**DOCKER:**

Container is a way to package the application with all the necessary dependencies and configurations.

The package is portable which can be moveable like any other artefact. it can be shared and moved around between the development team and the evolution team.

* The container needs some storage to save and move around so they are started in a container repository that is special for containers.
* Many companies have their own private containers
* There is a public repository for the docker container: [**https://hub.docker.com/**](https://hub.docker.com/)
* Before Docker, we have installation progress for each OS environment. Many steps can go wrong because the rest of the team members should install services.
* With containers, because it has its own isolated environment, we have everything packaged in one isolated environment, we do not need to install any of the services directly in our operating systems, containers provide us with that.
* Packaged with a configuration and the start script inside of one container so developers need to go and look at the specific container and download with just one command.
* We can have different versions of the same application running without conflict in docker.
* With docker instead of the traditional way(development team sends the instruction to the operation team and then deploy the server) development and operations working in one team package the application in a container.
* No environmental configuration is needed on the server (except Docker runtime).
* The only thing we need to do is run a docker command that uploads that container image we stored somewhere in the repository and run it.

**Container :**

Layers of images

Mostly Linux base Image, because small size

Application image top

When we restart the container everything that we configured in that container's application is gone like data. For data loss there is a concept called volumes that makes it possible to have persistence between container restarts.

**Image and container**

Image is the actual package, application pack together with all dependencies and configurations, an artefact that can be moved around

the container is when I pull that image my machine and i actually start and application actually starts and that creates the container environment

**Container port and Host port**

Multiple containers can run on our host machine

Laptops only have certain ports available

Conflict will occur if we open 2same port on host machine

Different ports can listen same container as long as we bind them different host on machine

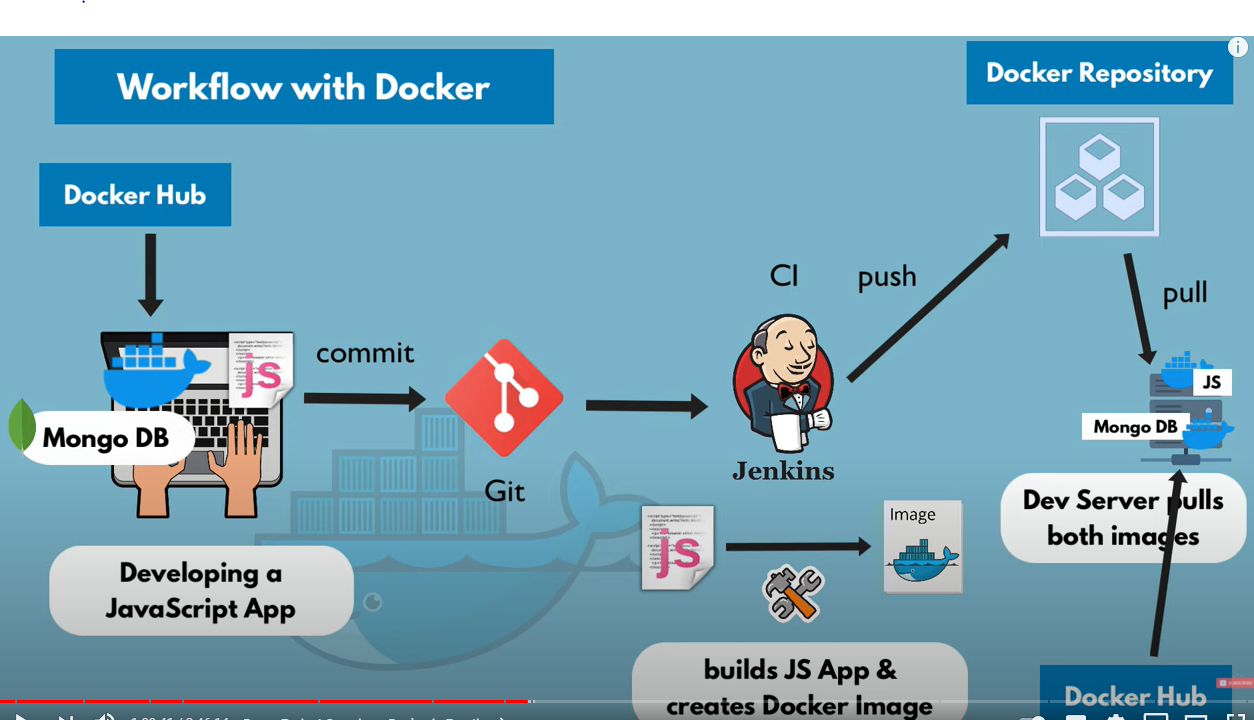
* docker run -p<port of the 1.host(we specify like 6000):same container port> for 1.image

docker run -p<port of the 2.host(we specify like 6001):same container port> for 2.image

Start the container using the binding between host and the container ports, if we want to run the same images but different versions we need to bind them with 2 different ports on the machine(host) listening to the same container ports.

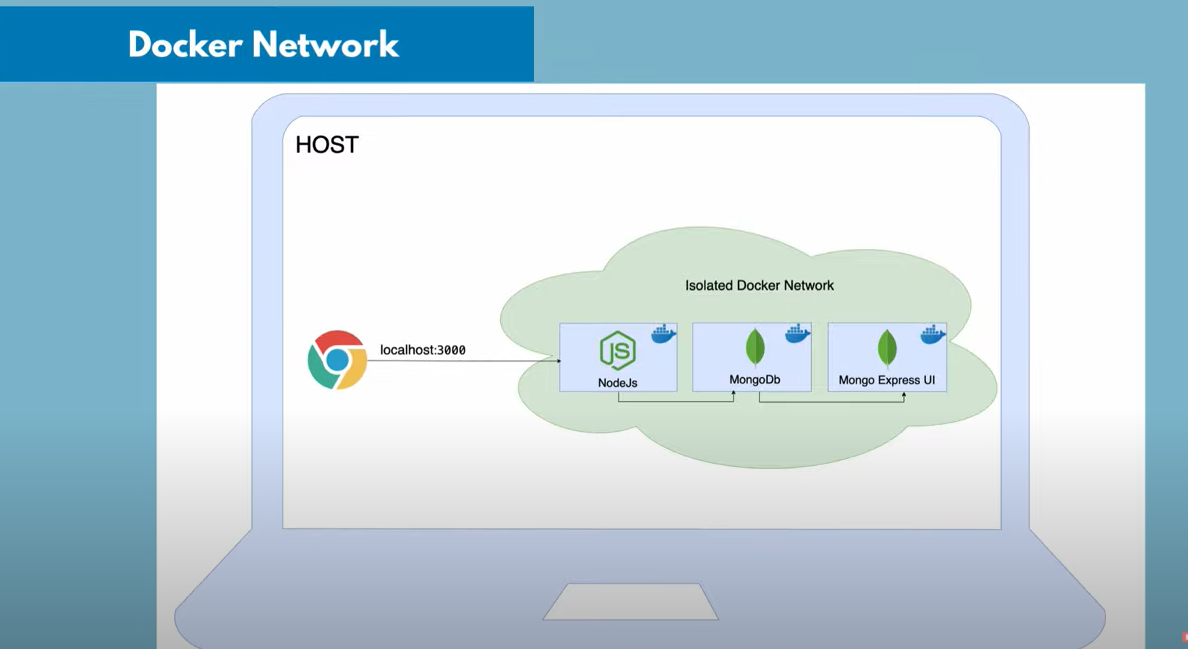
**Workflow with Docker**

Javascript with mongoDB from docker container(example app)



If developer or test server login the dev server they are all able to test the application.

**Docker Network**

Docker creates an isolated docker network where the containers run in it. So when deployed containers in the same docker network in case mongo and mongo express they can connect to each other just using the container name without localhost or port number an application which runs out of docker which just runs from a node server is goingt to connect to them from outside using localhost and port.

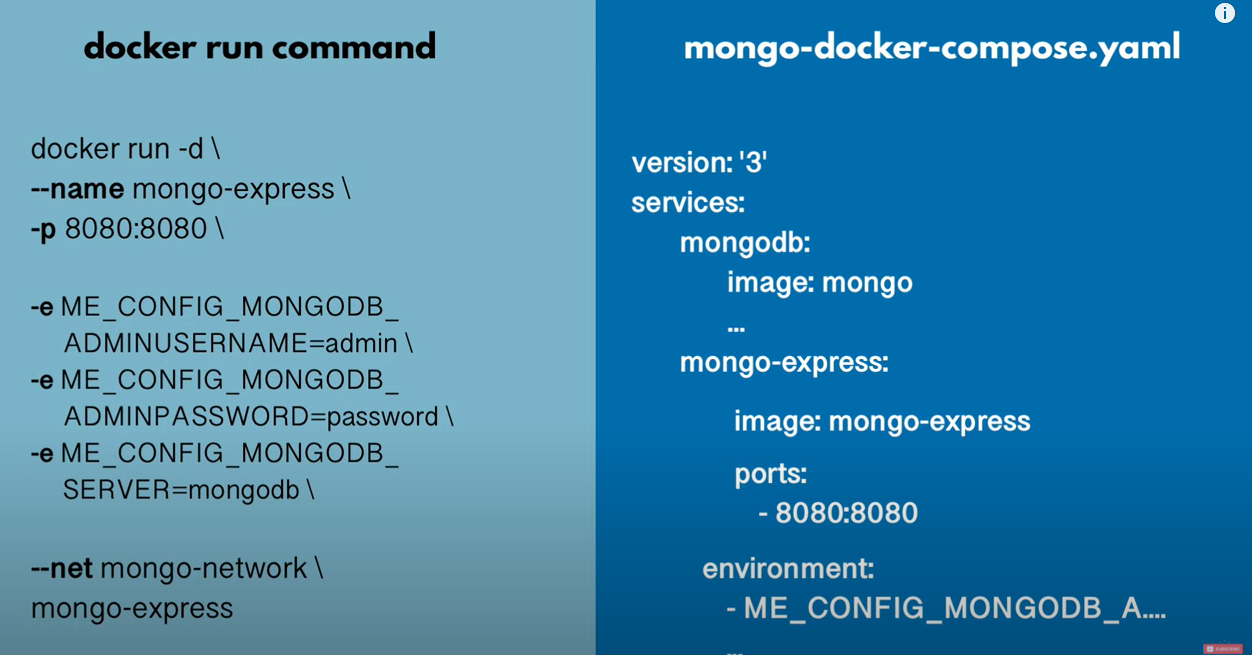
**Docker Compose**

Tool that runs multiple docker containers all this(instead run each time docker run command) configuration much easier than with docker run commands.

With docker compose we can write all the commands in one file and just run that file instead of running docker run command

Docker Compose takes care of creating a common Network.

For example:



**Dockerfile(push the docker repository)**

How to build docker images from an application, basically having to copy the content of that application into a docker file could be an artifact that we built in our case. The application we are using(example one) has 3 files so we will copy all files directly into the images. We will do this using a blueprint for building images which is called a docker file.

Docker file is a blueprint for creating docker images.

RUN(in docker file) == execute any linux command

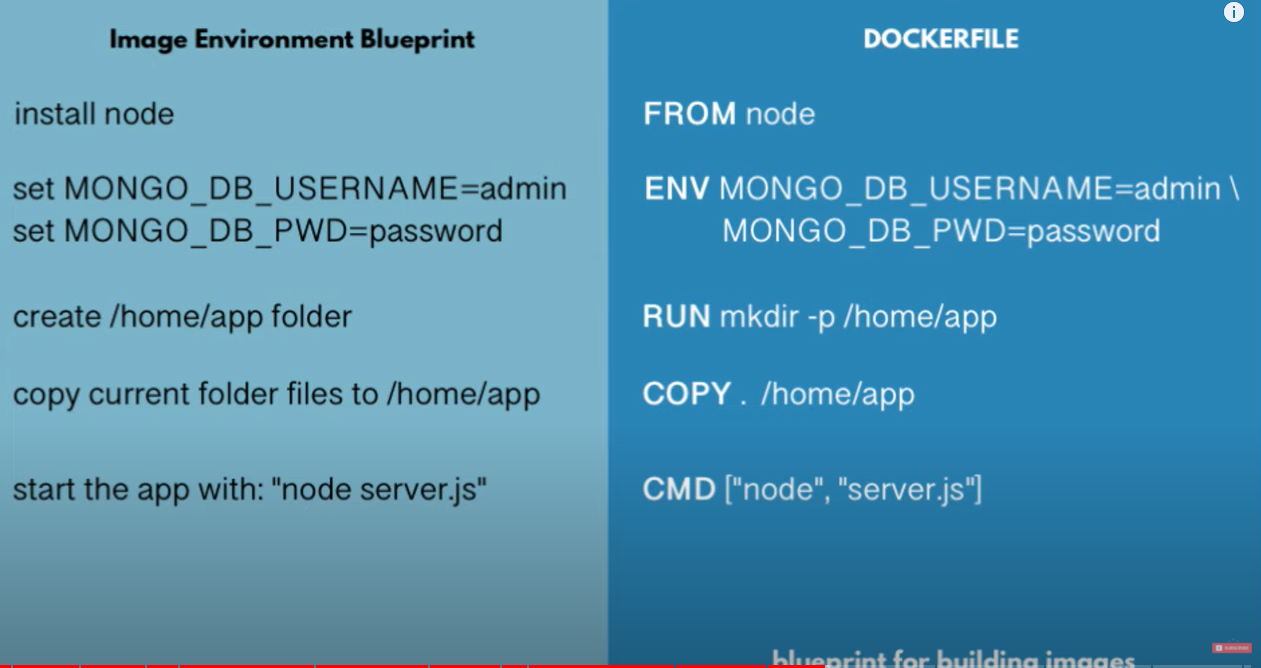
inside the docker file command mkdir command makes a directory that lives inside of the container. when we start the container directory just created will be created inside of the container not on the computer or host.

COPY() == executes on the HOST machine , i can copy from on my host to insine of that container image.because if i were to execute -run sp - that command would execute inside of the Docker container.

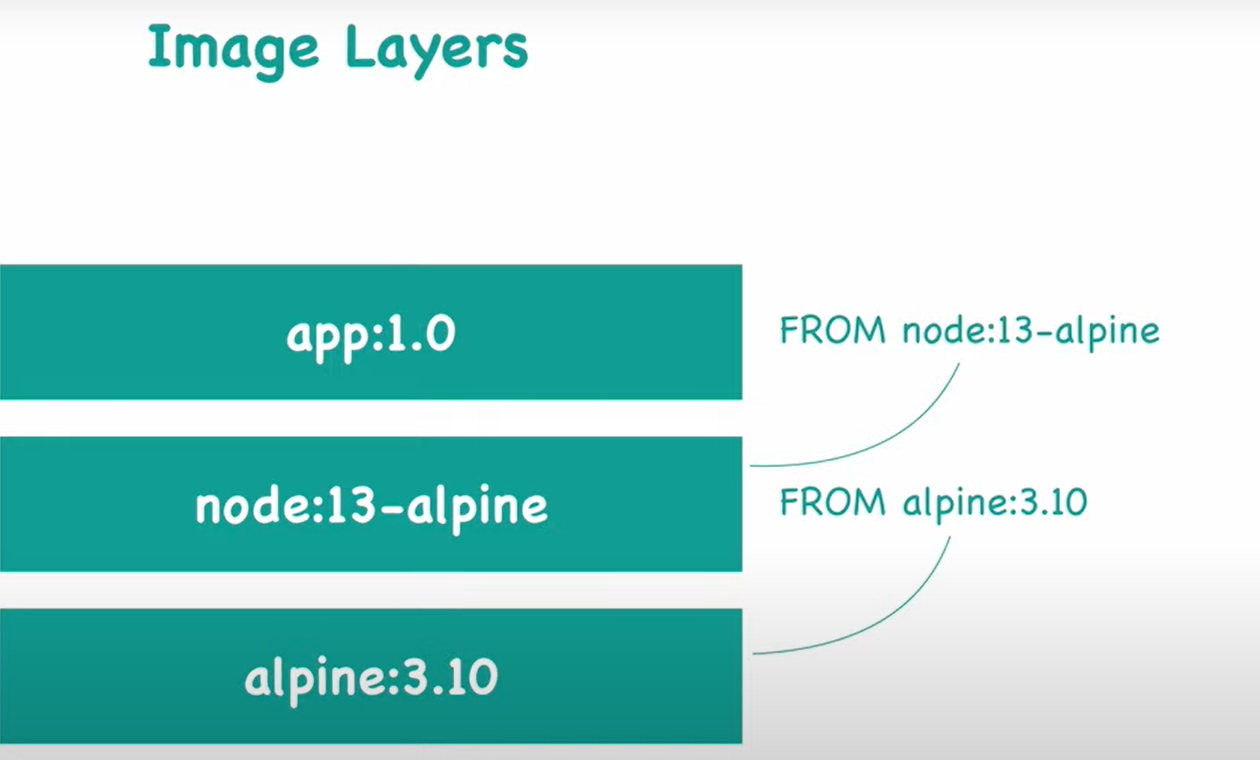
CMD == executes an entry point LUnix command.

What differences RUN and CMD : cmd entry level command, we can have multiple RUN commands.

we can't give any name for docker file it have to called “Dockerfile”



**Image Layers based on another image**

****

In order to build an image with a docker file we have to provide 2 parametrer.

* give image name in a tag
* parameter of location of docker file

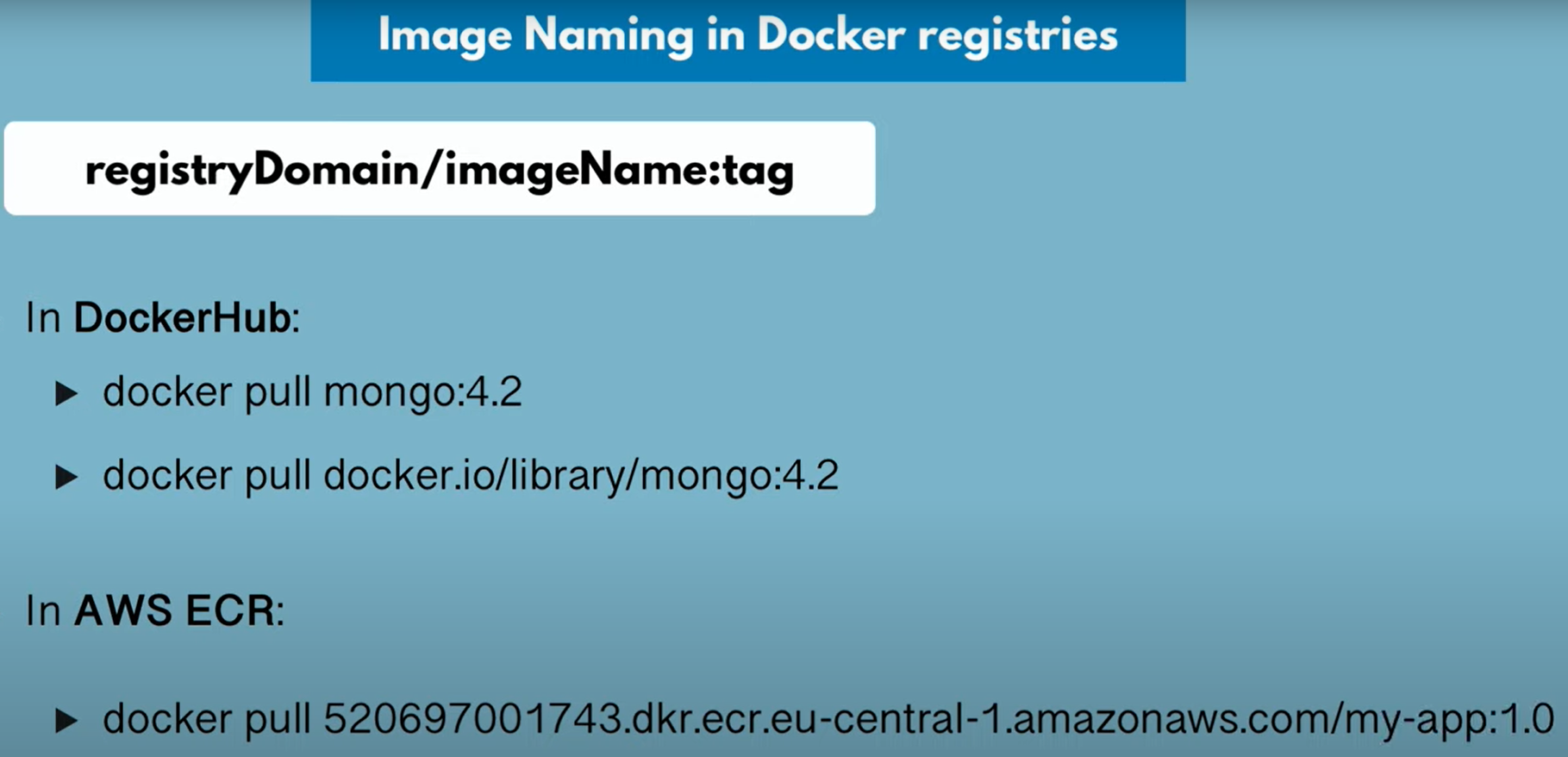
example command: docker build -t <app name:tag>my-app:1.0 <docker file location> . (“ . ” used this case because we are same folder of file)

**Private Docker Repository(AWS ecr)**

* Amazon container service docker repository per image
* We save different tags(versions) of the same image.
* View push commands for my app
* We always have to log in to a private repo from our local machine! = docker login
* Docker login 1. step and done ones and pull and push many time
* Prerequisites:

-aws cli needs to be installed

-credential configured



**Deploy containerized App**

After we package our docker application in a docker image and save it a private repo we need to deploy our dev or integration server with docker compose.

*docker-compose.yaml :*

version: '3'

services:

#for deployment part will be using all the applications/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

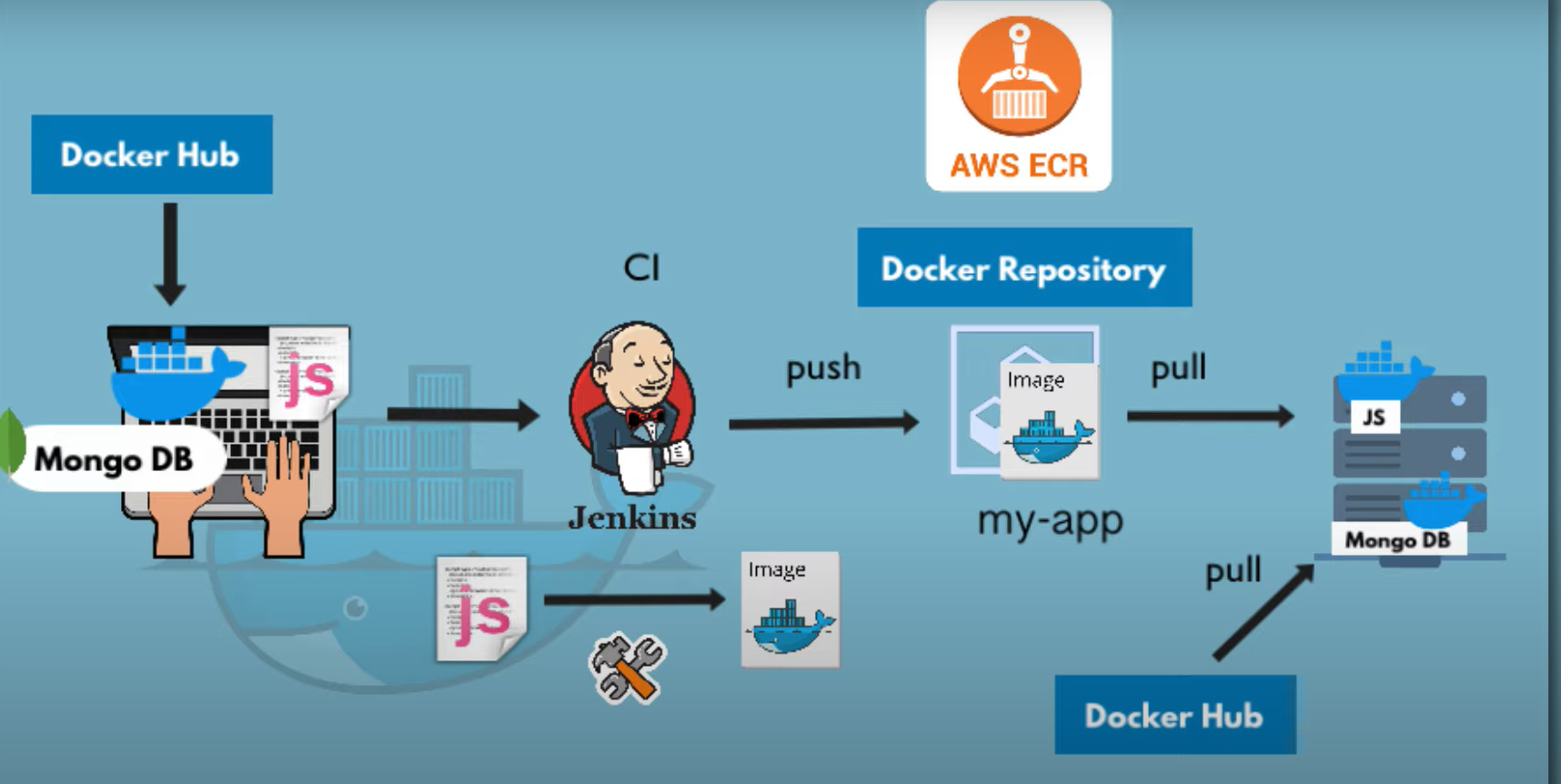
volumes:

mongo-data:

driver: local

-------------

PS:: every time the database will be lost when we recreate the container. we need to preserve the database data when the container restarts to use docker volumes for ideal state.

ALL STEPS

**Docker Volumes**

Nutshell docker volumes are used for data persistence.

How do we use docker volumes?:

In a host we have a physical file system and in a container we have a virtual file system. with docker volume, folder in the physical file system is **mounted** into the virtual file system of docker. Data gets automatically replicated.

**3 diff type docker volume**

* Host volume: using “docker run -v <folder you chose>/home/mount/data:/var/lib/mysql/data” command, we decide where on the host file system the reference is made
* Anonymous Volumes: using ”docker run -v/var/lib/mysql/data” for each container folder is generated that gets mounted. which directory mounted automatically created by docker
* Named Volumes: ”docker run -v name(up to you):/var/lib/mysql/data '', you can reference the volume using by name. mostly using one for production should be named volume.

you can use docker volumes in docker compose with specify

“volumes:

* db-data(ref name of volume):/var/lib?mysql/data”

then same level of services you need to define all volume you have

“ volumes:

db-data”

This is very useful if the containers share the data.

PS: each database has its own path and in

windows C:\ProgramData\docker\volume

linux and ios ->>/var/lib/docker/volumes

**Basic Commands**

* docker run <image name>:<version> = pulls and runs images in one command, it will start image in a container
* docker run = creates new container and starts with a command
* docker images = shows all the images that you have locally
* docker ps = shows running containers status
* docker run -d = will get id of container and runs the container detach mode
* docker stop <container ID>
* docker start <container ID> = restart the existed container
* docker ps -a = lists all container running or not
* docker - p = allows you to bind port of your host the container
* docker logs <container ID>/<container name> = see the all the log (ex..something happens your app and you cannot connect with redis you do this see what redi 's container presiding) --Trouble Shooter command--
* docker run -d -p <port of the 1.host(you specify like 6000):same container port> --name <type the container name> <image name> == ex..docker run -d -p6000:6379 --name radis-older redis:4.0
* docker exec -it(stand for Interactive Terminal) <container ID> /bin/bash = get the terminal of the running container exit to exit from the terminal, Interactive Terminal linux limited but it should be enough for debugging
* docker logs <container ID> | tail
* docker logs <container ID> -f == stream teh logs
* docker network create mongo-network == Create docker network
* docker run -d -p 27017:27017 -e MONGO\_INITDB\_ROOT\_USERNAME=admin -e MONGO\_INITDB\_ROOT\_PASSWORD=password --name mongodb --net mongo-network mongo == start mongodb
* docker run -d -p 8081:8081 -e ME\_CONFIG\_MONGODB\_ADMINUSERNAME=admin -e ME\_CONFIG\_MONGODB\_ADMINPASSWORD=password --net mongo-network --name mongo-express -e ME\_CONFIG\_MONGODB\_SERVER=mongodb mongo-express == start mongo-express
* <http://localhost:8081> == open mongo-express from browser
* docker-compose -f <file name>.yaml up == runs the compose file
* docker-compose -f <file name>.yaml down == go through all containers and shut them down (removing the container removes the networks next time restart it it will create again)
* docker tag = rename image
* for deploying
  + make sure you re login
  + change the yaml file(shown on the top)
  + have a docker compose file available that development server with creating mango.yaml file and copy pate the changed .yaml file in current directory:“vim mango.yaml”
  + run all 3 containers docker compose command :docker-compose -f mangoyaml up
* ctrl +a+k then type y close the section on mac