UNIVERSIDADE
BEIRA INTERIOR

Docker Containers

Tecnologias de Virtualização e Centros de Dados
Mestrado em Engenharia Informática

Alexandre Fonte
alexandre.fonte@ubi.pt
(Professor Convidado)
Departamento de Informática

Ano Letivo 2023/2024

Sumário

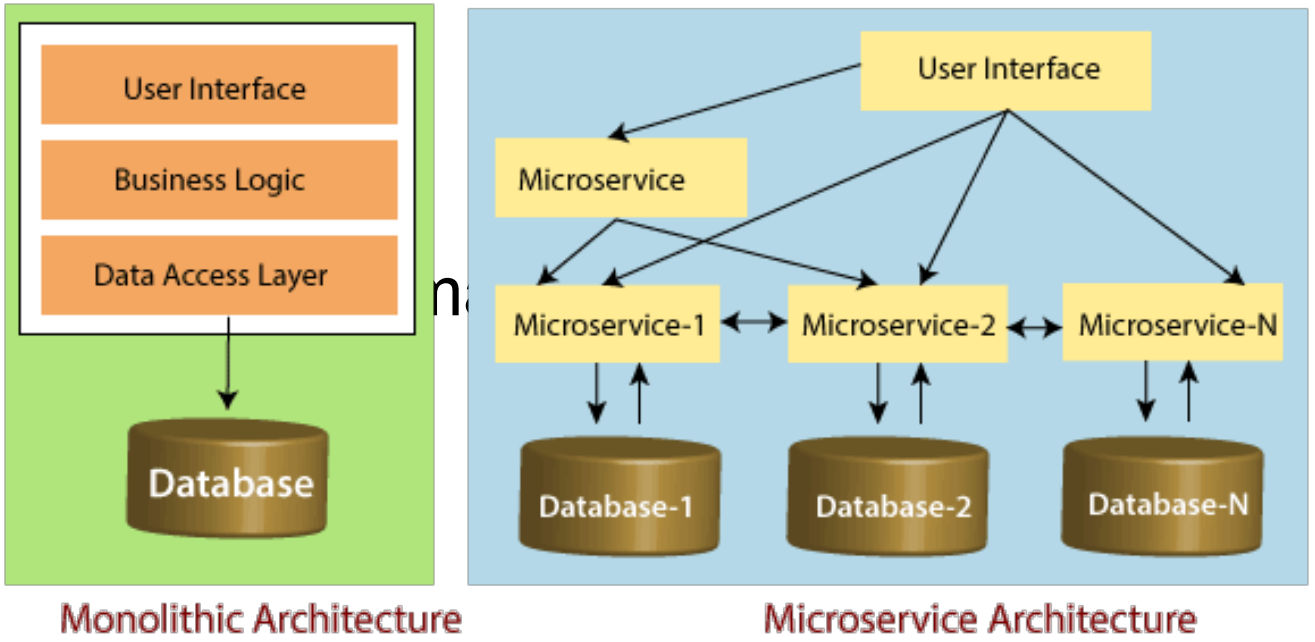
- Docker
- Docker compose

version: 04 april, 2023

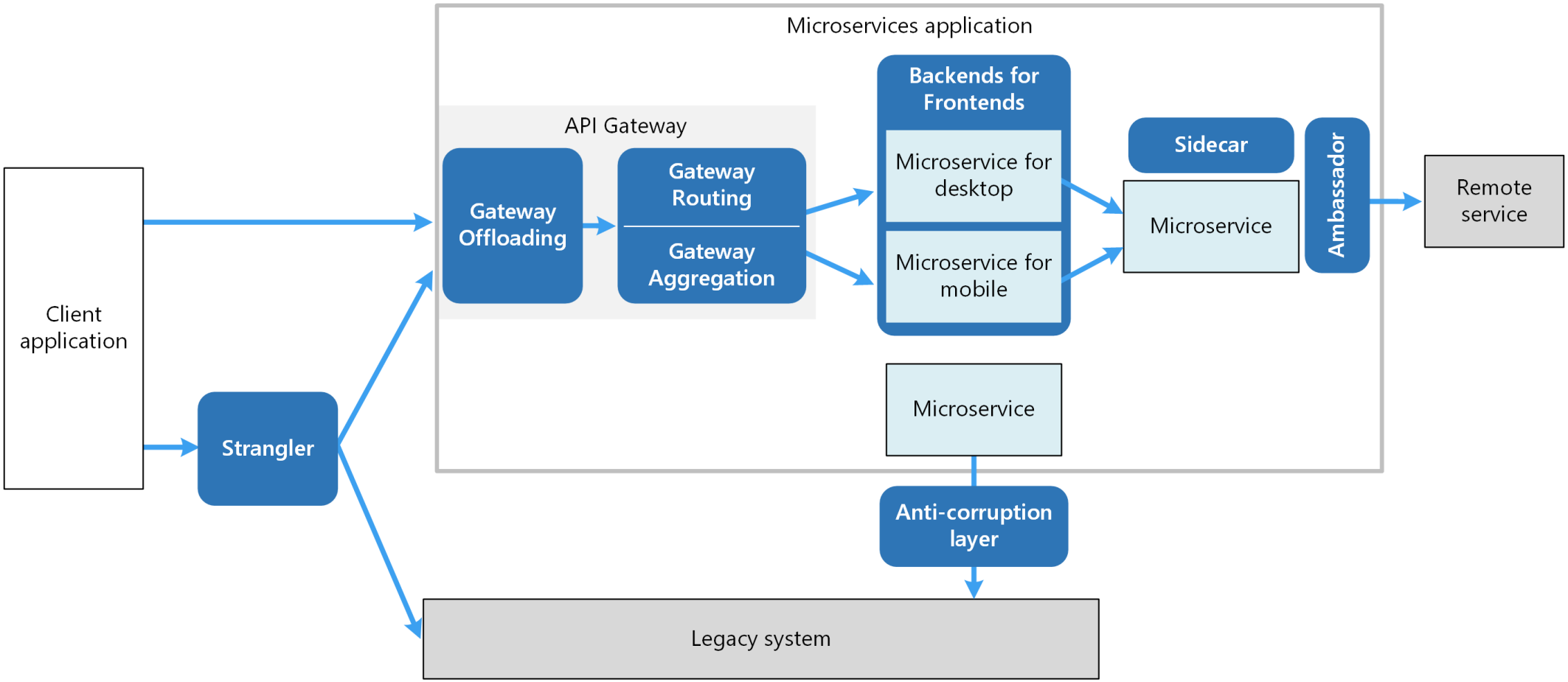
Microservices

Microservices

- Enterprises are heading toward microservices architectures
  - Build small focused microservices
  - Flexibility to innovate and build applications in different languages (Java, Python, JavaScript, etc)
  - BUT deployments become complex!
  - How can we have one way of deploying Java, Python, JavaScript Microservices? That's where containers come to the picture!



Microservices

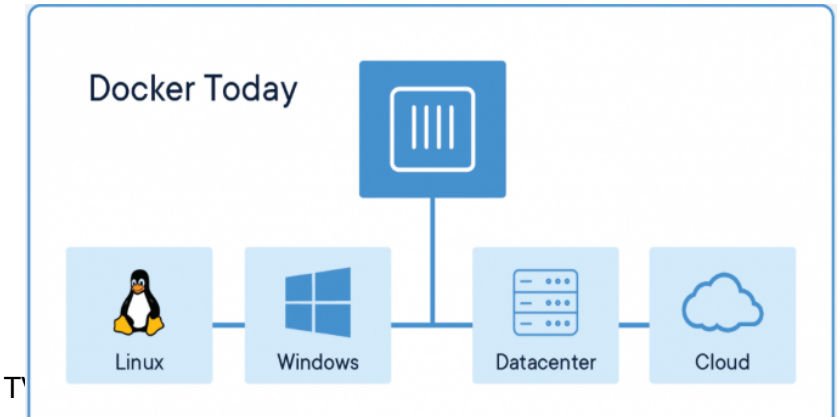


* Courtesy of Microsoft

Docker

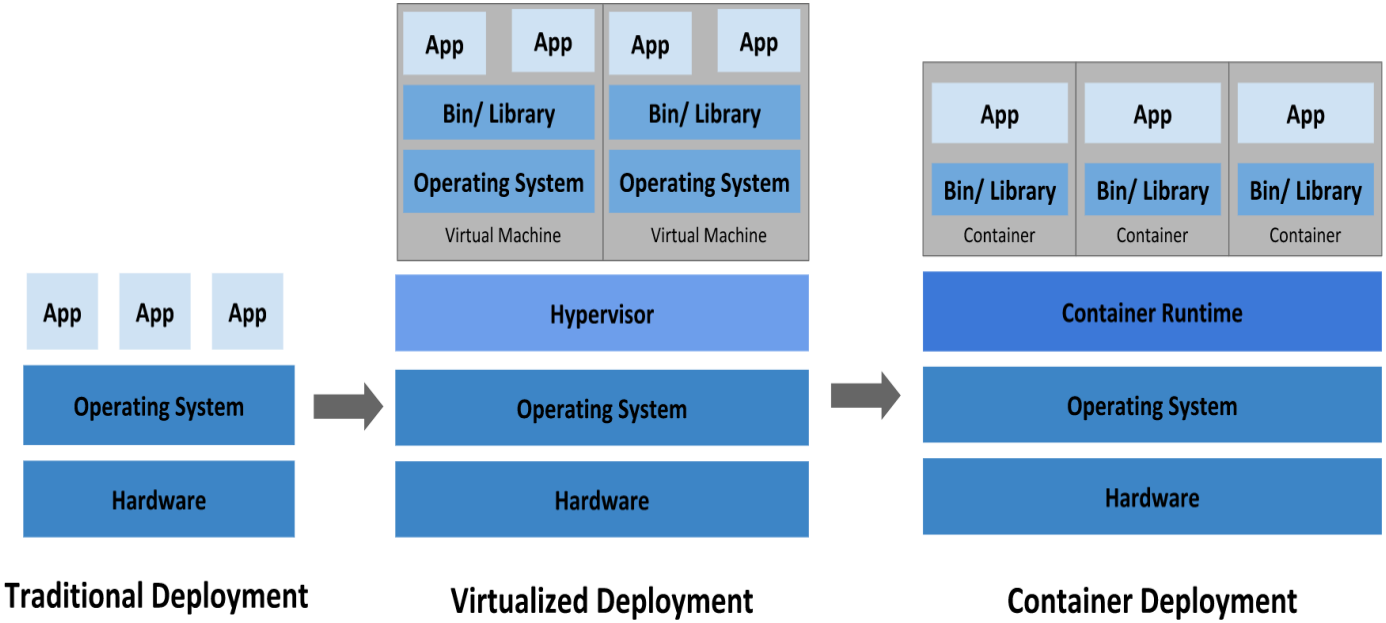
- You can create Docker images for each microservice
- Docker image contains everything a microservice needs to run:
  - -Application Runtime (JDK or Python or NodeJs)
  - -Application Code
  - -Dependencies
- You can run these Docker containers the same on any infrastructure
  - -Your local machine
  - -Corporate data center
  - -Cloud

Docker Containers are everywhere: Linux, Windows, Mac, Data center, Cloud, Serverless



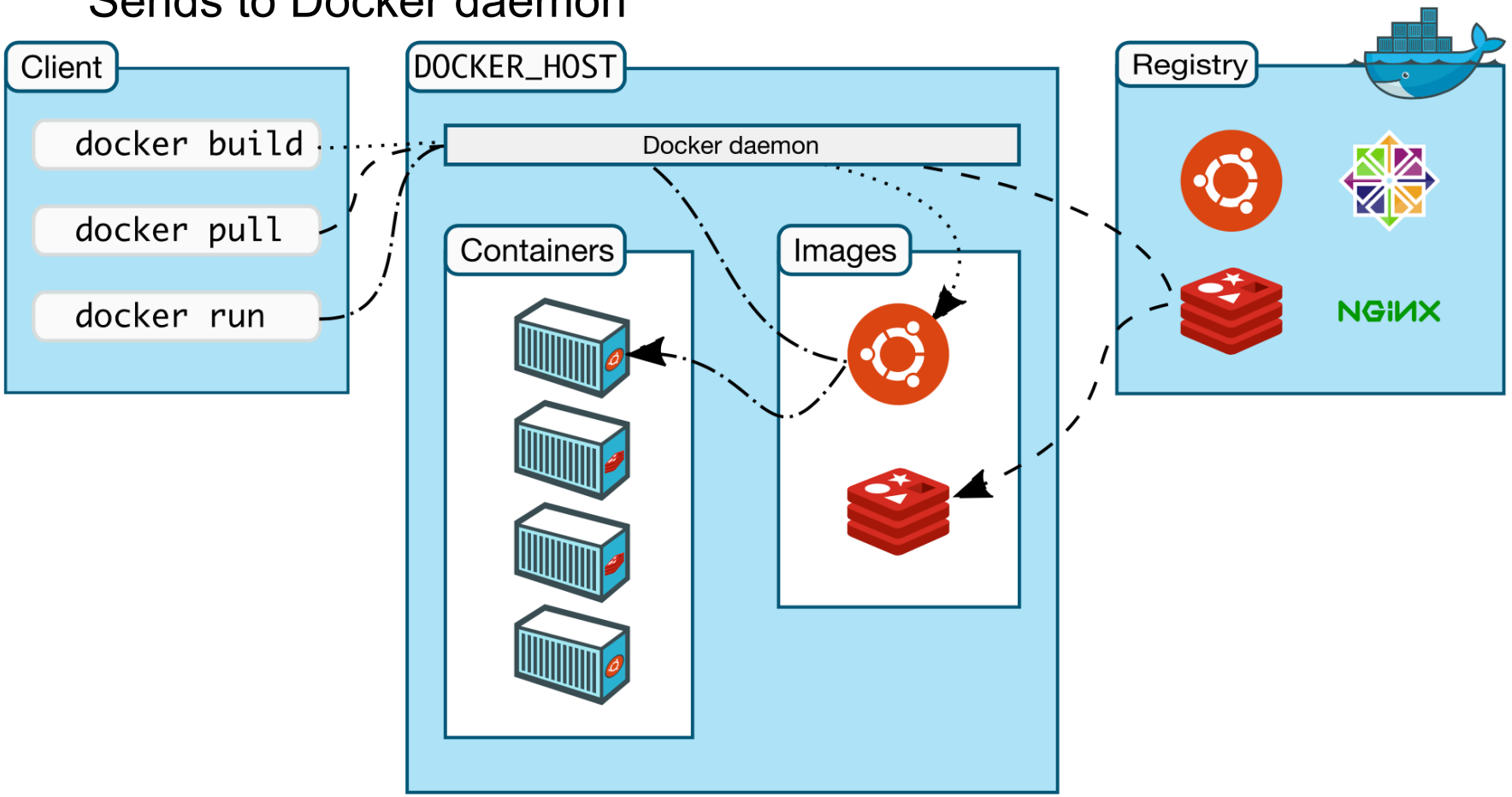
Docker

- Docker Inc. was founded by Solomon Hykes and Sebastien Pahl and launched in 2011. Docker debuted to the public in Santa Clara at PyCon in 2013 and it was released as open-source in March 2013.
- Docker uses OS-level virtualization to deliver software in packages called containers.
- Docker provides a way to run applications securely isolated in a container, packaged with all its dependencies and libraries.
- All containers are run by a single operating system kernel.



Docker Architecture

- Docker follows a client-server architecture
- When we install Docker Desktop we install
  - Docker client and Docker daemon

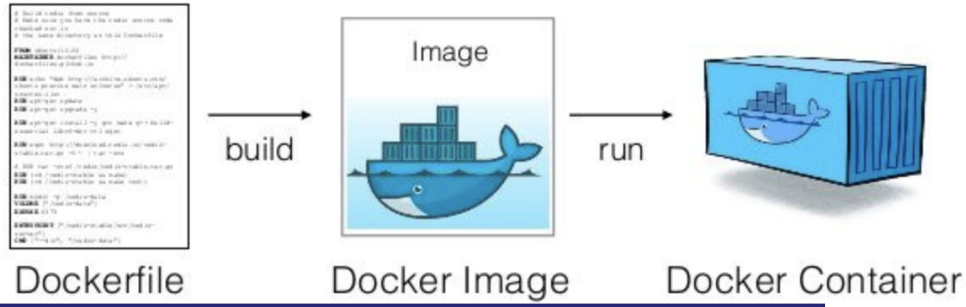


Docker Architecture

- The Docker daemon
  - The Docker daemon (**dockerd**) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.
- The Docker client
  - The Docker client (docker) is the primary way that many Docker users interact with Docker.
  - When one uses commands such as **docker run**, the client sends these commands to **dockerd**, which carries them out.
  - The **docker** command uses the Docker API to to control or interact with the Docker daemon through scripting or direct CLI commands.
  - The Docker client can communicate with more than one daemon.
- Docker registries
  - A Docker registry stores Docker images.
  - Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  - One can even run his/her own private registry.
  - When you use the docker pull or docker run commands, the required images are pulled from your configured registry.
  - When you use the docker push command, your image is pushed to your configured registry.

Docker image

- A Docker image é um pacote portátil que contém software e todas as dependências que esse software necessita
- Quando a imagem é carregada para memória e executada, passa a ser o **container**
- A imagem Docker é imutável, ou seja, não pode ser alterada. A única forma de "*alterar*" uma imagem docker é criar uma nova

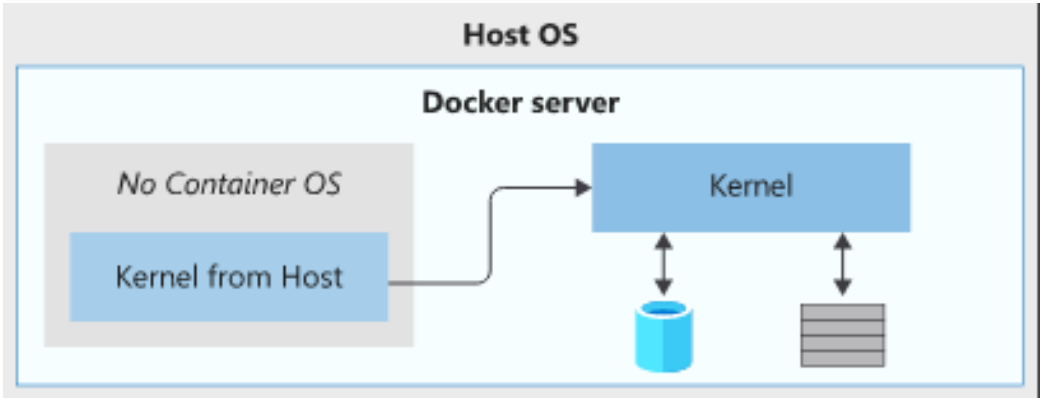


Images vs Containers

Image - a static template with instructions for creating a Container
Container is the Running version of an image
Image is like a Class; Container is like an Object!

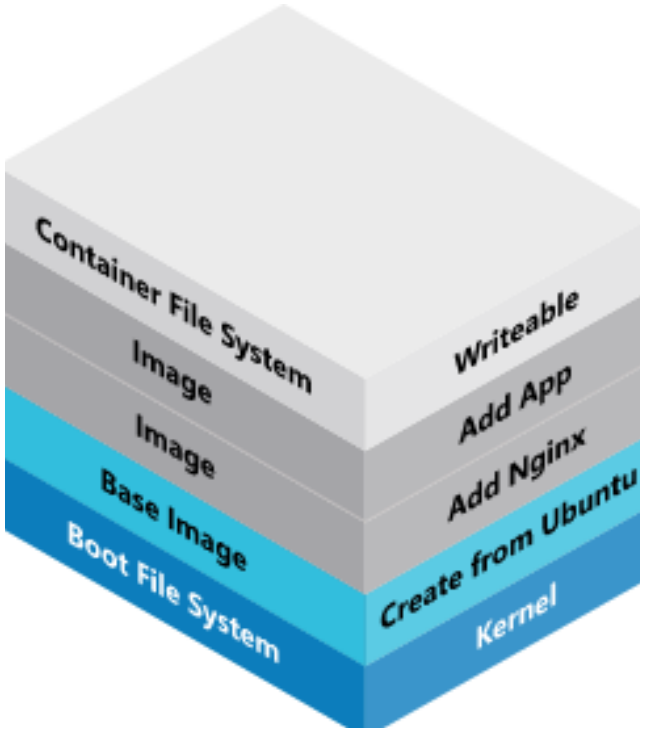
O container não usa virtualização de HW

- O container é uma unidade independente de computação
- É **um virtual runtime environment isolado,** que pode ter os seus processos ou serviços, as suas próprias interfaces de rede, os seus próprios mounts como as VMs, exceto que corre diretamente sobre Kernel do sistema operativo onde está instalado o Docker Engine, partilhando o SO com outros containers.
- Desta forma, há enormes ganhos de desempenho relativamente às Virtual Machines. O objetivo não é virtualizar, o objetivo é *containerizar* as aplicações, as transportar e as correr.
- Docker na verdade tem vindo a utilizar contentores LXC.



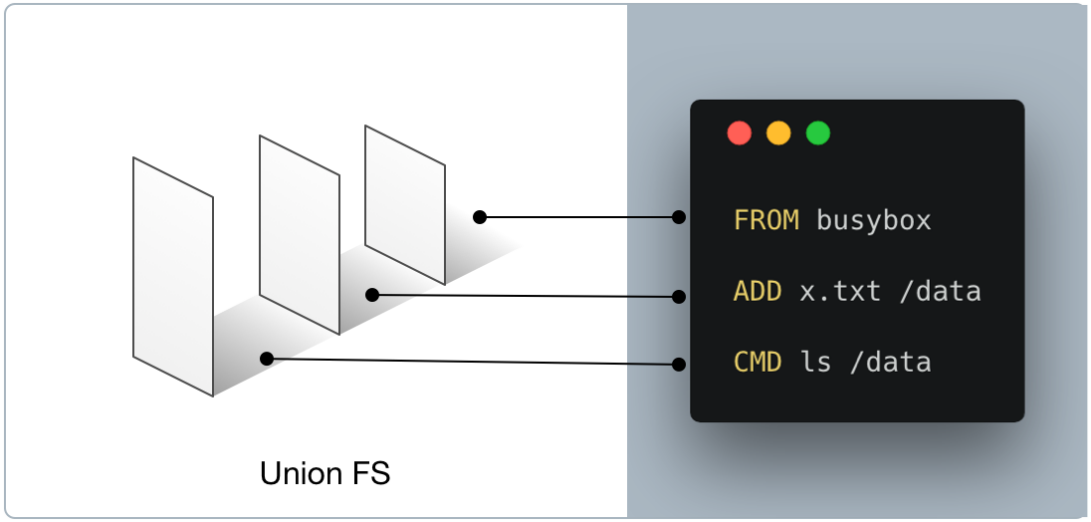
Sistema de ficheiros do Docker container

- As imagens docker usam um sistema de ficheiros chamado **unionfs (Union File System)**, que permite empilhar vários sistemas de ficheiros (branches) diferentes
- O conteúdo destes vários branches é fundido e visto de uma forma unificada
  - -Uma imagem é construída empilhando vários branches
  - -Só é possível ir adicionando branches, não é possível apagar ou substituir branches inferiores
  - -O último branch permite escrita de ficheiros, mas não é persistente -desaparece quando o container é desligado



Sistema de ficheiros do Docker container

- Os ficheiros necessários são definidos num Dockerfile
  - -Cada linha de código será eventualmente uma camada no UnionFS.
  - -Com a montagem em união, os directórios no sistema de ficheiros da camada subjacente fundem-se com os da camada superior do sistema de ficheiros.
  - -Os ficheiros com o mesmo nome nas camadas subjacentes são mascarados. No entanto, o programa em execução dentro do contentor não se importa de que camada provêm os ficheiros e directórios, mas sim de um sistema de ficheiros coerente.

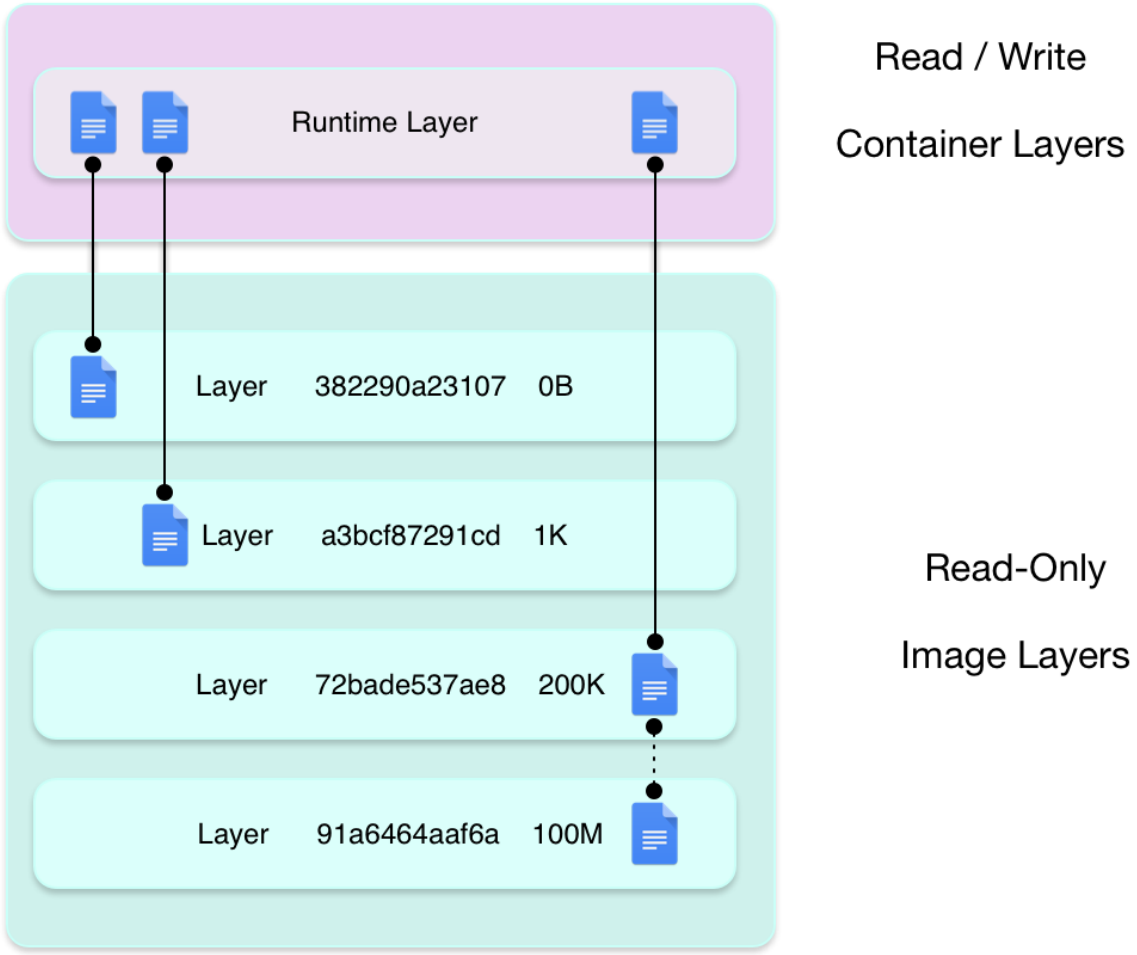



```
layer 1: /bin/sh, /bin/cp, /bin/cd
layer 2: /bin/cd
layer 3: /bin/zsh

result: /bin/sh, /bin/cp, /bin/cd (from layer 2), /bin/zsh
```

Sistema de ficheiros do Docker container

- Note that when a container is created, a **writable layer is also created on top** of the image layers.



Uma imagem pode ser feita a partir de:

- **Base Image**: é uma imagem que usa a imagem docker "scratch", uma imagem vazia que não cria uma camada com um sistema de ficheiros. Esta imagem pode ser usada para aplicações que usam diretamente o Kernel do SO onde está o Docker Engine
- **Parent Image**: é qualquer imagem que seja usada para criar outra imagem. A Parent Image pode conter já um SO, e software instalado, por exemplo o NGINX

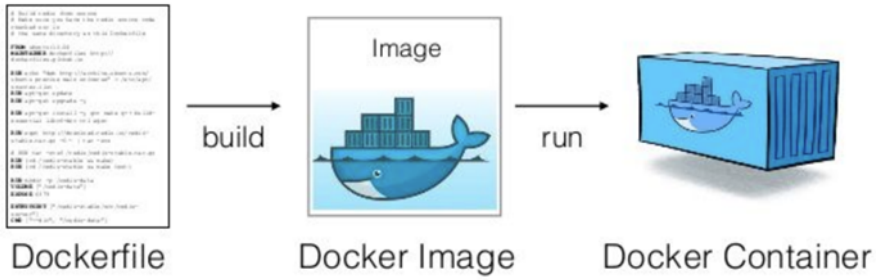
Dockerfile

- A dockerfile é um ficheiro de texto com as instruções para criar uma imagem Docker
- Exemplo de um dockerfile para criar uma imagem a partir do Ubuntu 18 e com o serviço MariaDB

```
FROM ubuntu:18.04

RUN apt-get update && \
    apt-get install -y mariadb-server && \
    apt-get clean

EXPOSE 3306
```



Construindo uma nova imagem...

- A forma mais simples de criar imagens é partir de uma imagem existente e usar um Dockerfile

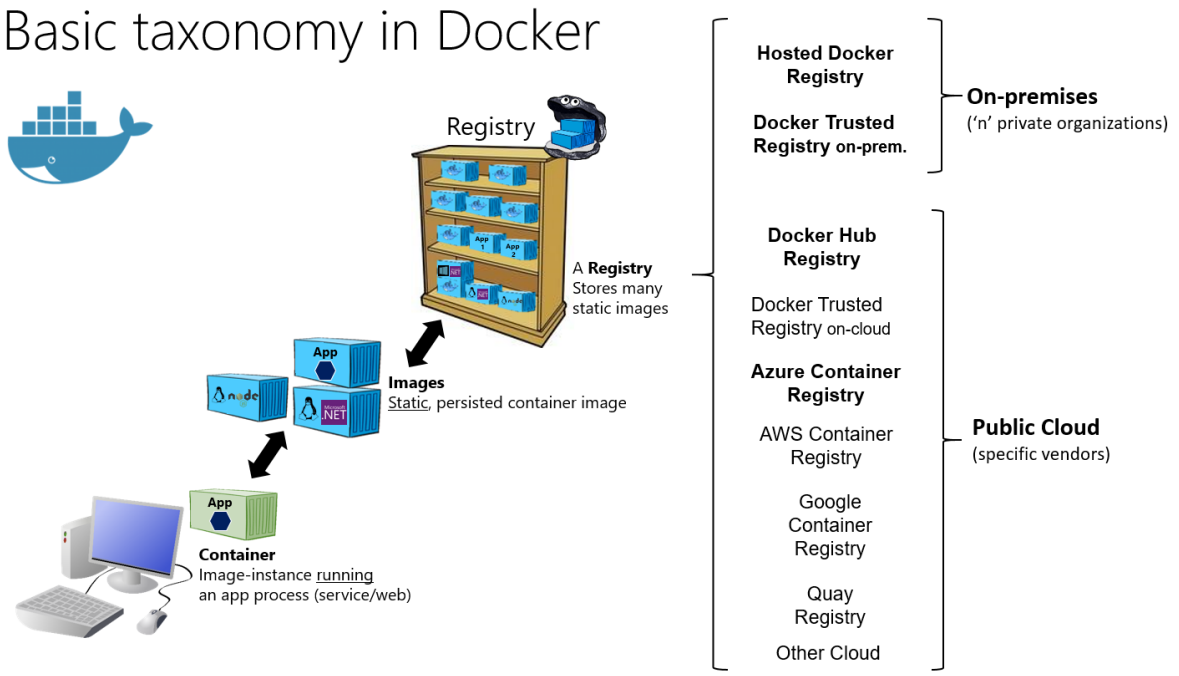
```
FROM mcr.microsoft.com/dotnet/core/sdk:2.2
WORKDIR /app
COPY myapp_code .
RUN dotnet build -c Release -o /rel
EXPOSE 80
WORKDIR /rel
ENTRYPOINT ["dotnet", "myapp.dll"]
```

In this file:

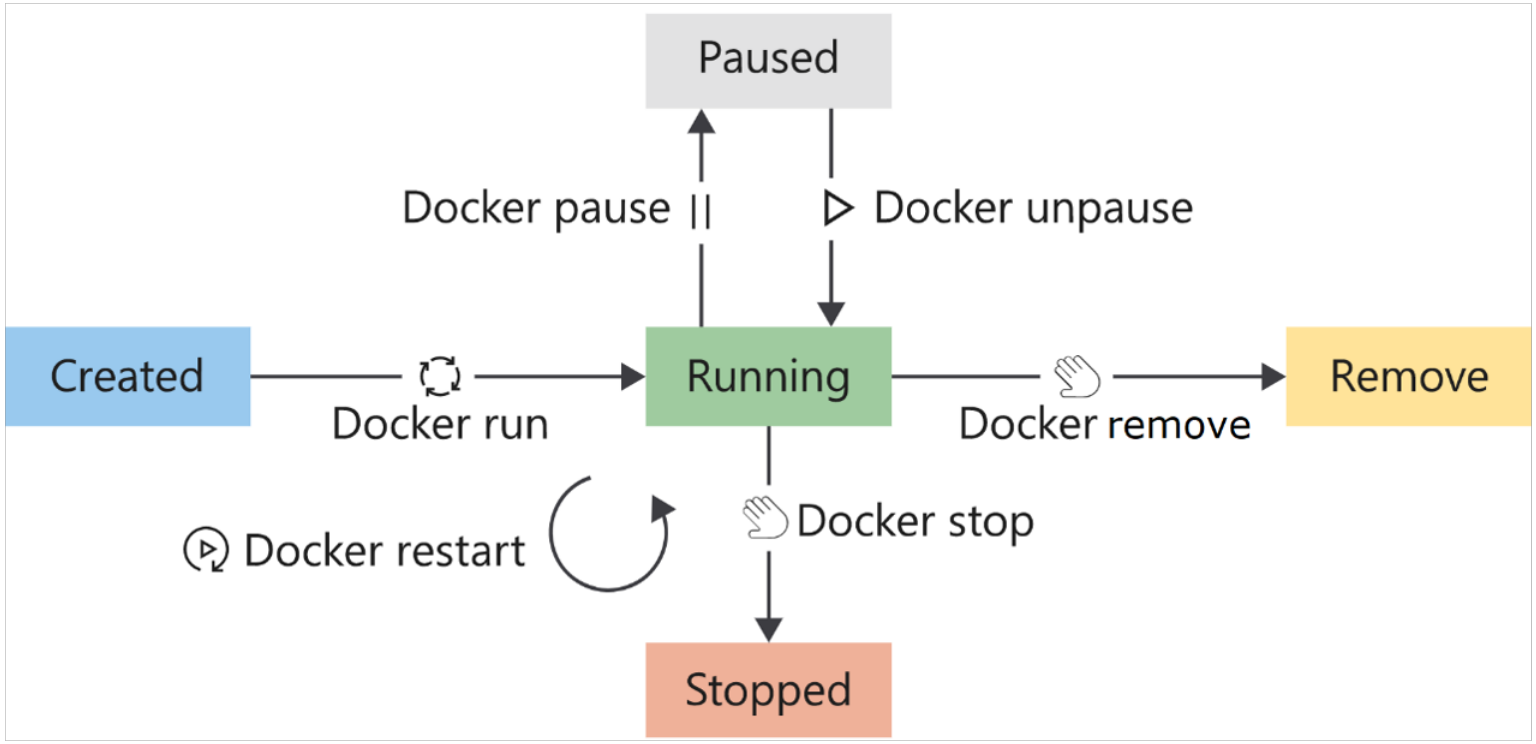
- The **FROM** statement downloads the specified image and creates a new container based on this image.
- The **WORKDIR** command sets the current working directory in the container, used by the following commands.
- The **COPY** command copies files from the host computer to the container. The first argument (`myapp_code`) is a file or folder on the host computer. The second argument (`.`) specifies the name of the file or folder to act as the destination in the container. In this case, the destination is the current working directory (`/app`).
- The **RUN** command executes a command in the container. Arguments to the RUN command are command-line commands.
- The **EXPOSE** command creates configuration in the new image that specifies which ports are intended to be opened when the container is run. If the container is running a web app, it's common to EXPOSE port 80.
- The **ENTRYPOINT** command specifies the operation the container should run when it starts. In this example, it runs the newly built app. You specify the command to be run and each of its arguments as a string array.

Image Registries

- As imagens Docker são registadas e armazenadas em registries
- Um registry é um serviço Web ao qual o Docker se liga para guardar ou descarregar imagens
- O registry público mais popular é o Docker HUB



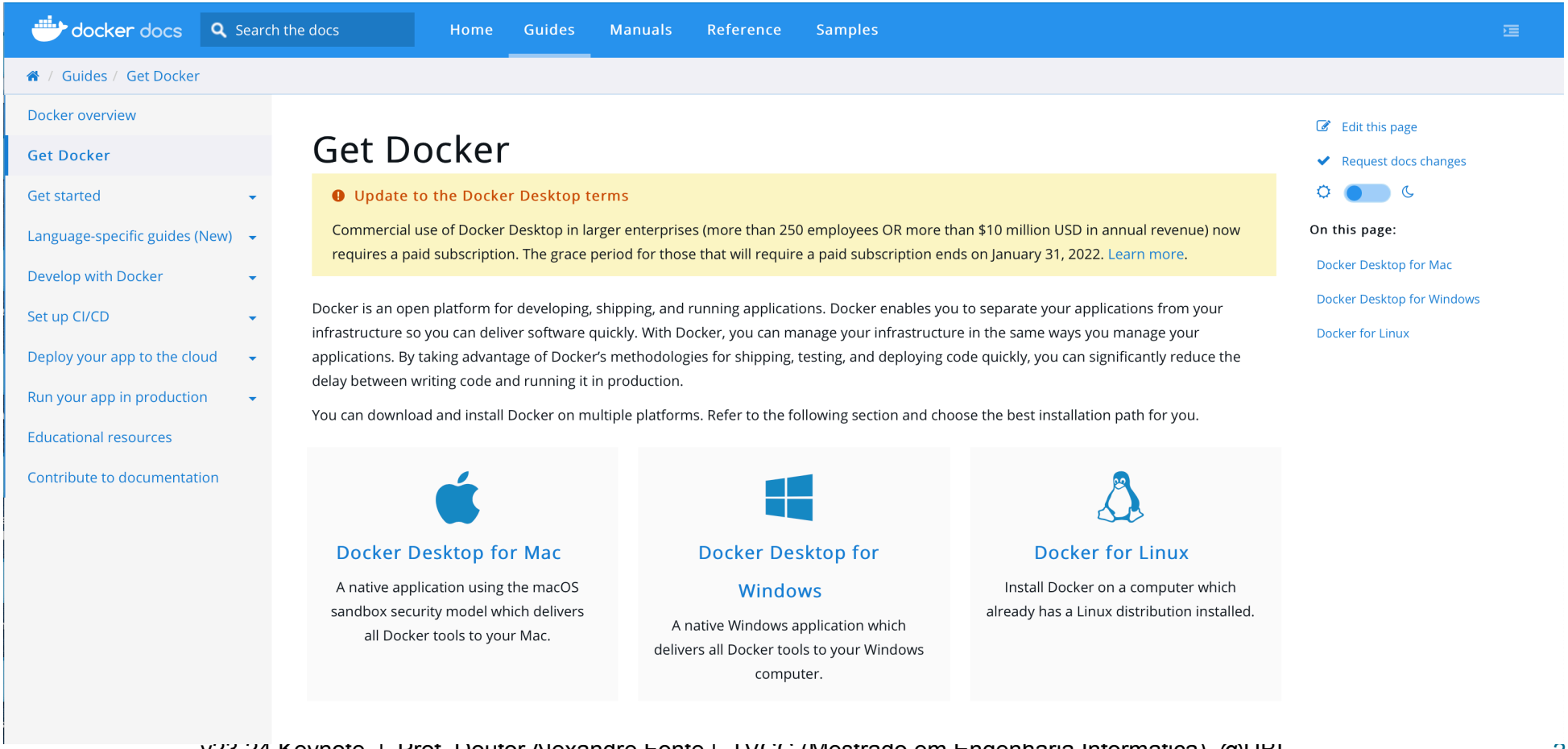
Docker container lifecycle



Learn Docker
Deploying Applications

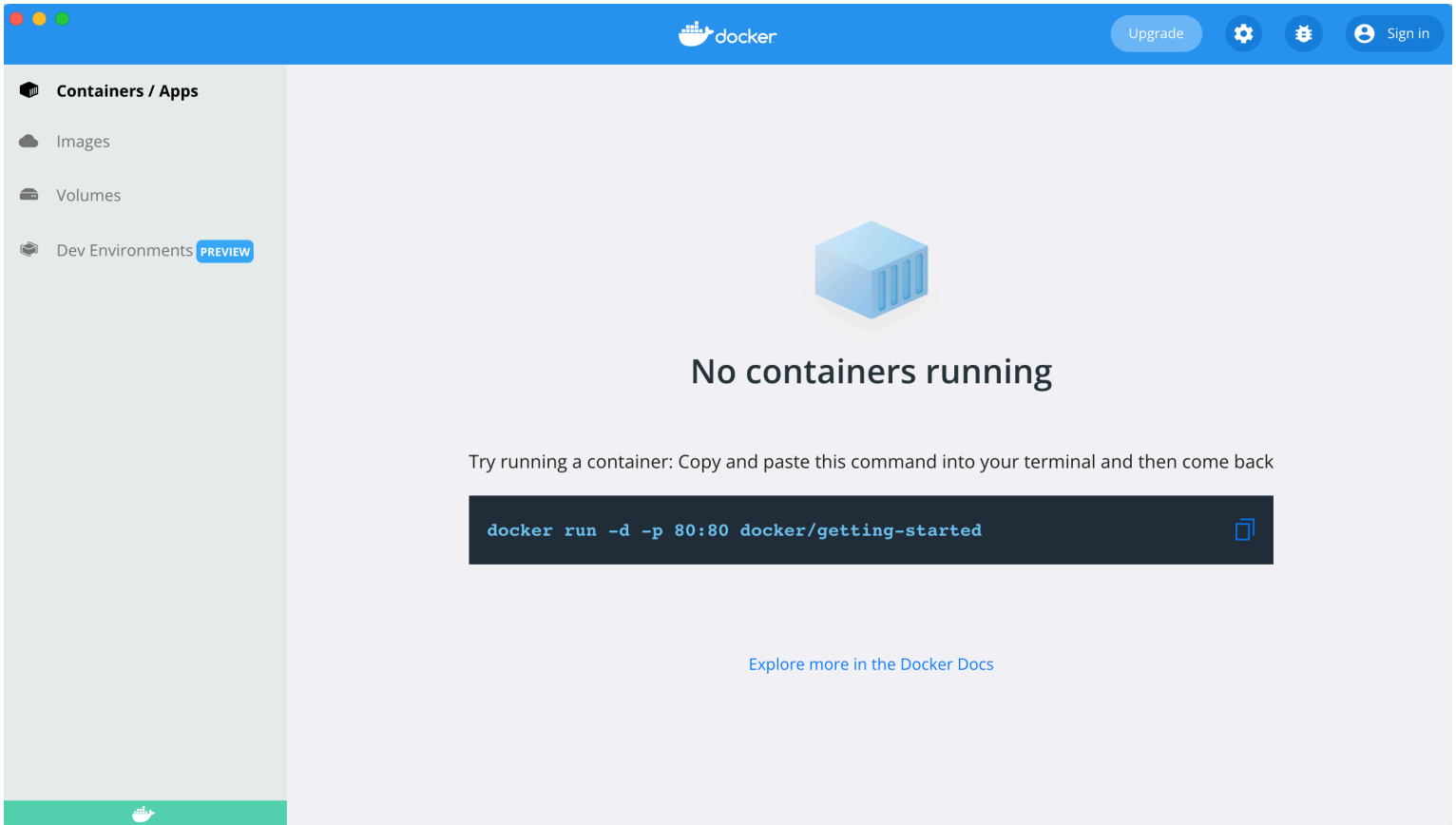
Docker

- **Step 1: Install Docker Desktop on Windows, Mac or Linux**
  - Go to Docker Documentation searching the Web or:
  - docs.docker.com/install
  - Select Download and Install



Docker

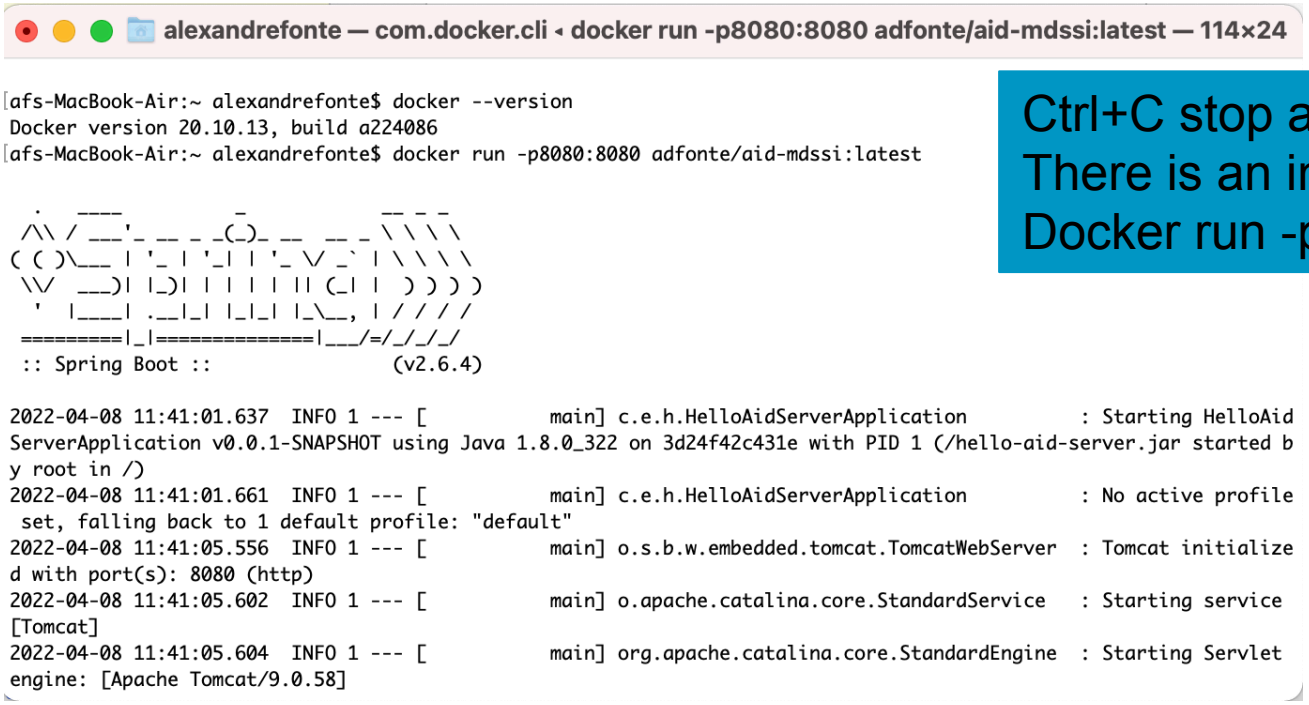
- **Step 2: Start Docker**
  - On launch, Docker may ask Admin Password to install network components
  - “Beginning on August 31, 2021 You must agree to the Docker Subscription Service Agreement to continue using Docker Desktop”



