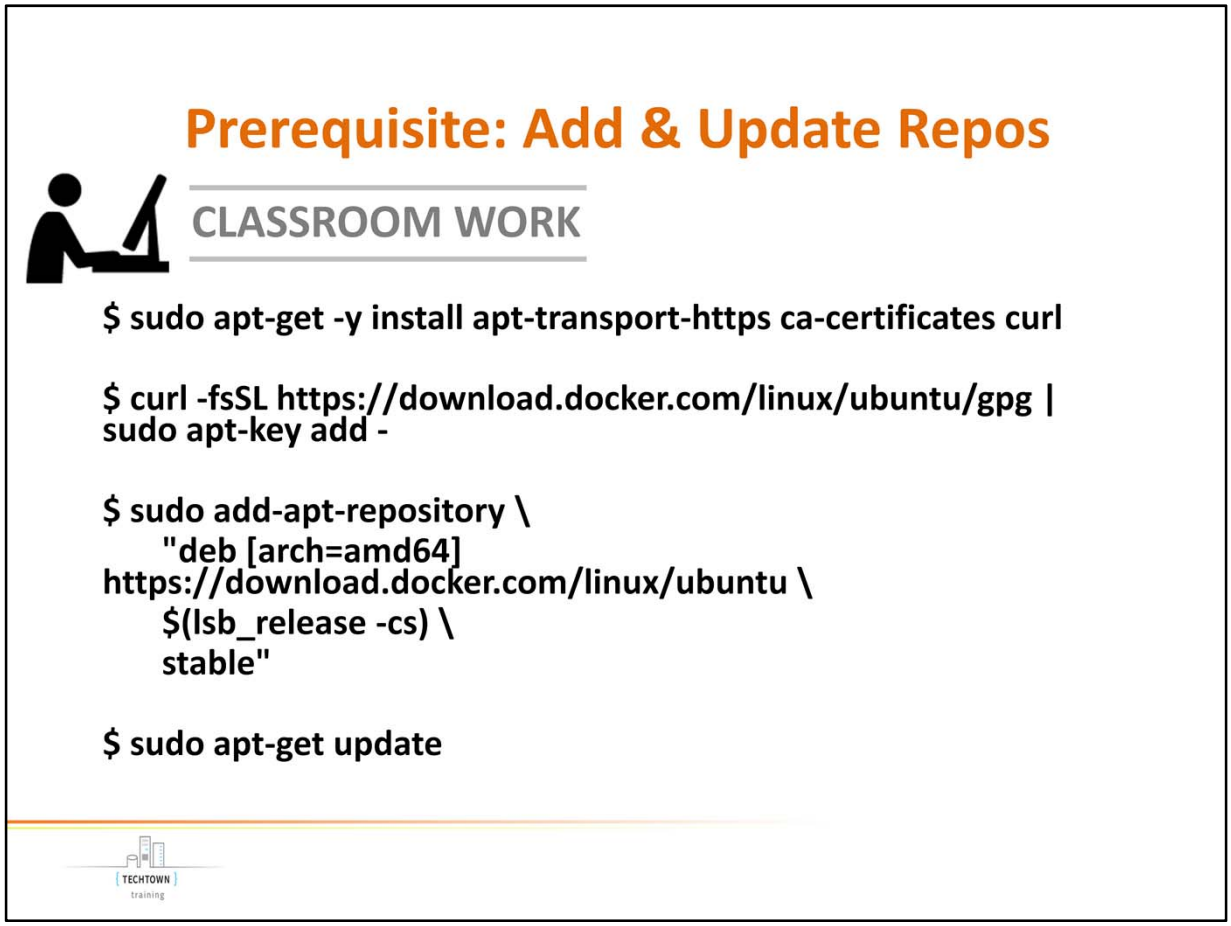
- Docker container engine is built from and on top of the Linux kernel, so it needs Linux to run natively*
  - Docker is increasingly being packaged with newer Linux distributions
  - Docker is also available on MacOS and Windows through the use of lightweight Linux VMs - HyperV-based on Windows, HyperKit-based on MacOS
- * Native Windows containers can also be used with Docker for Windows if you have very specific versions of Windows.

NOTES:



Ideally students are able to ssh directly into the classroom VMs.

NOTES:

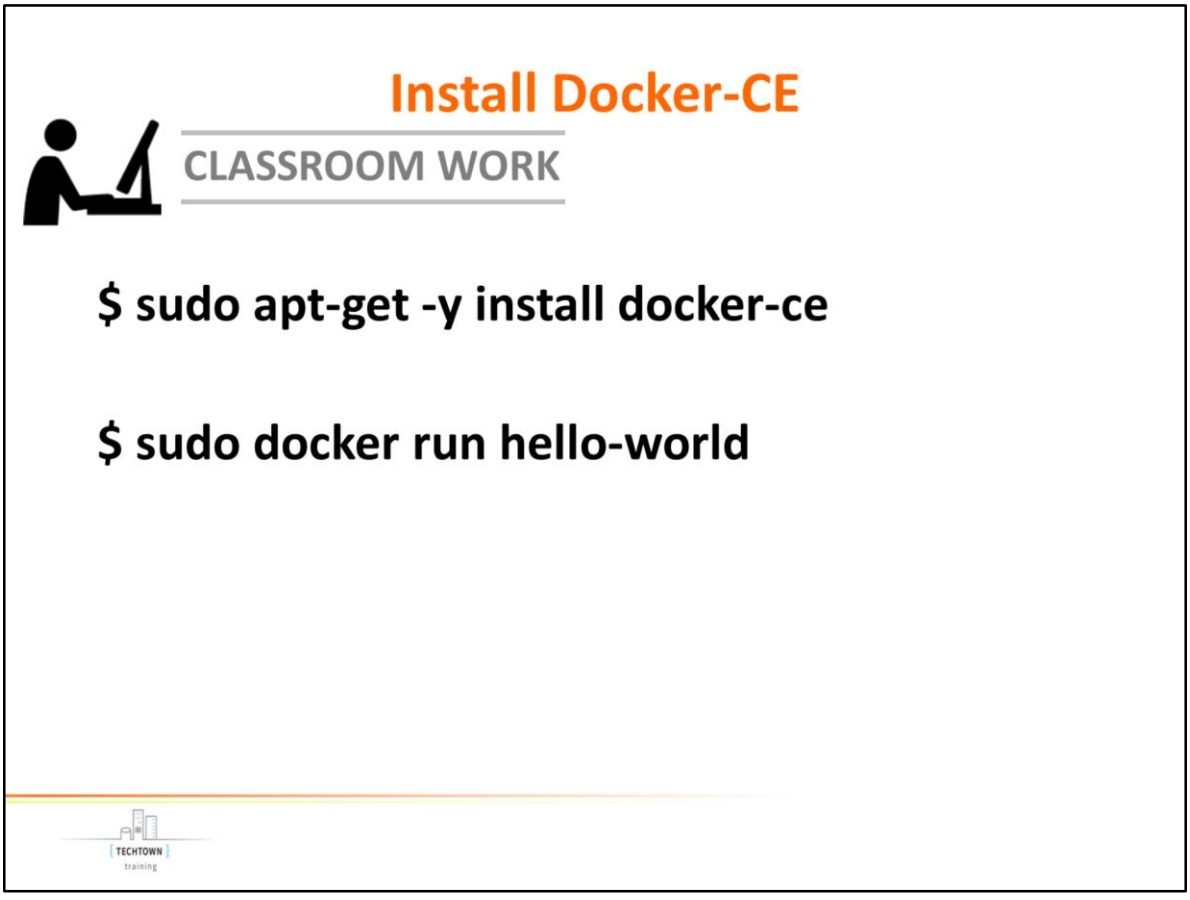


Set up the repository
Install packages (if necessary) to allow apt to use a repository over HTTPS:
$ sudo apt-get -y install apt-transport-https ca-certificates curl

Add Docker's official GPG key:
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Use the following command to set up the stable repository.
Note: The lsb_release -cs sub-command below returns the name of your Ubuntu distribution, such as xenial.

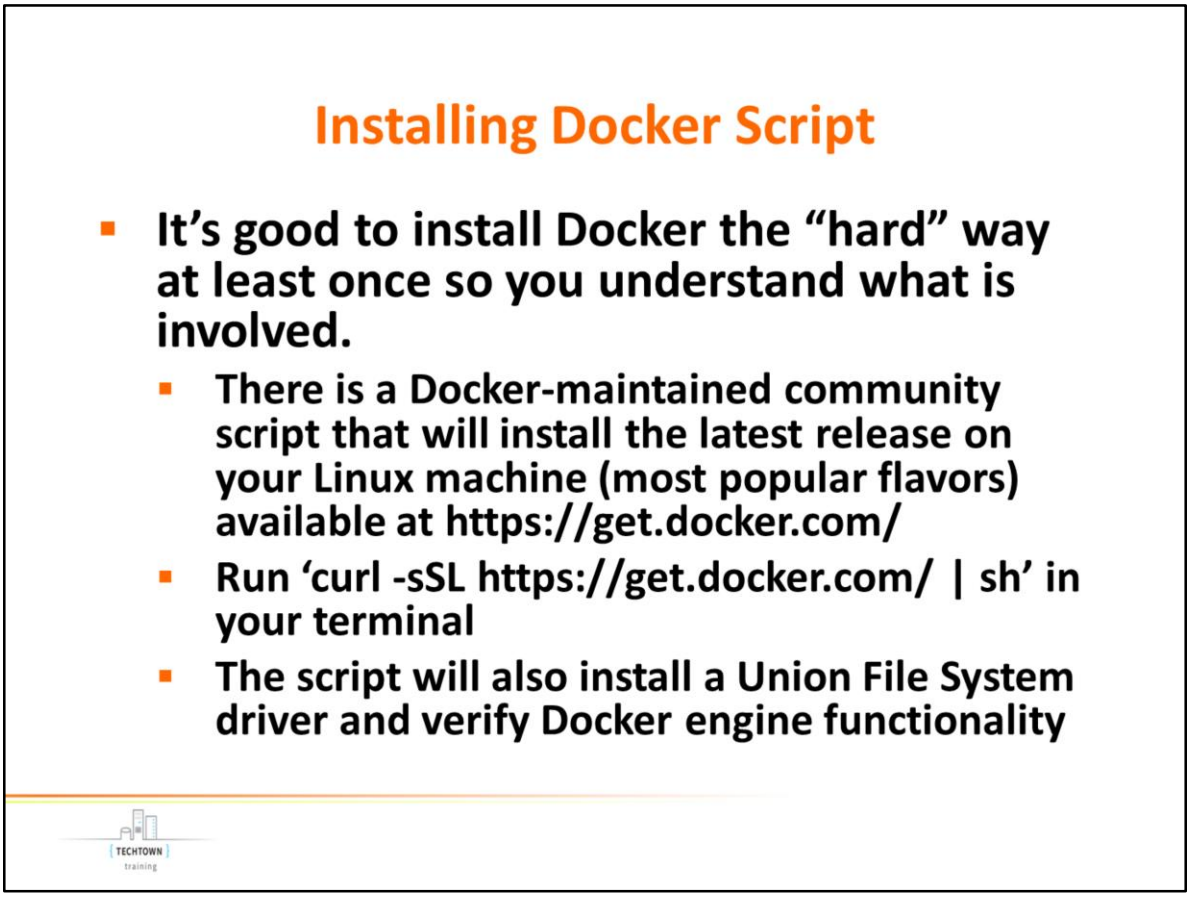
```
$ sudo add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) \
    stable"
```



Install docker Community Edition
$ sudo apt-get -y install docker-ce

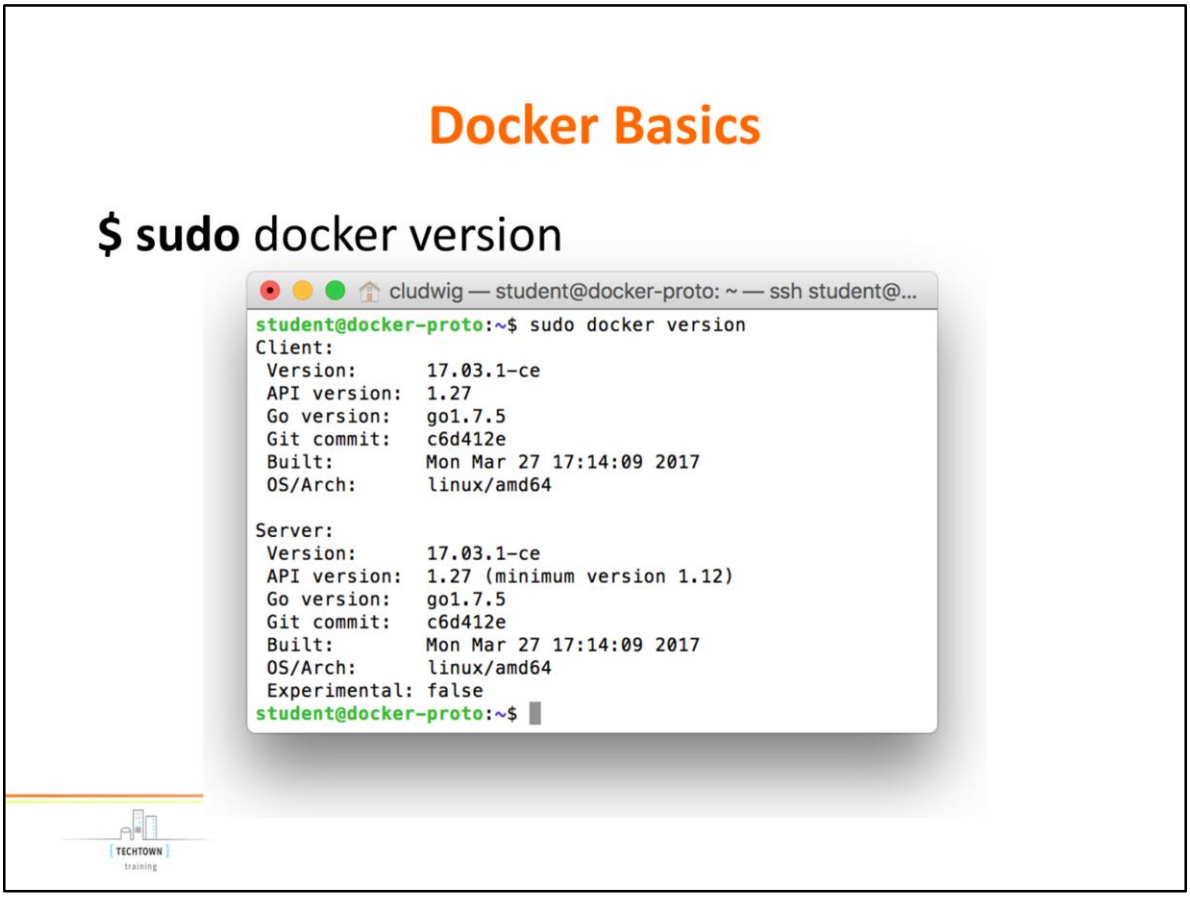
Verify docker is installed correctly
$ sudo docker run hello-world

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message. Then, it exits.



Docker is a fast moving project, and often critical bugs are fixed in newer versions. When working with Docker it is best to work with latest when possible.

Have students visit https://get.docker.com to see that it is a full script… we are passing the script to sh for execution.



Starting with v17.03.0-ce, Docker is on a monthly release cycle and uses a new YY.MM versioning scheme to reflect this. Two channels are available: monthly and quarterly.

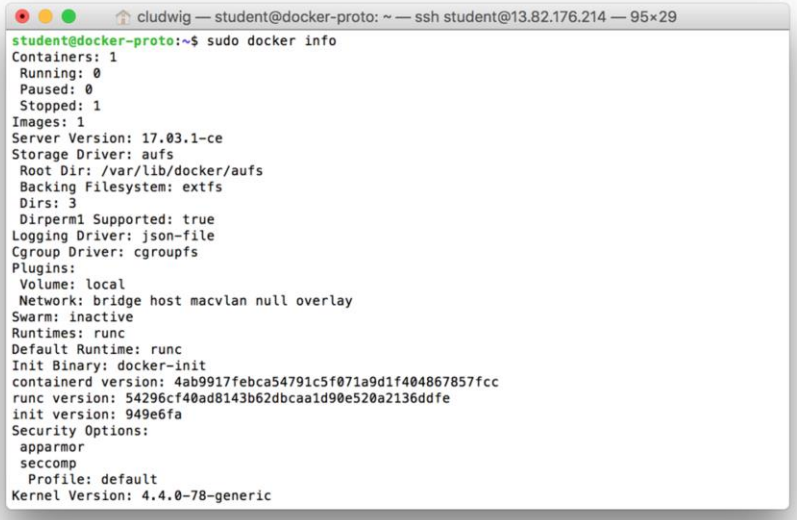
Any given monthly release will only receive security and bugfixes until the next monthly release is available. Quarterly releases receive security and bugfixes for 4 months after initial release. This release includes bugfixes for 1.13.1 but there are no major feature additions and the API version stays the same.

Docker version is useful for troubleshooting as well. The Server version will not be available if the Docker engine service is not running, which is a great place to start if you find docker commands are hanging or being unresponsive.

$ sudo service docker start # start the service

Docker Basics

$ sudo docker info



TECHTOWN training

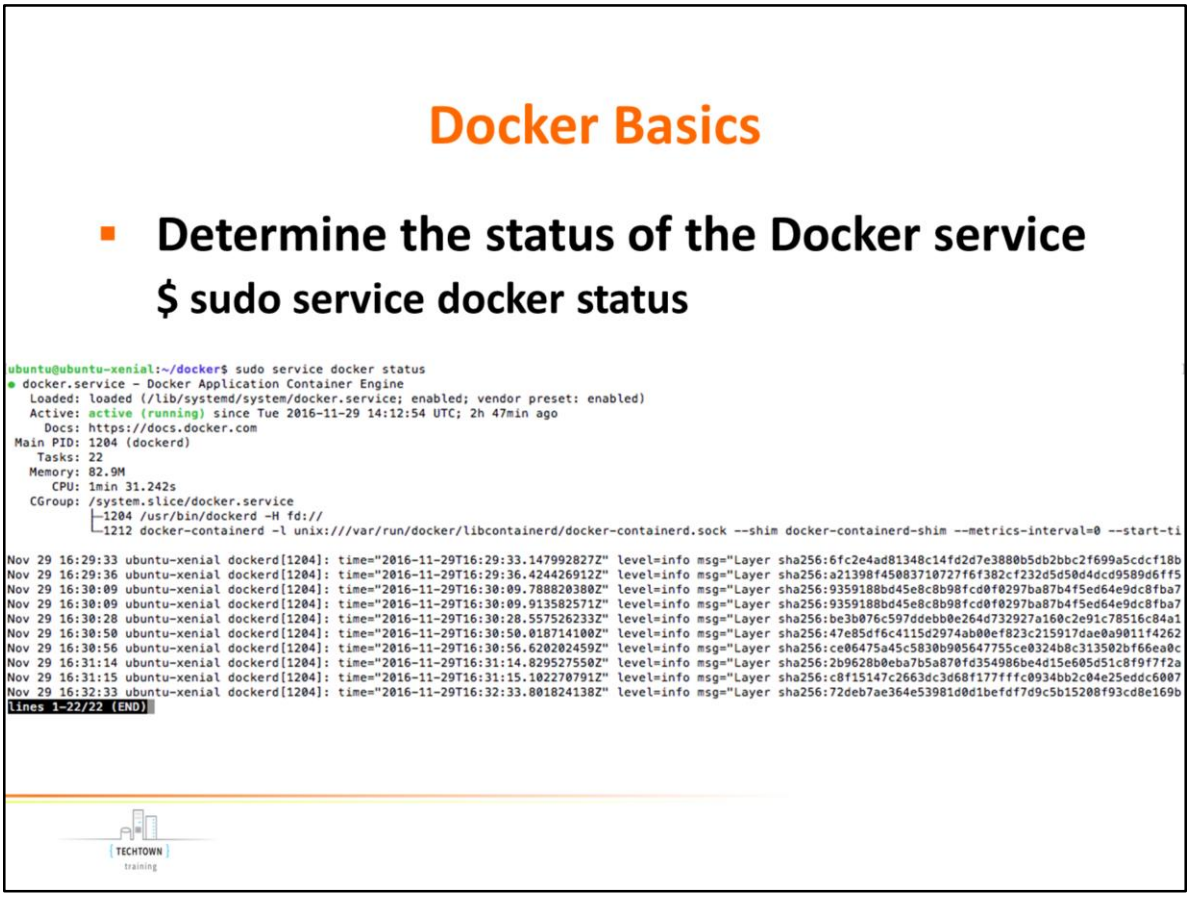
Provides a number of pieces of engine and system information.

