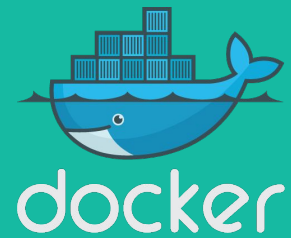
Docker 101 Workshop

Eric Smalling - Solution Architect, Docker Inc.

@ericsmalling



Who Am I?

- Eric Smalling
  - Solution Architect
    Docker Customer Success Team

- ~25 years in software development, architecture, version control admin, etc…

- ~10 years in build & test automation

- Docker user since pre-1.0 days

Agenda

Section 1: **(One Hour)**

What is Docker / What is Docker Not

Basic Docker Commands

Dockerfiles

Hands On Exercises

Break (15 minutes)

Section 2: (30 minutes)

Anatomy of a Docker image

Docker volumes

Volume use cases

Hands On Exercises

Before we get started

Make sure you can get to these sites:

https://github.com/ericsmalling/docker101-linux

http://labs.play-with-docker.com/

Section 1:

What is Docker
Basic Docker Commands
Dockerfiles

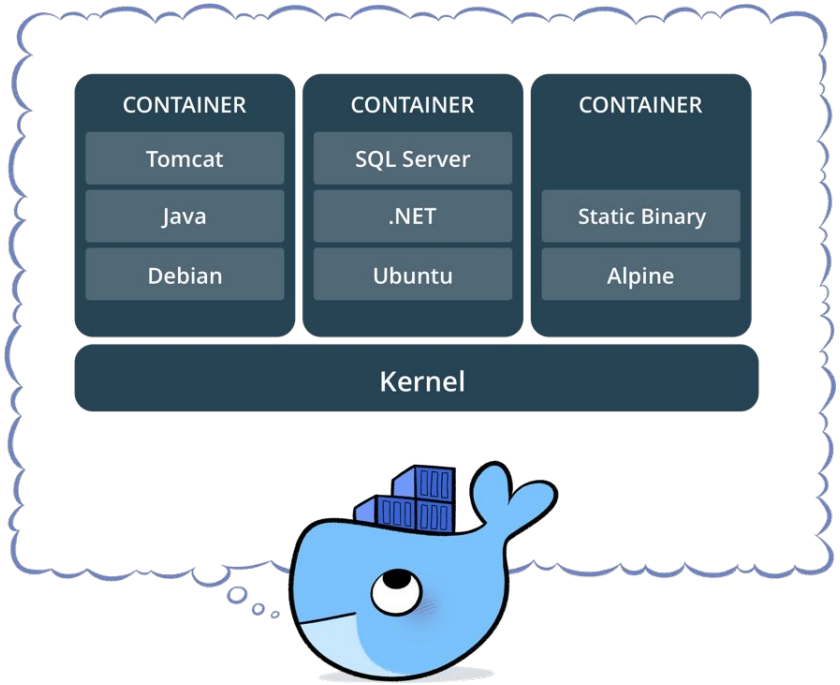
Docker containers are NOT VMs

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

VMs

Containers

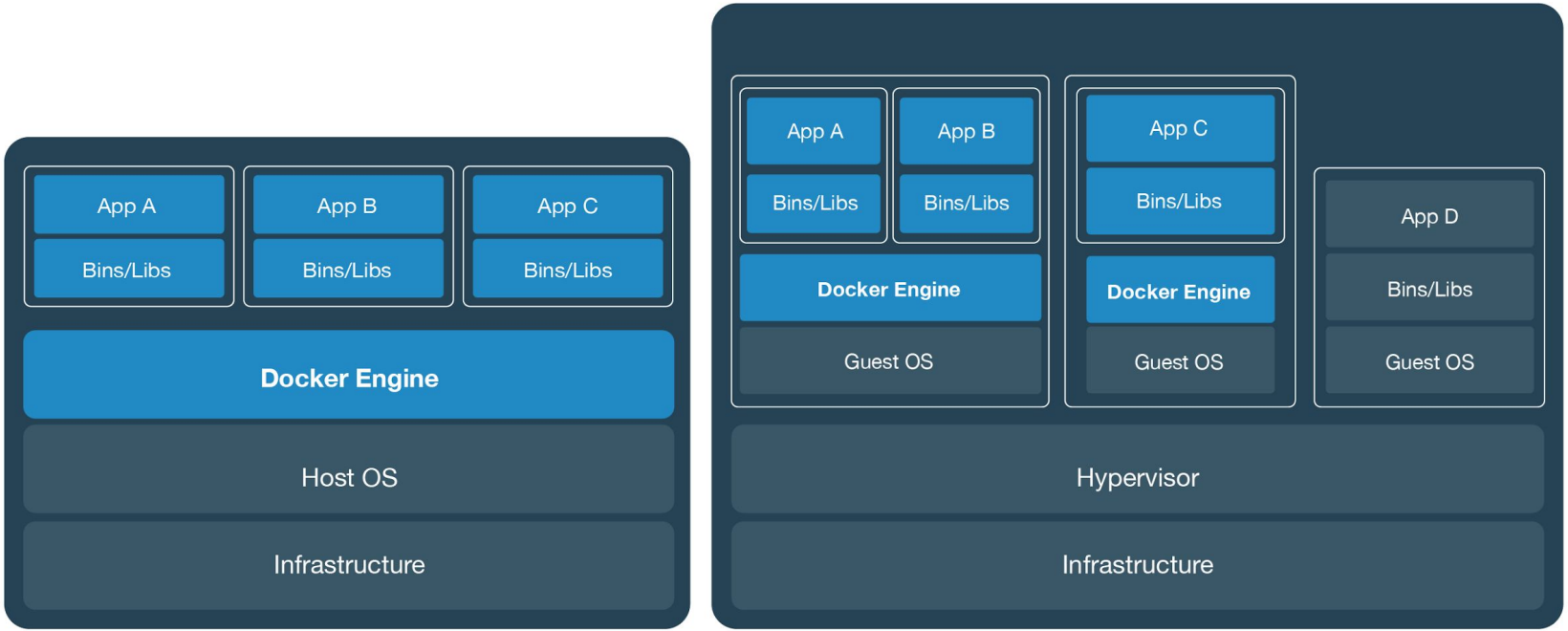
What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Containers native to Windows Server 2016

