1.125 – Software Engineering and Architecting

Lectures 7-8

Docker and Containers

Dockerfiles

- Dockerfiles define a **single image**
- We will look at how to create and connect multiple image when we look at the docker compose files.
- The Dockerfile outlines what our image will be made from and do
- Running the command “docker build” will tell docker to build an image based on the instructions

```
# creates a layer from the node:carbon Docker image
FROM node:carbon

# create the app directory for inside the Docker image
WORKDIR /usr/src/app

# copy and install app dependencies from the package.json (and the
package-lock.json) into the root of the directory created above
COPY package*.json ./
RUN npm install

# bundle app source inside Docker image
COPY . .

# expose port 8080 to have it mapped by Docker daemon
EXPOSE 8080

# define the command to run the app (it's the npm start script from
the package.json file)
CMD [ "npm", "start" ]
```

Dockerfile

Common ingredients for building the image

- FROM — this initializes a new build stage and sets the *Base Image* for subsequent instructions. As such, a valid Dockerfile must start with a FROM instruction.
- RUN — will execute any commands in a new layer on top of the current image and commit the results. The resulting committed image will be used for the next step in the Dockerfile.
- ENV — sets the environment variable <key> to the value <value>. This value will be in the environment for all subsequent instructions in the build stage and can be replaced inline in many as well.
- EXPOSE — informs Docker that the container listens on the specified network ports at runtime. You can specify whether the port listens on TCP or UDP, and the default is TCP if the protocol is not specified. This makes it possible for the host and the outside world to access the isolated Docker Container
- VOLUME — creates a mount point with the specified name and marks it as holding externally mounted volumes from the native host or other containers.

```
# creates a layer from the node:carbon Docker image
FROM node:carbon


# create the app directory for inside the Docker image
WORKDIR /usr/src/app


# copy and install app dependencies from the package.json (and the
package-lock.json) into the root of the directory created above
COPY package*.json ./
RUN npm install


# bundle app source inside Docker image
COPY . .


# expose port 8080 to have it mapped by Docker daemon
EXPOSE 8080


# define the command to run the app (it's the npm start script from
the package.json file)
CMD [ "npm", "start" ]
```

Example Dockerfile for a simple node app

// Once we have the code written, and a Dockerfile, to build the image we type:
// docker build <tag> <location of dockerfile>. E.g. To build in the same directory as the file the image nodeApp
docker build –t nodeApp .

Docker Images

Images can come from either:

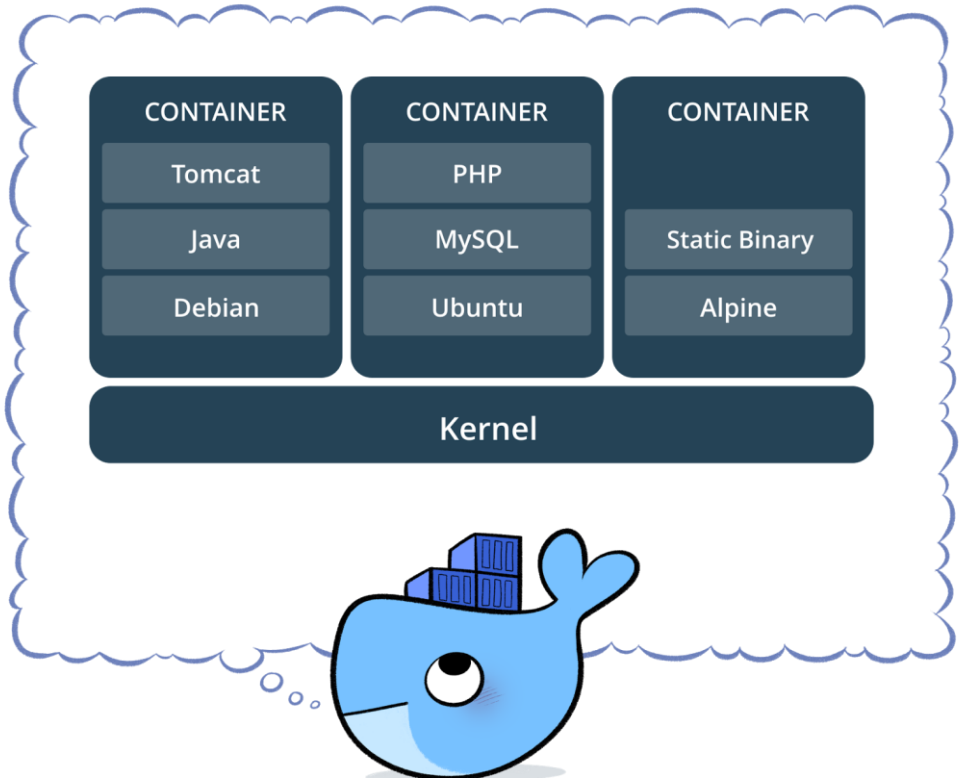
- Docker Hub

  docker run <nameOfImage>

- Our Code

  (After docker build –t nodeApp .)

  docker run <options like ports> nodeApp



Dockerfile for Express

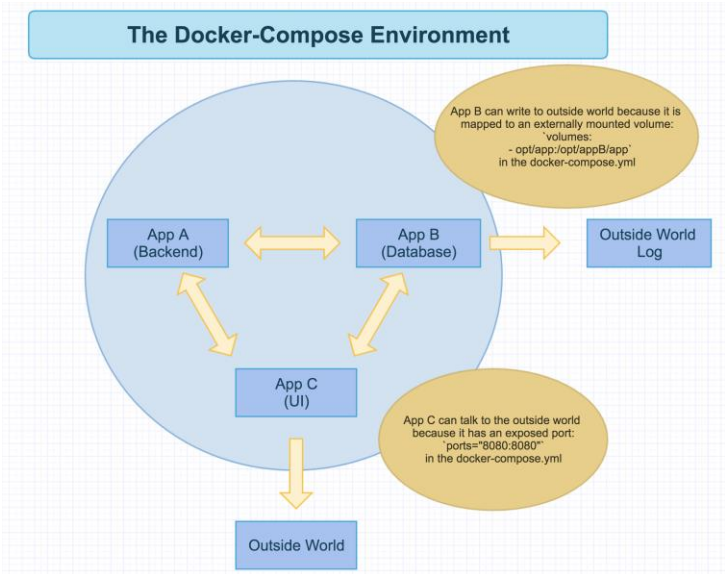
Docker compose

Creating our services and running multiple apps

Basic example

```
# docker-compose.yml
version: '2'

services:
  web:
    build: .
    # build from Dockerfile
    context: ./Path
    dockerfile: Dockerfile
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: redis
```



Commands

```
docker-compose start
docker-compose stop
```

```
docker-compose pause
docker-compose unpause
```

```
docker-compose ps
docker-compose up
docker-compose down
```

Docker-compose