NOTES:

Docker Basics

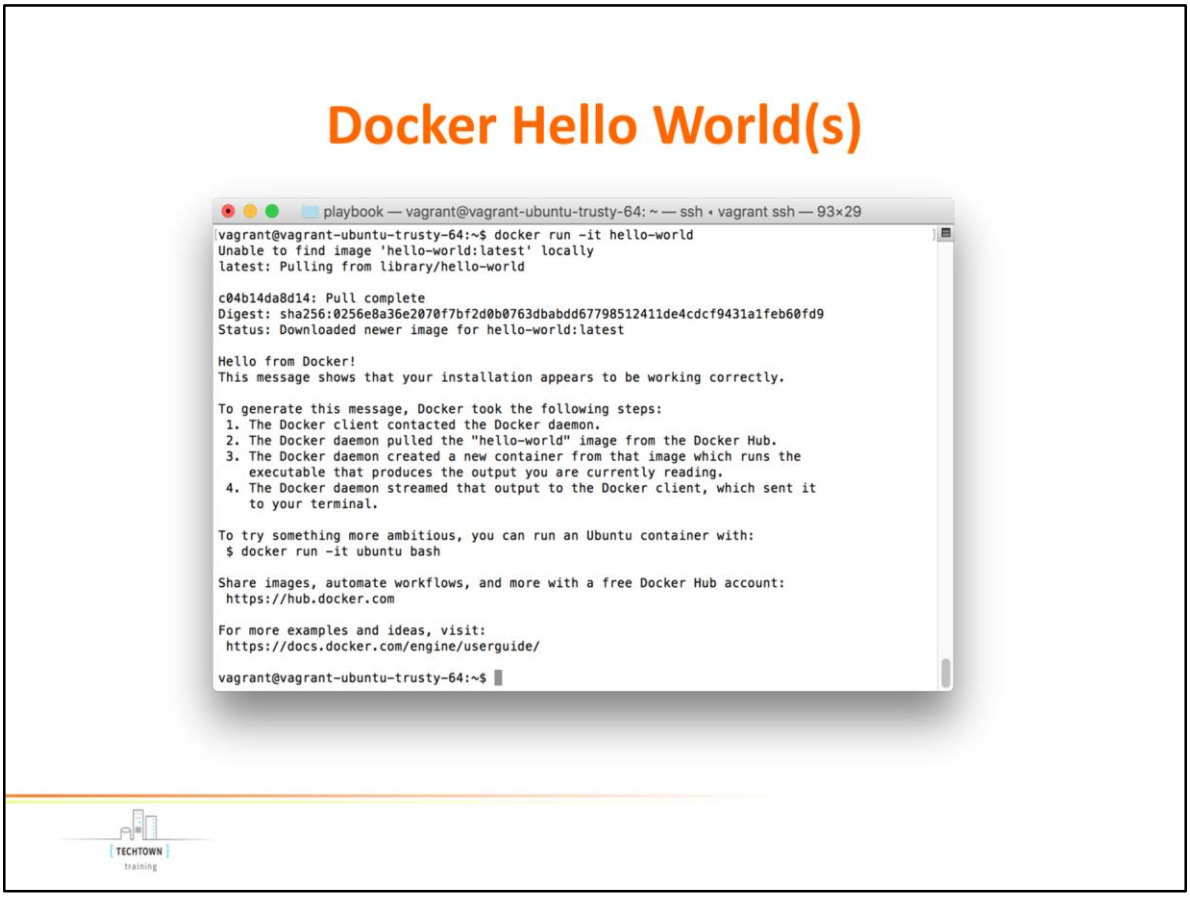
- Standard command format:
  - docker [NOUN] verb
- $ sudo docker --help #displays help and management subcommands (nouns)
- $ sudo docker NOUN --help
- There are reasonable defaults and short forms of option flags for many things. i.e.:
  - “docker pull” assumes “image”
  - “docker stop” assumes “container”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

NOTES:



Also a great starting point if Docker is unresponsive.

$ sudo service docker start # start the service

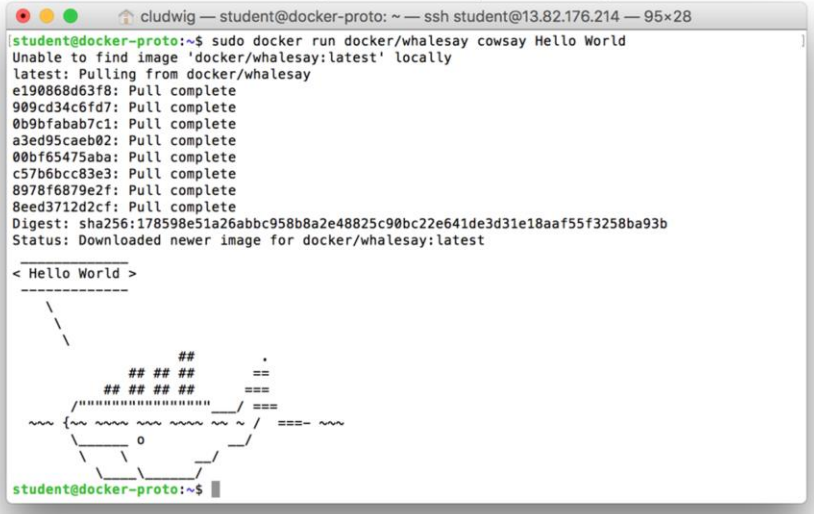


docker run -it hello-world

- This will likely NOT work at first… must use sudo. We will fix that soon.

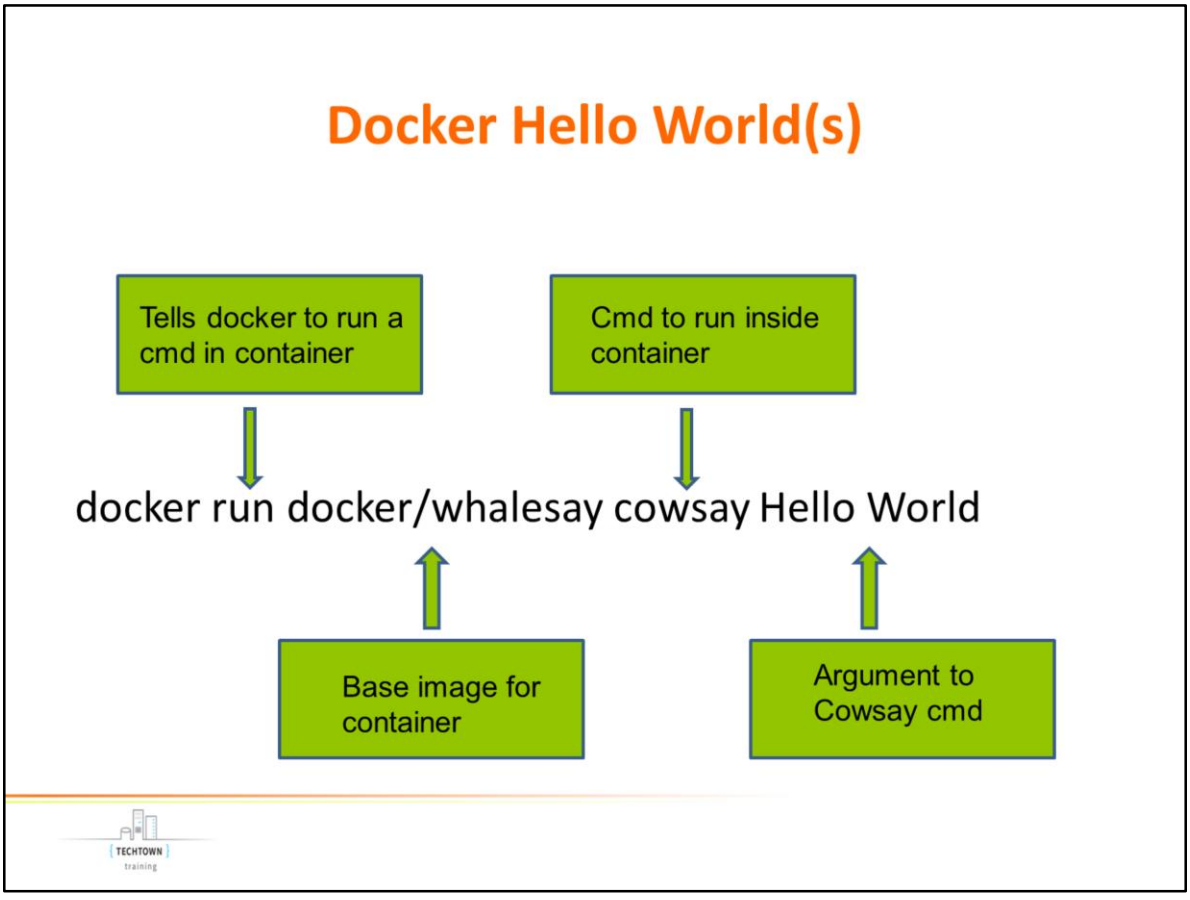
Docker Hello World(s)

$ sudo docker run docker/whalesay cowsay Hello World



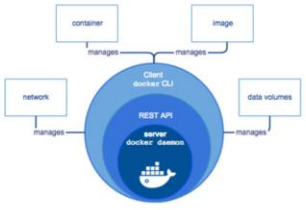
$ sudo docker run docker/whalesay cowsay Hello World

NOTES:



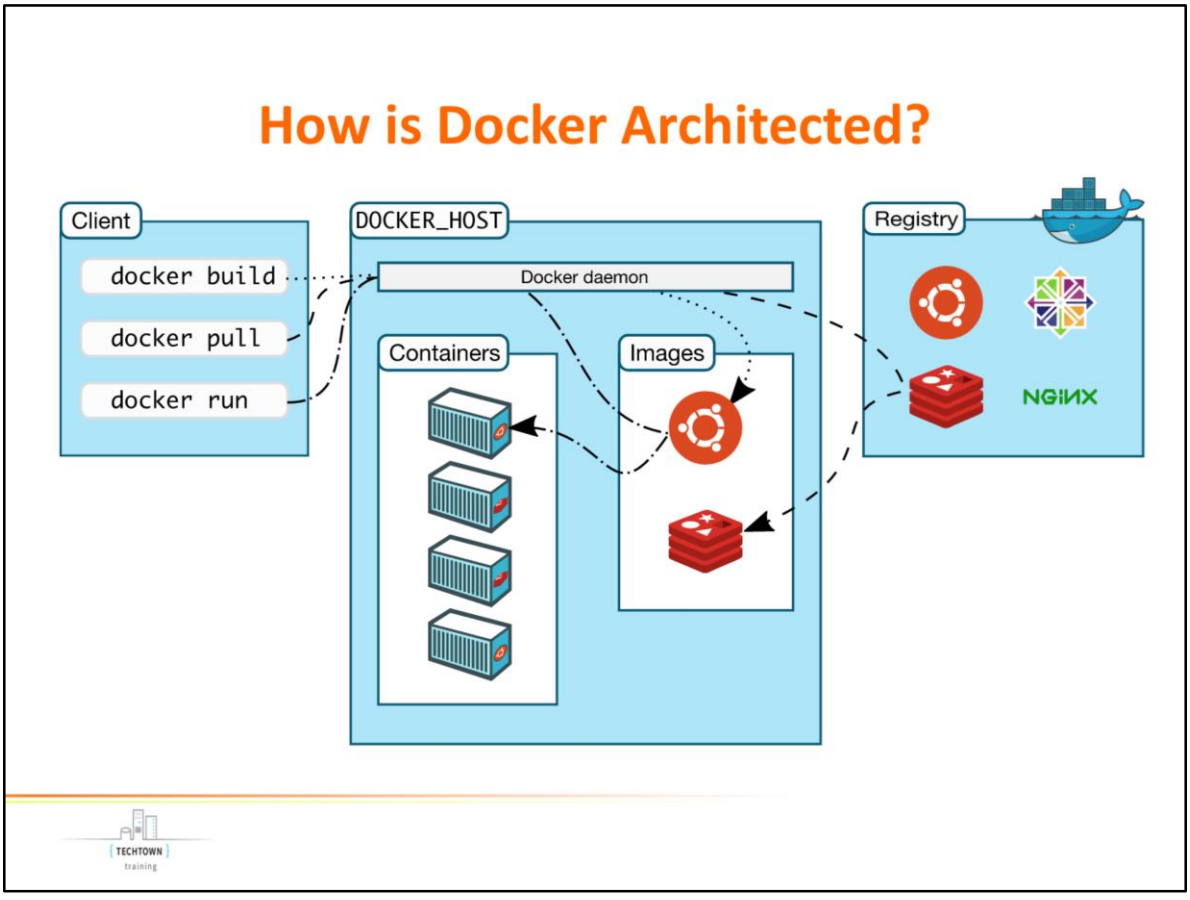
NOTES:

What is 'Docker Engine'?



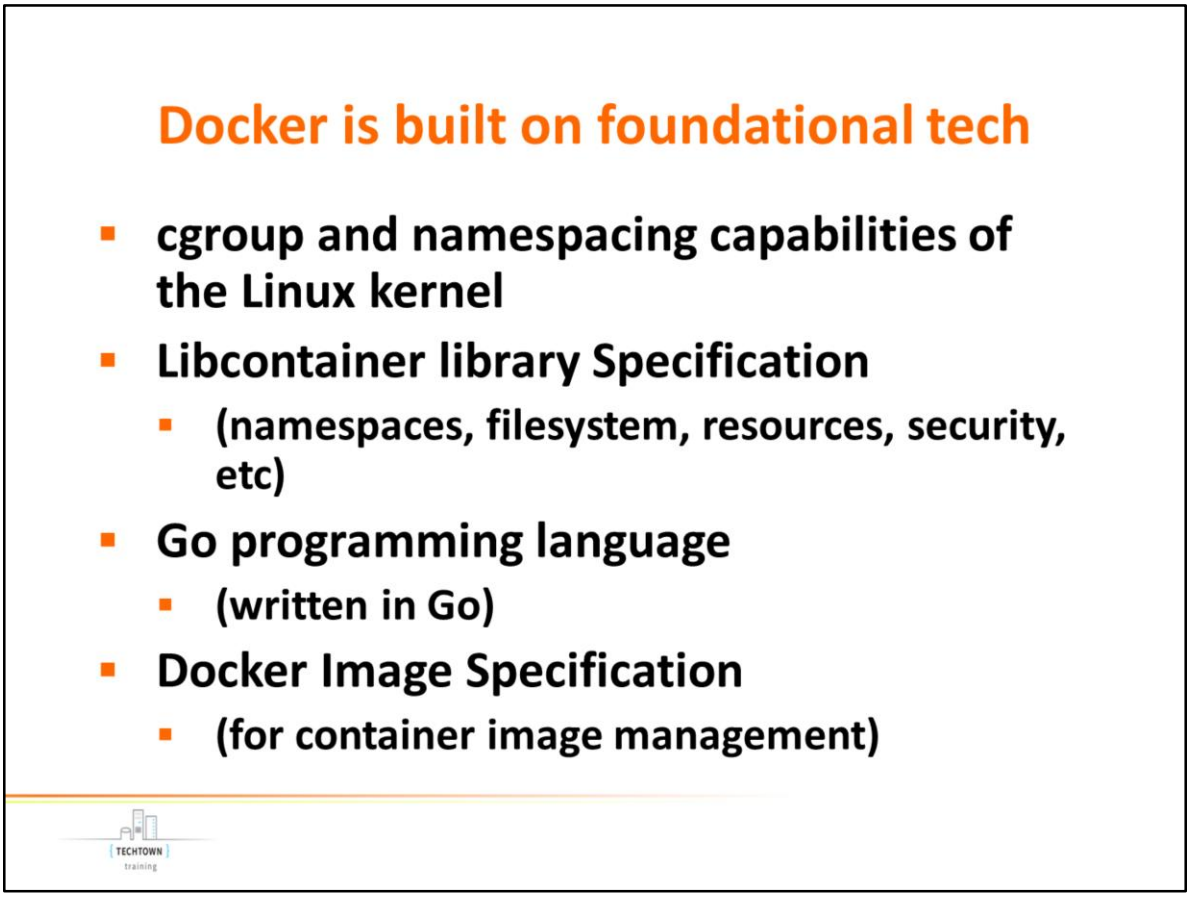
TECHTOWN
training

The CLI uses the Docker REST API to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI.
The daemon creates and manages Docker *objects*, such as images, containers, networks, and data volumes.



Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate via sockets or through a REST API.

Note: The Docker daemon runs with root privileges which introduces an attack opportunity. More behind this: https://docs.docker.com/engine/security/security/



cgroups (abbreviated from control groups) is a Linux kernel feature that limits, accounts for, and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes. Engineers at Google (primarily Paul Menage and Rohit Seth) started the work on this feature in 2006 under the name "process containers".

Namespaces are a feature of the Linux kernel that isolates and virtualizes system resources of a collection of processes. Examples of resources that can be virtualized include process IDs, hostnames, user IDs, network access, interprocess communication, and filesystems. Namespaces are a fundamental aspect of containers on Linux.

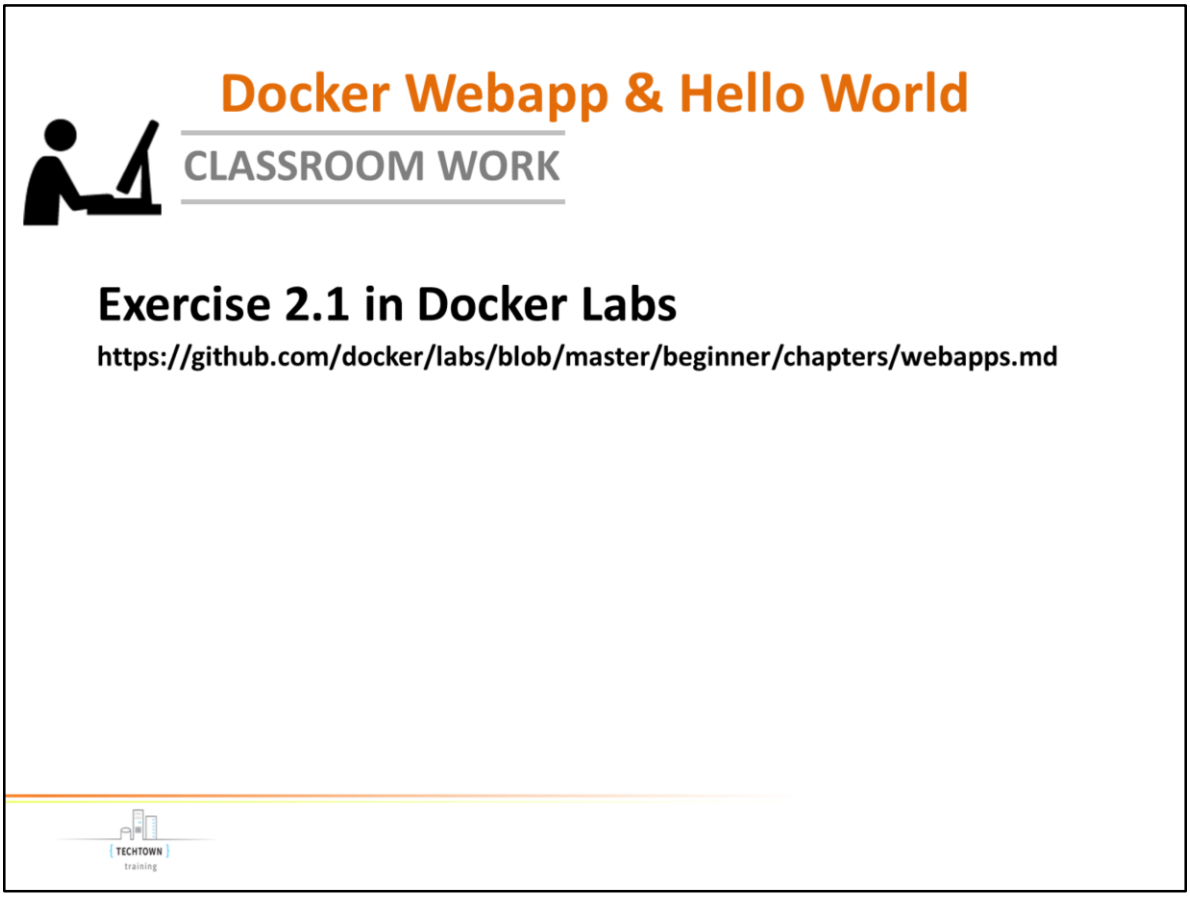
No ‘sudo’ Configuration

- Run docker without sudo
  - https://docs.docker.com/engine/installation/linux/linux-postinstall/
- Create Docker group:

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

Newer installs do this for you, but of course you still have to logout and back in.

NOTES:



https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

NOTES:

Docker Images

Part 2: DevOps Docker

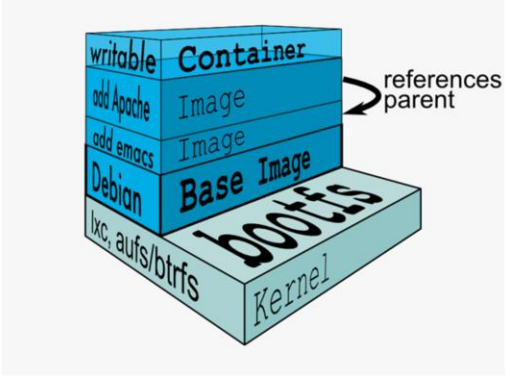
TECHTOWN training

NOTES:

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**



TECHTOWN training

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS (the default for Ubuntu), btrfs, vfs, and DeviceMapper (the default for Red Hat and Centos.

