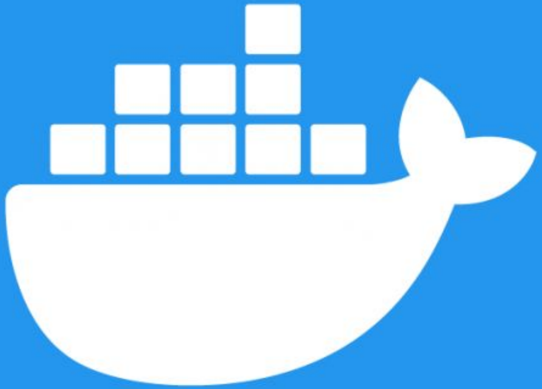


docker®

what will we learn

w3l

What will we learn?

- base concept of virtualization
- evolution of virtualization into containers
- docker containers
- docker history

virtualization

virtualization - intro -

The base concept of virtualization is the decoupling of hardware from the operating system, virtualized server are more flexible than bare metal (portable, programmatically handled)

Virtualization has been out there since 1960, when Big Blue used virtual machines on their mainframes.

The essence of virtualization is to parcel out:

- CPU
- Memory
- I/O resources

and dynamically allocate their use

Virtualization led the way to containerization.

Docker is one of the most used containerization engine

virtualization - types -

Virtualization need an hypervisor a software layer between virtual machine and hardware (i.e. QEMU)

Virtual machine do not know to be a virtual machine:

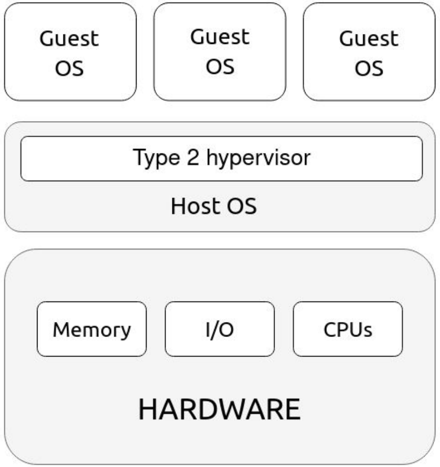
- hardware-assisted virtualization (hyper-v, amd-v)
- cpu and memory controller virtualized by hardware and controlled by hypervisor

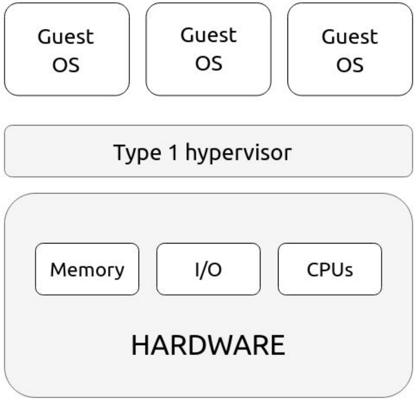
2 type of virtualization:

- hardware-hypervisor virtualization
- hardware-os-hypervisor virtualization

PROs:

- live migration
- VM images
- cli manageable





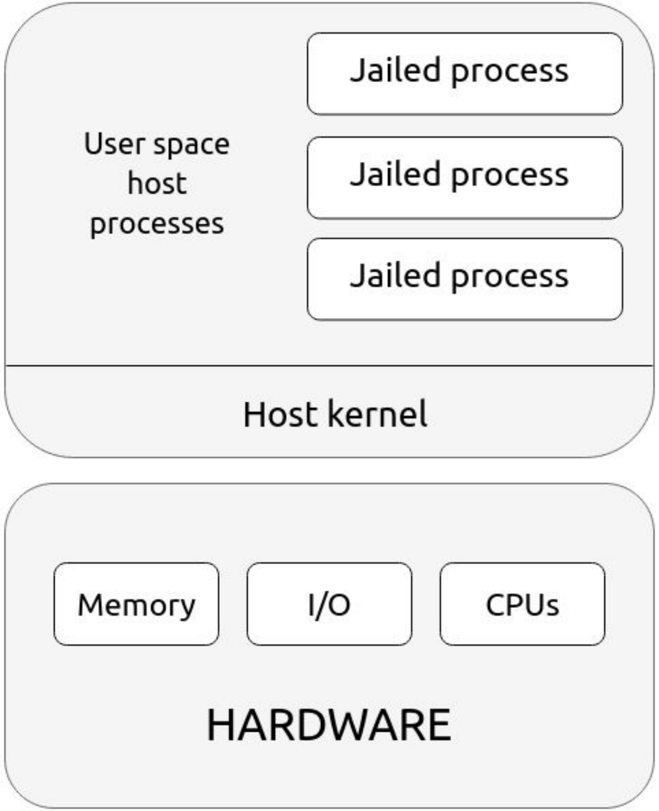
virtualization - containerization -

OS-level virtualization - or containerization - do not use hypervisor.

It relies on Kernel feature to isolate processes from the rest of the system.

Each process “container” or “jail” has a private root filesystem and root namespace.

The containered process shares kernel and other service of host OS but cannot access files or resources outside their container.



PROs:

- resource overhead low
- near native performance
- portable
- isolated execution environment

docker

docker - intro -

Most hyped technology of last years: docker, born in 2013 from dotCloud (debuted in Santa Clara PyCon), written in GO

It allows standard software packaging (a dream that long has been out of reach)

When you dispatch a web service:

- code and config
- libraries and other dependencies
- interpreter(python, ruby) or run time (JRE) to execute code
- localizations (account, env settings, service from SO)

Container is an isolated group of processes that are restricted to a private root filesystem and process namespace. Conteinered processes share kernel and other service from OS but by default cannot access files or system resource outside container.

Other containerization engine: rkt, systemd-nspawn

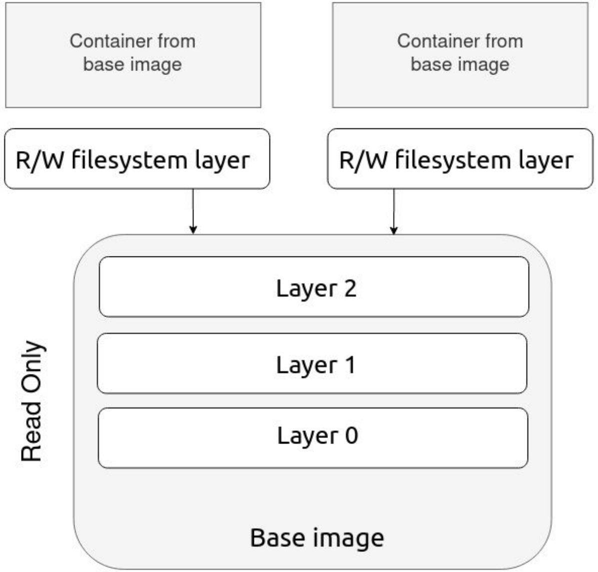
docker - hiw -

To make docker work you need:

kernel support, essential feature are:

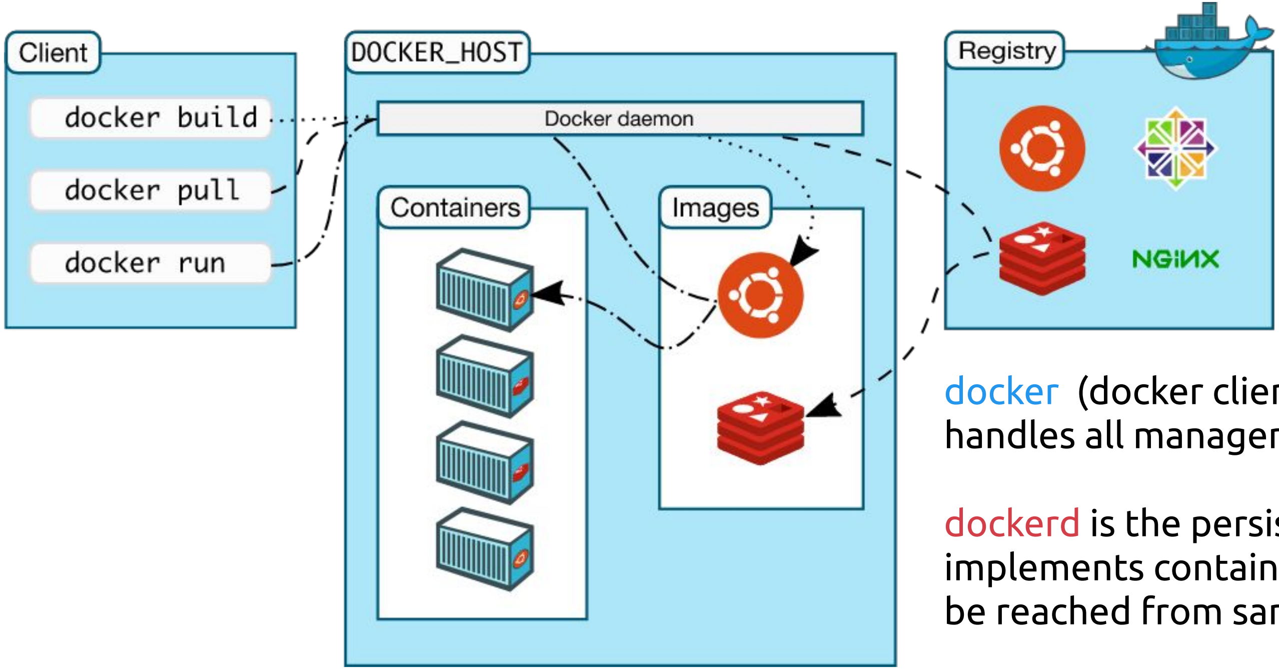
- namespaces (needed to isolate containered service from SO file system mounts, process management and networking)
- control groups (cgroups - limit the use of filesystem resources)
- capabilities (for kernel operations and system call)
- secure computing mode (seccomp - restrict access to system calls)

images: a sort of template for container, union of multiple filesystem that are organized to resemble the root filesystem of a typical Linux distribution. Docker typically rely on an image and adds a read/write layer that container can update



network: combination of network namespaces and bridge within the host. SO host proxies traffic between outside world e containers

docker - architecture -

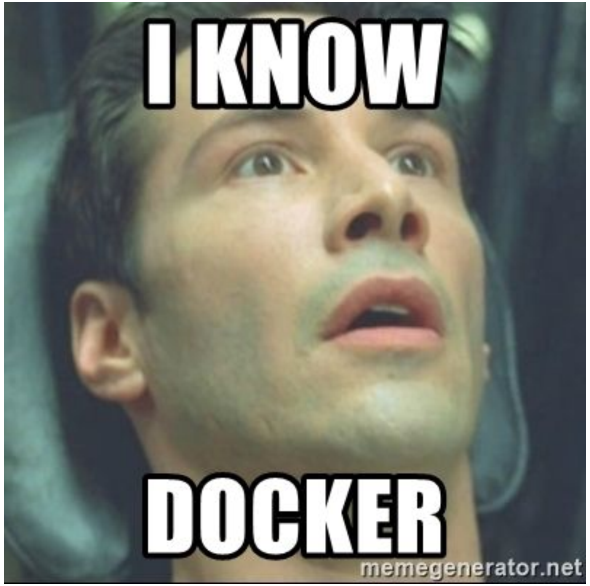


docker (docker client) is an executables that handles all management tasks for a Docker system

dockerd is the persistent daemon process that implements container and image operations (can be reached from same machine or over TCP)

docker registry stores docker images (Docker Hub is the default public registry)

docker - objects -



[citing documentation (one of the best ever seen)]

images are read-only template with instructions for creating a Docker container. Often an image is based on another image

containers are runnable instance of image. You can create, start, stop, move or delete a container using Docker API or CLI. You can connect a container to one or more networks, attach storage, and create a new image based on its current state.

service allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers.

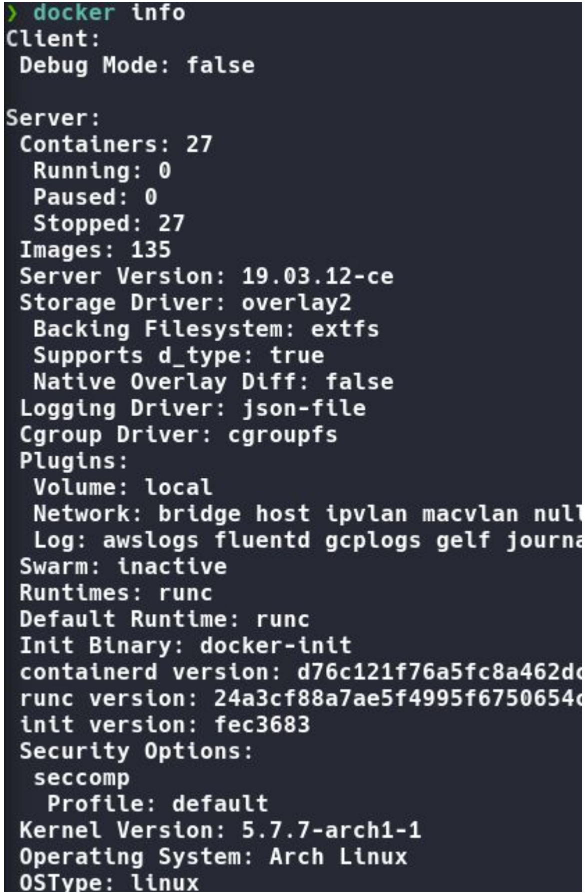
base commands

docker - cli -

docker info [options]

This command displays system wide information regarding the Docker installation. Information displayed includes the kernel version, number of containers and images.

options		
name	default	description
--format, -f		Format output using given GO template



```
❯ docker info
Client:
 Debug Mode: false

Server:
 Containers: 27
  Running: 0
  Paused: 0
  Stopped: 27
 Images: 135
 Server Version: 19.03.12-ce
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan nul
  Log: awslogs fluentd gcplogs gelf journa
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: d76c121f76a5fc8a462d
 runc version: 24a3cf88a7ae5f4995f6750654
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 5.7.7-arch1-1
 Operating System: Arch Linux
 OSType: linux
```

docker - cli -

docker command format

new management commands format:
docker command sub-command [options]

old management commands format (still works):
docker command [options]

docker - run -

docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

The docker run command creates a writeable container layer over the specified image, and starts it using the specified command.

A stopped container can be restarted with all its previous changes intact using docker start. See docker ps -a to view a list of all containers.

https://docs.docker.com/engine/reference/commandline/run/

options		
name	default	description
--detach, -d		Run container in background and print container ID
--name		Assign a name to the container
--publish, -p		Publish a container's port(s) to the host
--rm		Automatically remove the container when it exits

docker - run -

exercise:

run an nginx container and publish the 80 port of the container to 8080 of host (`--publish 8080:80`)

docker - run -

docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

--detach to make it run into background (docker ps to watch what’s running)

--name to name my container

options		
name	default	description
--detach, -d		Run container in background and print container ID
--name		Assign a name to the container
--publish, -p		Publish a container’s port(s) to the host
--rm		Automatically remove the container when it exits

docker - run -

what happens when I execute:

```
docker container run --publish 8080:80 --name webservice nginx
```

- docker looks locally for nginx image
- if not found look to remote (default registry Docker Hub)
- create a new container based on image and prepare to start
- gives it a virtual ip on virtual network inside docker engin
- open 8080 in host and forward to 80 in container
- start container with CMD into image Dockerfile

docker - run -

exercise:

run an nginx, a httpd and a mysql container
detach all of them
publish nginx on 80:80 port, httpd 8080:80 and mysql on 3306:3306
in mysql use --env MYSQL_RANDOM_ROOT_PASSWORD=yes

docker - logs -

docker container logs [OPTIONS] CONTAINER

The docker logs command batch-retrieves logs present at the time of execution.

options		
name	default	description
--details		Show extra details provided to logs
--follow, -f		Follow log output
--timestamps , -t		Show timestamps

docker - stop -

docker container stop [OPTIONS] CONTAINER [CONTAINER ...]

Stop one or more running containers

options		
name	default	description
--time, -t	10	Stop one or more running containers

docker - rm -

docker container rm [OPTIONS] CONTAINER [CONTAINER ...]

Remove one or more containers

options		
name	default	description
--force , -f		Force the removal of a running container (uses SIGKILL)
--link , -l		Remove the specified link
--volumes , -v		Remove anonymous volumes associated with the container

docker - inspect -

docker container inspect [OPTIONS] NAME|ID [NAME|ID...]

Return low-level information on Docker objects

options		
name	default	description
--format , -f		Format the output using the given Go template
--size , -s		Display total file sizes if the type is container
--type		Return JSON for specified type

docker - run -

no need for ssh, we can interact directly with container thanks to -it

```
docker container run -it --name myubuntu ubuntu
```

```
docker container exec -it myubuntu
```

docker image

docker - image -

A Docker image is a lightweight, stand-alone, and executable package that contains everything needed to run a piece of software, including the code, runtime, libraries, and system tools.

Images are the building blocks of containers in Docker. They are read-only and provide the basis for creating containers. (a metaphor with object-oriented languages: images are the classes and containers are the objects)

Docker images are often created from a **Dockerfile**, which defines the instructions to build the image.

Base operations:
pull, **push**, **build**

docker - image pull -

docker image pull [OPTIONS] NAME[:TAG|@DIGEST]

Download an image from a registry

https://docs.docker.com/engine/reference/commandline/pull/

options		
name	default	description
--all-tags , -a		Download all tagged images in the repository
--disable-content-trust	true	Skip image verification
--platform		Set platform if server is multi-platform capable
--quiet, -q		Suppress verbose output

docker - image build -

docker image build [OPTIONS] PATH | URL | -

Build an image from a Dockerfile

https://docs.docker.com/engine/reference/commandline/build/

options		
name	default	description
--file , -f		Name of the Dockerfile (Default is PATH/Dockerfile)
--network		Set the networking mode for the RUN instructions during build
--tag, -t		Name and optionally a tag in the name:tag format

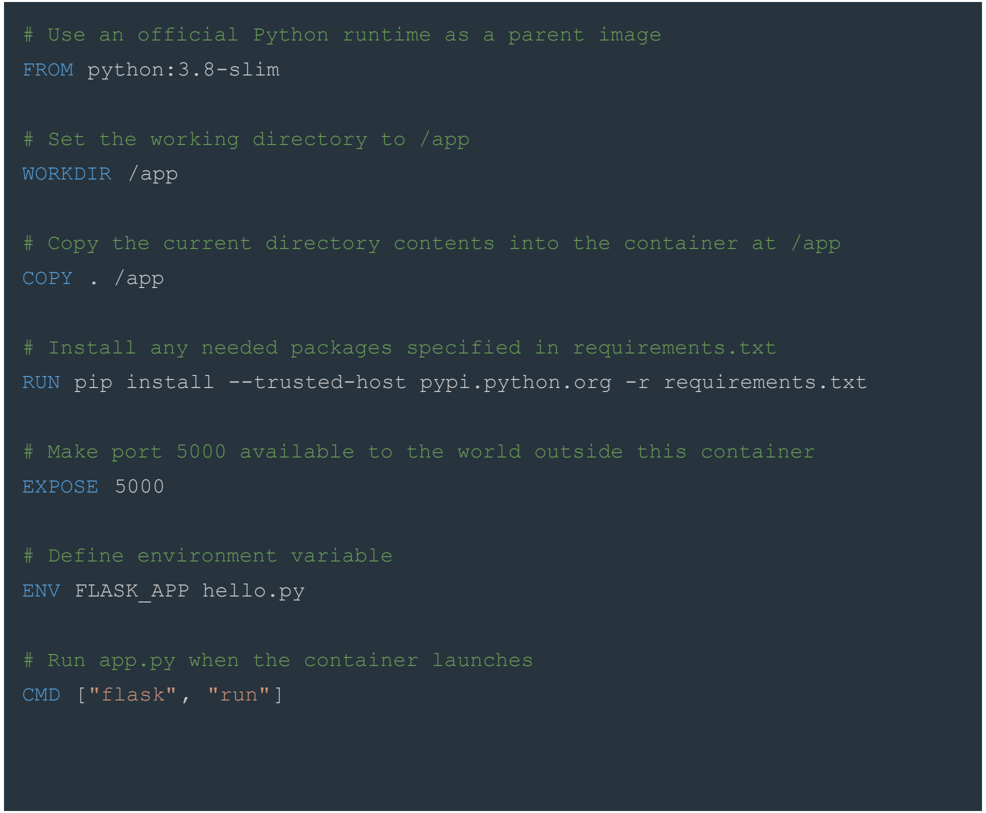
docker - Dockerfile -

Base syntax:

# Comment
INSTRUCTION arguments

Text file containing a set of instructions for building the image

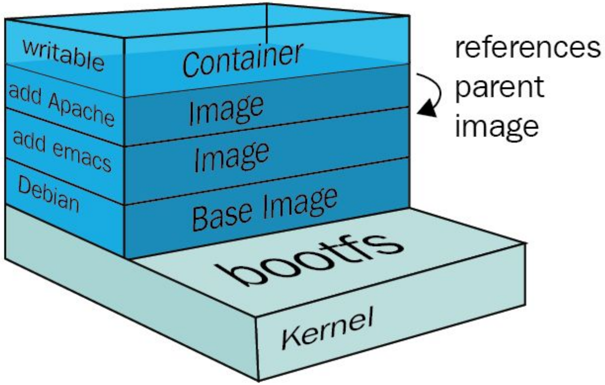
https://docs.docker.com/engine/reference/builder/



```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 5000 available to the world outside this container
EXPOSE 5000

# Define environment variable
ENV FLASK_APP hello.py

# Run app.py when the container launches
CMD ["flask", "run"]
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

docker - image -



https://subscription.packtpub.com/book/cloud-and-networking/9781788992329/1/ch01lvl1sec05/understanding-docker-images-and-layers

A Dockerfile is a script that defines a Docker image. It includes a set of instructions to create the image, specifying the base image, copying files, setting environment variables, and more

Docker images are composed of multiple layers, which are stacked on top of each other

Each instruction in a Dockerfile creates a new layer, and Docker caches these layers to optimize the build process

This layering system allows for efficient use of storage and sharing of common layers among multiple images

docker - image pull -

docker image push [OPTIONS] NAME[:TAG]

Upload an image to a registry

https://docs.docker.com/engine/reference/commandline/push/

options		
name	default	description
--all-tags , -a		Push all tags of an image to the repository
--disable-content-trust	true	Skip image signing
--quiet, -q		Suppress verbose output

docker compose

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

docker - compose -

Docker Compose is a tool for defining and running multi-container Docker applications

It allows you to define all your application services, networks, and volumes in a single docker-compose.yml file

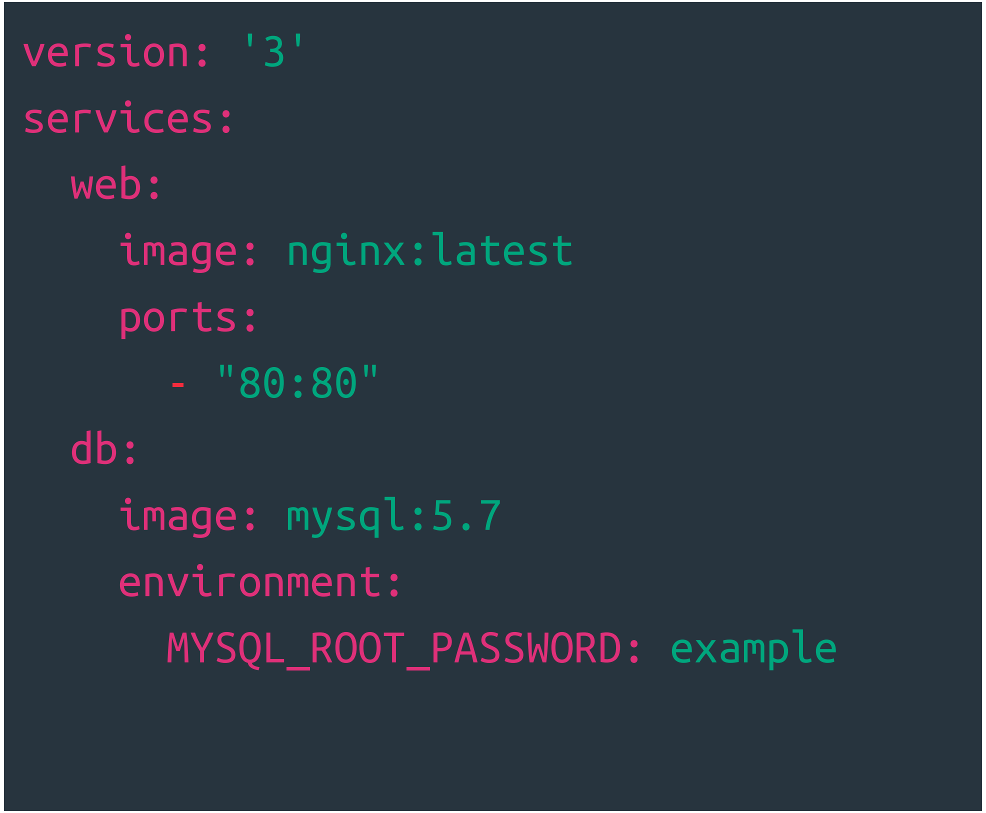
Use cases:

- Docker Compose is beneficial when you have a **multi-container application** with various dependencies
- It simplifies the process of **starting and managing complex applications** with multiple interconnected services
- Use cases include setting up **development** environments, **testing** environments, and **production** environments in a structured manner

docker - Dockerfile -

A docker-compose.yml file is a YAML configuration file that defines your application's services, networks, and volumes

https://docs.docker.com/compose/faq/



```yaml
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
```

docker - compose -

Once you have your docker-compose.yml file, you can use Docker Compose to start and manage your application

Common commands include:

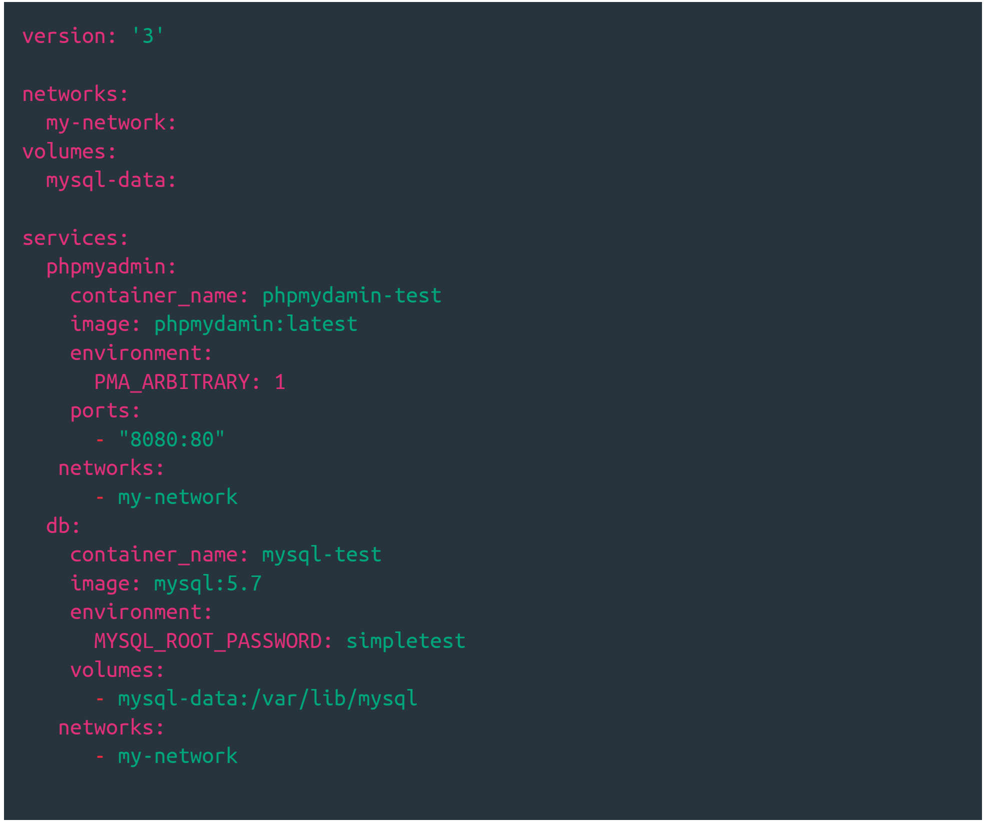
docker compose build: build all the application defined in the docker-compose.yml file (if they need build)
docker compose up: starts the application services defined in the docker-compose.yml file
docker compose down: stops and removes the application and its containers
docker compose ps: lists the status of the services
docker compose logs [<service-name>]: displays logs for the services
docker compose exec <service-name> <command>: executes a command in a running container

Me After Setting A Docker Container Up:
I'm something of a DevOps Engineer Myself

docker - compose -

Let’s compare running a simple phpmyadmin service plus mysql

docker network create mysql-test-net
docker volume create --name=mysql-data
docker container run -d --name phpmydamin-test --network mysql-test-net -e PMA_ARBITRARY=1 -p 8080:80 phpmyadmin
docker container run -d --name mysql-test --network mysql-test-net -v mysql-data:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=simpletest mysql



```
version: '3'

networks:
  my-network:
volumes:
  mysql-data:

services:
  phpmyadmin:
    container_name: phpmydamin-test
    image: phpmydamin:latest
    environment:
      PMA_ARBITRARY: 1
    ports:
      - "8080:80"
   networks:
      - my-network
  db:
    container_name: mysql-test
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: simpletest
    volumes:
      - mysql-data:/var/lib/mysql
   networks:
      - my-network
```

docker - play time -

Let’s start playing a bit! follow me more than writing exactly what I write!
Everything will be pushed to:

https://gitlab.com/frfaenza/cloudedgecomputing

“I would like to understand things better, but I don’t want to understand them perfectly.”

Douglas Hofstadter – 1985, Metamagical Themas: Questing for the Essence of Mind and Pattern