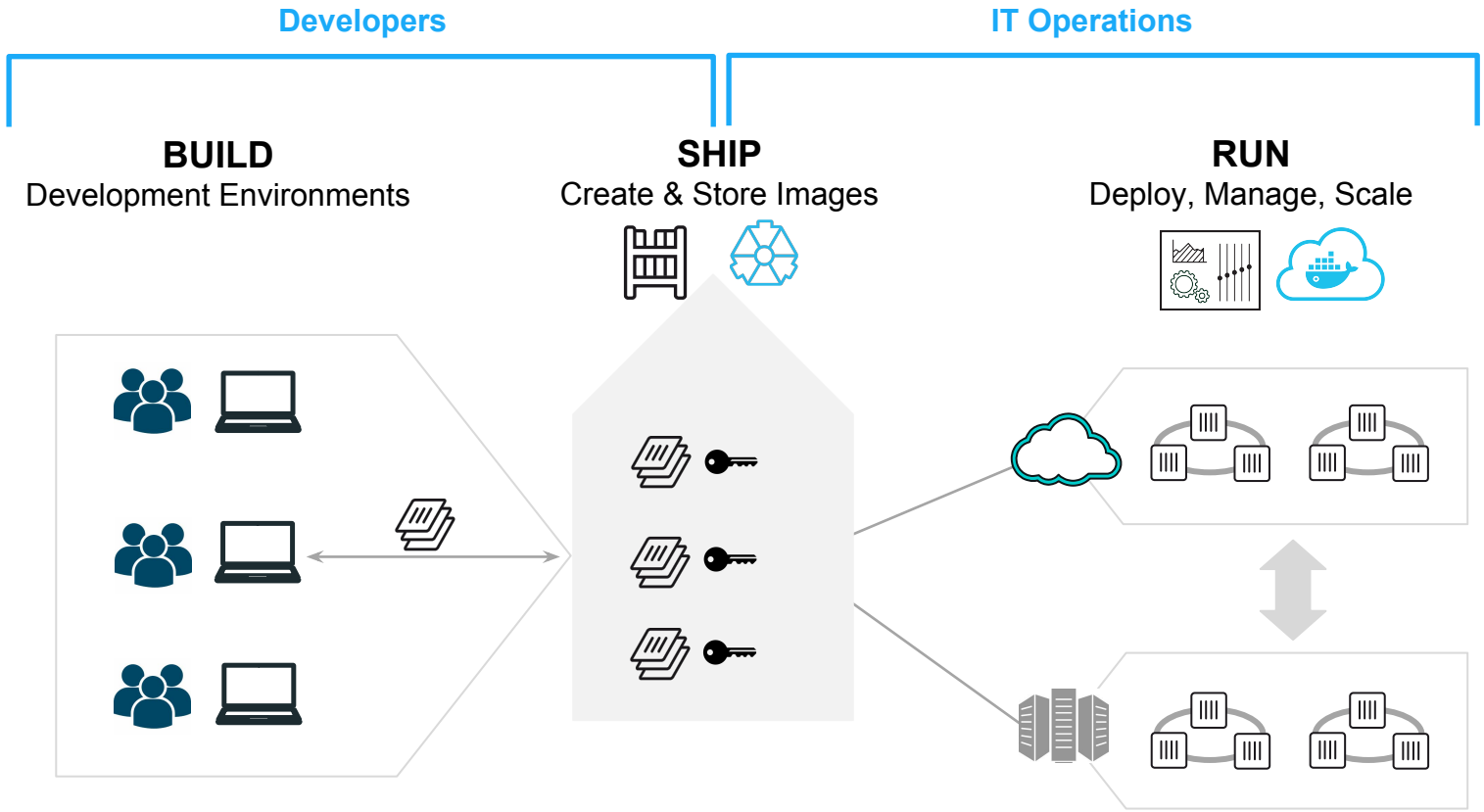
docker

They’re different, not mutually exclusive



docker

Using Docker: Build, Ship, Run Workflow



docker

Some Docker vocabulary

Docker Trusted Registry: Storage and distribution service for your images

Docker Universal Control Plane: Web-based management CaaS platform

Docker Image: The basis of a Docker container. Represents a full application

Docker Container: The standard unit in which the application service resides and executes

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

Service: An application (or part of an application) that provides a specific function (catalog lookup, web front end, payment processing)

Stack: A way of representing multi-service applications. Made up of 1-n services

Basic Docker Commands

```
$ docker image pull mikegcoleman/catweb:latest

$ docker image ls
$ docker container run –d –p 5000:5000 –-name catweb mikegcoleman/catweb:latest

$ docker container ps
$ docker container stop catweb (or <container id>)

$ docker container rm catweb (or <container id>)

$ docker image rm mikegcoleman/catweb:latest (or <image id>)

$ docker build –t mikegcoleman/catweb:2.0 .

$ docker image push mikegcoleman/catweb:2.0
```

Dockerfile – Linux Example

```
1  # our base image
2  FROM alpine:latest
3
4  # Install python and pip
5  RUN apk add --update py-pip
6
7  # upgrade pip
8  RUN pip install --upgrade pip
9
10 # install Python modules needed by the Python app
11 COPY requirements.txt /usr/src/app/
12 RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
13
14 # copy files required for the app to run
15 COPY app.py /usr/src/app/
16 COPY templates/index.html /usr/src/app/templates/
17
18 # tell the port number the container should expose
19 EXPOSE 5000
20
21 # run the application
22 CMD ["python", "/usr/src/app/app.py"]
```

- Instructions on how to build a Docker image
- Looks very similar to “native” commands
- Important to optimize your Dockerfile

Dockerfile – Windows Example

19 lines (15 sloc) | 832 Bytes — Raw | Blame | History

```
1  FROM microsoft/windowsservercore
2
3  ENV NPM_CONFIG_LOGLEVEL info
4  ENV NODE_VERSION 6.5.0
5  ENV NODE_SHA256 0c0962800916c7104ce6643302b2592172183d76e34997823be3978b5ee34cf2
6
7  RUN powershell -Command \
8      $ErrorActionPreference = 'Stop' ; \
9      (New-Object System.Net.WebClient).DownloadFile('https://nodejs.org/dist/v%NODE_VERSION%/node-v%NODE_VERSION%-win-x64.zip',
10     if ((Get-FileHash node.zip -Algorithm sha256).Hash -ne $env:NODE_SHA256) {exit 1} ; \
11     Expand-Archive node.zip -DestinationPath C:\ ; \
12     Rename-Item 'C:\node-v%NODE_VERSION%-win-x64' 'C:\nodejs' ; \
13     New-Item '%APPDATA%\npm' ; \
14     $env:PATH = 'C:\nodejs;%APPDATA%\npm;' + $env:PATH ; \
15     [Environment]::SetEnvironmentVariable('PATH', $env:PATH, [EnvironmentVariableTarget]::Machine) ; \
16     Remove-Item -Path node.zip
17
18 CMD [ "node.exe" ]
```

Hands On Exercises

Running Basic Docker Images
Building a Website with A Dockerfile

http://labs.play-with-docker.com
https://github.com/ericsmalling/docker101-linux

Section 2:

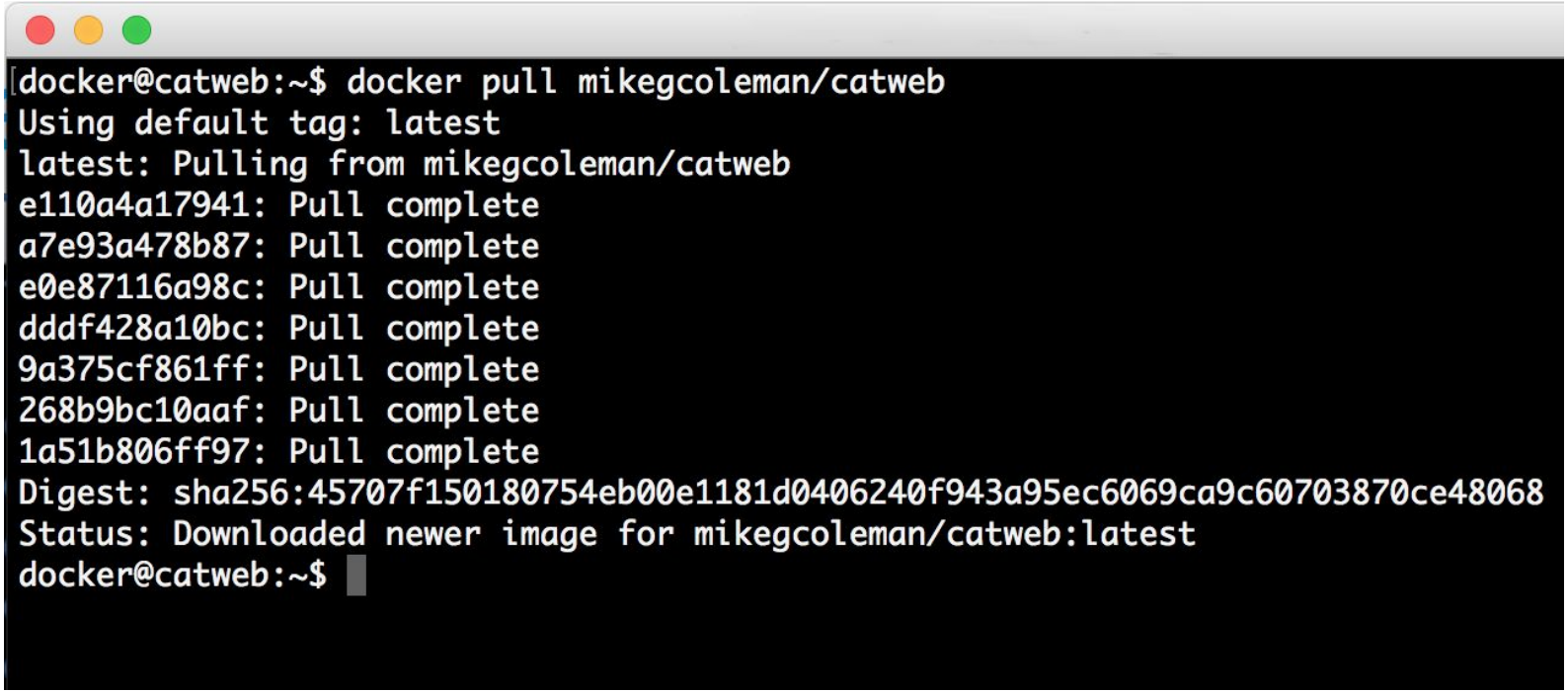
Anatomy of a Docker Container
Docker Volumes
Volume Use Cases

Let’s Go Back to Our Dockerfile

```
1  # our base image
2  FROM alpine:latest
3
4  # Install python and pip
5  RUN apk add --update py-pip
6
7  # upgrade pip
8  RUN pip install --upgrade pip
9
10 # install Python modules needed by the Python app
11 COPY requirements.txt /usr/src/app/
12 RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
13
14 # copy files required for the app to run
15 COPY app.py /usr/src/app/
16 COPY templates/index.html /usr/src/app/templates/
17
18 # tell the port number the container should expose
19 EXPOSE 5000
20
21 # run the application
22 CMD ["python", "/usr/src/app/app.py"]
```

Each Dockerfile Command Creates a Layer

Docker Image Pull: Pulls Layers



Layers on the Physical Disk

- Logical file system by grouping different file system primitives into branches (directories, file systems, subvolumes, snapshots)
- Each branch represents a layer in a Docker image
- Containers will share common layers on the host
- Allows images to be constructed / deconstructed as needed vs. a huge monolithic image (ala traditional virtual machines)
- When a container is started a writeable layer is added to the “top” of the file system

Copy on Write

Super efficient:

- Sub second instantiation times for containers
- New container can take <1 Mb of space

Containers appears to be a copy of the original image

But, it is really just a link to the original shared image

If someone writes a change to the file system, a copy of the affected file/directory is “copied up”

Docker Volumes

- Volumes mount a directory on the host into the container at a specific location

```
$ docker volume create hello
hello
$ docker run -d -v hello:/world busybox ls /world
```

- Can be used to share (and persist) data between containers
  - Directory persists after the container is deleted
    - Unless you explicitly delete it

- Can be created in a Dockerfile or via CLI

Why Use Volumes

- Mount local source code into a running container

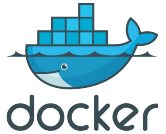
```
docker container run -v $(pwd):/usr/src/app/
mikegcoleman/catweb
```

- Improve performance
  - As directory structures get complicated traversing the tree can slow system performance
- Data persistence

Hands On Exercises (and Break)

Modifying a Running Container
Pushing Images to Docker Hub

http://labs.play-with-docker.com
https://github.com/ericsmalling/docker101-linux



More resources

- Official training resources from Docker: https://training.docker.com
  - Self paced link will take you to more hands-on-labs
  - Instructor led courses available
- Docker production documentation: https://docs.docker.com/
- Docker Certified Associate Program: https://success.docker.com/certification
-