**AmigoMongoSpringBoot**

* Docker pull <image>
* Docker images: get all images existing
* Docker run <image-name>
* Docker run <image>:version
* Docker run -d <image> : detach mode => return id of container
* Docker run –name <name of container> <image>
* Docker run –itd <image name> :

-itd: -i allow keyboard input, -t open a terminal for bash, -d run detach mode => if a image run bash in its entrypoint => must open a terminal for bash, otherwise imediately stop the container

* Docker ps : status of running image, id of running container
* Docker ps -a : show all running or not run images
* Docker stop <id of the container>
* Docker start <id of container> => start existing container
* Docker run -p<port host>:<port container> <image> => create new container
* Docker logs <container\_id>: check log of container running or not running
* docker exec -it <container-id> bin/bash : Go in side of docker container:
* Ls: get all folder inside container
* Env: get env params
* Exit: get out of container

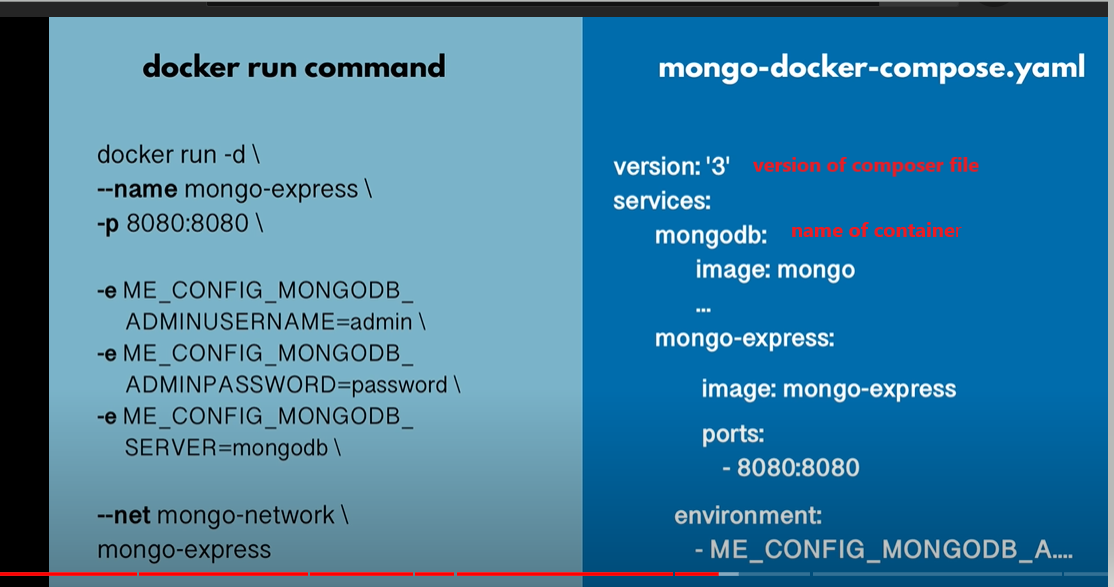
Docker compose:

db.createUser({user: "userFirstDB", pwd: "password", roles : [{role: "readWrite", db: "admin"}, [{role: "dbOwner", db: " admin "}]});

db.createUser({user: "testUser1", pwd: "pwd", roles : [{role: "root", db: "admin"}]})

db.createUser({user: "testUser", pwd: "pwd", roles : [{role: "readWrite", db: "admin"}

, {role: "dbOwner", db: " admin "}]})



Cmd in intelij or normal cmd : docker-compose -f <docker compose yaml file name> up

Stop: docker-compose -f <docker compose yaml file name> down

docker-compose down => remove all containers, not just shut them down

* Docker net work:

Docker network ls: list all network

* When running a container, this container is assigned to a network, containers assigned to the same network can communicate with each other (by ID or by DNS named), different network cannot connect with each other
* Each network type has different properties:

Bridge network stands behind host network, connection from outside must go through ip of host, using port that connect to the expose port of container.

* Default bridge network or user-defined bridge network
* CMD VS ENTRYPOINT jenkinsfile:

Entrypoint is always executed, only 1 entrypoint, cannot override, can accept cmd as params

Docker build -t <name of image> <location of docker file>

Docker build -t my-app:1.0 . : dot (.) : save the image in the current location

* Remove image:

Docker rmi <imageId >

Remove container : docker rm <containerId>

* Docker network:

Docker network create <name of network>

Docker network connect <network name> <container name>

Docker container inspect <container name>

* containerized app cannot connect database:

each container run on its own network => cannot connect with each other

* create a network
* connect DB container to this network
* run app image inside the network: using internal port (not host port)
* docker run --net spring-network -e HOST=mongoDB\_named -e PORT=27017 app
* docker run --net spring-network --env-file env.txt app

env.txt : HOST=mongoDB\_named  
PORT=27017

* Volumn:

Docker container after remove will lose all data => using volumn to mab to host disc

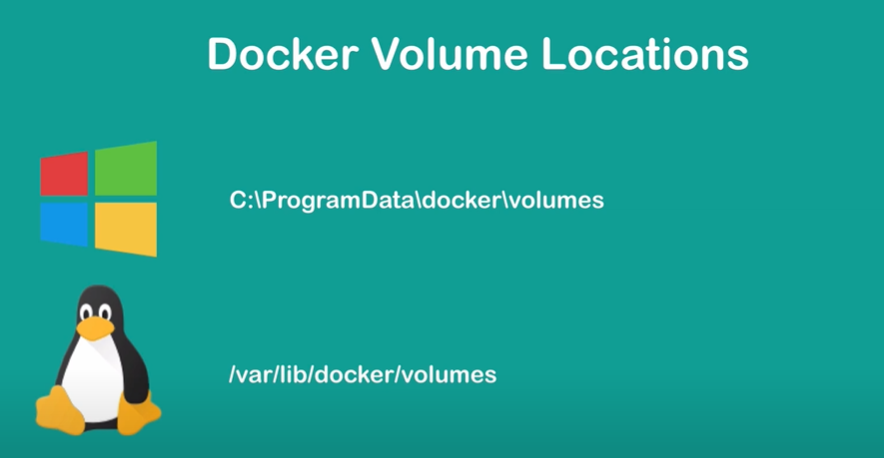
Any changes to virtual folder inside container will be replicated to host disc and vice versal

So that when run a container and mouting is virtual folder to the host disc then the new container will have all data persisted before

* docker run --net spring-network --env-file env.txt -v <host file>: /var/lib app

named volumn:

* don’t care the actual folder of host directory
* using the same name for different container => containers that share the data



Docker volumn location on host machine

* environment:

app in the image require some environment variables => when running the image must pass env through file or command