How AuFS Works

1. Mount a base image as read-only
2. Add read-write file system over read-only file system
3. Change some files and directories to top as read- write file system
4. It is committed
5. Save incremental files and directories in new image
6. Mount the image as read-only and add read-write file system on top

Example Docker Layers

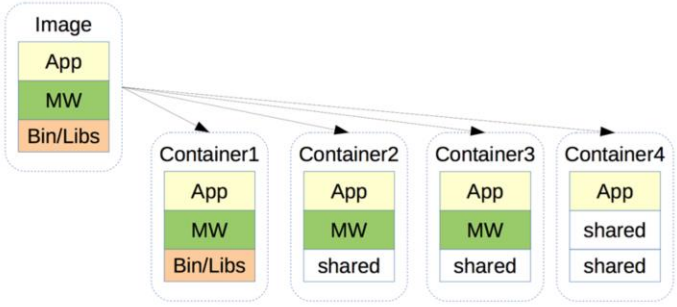
Docker Layer – Each Docker image is composed of a series of layers. Docker uses **Union File Systems** to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

TECHTOWN training

NOTES:

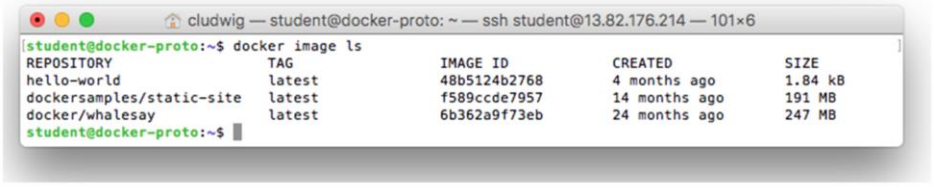
Benefits of Union File Systems

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



NOTES:

$ docker image ls



TECHTOWN training

Displays all local images.
Alternative:

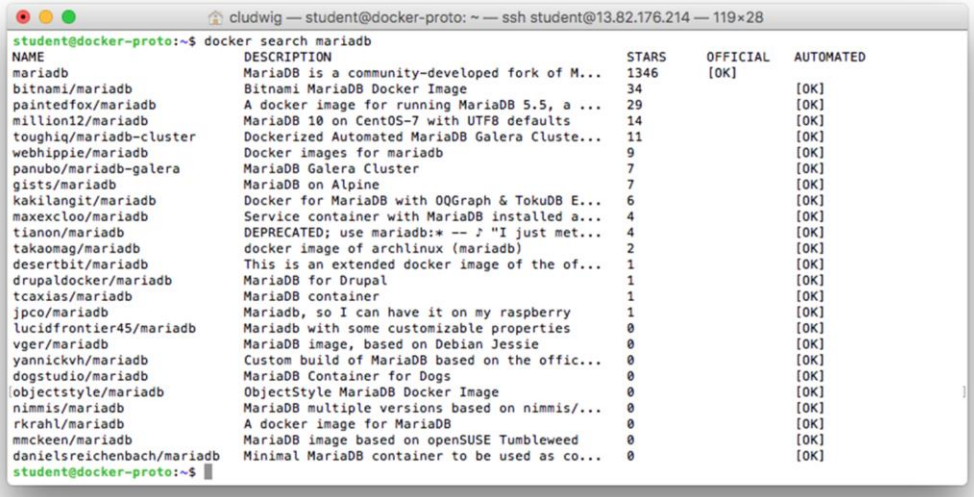
- $ docker images #old-style syntax (prior to 1.13)

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

NOTES:

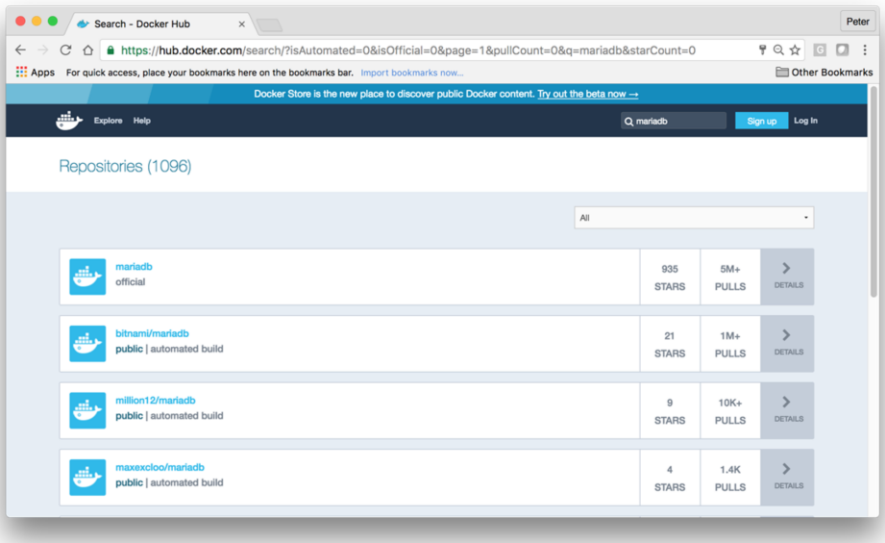
$ docker search [searchterm]



$ docker search mariadb

- Look for “Official” images.
- Look for images that the community upvotes with “Stars”

Docker Hub Equivalent



TECHTOWN training

NOTES:

$ 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

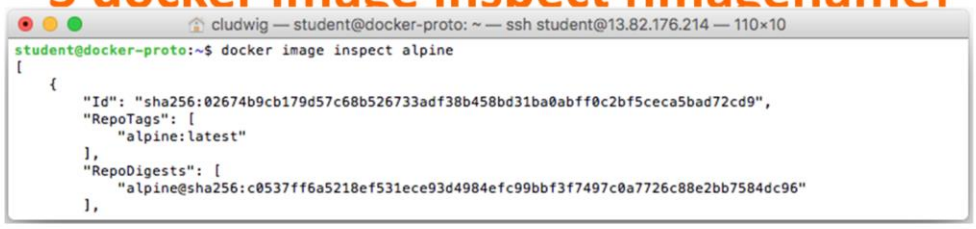
Debian – 123 MB
Alpine – 5 MB

```
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:~$
```

TECHTOWN training

NOTES:

$ docker image inspect [imagename]



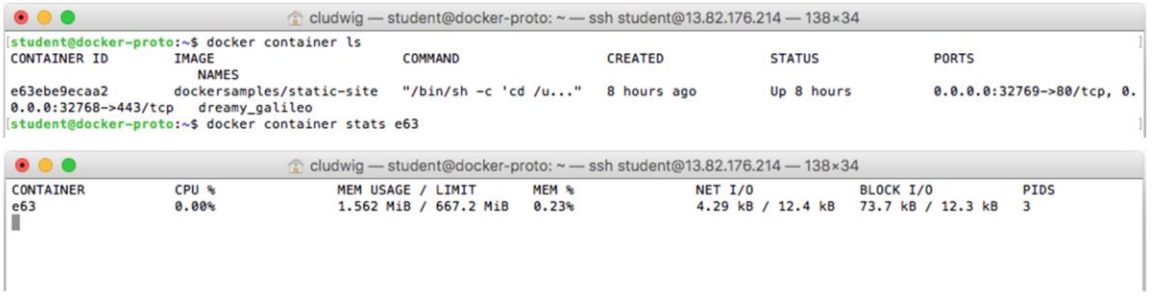
- **ID** The unique identifier for the container
- **State** This stanza has various status flags and the process id for the container. Using the ExitCode from this State element a graceful shutdown or recovery process could be initiated. The following format will return just the ExitCode of the most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

TECHTOWN training

It can be useful to narrow down information from docker inspect with grep as well

NOTES:

$docker container stats

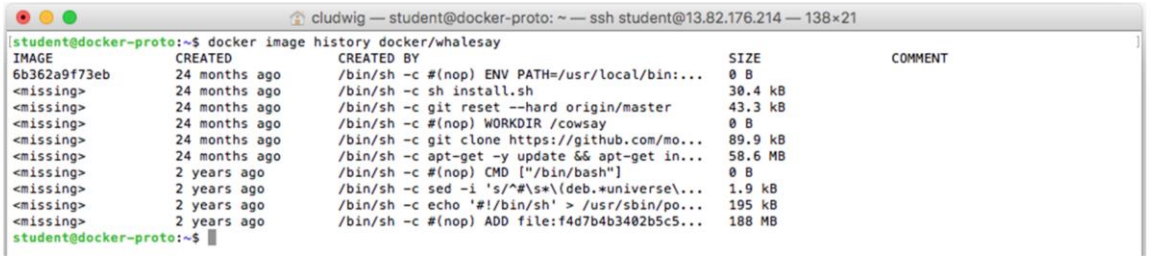


- Provides the CPU, memory, network, and other monitoring KPI's of a docker container.
- Multiple containers can be profiled as well
- All containers can be profiled by leaving out the container id(s)

TECHTOWN training

NOTES:

$ docker image history docker/whalesay



Use history to view layers in an image

TECHTOWN training

NOTES:

Removing Docker Containers & ImagesRemoving Docker Containers & Images

#!/bin/bash

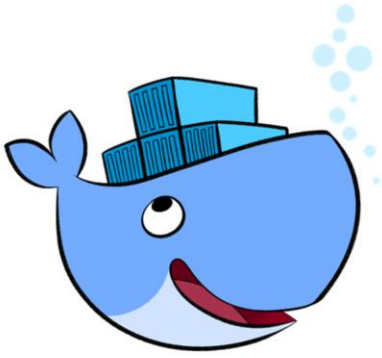
*# **Remove** all stopped containers*
docker container prune [-f]

*# **Remove** all unused images*
docker image prune [-f] [-a]

-f option skips coAnfirmation.
-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

NOTES:



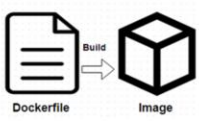
Docker Files

Part 3: DevOps Docker

TECHTOWN training

NOTES:

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

TECHTOWN training

NOTES:

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

NOTES:

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

NOTES:

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”]

NOTES:

Dockerfile

ADD/COPY command

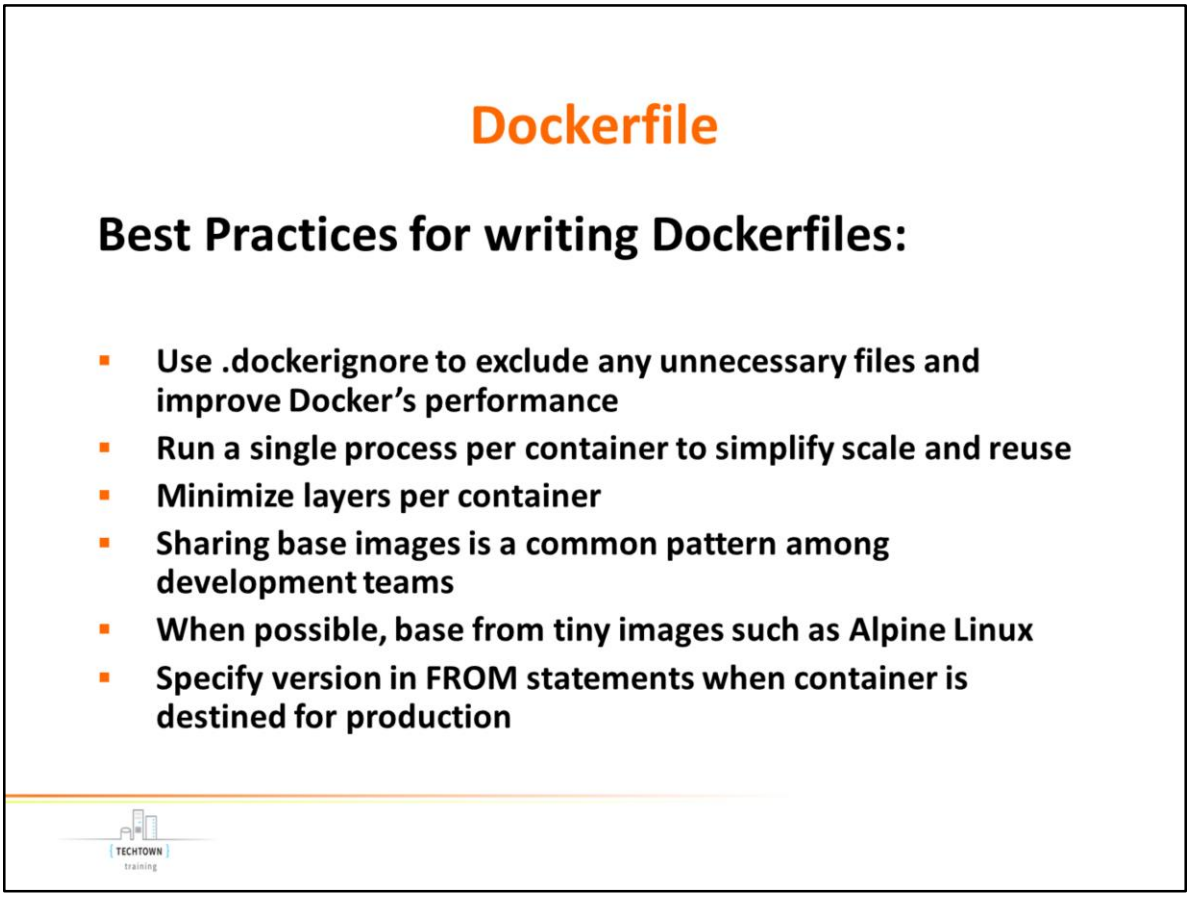
Places files onto a file system. ADD performs the same as copy, but also allows the <src> to be a URL. It will also unpack recognized compression formats.

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.

NOTES:



More here: https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Dockerfile

- Dockerfile build syntax

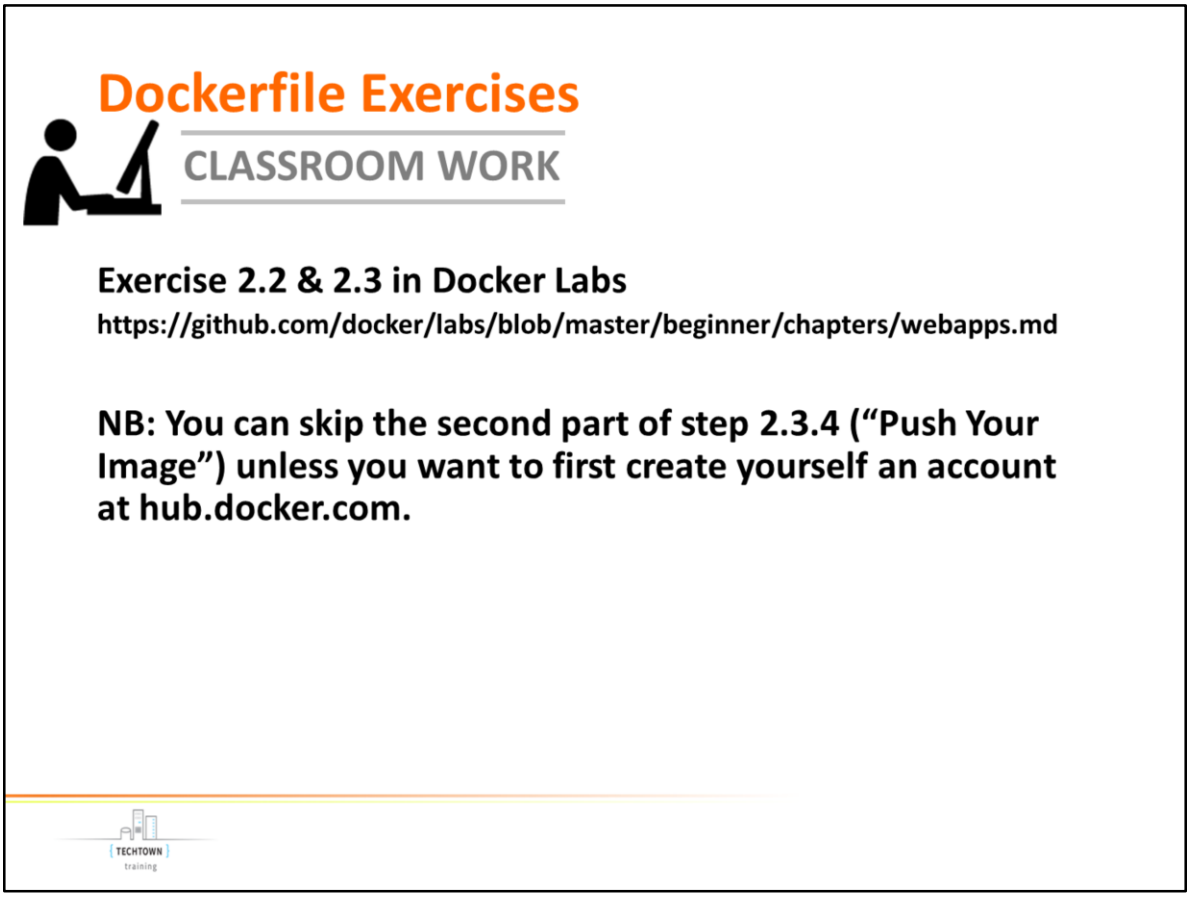
docker build .
docker build -t <**namespace**/image name:**versiontag**> /path
docker build -t myname/jenkins .

- Then run the newly created image

Docker run –t myname/jenkins

-t (--tag): assign tag

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.



https://github.com/docker/labs/blob/master/beginner/chapters/webapps.md

NOTES:

Dockerfile Optimization

CLASSROOM WORK

This exercise covers optimizing dockerfile to reduce the number of layers.

