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 paths
  - Different environment variables
  - etc.

Virtual Machines

- Simulate an entire machine
- Run 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

Security

- Can't break out of the VM
- 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 VM/Container

Containers

- Containers are the hot new thing!
  - [Not so new anymore]

- Effectively, lightweight VMs

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 and run the same containers

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 app-www.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 app-www.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 app-www.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 app-www.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 the 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 app-www.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 app-www.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 app-www.js
```

Dockerfile

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

```
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 app-www.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 app-www.js
```

Dockerfile

- There are many base images to choose from
- Start with an image for your language installed to simplify your docker file
  - Search "docker <language>" to find images/tutorials for your favorite language
- This image starts with node installed so we can remove the node installation lines

```
FROM node:13

ENV HOME /root
WORKDIR /root

COPY . .

# Download dependancies
RUN npm install

EXPOSE 8000

CMD node app-www.js
```

Dockerfile

- Example for Python

```
FROM python:3.8

ENV HOME /root
WORKDIR /root

COPY . .

# Download dependancies
RUN pip3 install -r requirements.txt

EXPOSE 8000

CMD python3 app.py
```

Docker Containers

- We can now build a Docker image
  - From the command line, run "docker build -t <image_name> ."
  - Builds an image and names it <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 run -p <local_port>:8000 <image_name>

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

Docker Containers

docker run -p <local_port>:8000 -d <image_name>

- Running your container with "-d" will run your app it in the background
  - Used for deployment

Docker Containers

docker run -p <local_port>:8000 <image_name>

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

Running Your App

- When preparing your app to run in a container
  - **Do not use localhost**
  - Use 0.0.0.0 as the host instead
  - This allows your app to be accessed from outside the container