Deployment and Docker

Vocab

- Development Environment (dev)
  - The environment where you write your code
  - Ex. Your laptop
    - Add features; Find and eliminate bugs
- Production environment (prod)
  - The environment where your app will eventually live
  - The live server with real end users
  - Do everything we can to avoid bugs in production

Deployment Headaches

- It works on my laptop!
- Run your code in production and it's broken
- Many causes
  - Different version of compiler/interpreter
  - Dependancies not linked
  - Hard-coded path
  - Different environment variables
  - etc

Virtual Machines

- Simulate an entire machine
- Run the the virtual machine (VM) in your development environment for testing
- Run an exact copy of the VM on the production server
- No more surprise deployment issues

- Simulating an entire machine can be inefficient
  - If you've ran a VM on your laptop you know how slow this can get

Containers

- Containers are the new [not so new anymore] hot thing
- Effectively runs lightweight VMs
- Cross platform
  - And portable

Security

- Can't break out of the container
- If an attacker compromises the server, they can only access what you put in the container
  - Can't "rm -f /" your entire machine
  - Patch the exploited vulnerability and rebuild the image
- The attacker can still cause significant damage and steal private data
  - The just can't destroy your physical server box

Security

- Sometimes an app has to allow code injection attacks to function
  - AutoLab
  - AWS
  - Heroku
  - Digital Ocean
- Run user code in their own container

Docker

- Docker is software that's used to create containers
- Install Docker in your development environment to test containers
- Install Docker in your production environment to run containers in the same environment

Dockerfile

- To start working with Docker, write a Dockerfile
- This file contains all the instructions needed to build a Docker image
  - Some similarities to a Makefile

Dockerfile

- Let's explore this sample Dockerfile

- This Dockerfile creates an image for a node.js app

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- The first line of your Dockerfile will specify the base image
- This image is downloaded and the rest of your Dockerfile adds to this image
- In this example: We start with Ubuntu 18.04
  - Our Dockerfile can run Linux commands in Ubunutu

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Use the RUN keyword to run commands in the base image
- Use this for any setup of your OS before setting up your app
- In this example: Updating apt-get which is used to install software

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Use ENV to set environment variables
  - Setting the home directory here
    - Can use ENV to setup any other variables you need
- Use WORKDIR to change your current working directory
  - Same as "cd"

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Since we're starting with a fresh image of Ubuntu
  - Only the default software is installed
- RUN commands to install all required software for your app
  - Typically your development tools for your language of choice

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- COPY all your app file into the image
- "." denotes the current directory
- Run docker from your apps root directory
  - The the first "." will refer to your apps directory
- We changed the home and working directory to /root
  - The second "." refers to /root in the image

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Now that your apps files are in the image, run all app specific commands
  - Order is important
    - Don't depend on your app files before copying them into the image
- Use RUN to install dependancies and perform any other required setup

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Use EXPOSE to allow specific ports to be accessed from outside the container
- By default, all port are blocked
  - Container is meant to run in isolation
- To run a web app in a container, expose the port that your runs on

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Dockerfile

- Finally, use CMD to run you app
- Important: Do not use RUN to run your app!
- RUN will execute the command when the image is being built
- CMD will execute when the container is ran
- We do not want the app to run when the image is being built

```
FROM ubuntu:18.04

RUN apt-get update

# Set the home directory to /root
ENV HOME /root

# cd into the home directory
WORKDIR /root

# Install Node
RUN apt-get update --fix-missing
RUN apt-get install -y nodejs
RUN apt-get install -y npm

# Copy all app files into the image
COPY . .

# Download dependancies
RUN npm install

# Allow port 8000 to be accessed
# from outside the container
EXPOSE 8000

# Run the app
CMD ["node", "ecom_app.js"]
```

Docker Containers

- We can now build a Docker image
  - From the command line run "docker build -t <image_name> ."
  - Great, but we wanted a container
  - An image is use to create containers
    - Similar to using a class to create objects

Docker Containers

docker container run --publish <local_port>:8000 --detach <image_name>

- Once you have an image, run this line in the command line to create and run a container where
  - --publish <local_port>:8000 - maps a port on the host machine to an exposed port in the container
  - --detach - runs the container in the background
  - <image_name> matches the image name chosen when you created the image

Docker Containers

docker container run --publish <local_port>:8000 --detach <image_name>

- After running this command your app should be accessible from <local_port>