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

TECHTOWN training

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

NOTES:

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
```

https://github.com/codeship/go-hello-world

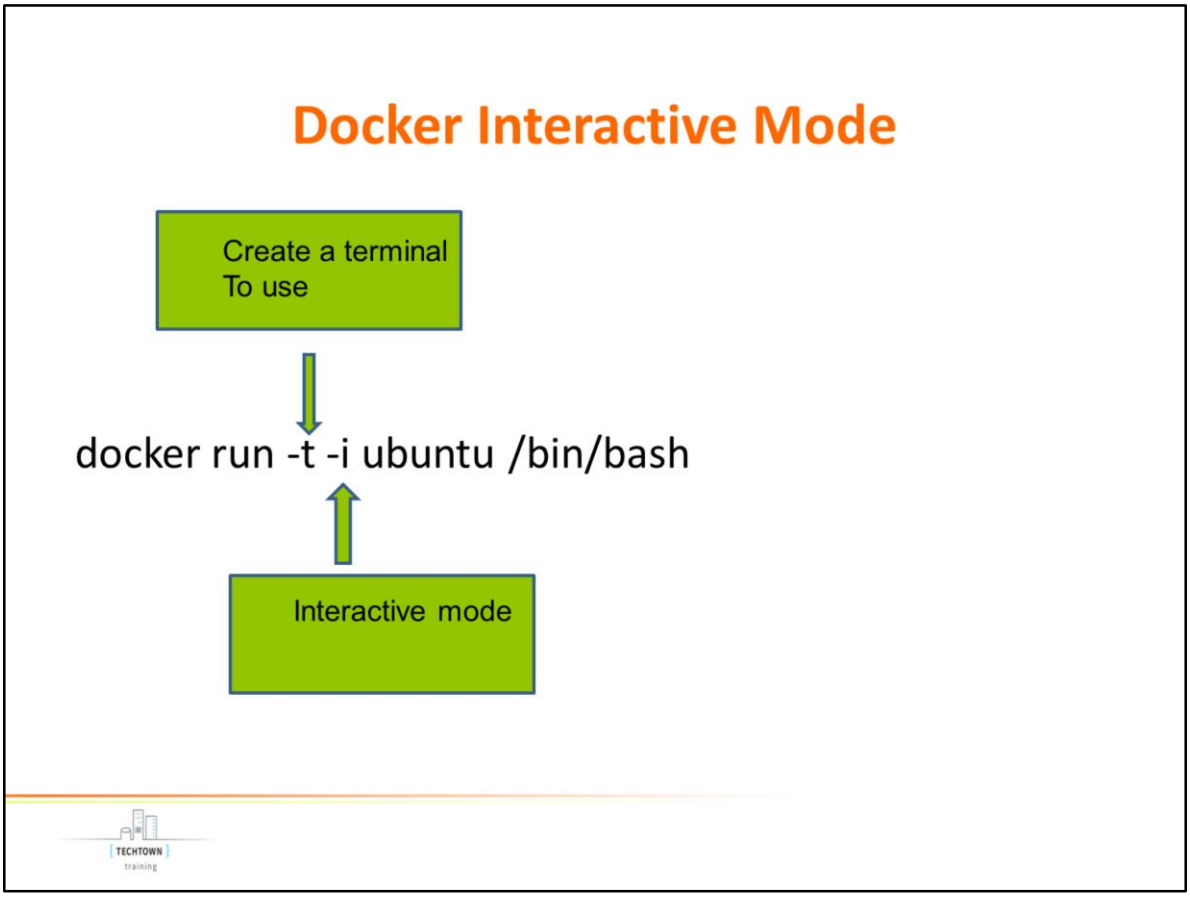
NOTES:

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:/#
```

NOTES:



As we've seen, we can combine these two options to the more common **-it**

NOTES:

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

NOTES:

Docker Interactive Mode

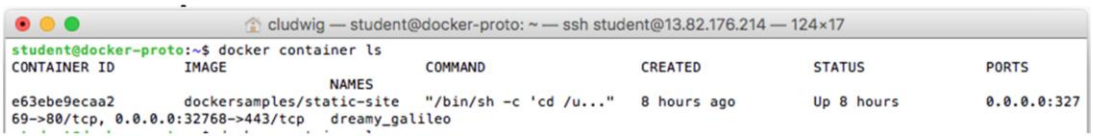
Quick Tip: Be aware that alpine and other distros may have a different entry point (sh vs bash)

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

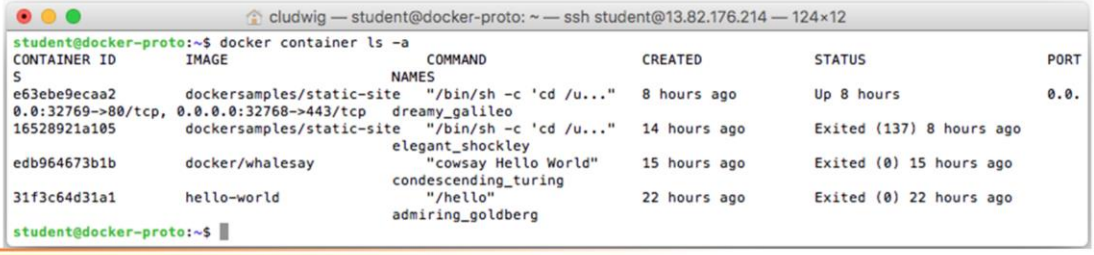
NOTES:

See all containers

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

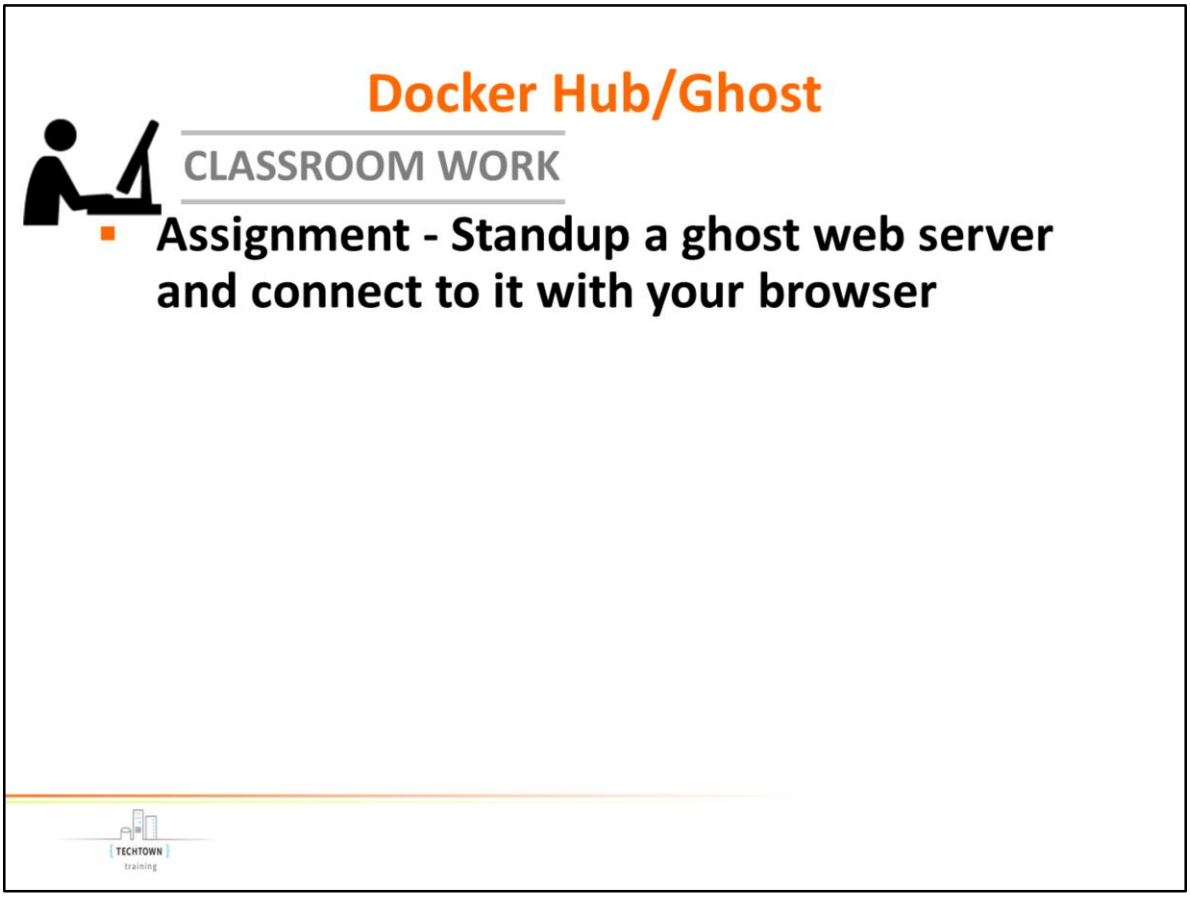


- $ docker container ls -a #show all containers (including running, stopped, exited, etc.)



TECHTOWN training

NOTES:



Answer:

1. Find ‘ghost’ on Docker Hub

https://hub.docker.com/_/ghost/

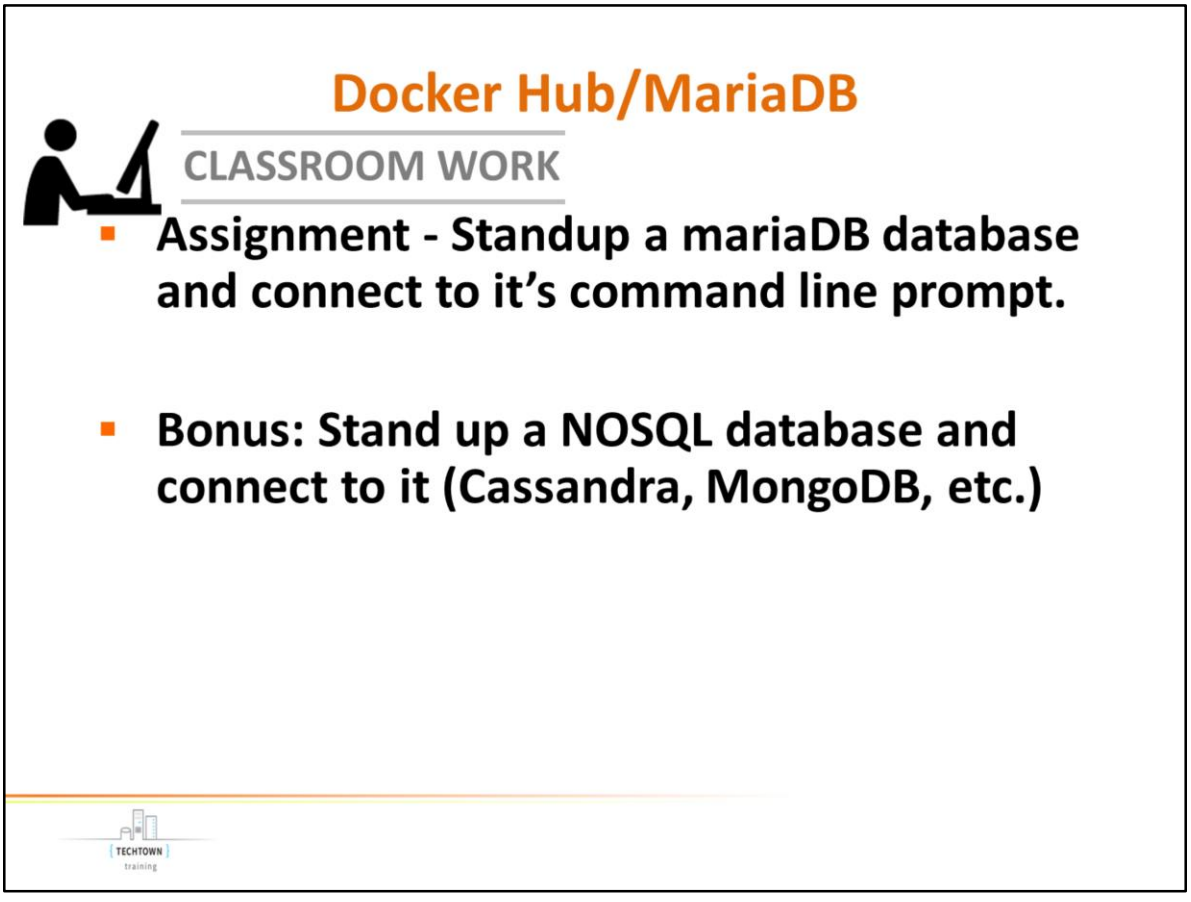
2. Read instructions and execute the docker run command with ghost

docker run --name some-ghost -p 8080:2368 -d ghost

3. Navigate to <lp>:8080 and observe the running web server

Bonus: Run ’docker top <container id>’ against the ghost container to observe running processes

Docker is extremely useful for web servers. You don’t have to worry about dependencies, incompatible versions, setting environment variables, just call and run. Any service that exposes itself over TCP can be containerized and easily run.



Answer:

docker run --name some-mariadb -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mariadb:latest

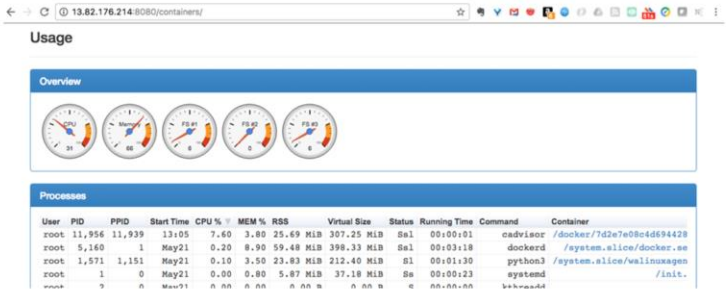
docker run -it --link some-mariadb:mysql --rm mariadb sh -c 'exec mysql -h"$MYSQL_PORT_3306_TCP_ADDR" -P"$MYSQL_PORT_3306_TCP_PORT" -uroot -p"$MYSQL_ENV_MYSQL_ROOT_PASSWORD"'

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



https://github.com/google/cadvisor

NOTES:

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

NOTES:

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
```

https://github.com/docker/docker.github.io/blob/master/registry/deploying.md

NOTES:

Docker Registry

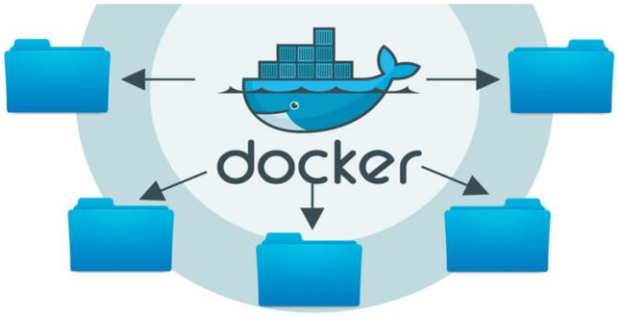
- Search Registry
  - https://docs.docker.com/registry/spec/api/#listing-repositories

```
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"]}
```

https://docs.docker.com/registry/spec/api/#listing-repositories
https://docs.docker.com/registry/spec/api/#listing-image-tags



Docker Volumes

Part 4: DevOps Docker

TECHTOWN training

NOTES:

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.

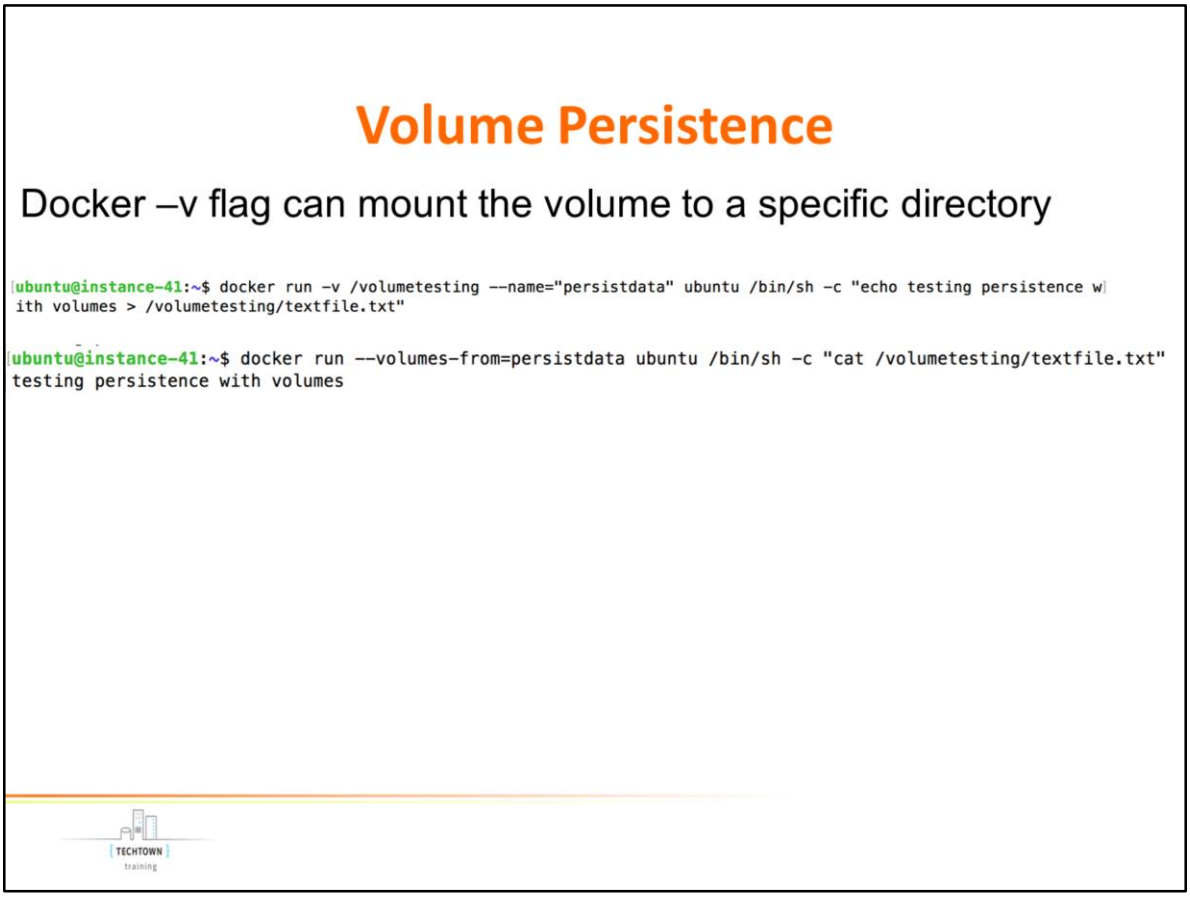
Data volumes are designed to persist data, independent of the container's lifecycle. Docker therefore *never* automatically deletes volumes when you remove a container, nor will it "garbage collect" volumes that are no longer referenced by a container.

NOTES:

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.

NOTES:



docker run -v /volumetesting --name="persistdata" ubuntu /bin/sh -c "echo testing persistence with volumes > /volumetesting/textfile.txt”

docker run --volumes-from=persistdata ubuntu /bin/sh -c "cat /volumetesting/textfile.txt"

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

NOTES:

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
```

NOTES:

Dangling Volumes

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

$ docker volume prune

NOTES:

Dangling Volumes

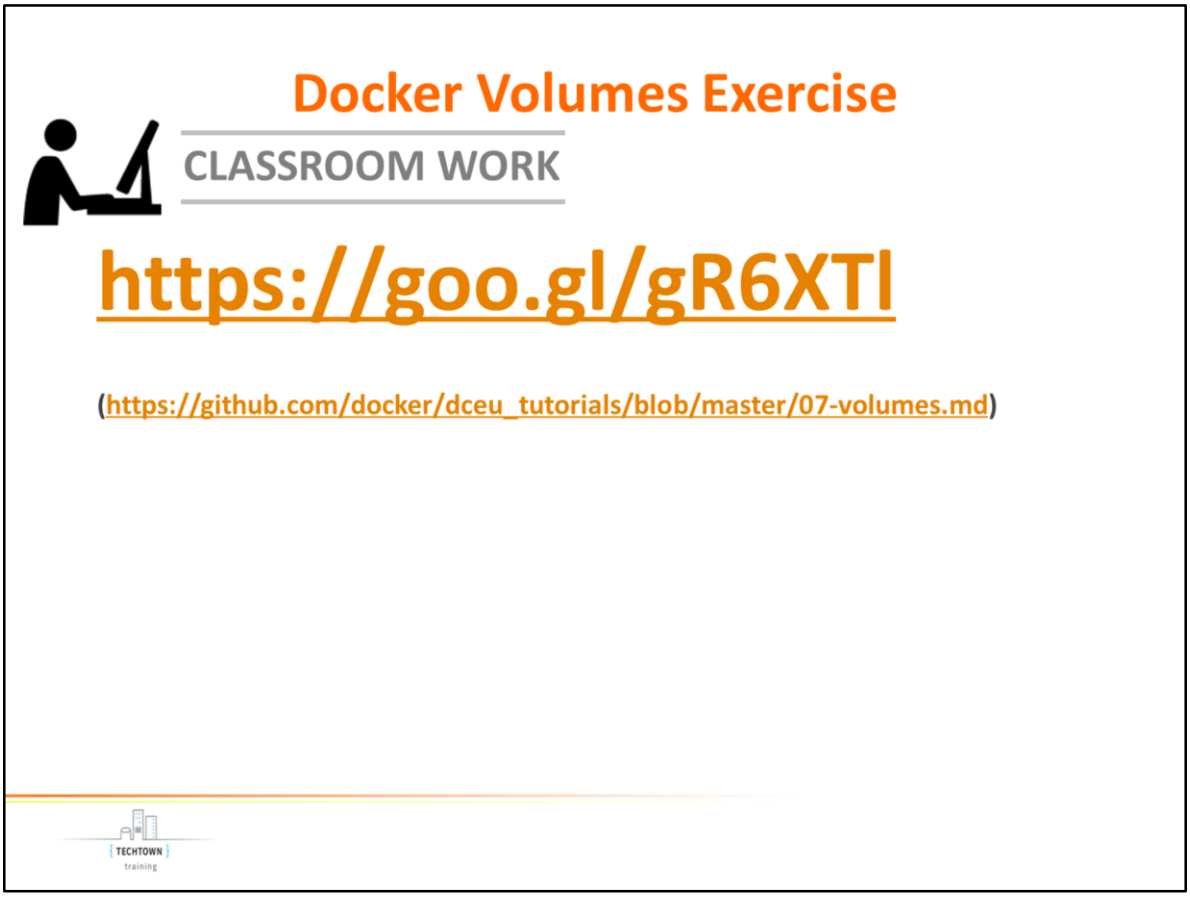
- Before

```
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
```

NOTES:

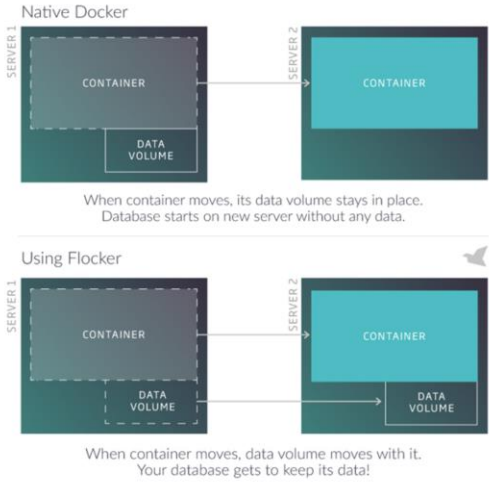


This is a nice full volumes exercise that shows volume persistence, cross-container volumes, etc.

