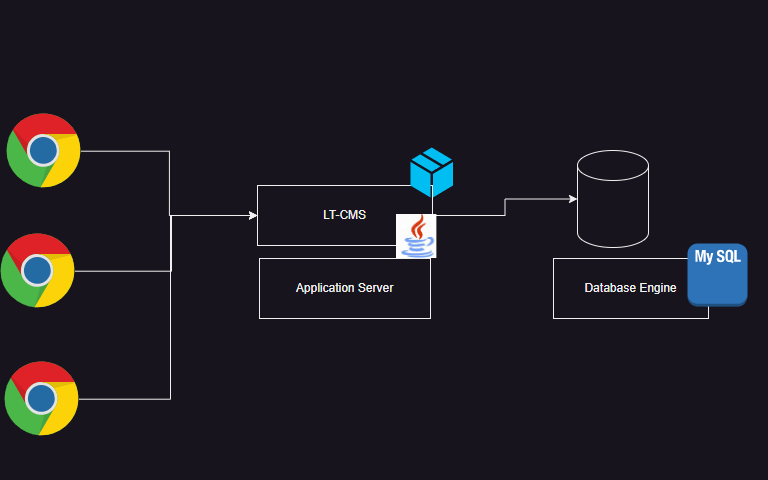
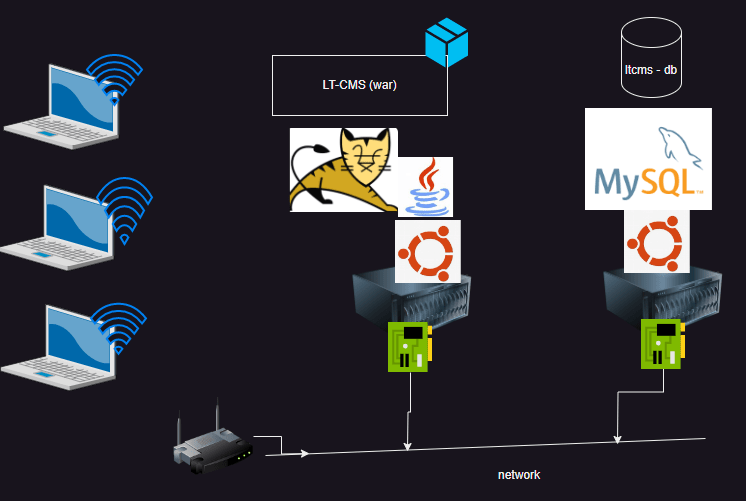
October 1, 2024

DevOps Classroom notes 01/Oct/2024

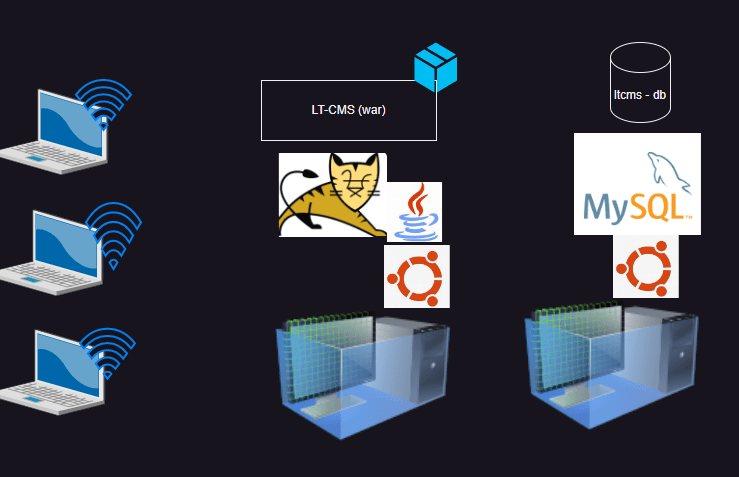
**Sample Application Architecture (College Management System)**

* Overview  
  
* This application has a database (mysql)
* This application is developed on Java and it needs tomcat /weblogic/websphere server to the run the application
* Users access the application over the browser
* This architecture is also referred as monolith

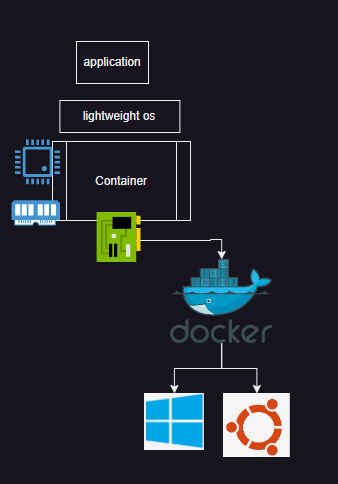
**Generation 1: Physical server deployment**

* The deployment of this in the first generation is on physical servers  
  
* Cons:
  + Ineffective Capex (Capital expenditure)
  + Resource utilization is mostly underusage

**Generation 2: Hypervisors**

* With Hypervisors now we can create multiple virtual machines from a single physical server  
  

**Generation 3: Container**

* Container gives an isolated space for the applications to run  
  

**Share this:**

October 2, 2024

# DevOps Classroom notes 02/Oct/2024

## What is a container

* Container is a isolated area. This has everything required to run the [application](https://directdevops.blog/2024/10/02/devops-classroom-notes-02-oct-2024/)
* Containers are lightweight and can be scaled
* To run a [docker](https://directdevops.blog/2024/10/02/devops-classroom-notes-02-oct-2024/) container we need docker image
* Docker image contains all the necessary dependencies to run your application
* All the images which can be used are stored in Registry
* Every Docker will need a registry to get the images
* Default Registry is docker hub which is public
* Organizations will have private Registries where they will be storing application images
* Next Steps:
  + Explore running containers
  + Containerizing Applications (Running applications build by developers inside containers)

#### Exploring the Installations

* Windows:
  + Ensure you have a non home edition windows 11 or 10 [Refer Here](https://docs.docker.com/desktop/install/windows-install/)
* Mac: [Refer Here](https://docs.docker.com/desktop/install/mac-install/)
* Linux: [Refer Here](https://get.docker.com/)

curl -fsSL https://get.docker.com -o install-docker.sh

sudo sh install-docker.sh

* Exploring Docker
  + Create an account in Docker Hub [Refer Here](https://hub.docker.com/)
  + Docker Playground

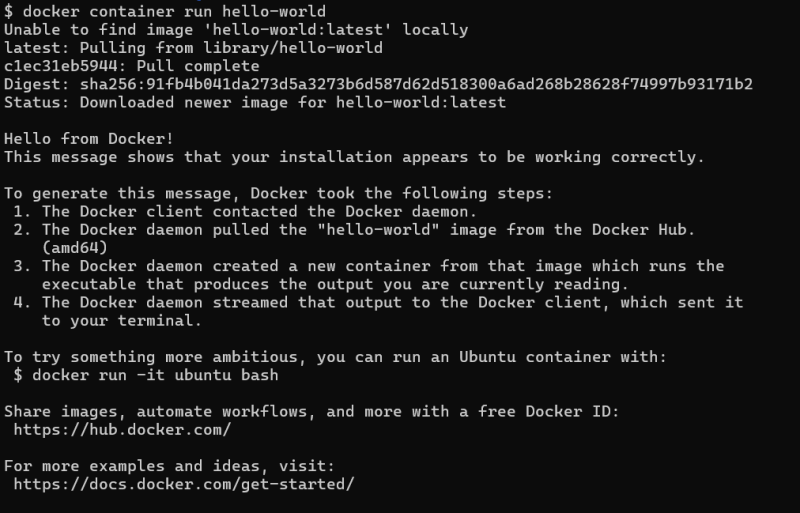
#### Running containers based on popular images

