Docker Q & A

💡 **YOU NEED TO KNOW:**

- Anatomy of a DockerFile
- 3 Keywords: **Image**, **Container**, **Volume**
- 3 Commands: `docker build` / `docker run` / `docker volume create`
- 2 optional commands: `docker ps` / `docker prune`

https://gitlab.com/gsei19/nineteen-week/lecture-repos/week-07/json-view-lecture.git

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ed4978e7-51e1-4b71-bff5-c39d86fffef1/Docker_Example_Code.zip

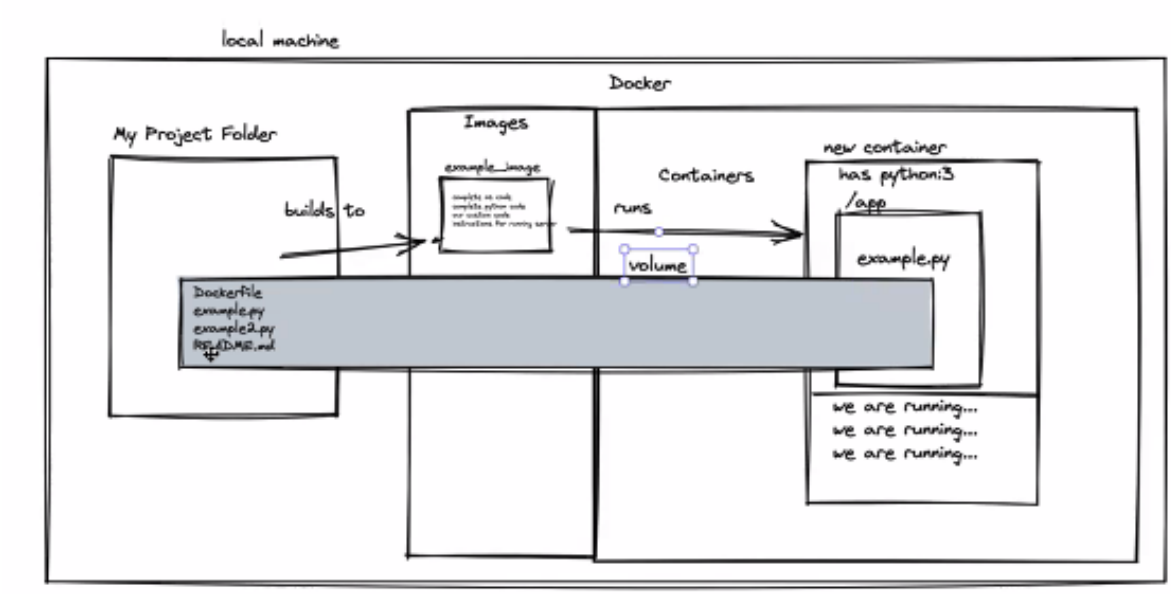
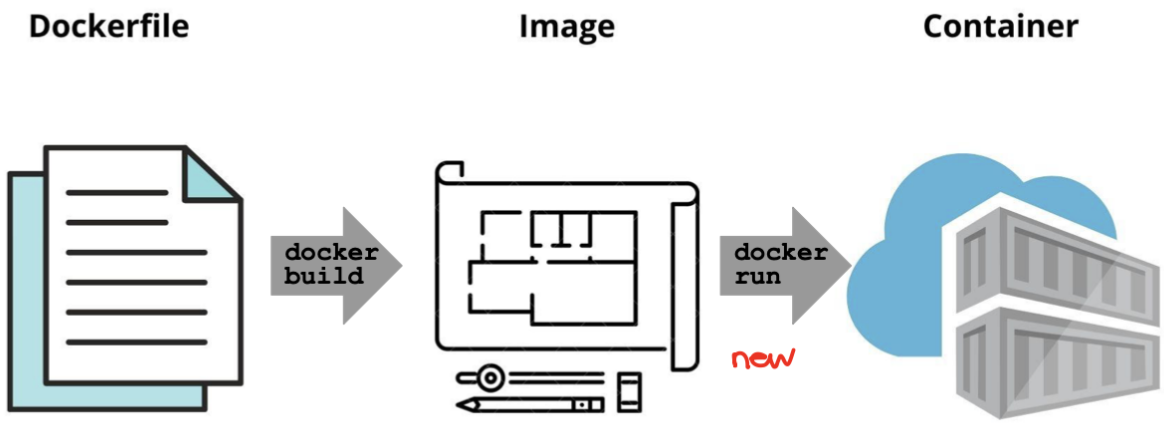


Diagram from Warren's Lecture this morning

```
FROM python:3
# installs an existing image

ENV PYTHONUNBUFFERED 1
# create environmental variables that allows us to see python logs

WORKDIR /app
# creates a directory on our container to put files

COPY requirements.txt requirements.txt
# copies files from our local project folder to our container

RUN pip install -r requirements.txt
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```




```
docker build -f Dockerfile.dev . -t example-image
```

- -f <file name>
- -t <tags this image, and gives it a name>

```
docker run -p 8000:8000 -v "$(pwd):/app" example_image
```

- -p <port mapping>
- -v <setting the volume>

```
docker run –name example_container example_image
```

- —name <give the container a name>

```
docker volume create conference-go-db
```

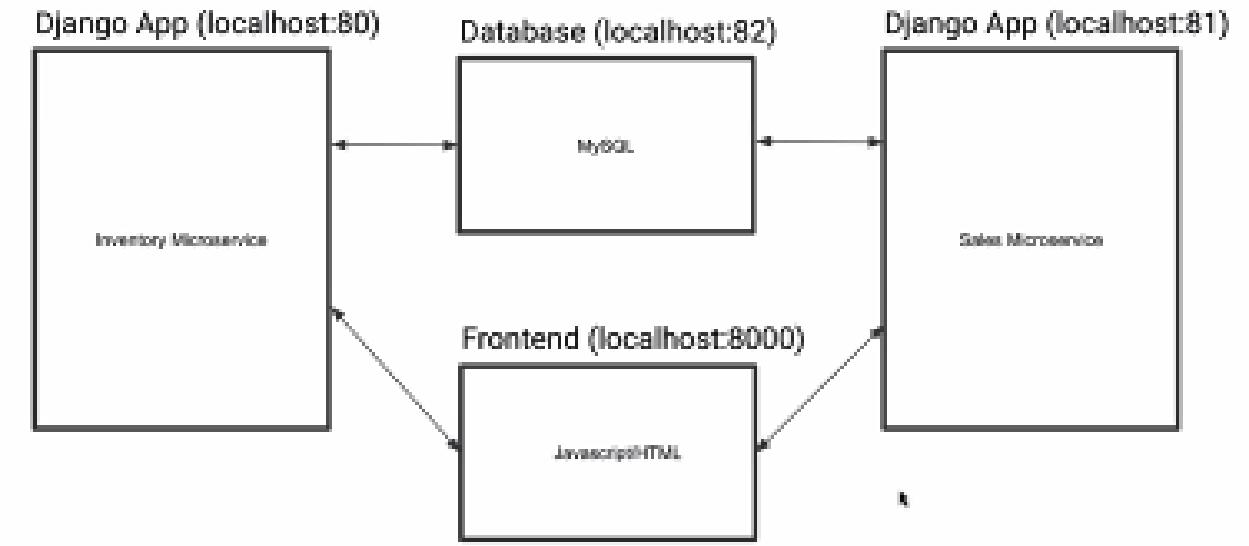
Create a volume called “conference-go-db”

```
docker ps
```

List all the containers you have running!

```
docker container prune
docker image prune
```

Get rid of any containers or images that you are not using!



Installing Docker

Mac

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Apple Silicon (M1/M2) command:

`docker build --platform linux/amd64 -t getting-started .`

Having trouble? Add:

```
--platform linux/amd64
```

Windows

Get Docker Desktop!

> https://www.youtube.com/watch?v=4xK-zaCRiPQ

Anatomy of Dockerfile

FROM

- `FROM <image>`
- `FROM` must be the first non-comment instruction in the Dockerfile.

RUN

- `RUN <command>` (shell form, the command is run in a shell, which by default is `/bin/sh -c` on Linux or `cmd /S /C` on Windows)
  - Run is an image build step, the state of the container after a `RUN` command will be committed to the container image. A Dockerfile can have many `RUN` steps that layer on top of one another to build the image.

WORKDIR

- `WORKDIR </path/to/workdir>`

• Sets the working directory for any `RUN`, `CMD`, `ENTRYPOINT`, `COPY`, and `ADD` instructions that follow it.

CMD

• `CMD <command> <param1> <param2>` (shell form) is the command the container executes by default when you launch the built image.

You can only have one CMD in a Dockerfile. A Dockerfile will only use the final `CMD` defined.

The `CMD` can be overridden when starting a container with `docker run $image $other_command`

COPY

• `COPY <src> [<src> ...] <dest>`

• Copies new files or directories from `<src>` and adds them to the filesystem of the image at the path `<dest>`.

VOLUME

• `VOLUME ["<path>", ...]`

ENV

• `ENV <key> <value>`

Passes in environment variables into your Docker container.

Gunicorn