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

Docker Volume Plugin - Flocker

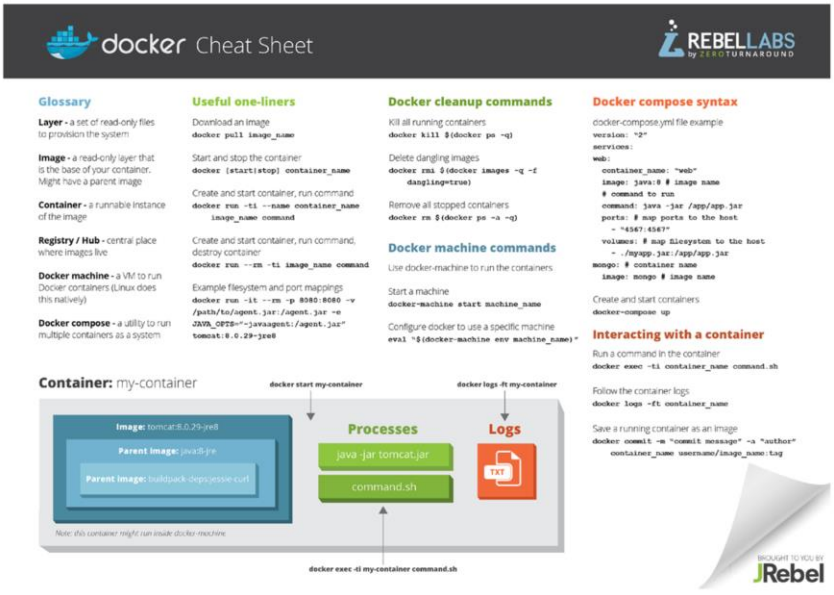
https://clusterhq.com/flocker/



TECHTOWN training

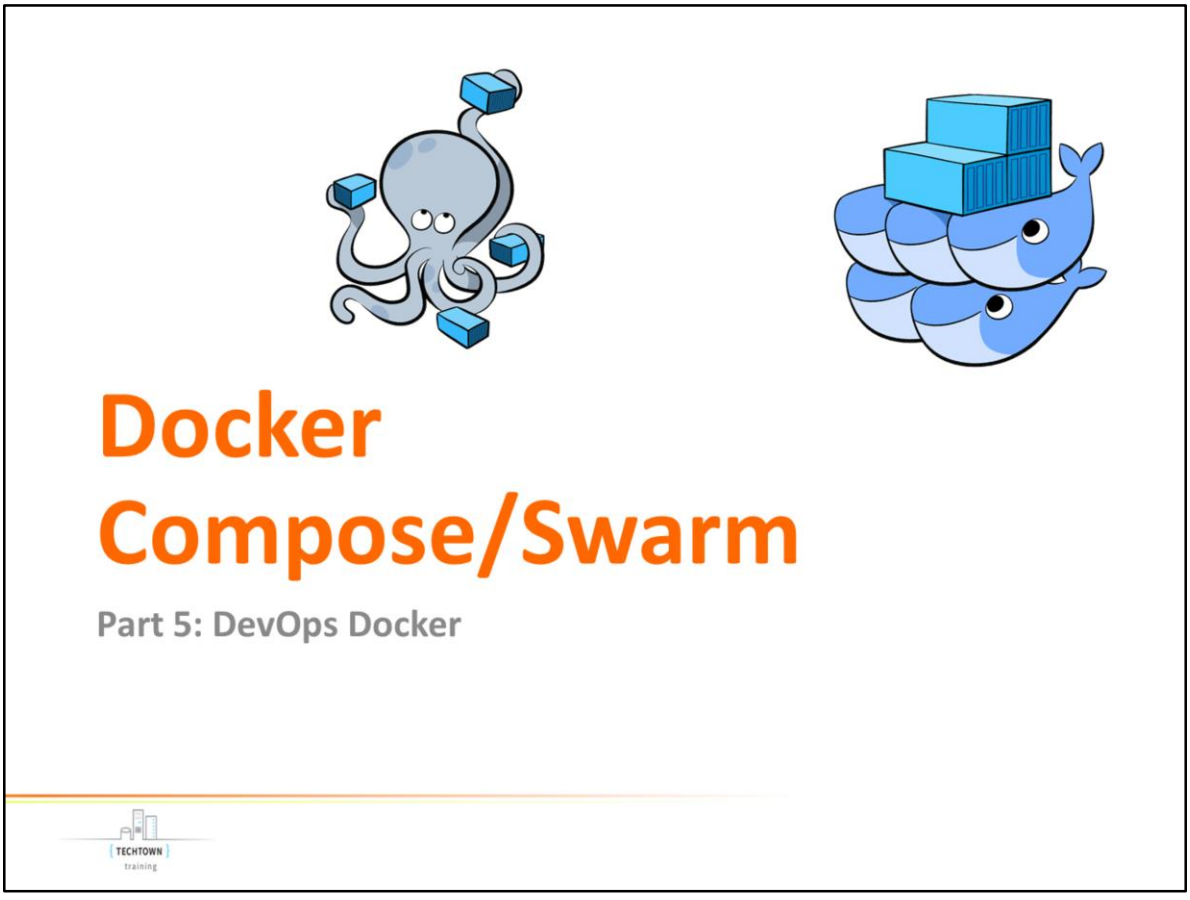
NOTES:

Docker Cheat Sheet



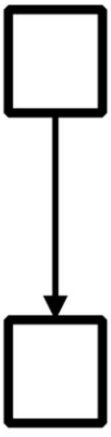
Printable sheet available at:

https://zeroturnaround.com/rebellabs/docker-commands-and-best-practices-cheat-sheet/



- Swarm used to be a standalone daemon that was Docker API compatible.
- Swarm is now a “mode” of the standard Docker Daemon
- With Swarm now integrated directly into docker, many (if not all) use cases for stand-alone Docker Compose can now be implemented using Swarm mode. You will see that Docker labs have been updated to demonstrating clustering using Swarm instead of Compose because of the in-built ability to span Docker hosts that Swarm offers.

Launching multiple containers is clumsy

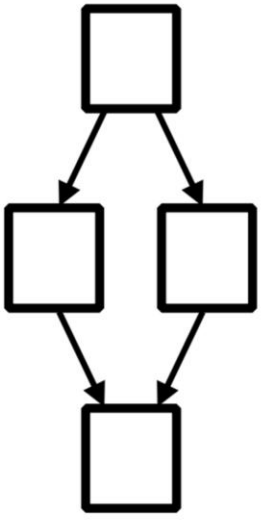


```
$ docker pull redis:latest

$ 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
```

TECHTOWN training

NOTES:

