We need to run top command in mobaxterm so that it will not go in inactive state

Section 1 Docker

Section 2- Dockerize Micro Services

Section 3: Docker Compose

**Objective** : why docker is more popular containerization tool or framework

To deploy java application or any other application we need physical machine or virtual machine that can even on be a cloud . Once we have this machine we can be installing all the sw that are required for our application

Will do memory setup,networking,db creation and finally our application will be deploy on this machine.

For java this could be a war file or jar file.

The entire process of creating the machine , installing the sw and pushing the app itself is very time taken

And expensive that is where **containerization or containers** come in picture.

Containers can be very quickly started and containers can be moved **across environment and machines**

Very easily while maintaining the same level of consistency that is once we have a container all the s/w

Required for our application and the application itself is part of container we can easily move whole setup as a container in a from dev to test to stage and prod without any inconsistency

So **containerization is a process of bringing together all the components that are required for**

**Our application to work along with application itself and we launch container that can be easily moved across environment or scaled up and scaled down**

Graphical user interface

Description automatically generated with medium confidence

**What is Docker and why it is used?**

Docker is **an open source containerization platform**. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.

**Docker :---**

**is most popular containerization tool** that follows OCI standard( open container initiative standard)

Instead of deploying as war/dll/ code . Once we dockerize our project , we will be deploying a image.

Forget about war etc, will be creating a docker image and this **docker image will be use to launch our application** . Out of this docker image will create docker container and once we have those container

And these container can be deployed across environment as required.

So we have one container which will have all the component of our application or we can have multiple container each container can host one particular component of our application. Here

A container with cent os operating system which is running my application server. Web server is running

On ubuntu server , db server is running on macos and application is running in windows

We can have like this or we can put everything for our application into one single container

**A Docker Image is a template of instructions which is used to create containers**

**A docker container or any container in general is a runtime instance of an Image. So once you create an image for your application, you need to start it as a container.**

**A container runs your image and also takes care of its resource requirements like networking, storage, cpu and configurations. A container runs on top of a container run time platform like docker.**

Diagram

Description automatically generated

**Note :--**

**We can launch each of the image with separate container which can communicate easily**

Kubernetes can launch automatically other instance of same container if load increases.

Once we create a docker image we can create a docker container which will host the entire application

**Summarize :--**

Docker simplifies application deployment process by giving us ability to create a image.

This image will have all the infrastructure required for our application to work along with the application itself

Or

We can have images for each component that our application need separately.

We then use these image to launch a container using Virtualization platform.

**The container itself will have base operating system which could be linux or any other OS** on top of it

We can have all the software that is required for our application to work and finally the application itself

So docker give us ability to **package one run anywhere**. Once our application docker image is build.

We can run it on any OS that has docker sw installed.

Graphical user interface

Description automatically generated with low confidence

**PODA : package once deploy anywhere**

**Docker Architecture and workflows**

**Objective :**

Imp component of docker architecture

**First component : Registry**

This is where all the docker images are stored. It is a central repository

**Second Component : Docker host**

This is the machine where we install docker engine or docker virtualization software

**Third Component : Docker client**

It will give ability to run command against the docker engine

**Example :**

When we run **docker pull <image name >** docker host will take this command and pull the image from central registry or repository . It will pull the image on docker host machine and it will store them locally

And then we can execute **docker run <image name>** command which will launch container which will have sw

Up and running

We can create our own image also using docker build <image name > and we can push the image into central repository and other developer start using our image.

Typically docker client and docker host running on same machine . They could run on different machine as well

Diagram

Description automatically generated

**Docker in action**

**Objective :--** will install docker in linux box (aws) on ec2 machine

**Step 1 -Launch a new instance in ec2**

Graphical user interface, text, application, chat or text message

Description automatically generated

**Step -2 To install docker in ec2 machine**

Text

Description automatically generated

**Step 3- Docker version**

Text

Description automatically generated

Will get all features except some security features and support from the docker

**Step 4-** We install docker successfully but we havent started docker engine or the server which is the key

Text

Description automatically generated

**Step 5 :- that will start docker daemon for us** :--- service docker start

A black background with white text

Description automatically generated with low confidence

Now if we type d**ocker info** it will give complete info about entire docker server

Text

Description automatically generated

**Note :**

By default **docker swarm** is inactive.

It allows to orchestrate the container

**Step 6 : docker images :** this will list out all the docker images that linux box have

Text

Description automatically generated

**Step 7** :-Hello-world is image name which is there in docker-hub let run it

Docker typing in linux box is nothing but **docker client**

For the very first time it will pull images from **docker hub just like maven build tool**

Text

Description automatically generated

Text

Description automatically generated

They will have unique image id that will assigned by docker engine. It is a 12 character id

Text

Description automatically generated

Linux

Yum install docker

Docker --version

Docker info

Service docker start ------------------ it will start docker container

Docker images

Docker run -it ubuntu bash

Docker pull mysql:5.7

**Here after : version is there in last command**

**Docker run**

**Objective :-**

To learn more about docker run and also will launch nginx container and will access that container from the browser running on our local machine

Docker run --help

Graphical user interface, text

Description automatically generated

**By default container port will not expose to outside world**

**Container port are not accessible directly**

**Lets quickly use these option and launch a container**

**Docker run -I -d -t -p 80:80 nginx**

**Note :-** Nginx is running on port 80 and we are exposing outside world

**-I** interactive mode

-d detached

-t want a terminal

-p publish a container port to the host

**we can exposed port to outside by using -p**

**80: (lhs)** should be exposed **outside on port 80 and :80 (rhs)** should be exposed on **port 80 in container**

**The container port 80 can be accessed on port 80 of host machine.**

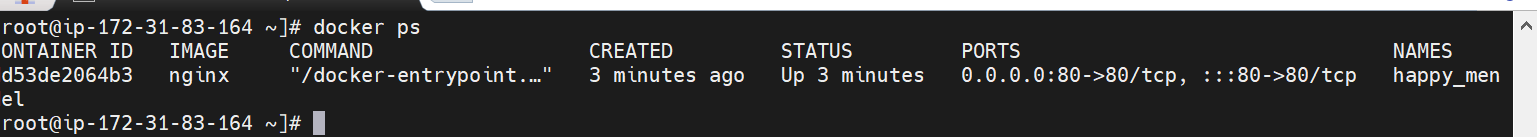
**In this case the host machine is aws linux box**

Text

Description automatically generated

**Docker ps :-** will tell us process status of container

Nginx container is up and running



Graphical user interface, application, Word

Description automatically generated

Since everything er are don’t on ec2 machine

Graphical user interface, text

Description automatically generated

When we hit this url it will map to nginx port which running in container of e2 machine

So if our application is containerized we can take the application put it on any box and then it can access from anywhere that is the beauty of docker.

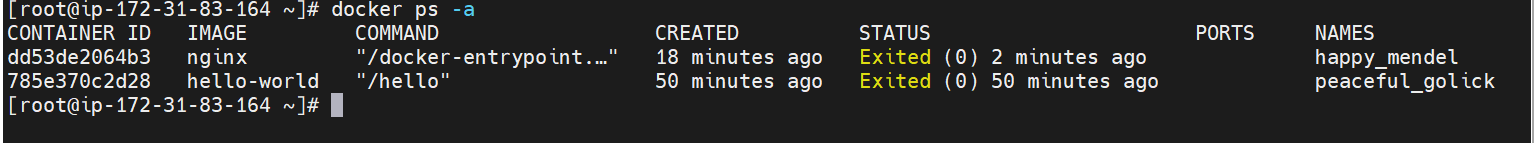
Stop a container :- docker stop container id

A screenshot of a computer

Description automatically generated with medium confidence

Docker ps :- will tell us process status of container and it will show those container which are up and running

Docker ps -a :-- will list out all the container



What is docker-compose vs Kubernetes?

Kubernetes and Docker Compose are both container orchestration frameworks. **Kubernetes runs containers over a number of computers, virtual or real.** **Docker Compose runs containers on a single host machine**

**Docker Clean up**

**Delete container and images**

We have 2 container and 2 images so let clean up

Text

Description automatically generated

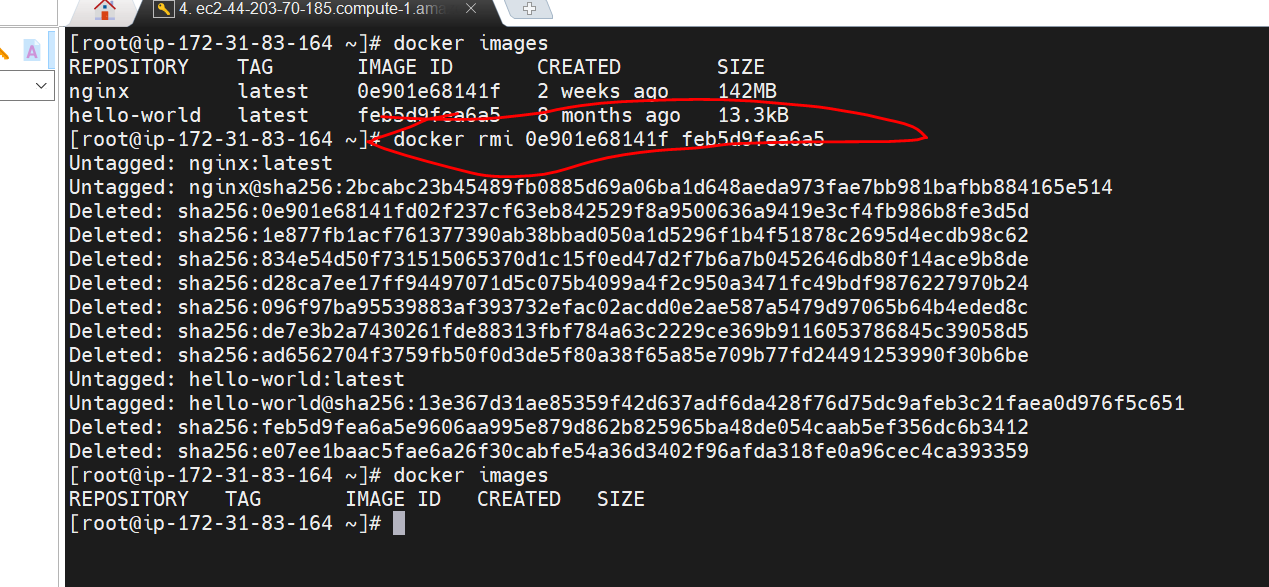
Note : if the container is up and running we need to stop a container before it deleted

A screenshot of a computer

Description automatically generated with medium confidence

**Remove container** docker rm container id

**Remove images** : docker rmi image id



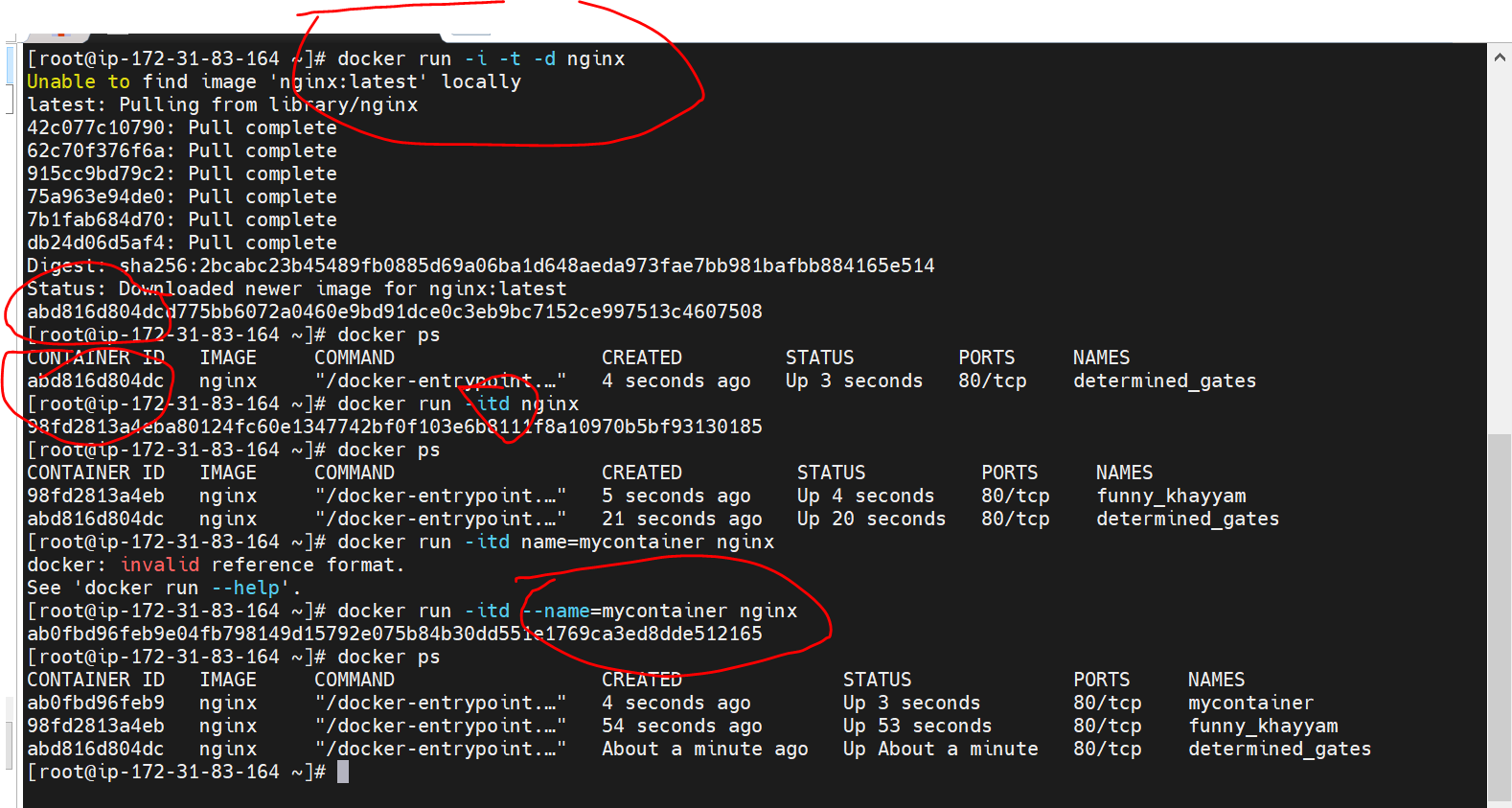
**Get Comfortable with docker run**

**Note**

we can assign container name when we run them

We can use -itd instead of -I -t -d

Next time it will not pull image since it is there in local but it will launch new container