Gunicorn is a pure-Python **HTTP server** for WSGI applications. It passes the request from a web server to your application. It allows you to run any Python application

concurrently by running multiple Python processes. **It’s fast, stable, and simple to configure.**

WSGI - Web Server Gateway Interface: a simple standard calling convention for web servers to forward requests to web applications or frameworks written in Python. for standard web frameworks like Django or Flask.

Link to *gunicorn (20.1.0)* below:

gunicorn

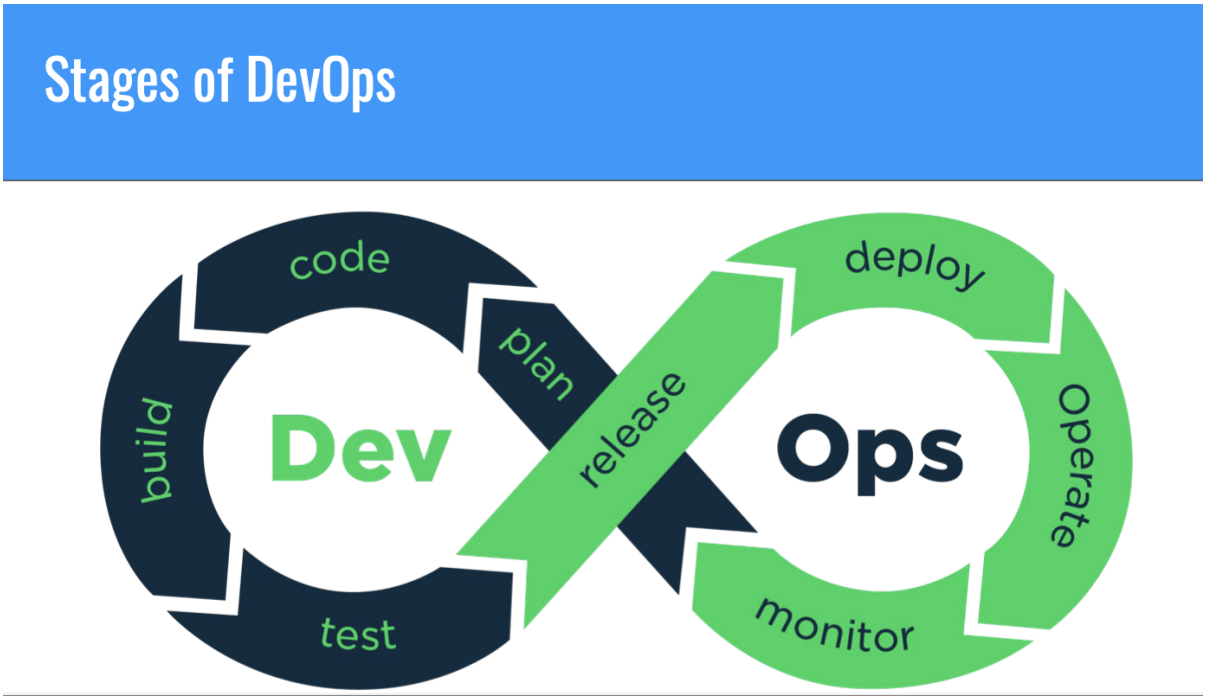
Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX. It's a pre-fork worker model ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with various web

https://pypi.org/project/gunicorn/

In our Lab:

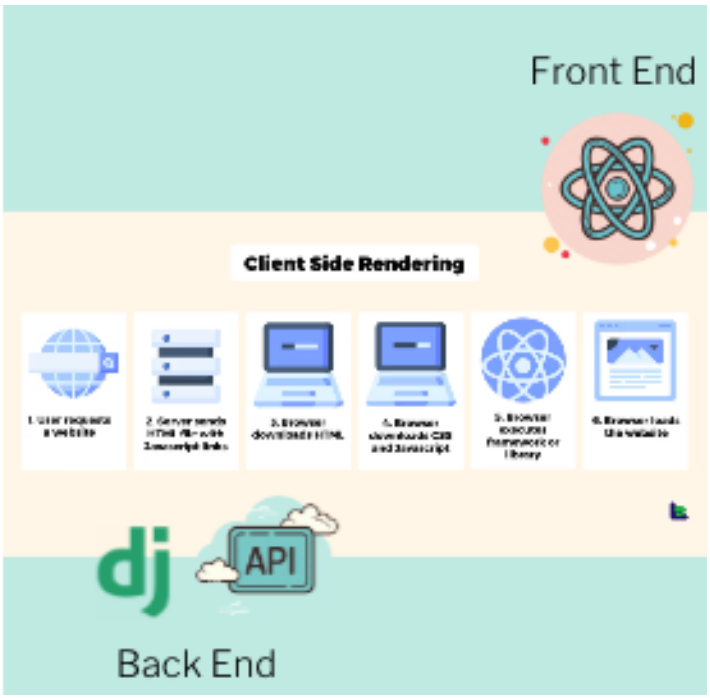
- For deployment Dockerfiles, we use **gunicorn** so that it's as fast as possible.
- For development Dockerfiles, we use the Django development server so that it will restart when we make changes, and show us Yellow Pages of Sadness so we can figure out any errors.

Docker *makes it possible for us to have development and deployment containers for CI/CD — continuous integration / continuous deployment:*



Course Preview:

How You Will Use Docker with React



Dockerfile

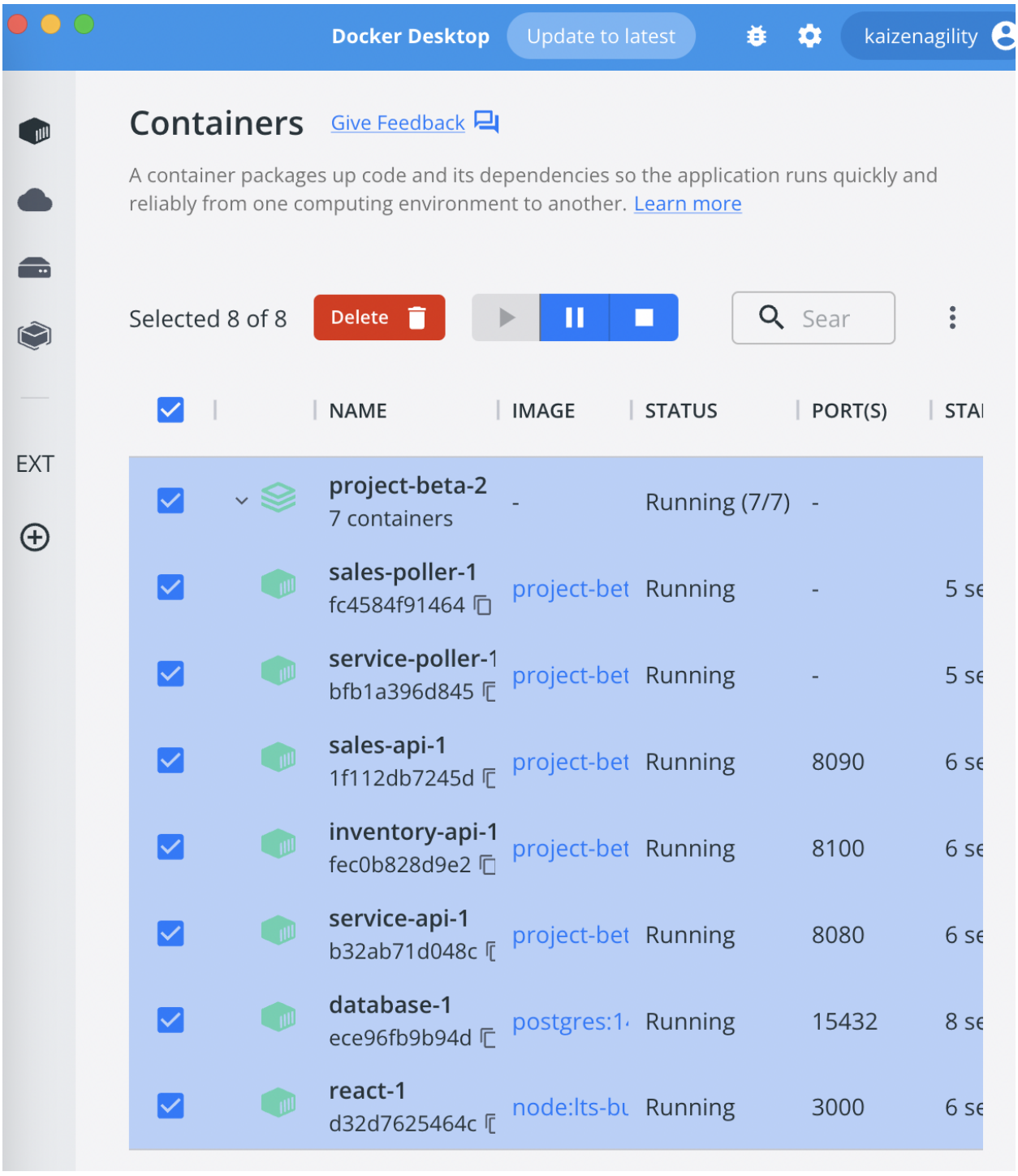
```
# Use the latest version of Node.js
FROM node:latest

# Use the /app directory as the work directory
WORKDIR /app

# Install the tool to create React apps
RUN npm install create-react-app
```

Terminal

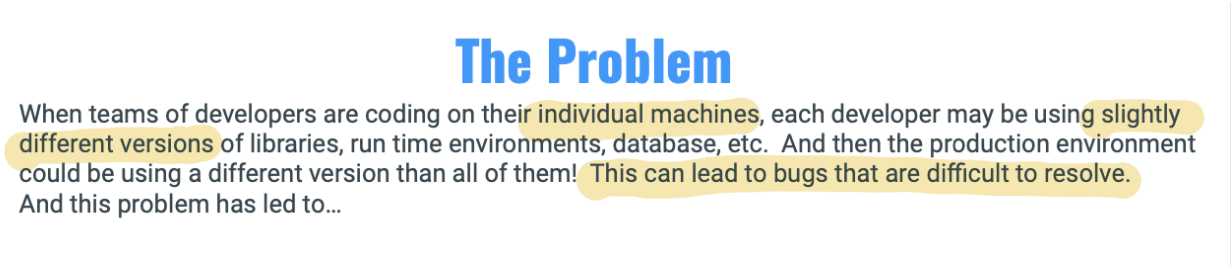
```
docker build . -t react-docker
```

```
docker run -it -v "$(pwd):/app" -p 3000:3000 react-docker bash
```

```
npx create-react-app example
cd example
npm start
```


Docker Desktop
Update to latest
kaizenagility
Containers
Give Feedback
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more
Selected 8 of 8
Delete
Sear
EXT
NAME
IMAGE
STATUS
PORT(S)
STA
project-beta-2
7 containers
-
Running (7/7)
-
sales-poller-1
fc4584f91464
project-bet
Running
-
5 se
service-poller-1
bfb1a396d845
project-bet
Running
-
5 se
sales-api-1
1f112db7245d
project-bet
Running
8090
6 se
inventory-api-1
fec0b828d9e2
project-bet
Running
8100
6 se
service-api-1
b32ab71d048c
project-bet
Running
8080
6 se
database-1
ece96fb9b94d
postgres:1
Running
15432
8 se
react-1
d32d7625464c
node:lts-bu
Running
3000
6 se

EXAM Hint



…the creation of Docker.

One major benefit of using Docker is the standardization of development environments for your entire team. All developers across all teams will be developing with the same OS within that virtual environment, same language environments, same system libraries -- everything is the same regardless of the differences between the host machines' OS and production environment. There will be no surprises between developers' machines.

Why Use Docker

Containers as the foundation for DevOps collaboration

KEY TERMS:

- **Containerization** allow us to keep our entire ecosystem in sync across our team and infrastructure
- **Images** are templates for containers that can be shared from a central repository
- **Volumes** allow us to mount directories on our host filesystem in the container so we can share files and persist data
- **Docker Hub** is a central repository for images
- We can automate spinning up containers with **docker-compose**

RESOURCES

Docker commands cheatsheet:

https://dockerlabs.collabnix.com/docker/cheatsheet/

Dockerfile cheatsheet:
https://medium.com/@oap.py/dockerfile-cheat-sheet-4ad12569aa0b

Docker curriculum tutorial:

https://docker-curriculum.com

Got More Questions?

Add below as comments: