Fixing the "*It works on my machine!*" Problem with Docker

Jared M. Smith

@jaredthecoder

About Me

- Cyber Security Research Scientist at Oak Ridge National Lab
- BS and MS in Computer Science from the University of Tennessee-Knoxville, current PhD student
- Guest Teacher at Treehouse
  - Intro to Big Data
  - Intro to Docker
  - Basic Web Security
  - OWASP Top 10

A Tale of Two Developers

Monday

Wednesday

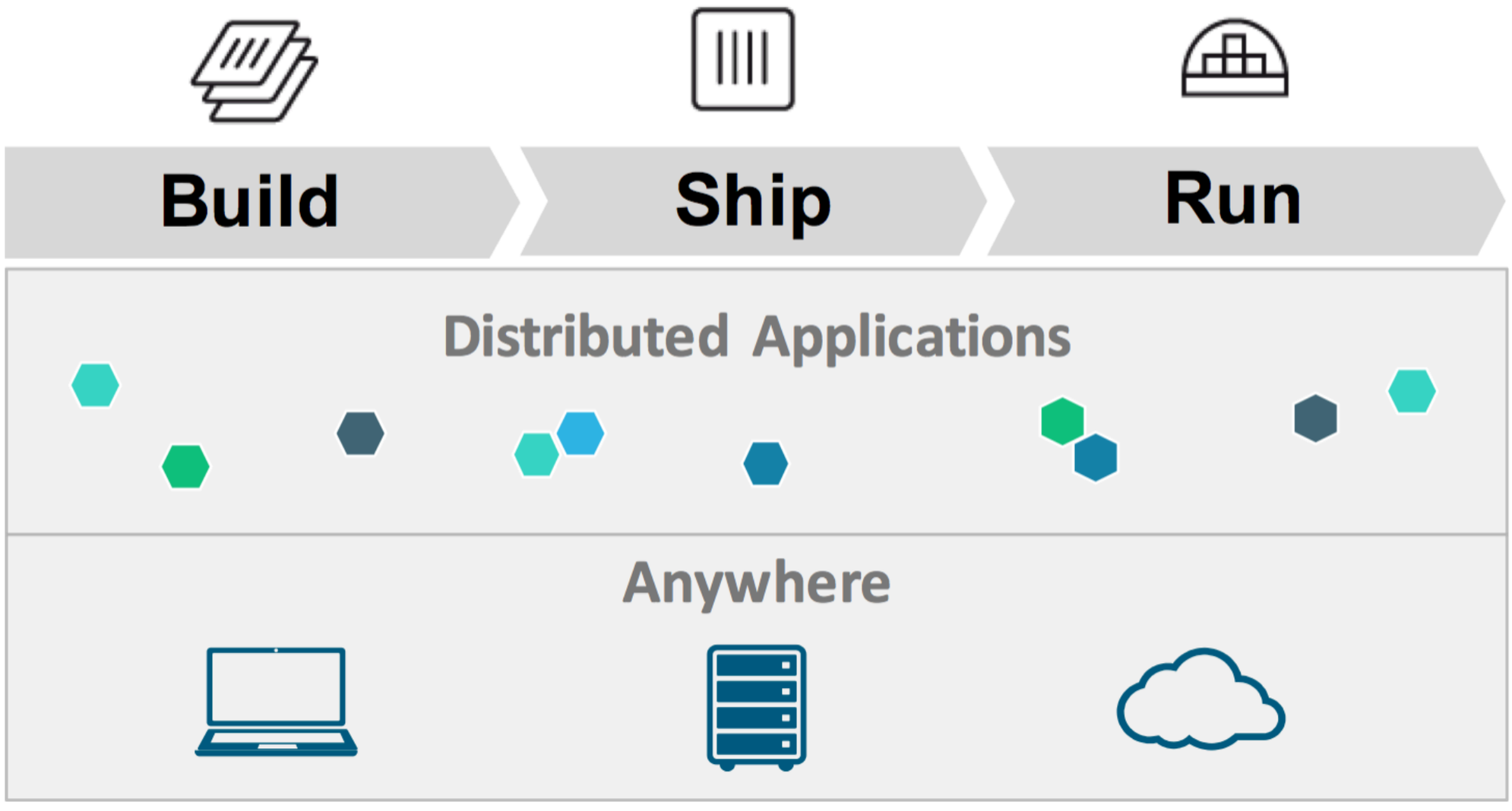
Friday

Following Monday

This does not have to be you or your team...

Docker

Docker's Mission



Released in 2013, now Docker is in use almost everywhere.

Docker makes packaging software simple.

Docker makes deploying software simple.

Docker makes scaling software simple.

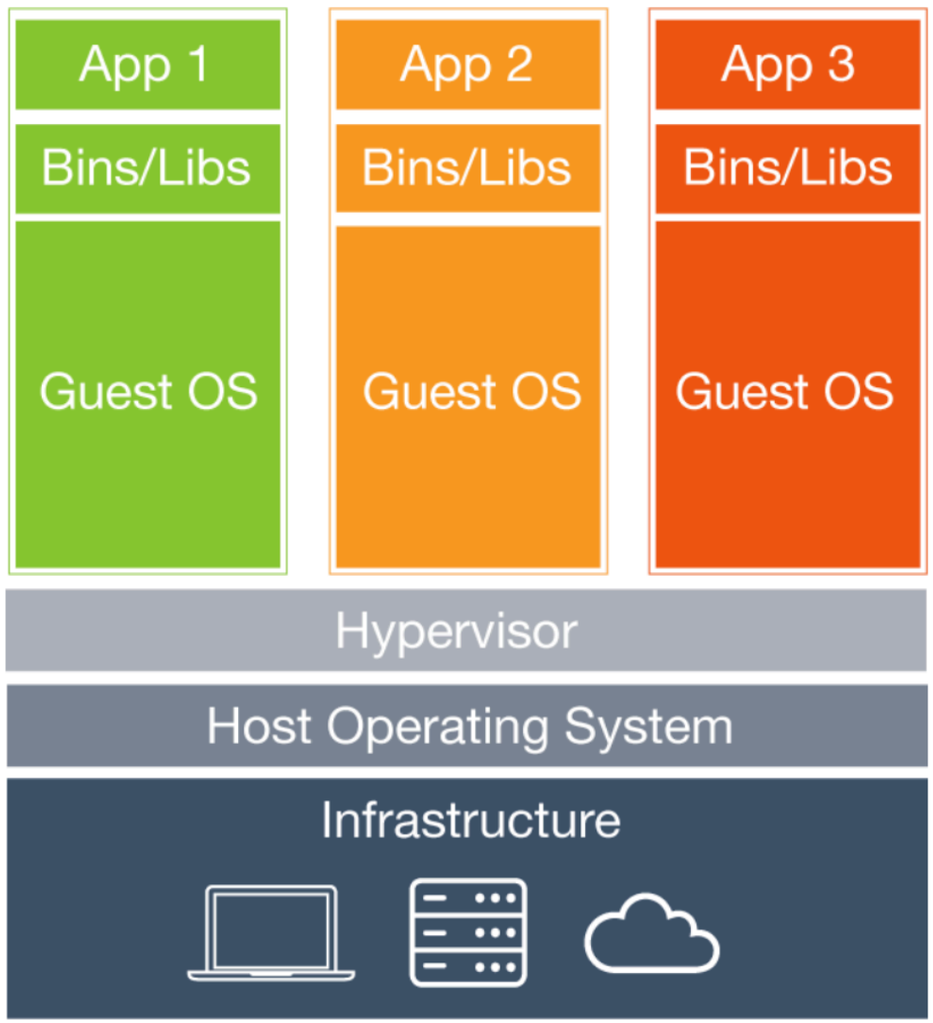
Docker makes securing software simple.

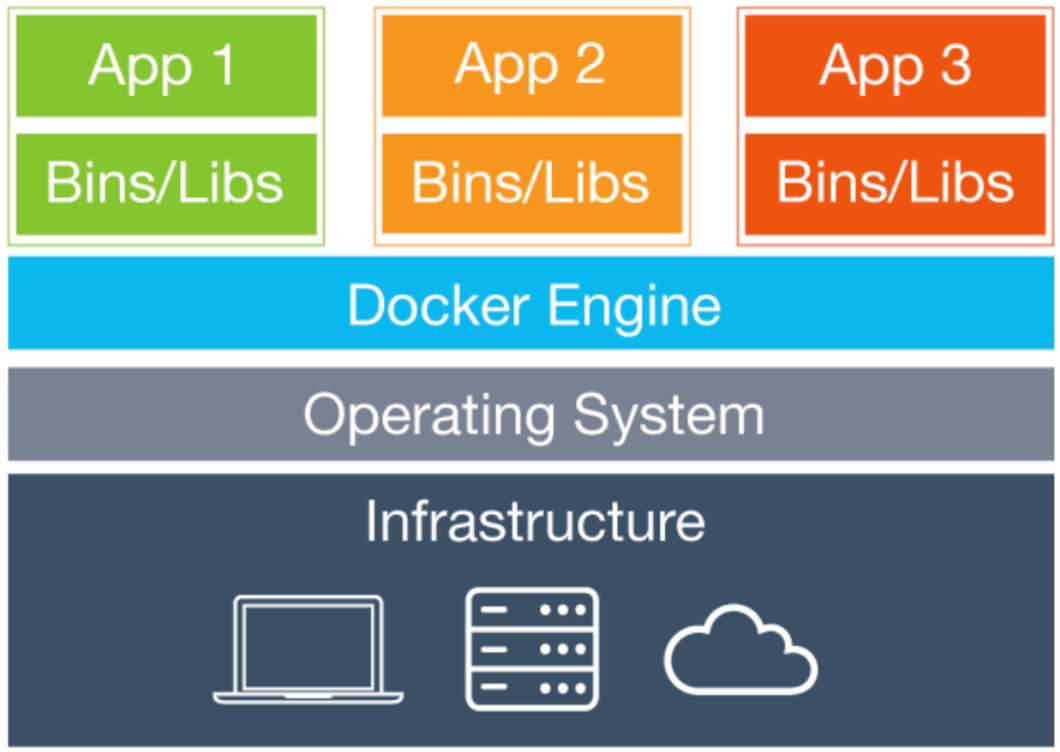
Docker simplifies the following:

— Packaging software
— Deploying software
— Running complex dependencies like DBs or isolating entire OSes for testing
— Connecting and scaling microservices
— Building CI/CD pipelines

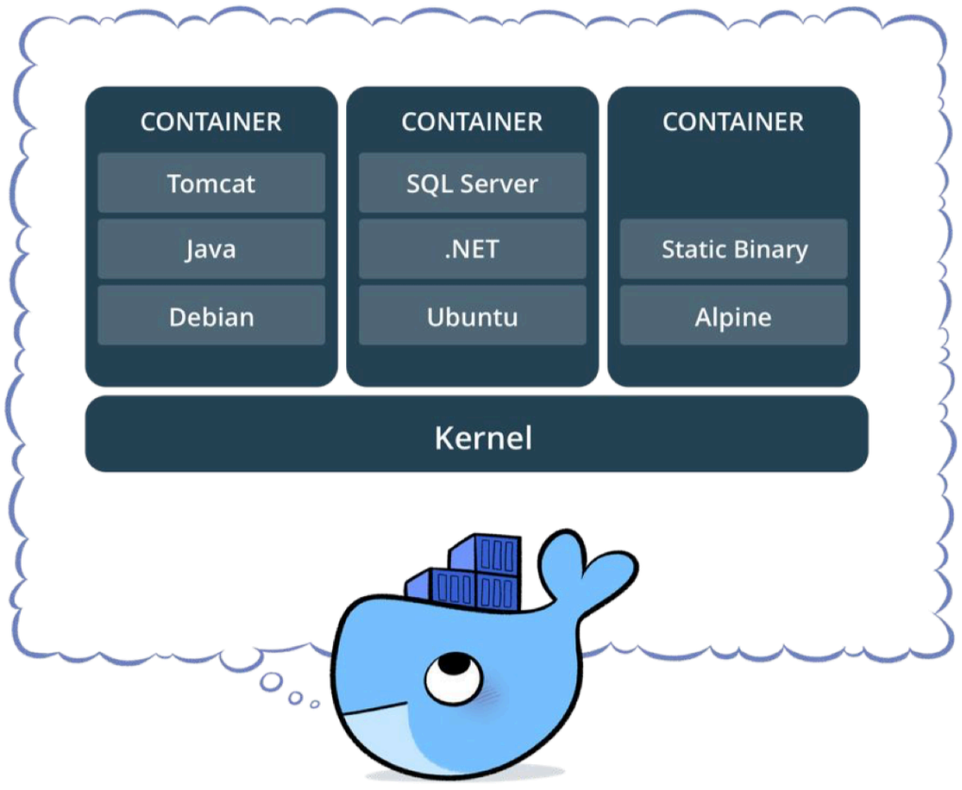
Background

Containers not VMs





What's a Container?



— Standardized packaging for software and dependencies
— Isolate apps from eachother
— Shares the same OS kernel
— Works for all major Linux distributions and in Windows Server 2016+

Terminology

— Docker Image
— Docker Container
— Docker Engine
— Registry Service
— Docker Hub
— Docker Trusted Registry

Diving In

Basic Docker Commands

```
$ docker pull nashcash/payment-
gateway:latest

$ docker images

$ docker run -d -p 5000:5000 --name
payments nashcash/payment-gateway:latest

$ docker ps

$ docker exec -it <container id> /bin/bash
```

Basic Docker Commands

```
$ docker stop payments (or <container id>)

$ docker restart/start payments (or
<container id>)

$ docker rm payments (or <container id>)

$ docker rmi nashcash/payment-
gateway:latest (or <image id>)
```

Basic Docker Commands

```
$ docker build -t nashcash/payment-gateway:
2.0 .

$ docker image push nashcash/payment-
gateway:2.0
```

```
$ docker search node
```

Dockerfile

```
FROM node:latest
USER node
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY package.json /usr/src/app/
RUN npm install
COPY . /usr/src/app
EXPOSE 3000
CMD [ “npm”, “start” ]
```

— The Dockerfile declares how to deploy your app or service

Each Dockerfile command creates a layer of the image.

New and old images share layers.

Docker copies data *on write* enabling fast startup and minimal disk usage.

By wrapping up app install and setup into a Dockerfile, and then using the Docker CLI, building and deploying can be very simple.

Installation

— Docker provides native apps for Mac and Windows, and via package managers for Linux

— docker.com/getdocker

— AWS, Microsoft Azure, and Google Cloud **all** support Docker as well

Scaling Up

Networking

- Connect multiple containers with bridge networking:
  - `$ docker network create -d bridge --name bridgenet1`
- Map ports from container to host:
  - `$ docker container run -p 8080:80 ...`
- Connect multiple hosts with their own containers with an overlay:
  - `$ docker network create -d overlay --name overnet`

To dockerize apps + external services or other apps, use Docker Compose.

Remember our Dockerfile earlier?

```
FROM node:latest
USER node
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
COPY package.json /usr/src/app/
RUN npm install
COPY . /usr/src/app
EXPOSE 3000
CMD [ “npm”, “start” ]
```

Let's Add MongoDB!

```
services:
  app:
    build: .
    ports:
      - "3000:3000"
    links:
      - mongo
mongo:
  image: mongo
  volumes:
    - ./data:/data/db
  ports:
    - "27017:27017"
```

And Redis!

```
services:
...
  redis:
    image: redis
    ports:
        - "6379:6379"
```

Docker Compose

```
$ docker-compose build

$ docker-compose up

$ docker-compose down
```

What about running multiple (even tens or thousands) Docker containers across hosts?

You need Orchestration.

The community has you covered

— Docker Swarm
— Apache Mesos
— Kubernetes
— ...

Running In Production

Monitoring Docker

— Stats: `docker stats`

— Logs: `docker service logs`

— Prometheus Endpoint (new in Docker 1.13)

