**Week 15**

**15.1 Docker Intro**

* Container 🡪 package many things as well as code and give it to someone to run.
* We can give the same container to multiple operating systems.
* docker-compose.yaml and Dockerfile are used when setting up open source project.
* hub.docker.com has all the images. It’s like GitHub but GitHub has codebases, this has images. This is called Docker registry.
* Imagine images as source code and the container as running the code. We can run the container many times. Run instances of the same image many times.
* Port mapping is important as we want to tell that please send request 27017 to mac machine to mongo container’s 27017.
* We don’t need to add port mapping if we don’t need to expose it over the internet.
* -d runs the container in the background so that the terminal becomes free.
* docker kill [container\_id] to kill the container.
* docker rmi mongo –force to remove mongo image.
* docker build would be used when we would create our own containers.
* Dockerfile is used when we want to containerize our project.
* Whatever you write after CMD, runs after you start the container. Everything else runs after you create the image.
* docker build -t backend-app .
* Do not use docker for database in production.
* docker run -p 3000:3000 backend-app
* docker run -p 3000:3000 -e DATABASE\_URL="postgres://avnadmin:AVNS\_EeDiMIdW-dNT4Ox9l1n@pg-35339ab4-harkirat-d1b9.a.aivencloud.com:25579/defaultdb?sslmode=require" image\_name
* This is how the database URL is passed as an environment variable when running the program using docker method.
* DATABASE\_URL=postgres://avnadmin:AVNS\_EeDiMIdW-dNT4Ox9l1n@pg-35339ab4-harkirat-d1b9.a.aivencloud.com:25579/defaultdb?sslmode=require node dist/index.js
* This starts the program normally with the database URL injected.
* docker exec -it a7aa736d226f /bin/sh to ssh into your container.

**15.2 Docker 2**

* Layers, Networks, Volumes, Docker Compose
* An image is made up of multiple layers. Basically, each line represents a new layer.
* Benefit of layers is caching.
* If a layer is cached then all layers above that are cached.
* If you change index.ts, then COPY . . command would not be cached.
* Package.json and the prisma folder don’t get changed very often. So, we can copy over them first so that they can be cached.
* More layers doesn’t mean more computationally expensive.
* Containers are transitory. If you’ll kill the container, the data is lost. That is why volumes are used.
* If we want containers to talk to each other, then they need to be part of the same network.
* When we start a mongo container, put some documents and if the container is killed in the future, then all the data is lost.
* docker volume create volume\_database
* docker run -v volume\_database:/data/db -p 27017:27017 mongo
* Networks allow containers to talk to mac machines.
* When containers are talking to each other using networks we do not need port mapping as port mapping allows conversation to the mac machine.
* The name you give to the mongo container should be put in the connection url of your node.js process.

**15.3 Docker-Compose, Push, and Exec**

* When you know that you would be pushing the image to docker hub then when building the image use your\_usersname/image\_name convention.
* The tag that gets attached by default is the latest tag.
* Creating custom tag: docker build -t your\_usersname/image\_name:custom\_tag .
* Imagine tag as version numbers.
* Yaml and json are both key value pairs.
* docker-compose lets you start multiple containers.
* services have all the run commands.
* Whenever you multiple services in the same docker-compose, they are by default attached to the same network.
* docker-compose up
* Stop everything (including volumes) docker-compose down –volumes
* backend:  
   build: .
* To let docker-compose build the current Dockerfile in the directory.

**15.4 Bind Mounts**

* Bind mounts binds a folder in your mac machine to a folder in your container. This was used in pre-volumes era.
* When running next-app inside a container, then any changes you make to the codebase would not be reflected as the Dockerfile copied over the contents once. Basically, hot reloading stops working.
* Bind mount can help in this case.
* docker run -p 3000:3000 -v .:/nextapp nextapp
* .:/nextapp this means mounting a folder from my mac machine to the folder in the container. Any changes in the mac machine’s folder would propagate changes in the container’s folder.