Docker Workshop

A collaboration between GDSC and CSSC.

$whoami

> Daniel Laufer
> GDSC Workshop lead

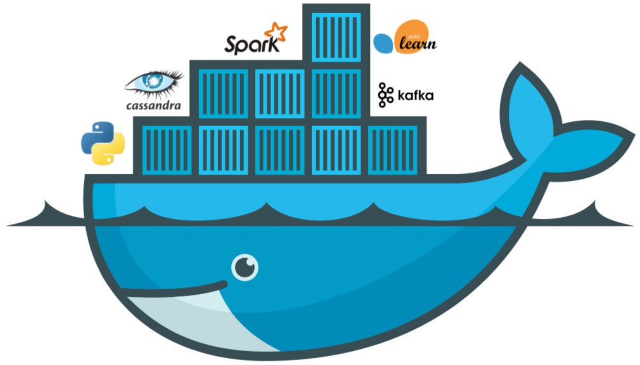
> Ritvik Bhardwaj
> CSSC Tech Director

Today’s Agenda

- What is a VM?
- What is Docker?
- Hands-on activity with Docker
- Why do we need Docker?
- What is Docker compose?
- Hands-on activity with Docker-compose

What is Docker

- Software that allows you deploy your applications in containers
- Allows you to essentially *bundle* up your codebase into a package (called a container) that can be deployed and ran anywhere with ease.
- If it runs on one machine, it'll run on the rest!

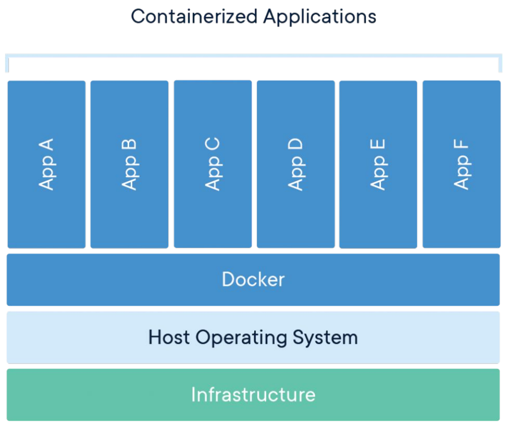


Docker terminology

- Dockerfile - the file used to tell docker how to build your image
- Image - The blueprint used to create a container, you can share this with people and build on top of it!
- Container - The running instance of your image, usually what’s deployed
- Scaling up/down - the process in which to add more RAM and storage to a container, allows for dynamic usage
- YAML - yet another markup language, the main language used for docker compose
- Docker Daemon - the service responsible for running the containers, (comes paired with docker desktop and needs to be running to be able to use Docker in the first place)

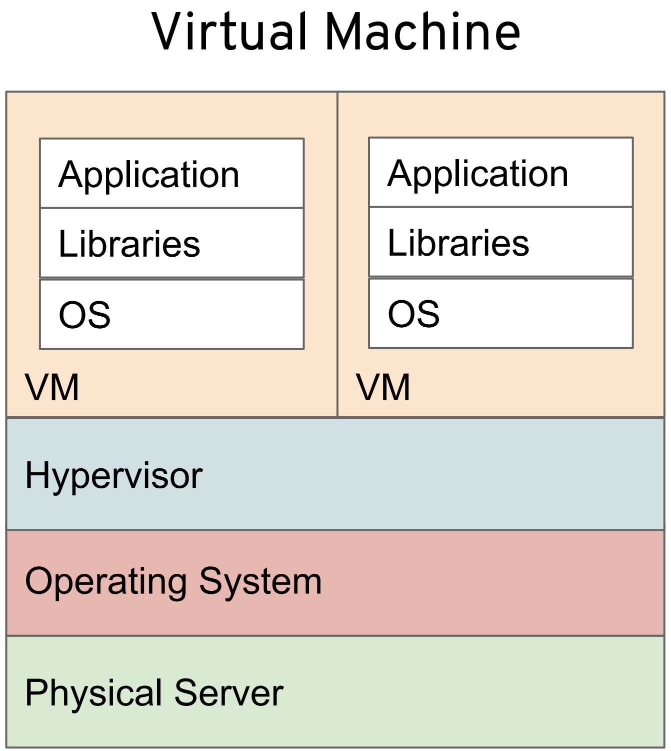
More on Containers

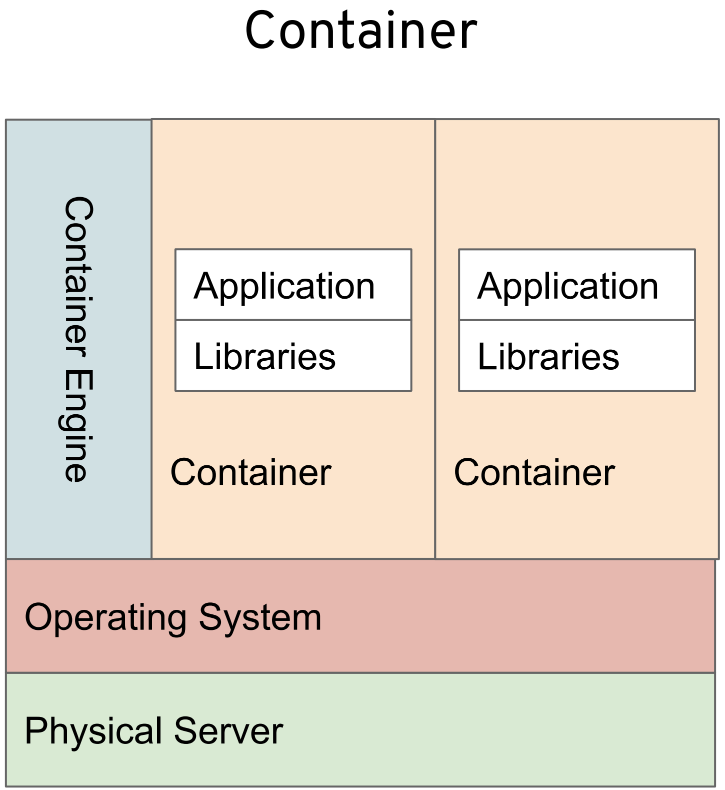
- A standard unit of software that **packages up code and all its dependencies** so the application **runs quickly and reliably** from one computing environment to another.
- Each container is created from an “image”
  - Think of an image as a template/blueprint for all the contains that are created from it
    - Similar to how objects are created from classes in Java, Python,etc
- Docker containers are similar to Virtual Machines in many ways



*Slide taken from krish Chowdhary 's WhiteVan Docker workshop

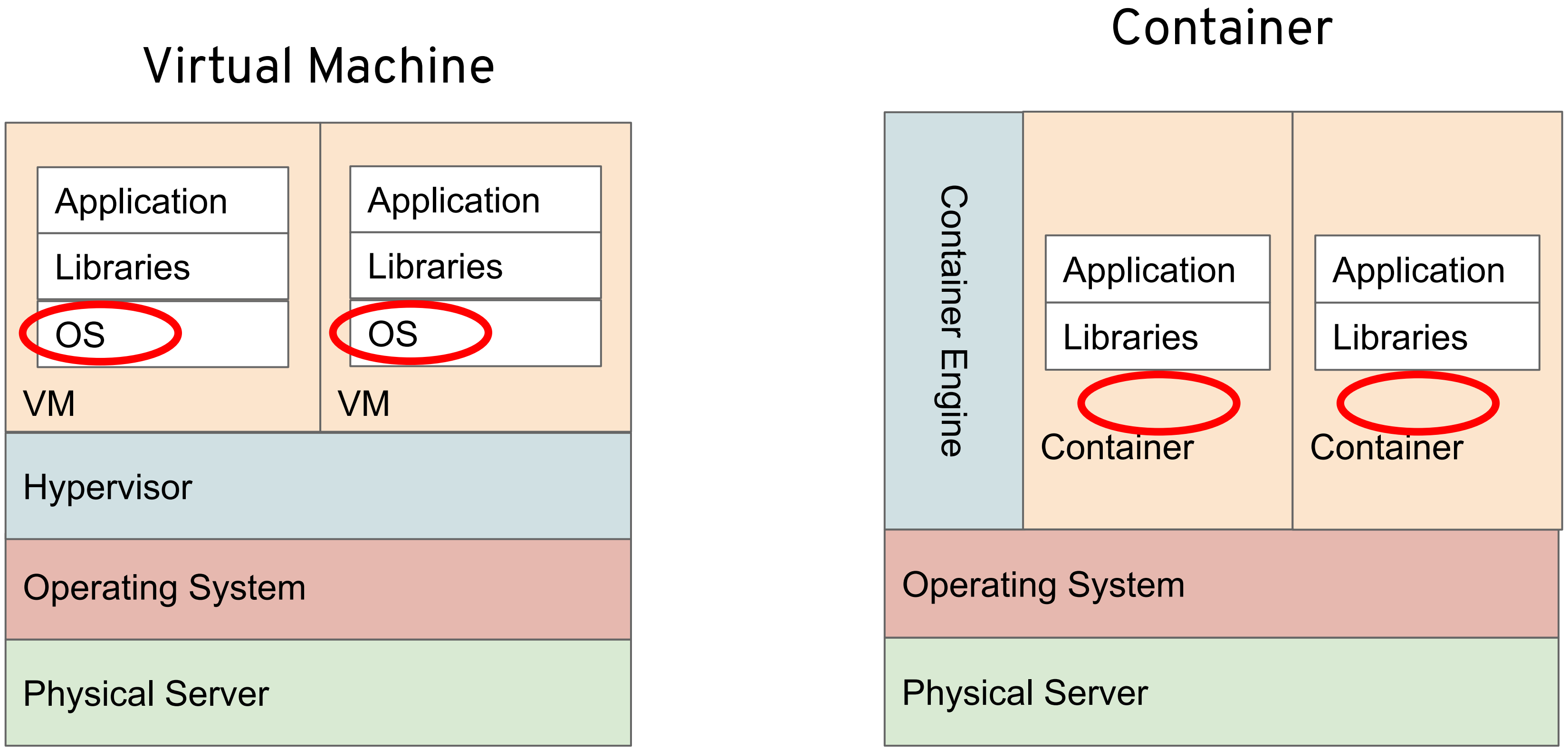
Docker Container vs Virtual Machine





*Slide taken from krish Chowdhary 's WhiteVan Docker workshop

Docker Container vs Virtual Machine



So why is Docker useful?

- Say we want to manually deploy our React app on a server.
- These are the steps we'd have to take to deploy it:
  1. Transfer all the files over the server
  2. cd into the project directory and run **npm install**

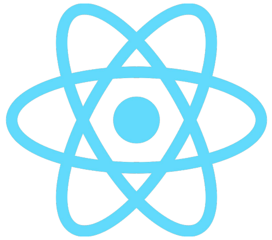
     Oh no! It turns out our server doesn't have node installed. So let's install it

  3. Run **npm install** again

     Oh no! It turns out we installed the wrong version of node and we are unable to install our required dependencies and/or run the app

  4. Run ***npm install** again*

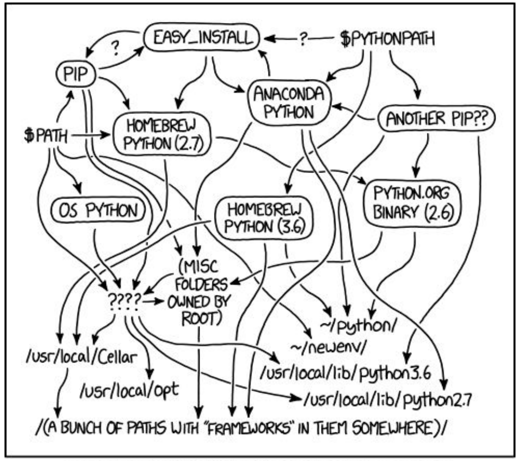
     It finally works!



So why is Docker useful?

- In this case it **was difficult to deploy our app to our server** because of missing/outdated dependencies required to run our app
- We entered "Dependency hell" 💀😔
- It's **very difficult** to ensure our software runs on all types of computers, operating systems, etc
- Think of having to deploy this app on many different servers, that each can present their own unique problems!
- What if we want to scale our program up? Or if we want to reduce it down?

Docker provides a solution to this!



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

How can we use Docker to avoid these issues?

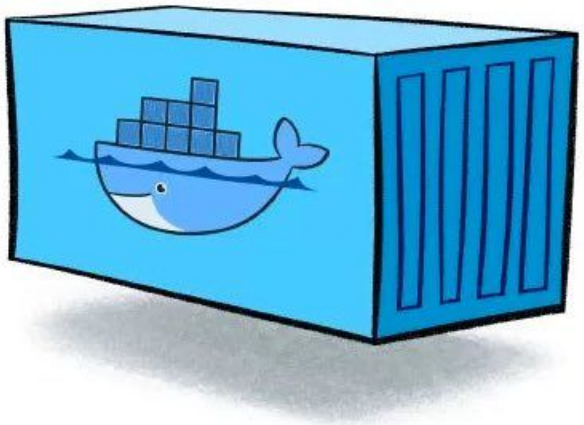
- Idea:
    - Ensure that the docker desktop is running
    - **Create a Docker image** that describes how to run our react app in a container
    - Push this image **to docker hub** (a website where you can upload the docker images you make. Similar to GitHub)
    - **Pull this image from docker hub** (ex. docker pull mywebsiteimage) onto your server

Quick example:

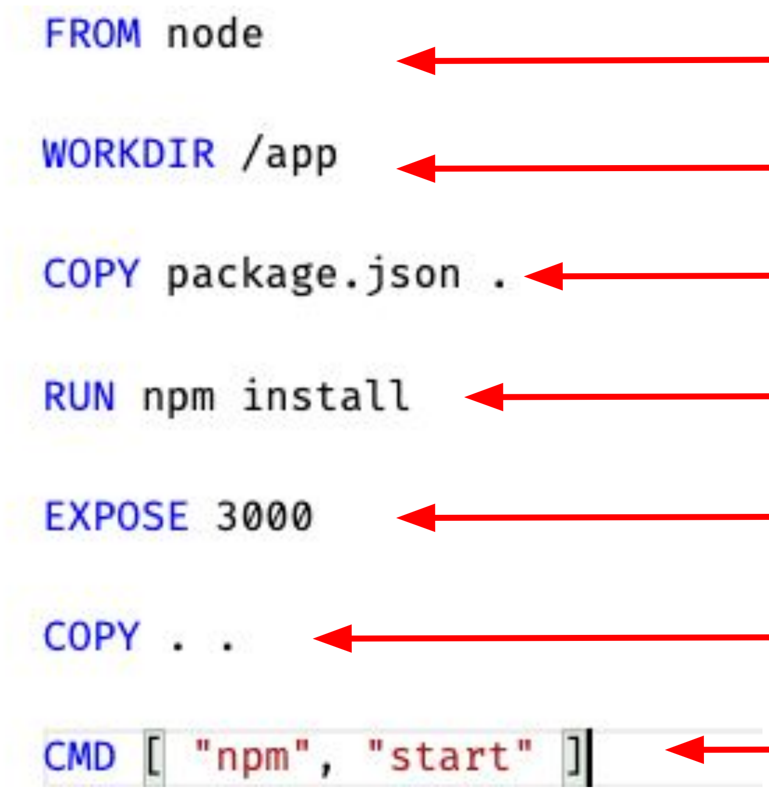
- docker pull hello-world
    - Found here: https://hub.docker.com/_/hello-world
- docker run hello-world

How can we use Docker to avoid these issues?

- Use the **docker run <yourimagenamehere>** command to create a container from the image and start it up on your server.
- We don’t have to worry about what’s installed on the actual server (all you need is docker installed).
- The docker container is running your website in it’s own isolated environment!



Dockerfiles in depth



```
FROM node

WORKDIR /app

COPY package.json .

RUN npm install

EXPOSE 3000

COPY . .

CMD [ "npm", "start" ]
```

Specify a “base image”. Need some sort of starting point to build our container off of. This instead could be “ubuntu”, or “postgres”, etc.

The directory that you want to create/use inside the container

From **our computer** copy over the file package.json **into** the /app directory **inside the** docker container (when it’s created)

The command that’ll be run when the container is created

From **our computer** copy over all the other files in directory ‘.’ on our computer into the /app directory **inside** the container

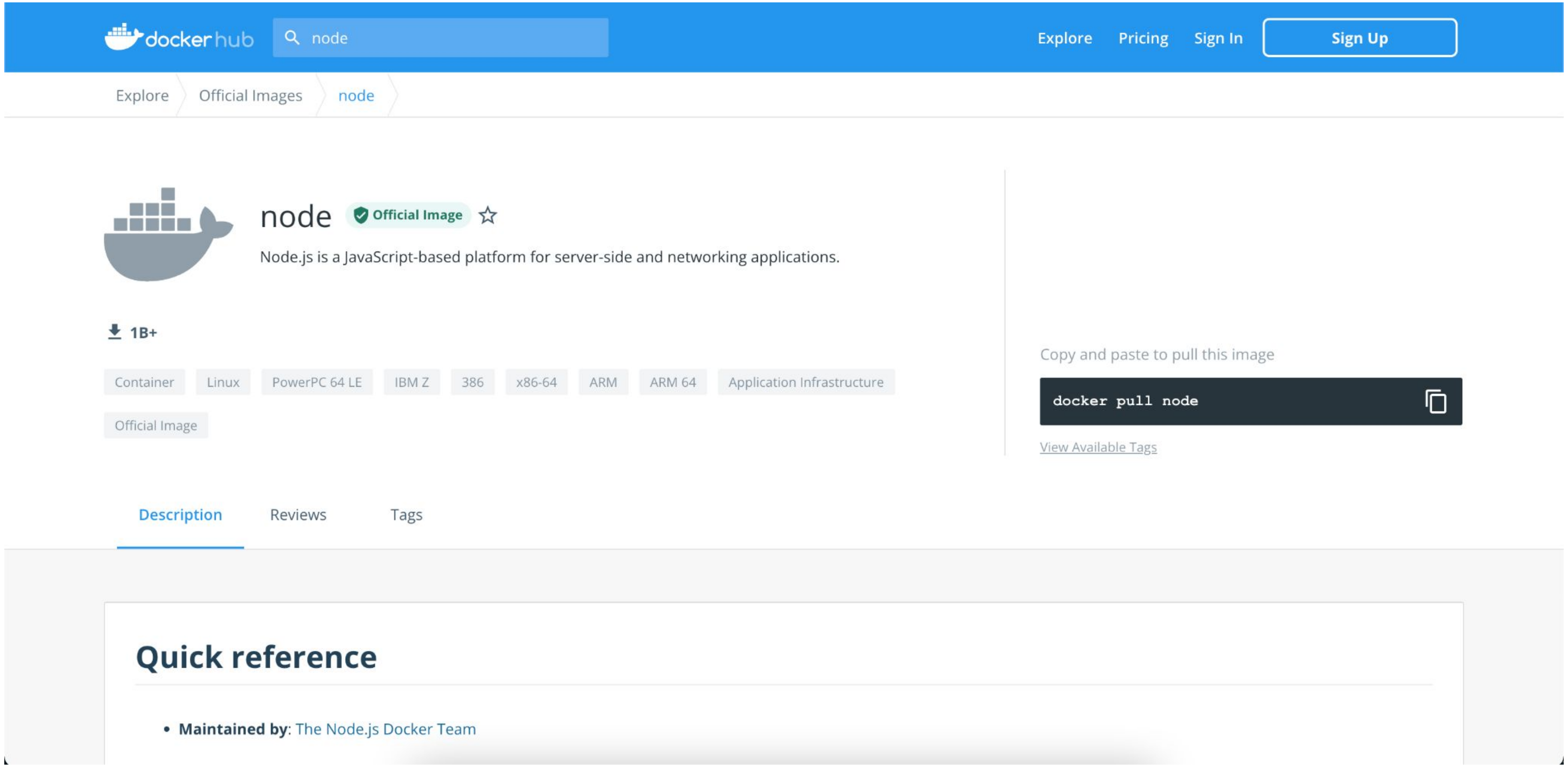
The command that’ll be run every time we start up the container (note: creating a container is different from starting up a container)

Docker Installation

Install Docker here: https://www.docker.com/products/docker-desktop

Ensure you run Docker Desktop after installing!

Docker Hub



node Official Image

Node.js is a JavaScript-based platform for server-side and networking applications.

1B+

Container | Linux | PowerPC 64 LE | IBM Z | 386 | x86-64 | ARM | ARM 64 | Application Infrastructure | Official Image

Copy and paste to pull this image

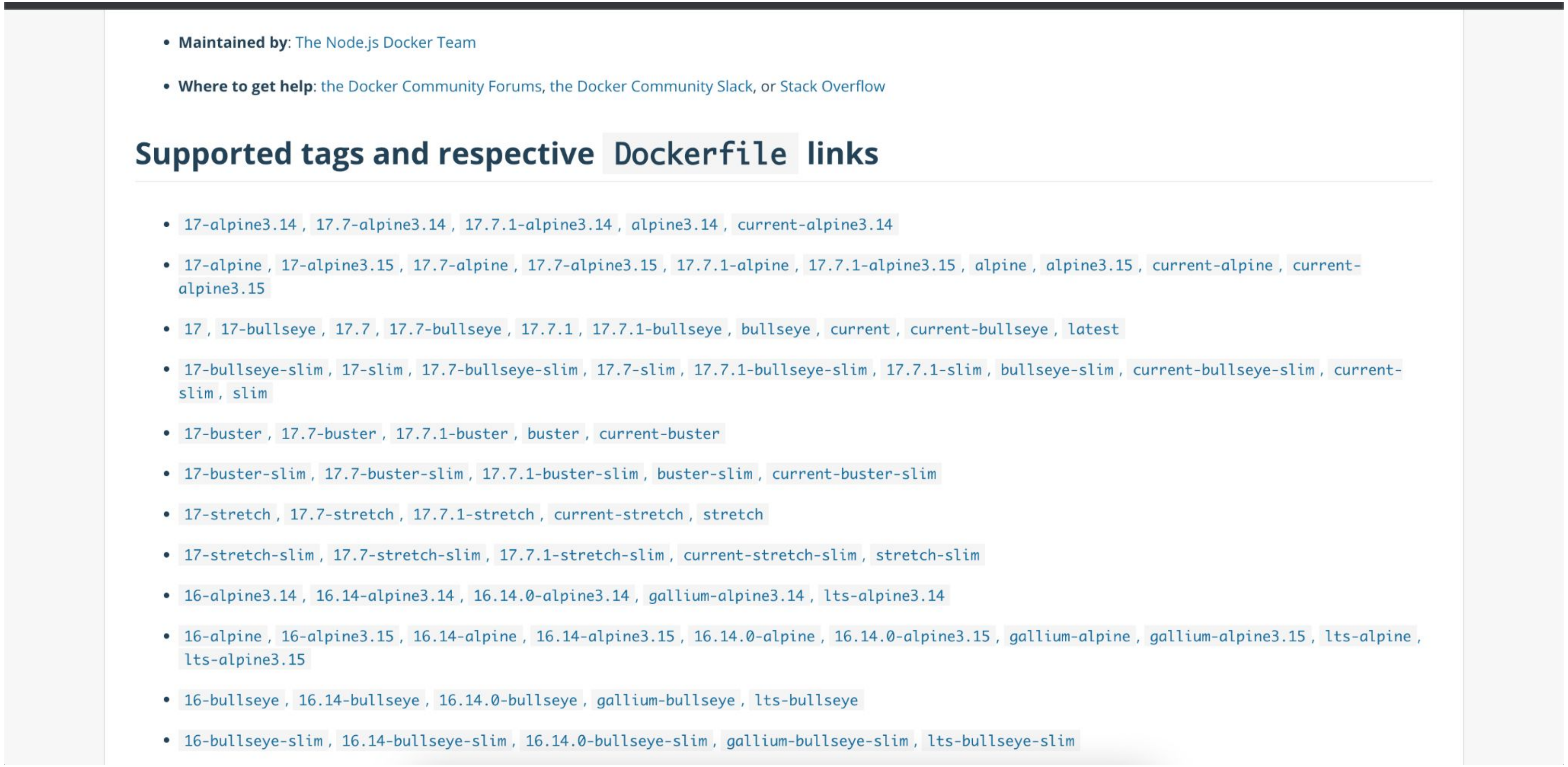
```
docker pull node
```

View Available Tags

Quick reference

- **Maintained by**: The Node.js Docker Team

Docker Hub



- **Maintained by**: The Node.js Docker Team
- **Where to get help**: the Docker Community Forums, the Docker Community Slack, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `17-alpine3.14`, `17.7-alpine3.14`, `17.7.1-alpine3.14`, `alpine3.14`, `current-alpine3.14`
- `17-alpine`, `17-alpine3.15`, `17.7-alpine`, `17.7-alpine3.15`, `17.7.1-alpine`, `17.7.1-alpine3.15`, `alpine`, `alpine3.15`, `current-alpine`, `current-alpine3.15`
- `17`, `17-bullseye`, `17.7`, `17.7-bullseye`, `17.7.1`, `17.7.1-bullseye`, `bullseye`, `current`, `current-bullseye`, `latest`
- `17-bullseye-slim`, `17-slim`, `17.7-bullseye-slim`, `17.7-slim`, `17.7.1-bullseye-slim`, `17.7.1-slim`, `bullseye-slim`, `current-bullseye-slim`, `current-slim`, `slim`
- `17-buster`, `17.7-buster`, `17.7.1-buster`, `buster`, `current-buster`
- `17-buster-slim`, `17.7-buster-slim`, `17.7.1-buster-slim`, `buster-slim`, `current-buster-slim`
- `17-stretch`, `17.7-stretch`, `17.7.1-stretch`, `current-stretch`, `stretch`
- `17-stretch-slim`, `17.7-stretch-slim`, `17.7.1-stretch-slim`, `current-stretch-slim`, `stretch-slim`
- `16-alpine3.14`, `16.14-alpine3.14`, `16.14.0-alpine3.14`, `gallium-alpine3.14`, `lts-alpine3.14`
- `16-alpine`, `16-alpine3.15`, `16.14-alpine`, `16.14-alpine3.15`, `16.14.0-alpine`, `16.14.0-alpine3.15`, `gallium-alpine`, `gallium-alpine3.15`, `lts-alpine`, `lts-alpine3.15`
- `16-bullseye`, `16.14-bullseye`, `16.14.0-bullseye`, `gallium-bullseye`, `lts-bullseye`
- `16-bullseye-slim`, `16.14-bullseye-slim`, `16.14.0-bullseye-slim`, `gallium-bullseye-slim`, `lts-bullseye-slim`

Let’s get some hands-on Docker experience!

Code we will be using is found at github.com/UTM-GDSC/docker-workshop

Some useful commands

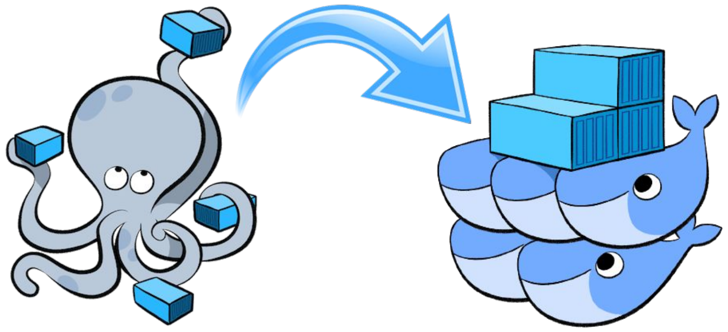
- **docker build -t docker-workshop-mar-11 .**
  - The '.' indicates to Docker that our Dockerfile exists in the directory '.' on our computer
  - Essentially builds our image and gives it a name of 'docker-workshop-mar-11'
- **docker run -p 3000:3000 docker-workshop-mar-11**
  - Take anything arriving at port 3000 on your computer and direct it into port 3000 inside the container
  - Docker run builds the image that we created in the previous step and then starts the container!
- **docker image rm docker-workshop-mar-11 --force**
-

Docker Compose

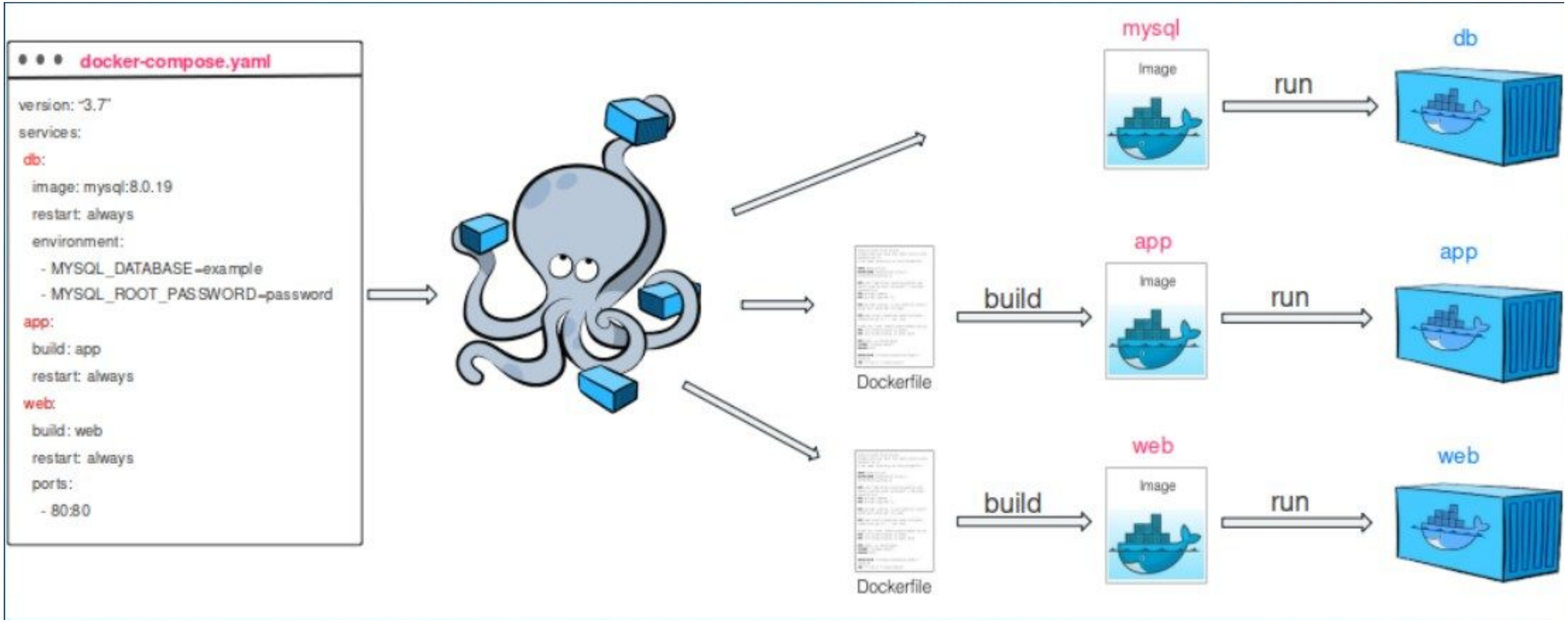
Creating Multi-container Docker applications

What is Docker compose?

- A tool that allows you to **create and run multi-container docker applications**
- You can define container restart policies for containers (what to do when a container, stops, or exits with an error code, etc)
- You can ensure that containers are started up in a predefined order
- Allows you to define how containers can communicate with each other
  - Allows you to create communication channels between your containers
- And much, much more!



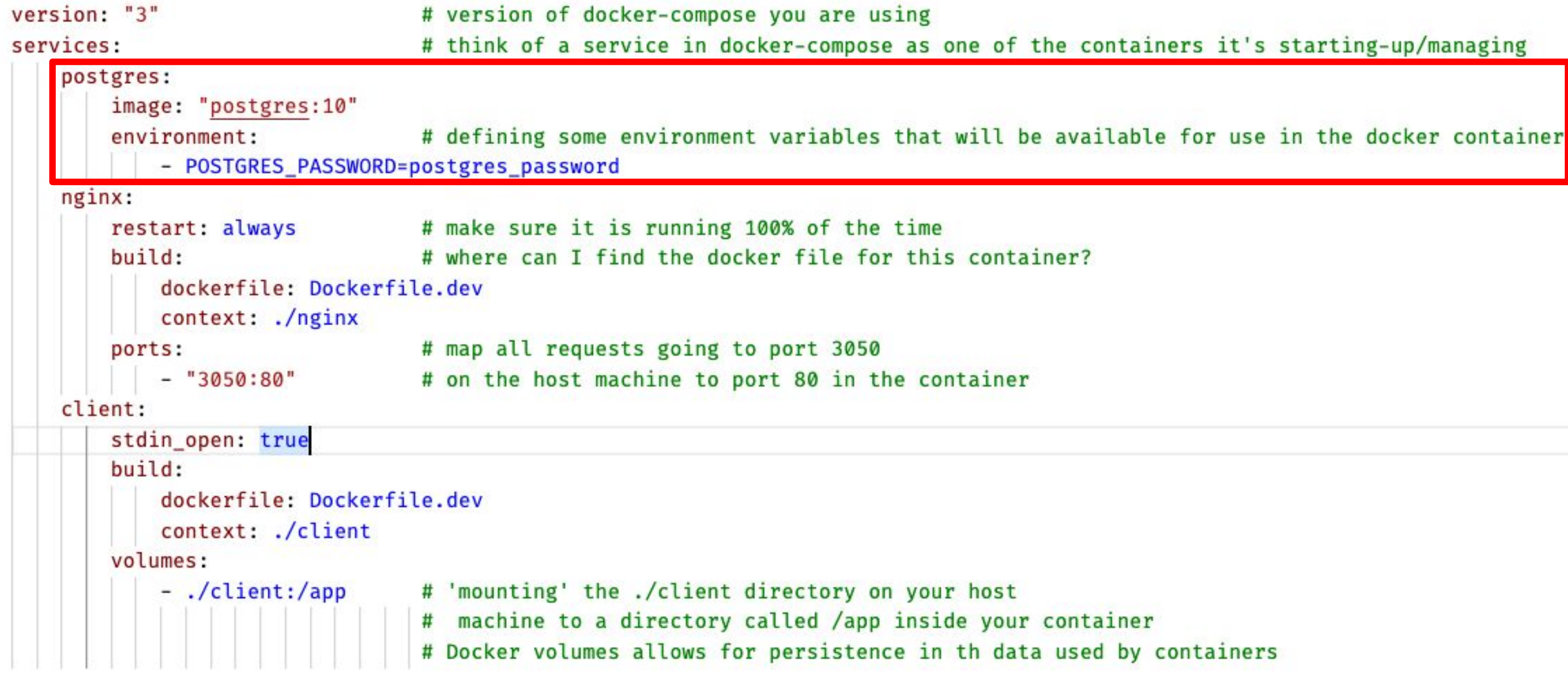
What is Docker compose?



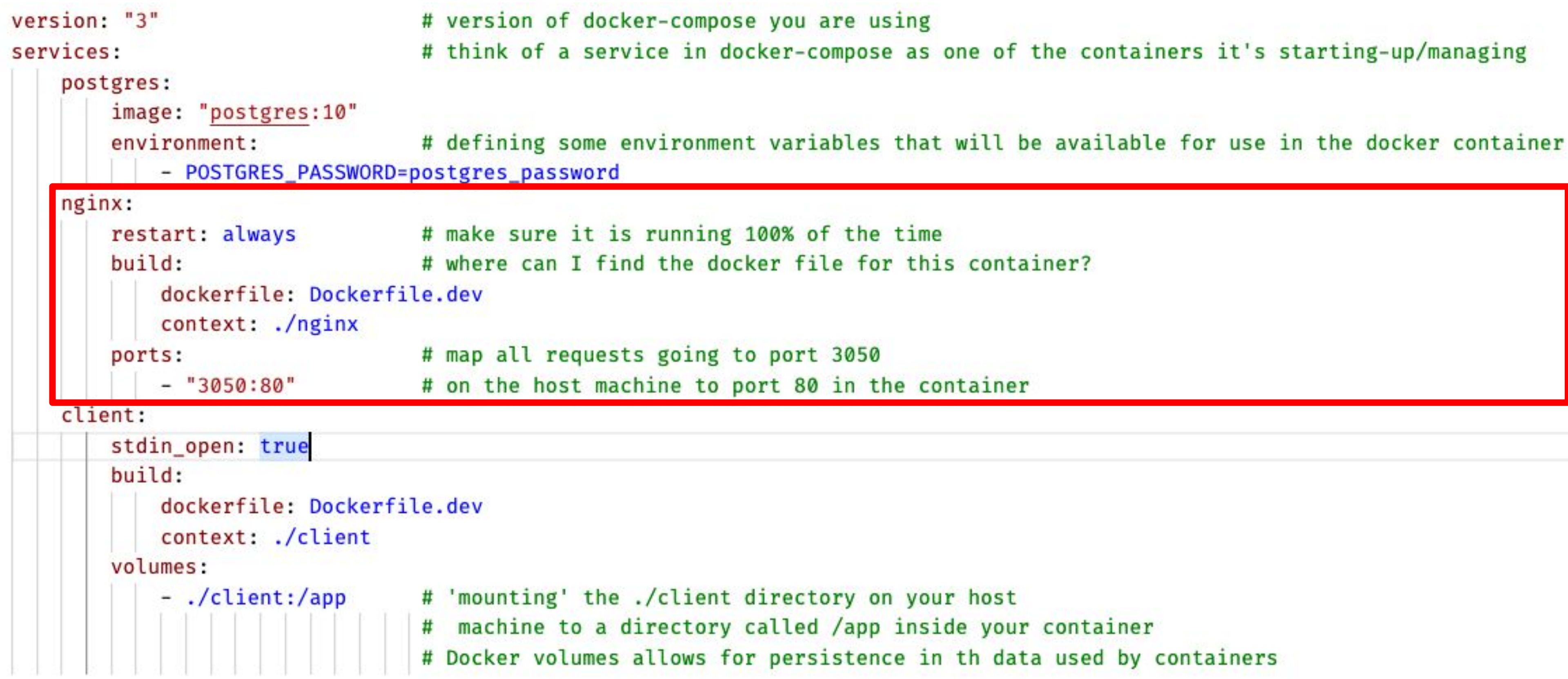
```
docker-compose.yaml

version: "3.7"
services:
 db:
   image: mysql:8.0.19
   restart: always
   environment:
     - MYSQL_DATABASE=example
     - MYSQL_ROOT_PASSWORD=password
 app:
   build: app
   restart: always
 web:
   build: web
   restart: always
   ports:
     - 80:80
```

Writing Docker compose yaml files

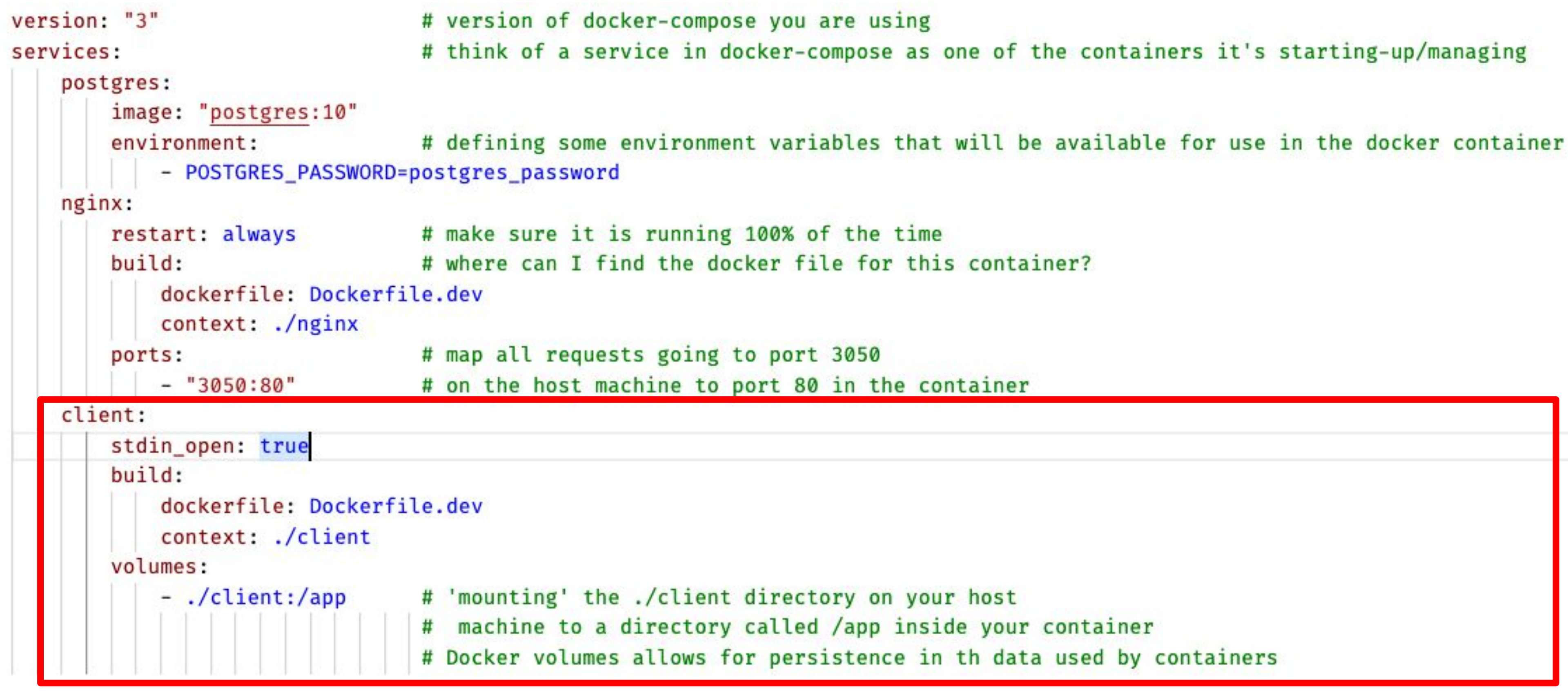
- You write your docker compose configuration in a file called ***docker-compose.yaml***
  - You can use other file names but this is the default one
    - ***Example: daniel-compose.yaml is completely valid***
      - *You will need to tell Docker about your custom naming scheme though!*
  - Let's see an example of a docker-compose file!



```yaml
version: "3"                     # version of docker-compose you are using
services:                        # think of a service in docker-compose as one of the containers it's starting-up/managing
    postgres:
        image: "postgres:10"
        environment:             # defining some environment variables that will be available for use in the docker container
            - POSTGRES_PASSWORD=postgres_password
    nginx:
        restart: always          # make sure it is running 100% of the time
        build:                   # where can I find the docker file for this container?
            dockerfile: Dockerfile.dev
            context: ./nginx
        ports:                   # map all requests going to port 3050
            - "3050:80"          # on the host machine to port 80 in the container
    client:
        stdin_open: true
        build:
            dockerfile: Dockerfile.dev
            context: ./client
        volumes:
            - ./client:/app      # 'mounting' the ./client directory on your host
                                 #  machine to a directory called /app inside your container
                                 # Docker volumes allows for persistence in th data used by containers
```



```yaml
version: "3"                     # version of docker-compose you are using
services:                        # think of a service in docker-compose as one of the containers it's starting-up/managing
    postgres:
        image: "postgres:10"
        environment:             # defining some environment variables that will be available for use in the docker container
            - POSTGRES_PASSWORD=postgres_password
    nginx:
        restart: always          # make sure it is running 100% of the time
        build:                   # where can I find the docker file for this container?
            dockerfile: Dockerfile.dev
            context: ./nginx
        ports:                   # map all requests going to port 3050
            - "3050:80"          # on the host machine to port 80 in the container
    client:
        stdin_open: true
        build:
            dockerfile: Dockerfile.dev
            context: ./client
        volumes:
            - ./client:/app      # 'mounting' the ./client directory on your host
                                 #  machine to a directory called /app inside your container
                                 # Docker volumes allows for persistence in th data used by containers
```



```yaml
version: "3"                    # version of docker-compose you are using
services:                       # think of a service in docker-compose as one of the containers it's starting-up/managing
    postgres:
        image: "postgres:10"
        environment:            # defining some environment variables that will be available for use in the docker container
            - POSTGRES_PASSWORD=postgres_password
    nginx:
        restart: always         # make sure it is running 100% of the time
        build:                  # where can I find the docker file for this container?
            dockerfile: Dockerfile.dev
            context: ./nginx
        ports:                  # map all requests going to port 3050
            - "3050:80"         # on the host machine to port 80 in the container
    client:
        stdin_open: true
        build:
            dockerfile: Dockerfile.dev
            context: ./client
        volumes:
            - ./client:/app     # 'mounting' the ./client directory on your host
                                #  machine to a directory called /app inside your container
                                # Docker volumes allows for persistence in th data used by containers
```

Running Docker compose

- First cd into the directory that contains that your docker-compose.yaml file
- You can then start up your multi-container docker application by running the command:
  - ***docker compose up***
- Or if you wrote your code in a file named something other than docker-compose.yaml
  - ***docker compose -f <somefilename>.yml up***
- Then to stop (aka *shut down*) your containers, run:
  - ***docker compose down***
- Or ***docker-compose -f <somefilename>.yml*** **down** for non-default docker-compose file names

Let’s get some hands-on Docker Compose experience!

Code we will be using is found at github.com/UTM-GDSC/docker-workshop

Thank you for Attending!