* Verify the docker is working or not

docker info

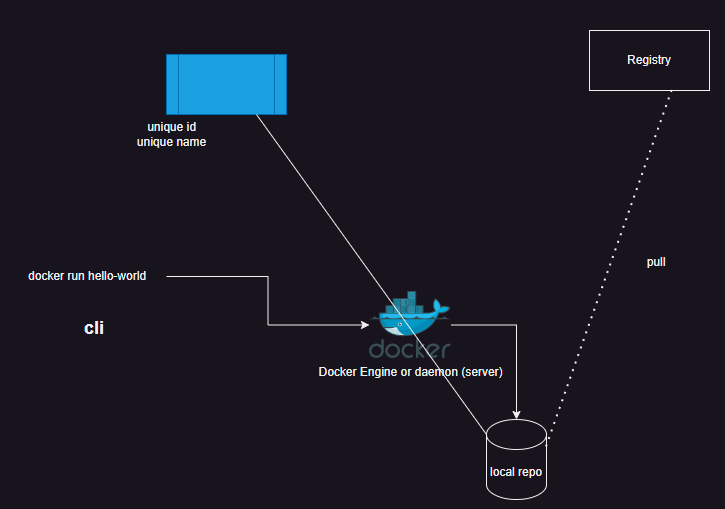
* Run our first container

docker run hello-world

  
\* Lets see what is the status of this contianer

docker container ls

docker container ls -a



### I want to run a java application (spring petclinic)

* [Refer Here](https://khajareferenceapps.s3.ap-south-1.amazonaws.com/spring-petclinic-3.2.0-SNAPSHOT.jar) for java app
* This application requires java 17
* This application uses port 8080
* to run this application the command would java -jar spring-petclinic-3.2.0-SNAPSHOT.jar

October 3, 2024

DevOps Classroom notes 03/Oct/2024

[**Docker**](https://directdevops.blog/2024/10/03/devops-classroom-notes-03-oct-2024/)**Installation**

* For Guided creation of ec2 instance watch classroom recording
* Docker [installation](https://directdevops.blog/2024/10/03/devops-classroom-notes-03-oct-2024/)
  + Simplest  
    bash  
    curl -fsSL https://get.docker.com -o install-docker.sh  
    sh install-docker.sh
  + NOw add the current user to docker group  
    “`

</ul>  
<h1>sudo usermod -aG docker $USER</h1>  
sudo usermod -aG docker ubuntu  
“`

* + When we install docker the following components will be installed
    - docker ce (docker daemon)
    - docker ce cli (docker client)
    - containerd
    - docker compose
    - docker buildx

**Docker Package based installation**

* + [Refer Here](https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository) for ubuntu steps

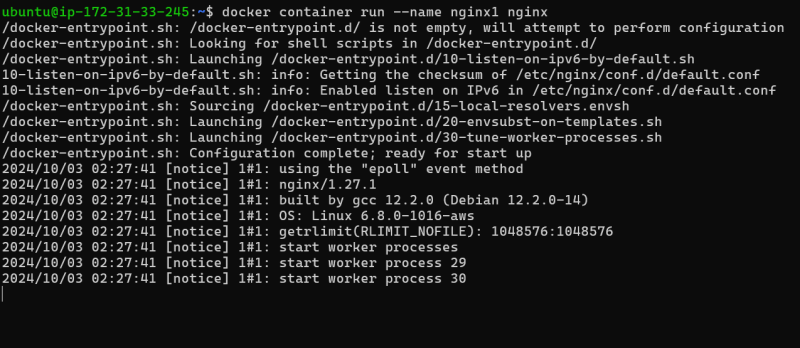
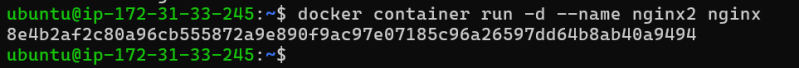
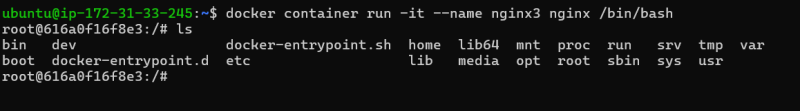
**Docker Images**

* + Docker images have tags which represent versions. If you don’t pass tag, docker will assume it to be latest

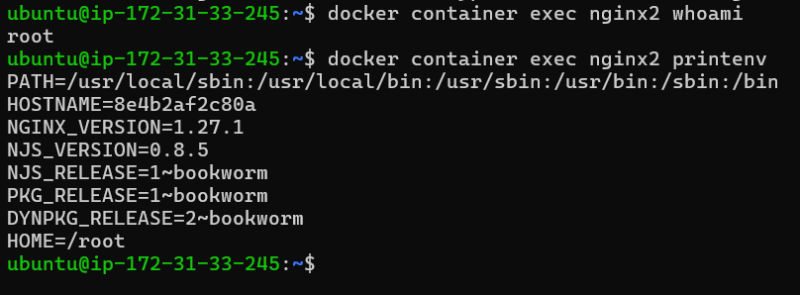
docker pull nginx

* + Docker images from docker hub
    - official images <application>:<tag>
    - community images organization/application:tag
    - private images <username>/application:tag

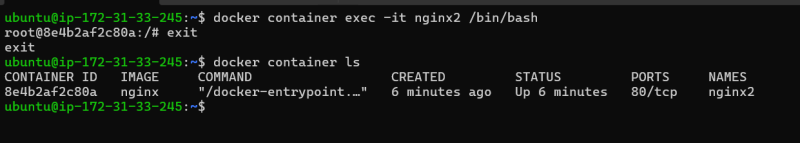
**Docker container creation modes**

* + Attached/Foreground  
    
  + Detached  
    
  + interactive termninal: you want to get into shell while creating container  
    

**Execution in running contianers**

* + Docker has a command called as exec  
    
  + We can use exec to run a command inside container
  + we can also use exec in an interactive way

