

Docker :

Container :

1. A way to package application with all the necessary dependency and configuration.
2. Portable can easily share between other member or client.
3. Make development and deployment easy.
4. Live in container repository (that can be private or public e.g., Docker Hub)
5. Own isolated environment.
6. Package with all needed configuration and one command to install the app.

So basically container is the layers of the images, mostly Linux base image because of small size.

Application image on the top.

Difference in Virtual Machine and Docker :

- Docker virtualise the application layer of the OS. As it uses the host kernel as it does not have its personal.
- VM virtualise at the OS kernel layer. It uses its own kernel so uses more hardware and computational power.

Container and Image:

- Container is a running environment for image.
- Application images are the applications like Postgres, redis, Django etc
- Container consists of port mapping: talk to application running inside the container.
- Images can exist without containers, whereas a container needs to run an image to exist. Therefore, containers are dependent on images and use them to construct a run-time environment and run an application. The two concepts exist as essential components (or rather phases) in the process of running a Docker container.

Basic Command :

Image Installation:

```
docker pull image_name:image_version
```

- Here image_name can be the name of the application image like postgres, redis, django etc.
- And image version is the version of the package, leave blank if need latest version

All Image:

`docker images`

Shows all the images available or downloaded.

Here is the some command cheats below :

Cheatsheet for Docker CLI			
Run a new Container	Manage Containers	Manage Images	Info & Stats
<pre>Start a new Container from an Image docker run IMAGE docker run nginx ...and assign it a name docker run --name CONTAINER IMAGE docker run --name web nginx ...and map a port docker run -p HOSTPORT:CONTAINERPORT IMAGE docker run -p 8080:80 nginx ...and map all ports docker run -P IMAGE docker run -P nginx ...and start container in background docker run -d IMAGE docker run -d nginx ...and assign it a hostname docker run --hostname HOSTNAME IMAGE docker run --hostname svr nginx ...and add a dns entry docker run --add-host HOSTNAME:IP IMAGE ...and map a local directory into the container docker run -v HOSTDIR:TARGETDIR IMAGE docker run -v ~/:/usr/share/nginx/html nginx ...but change the entrypoint docker run -it --entrypoint EXECUTABLE IMAGE docker run -it --entrypoint bash nginx</pre>	<pre>Show a list of running containers docker ps Show a list of all containers docker ps -a Delete a container docker rm CONTAINER docker rm web Delete a running container docker rm -f CONTAINER docker rm web Delete stopped containers docker container prune Stop a running container docker stop CONTAINER docker stop web Start a stopped container docker start CONTAINER docker start web Copy a file from a container to the host docker cp CONTAINER:SOURCE TARGET docker cp web:/index.html index.html Copy a file from the host to a container docker cp TARGET CONTAINER:SOURCE docker cp index.html web:/index.html Start a shell inside a running container docker exec -it CONTAINER EXECUTABLE docker exec -it web bash Rename a container docker rename OLD_NAME NEW_NAME docker rename 096 web Create an image out of container docker commit CONTAINER docker commit web</pre>	<pre>Download an image docker pull IMAGE[:TAG] docker pull nginx Upload an image to a repository docker push IMAGE docker push myimage:1.0 Delete an image docker rmi IMAGE Show a list of all Images docker images Delete dangling images docker image prune Delete all unused images docker image prune -a Build an image from a Dockerfile docker build DIRECTORY docker build . Tag an image docker tag IMAGE NEWIMAGE docker tag ubuntu ubuntu:18.04 Build and tag an image from a Dockerfile docker build -t IMAGE DIRECTORY docker build -t myimage . Save an image to tar file docker save IMAGE > FILE docker save nginx > nginx.tar Load an image from a tar file docker load -i TARFILE docker load -i nginx.tar</pre>	<pre>Show the logs of a container docker logs CONTAINER docker logs web Show stats of running containers docker stats Show processes of container docker top CONTAINER docker top web Show installed docker version docker version Get detailed info about an object docker inspect NAME docker inspect nginx Show all modified files in container docker diff CONTAINER docker diff web Show mapped ports of a container docker port CONTAINER docker port web</pre>

For more details regarding docker command visit :

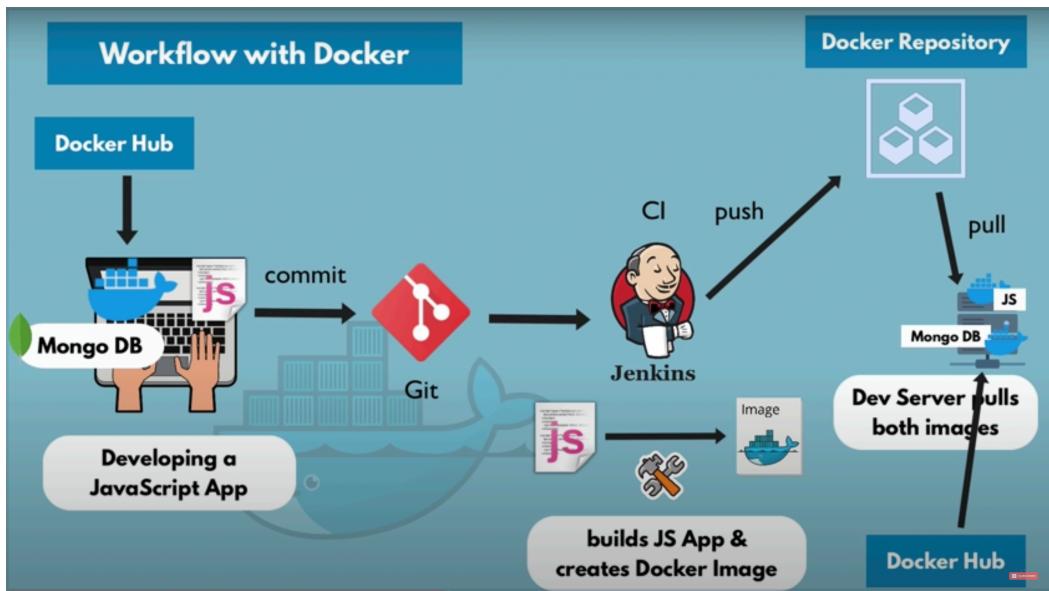
The Ultimate Docker Cheat Sheet | do...

Docker – Beginners Intermediate Advanced
dockerlabs.collabnix.com



Docker run is used to create a new container for the image but docker start is use to restart the existing container using its name or container id.

Docker Workflow :



Docker Compose:

It is a structure way to contain basic commands of the docker. We use docker compose as it is easy to change in a file like environment variable etc. So we create a docker compose file inside we add our docker commands. We define the version, image name, ports, name, environmental variable in the file.

We don't mention the network in the docker compose as it itself take care of the common network.

File extension of docker compose file is
 "file_name.yml"

Example file ;

```
version: '3'
services:
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password
  mongo-express:
    image: mongo-express
    ports:
      - 8080:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
      - ME_CONFIG_MONGODB_SERVER=mongodb
```

Indentation in yaml-File
is important!

Using docker compose :

We can use docker compose using the command,

```
'docker-compose -f file_name.yml up'
```

Here, -f denotes the file and up means start the container, to stop container use down instead of up.

When we restart the container every configuration and data of the container will be lost. And it creates new network every time when we start the container and deletes the network when we stop the container.

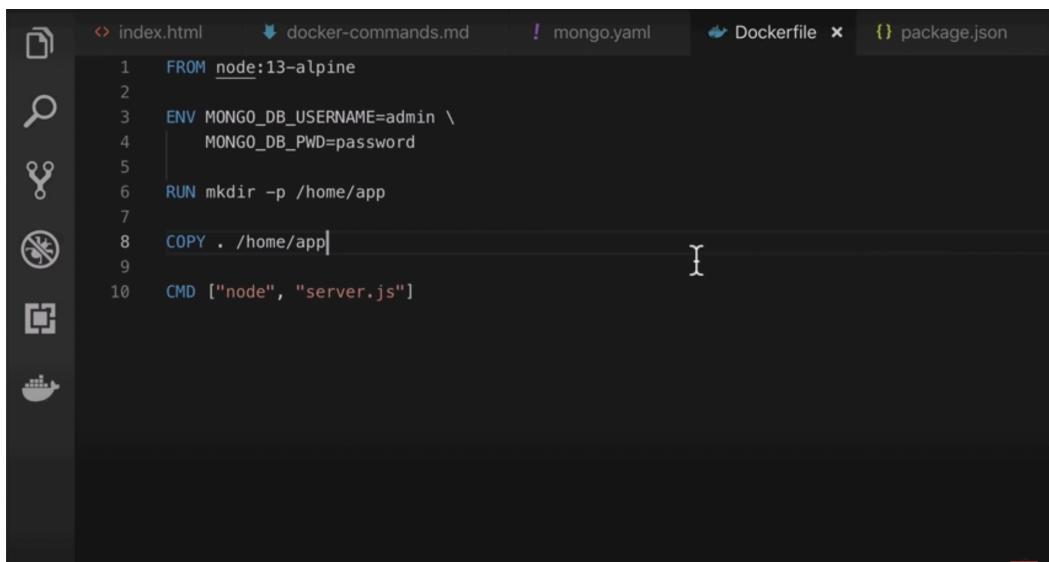
Docker File :

It is a blueprint for building the images.

When we deploy our application on the server before that we create the docker image of the application. So to achieve this we create the docker file. This file contains the artefacts of the application.

Image Environment Blueprint	DOCKERFILE
install node	FROM node
set MONGO_DB_USERNAME=admin set MONGO_DB_PWD=password	ENV MONGO_DB_USERNAME=admin \\\nMONGO_DB_PWD=password
create /home/app folder	RUN mkdir -p /home/app
copy current folder files to /home/app	COPY . /home/app
start the app with: "node server.js"	CMD ["node", "server.js"]
CMD = entrypoint command	
You can have multiple RUN commands	
blueprint for building images	

Example File :



```

1  FROM node:13-alpine
2
3  ENV MONGO_DB_USERNAME=admin \
4      MONGO_DB_PWD=password
5
6  RUN mkdir -p /home/app
7
8  COPY . /home/app
9
10 CMD ["node", "server.js"]

```

Docker file name should be "Dockerfile"

After creating the docker file, to build the image use the following command,

"docker build -t image_name:tag "

Here this command takes 2 parameter,

First is -t which is tag name, here we have to give image name and in tag we can give version-1 or 1.0

Second parameter is the allocation of the docker file. If the directory is same folder as Dockerfile so we can use " . ".

"App including Dockerfile is committed to git"

"When we adjust the docker file, we must rebuild the image."

Docker Volume :

Docker Volume is used for the data persistence.

Basically if we restart or stop the container, then the data in the container will get removed or deleted. So to resolve that issue we use the docker volume.

So what docker volume does is, it connects the host machine physical file system with the directory of the folder in the virtual file system of the docker or can say connects with the container.

As a result data gets automatically replicated from the virtual file system to the host file system and vice versa.

So we can use volume in the following way.

Using Command :

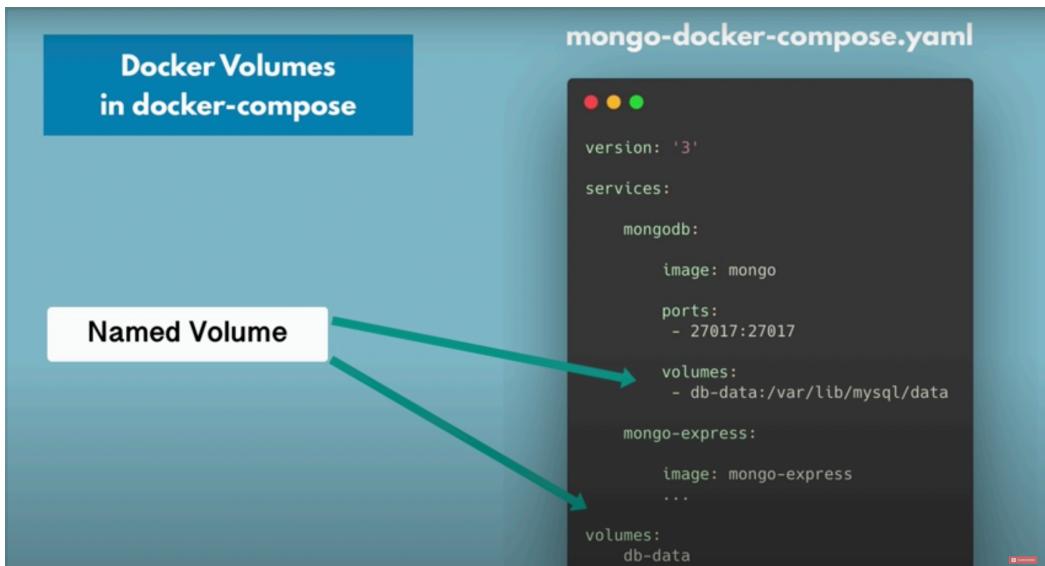
So in command we specify the -v property in the run command. i.e.,

"Docker run -v host_file_directory:virtual_file_directory image_name"

So here we can give host file directory Or the folder name in the place of host_file_directory

We can also set it blank, in that case docker will take care of the host file directory.

Using Docker Compose :



```

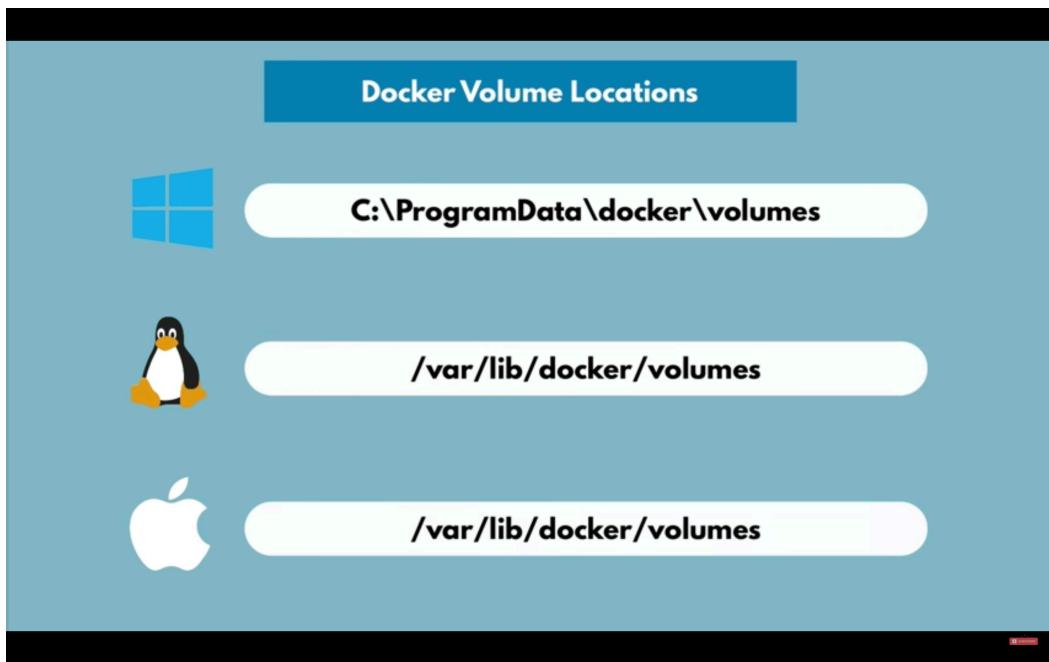
version: '3'
services:
  # my-app:
  #   image: ${docker-registry}/my-app:1.0
  #   ports:
  #     - 3000:3000
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password
    volumes:
      - mongo-data:/data/db
  mongo-express:
    image: mongo-express
    ports:
      - 8080:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
      - ME_CONFIG_MONGODB_SERVER=mongodb
  volumes:
    mongo-data:
      driver: local

```

mysql: var/lib/mysql
postgres: var/lib/postgresql/data

Here the volume define at service level and used in the image level.

Data Volume Location



Docker for Mac creates a Linux virtual machine and stores all the Docker data here.