Launching multiple containers is clumsy

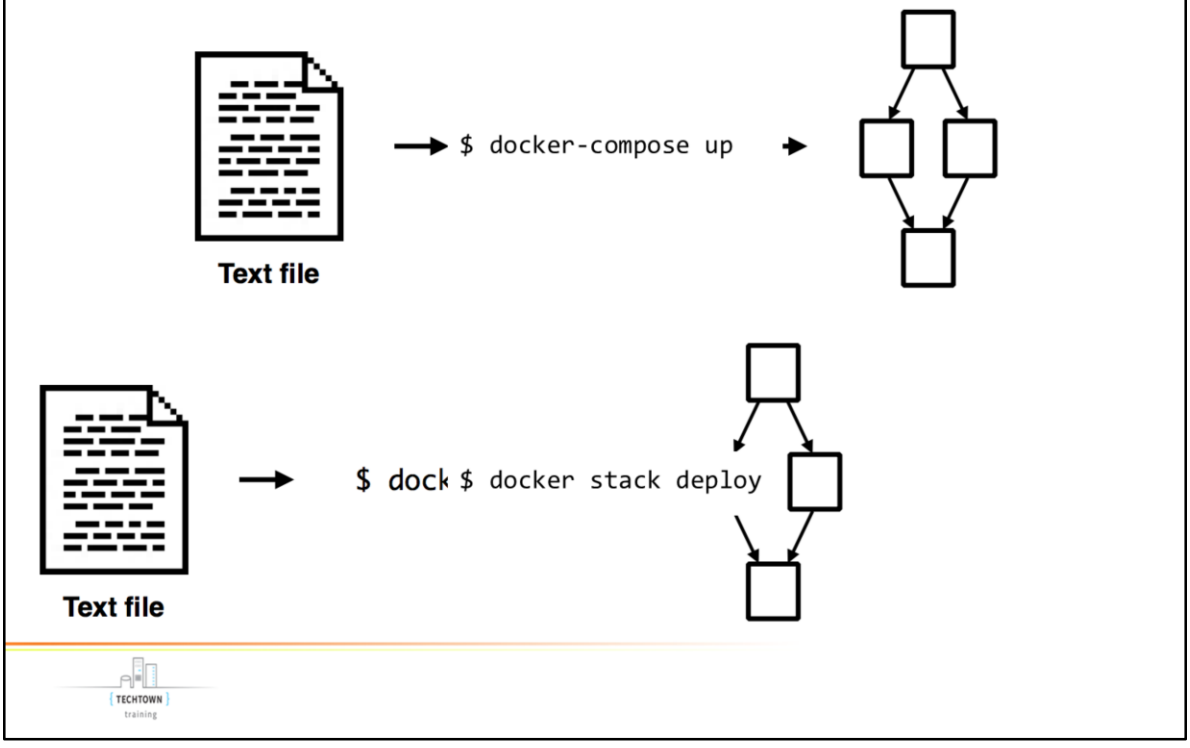


```
$ docker pull ...
$ docker pull ...
$ docker build ...
$ docker build ...

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

TECHTOWN training

NOTES:

Docker Compose or Swarm launches



NOTES:

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

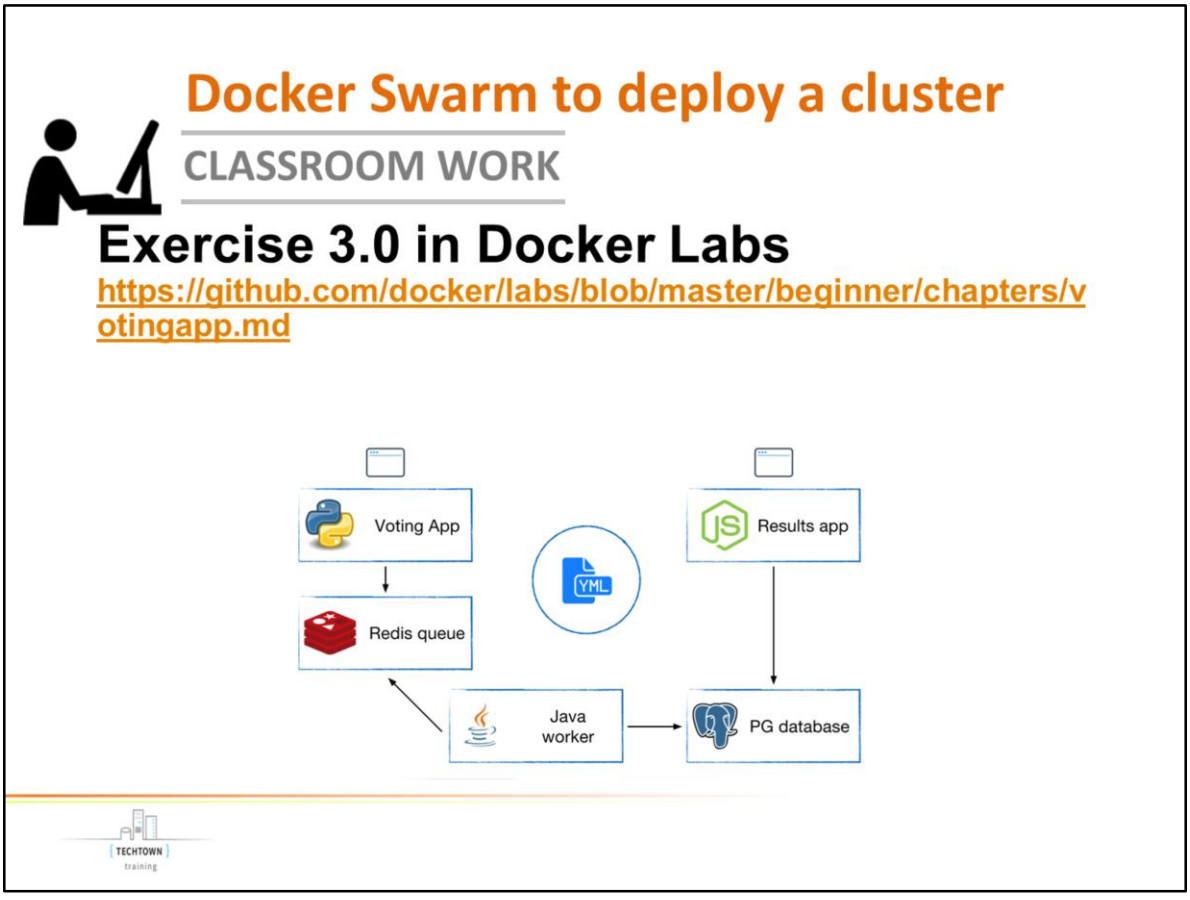
NOTES:

This exercise shows how docker-compose can be used to deploy a multi-container app in one step against a single docker daemon. Very useful for development, test, and CI environments.

Docker Swarm commands

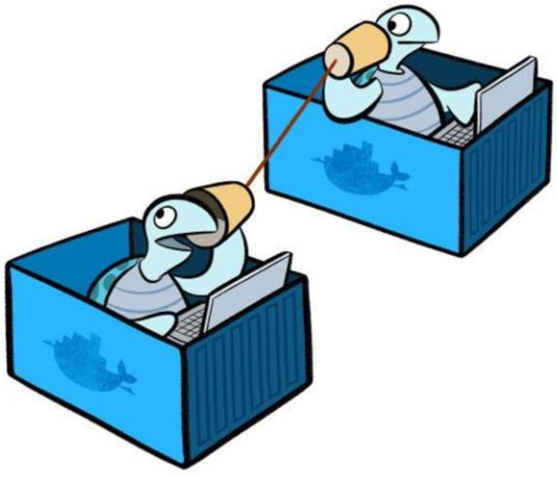
Command	Description
docker swarm init	Create a swarm
docker stack deploy	Deploy the swarm
docker stack services	Show swarm services
docker stack rm	Remove stack

NOTES:



Imagine what it would take to support something like American Idol's voting system… you'd probably define a similar architecture.

Docker Networking

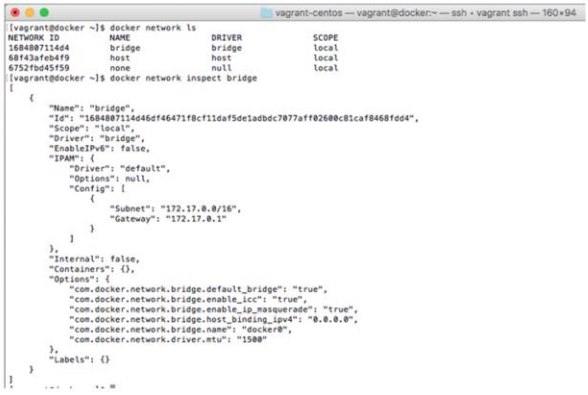


TECHTOWN training

NOTES:

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.

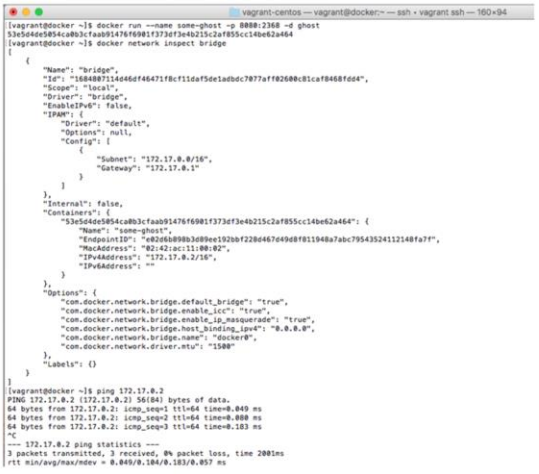


TECHTOWN training

$ docker network ls
$ docker network inspect bridge

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



TECHTOWN training

1. docker run --name some-ghost -p 8888:2368 -d ghost
2. docker inspect network bridge
3. ping 172.17.0.2

Note: The ‘some-ghost’ container is still exposed at <ip>:8888 through use of the –p 8888:2368 flag and argument

Read more about docker networking
https://docs.docker.com/engine/userguide/networking/

Docker network commands

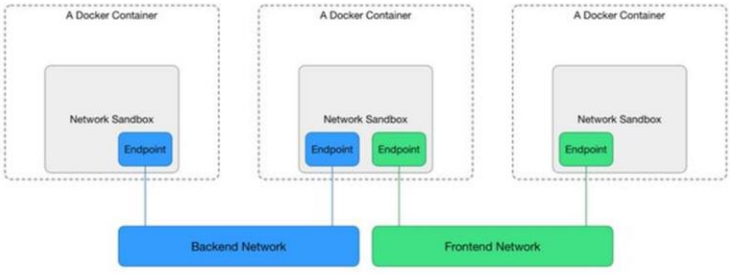
Command	Description
docker network create	Create a network
docker network ls	List networks
docker network inspect	Detailed information on network
docker network rm	Remove network
docker network disconnect	Disconnect container from network
docker network connect	Connect container to network

Commands follow the same general pattern as docker volume.

More documentation available at
https://docs.docker.com/engine/userguide/networking/

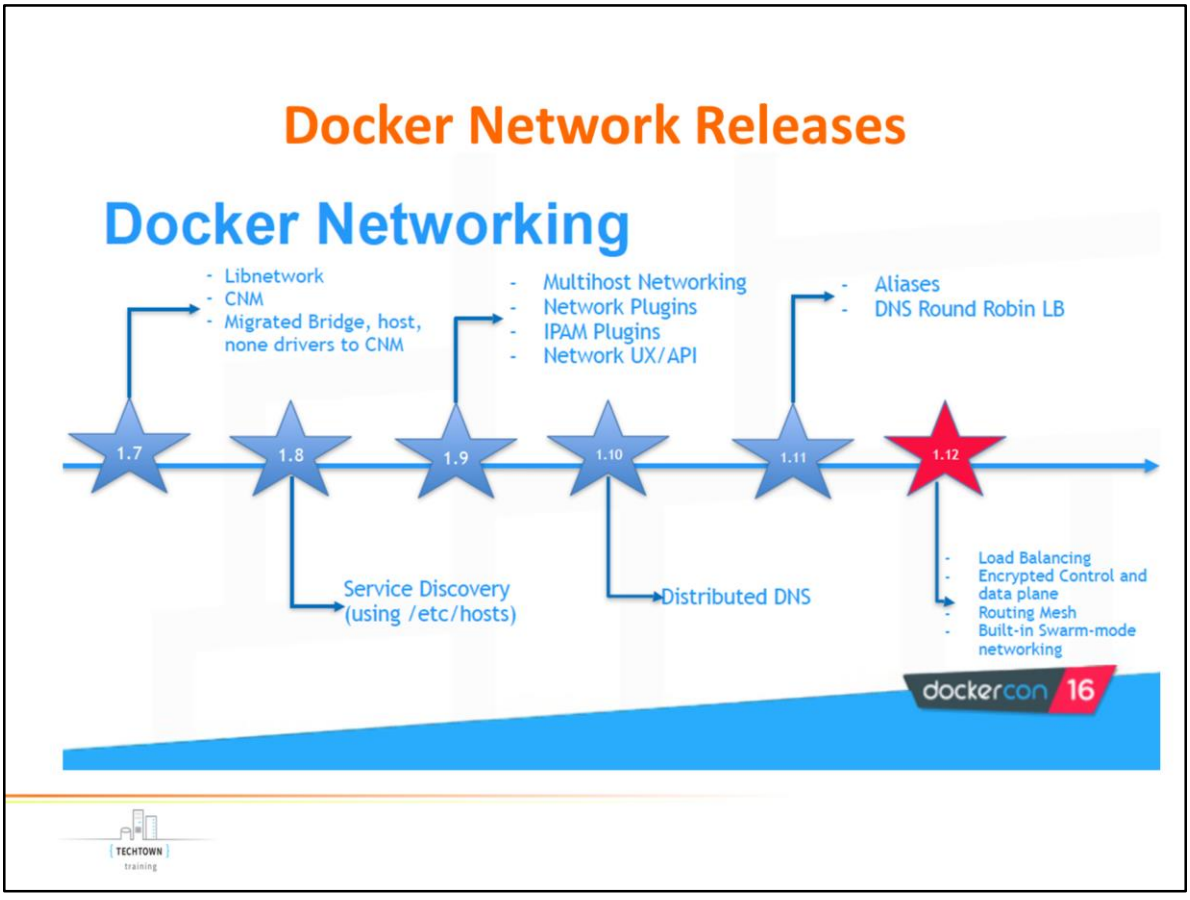
Isolated User Networks

- 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.



TECHTOWN training

NOTES:



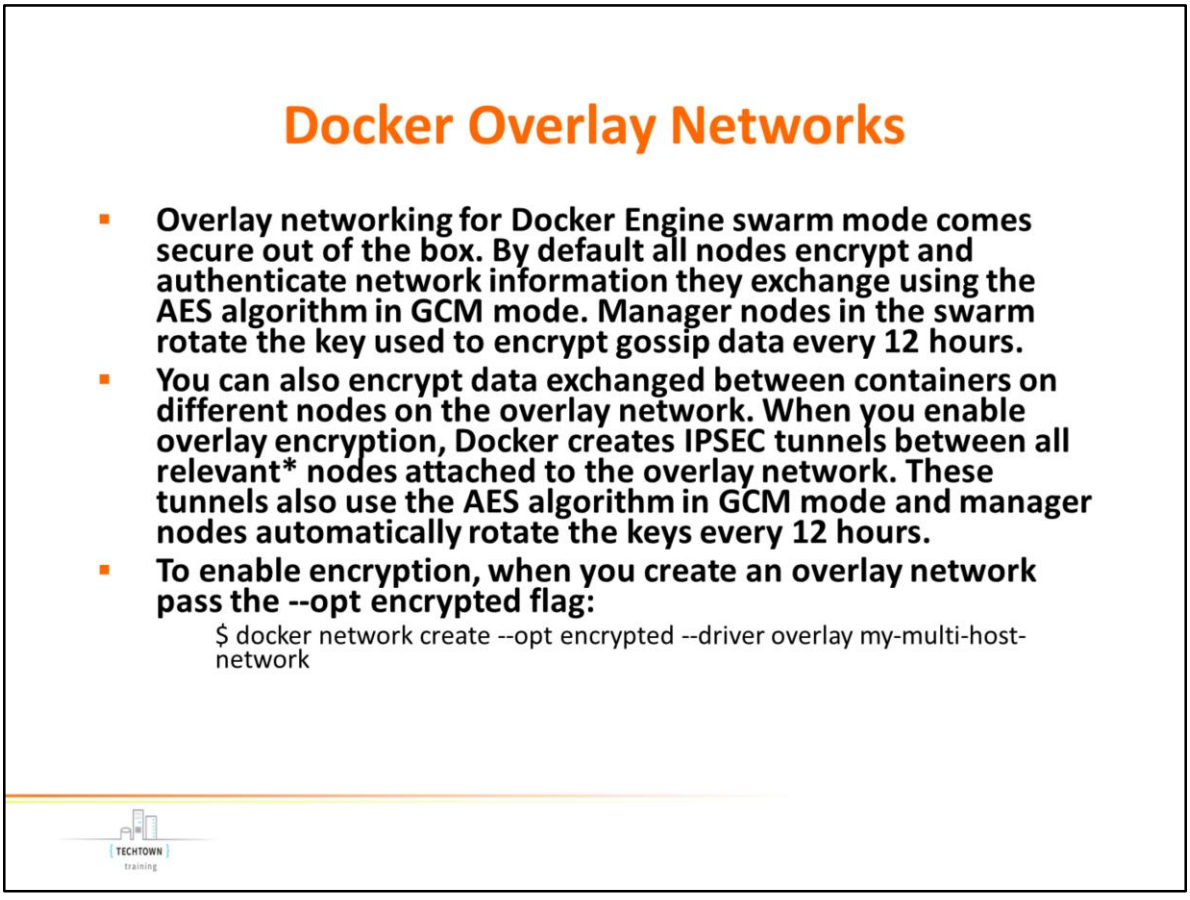
Reference:
http://www.slideshare.net/Docker/docker-networking-deep-dive

Docker has recently added quite a few networking features on top of their Libnetwork functionality. These use cases have traditionally been served by Docker networking plugins such as Weave, Flannel, Calico, and OpenVSwitch.

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

Reference:
https://github.com/docker/docker/releases/tag/v1.13.0



*The swarm makes the overlay network available only to nodes in the swarm that require it for a service. When you create a service that uses the overlay network, the manager node automatically extends the overlay network to nodes that run service tasks.

Reference:
https://docs.docker.com/engine/userguide/networking/overlay-security-model/

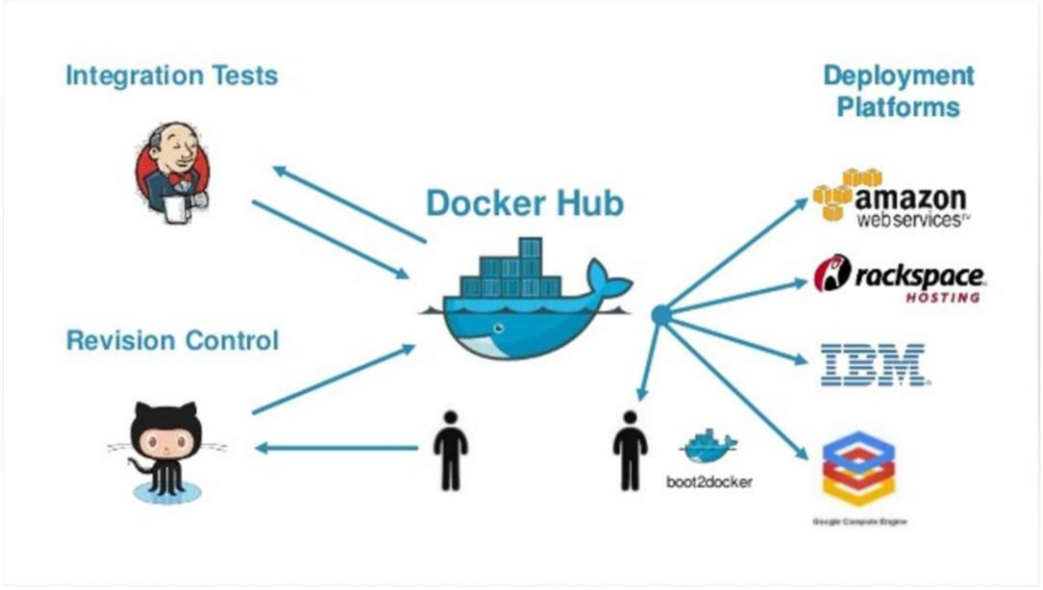
Docker Patterns

Part 6: DevOps Docker

TECHTOWN training

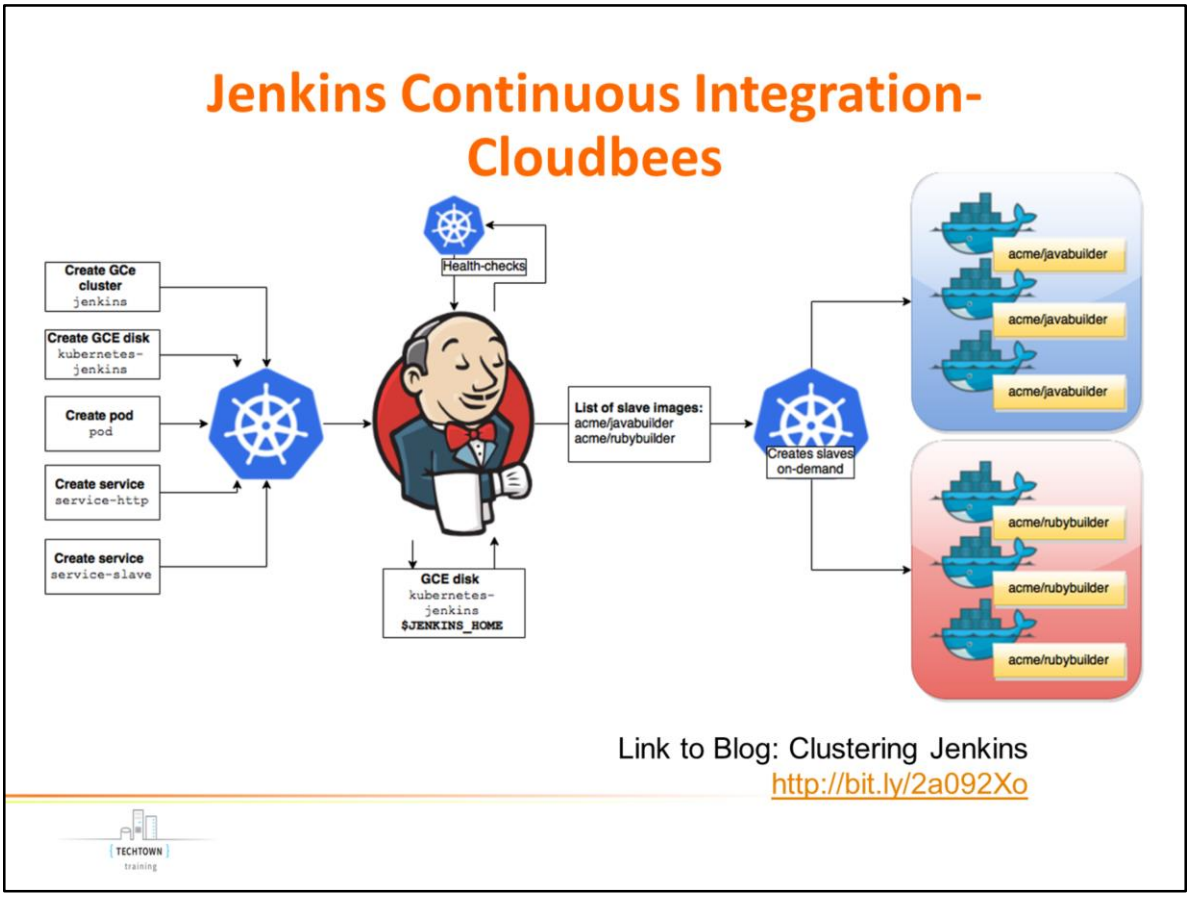
NOTES:

Docker Patterns

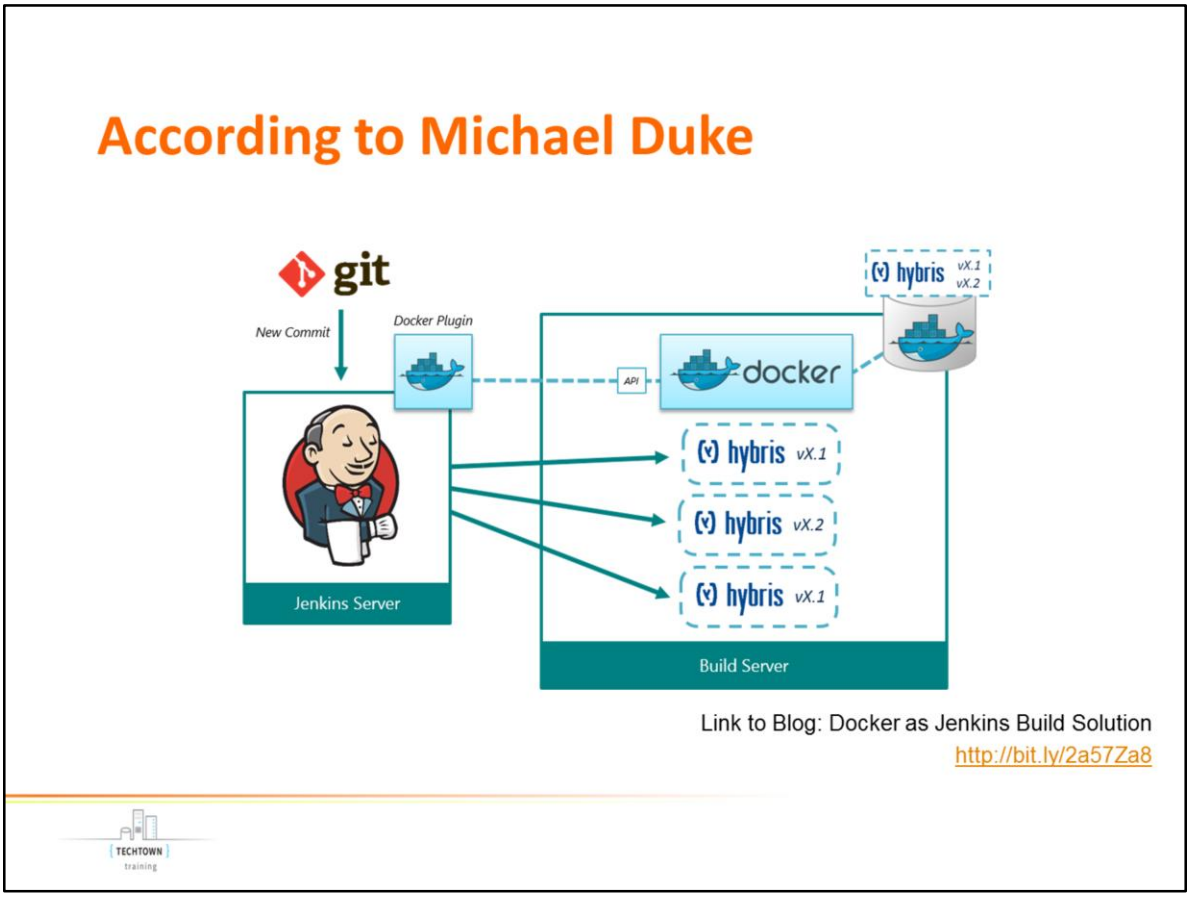


TECHTOWN training

NOTES:



The gist is that Jenkins master runs from a Docker image and is part of a Kubernetes Jenkins cluster. The master itself must have its own persistent storage where the $JENKINS_HOME with all of its credentials, plugins, and job/system configurations can be stored. This separation of master and $JENKINS_HOME into 2 locations allows the master to be fungible and therefore easily replaced should it go offline and need to be restarted by Kubernetes. The important "guts" that make a master unique all exist in the $JENKINS_HOME and can be mounted to the new master container on-demand. Kubernetes own load balancer then handles the re-routing of traffic from the dead container to the new one.



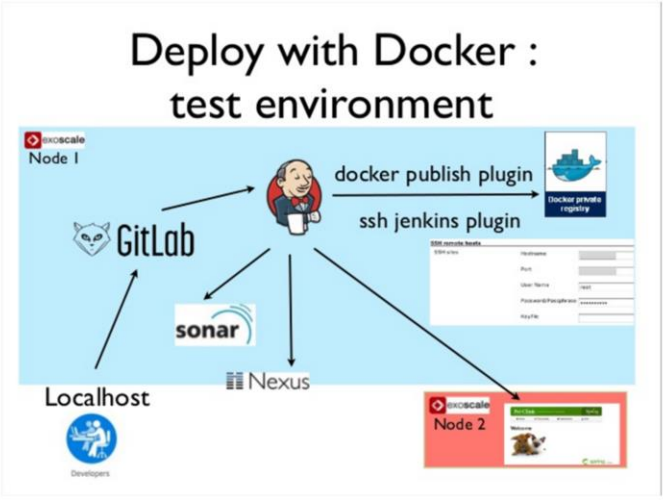
This blog post describes using the Docker Plugin for Jenkins to allow Jenkins to provision new containers which act as Jenkins slaves and compile code for a particular version of the popular Hybris eCommerce platform.

This solved several problems:

- Hybris can only compile one application at a time, so developers often ended up waiting in queues (up to 45+ minutes) to see the result of the code in the application.
- Hybris cannot cross-compile applications for different versions of the platform, so separate slaves are required for each version

Using Docker solves these problems by allowing a new lightweight application server to be spun up for each code commit… the container contains the correct version of Hybris for the branch the code comes from as well as a Jenkins slave to start the compilation.

According to Julia Mateo



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

TECHTOWN training

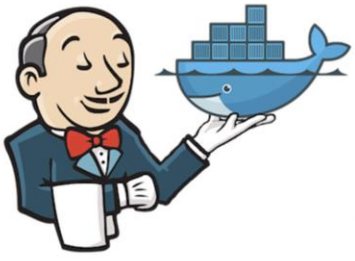
This presentation describes a workflow for true Continuous Deployment using Jenkins with the Docker plugin to publish images and the SSH plugin to pull images onto production environments. The code must first pass code quality tests in SonarQube (formerly Sonar) and artifacts are stored using Nexus. A Blue/Green / Canary routing model allows new code to be gradually moved to the bulk of users, and allows for user-selection of application version, if desired.

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.**

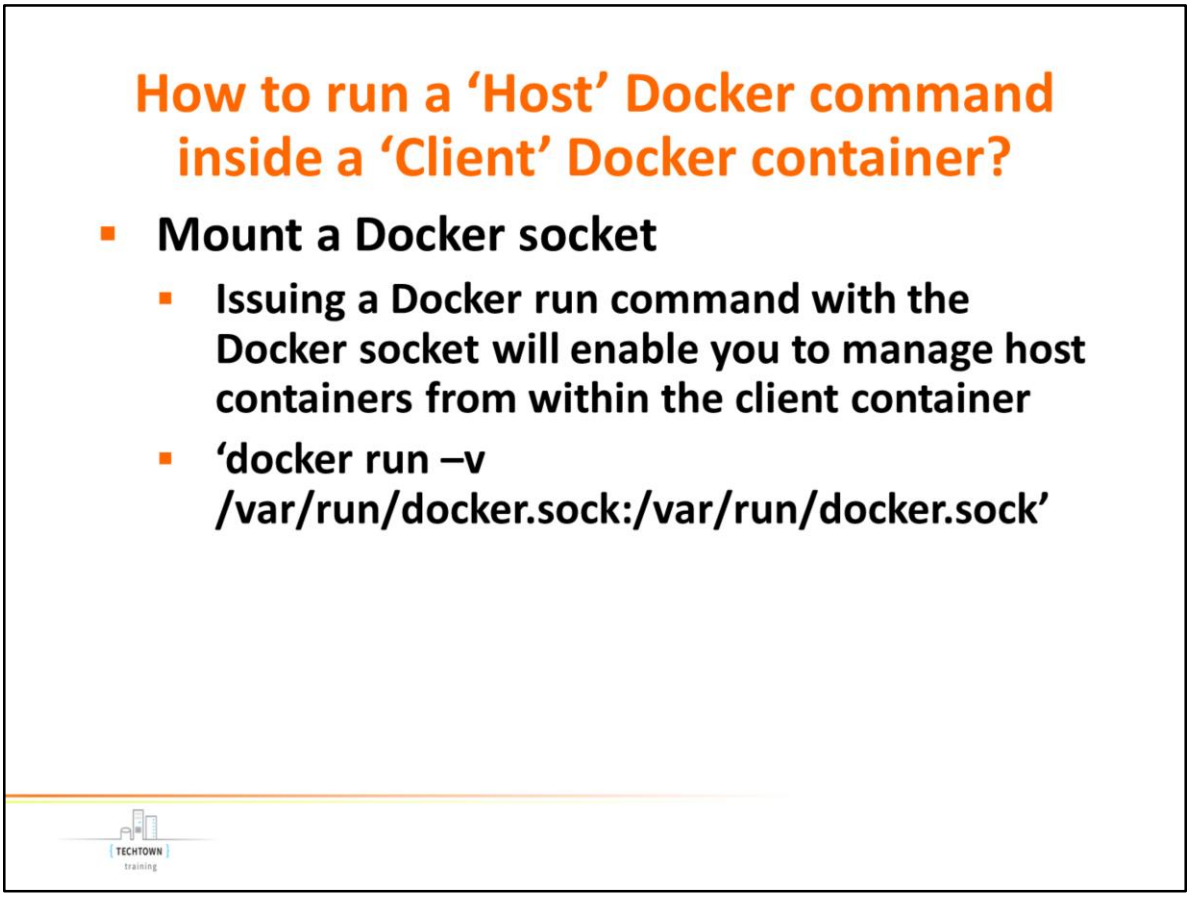
Spin up Jenkins in a Docker Container

Enable Jenkins to create other containers on the Docker Host computer



TECHTOWN training

It would be useful to orchestrate Docker containers with an automation tool. This allows us to create a variety of helpful automation including continuous integration pipelines for example.
In order to do so, we must figure out how to create containers from another application. Next we will walk through an example to do so.



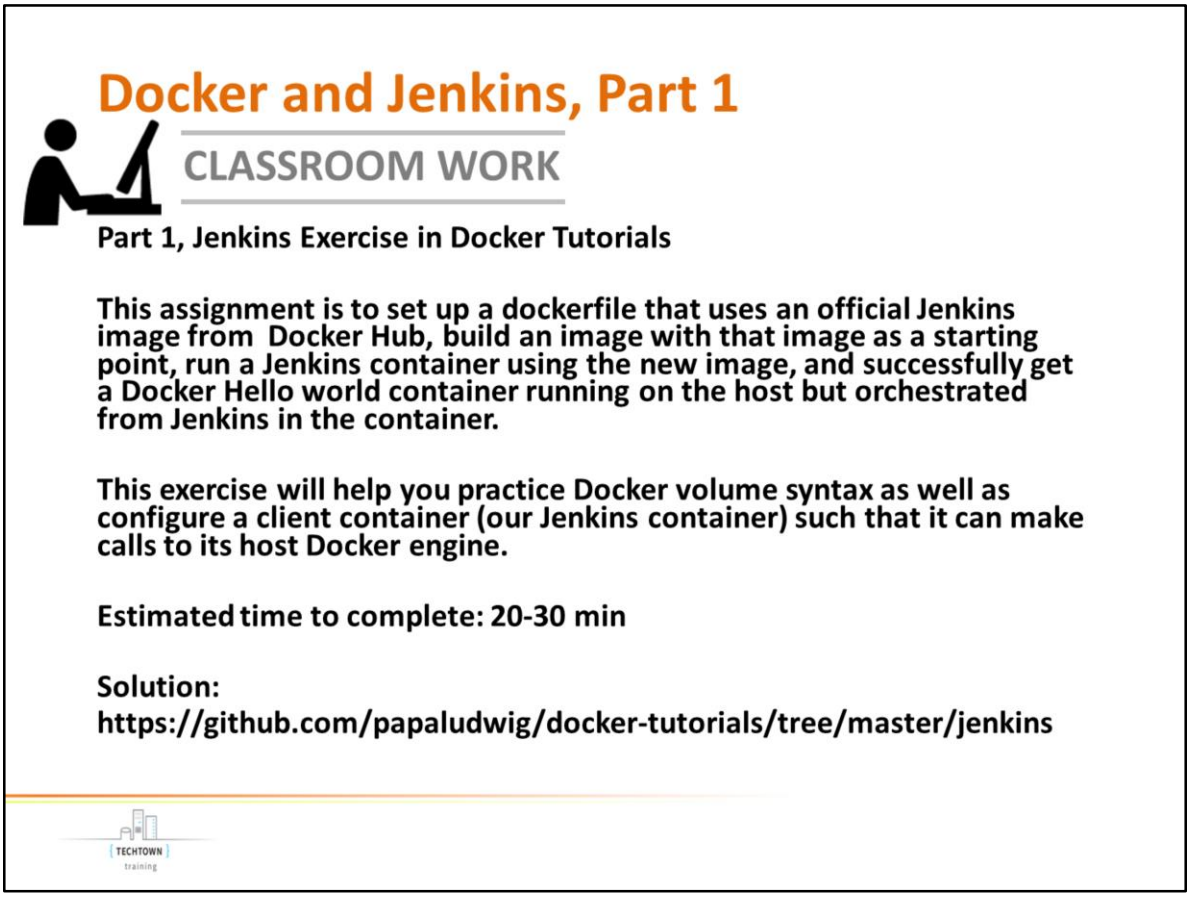
You can run the following command if you would like a fully functional example of this behavior.

docker run -it -v /var/run/docker.sock:/var/run/docker.sock ubuntu:latest sh -c "apt-get update ; apt-get install docker.io -y ; bash"

Once in this container, you can execute docker commands on the host such as 'docker ps' or 'docker run ubuntu /bin/**echo** 'Hello world''

Remember our syntax from docker volumes, this is mapping the host /var/run/docker.sock to the client container /var/run/docker.sock

Finally, consider the security aspects if doing this in a non-developer situation as the client container has access to all the other running containers. An attacker could access the socket and create another container to mount /etc/ and gain root access to the host, so it is wise to be careful when mounting docker.sock into a container.

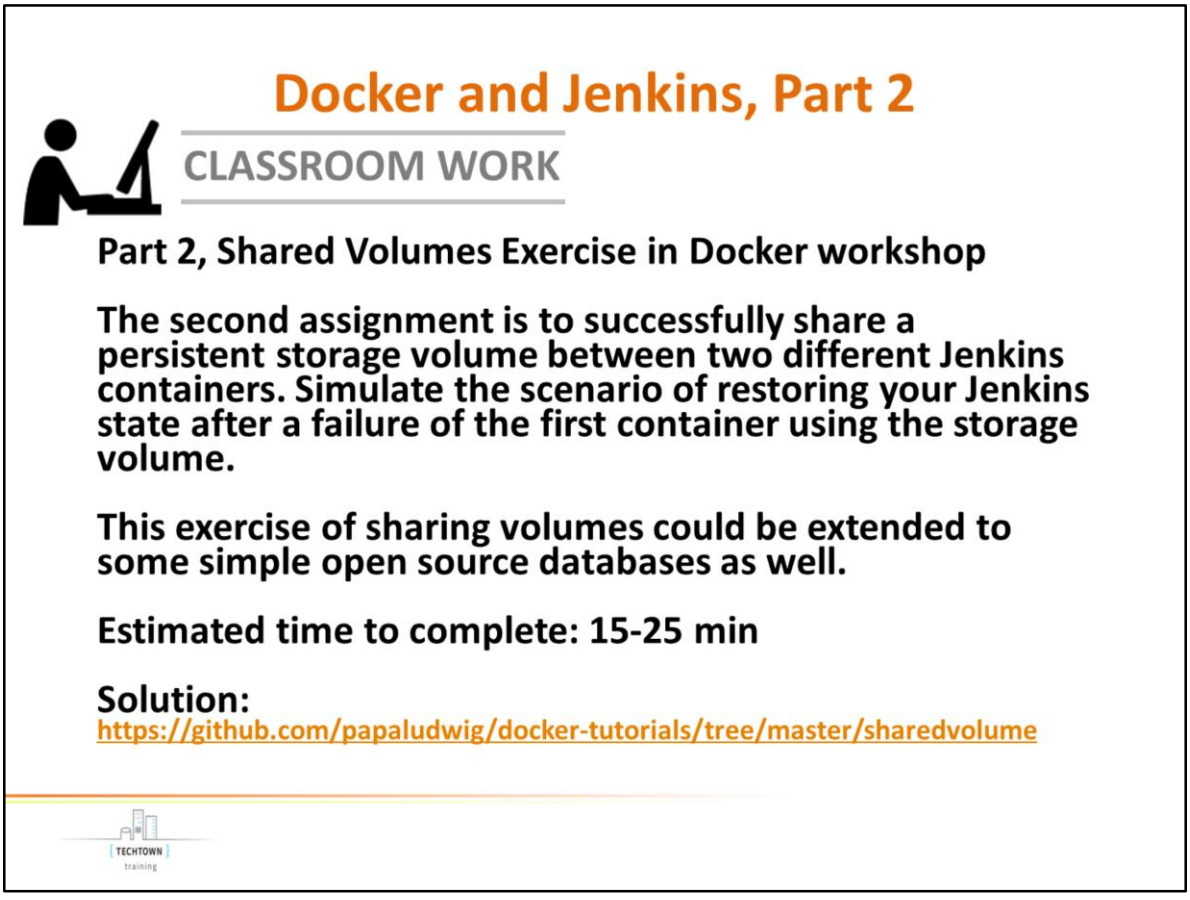


Jenkins Exercise in docker-tutorials is an example that will enable you to leverage Docker to easily setup and run Jenkins. Furthermore, by setting up a volume that is a shared docker.sock, you will give this container the capability to execute Docker commands on the host machine.

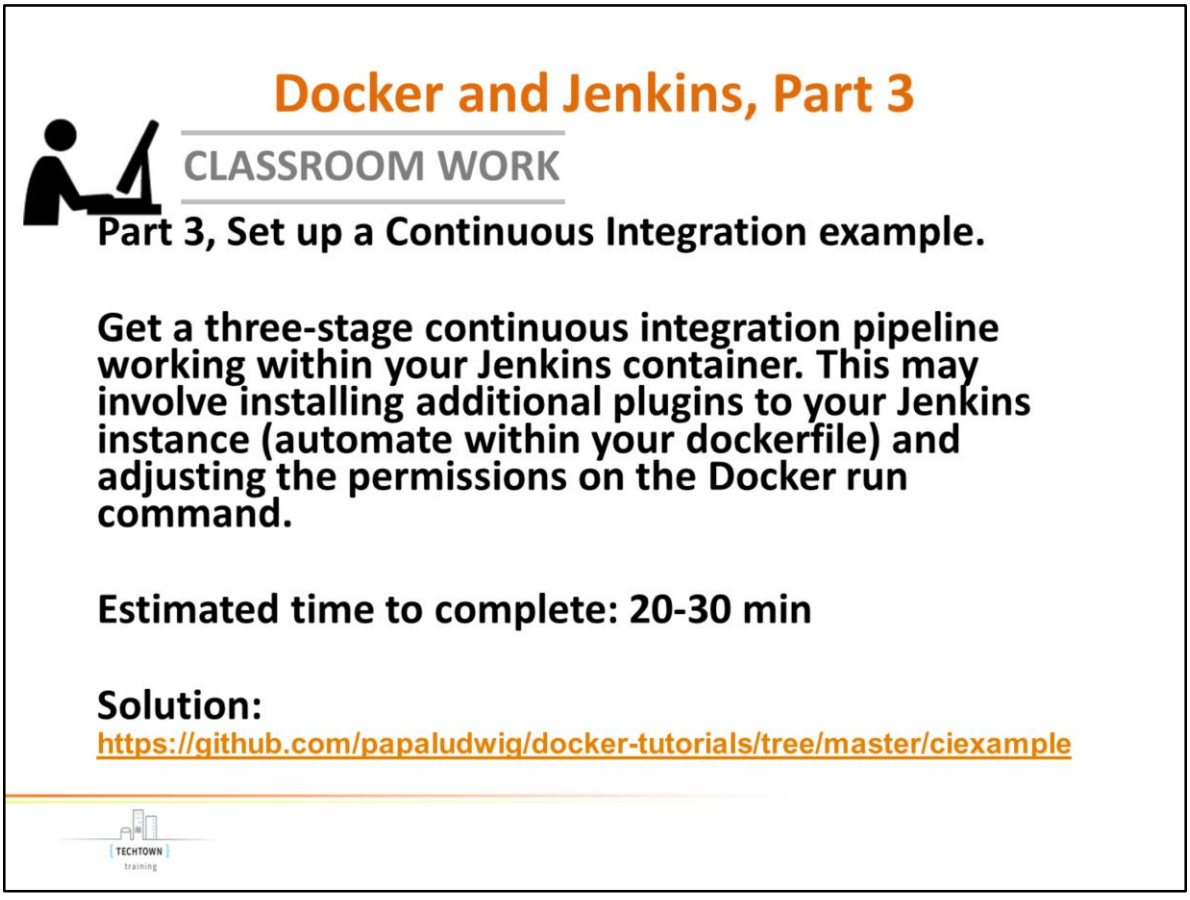
Sharing Persistent Volumes

