* **Reference video:** [Youtube](https://youtu.be/3c-iBn73dDE)
* **What is docker?**
  + Docker uses the OS kernel of the host machine where the docker container is running eg: our server
  + Docker virtualizes the applications not the OS
  + If the docker image is not compatible with our local machine OS versions then use docker toolbox to settle up the conflicts of the OS.
* **What is the container?**
  + layers of images
  + first layer will be of the linux based image for OS environment
  + One container may need more than one images download each image separately
  + For egdata\_base image, etc.
  + Images are available on docker hub for free
  + Container is the running state of the images.
  + The upper most image will be our own application running in the container.
  + container gives the environment to the image to actually run.
* **Difference between VM and container?**
  + VM virtualizes both OS and application, but container virtualizes the app running on it and container will use the OS enviorment of the host\_machine itself(host machine can also be one VM running on our laptop or running on the AWS cloud i.e EC2 instance).
* **Advantage of the containers:**
  + Let's say our application uses 2 diff types of mySQL versions. Then I can make one container running MYSQL V1 and another container running MYSQL V2.
* **What is an image?**
  + Image is the actual package that contains all the dependencies, environment, configurations, application code needed to run the image.
* **Difference between container and image**
  + Container is the running environment for image
  + Means container will give the environment, file system needed to store some data, etc to the image(our application or predefined some image downloaded from the docker hub).
  + Container will bind one port to the image running on the container.
* **What do we mean by port mapping of containers?**
  + On 1 host machine/ VM more than 1 docker container can run parallely.
  + 2 containers can run on the same port in the same host machine.
  + Now question arises that how will the API call bifurcated between the

2 containers(basically 2 images running on different containers).

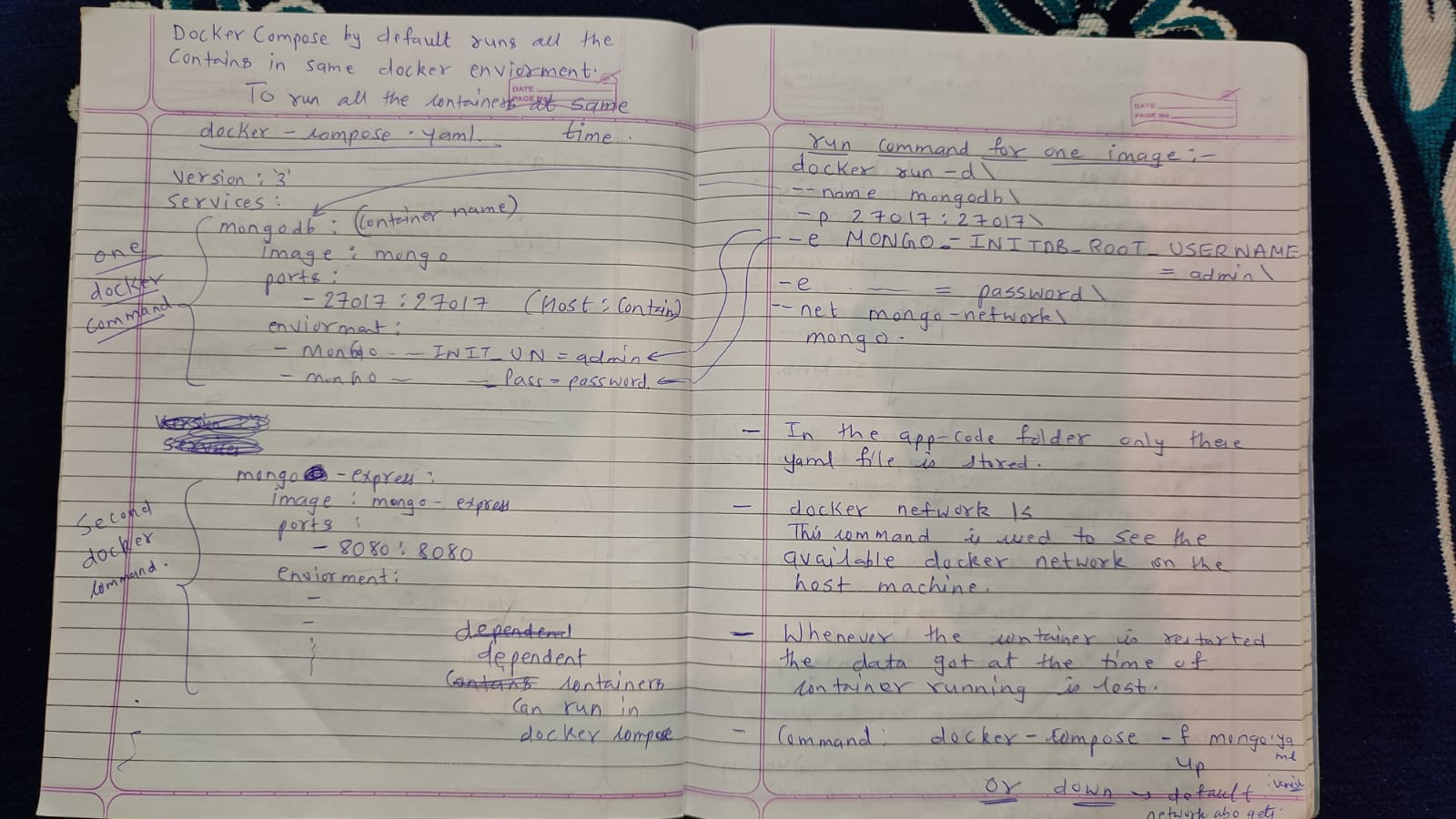
* + Now let's say there are 2 containers running on the same port 3000.
  + Here the concept of binding the host port to the container port comes in.
  + Whenever a container starts running on a VM/host machine, the machine will assign one port to that container(eg: 5000). Now, this 5000 port will be mapped with one of the ports of the container(eg:3000).
  + Now for the other container having the same port(eg: 3000) will be binded to another port number assigned by the machine(eg: 6000).
  + So, whenever the request comes it will come on the port of the host machine/ VM not directly to the container.
  + According to the mapping done between the machine port and container port(basically image is being binded to the host port) the request will be forwarded to that specific container(image-our sample app) by the host machine/ VM.
  + It means the API calls will be made on the port number of the host machine/ VM and not the actual container.
* **What is stored in the docker container?**
  + Docker uses storage drivers to store image layers, and to store data in the writable layer of a container.
  + The container's writable layer does not persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime.
* **General Points**
  + Whenever we are deploying the app the database used by that app will run on different containers.
  + It simply means that let's say we have one java-script app and that app uses mongodb as the data-base for the storage.
  + We will use the image from the docker hub of the mongo db directly. For our app code one separate custom docker image will be formed which will be stored into some private docker repository of the organisation.
  + Now when we want to run both the images, server/host\_machine will pull both the images mongo-db from the public docker hub and app-code image from the private docker repository and each image will be deployed on individual separate containers running parallely.
  + There will be some intermediator called jenkins which will make the docker image of our app-code based on the given artifacts of the app-code. (artifacts like the environment info, version info of the language used by the application, etc)

**DOCKER COMMANDS TABLE**

| **When to use** | **Command** |
| --- | --- |
| build image | docker build -t <image\_name>:<tag> <path\_to\_Dockerfile> |
| run image | docker run -p <host\_port>:<app\_port> <image\_name>:<tag> |
| list running containers | docker ps |
| list all containers | docker ps -a |
| start container | docker start <container\_name/id> |
| stop container | docker stop <container\_name/id> |
| remove container | docker rm <container\_name/id> |
| remove image | docker rmi <image\_name>:<tag> |
| remove all containers | docker rm -f $(docker ps -aq) |
| remove all images | docker rmi -f $(docker images -aq) |
| get container id | docker container ls –quiet –filter name=^<container\_name>$ |
| get all info of container | docker inspect <container\_name/id> |
| check container status | docker inspect -f ‘{{.State.Status}}’ <container\_name/id> |
| live logs of container | docker logs <container\_name/id> -f |
| go inside file-system of container | docker exec -it <container\_name/id> /bin/sh |

**Reference:** [Docker commands official](https://docs.docker.com/engine/reference/commandline/docker/)

**Additional Images**

****