**To stop container :** we can either use container name or container id

Graphical user interface

Description automatically generated with medium confidence

Temporarily pause a container :

docker pause container name

Docker unpause container name

**Docker Commit**

**Objective :** docker commit and how to use it

**Docker commit** save the current state of container into an image ----------------imp

Chart

Description automatically generated with low confidence

Suppose we pull ubuntu image and run it in container where we installed apache2 now we can create our own image

Which will be having ubuntu and apache2 using docker commit and we can share this image to other developer

This is not recommended way but we can do this in fast way so other developer can test it

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated with low confidence

Docker commit container id of ubuntu mywebserver

Means whatever current state of container ir we are committing in mywebserver which will act as image and new image created

Chart

Description automatically generated with low confidence

**Docker run -itd --name myweb mywebserver** :----------running the images into container

**Docker exec -it myweb bash** :----------------- entering into the container

Note : in this image we already have demo folder and apache2 installed

**Docker layers and Overlay storage**

Text

Description automatically generated

**NOTE :**

**ADD and CMD** are dockerfile command

Graphical user interface, text

Description automatically generated

**Where the missing image is stored in machine**

**Overlays :-- is storage driver of docker**

**Run docker info** command and go in this path in ec2 machine

Text

Description automatically generated

Text

Description automatically generated

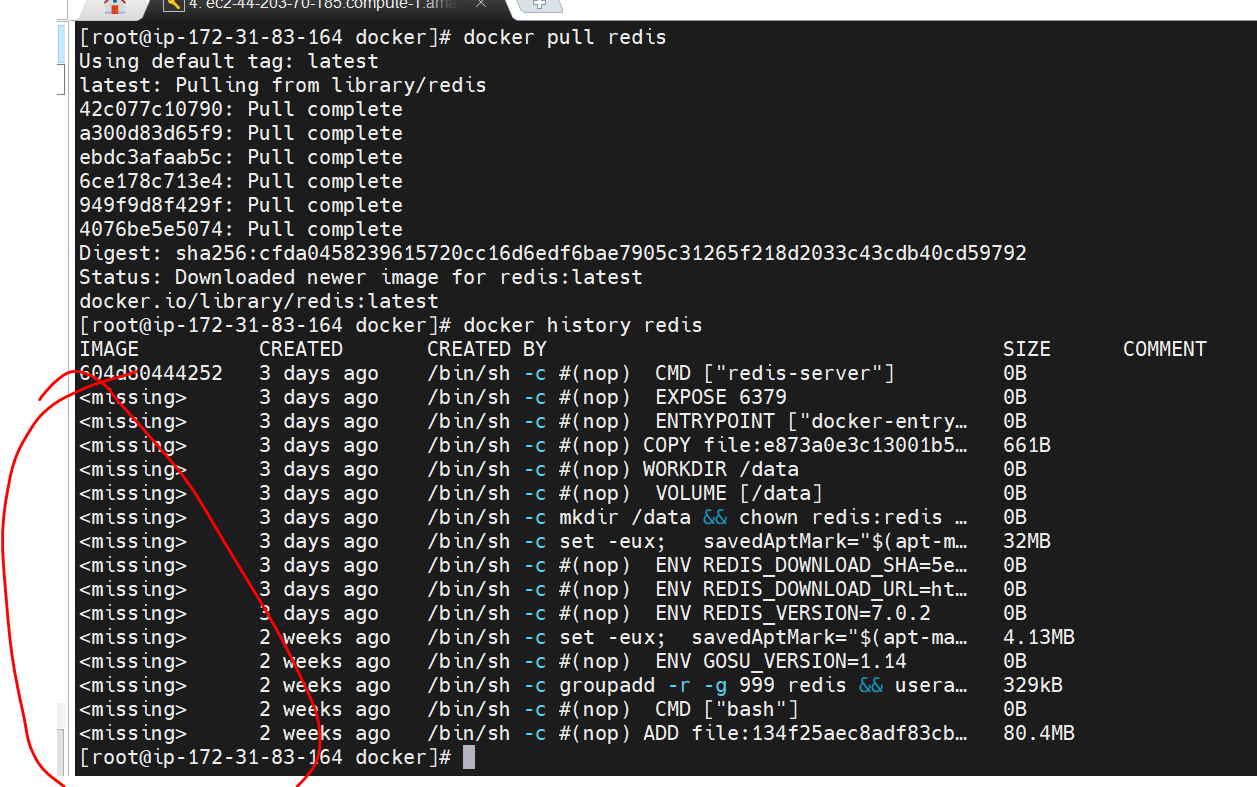
Overlays2 :--- responsible for storing the images, maintaining the images

Chart

Description automatically generated

All the layers which downloaded when we pull ubuntu . These all are read only. We will not touch them.

**Pull redis**



**And we understand this better when we will create our own docker images.**

**Note :**

Images creates using all those layers

Docker uses union file system where it will integrates various layer together to come up with a image

Or

All these layers are read only layer when we launch a container we can write only within a container and the whole thing will become a

Image of its own as we have seen using docker commit but these layers are read only.

Overlay is storage driver that is responsible for bringing the layers together and creating the final image that we want and the location

/var/lib/docker in this location everything related to docker is stored.

Note :

In repositories.json all the image info will be there

A screenshot of a computer

Description automatically generated

Note : it will store the redis container id in this path

Chart, text

Description automatically generated

**Launch Mysql container**

**Objective:**

How to launch mysql container go into that container and even execute commands in the process

Will learn how to pass environment variable while launching the container

Service docker start

Automatically mysql will read the env variable and set it up

Once mysql up and and running it will create database for us

It will pull image first then it will launch container in detached mode and it will set pw 1234

A screenshot of a computer

Description automatically generated

Go into the container

Text

Description automatically generated

Here we execute all sql statement

Text

Description automatically generated

**Volumes and Bind Mounts**

When we launch a container store some data in container **writeable layer.**

**Our application needs this data.**

**When we Stop the container will still have data** but if we delete the container when we relaunch another container from

Same image that data will no longer be there that is where concept of bind mount and volumes come in

**That will help us to persist the data** on host machine that is running the docker for these containers

So instead of storing the data directly into container we will use bind mount or volume and once the container is deleted

That data is still on host machine

When we launch another container it will reuse data by using same bind mount or volume on host machine

Chart, diagram

Description automatically generated

**Bind mount**

Will use any folder on the host machine to communicate the data or to store the data

Chart

Description automatically generated

So we can randomly pick a folder on machine then we can mount it onto the container and then we can start exchanging data

Between host machine and the container using this directory

**We put any content into directory from the container u can see on the host , put any file on the host we can see inside the container**

**Volume :-**

Is not just a folder on machine . We create volumes using the docker commands and docker will maintain all those volumes

Inside a folder called /var/lib/docker/volumes

Chart, waterfall chart

Description automatically generated

**Key diff.**

Bind mount can be any folder on machine and they will not managed by docker so this can be access by any other process as

Well but these volumes are docker object and we can easily mapped these volumes to any docker container but they are maintained

By docker

**Volumes are recommendable approach for persist data using docker.**

**Use Volumes**

**Objective :**

will create docker volume , will mount volume on a container that will launch and also we will exchange file b/w

Docker host and container itself

**docker volume ls :** this will list all the volumes this docker is managing

This is local default volume that gets created when we install docker

Text

Description automatically generated

**Create volume**

Docker volume create myvol

Text

Description automatically generated

To mount this on any container let launch a container

Source =myvol-----which volume we want to mount

target= /tmp once we inside container /tmp is map to myvol

-v ----use for both volume and bind mount

Text

Description automatically generated

Before go into container volumes creates locally in this path



As of now folder is empty because we haven't created anything inside container

**Lets go into the container and we are inside container**

A picture containing text

Description automatically generated

It will create a file under /tmp

Text

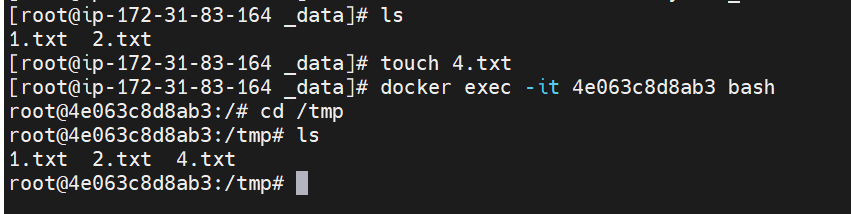
Description automatically generated

Text

Description automatically generated

So this volumes was mounted to the /tmp folder inside the container wtever will add inside the container will show in /tmp

folder. Its vice versa

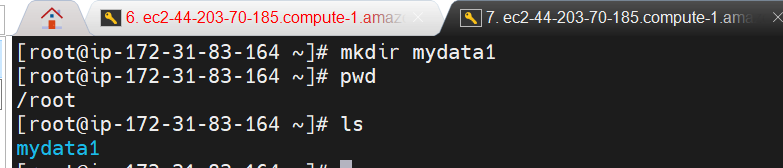


Note :--- **we can share volume across the container**

**Use Bind Mounts**

**Objective :** Bind Mount in action

Let create a local folder



**Lets launch a container and go inside container**

Here we are mapping a random directory /tmp once container is up , we are not mapping any volumes

**Note:** docker don’t manage bind mount

Text

Description automatically generated

**Recommended way :**  to go with docker volumes

**Docker networking in action**

**Objective :** docker networks

Docker networking is **primarily used to establish communication between Docker containers and the outside world via the host machine where the Docker daemon is running**.

**Docker network ls** :-- this will list of all network by default

Text

Description automatically generated

To know the details of particular network we will use

**Docker network inspect network name**

Text

Description automatically generated

*We are going to create a network of our own , similar to this we can create any ip or subnet network.*

**Create our own network**

**Docker network create demonw --subnet=172.19.0.0/16**

We have now new network created i.e network id

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

**Now we can use this network and assign to a container when we launch it**

So lets launch a new container

**--net :** we need to use to assign a network

I want this container to be a part of demonw which we just created also assign a ip address . We can also give host name.

Simply give a host name for our new docker container that we are launching

**docker run --name webserver3 --net demonw --ip 172.19.0.2 -h web.saurabh.com -p 82:80 -it ubuntu /bin/bash**

**Hold ctrl pq :---** come out of container

**Note :-**

**Once we come out of container we can reattach container**

Docker attach webserver3

Text

Description automatically generated

If we do not provide a hostname then automatically docker will provide a unique name

**We make a container a part of our network that we have created**

**Change a container network**

We learn how to assign n/w to container when we launch it

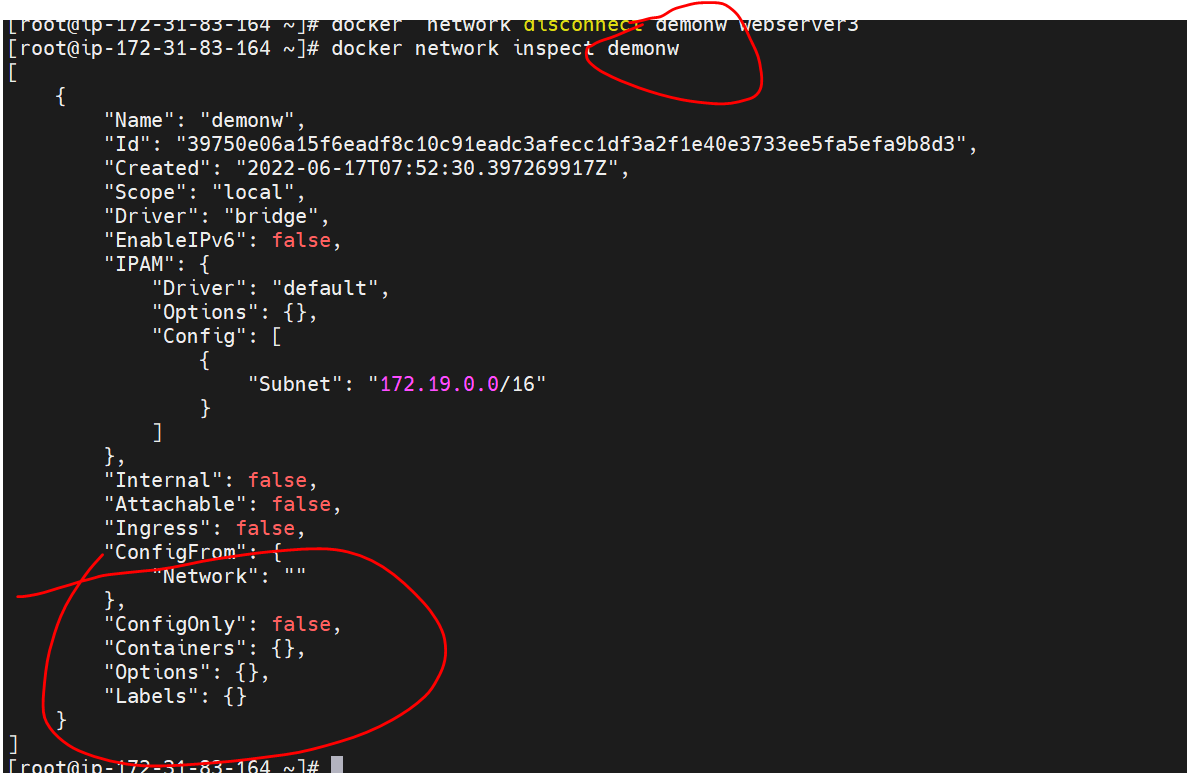
**Objective** Will learn how to disconnect a containers from n/w and attach to other n/w.

