Setup

- After creating a new VM, connect to it through powershell through ssh.
- INSTALL DOCKER :
  - `curl -fsSL https://get.docker.com -o get-docker.sh`
  - `sh get-docker.sh`
- Now type `docker info` and check whether are you able to see both client and Server versions ??
  - if NO :
    - we need to add our username to docker group, we can add by using below command.
    - `sudo usermod -aG docker Krishna`
    - after doing this exit from server and re-connect to it for effective results. `exit`
    - after re-connecting to it type `docker info`, now you will be able to see both Client and Server versions.

View Docker images and Containers

Images:

- To view Docker images in local repository
  - `docker image ls`
- To get list of image id's
  - `docker images -q`
- To delete all images in single shot
  - `docker rmi $(docker images -q) -f`

Container:

- To view Docker continer in local repo
  - `docker ps -a`
- To list of container id's
  - `docker ps -q`
- To delete all containers in single shot
  - `docker rm $(docker ps -q) -f`

download images

- To pull images from online repos that is Docker hub
  - `docker image pull _image-name_`
- If you want to download specific version of image you need to mention tag[:*tag-name*]
  - `docker image pull nginx:perl`

Create and Run container

Create and run container directly in single command

- `docker run -d -P nginx` [name and port will be assigned by docker]
- `docker run -P httpd` [image will be created in exited state, you can manually run the container]

Create Image and then create container with image

- Create Image/pull image
  - `docker image pull nginx`
- Create Container with ngnix
  - `docker container create --name abcd -P nginx`
- Now start container through id/name
  - `docker container start abcd`
- Create and Run container at a time and by assigning specific port
  - `docker container run --name nginx1 -d -p 5000:80 nginx`
  - To get inside of this container to use bash
    - `docker exec -it nginx1 bash`

Run JAVA Application in container

- we have image in docker hub with pre installed java , i choosed `amazoncorretto:17`
  - Pull image
    - `docker image pull amazoncorretto:17`
  - Create container and run it :
    - *create and then run* :
      - *create*:
        - `docker container create --name java2 -p 5003:8080 -it amazoncorretto:17`
      - *Run*:
        - `docker container start java2`
    - *Create and run in single shot*:
      - `docker container run --name java2 -d -p 5003:8080 -it amazoncorretto:17`
  - *Get inside of container to use bash*:
    - `docker exec -it java2 bash`
  - *Download the java Application*:
    - `curl -O _link_`
  - *Run the JAVA application*:
    - `java -jar _filename_`

Creating Image from Container

- To create image from contianer
  - `docker container commit _existing container name_ _new container name_:tag`
  - EX : `docker container commit java2 java3:1`

Docker file

- go to the folder where docker file exists
  - `docker build -f dockerfile -t spc:v1`
  - `docker buildx build -f dockerfile1 -t nop:v1`
    - -f refers to file name.
    - -t refers to image name.
- Useful command:
  - `docker stats` to check how much memory containers is utilised.

Run Container for Maven

- `docker container run --name spc -it maven:3.97-eclipse-temurin-11-alpine /bin/bash`

Docker Image layers

- Docker image is a combination of layers and these layers are only readable, whenever you create image all these layers are comibined together to form as disk.
- inorder to this for every image a writable layer will be created, Now all these layers are comibed to form as a disk
- Ref class notes for pictorial view
- whenever you create two images like nginx and httpd, when nginx is getting pulled you can observe the comment as pull complete next when you are pulling httpd you can observe that for some id's comment shows as Already exists. that means the existing layer is alredy persists
- when you create somthing in image this will be stored in writable layer
- if you create n containers the layers which exists already won't download again, it will use existing layer for wherever its needed.
- lets assume you are running two apps in two containers for the first application there are some layers got downloaded take it as A,B,C layers and when you are runnig app2 the layers which alredy exists it won't download it will use the app1 layers. by doing this it can reduce space. to achevie this each layer will be having unique id made of Hashing algorith SHA256.
- Now inorder to form a disk for a particular container all the necessary layers needs to be combined[a disk is a combination of layers], these comined layers will be stored in disk driver, type `docker info` form server version you will find a line with Storage Driver.