```
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
```

NOTES:



This exercise expands on the ideas in the previous Jenkins Exercise setting up cross-container sharing of persistent volumes.



A Continuous Integration pipeline is the first step in many development projects.

NOTES:

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

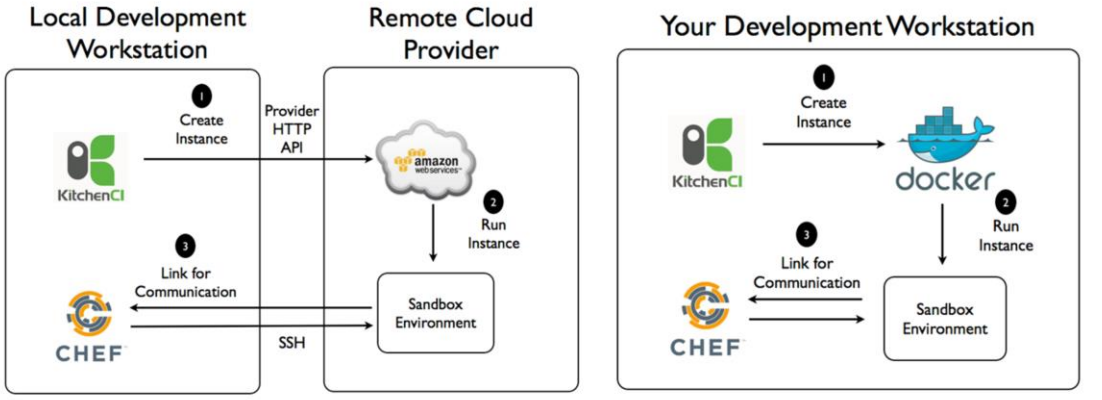
Estimated time to complete: 15-25 min

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

TECHTOWN training

NOTES:

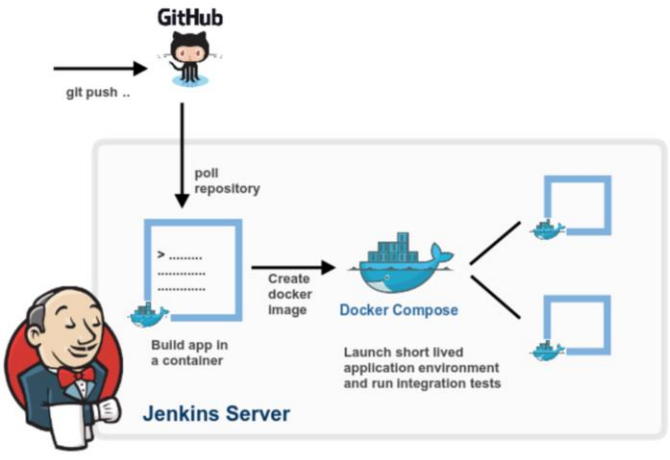
According to Mischa Taylor



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

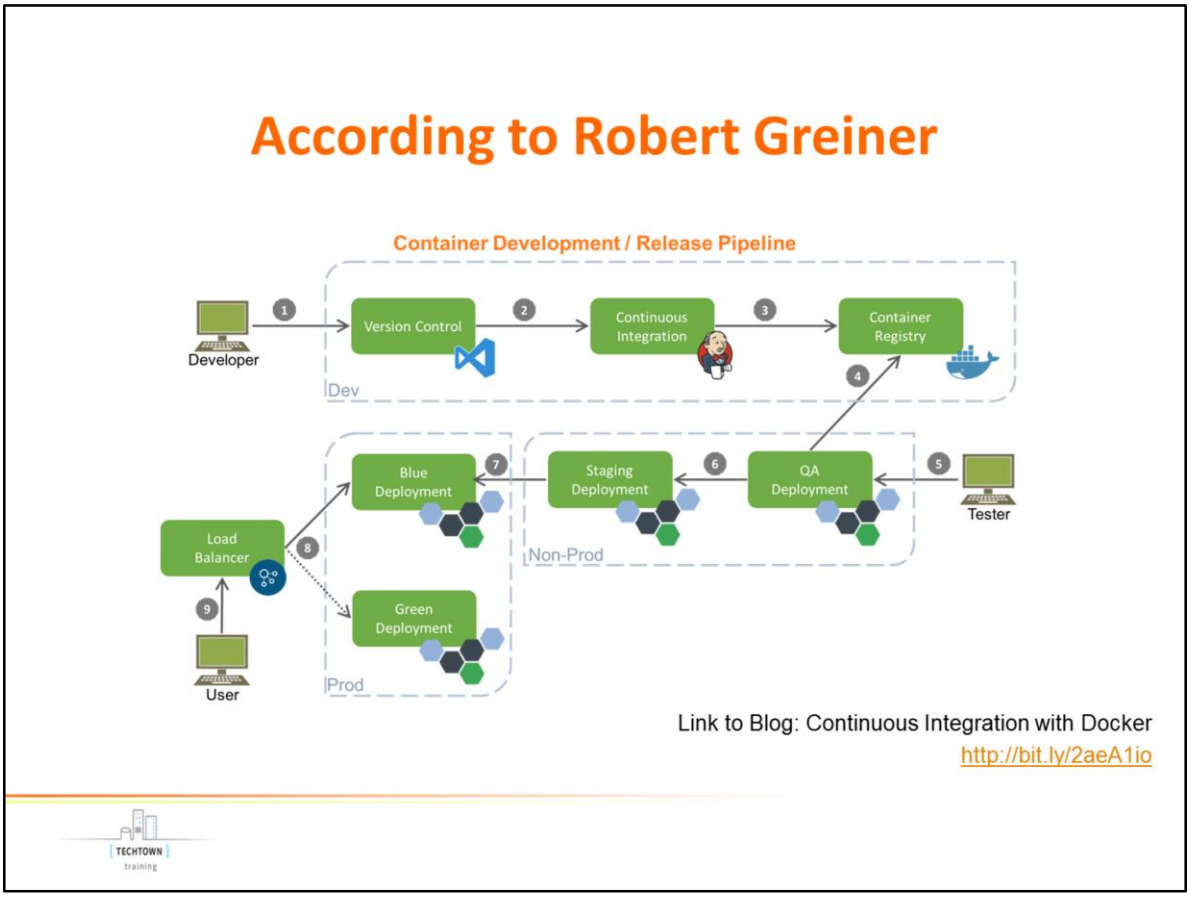
This blog post shows some ways to get the Chef SCM Test Kitchen test harness environment running against various deployment environments, including a Docker-based system.

According to Rancher



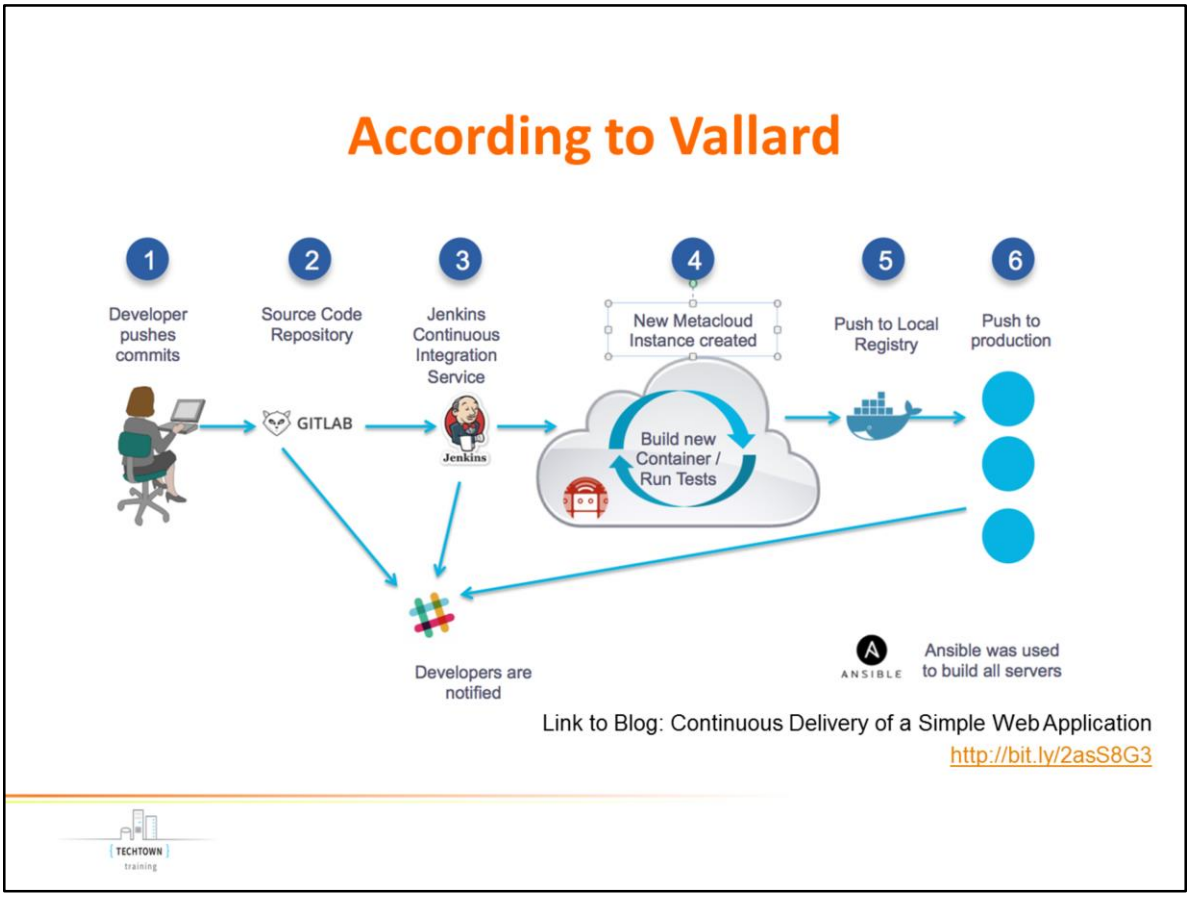
Link to Blog: Docker-based Build Pipelines
http://bit.ly/2aHtu0K

This blog post discusses using Jenkins and Docker to build a CI pipeline based around the git-flow branching model (with develop(ment) branch and master branch).



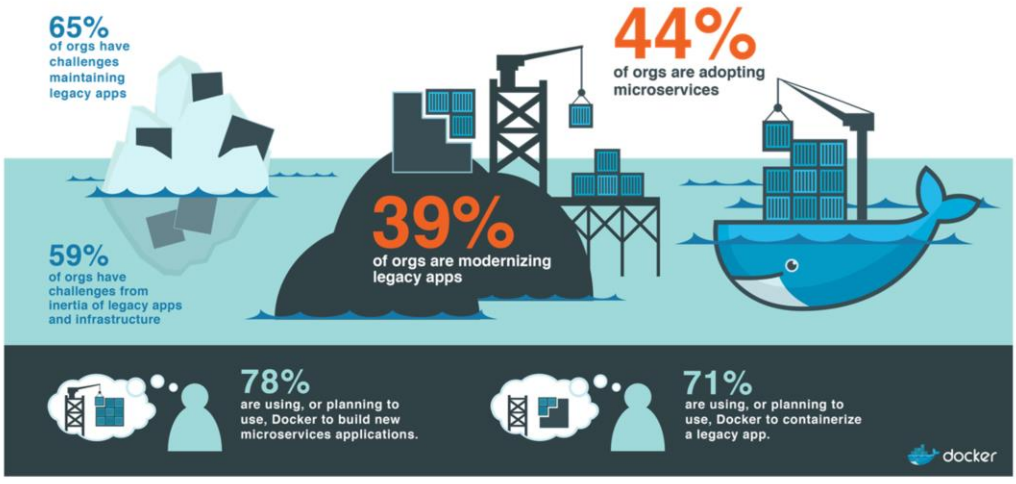
This blog post describes a CI/CD pipeline. The pipeline features container re-use throughout the cycle and allows a blue/green deployment.

1. Developer commits code to version control
2. Continuous Integration tool (Jenkins) triggers on check-in, tests the changes, and builds a Docker Image
3. Jenkins submits new version of Image to Registry
4. Docker Container is built and executed on QA server for testing
5. Testing begins in Non-Prod / Prod environments
6. Container is promoted to Staging server after successful tests
7. Container is promoted to “Blue” Production environment (non-live) after successful tests
8. Container is promoted to “Green” Production release, after successful tests, and is live
9. User access new functionality



This blog post is structured as a tutorial and walks you through building a CI pipeline to deliver a web app. It features Jenkins CI and tight Slack integration for high visibility.

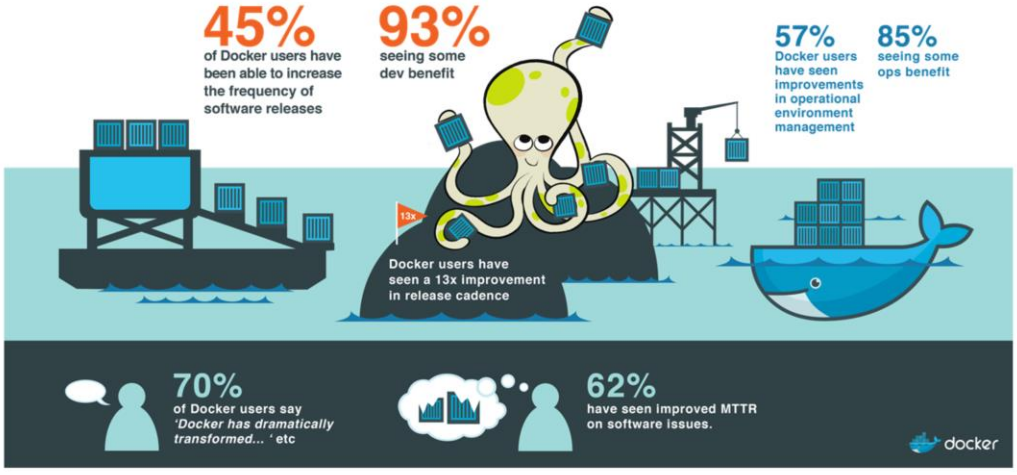
Docker Adoption & Usage Data



TECHTOWN training

NOTES:

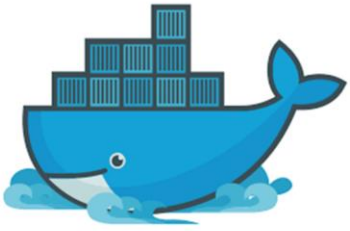
Docker Adoption & Usage Data



TECHTOWN training

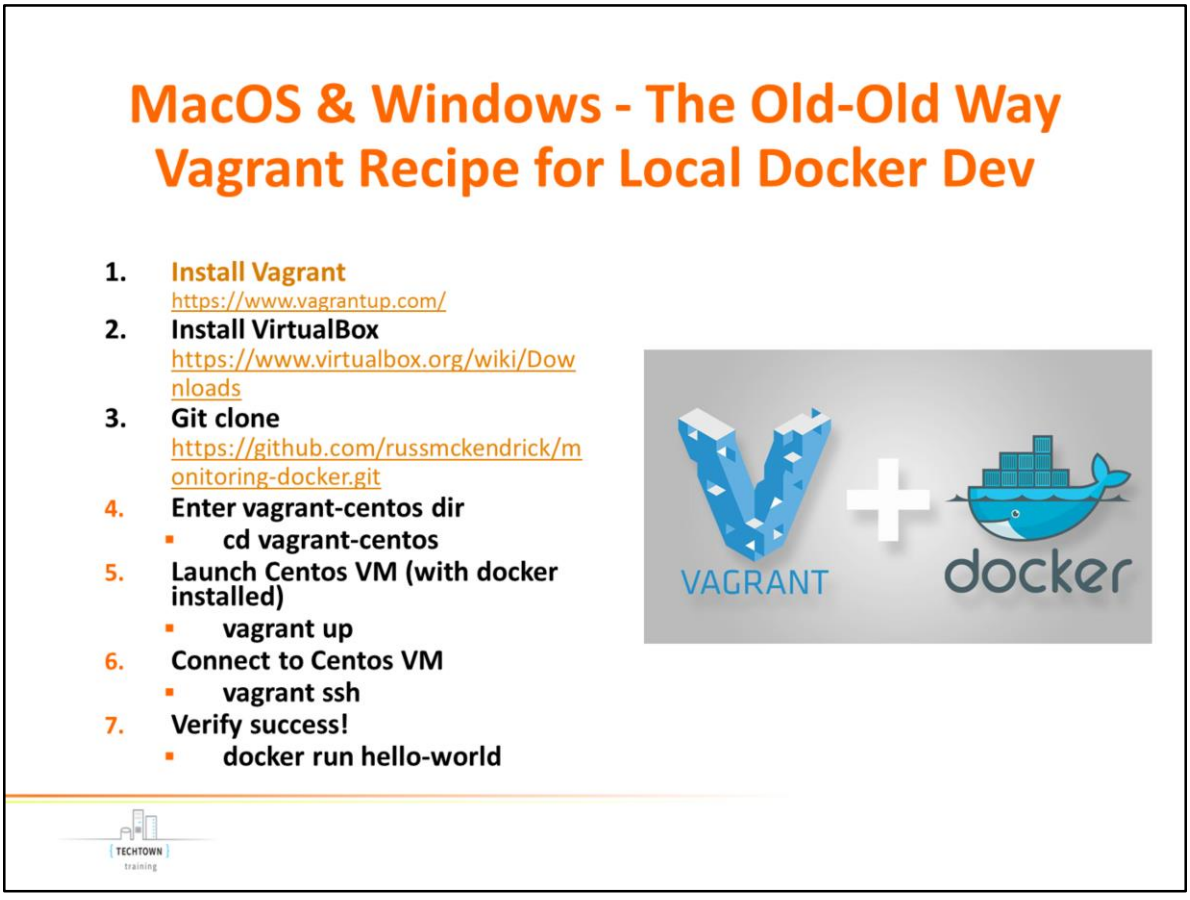
NOTES:

Local Docker Development



TECHTOWN training

NOTES:



The vagrant files provided in this project:

https://github.com/russmckendrick/monitoring-docker

are excellent.

https://github.com/russmckendrick/monitoring-docker/blob/master/vagrant-centos/Vagrantfile

In addition, they give the VM a static IP of 192.168.33.10. This makes it much more convenient to access your containers from your host PC through the VM. For example, if you ran 'docker run --name some-ghost -p 8080:2368 -d ghost' on this VM, you could then access the container ghost program at: 192.168.33.10:8080 in the internet browser of your host MacOS or Windows machine. This makes local development much easier and a more pleasant experience if you don't have a dedicated Linux machine at all times.

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**

TECHTOWN training

NOTES:

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

TECHTOWN training

NOTES:

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-access-key AKI******* \ --amazonec2-secret-key 8T93C******* \
  aws-sandbox
- Local and Cloud drivers available

TECHTOWN training

When you install Docker Machine, you get a set of drivers for various cloud providers (like Amazon Web Services, Digital Ocean, or Microsoft Azure) and local providers (like Oracle VirtualBox, VMWare Fusion, or Microsoft Hyper-V).

https://docs.docker.com/machine/get-started-cloud/

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")

NOTES:

Docker Security

Part 7: DevOps Docker

NOTES:

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

NOTES:

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

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

TECHTOWN training

NOTES:

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
```

TECHTOWN training

1.1 – Creating a separate partition for containers – mapping /var/lib/docker to a separate partition
2.1 – Restricting network traffic – pass icc=false to the Docker daemon startup process
2.5 – Do not use the aufs storage driver – Use the appropriate storage driver for the OS of the Docker host
4.1 - Create a user for the container – Run containers as non-root users
5.7 – Do not run ssh within containers – Use tools provided by Docker to access containers including docker exec, etc

If time allows 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?
- Which, if any, do you disagree with?

(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.)

http://training.play-with-docker.com/security-seccomp/

NOTES:

Use Docker to Set Up an ELK Stack

CLASSROOM WORK

Use Docker to (quickly!) set up an ELK (Elasticsearch, Logstash, Kibana) stack which can be used for monitoring (and alerting if you add Elastalert) based on logfile activity.

https://github.com/docker/dceu_tutorials/blob/master/10-logging-and-monitoring.md

TECHTOWN training

NOTES:

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

https://groups.google.com/forum/#!forum/docker-dev

NOTES:

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

NOTES:

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

https://blog.quay.io/security scanningbeta/

https://coreos.com/blog/vulnerabilityanalysisfor containers/

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

NOTES:

Swarm Security Basics

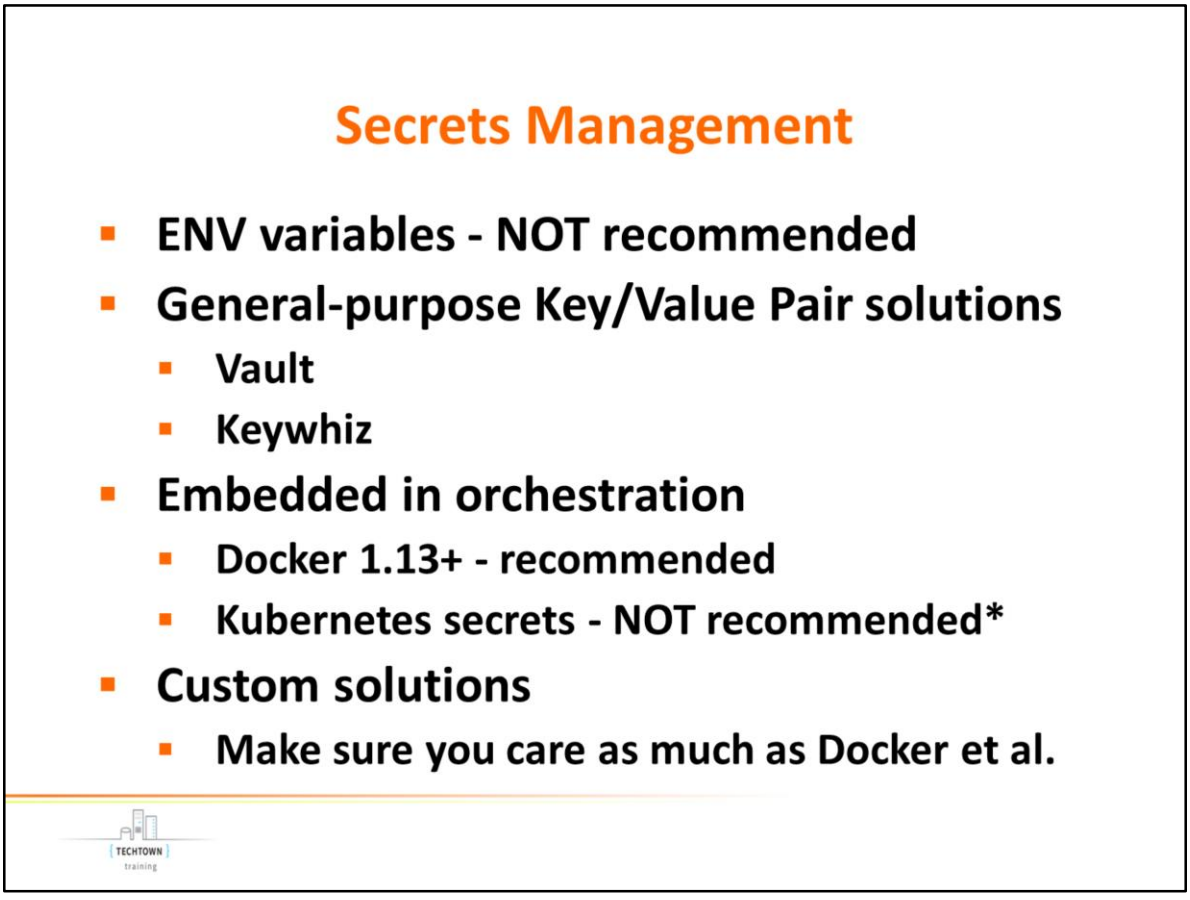
CLASSROOM WORK

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

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

TECHTOWN training

NOTES:



- * As of 2016-11 https://www.conjur.com/blog/security-holes-persist-in-kubernetes-kubecon-2016-recap
  - Secrets are stored in plaintext on disks, etc.

Swarm Secrets Management

CLASSROOM WORK

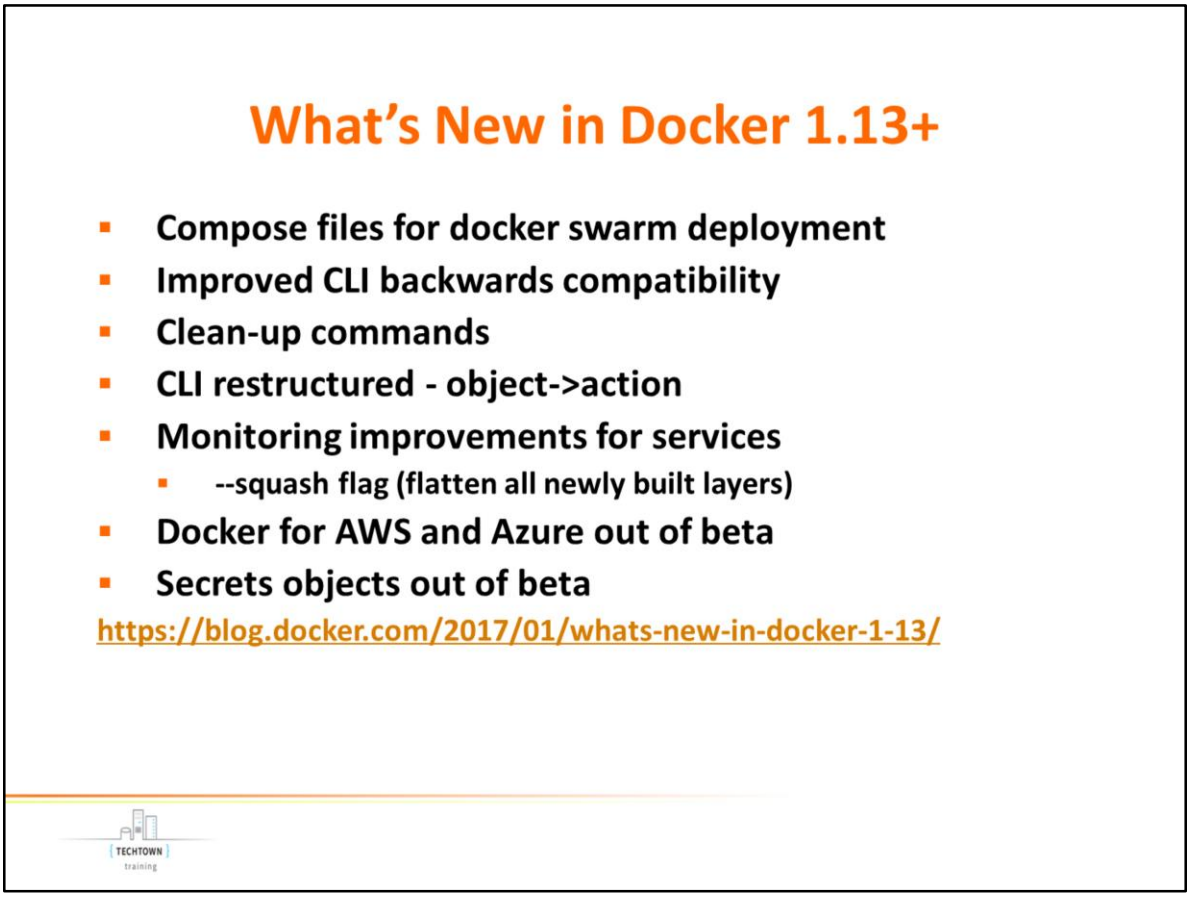
- In this exercise, we'll use Docker's secrets objects to share a secret across nodes in a swarm.

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

https://github.com/docker/labs/tree/master/security/secrets

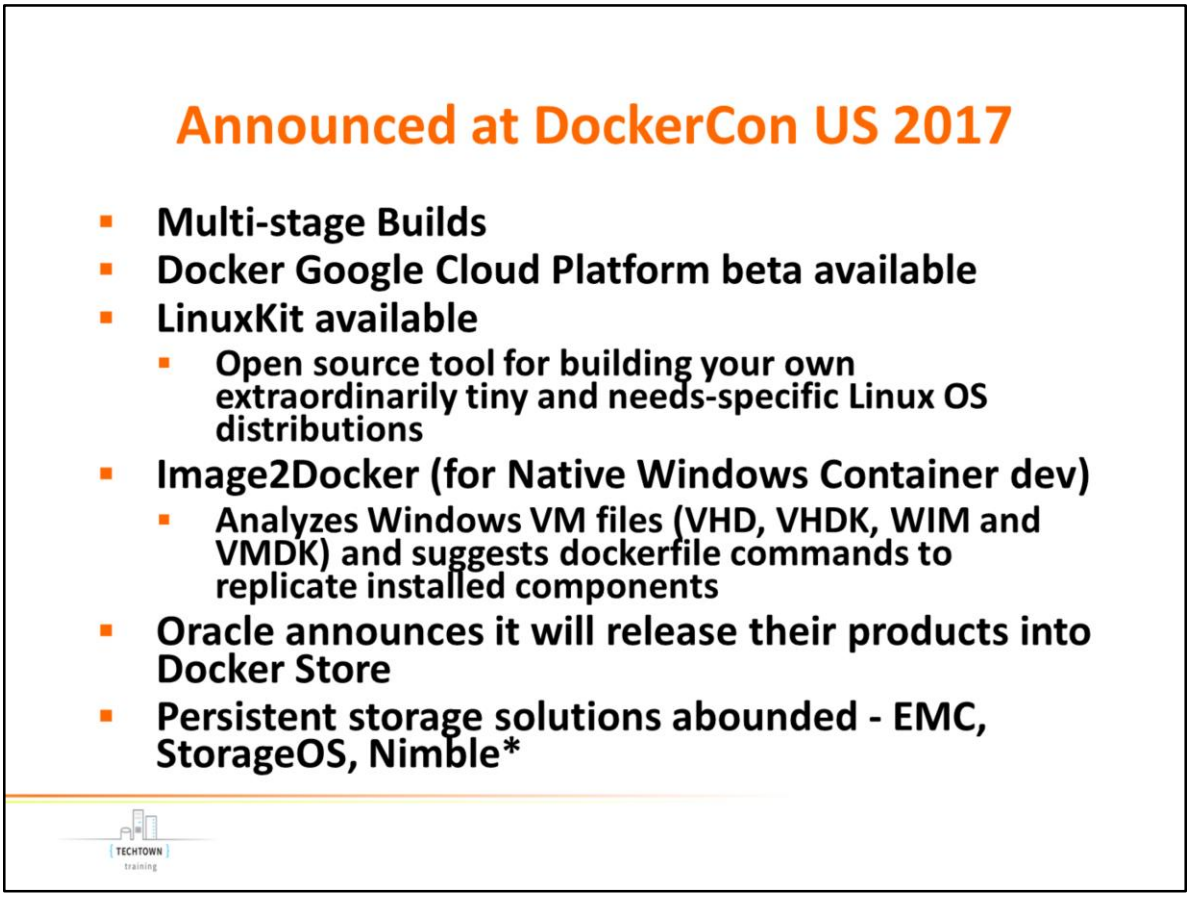
TECHTOWN training

NOTES:



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

NOTES:



https://blog.docker.com/2017/04/introducing-linuxkit-container-os-toolkit/

Storage solutions as third party offerings becoming so popular might very well indicate that Docker will fold a storage solution into Docker Engine…. this has happened in the past.

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

NOTES:

Thank you!
...and see you at the next DockerCon!

TECHTOWN training

NOTES:

Course Evaluation

Please take a moment to complete your course evaluation which gives you immediate access to your certificate of completion.

All responses are confidential even though an email address is requested.

- **Go to http://www.metricsthatmatter.com/ASPE**
- **From the drop down menu choose Docker Containerization Boot Camp, site, instructor name, class start date**
- **Fill out and submit the form**
- **It is available for 10 calendar days**

Thank You!

NOTES: