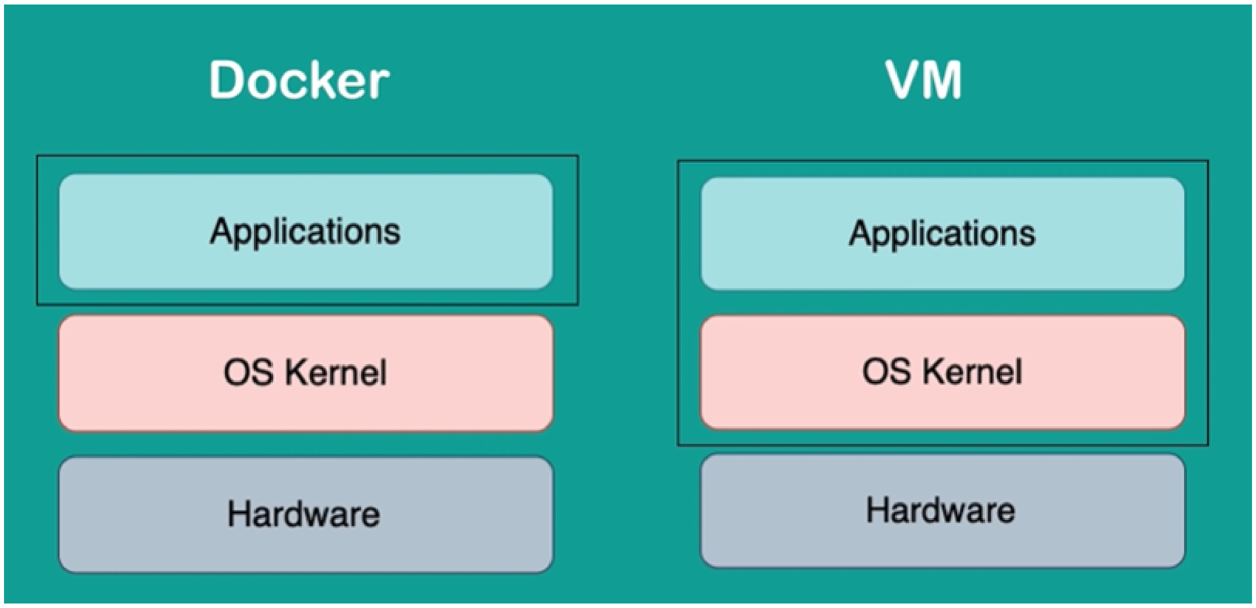
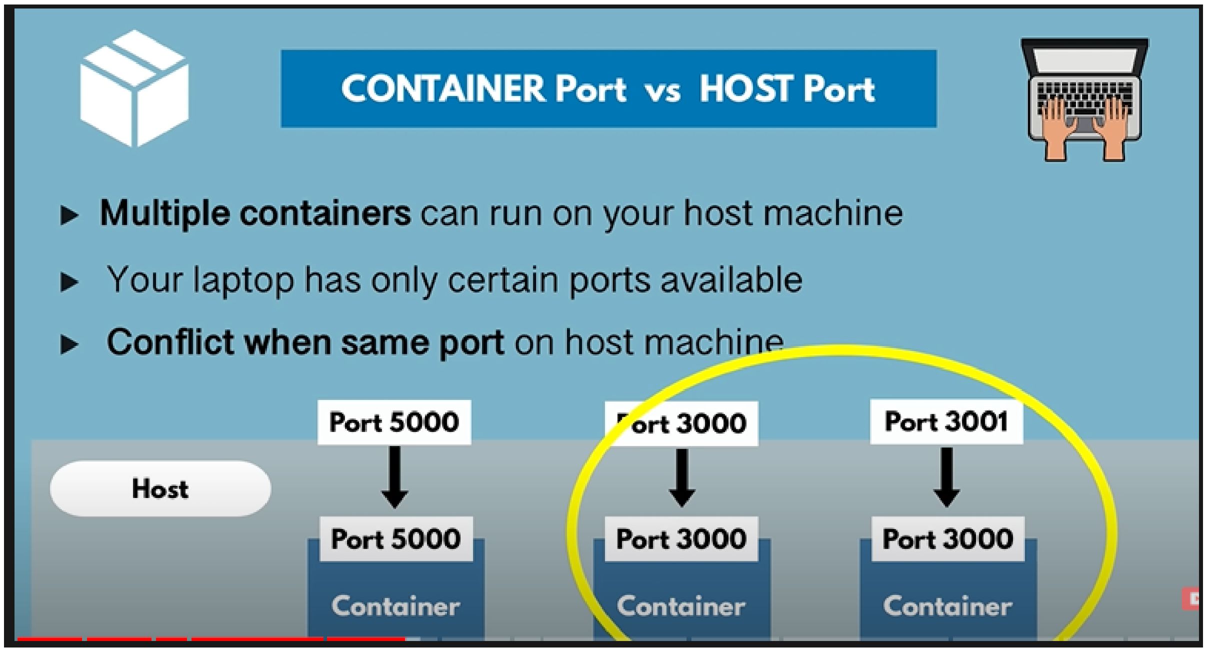
Docker Nana:
Docker virtualizes Application layer only while VM virtualize Applications layer and OS kernel layer.



🚀 5. Main Docker Commands
► docker pull, docker run, docker ps, docker stop, docker start, port mapping


CONTAINER Port vs HOST Port

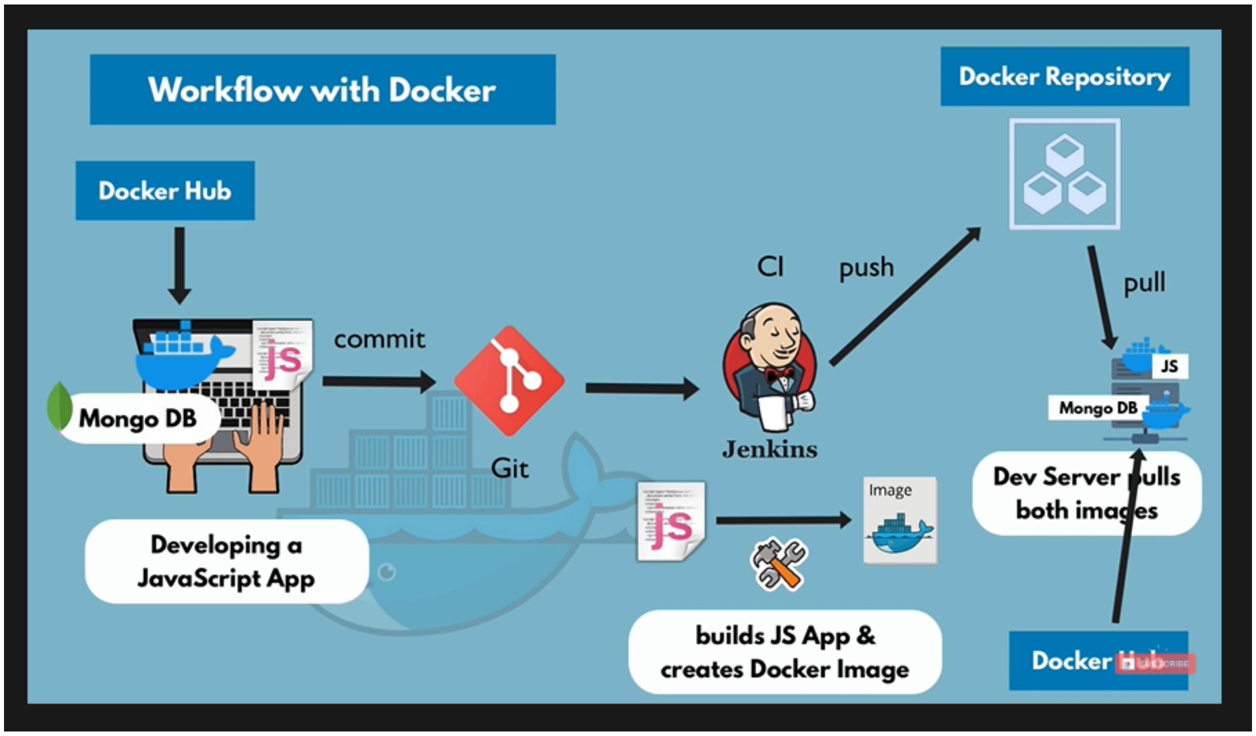
- **Multiple containers** can run on your host machine
- Your laptop has only certain ports available
- **Conflict when same port** on host machine

The above is port binding -connecting local host port to container port,If repeated then we bind with different local host port.
for the above you will visit localhost:3001
docker run -p 3001:3000 -d redis //first one of host,second of container

🚀 6. Debugging a Container
▶ docker logs, docker exec -it

docker exec can get terminal of running container, -it means interactive terminal
docker exec -it panda /bin/bash

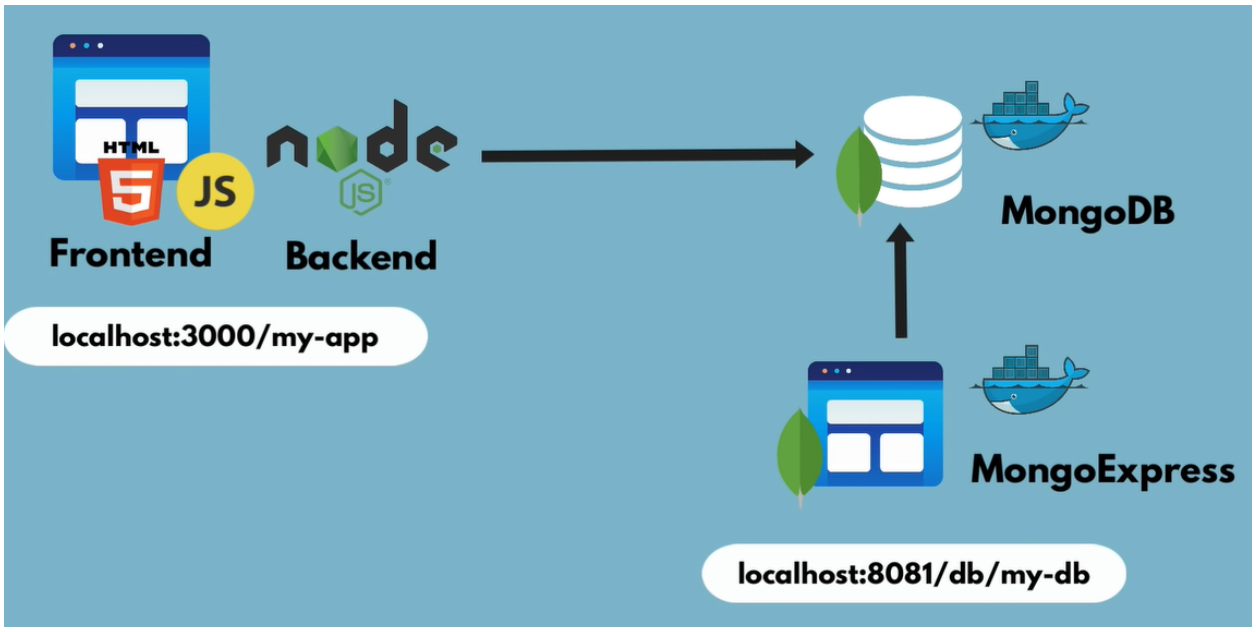
🚀 7. Demo Project Overview - Docker in Practice (Nodejs App with MongoDB and MongoExpress UI)



The dev will be an image pulled from Docker repisotry that you made with js, and a ready made Mongo DB from docker hub pulled

🚀 8. Developing with Containers
▶ JavaScript App (HTML, JavaScript Frontend, Node.js Backend)
▶ MongoDB and Mongo Express Set-Up with Docker
▶ Docker Network concept and demo

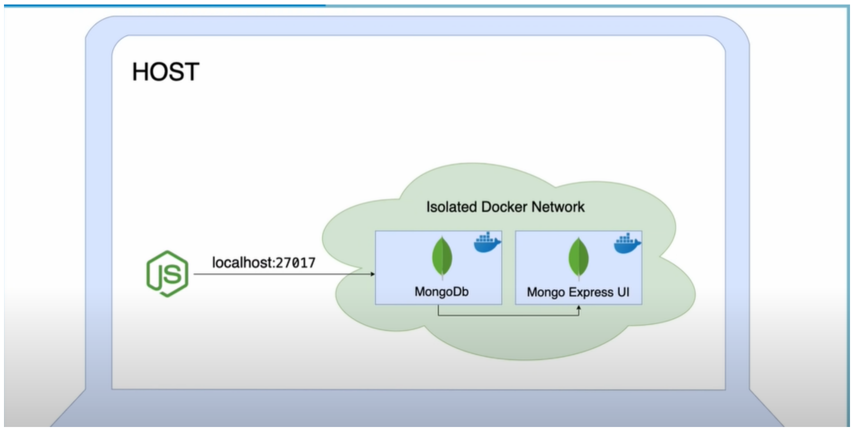
creating frontend app with mongo db



Mongo Express is mongo db User interface so that we can deal with the database.
We have index.html and server.js when you edit you can make changes but once refresh it will delete it so we will have to set mongo db with mongo express to check db

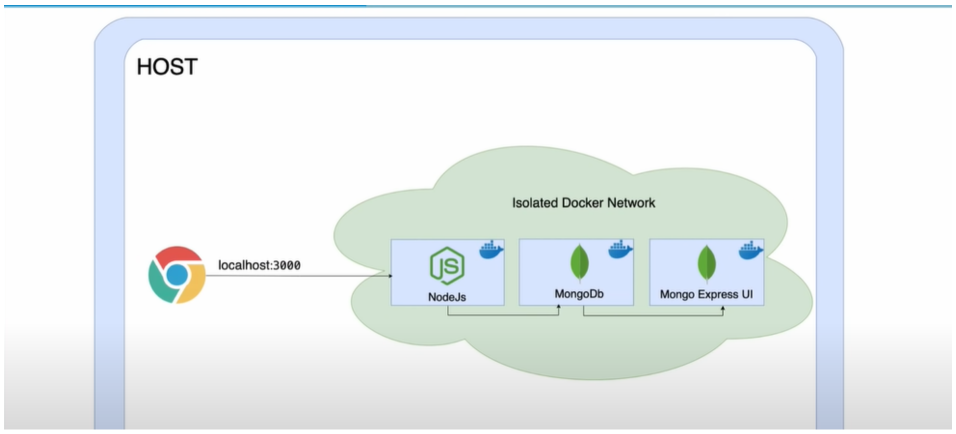
Step 0"local test":
Installed mongo from mongo website, installed express.
Commented container class in css as it was making an issue
npm -v   node -v   express -v   ///to make sure they are installed
node server.js   //to run it
//opened port localhost:3000 and yaaaaayyy :)

step1: go to docker hub , **docker pull mongo**, **docker pull mongo-express**
Then run both containers
step2: Make connection between these 2 containers. They can talk to each others with just container name as they are both on same isolated docker network. Javascript app will connect to them with local host and port number. As in below figure.



When we package an app to its own docker image it will go inside the docker isolated network also :) , So there will be a container for the nodejs app

So it will be only the browser that will connect to our Javascript application from outside isolated docker network.



So already some networks on docker are provided.

docker network ls

//We will create our own network for mongoDb and mongoExpress..called mongo network

docker network create mongo-network

docker network ls //created :)

**docker run -d **
**-p 27017:27017 **
**-e MONGO_INITDB_ROOT_USERNAME=admin **
**-e MONGO_INITDB_ROOT_PASSWORD=passme **
**--name mongodb1 **
**--net mongo-network **
mongo

//27017 is port for mongo, you have to set environments variable ,you can see it on docker hub

//net is for network to connect it to our network

docker logs mongodb1 //To check what is happening

//////////////Now we will start mongo express… you can check documentation on docker hub also

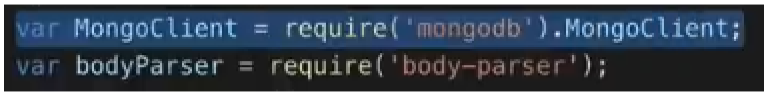
docker run -p 8082:8081 -d -e ME_CONFIG_MONGODB_ADMINUSERNAME=admin -e ME_CONFIG_MONGODB_ADMINPASSWORD=passme
-e ME_CONFIG_MONGODB_SERVER=mongodb1
–name mongo-express1 - -net mongo-network mongo-express

//notes on above config mongodb server takes the name of the previous mongo db container
docker logs mongo-express1 //to check connected or not
//you can open **localhost:8082** and you will see mongo express :)

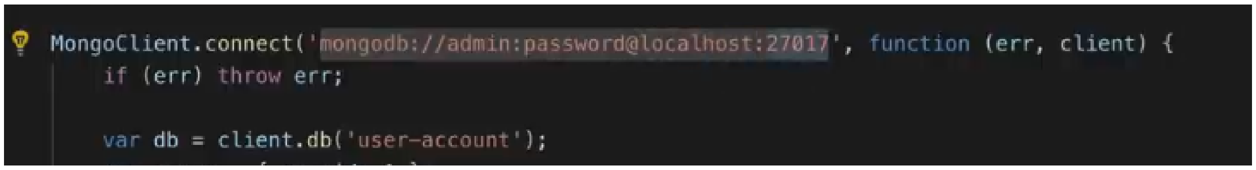
create new database called “user-account” from web
we will connect to “user-account” database from node.js
Now 2 containers are running

step 3: Connect node.js with database “user account”

in server.js



```
var MongoClient = require('mongodb').MongoClient;
var bodyParser = require('body-parser');
```



```
MongoClient.connect('mongodb://admin:password@localhost:27017', function (err, client) {
    if (err) throw err;

    var db = client.db('user-account');
```

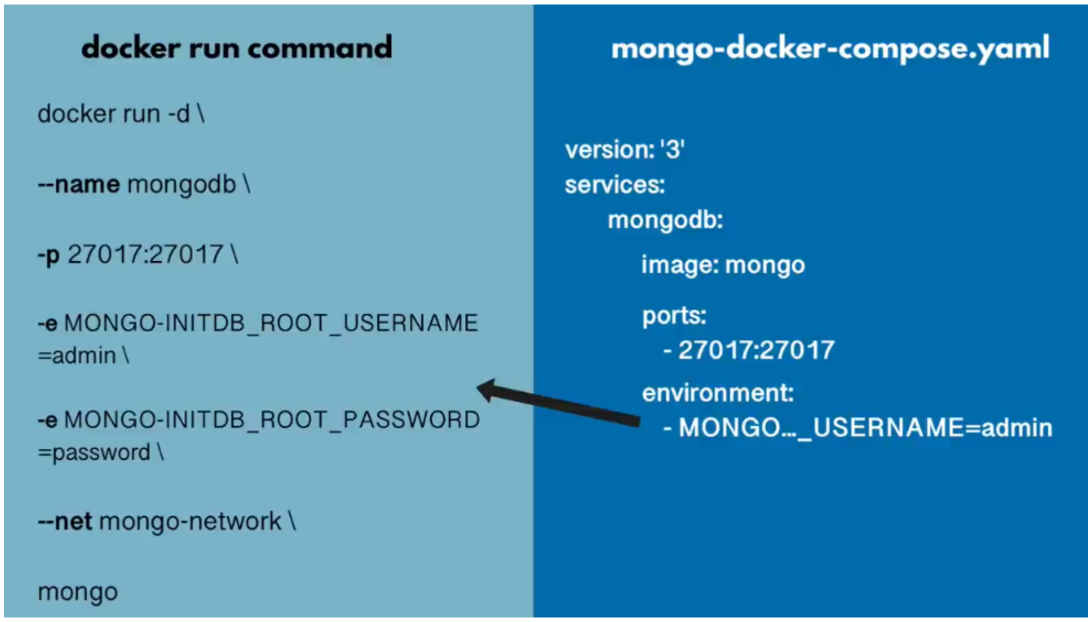
using port, protocol, user name and password to connect as above should be done for update, delete,add….
We have to give protocol of database and URI of mongo db which is local host and port.
That is configured in server.js

localhost:8000 make new changes and save it ,you will find it in database :)
docker logs mongo //to check logs or docker logs mongo | tail or
docker logs mongo -f //keeps live streaming

000000000000000000000000000000000000000000000000000000000000000000000000000000
The above example had 2 docker containers mongodb and mongo express while js was outside….

🚀 9. Docker Compose - Running multiple services

write all commands in a structured way.

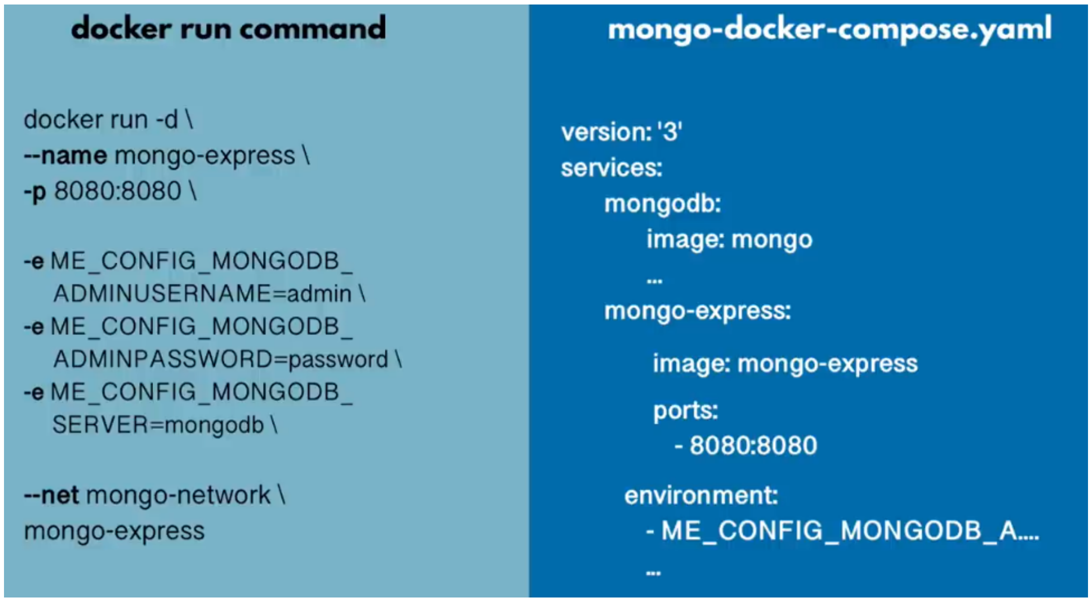


docker run command

```
docker run -d \

--name mongodb \

-p 27017:27017 \

-e MONGO-INITDB_ROOT_USERNAME
=admin \

-e MONGO-INITDB_ROOT_PASSWORD
=password \

--net mongo-network \

mongo
```

mongo-docker-compose.yaml

```
version: '3'
services:
    mongodb:
      image: mongo
      ports:
        - 27017:27017
      environment:
        - MONGO..._USERNAME=admin
```



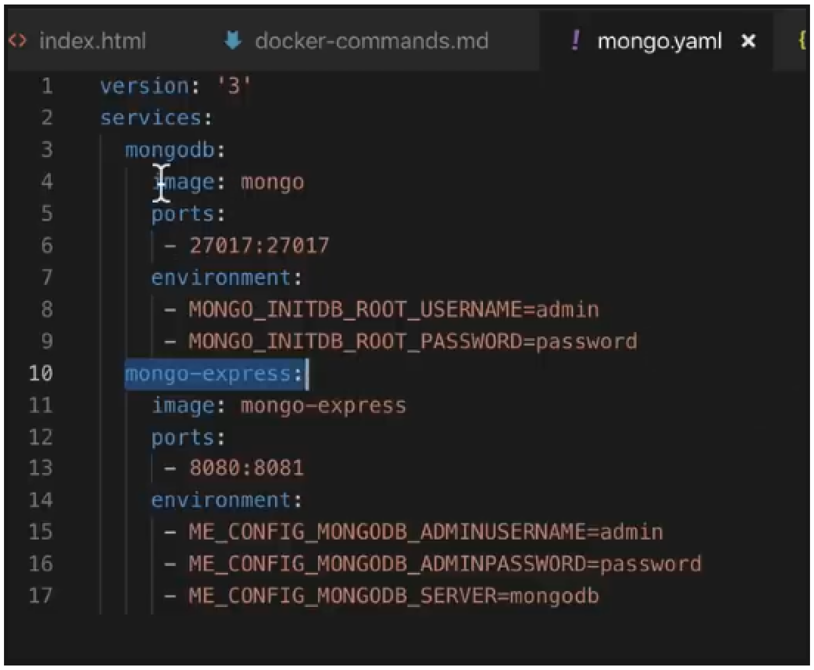
docker run command

```
docker run -d \
--name mongo-express \
-p 8080:8080 \

-e ME_CONFIG_MONGODB_
   ADMINUSERNAME=admin \
-e ME_CONFIG_MONGODB_
   ADMINPASSWORD=password \
-e ME_CONFIG_MONGODB_
   SERVER=mongodb \

--net mongo-network \
mongo-express
```

mongo-docker-compose.yaml

```
version: '3'
services:
    mongodb:
        image: mongo
        ...
    mongo-express:
        image: mongo-express
        ports:
          - 8080:8080
      environment:
        - ME_CONFIG_MONGODB_A....
        ...
```

Docker compose takes care of creating a common network,, no need to mention it.



```yaml
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
```

docker-compose -f mongo.yml up //-f for file
//open mongo create database and table for users then update and check inputs on db ;)

docker-compose -f mongo.yml down //to stop containers and remove network

🚀 10. Dockerfile - Building our own Docker Image
Build docker image for our js application

Environmental variable was already done in docker compose ,which was better so that if something changes instead of rebuilding docker file and rebuilding image.

FROM (image you want)
ENV (To set environmental variables)
RUN (execute any linux command) inside of container
COPY (executes on host machine to copy from host) //if i used it in run it reach container only
CMD () //executes entry point linux command

Difference between cmd and run in docker file ?

In a Dockerfile, both the `CMD` and `RUN` instructions are used to execute commands, but they serve different purposes and are used at different times during the Docker image build and container run processes.

Here's the key difference between `CMD` and `RUN` in a Dockerfile:

1. **`RUN` Instruction:**
   - The `RUN` instruction is used during the image build process.
   - It is responsible for executing commands within the Docker image's file system.
   - The commands specified with `RUN` are executed at the time the Docker image is being created.
   - `RUN` commands are typically used to install software, update packages, configure the environment, and perform other tasks needed to set up the image.

   Example:
   ```Dockerfile
   FROM ubuntu:latest
   RUN apt-get update && apt-get install -y nginx
   ```

2. **`CMD` Instruction:**
   - The `CMD` instruction is used to specify the default command that should be executed when a container is started from the image.
   - It is not executed during the image build process but rather when a container is launched from the image.
   - You can only have one `CMD` instruction in a Dockerfile, and it specifies the primary command that will run when the container starts. If you have multiple `CMD` instructions, only the last one will take effect.

   Example:
   ```Dockerfile
   FROM ubuntu:latest
   CMD ["nginx", "-g", "daemon off;"]
   ```

   In this example, when a container is started from the image, it will automatically run the `nginx` command with the specified arguments.

In summary, `RUN` is used for image build-time operations, while `CMD` is used to define the default command to run when a container is started. You can combine both instructions in a Dockerfile to create an image that has the required software installed and specify the command to execute when the container runs.

difference between add and copy in docker file ?

n a Dockerfile, both the `ADD` and `COPY` instructions are used to copy files and directories into a Docker image, but they have some differences in terms of functionality and behavior:

1. **COPY Instruction:**
   - The `COPY` instruction is a straightforward way to copy files or directories from the host machine into the Docker image.
   - It copies files or directories from the host machine's file system directly to the specified destination in the image.
   - `COPY` is preferred when you want to copy local files or directories into the image, especially if you are copying files from the host's file system.
   - `COPY` has a simple syntax: `COPY <src> <dest>`, where `<src>` is the source path on the host, and `<dest>` is the destination path in the image.
2. Example:
   Dockerfile

```
FROM ubuntu:latest
COPY app.py /app/
```

ADD Instruction:

- The `ADD` instruction is more powerful and versatile than `COPY`.
- In addition to copying local files and directories into the image, `ADD` also supports the extraction of tar files and remote URL retrieval.
- When using `ADD` with a URL as the source, Docker will automatically download the file from the URL and copy it into the image.
- `ADD` can be useful for scenarios where you need to download files from the internet or extract compressed archives within the Docker image.
- However, because of its added complexity, it's recommended to use `COPY` for simple file copying operations to improve transparency and predictability.

Example:
Dockerfile

```
FROM ubuntu:latest
ADD https://example.com/myapp.tar.gz /app/
```

1.

In general, it's advisable to use the `COPY` instruction when your intention is simply to copy local files or directories into the Docker image. `ADD` should be used when you need additional features like automatic downloading of resources or extracting compressed files within the image. However, be cautious when using `ADD` with URLs, as it can introduce unexpected behavior due to URL caching, and it may not always work as expected in certain environments.

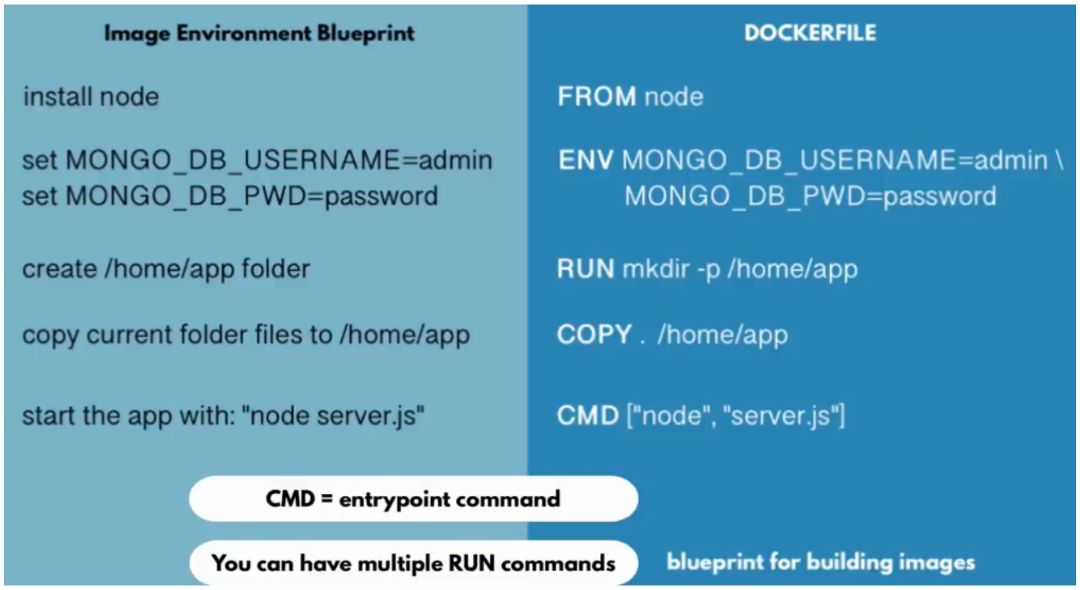


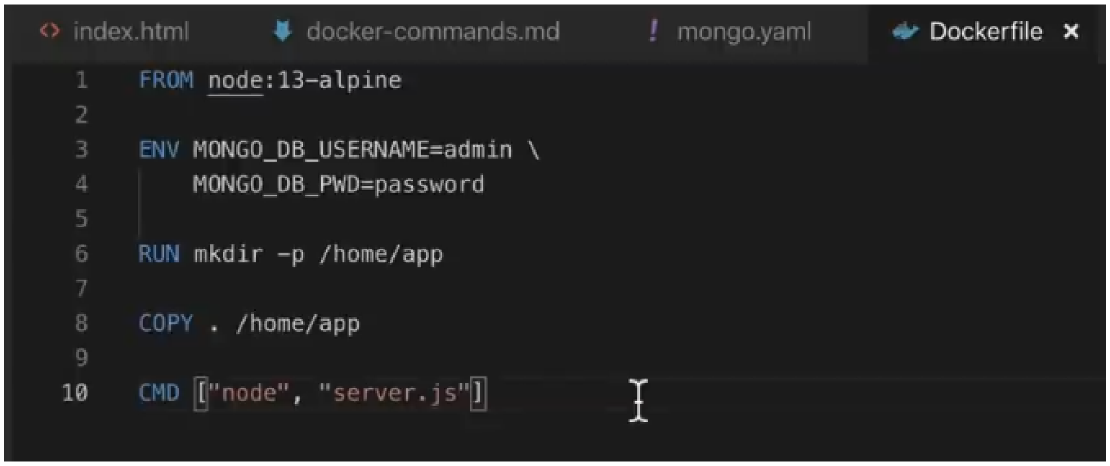
Image Environment Blueprint	DOCKERFILE
install node	FROM node
set MONGO_DB_USERNAME=admin set MONGO_DB_PWD=password	ENV MONGO_DB_USERNAME=admin \\ MONGO_DB_PWD=password
create /home/app folder	RUN mkdir -p /home/app
copy current folder files to /home/app	COPY . /home/app
start the app with: "node server.js"	CMD ["node", "server.js"]

CMD = entrypoint command

You can have multiple RUN commands

blueprint for building images

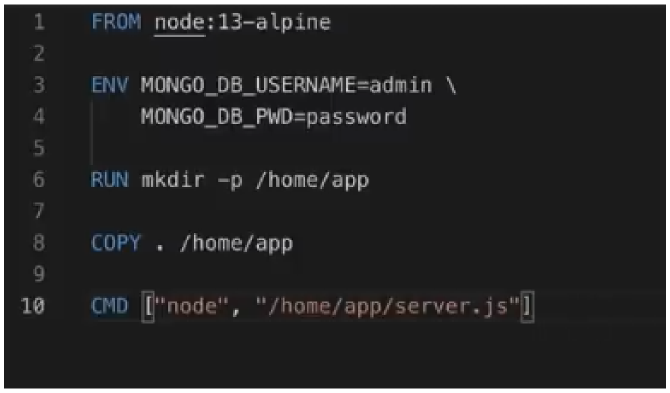
name of file must be "Dockerfile"



```
FROM node:13-alpine

ENV MONGO_DB_USERNAME=admin \
    MONGO_DB_PWD=password

RUN mkdir -p /home/app

COPY . /home/app

CMD ["node", "server.js"]
```

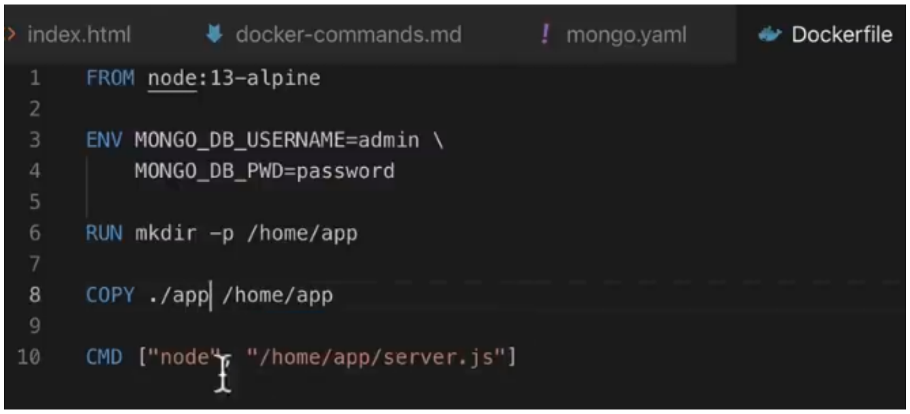
docker build -t my-app:1.0 .   //-t for tag , . as we are in same directory
docker images   //you will find the image
docker run my-app:1.0

error: can not find server.js .. because the directory stopped before CMD on copy inside app itself.
sol:



```
FROM node:13-alpine

ENV MONGO_DB_USERNAME=admin \
    MONGO_DB_PWD=password

RUN mkdir -p /home/app

COPY . /home/app

CMD ["node", "/home/app/server.js"]
```

docker build .
docker run my-app:1.0
docker exec -it my-app:1.0 bin/bash //to check it 🙂
The above did not work so we will try
docker exec -it my-app:1.0 bin/sh //to check it 🙂
ls
env //you will see username and password
ls /home/app //the files for node are here :)
we actually does not need to have dockerfile and dockercompose ,we only need js files.lets improve that…
make app directory "app" in my pc without these files… and update my dockerfile to be…



```
FROM node:13-alpine

ENV MONGO_DB_USERNAME=admin \
    MONGO_DB_PWD=password

RUN mkdir -p /home/app

COPY ./app /home/app

CMD ["node", "/home/app/server.js"]
```

then run again….

🚀 11. Private Docker Repository - Pushing our built Docker Image into a private Registry on AWS

Amazon ECR , NEXUS, Digital oceane are private repos..
open aws.amazon.com create your account
(Elastic container registry)
create repository , name it… create….
For ECR it is for each it has its own repository , you can not store more than one , you can add different versions of it , but not from different images.

on ECR click on your repo and then view push commands
Prerequisites for those commands:

1) AWS CLI needs to be installed
2) Credentials configured

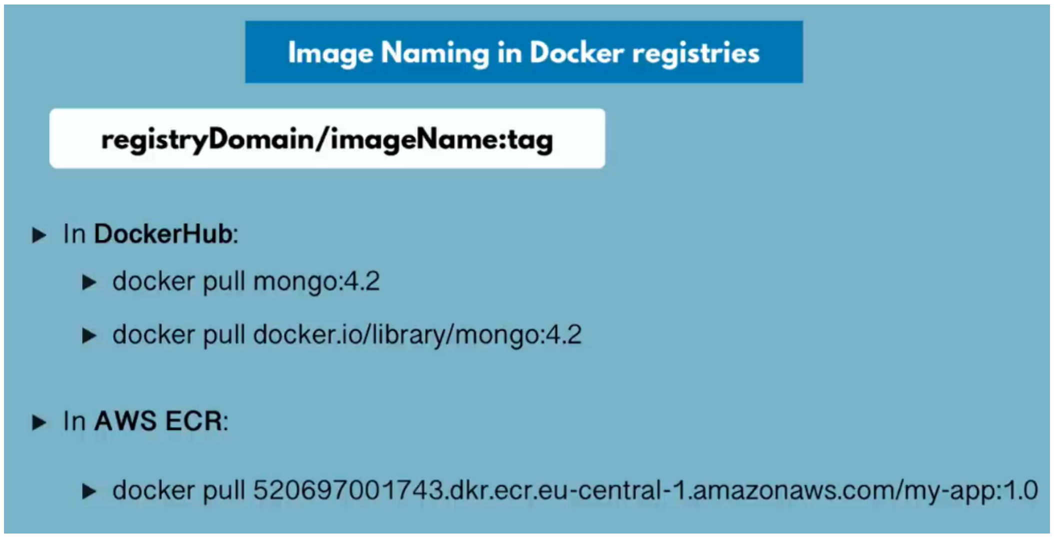


Image Naming in Docker registries

registryDomain/imageName:tag

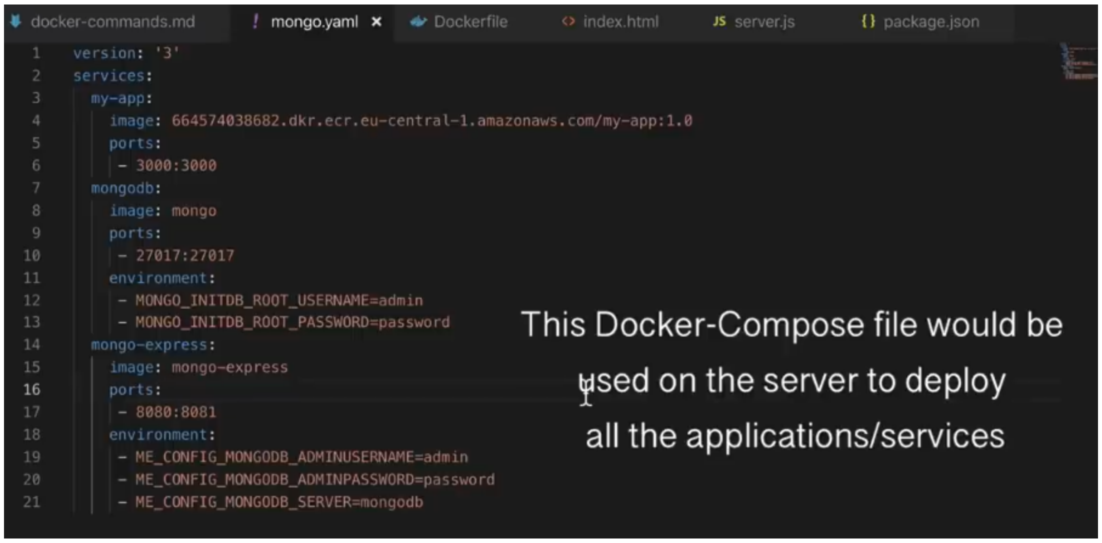
- In **DockerHub**:
  - docker pull mongo:4.2
  - docker pull docker.io/library/mongo:4.2
- In **AWS ECR**:
  - docker pull 520697001743.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0

made some changes to code and dockerfile
docker build -t my-app:1.1 .
push again to same repository on ECR..
Now you have 1.0 and 1.1 ,, you can add up to 1000 versions on same ECR

🚀 12. Deploy our containerized application

We want to run our image js and mongo and mongo express. 2 will be pulled from docker hub and one from ECR.

we will update our docker compose file to have the private repo:
Development server must be logged in to AWS…



```yaml
version: '3'
services:
  my-app:
    image: 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
    ports:
      - 3000:3000
  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
```

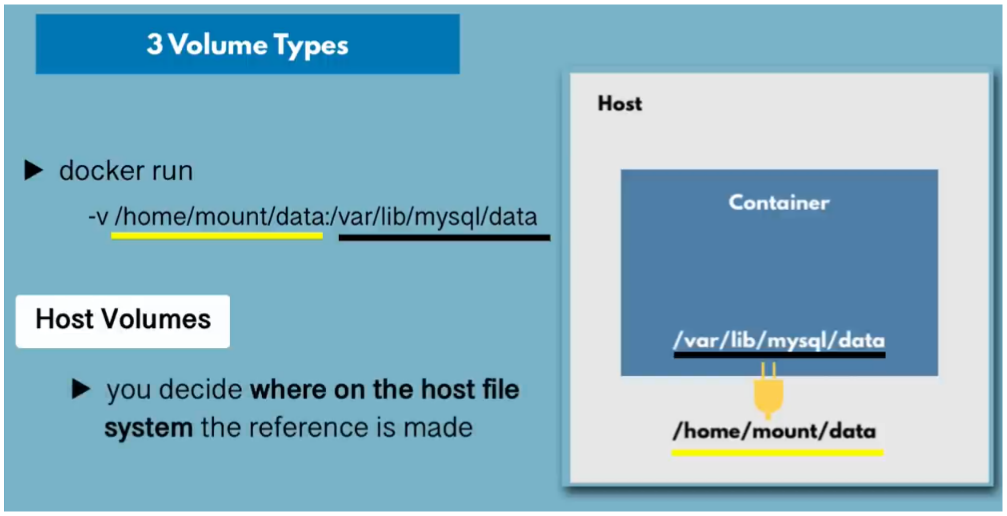
This Docker-Compose file would be used on the server to deploy all the applications/services

on terminal:
docker login with the command from AWS.
vim mongo.yml //copy and paste the code to development server
docker-compose -f mongo.yml up
app is up 🙂

🚀 13. Docker Volumes - Persist data in Docker

Used for data persistence as for database or other stateful apps.
When is it needed ? container runs on host and has database .where data is saved in virtual file system, when it is restarted everything is gone.
Solution: plug physical file system on host is mounted in virtual file system on container. So they sync on each data inserted,deleted ,etc…

3 volume types:
1)Host Volumes:


3 Volume Types

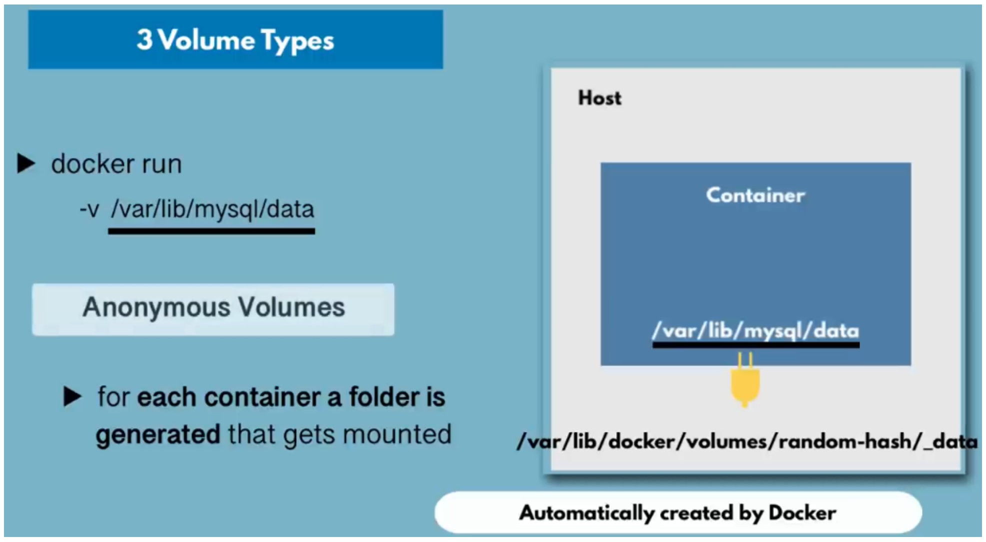
▶ docker run

```
-v /home/mount/data:/var/lib/mysql/data
```

Host Volumes

▶ you decide **where on the host file system** the reference is made

2)Anonymous volumes: you do not choose folder on host:


3 Volume Types

▶ docker run

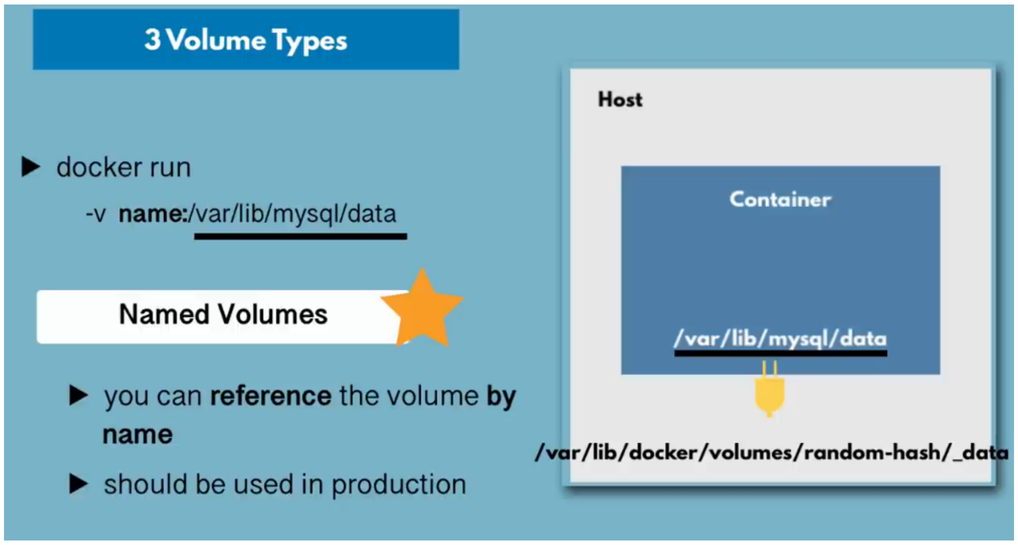
```
-v /var/lib/mysql/data
```

Anonymous Volumes

▶ for **each container a folder is generated** that gets mounted

Host
Container
/var/lib/mysql/data
/var/lib/docker/volumes/random-hash/_data
Automatically created by Docker

3) Named Volumes: similar to anonymous but lets you specify the name of folder on host.



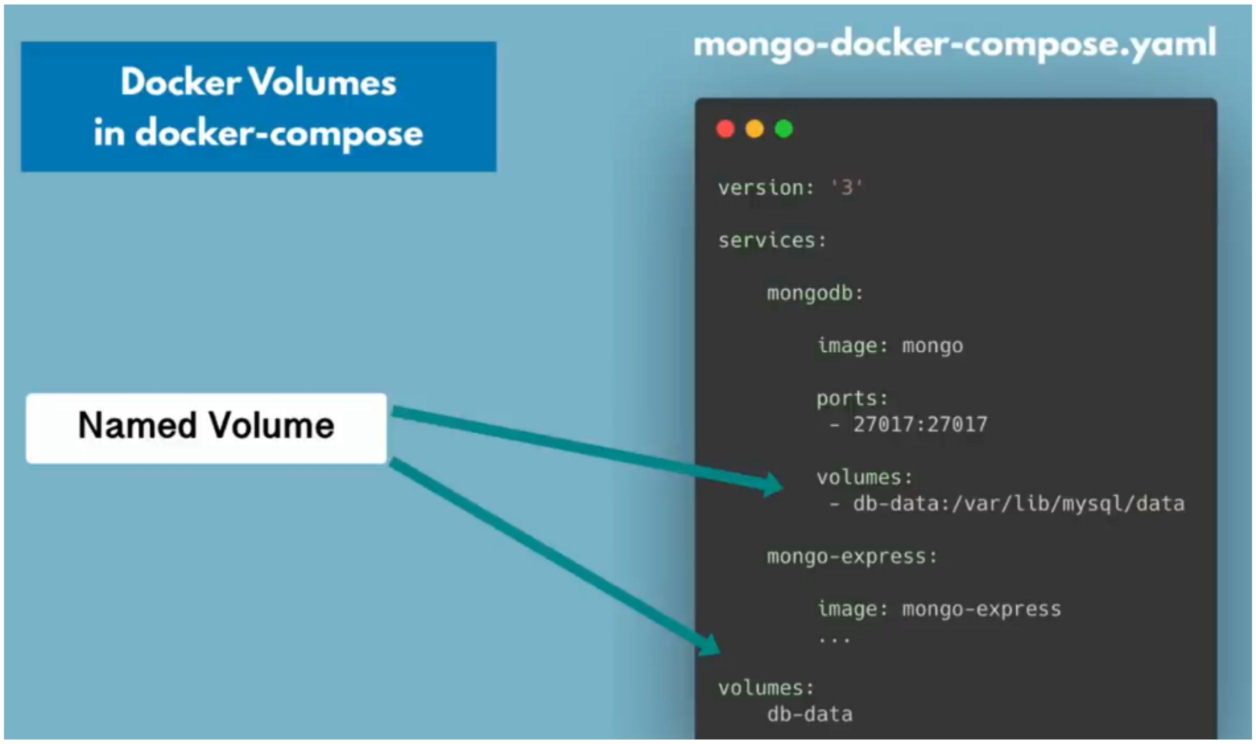
3 Volume Types

- docker run
  -v **name:**/var/lib/mysql/data

Named Volumes

- you can **reference** the volume **by name**
- should be used in production

Most used is named volumes:
In terminal used as above, while in docker compose is as below:



Docker Volumes in docker-compose

Named Volume

mongo-docker-compose.yaml

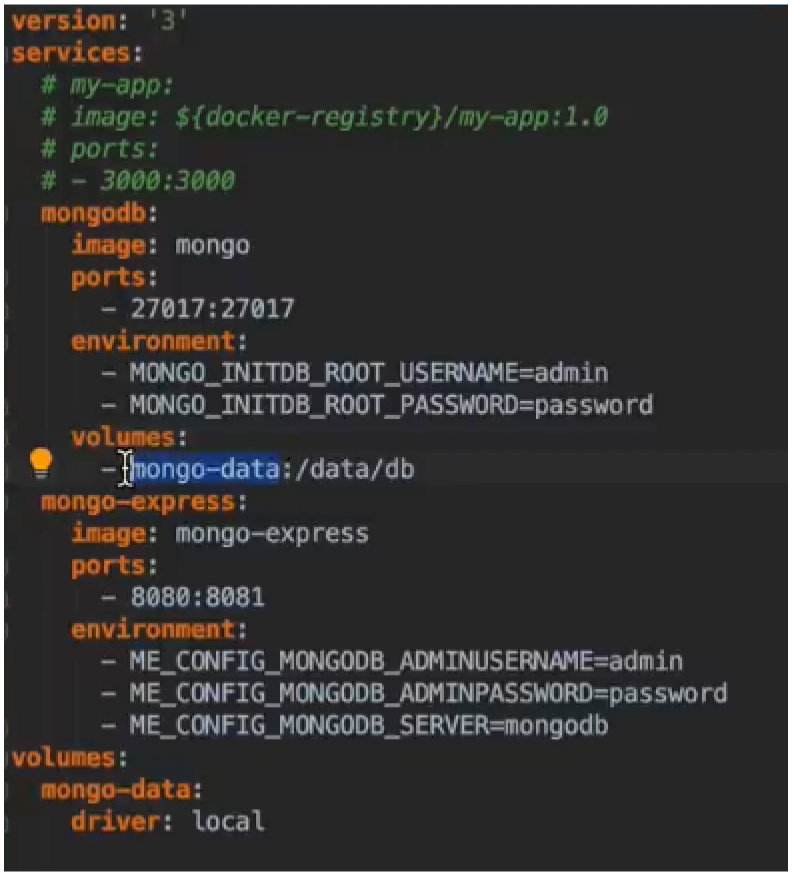
```yaml
version: '3'
services:
    mongodb:
        image: mongo
        ports:
         - 27017:27017
        volumes:
         - db-data:/var/lib/mysql/data
    mongo-express:
        image: mongo-express
        ...
volumes:
    db-data
```

In above db-data:/var/lib/mysql/data   is named volume same as in terminal.
But you must come at the end in services and create a volume that have all named volumes used in file..and by this way it can be shared between different containers.

🚀 14. Volumes Demo - Configure persistence for our demo project

Using same project used above…but if restarted as in previous way all data will be lost.

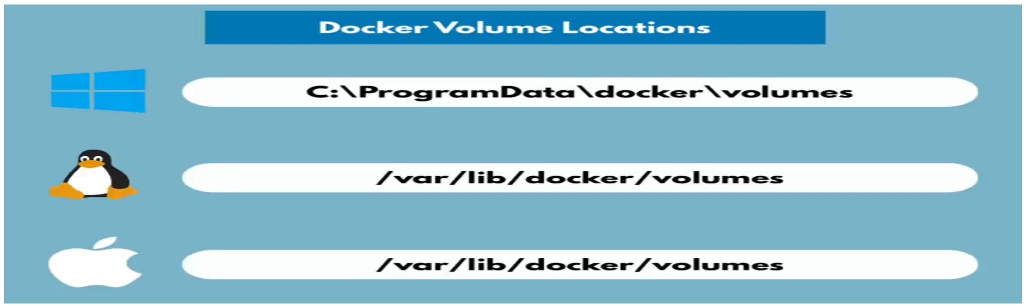
Docker compose updated.



```yaml
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
```

mongo-data:/data/db it has to be path in container where mongo db stores its data, you can search it “mongo data path”from internet on google for each app. we searched and it was /data/db

driver:local //to inform it make sure it is local drive



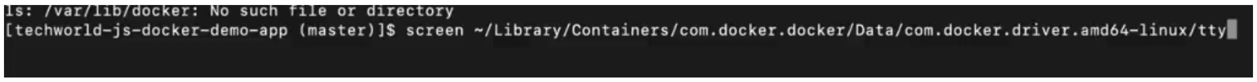
Docker Volume Locations

C:\ProgramData\docker\volumes

/var/lib/docker/volumes

/var/lib/docker/volumes

They are stored on host in a virtual machine created by docker.
To login to that VM, you use:



```
ls: /var/lib/docker: No such file or directory
[techworld-js-docker-demo-app (master)]$ screen ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/tty
```

Now you will be logged in for that vm.

now you can use above paths: var/lib/docker   //to see list of volumes on host
//In this list some of them will have names which are defined by us, and other who have random numbers and names which are anonymous…

to exit out of this vm
ctrl+a+k   //then yes

THE END 😉
000000000000000000000000000000000000000000000000000000000000000000000000000