**Disconnect container from network**

**Docker network disconnect demonw webserver3 ---**Docker network disconnect network name container name

That will remove this container from network

Now this network don’t assign any container



We can also do **docker inspect webserver3**

**Now we can see that with container no ip is attached**

Text

Description automatically generated

No ip address because it is out of the network but we can see that network is still there

Text, chat or text message

Description automatically generated

**To connect this container with new network or same network we can use**

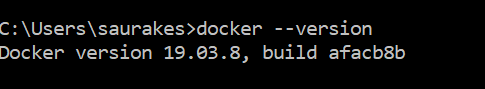
Connecting with bridge n/w which came by default

**Docker network connect bridge webserver3**

This time we see a ip address

Text

Description automatically generated



**Dockerfile introduction**

So far we pull existing images from docker hub and we use it to run docker container

**Objective :-** how to create docker image of own and use then to run container

**Dockerfile** :- a text file will have all the command that are required to assemble a image

Execute build command which will give dockerfile to docker engine and it will read a instruction and create a image

**Docker cli ----------build --------------------dockerfile------------------docker engine---------------image**

Once we have image we can run any number of container from that images

Diagram

Description automatically generated

**What goes in dockerfile**

List of commands provided by docker such as **FROM,COPY,ENV,RUN,CMD,EXPOSE** and more

Very first line in docker file is **FROM command**

**Note**

We always create our images based on existing images

We can us**e copy command** to copy our application files to particular folder on images

Env : we can set like java home ,maven home etc

Run : when image is build run command will become part of itsel

**CMD** will be executed after image is build

That is difference between RUN and CMD :

Run will execute during the image is creating while CMD will execute after image creation

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. CMD is the command the container executes by default when you launch the built image.

And **if you want to run a container with the condition that a particular command is always executed, use ENTRYPOINT**

**Expose :** will expose a particular port on docker image

A screenshot of a computer

Description automatically generated with medium confidence

***First Dockerfile***

**Objective** : will create a dockerfile and build a image of own

**Requirement :--- see from bottom top top**

80 : and expose it on port 80 so that world outside can access webserver runinng on docker container once it build and run

Index.html :----add a file index.html

Httpd :--- on top of that want to install httpd webserver

Centos :----use centos as base operating system

From centos (this is base image from which our own image will be build)

RUN yum install -y httpd (run a command so that we can install a webserver)

ADD index.html ( add index.html to the /var/www/html of the container)

CMD apachectl -D FOREGROUND ( we are starting web container now)

Expose 80 ( we are telling docker engine to expose 80 since this is a web server)

MAINTAINER Saurabh ( telling who is maintaining this dockerfile)

ENV myenv myval ( also we can pass environment value---- key ,value pair)

Text

Description automatically generated

Let build a images before we do that we need to create index.html

Text

Description automatically generated

-t for tag the image

Docker build -t my-webserver . (here we need to give path of dockerfile our dockerfile is in current directory so giving .)

**Docker build will automatically search file DOCKERFILE**

Text

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

**Working**

Text

Description automatically generated

**Note**

It removing the imtermediate container also. Every step ends in a layer and finally we end up with final image

**Beauty of docker layer**

**Objective :-** will learn few more thing about docker image and image creation process

Let see history of docker images : docker history image id

This is final image and all are temporary layers that are build for us

Graphical user interface, text

Description automatically generated

Add one more env variable in dockerfile and this time it is taking from cache

Text

Description automatically generated

**Step 1-Before we push image to central hub we need to tag image**

Docker tag current image name docker hub user id /name of the image

Docker tag my-webserver saurkesa/my-webserver

**Step -2** before we push image we need to do docker login

Docker login

**Step 3:**- once login is succeded we can push image

Docker push saurkesa/my-webserver

A picture containing chart

Description automatically generated

Graphical user interface, application

Description automatically generated

**Docker RUN vs CMD vs ENTRYPOINT**

Some Docker instructions look similar and cause confusion among developers who just started using Docker or do it irregularly. In this post I will explain the difference between CMD, RUN, and ENTRYPOINT on examples.

**In a nutshell**

* + RUN executes command(s) in a new layer and creates a new image. E.g., it is often used for installing software packages.
  + CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.
  + ENTRYPOINT configures a container that will run as an executable.

Section 2- Dockerize Micro Services

Install docker in visual studio : help us to docker compose file, yaml file very easy as well as we can create kubernetes file

**Launch my sql container**

**Objective**

Containerize our m/s

Diagram

Description automatically generated

**Dockerize m/s application**

**Setup the mysql container:**

docker run -d -p 6666:3306 --name=docker-mysql --

env="MYSQL\_ROOT\_PASSWORD=test1234" --env="MYSQL\_DATABASE=mydb" mysql

docker exec -it docker-mysql bash

# mysql -uroot -p

test1234

mysql> show databases;

mysql> show tables;

Another Terminal:

docker exec -i docker-mysql mysql -uroot -ptest1234 mydb <tables.sql

**Launch the Application Containers:**

docker build -f Dockerfile -t coupon\_app .

docker run -t --name=coupon-app --link docker-mysql:mysql -p 10555:9091

coupon\_app

docker build -f Dockerfile -t product\_app .

docker run -t --link docker-mysql:mysql -p 10666:9090 product\_app

docker run -t --link docker-mysql:mysql --link coupon-app:coupon\_app -p

10666:9090 product\_app

**Testing:**

<http://localhost:10555/couponapiapi>

<http://localhost:10666/productapi>

The --link command will allow the Container

**Step 1:**-

first container we are going to launch is my sql server and it will use by both other m/s which will get their own docker container

**Launch d/b container**

**-d :- detached**

**-p :-port exposed default is -p 6666:3306 (c**an access docker container on port 6666)

We can pass env variable to a container

**Setup the mysql container:**

docker run -d -p 6666:3306 --name=docker-mysql --

env="MYSQL\_ROOT\_PASSWORD=test" --env="MYSQL\_DATABASE=mydbnewtest" mysql

**To go inside container**

docker exec -it docker-mysql bash

**To go in sql**

# mysql -uroot -p

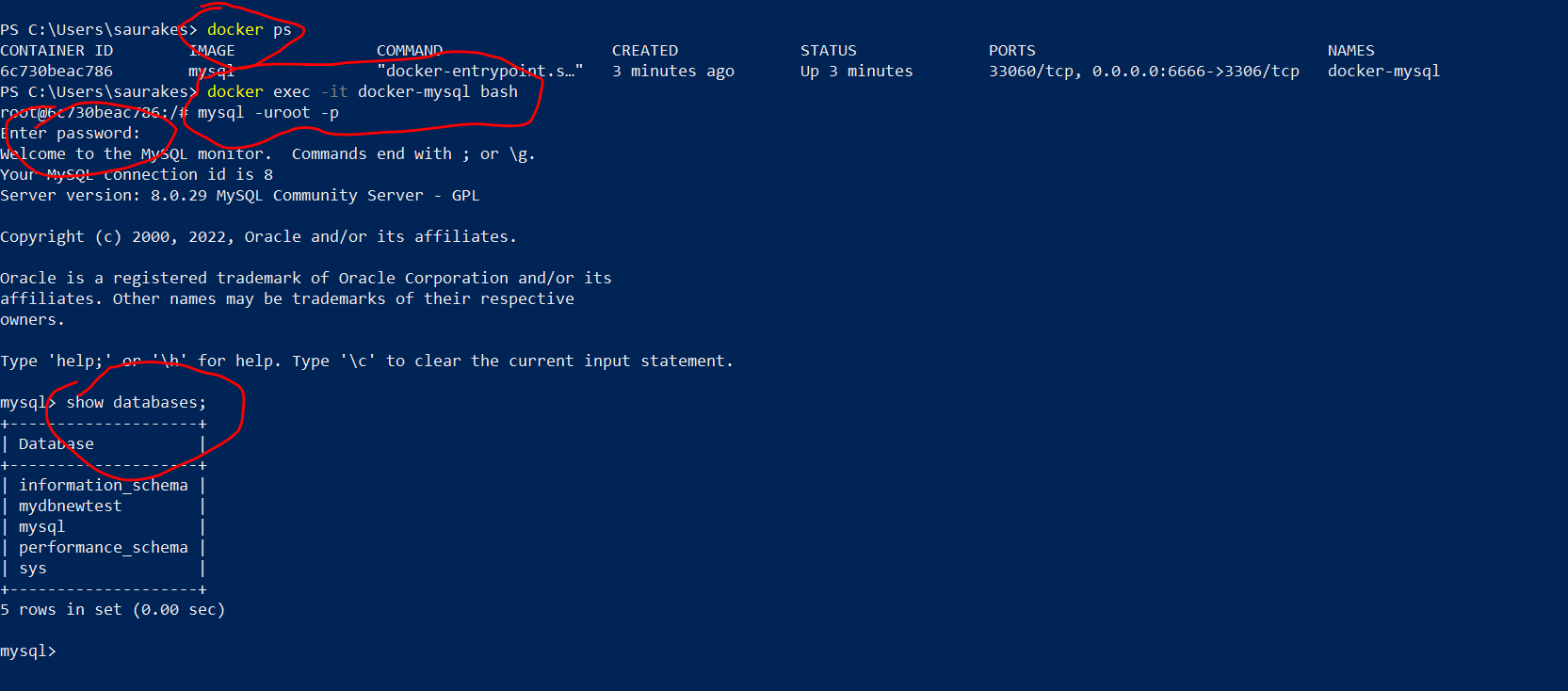
test1234

mysql> show databases;

mysql> show tables;

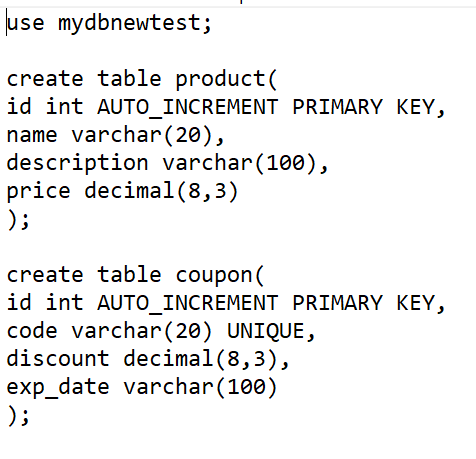
Text

Description automatically generated



**Create Tables**

Come out docker container completely will execute query in another way



-I --------

Want to execute command on docker container, which docker container want to execute command --------docker-mysql,

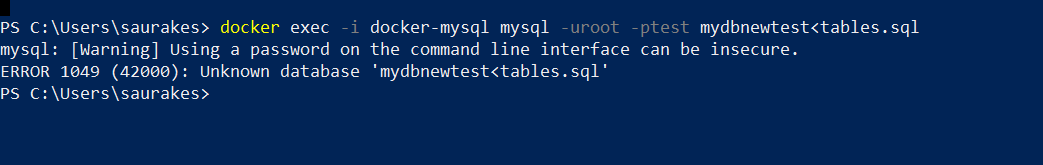
Mysql point to client running on the machine

run tables.sql on this database mydbnewtest

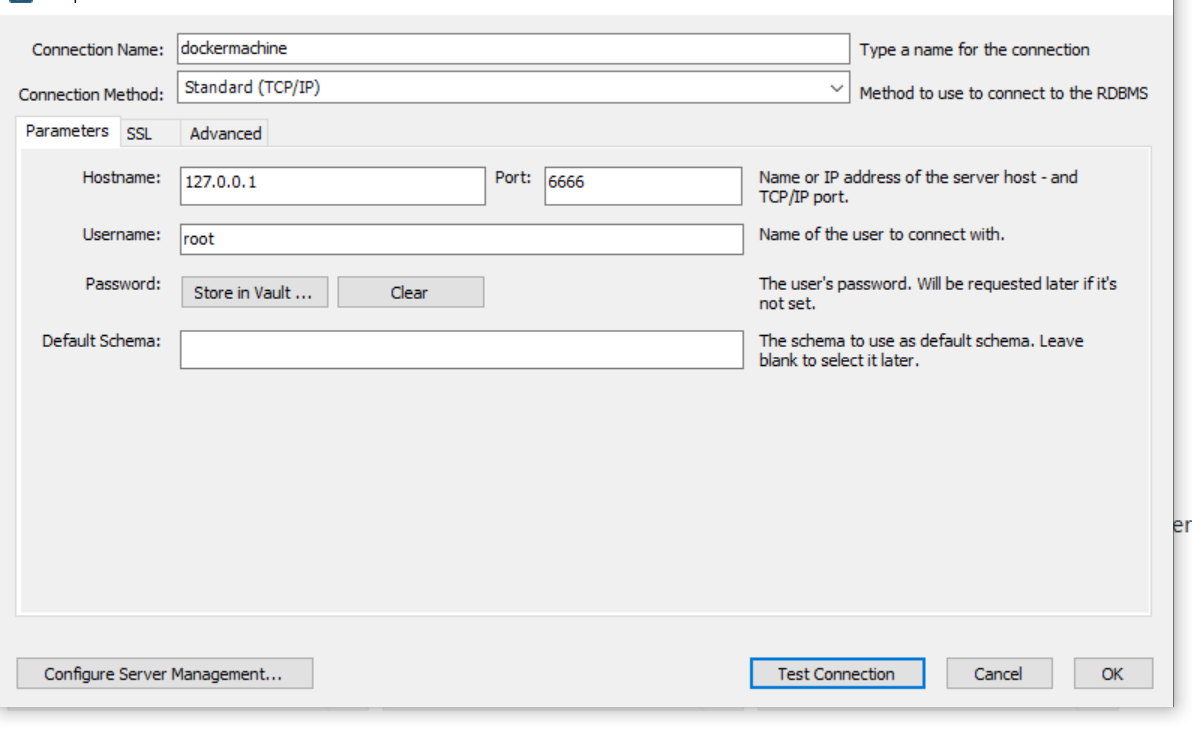
**Note**

This is one of the way of creating table or we can directly create table in mysql

**docker exec -i docker-mysql mysql -uroot -ptest mydbnewtest<tables.sql**



We can connect from my workbench running on local and we can test **whether table is created**



Now this will execute that sql is set up on docker container

Graphical user interface, text, application

Description automatically generated

**Dockerize the apps**

**Objective :-** Containerize coupon and product m/s

Note :

Visual studio automatically associate this docker file with docker

Graphical user interface, text

Description automatically generated

ADD source dest

We are copying jar file which is under target directory and destination just jar file name so it will directly copied into

Container home page

From target add jar into container home page with following name

Entrypoint(executable) is always first thing that will be execute when container is up

So that our springboot application start in embedded tomcat

A screenshot of a computer

Description automatically generated

Text

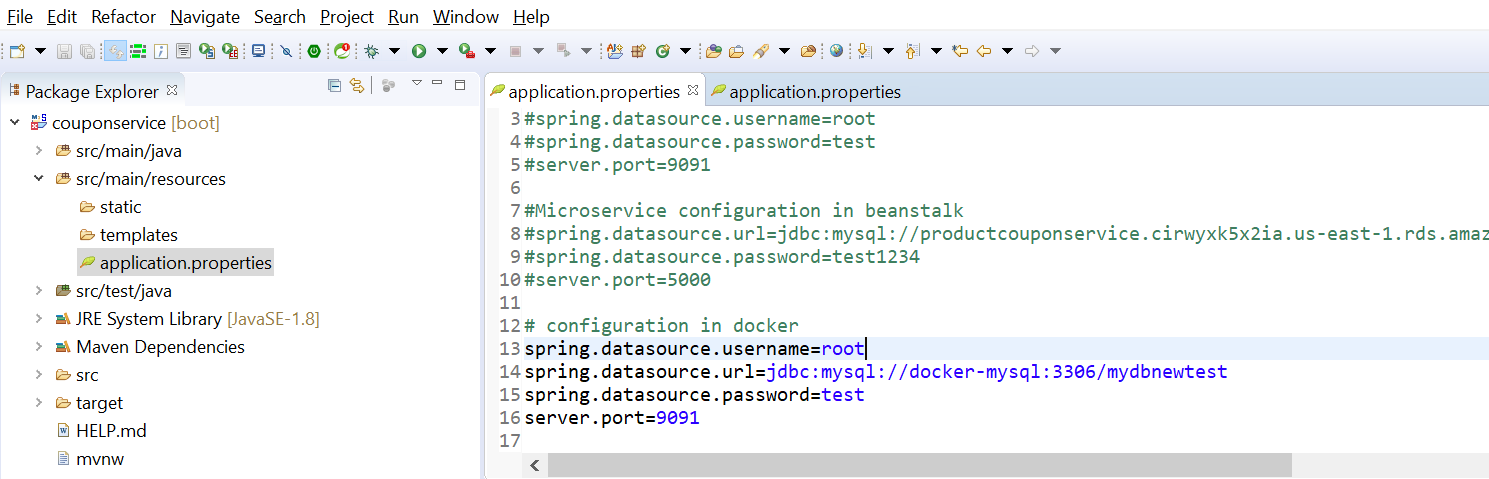
Description automatically generated

Every Object will have a dockerfile

**Build Images**

We have to make a couple of changes in properties file of coupon and product m/s

couponservice



Productservice

Text

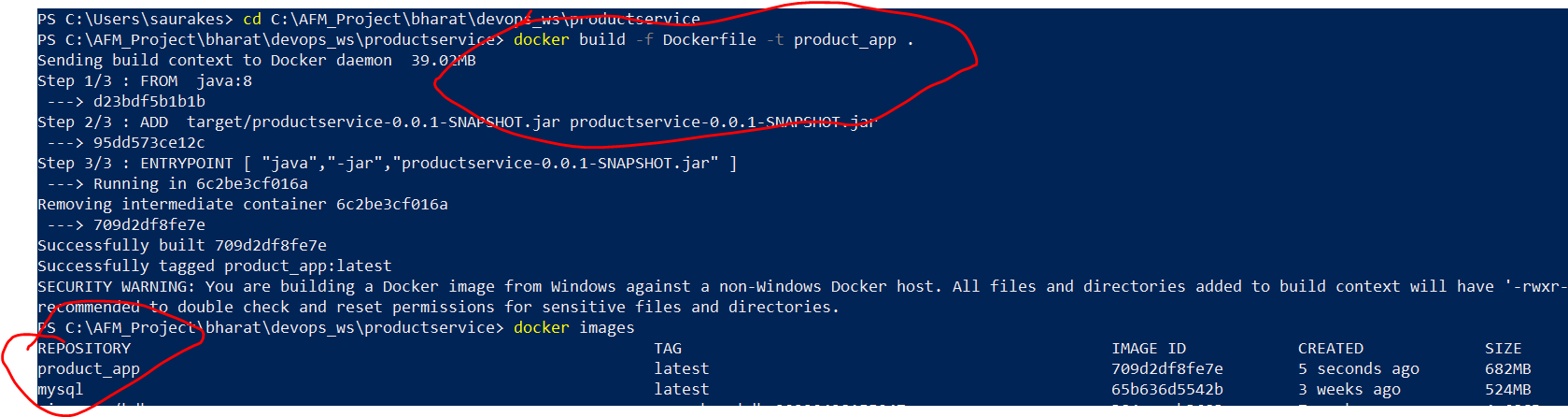
Description automatically generated

Coupon-app is the container name where coupon service is running

**Build both project and skip test**

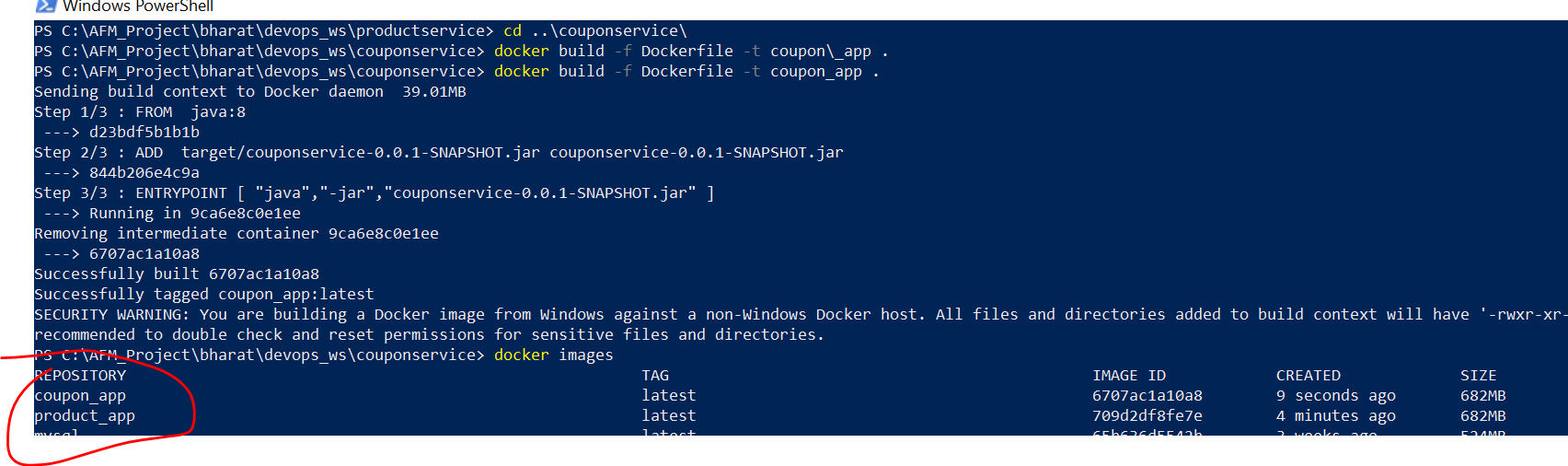
**Create image of productservices**

**-f :--** which docker file to use



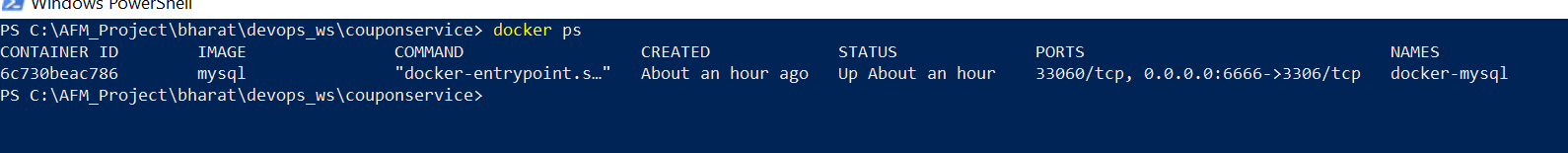
**Create image of coupon m/s**

docker build -f Dockerfile -t coupon\_app . ---------------coupon\_app is the name of image



**Launch the m/s container**

The only container is up and running is mysql



Now we need to launch coupon container and product container

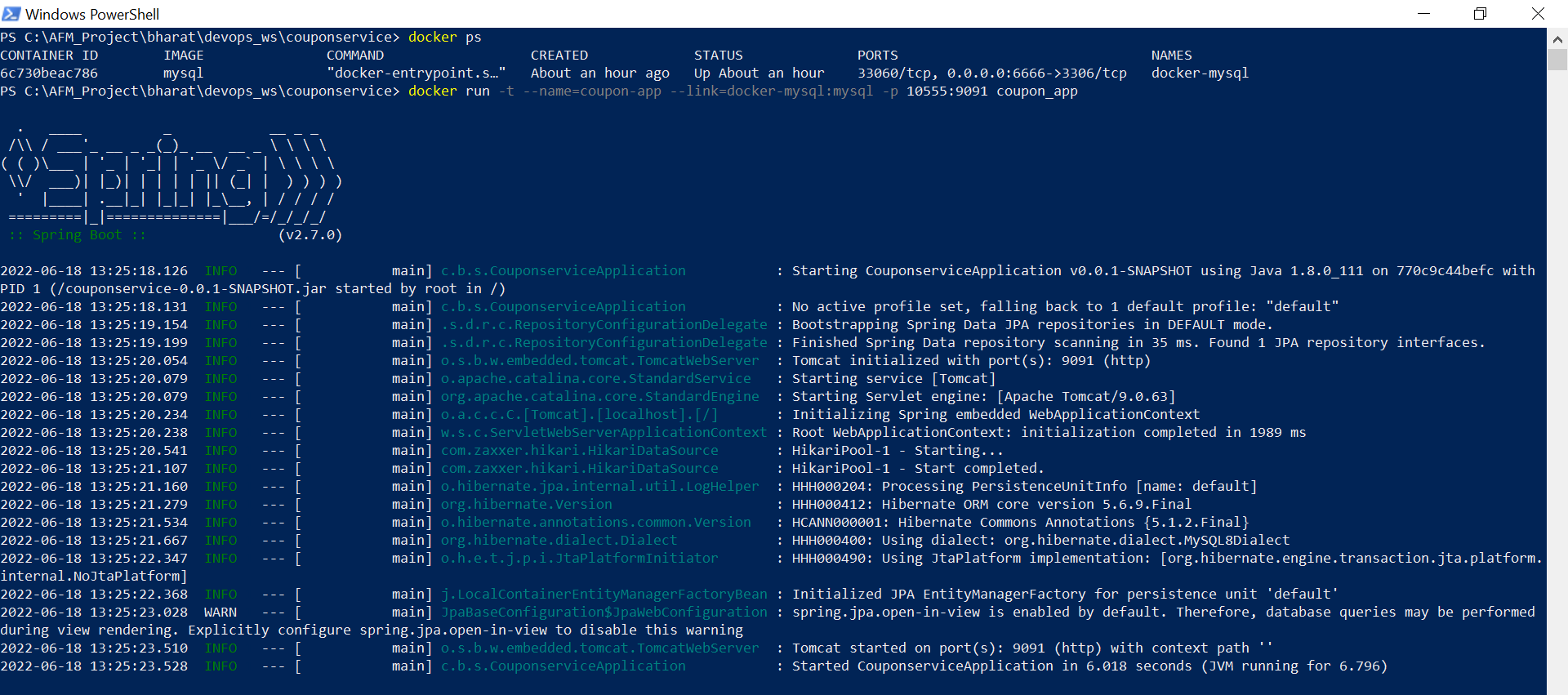
**First launch coupon container**

**-- link :----**we need to link with my sql container then only they are able to communicate (name and image)

Coupon\_app -----image name

**docker run -t --name=coupon-app --link=docker-mysql:mysql -p 10555:9091 coupon\_app**

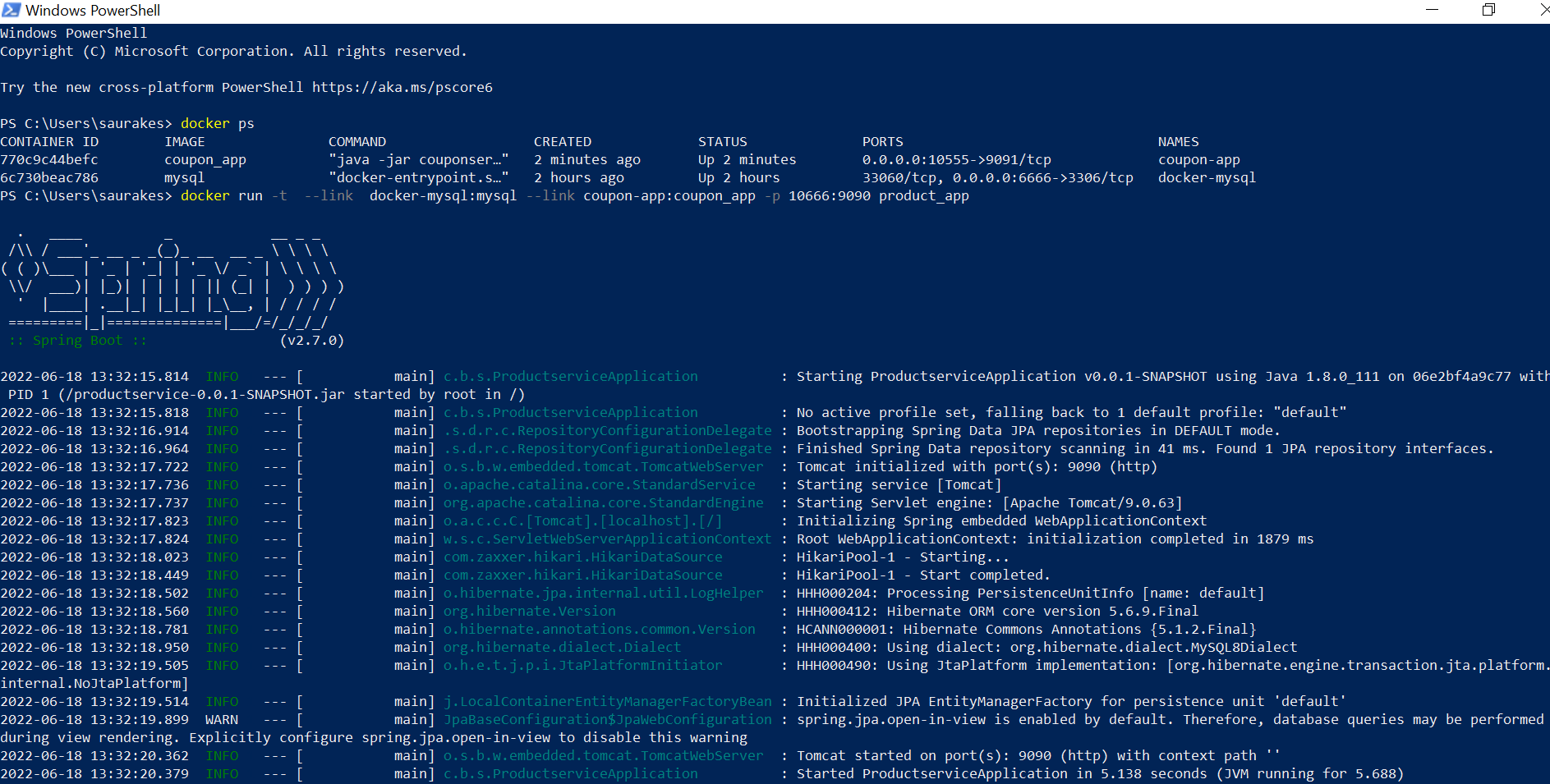
It will launch application and it will connect to mysql container running in docker container



**Let launch another container productservice**

We are running the container and linking it with two other container so that they can communicate **( couponservice and mysql)**

**docker run -t --link docker-mysql:mysql --link coupon-app:coupon\_app -p 10666:9090 product\_app**



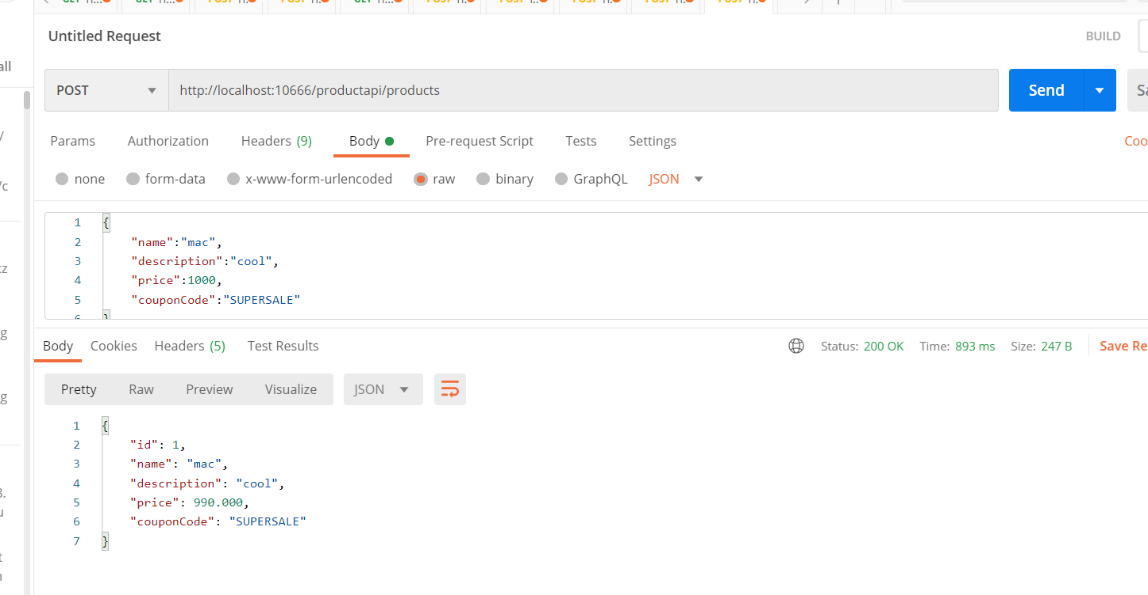
**Test**

Both container are up and running

From the local machine we are hitting coupon service running on docker container

Graphical user interface, text, application, email

Description automatically generated



Graphical user interface, text, application

Description automatically generated

Dockerhub

User id- saurkesa

Mail id : skesarwa@rediffmail.com

**Push to Docker hub**

**Note :**

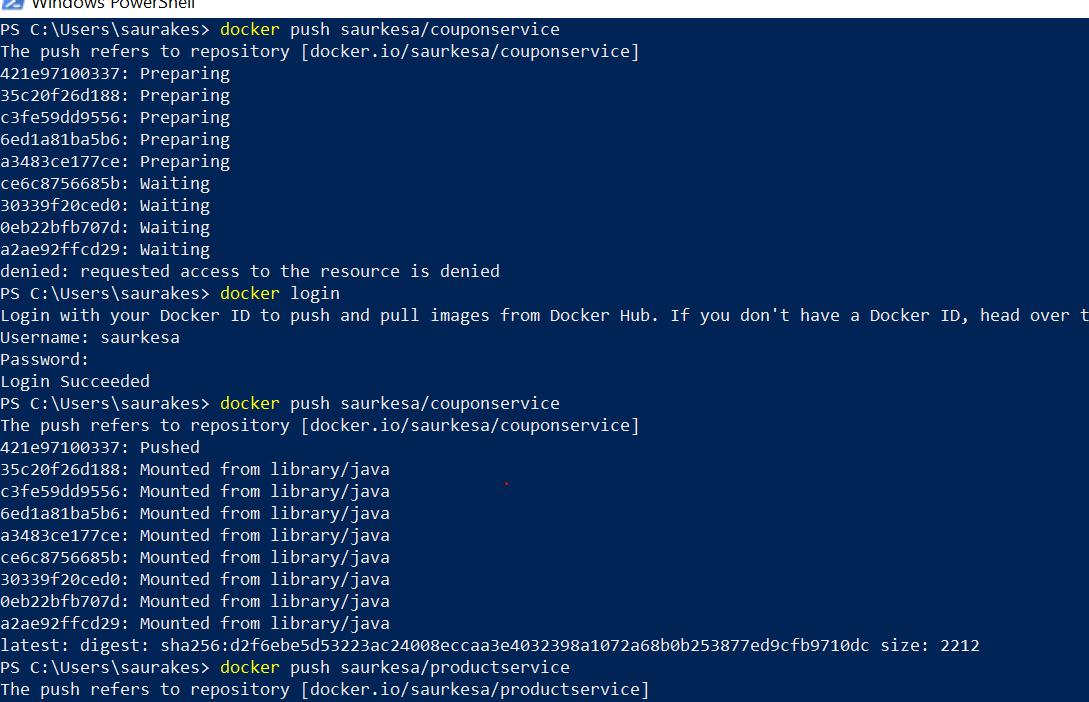
We will push docker images to docker hub so that we can pull the images when we will work with docker hub

And later on with kubernetes.

When we work with docker compose everything (m/s) is a service



Docker tag image name docker hub user name/ any name



Graphical user interface, text, application

Description automatically generated

**Docker Prone Command**

Dangling Docker Images. **Dangling images are created when we overwrite them with a new image of the same name and tag**

**What is docker prune?**

The docker image prune command **allows you to clean up unused images**. By default, docker image prune only cleans up dangling images. A dangling image is one that is not tagged and is not referenced by any container. To remove dangling images: $ docker image prune WARNING!

**Docker container prune** :-- it will remove all image that are not being used

**Docker container prune -a** :-- it will removed all container

Section 3: -Docker Compose

**Objective**

What docker compose is and why we need to use it

Docker compose is a tool using which we can run one or more container as required for our m/s application.

With docker compose no need to write script or any other tool to start container

Using docker compose will create one single configuration file , within the configuration file each of these container will be

Defined as services and within a single command called **docker compose up** it will launch all these container for us

Diagram

Description automatically generated

Previous use case

Dockerize coupon service and product service and start in separate container and also we start mysql in separate container.

Instead of all these we will use single docker compose yaml file.

Within the yaml file we will define multiple services (m/s).

So each service can depends on another service.

So mysql will launch first then other service will launch

Docker-compose up :---- it will automatically search **docker-compose.yaml** (default file name) and it will use that file to launch the container.

There are several other option that docker-compose command give us

Docker-compose up :----it will create and start the container

Docker-compose restart :--- will restart all the services or container

Docker-compose build :- will rebuild all the services again

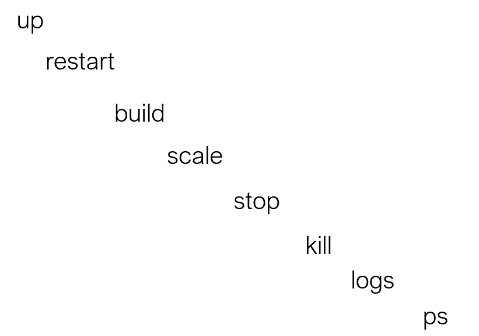
Docker-compose scale :-- will allow us to scale how many number of container we want for each service

Docker-compose stop :--- will stop the service

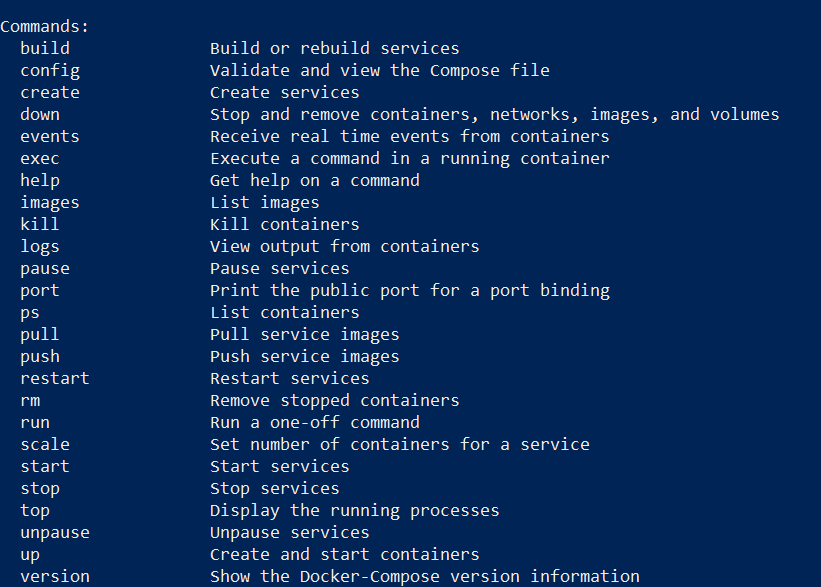
Docker-compose kill :--- will kill the service abruptly

Docker-compose logs :- want to check the output of each container we can issue the logs command and it will show the logs

Docker-compose logs :--ps will list out all container up and running



**Docker-compose --help**



**Note**

Docker compose it already come with docker desktop , we don’t need to install again

**Build: ./** -- search for docker file in the current directory. It will build image from dockefile and launc on container.

Whatever argument we are passing from command line can be specified in this file

**Yaml Syntax**

**Objective :** to learn yaml file

Yaml is required for **docker-compose, all the kubernetes** file and will see it in ansible section.

Yaml has become a standard in dev-opps world for configuration

Demo.yaml or demo.yml both are fine

**Note**

Visual studio has support for yaml file

YAML stands for **yet another markup language** or **YAML ain't markup language** (a recursive acronym), which emphasizes that YAML is for data, not documents. YAML is a popular programming language because it is **human-readable** and easy to understand.

Yaml is superset of json

**Yaml uses**

Simple types

Lists

Objects

Dictionary or map

We can validate using a validator

**Examples : lets say we are processing order of amazon**

**Note :**

**The value we pass from yaml file will give to docker container or a tool like docker**

* + **simple data**

There will be space after : only then yaml processor knows that this is key and this is value

customerName: saurabh

orderId: 123

date: 12/12/2022

shipped: false

* + **List**

What products does this order contains

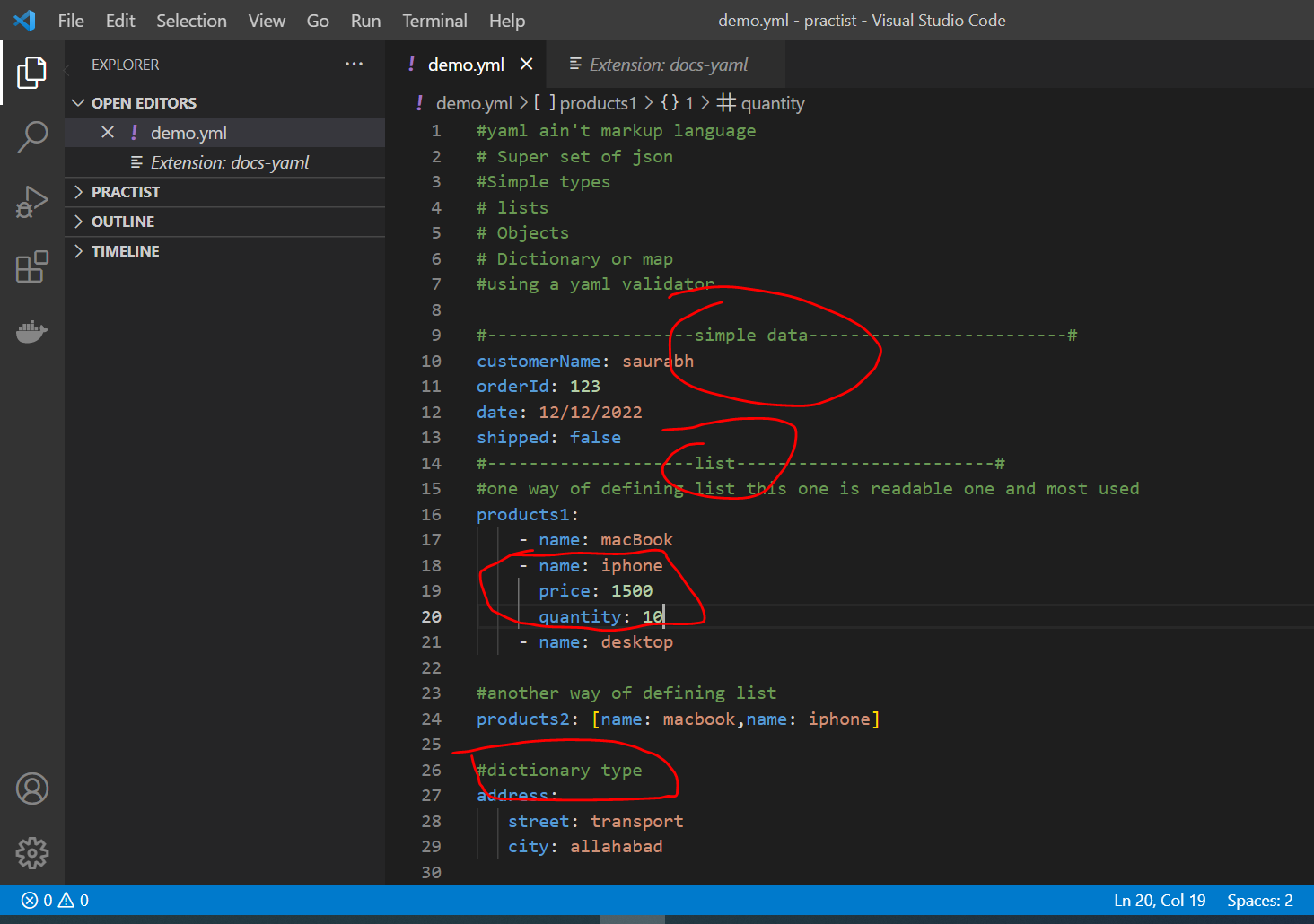
To create a list first thing we need to do - (hyphen and space) and element in the list

Within the list we can give name value pair as well

* 1. **Object: entire thing is object**

Graphical user interface, application

Description automatically generated



**Important yaml element in docker compose**

**Objective :**

Imp element of docker compose file

Version : '3'

With every version some element will go away and some will add.

We represent each m/s or container we want to launch as service.

Root element in every docker compose file is services and inside the services we define all the docker container.

Each container will have a unique name and then image from which this container should launch

Restart policy :--- always launch the sql

**Compose in action**

**Objective:**

Will learn syntax of docker compose and docker command by working on them

In each version of docker compose, docker will introduce several new elements

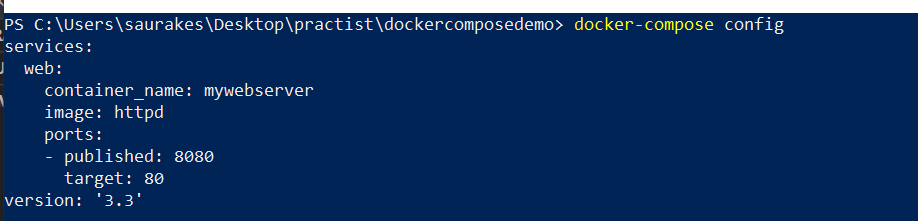
A docker compose file is group of services. Here we are defining one services

Both are equivalent



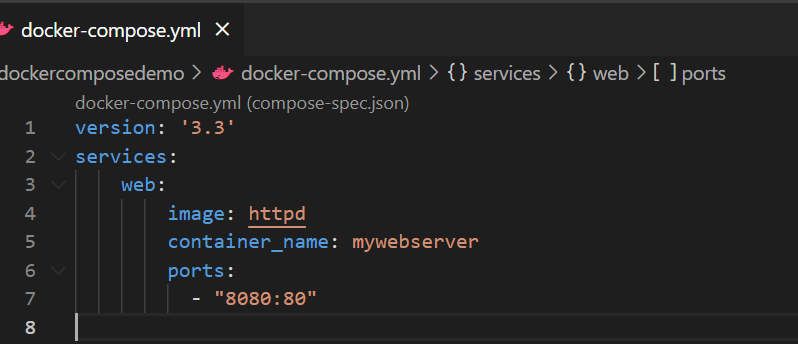
Note :

If docker-compose.yml file is correct then by using **docker-compose config** it will display it



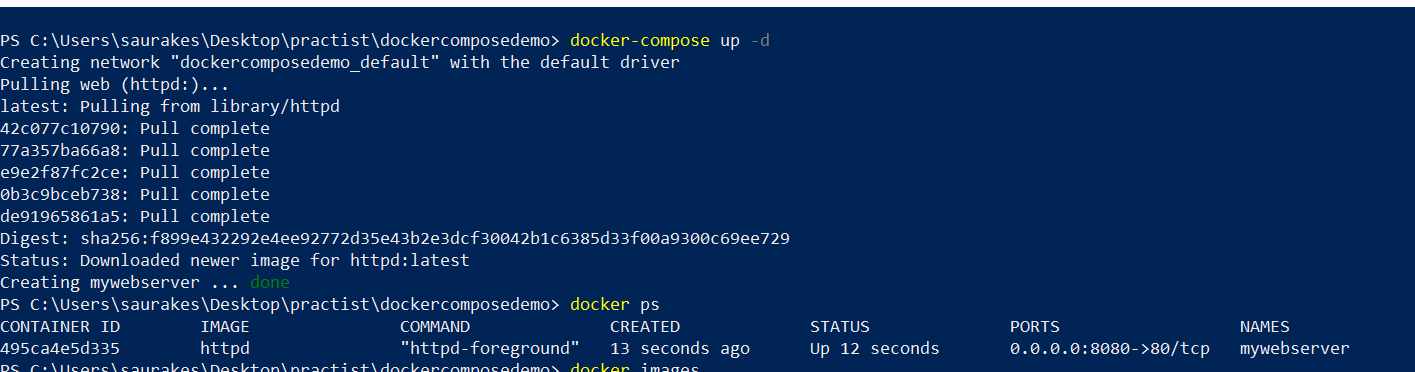
**What is detached mode in Docker?**

Detached mode, shown by the option --detach or -d , means that **a Docker container runs in the background of your terminal**.

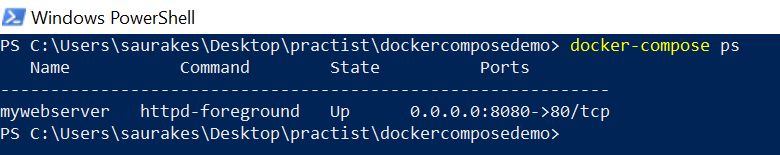


Docker-compose up -d

It will create default network



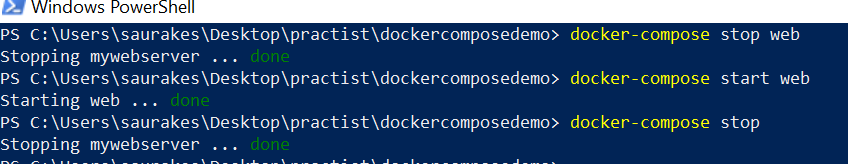
Docker-compose ps :-- it will show all the container through this file is running



*Docker-compose down :-- will down all the services that are running*

Docker-compose start web

Docker-compose stop web (web name of service)



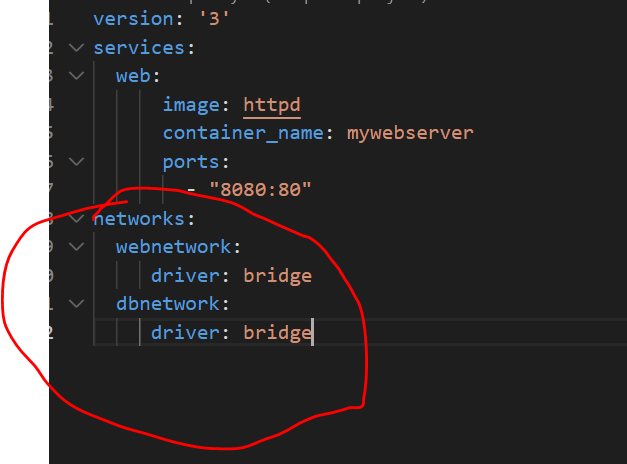
**Compose Networks**

**In previous lecture** when we run docker-compose up -d , it created a default network and it uses it for our services or container

**Objective**

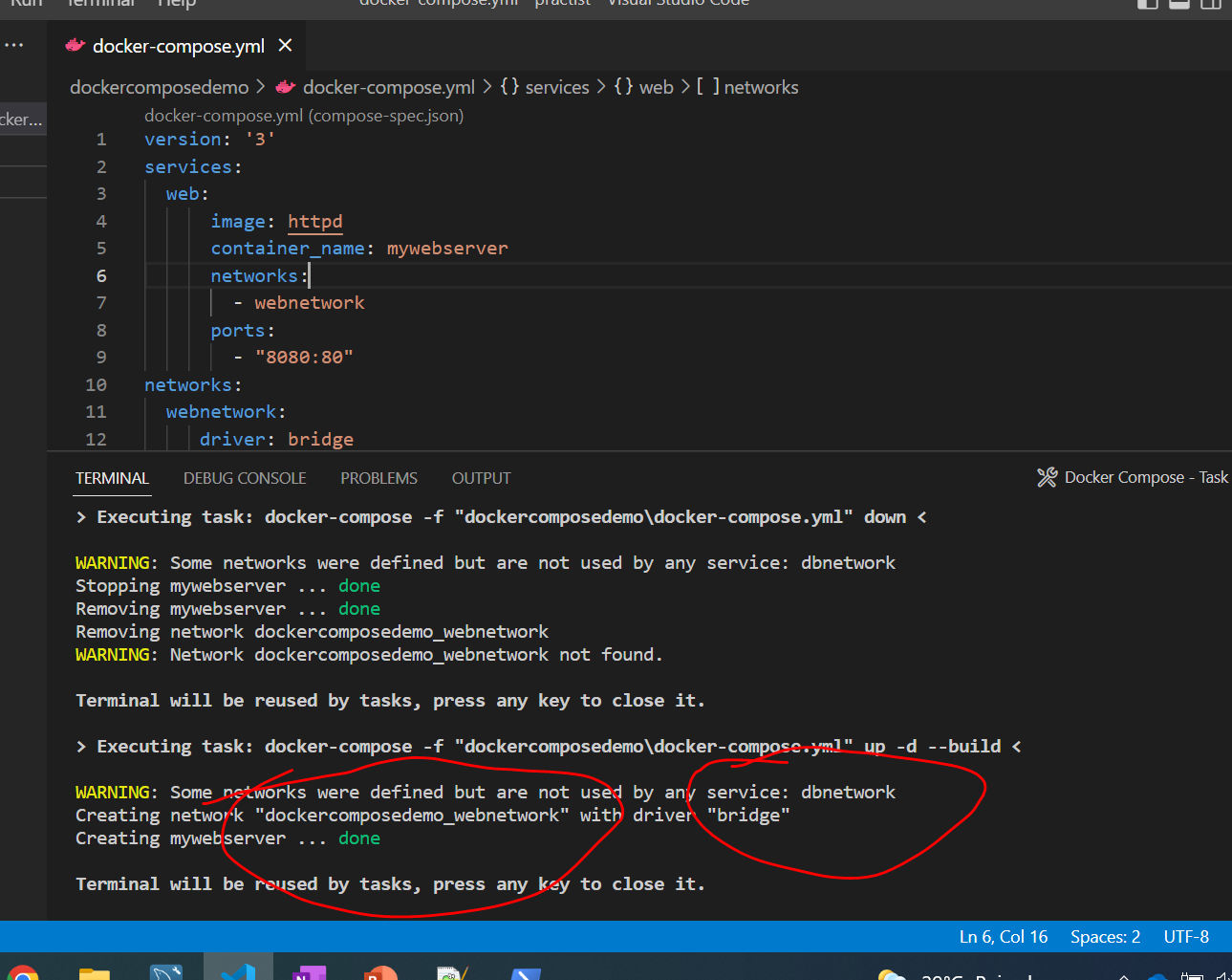
How to create our own network

This is like defining a network , we should use it otherwise it will not create



Here dbnetwork we defined but didn’t use it.

**We create network and assign container to network**

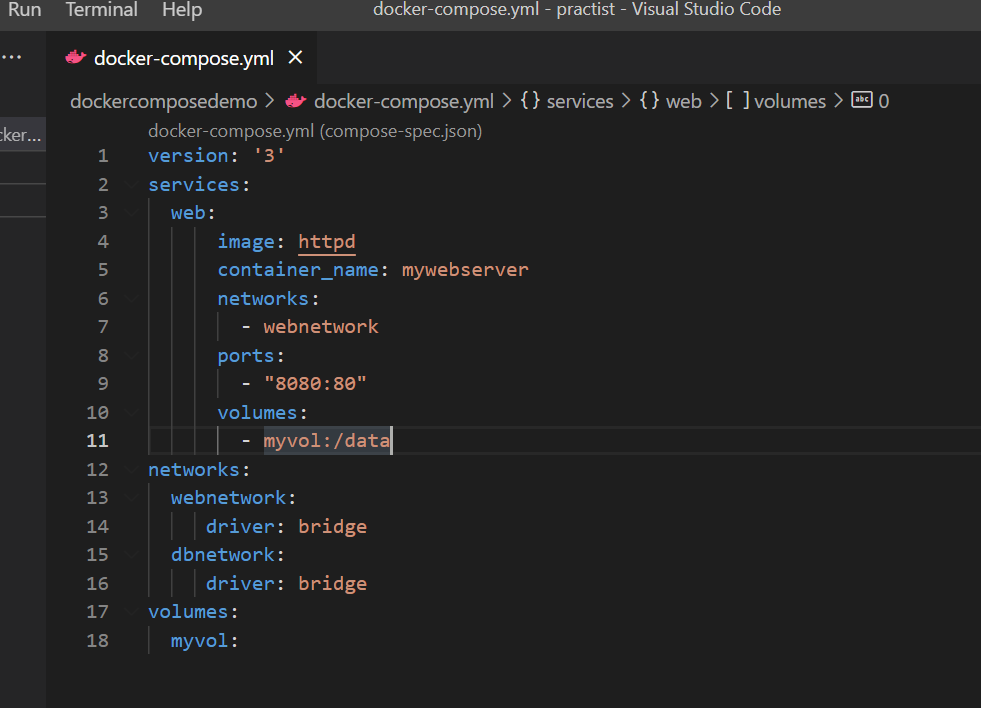


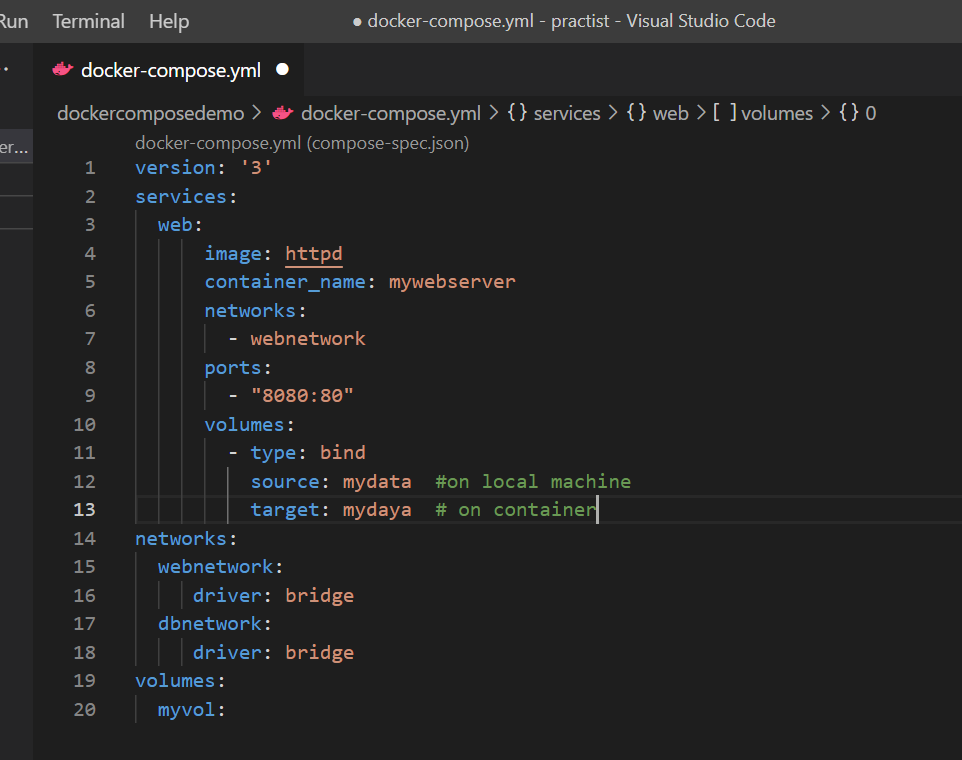
**Compose Volumes**

**Objective :-** how to mount volume to docker compose services

Just like networks we need to define volumes.

It will create a volume myvol under the docker folder and then it will mounted to data folder on the docker machine





**Compose Mysql Service**

Create docker-compose.yaml under couponservice

Define version

Docker-compose file hold group of services

**MYSQL\_ROOT\_HOST** :--- this property tell my sql server running inside the container which machine or which ip addresses

Access the my sql server running on the container % is wild card so we can access it from anywhere

We are mounting .sql under our project which is having tables.sql to a folder called **docker-entrypoint-initdb.d**

So my sql container automatically look this folder(docker-entrypoint-initdb.d) inside it

So when container come up mysql will search this folder and execute the query inside it

**Optional element**

Using health check we can make sure mysql server is up and running

Interval :4s --------it will check after every 4 second if health is up

Timeout: 20 sec if it dosent come up for 20 second

Retries : 5



**Compose Microservice**

**Important**

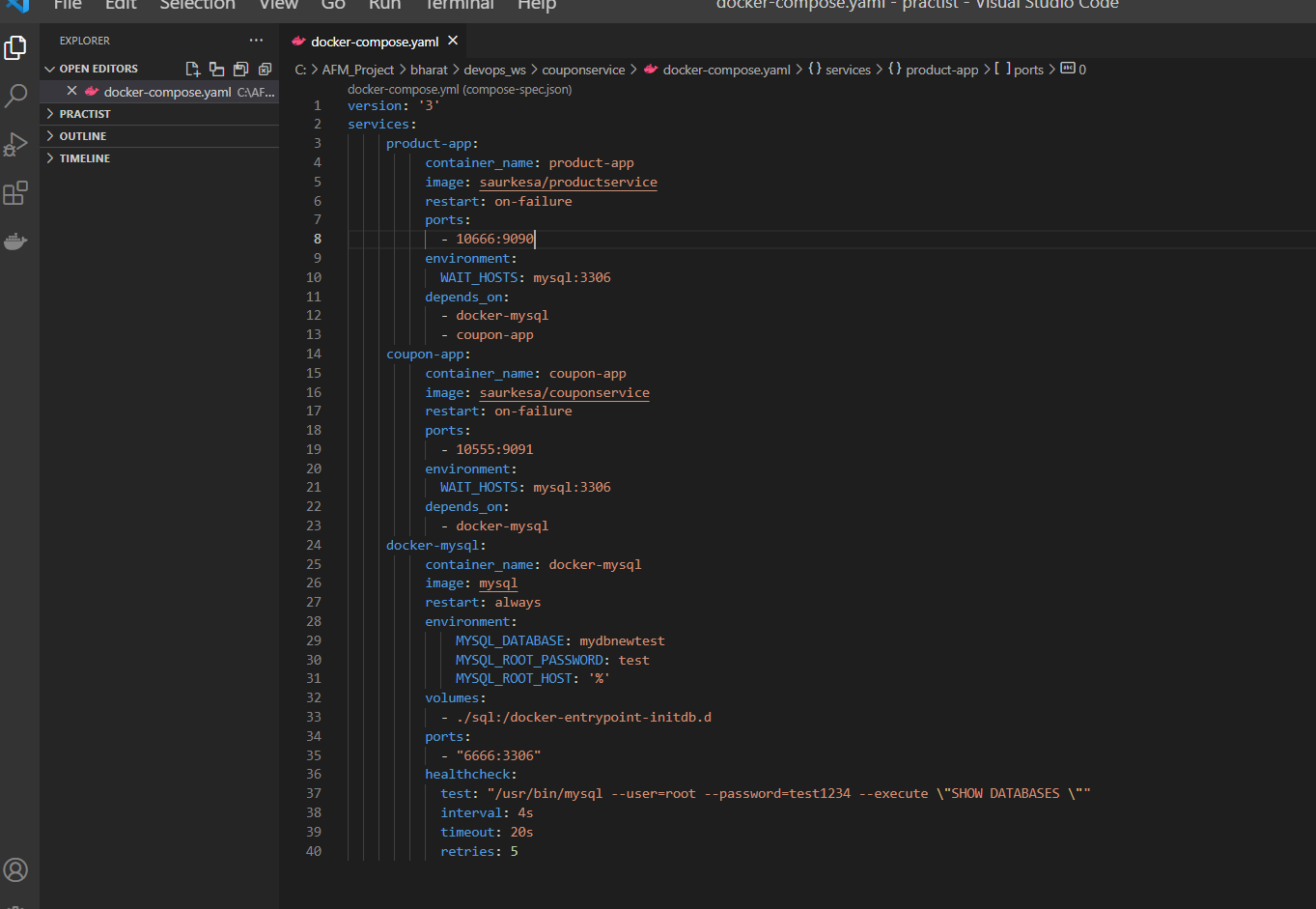
**To write in coupon m/s**

**Restart : on-failure** since if mysql service doesn’t come up yet then coupon m/s need to restart on failure

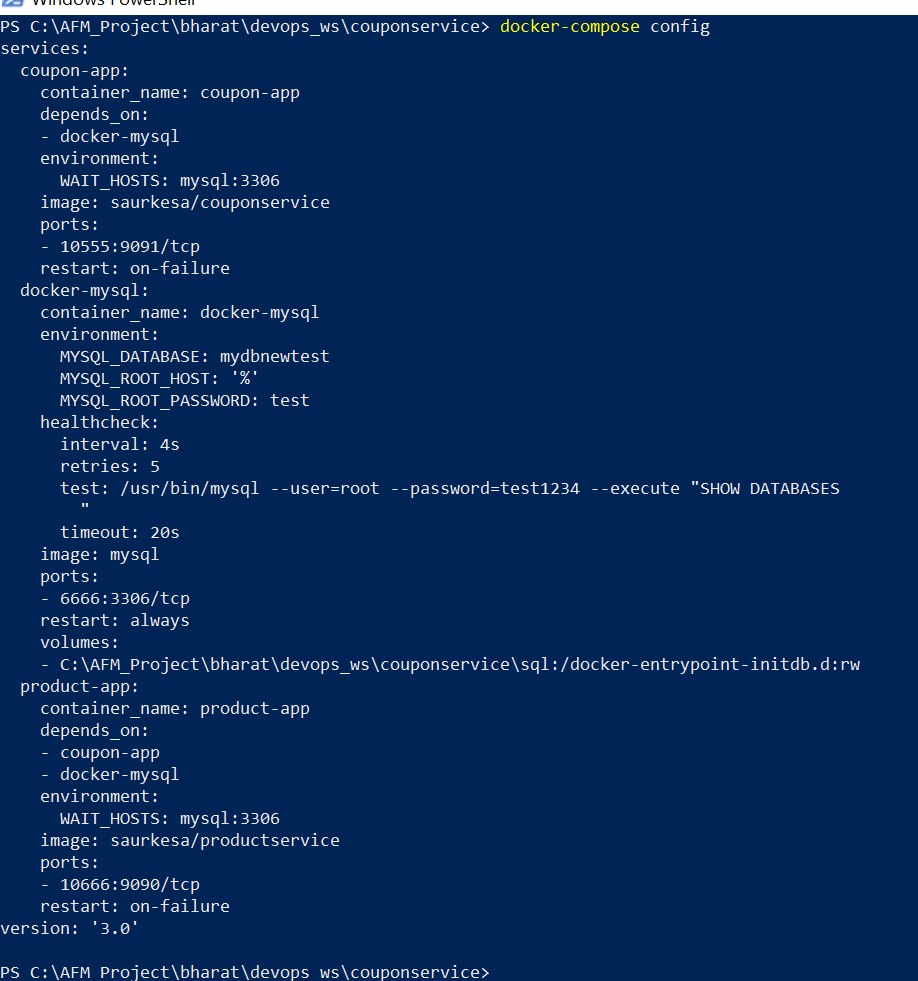
Under depends-on ::---- need to give name of the service

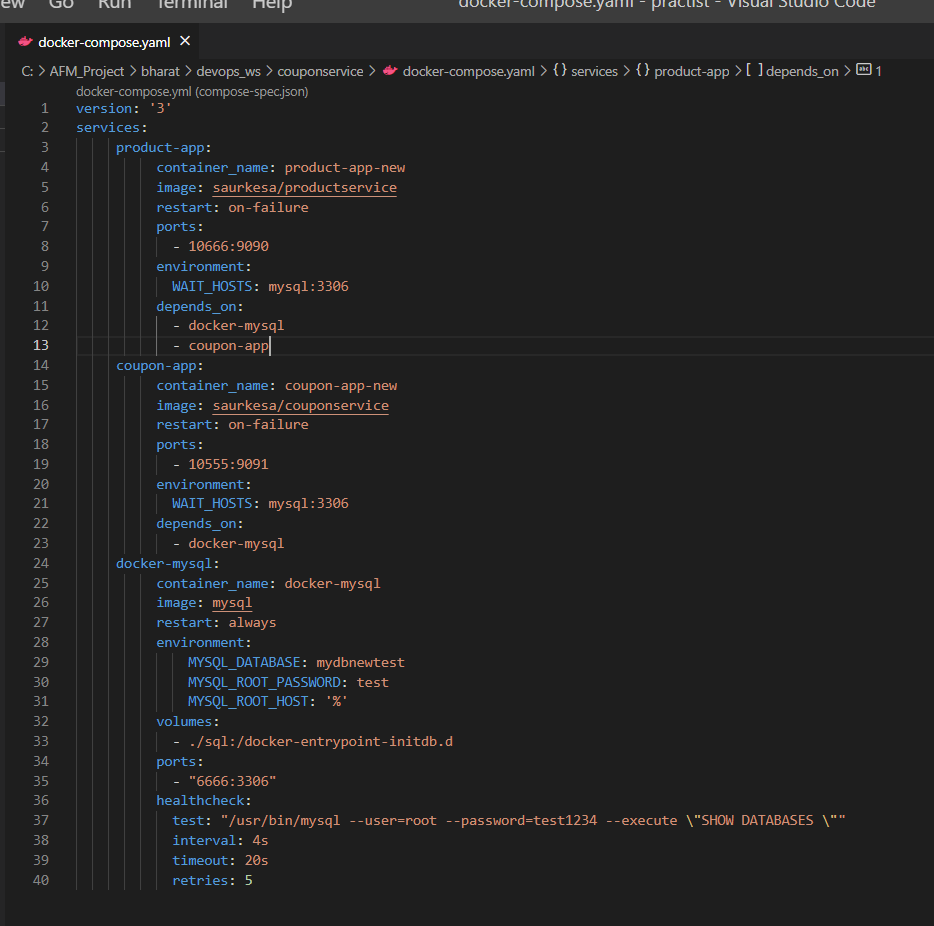
WAIT\_HOST : for which host this service need to wait

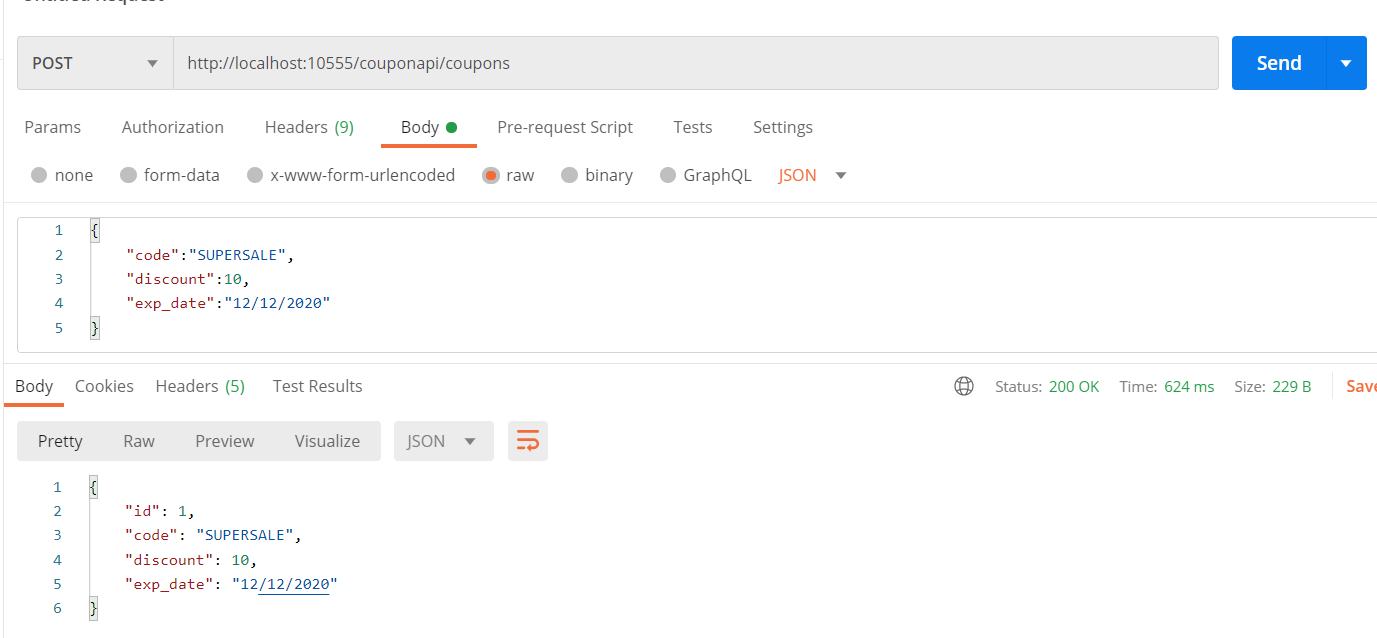
Imp

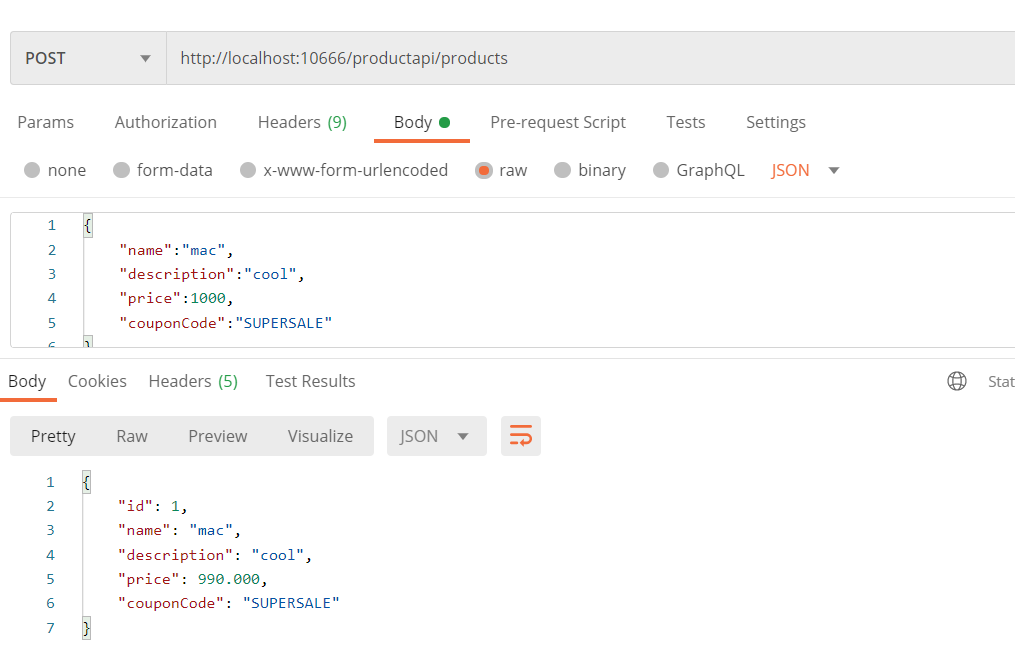


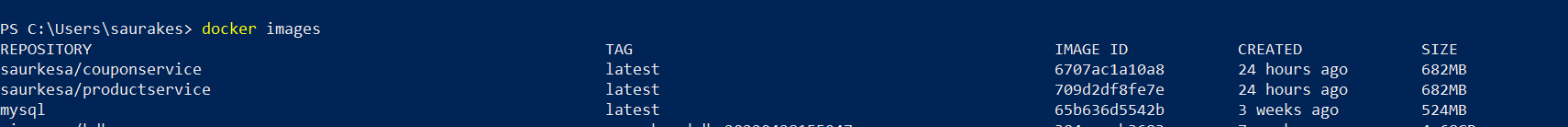
**Launch and Test**

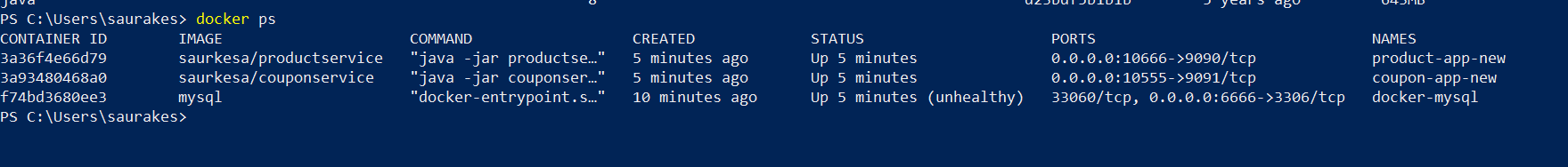


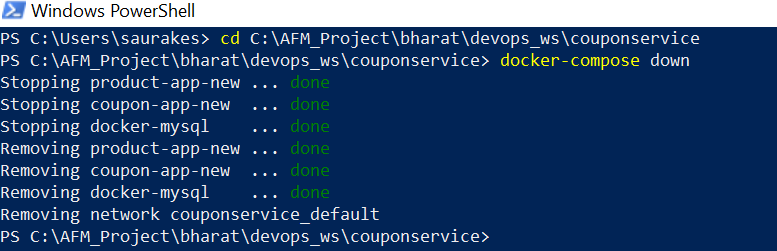












We can run docker compose even from visual studio but we need complete project