Impact

- whenever you write or store something in the container it will be stored in a writable layer and you remove a container this layer will get removed. due to this there is a problem created *how to keep data without loosing even after container is removed.*
- now if you store a db when you remove the container the db will also get removed.
- whenever you are deleting image that means you are deleting all layers.
- password are saved using this algorith, it will match your entered password with stored password.(if hash matches it will allow)

Inspection of image to find layers

- delete all the images and containers just for clear unerstanding.
- execute `docker pull alpine`
- `docker image inspect alpine` if you try to obseve there will be a layer created with hash value
- now write a sample docker file with
  - FORM alpine CMD ["ping","google.com"]
- build image and try inspecting this image and compare the both layer you will both are same
- if the layers are same it take from parent layers
- if you re write docker file with some changes that creates changes in image then new layers wiil be created
- change in docker file
  - FORM alpine RUN apk update ADD test / CMD ["ping","google.com"]
- create a file in test folder `touch test`
- now build the image and inspect it and compare the layers, you will find 3 layers

solution for Impact

```
ENV command
* ENV command is used to pass dynamic values when the container is running.
* to pass like username and passwords, Urls .. etc
* write a docker file for testing
* FROM alpine
  ENV APP="test"
  CMD ["sleep","1d"]
* build image and run container:
execute these commands to understand about ENV command
    * docker image build -f d2 -t i3 .
    * docker run -d --name envexp i3
    * docker container exec envexp  printenv
    * docker run -d --name envexp1 -e "App=nginx" i3
    * docker container exec envexp1 printenv
```

Create Db

- execute to create container
  - `docker container run -d -P -e 'MYSQL_ROOT_PASSWORD=admin' -e 'MYSQL_DATABASE=books' -e 'MYSQL_USER=aravindh' -e 'MYSQL_PASSWORD=admin' --`

`name abcd mysql:8.0-debian`

- `docker container exec -it abcd mysql -u aravindh -p` give enter and provide password
- `USE books;`
- `CREATE TABLE authors (id INT, name VARCHAR(20), email VARCHAR(20));`
- `INSERT INTO authors (id,name,email) VALUES(1,"Vivek","xuz@abc.com");`
- `SELECT * FROM authors;`

inorder to protect this data we need to find out on which folder the date is storing by browsing in the internet

- to protect we will create a volume outside(local) of docker and attach to inside of docker, even if you lost the container the volume will exists
- there are two ways to create volume
  - you can write some extra commands while creating container
  - you can write a docker file mentioning on creation of volume , this is a best practice, even if user is not aware of this when he executes the docker file it will automatically creates the volume

Experiment1

- create a volume in local
  - `docker volume create vol1`
- create a container and mount this volume to it
  - `docker container run -d -P -e 'MYSQL_ROOT_PASSWORD=admin' -e 'MYSQL_DATABASE=books' -e 'MYSQL_USER=aravindh' -e 'MYSQL_PASSWORD=admin' --name s1 -v vol1:/var/lib/mysql mysql:8.0-debian`
- now login to mysql and save some data as above we did
- remove container after doing it, and create new container with different name with same volume
- try login and check whether the data exists or not.
- you will find the data, using this way we can protect our data.

even if you won't give `-v` argument it will automatically creates an anonymous volume

- `docker container run -d -P -e 'MYSQL_ROOT_PASSWORD=admin' -e 'MYSQL_DATABASE=books' -e 'MYSQL_USER=aravindh' -e 'MYSQL_PASSWORD=admin' --name s3 mysql:8.0-debian`
- after executing this check volumes you will find an anonymous volume
- it is necessary to name a volume to find out it belongs to which one

Docker Networks

- ref class room notes for picture

- when we install docker we found that in `ifconfig` there is new network interface got created with a name of docker a bridge network for docker to local , like same we can create networks.

- to create a new netowork

  - `docker create -d bridge --subnet "10.100.0.0/16" nop-net`

- do `ifconfig` to check whether new interface is added or not

- attach this to a container and it

    - `docker container run -d -P -e 'MYSQL_ROOT_PASSWORD=admin' -e 'MYSQL_DATABASE=books' -e 'MYSQL_USER=aravindh' -e 'MYSQL_PASSWORD=admin' --name s1 --network nop-net mysql:8.0-debian`

- do `docker inspect s1` at last find the ip it will be in the range of what we have given.

checking communication between two containers

- if you create two containers with default network interface when you are checking communication you need check through ip address
- if you create a new network interface and create a container with that network this allows to communicate using container names

Default networks:

- `docker run --name d1 -d alpine sleep 1d`
- `docker run --name d1 -d alpine sleep 1d` To check communication:
    - `docker exec d1 ping -c 4 _d2_ip_`
    - `docker exec d2 ping -c 4 _d1_ip_`

newly created network:

- `docker run --name n1 -d --network nop-net alpine sleep 1d`
- `docker run --name n2 -d --network nop-net alpine sleep 1d` To check communication:
    - `docker exec n1 ping -c 4 n2`
    - `docker exec n2 ping -c 4 n1`