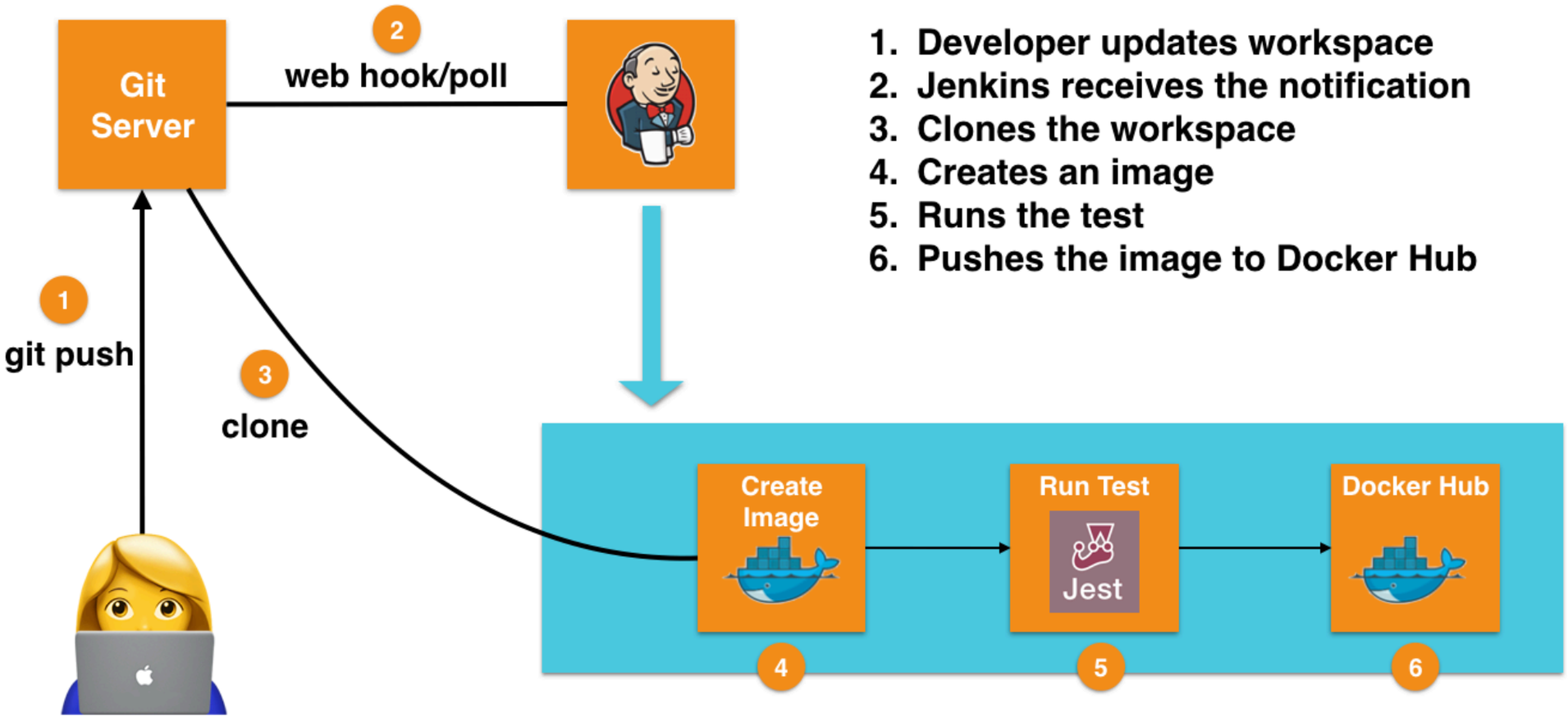
— Docker's Remote API: `/container/{container-name|cid}/stats`

— **cAdvisor**: https://github.com/google/cadvisor

Service Mesh

— **linkerd**: https://linkerd.io/
    — service discovery
    — load balancing
    — failure handling,
    — instrumentation
    — routing to all inter-service communication
— **Envoy**: https://www.envoyproxy.io/

CI/CD with Docker and Jenkins



1. Developer updates workspace
2. Jenkins receives the notification
3. Clones the workspace
4. Creates an image
5. Runs the test
6. Pushes the image to Docker Hub

*Graphic based on image from Arun Gupta of Couchbase.

Dockerizing React-Slingshot by Cory House

What We're Working With

— **Repo**: https://github.com/coryhouse/react-slingshot

— From the repo: "*React + Redux starter kit / boilerplate with Babel, hot reloading, testing, linting and a working example app, all built in*"

— We're also going make the app connect to a backend server that will talk to MongoDB

Let's begin!

Lessons Learned

— Docker is *usually* easy to use and can improve software dev **a lot**

— Integration, scaling, security, and testing are all well-explored areas with Docker

— With Docker, you can't tell your team that it works for you so it should for them

— If you do, now you're the kid saying the "dog ate my homework"

Thank you!

— Follow @jaredthecoder for lots of web security, devops, and data science

— Checkout my 2-hour Intro to Docker course on Treehouse (teamtreehouse.com) for more

— Links to slides and code to be posted on Github, Twitter, and jaredthecoder.com/talks