docker container exec -it nginx2 /bin/bash



**Observations:**

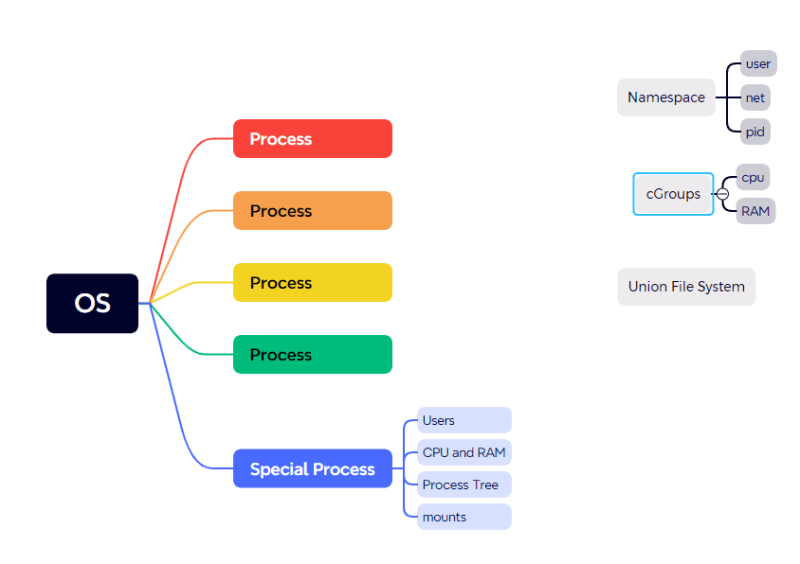
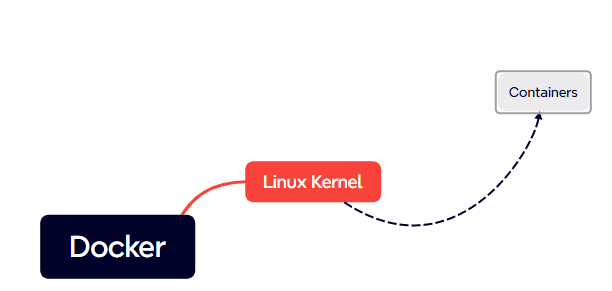
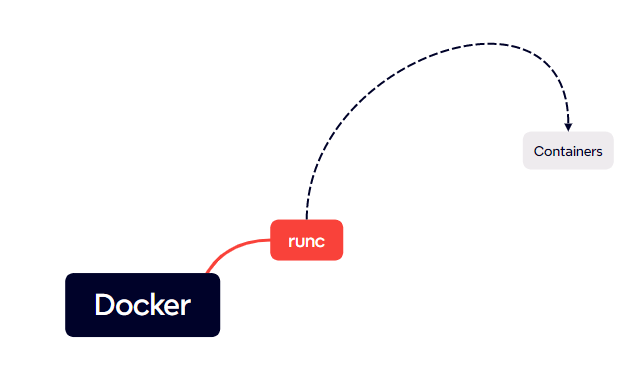
* + Inside containers, the PID 1 is the application startup
  + Container will have its own process tree
  + Contianer will have its own network, diskmounts, CPU and RAM allocated
  + Restrictions can be placed on RAM and CPU
  + Inside containers we have its own set of users

**Share this:**

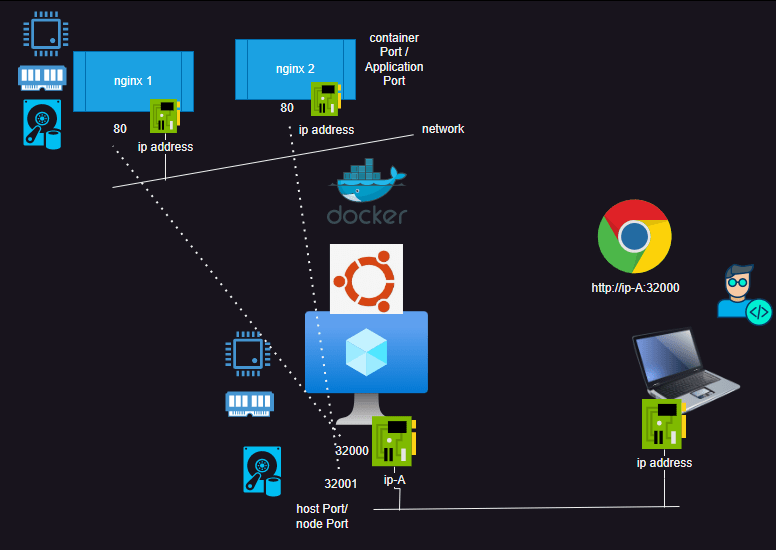
October 5, 2024

DevOps Classroom notes 05/Oct/2024

**How Docker Works ?**

* Container is an isolated space which will have its own [Refer Here](https://directdevops.blog/2019/01/31/docker-internals/)
  + process tree with the help of pid namespace
  + disk mounts with the help of mnt namespace
  + users with the help of usr namespace
  + network with the help of net namespace
  + CPU and RAM : with the help of [cGroups](https://directdevops.blog/2024/10/05/devops-classroom-notes-05-oct-2024/)  
    
* Earlier Generation of Container creation incuded
  + [docker engine](https://directdevops.blog/2024/10/05/devops-classroom-notes-05-oct-2024/) directly interacting wiht linux kernel to create containers  
    
* Docker has implemented runc to create containers  
  
* OCI (Open container initiative was formed ) and the architecture of Docker has changed from monolith to layered

**Port forwarding**

* Overview  
  
* Let me create three applications in two containers each
  + nginx (it runs on port 80 )
  + jenkins (it runs on port 8080)
  + mysql container (it runs on port 3306)

docker container run -d --name nginx1 nginx

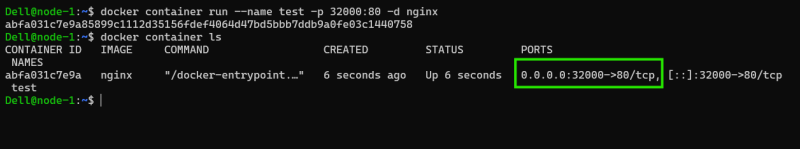
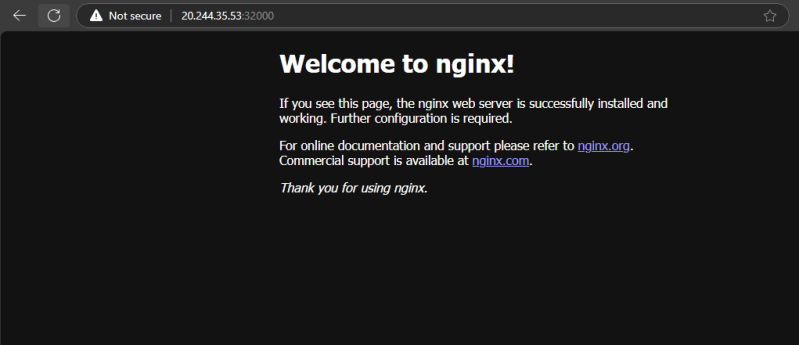
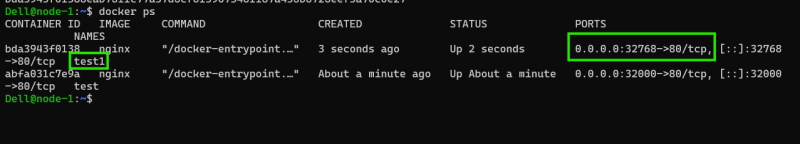
docker container run -d --name nginx2 nginx

docker container run -d --name jenkins1 jenkins/jenkins

docker container run -d --name jenkins2 jenkins/jenkins

docker container run -d --name mysql1 -e MYSQL\_ROOT\_PASSWORD=password mysql:8.0

docker container run -d --name mysql2 -e MYSQL\_ROOT\_PASSWORD=password mysql:8.0

  
\* The above containers are not userful for me as i cannot access them from outside the vm.  
\* Delete all the containers docker container rm -f $(docker container ls -q -a)  
\* Delete all the image docker image rm $(docker image ls -q)  
\* For port forwarding we have two modes  
\* static -p <hostport>:<containerPort>  
  
  
\* Dynamic -P  
  
\* Lets apply dynamic Port forwarding

docker container run -d -P --name nginx1 nginx

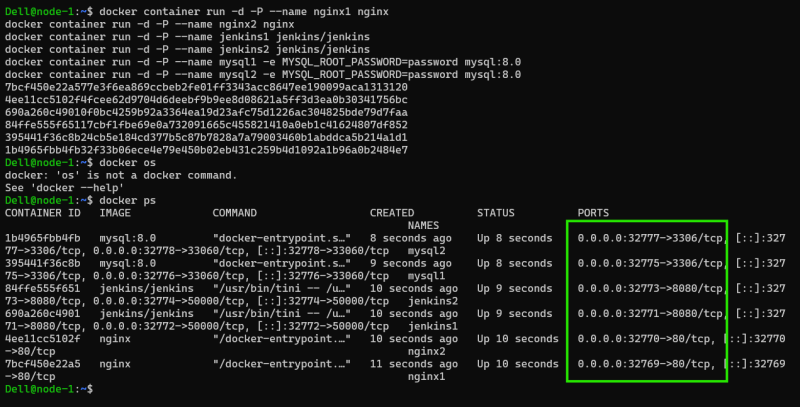
docker container run -d -P --name nginx2 nginx

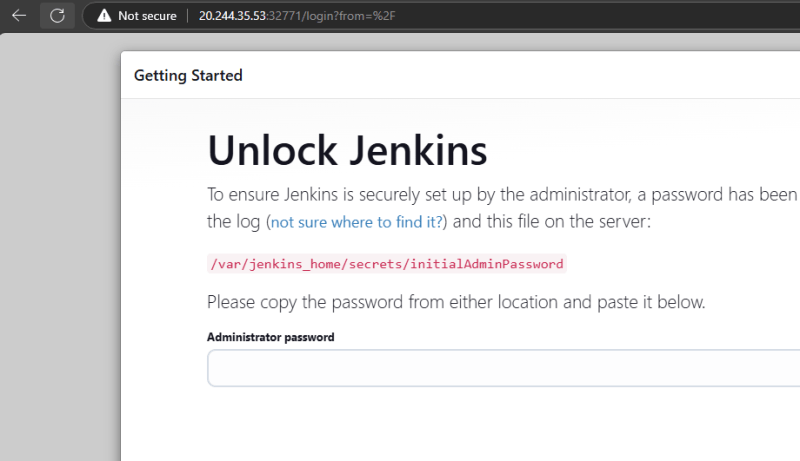
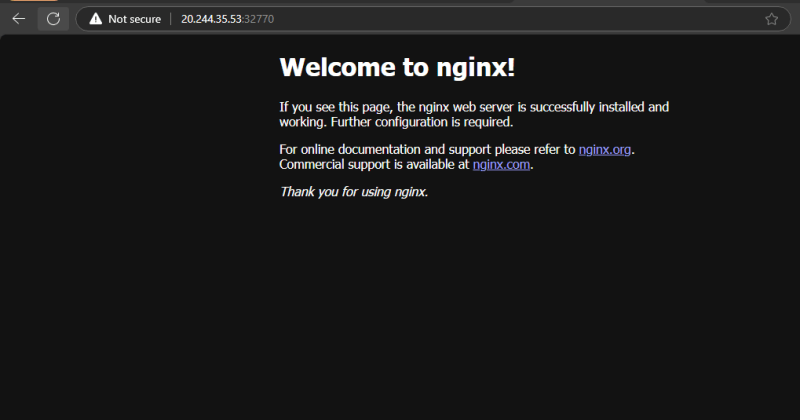
docker container run -d -P --name jenkins1 jenkins/jenkins

docker container run -d -P --name jenkins2 jenkins/jenkins

docker container run -d -P --name mysql1 -e MYSQL\_ROOT\_PASSWORD=password mysql:8.0

docker container run -d -P --name mysql2 -e MYSQL\_ROOT\_PASSWORD=password mysql:8.0



* Lets connect to jenkins  
  
* Lets connect to nginx  
  
* Lets connect to mysql

**Share this:**

* [Click to share on Twitter (Opens in new window)](https://directdevops.blog/2024/10/05/devops-classroom-notes-05-oct-2024/?share=twitter&nb=1)
* [Click to share on Facebook (Opens in new window)](https://directdevops.blog/2024/10/05/devops-classroom-notes-05-oct-2024/?share=facebook&nb=1)
* [Click to share on WhatsApp (Opens in new window)](https://directdevops.blog/2024/10/05/devops-classroom-notes-05-oct-2024/?share=jetpack-whatsapp&nb=1)

October 6, 2024

DevOps Classroom notes 06/Oct/2024

**Containerize Applications**

* Making [applications](https://directdevops.blog/2024/10/06/devops-classroom-notes-06-oct-2024/) run in the container is referred as containerizations i.e. Building images with our [app](https://directdevops.blog/2024/10/06/devops-classroom-notes-06-oct-2024/) in it
* For doing this we have two approaches
  + Manual Process
  + Dockerfile Approach

**Activity 1: Create a website**

* Template: [Refer Here](https://www.free-css.com/free-css-templates/page296/finexo)
* Websites require webservers, We can use any webserver (nginx/apache)
  + We need nginx
* Manual steps to create website on a server (ubuntu)

sudo apt update

sudo apt install nginx -y

sudo apt install unzip -y

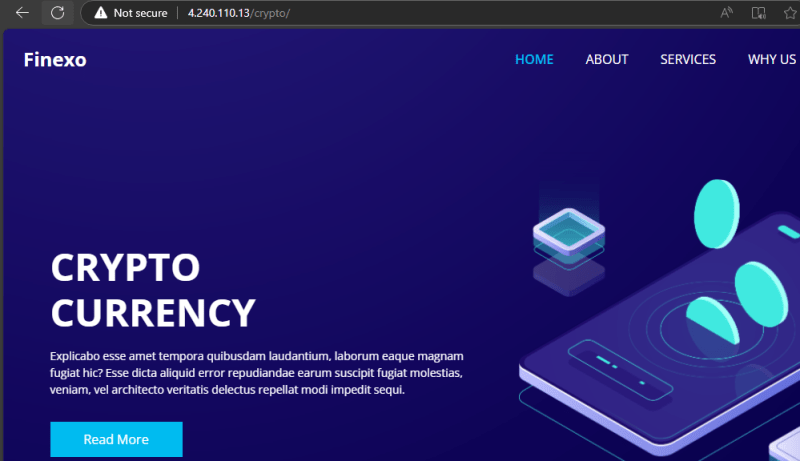
cd /tmp

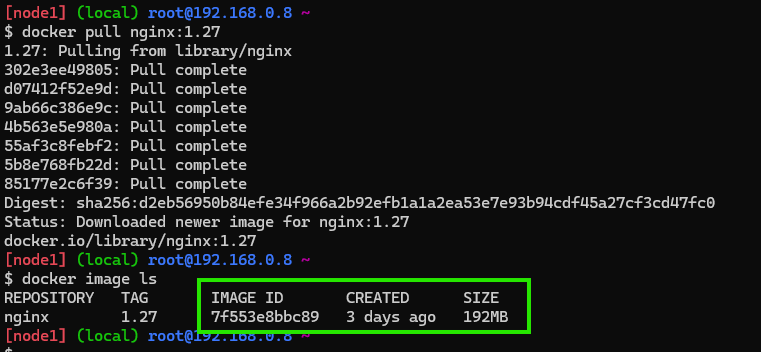
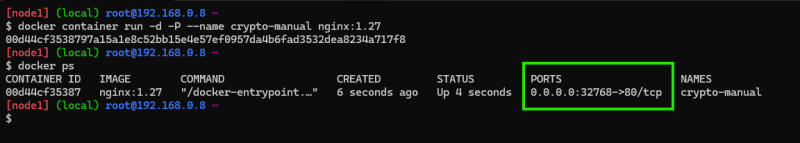
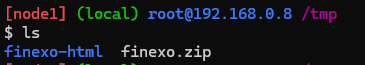
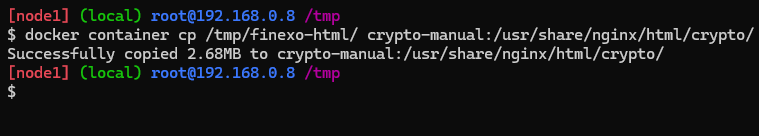
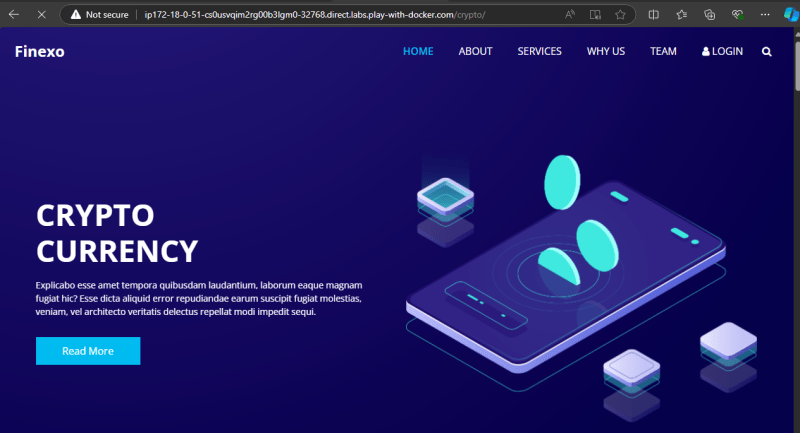
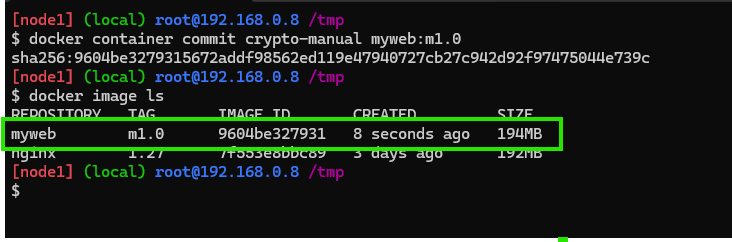
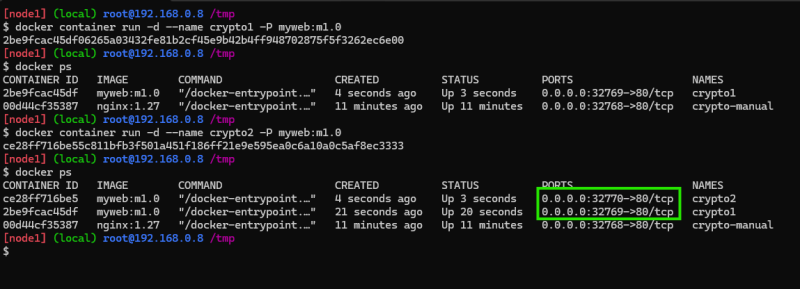
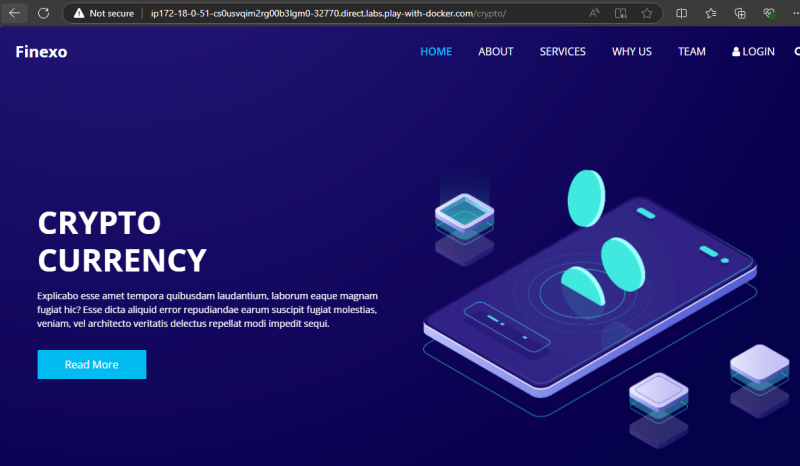
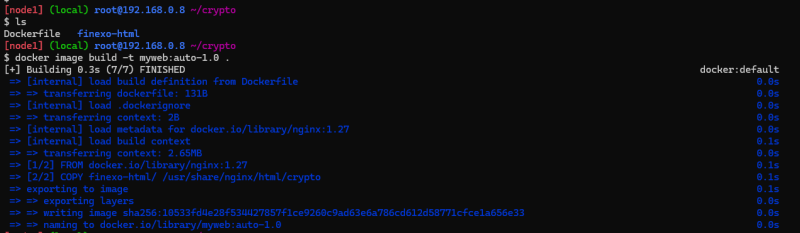
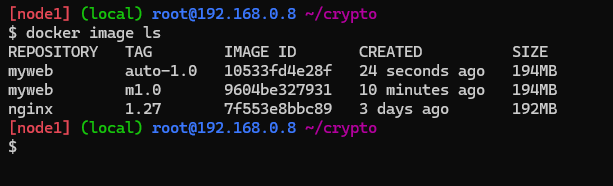
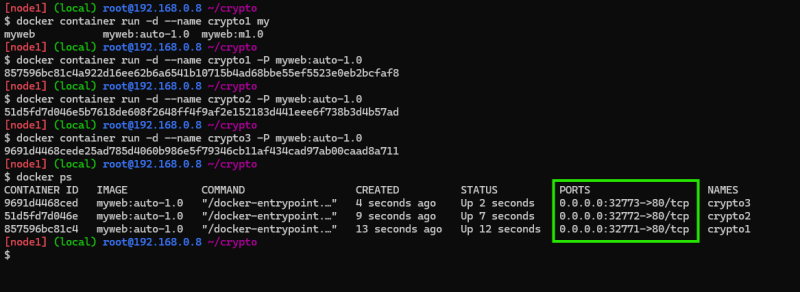
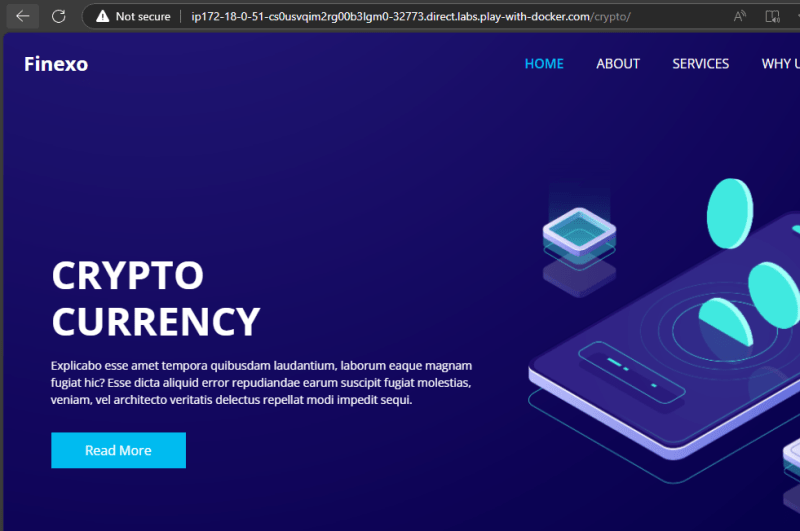
wget https://www.free-css.com/assets/files/free-css-templates/download/page296/finexo.zip

unzip finexo.zip

* All the websites have a folder from which websites are served

sudo mv /tmp/finexo-html /var/www/html/crypto



* Manual approach in docker
  + Lets pull the nginx image nginx:1.27 and view details  
    
  + Now run the container in the detached mode with port exposed  
    docker container run -d -P --name crypto-manual nginx:1.27  
    
  + Download the zip file locally cd /tmp && wget https://www.free-css.com/assets/files/free-css-templates/download/page296/finexo.zip  
    
  + Now copy the folder into nginx container  
      
    
  + Now the container crypto-manual has application running inside, we need to create a image of this container  
    
  + Now lets create a new container from the image  
    docker container run -d --name crypto1 -P myweb:m1.0  
      
    
* Dockerfile Approach
  + Create a file called as Dockerfile in a folder with following content  
    Dockerfile  
    FROM nginx:1.27  
    LABEL author='khaja'  
    EXPOSE 80  
    COPY finexo-html/ /usr/share/nginx/html/crypto
  + Now build the image using command docker image build -t myweb:a1.0 .  
      
    
  + Now create some containers and cross check  
      
    
* Dockerfile approach is widely used to create container images as it is repeatable and automatable.

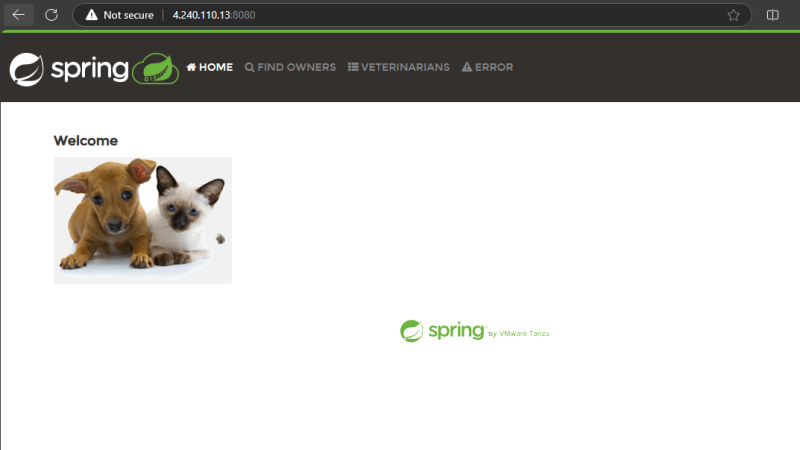
**Activity 2: Run a Spring boot application inside container (Containerizing Spring boot application.)**

* [Refer Here](https://khajareferenceapps.s3.ap-south-1.amazonaws.com/spring-petclinic-3.2.0-SNAPSHOT.jar) for springpetclinic jar file
* To run this application we need java 17 jdk

cd /tmp

wget https://khajareferenceapps.s3.ap-south-1.amazonaws.com/spring-petclinic-3.2.0-SNAPSHOT.jar

java -jar /tmp/spring-petclinic-3.2.0-SNAPSHOT.jar



* What is required to run this application
  + java 17
* Java comes with two different installations
  + JDK: This is required to build and run the java applications
  + JRE: This is required to run the java applications

**Activity 3: Run a reactjs application**

* Sample application [Refer Here](https://github.com/aditya-sridhar/simple-reactjs-app)
* Install nodejs and npm
* manual steps

git clone https://github.com/aditya-sridhar/simple-reactjs-app.git

cd simple-reactjs-app

npm install

npm run start

**Approach**

1. Figure out the manual steps on how to deploy your applicaton onto server (os level)
2. Ensure the dependencies (softwares required to run application)
3. Now Create a Dockerfile with instructions to containerize your application.

October 8, 2024

DevOps Classroom notes 08/Oct/2024

**Dockerfile**

* [Refer Here](https://docs.docker.com/reference/dockerfile/) for official docs
* Dockerfile contains set of instructions required to build the image

INSTRUCTION args

* examples

FROM nginx

FROM openjdk:17

RUN

COPY . /app

* Each Instruction has a purpose
* Lets look at widely used instructions
  + FROM [Refer Here](https://docs.docker.com/reference/dockerfile/#from) for officials docs & FROM is used to choose the base image
  + LABEL: This is used to add the metadata
  + RUN: This instruction executes the commands written while building the image
  + EXPOSE: publishes the port information
  + ADD/COPY: They are used to copy the files into [docker](https://directdevops.blog/2024/10/08/devops-classroom-notes-08-oct-2024/) image
  + CMD: This will be executed as PID1 i.e. when we start the container this command executes and container will be in running state as long as the PID1 is running
* For experiments done kindly refer classroom video

**Shell form and Exec Form**

* [Refer Here](https://docs.docker.com/reference/dockerfile/#shell-and-exec-form) for official docs
* Widely most of the Dockerfiles contains
  + RUN in shell FORM
  + CMD and Entrypoint in EXEC FORM

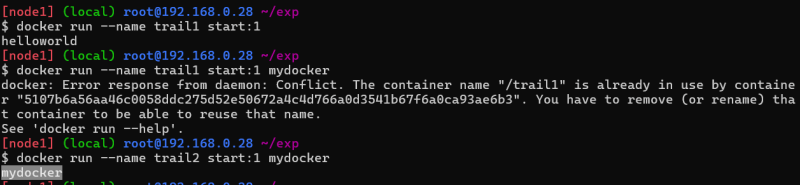
**Docker Image**

* [Docker](https://directdevops.blog/2024/10/08/devops-classroom-notes-08-oct-2024/) Image should contain all the files necessary to run application
* Every Docker Image has a unique image id and name and tag
* Docker Image is collection of Readonly Layers
* A Layer is created during image creation by any instruction which leads to changes in the disk contents (RUN, ADD, COPY)
* Best Practice is to create few or less reusable layers

October 9, 2024

DevOps Classroom notes 09/Oct/2024

**Dockerfile (Containerization contd…)**

* CMD and ENTRYPOINT
  + When a [docker](https://directdevops.blog/2024/10/09/devops-classroom-notes-09-oct-2024/) container starts it executes
    - Whatever is present in entrypoint and what ever has been written in CMD will be arguments to ENTRYPOINT
    - If entrypoint is present but CMD is not present in Dockerfile then whatever is written in ENTRYPOINT will be executed
    - IF ENTRYPOINT is not found and ony CMD Exists then CMD will be executed
  + CMD can be overriten by passing arguments after image name in docker run command  
    
  + IF We donot write entrypoint or CMD the base images entrypoint or CMD will be considered
* Your container will be in running state as long as PID1 (ENTRYPOINT + CMD) is running
* Ideally anything can go in CMD or entrypoint, But in practice we execute the commands that start our [application](https://directdevops.blog/2024/10/09/devops-classroom-notes-09-oct-2024/) and wait till the application is executing.
* Typical startup commands
  + java:
    - spring boot: java -jar <jarfile path>
    - app server: if your app runs on some external server then take the base image of the app server and dont write CMD or ENTRYPOINT
  + react js/angular js/vue js
    - first option: npm run start or equivalent
    - best option is to create a html site and run in nginx
  + dotnet (asp.net core)
    - dotnet <dll path>
  + Python
    - flask python app.py or server based startup
    - django python app.py or server based startup
    - fastapi uvicorn main:app --host 0.0.0.0 ...

**Other instructions**

* [Refer Here](https://docs.docker.com/reference/dockerfile/) for official docs for reference
* WORKDIR
* USER

**Lets Build a image for Python fastapi**

* [Refer Here](https://github.com/dummyrepos/Fast-Api-example/tree/main) for the code

**Share this:**

* [Click to share on Twitter (Opens in new window)](https://directdevops.blog/2024/10/09/devops-classroom-notes-09-oct-2024/?share=twitter&nb=1)
* [Click to share on Facebook (Opens in new window)](https://directdevops.blog/2024/10/09/devops-classroom-notes-09-oct-2024/?share=facebook&nb=1)
* [Click to share on WhatsApp (Opens in new window)](https://directdevops.blog/2024/10/09/devops-classroom-notes-09-oct-2024/?share=jetpack-whatsapp&nb=1)

October 10, 2024

DevOps Classroom notes 10/Oct/2024

**Optimizing Images**

**Simple**[**Fastapi**](https://directdevops.blog/2024/10/10/devops-classroom-notes-10-oct-2024/)**application**

* [Refer Here](https://github.com/dummyrepos/SimpleFastAPI/blob/main/README.md) for steps
* Required:
  + [python](https://directdevops.blog/2024/10/10/devops-classroom-notes-10-oct-2024/) 3.13
* Steps:
  + copy code
  + install dependencies
  + Expose port
* How to run Appl uvicorn main:app --host 0.0.0.0 --port 8000 --reload
* [Refer Here](https://github.com/dummyrepos/SimpleFastAPI/commit/d8d130a149675862244de5e3dc1bd1184393119a) for first version of Dockerfile
* [Refer Here](https://github.com/dummyrepos/SimpleFastAPI/commit/d0e876352d9a030f1e01705922c45207630bb277) for change to slim version
* [Refer Here](https://github.com/dummyrepos/SimpleFastAPI/commit/b2f56b74739ea07a9e9ac848b62ae19a519dba4f) for changes to use a non root user
* When we tried as a non root user, we are getting permission issues
* [Refer Here](https://github.com/dummyrepos/SimpleFastAPI/commit/acfc848f3352324a22e7a03531c8c9ab4bda184d) for the possible solution where we build dependencies in one stage and copy them into other with permissions to the user

**Simple Spring boot (Java) application**

* Existing Dockerfile

FROM eclipse-temurin:17-jre

ADD https://khajareferenceapps.s3.ap-south-1.amazonaws.com/spring-petclinic-3.2.0-SNAPSHOT.jar /spc.jar

EXPOSE 8080

CMD ["java", "-jar", "/spc.jar"]

* We are building the jar file seperately and using them in Docker Image building process.
* The idea is to build the jar and use the jar as part of Docker Image building itself

**Multi stage Dockerfiles**

* This is inspired from a builder pattern
* [Refer Here](https://github.com/dummyrepos/docker-spring-petclinic/commit/bde1d884a2354dcc8539fe69939a45490044316d) for multistage docker for spring petclinic

FROM maven:3.9-eclipse-temurin-17 AS builder

# build the java code

COPY . /spc

WORKDIR /spc

RUN mvn package

# this will create a spring petclinic jar file

FROM eclipse-temurin:17-jre AS runner

COPY --from=builder --chown=ubuntu /spc/target/spring-petclinic-3.3.0-SNAPSHOT.jar /app/spring-petclinic.jar

USER ubuntu

WORKDIR /app

EXPOSE 8080

CMD ["java", "-jar", "spring-petclinic.jar"]

**Share this:**

October 11, 2024

DevOps Classroom notes 11/Oct/2024

**Docker Images, Containers and volumes**

* [Refer Here](https://directdevops.blog/2019/09/26/docker-image-creation-and-docker-image-layers/) for article on image layers and [Refer Here](https://directdevops.blog/2019/09/27/impact-of-image-layers-on-docker-containers-storage-drivers/) for impact of image layer and [Refer Here](https://directdevops.blog/2019/10/03/docker-volumes/) for volumes
* Dockerfile instruction VOLUME: [Refer Here](https://docs.docker.com/reference/dockerfile/#volume)

**ARG and ENV**

* ARG can be used as a parameter while building images [Refer Here](https://docs.docker.com/reference/dockerfile/#arg)
* ENV can be used as a parameter while running containers [Refer Here](https://docs.docker.com/reference/dockerfile/#env)

**.dockerignore**

* [Refer Here](https://docs.docker.com/build/concepts/context/#dockerignore-files)

**Lets run a postgres database inside a container**

* Lets create a postgres database with password, user, database and volume mounting (refer classroom video for commands)

docker run --name lib-db -d -e 'POSTGRES\_PASSWORD=password' -e 'POSTGRES\_USER=lt' \

> -e 'POSTGRES\_DB=library' -v lib-vol:/var/lib/postgresql/data postgres:17

**Exercise**

* Run mongodb, mysql and microsoftsqlserver with volumes in a contianer

October 15, 2024

# DevOps Classroom notes 15/Oct/2024

## Connectivity Between Application and Database

### Connection strings

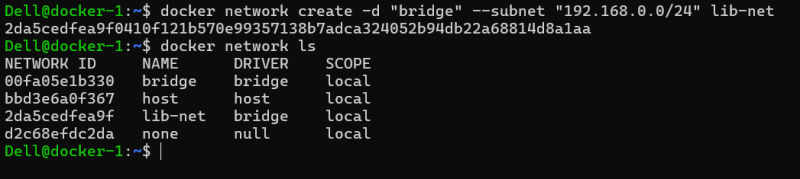
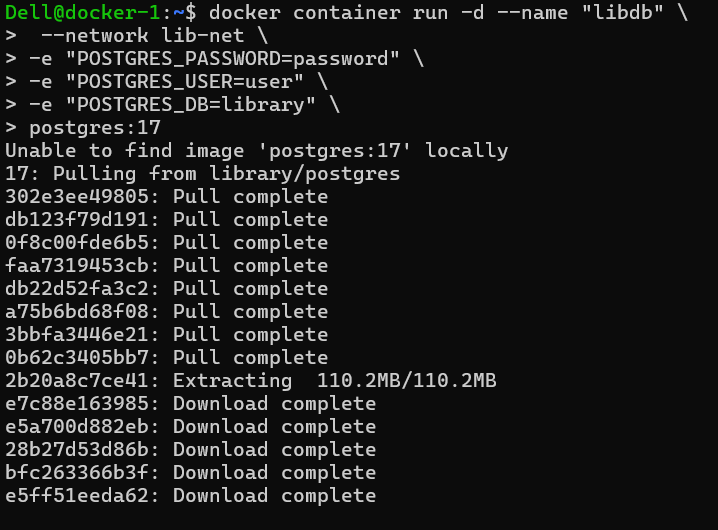
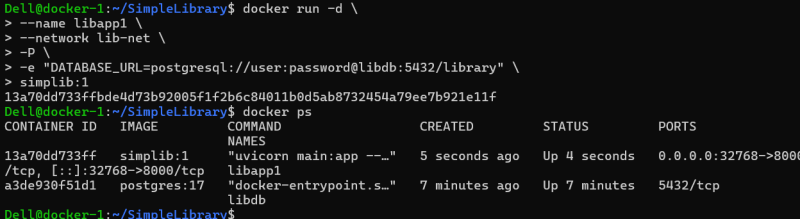
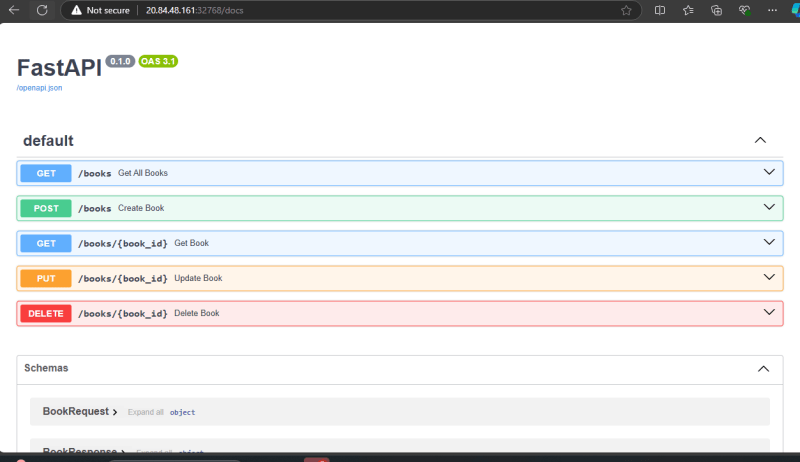
* Every Database generally has a
  + hostname (name/ip address)
  + username
  + password
  + database name (optional)
* Generally all languages use a connection string to establish connection
* When we run our application in containers in most of the cases the db details are passed as environmental variables and in somecases it will be file

### Networking

* Docker supports creating networks of different types
  + bridge network (single host)
  + host network (single host)
  + macvoverlan (single host)
  + overlay (multi host)
* Default bridge network cannot resolve by names but uses only ip, custom bridge networks can resolve with names as well as ips

### Running a simple 2 tier application

#### Python FastApi and Postgres

* [Refer Here](https://github.com/dummyrepos/SimpleLibrary) for the repo
* Database: postgres
  + Environmental variables
    - POSTGRES\_USER=user
    - POSTGRES\_PASSWORD=password
    - POSTGRES\_DB=library
* Steps:
  + Create a bridge network with a predefined range of ip address 192.168.0.0/24  
    
  + create a postgres container with name libdb with the environmental variables as shown above and attached to the network.  
    
  + Build the image if you donot have it yet from the repo
  + Now run the container in the same network as db with name libapp1, and set DATABASE\_URL environmental variable  
      
    
* Repeat the above steps by attaching volume for libdb container
* Use this following dockerfile as reference, to build the docker image of library application

# Stage 1: Build Stage

FROM python:3.11-alpine AS builder

# Set environment variables to avoid Python writing .pyc files and to buffer output

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Install necessary build dependencies

RUN apk update && apk add --no-cache gcc musl-dev libffi-dev openssl-dev postgresql-dev

# Create a directory for the application

WORKDIR /users

# Copy the requirements file to install dependencies

COPY requirements.txt .

# Install the application dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Stage 2: Runtime Stage

FROM python:3.11-alpine

# Set environment variables to avoid Python writing .pyc files and to buffer output

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

# Install runtime dependencies, including PostgreSQL client libraries

RUN apk add --no-cache libpq

# Create a directory for the application

WORKDIR /users

# Copy installed dependencies from the builder stage

COPY --from=builder /usr/local/lib/python3.11/site-packages /usr/local/lib/python3.11/site-packages

COPY --from=builder /usr/local/bin /usr/local/bin

# Copy the application source code to the working directory

COPY . .

# Ensure uvicorn is installed in case it wasn't properly copied or was missed

RUN pip install --no-cache-dir uvicorn

# Expose the port the app runs on

EXPOSE 8000

# Command to run the application

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

# Findout What a docker registry is and what is public registry and private registry

* Private registries are used to store images securely and access will be provided within your organization
* Popular private registries
  + Docker hub (Paid Plan)
  + Jfrog
  + Azure Container Registry
  + AWS Elastic Container Registry
* When we create a private registry, we create repository for each image
* Exercise:
  + Build library image usign python:3.11-alpine and push into
    - dockerhub (public)
    - acr
    - ecr