Docker

- **Step 3: Deploy an Spring Boot Application**
  - Open terminal window, check docker version and deploy a Spring Boot Application as a Container

```
docker --version
docker run -p8080:8080 adfonte/hello-server:latest
```



Ctrl+C stop a container
There is an internal docker network
Docker run -p {HostPort}:{ContainerPort}

Recommendations for Windows Users

- If you are using Windows, make sure you are using PowerShell
- If you are using Windows 10 use IP 192.168.99.100 instead of localhost. Use command docker-machine ip (maybe you need to install this command)

Docker

- **Step 3: Deploy a Spring Boot Application (Cont.)**
  - Go to the Browser and Enter the following URI

    `http://localhost:8080/messages/estudantes/TVCD`

Good moorning Dear estudantes from our TVCD Docker Container!

It is 07:08, now.

Ctrl+C stop a container
There is an internal docker network
Docker run -p {HostPort}:{ContainerPort}

Recommendations for Windows Users

- If you are using Windows, make sure you are using PowerShell
- If you are using Windows 10 use IP 192.168.99.100 instead of localhost. Use command docker-machine ip (maybe you need to install this command)

Docker

Ctrl+C stop a container
There is an internal docker network
Docker run -p {HostPort}:{ContainerPort}

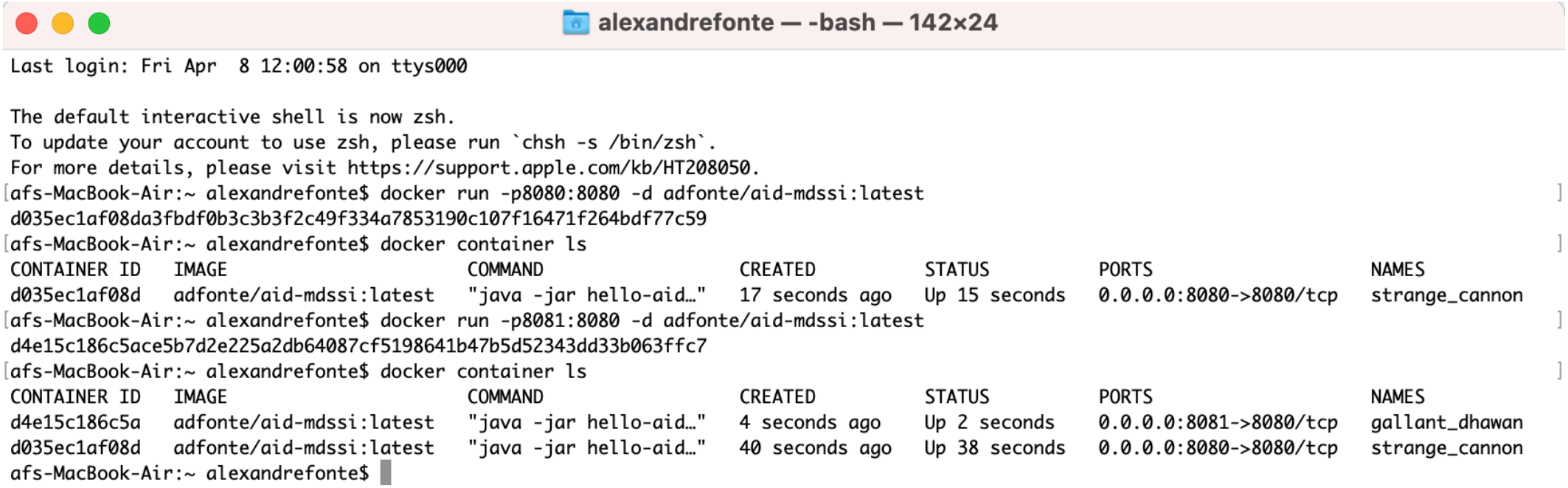
- **Step 4: Go to Docker Registry**
  - https://hub.docker.com
  - Docker Registry has many repositories, each contains applications and different versions of applications
  - Deploy two Web Services
    - Apache Web Service
    - Tutum

```
docker pull tutum/hello-world
docker run -p80:80 tutum/hello-world:latest
docker pull httpd
docker run -p81:80 httpd:latest
```

Docker

- **Step 5: Run in Background two Containers from the same image**
  - Open a new terminal window
  - Option **-d** run in background

```
docker run -p8080:8080 -d adfonte/hello-server:latest
```



Docker

- **Step 6: Stop a Container**
  - **`docker container stop` *`container id`***

alexandrefonte — -bash — 165×24

```
afs-MacBook-Air:~ alexandrefonte$ docker container ls
CONTAINER ID   IMAGE                       COMMAND                  CREATED         STATUS         PORTS                    NAMES
d4e15c186c5a   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   3 minutes ago   Up 3 minutes   0.0.0.0:8081->8080/tcp   gallant_dhawan
d035ec1af08d   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   3 minutes ago   Up 3 minutes   0.0.0.0:8080->8080/tcp   strange_cannon
afs-MacBook-Air:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS                      PORTS                    NAMES
d4e15c186c5a   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   3 minutes ago    Up 3 minutes                0.0.0.0:8081->8080/tcp   gallant_dhawan
d035ec1af08d   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   3 minutes ago    Up 3 minutes                0.0.0.0:8080->8080/tcp   strange_cannon
3d24f42c431e   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   11 minutes ago   Exited (130) 4 minutes ago                           zealous_northcutt
5a309ffd8f59   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   12 minutes ago   Created                                              loving_tereshkova
afs-MacBook-Air:~ alexandrefonte$ docker container stop d4e15
d4e15
afs-MacBook-Air:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS                      PORTS                    NAMES
d4e15c186c5a   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   3 minutes ago    Exited (143) 3 seconds ago                           gallant_dhawan
d035ec1af08d   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   4 minutes ago    Up 4 minutes                0.0.0.0:8080->8080/tcp   strange_cannon
3d24f42c431e   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   12 minutes ago   Exited (130) 4 minutes ago                           zealous_northcutt
5a309ffd8f59   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   12 minutes ago   Created                                              loving_tereshkova
afs-MacBook-Air:~ alexandrefonte$ docker container ls
CONTAINER ID   IMAGE                       COMMAND                  CREATED         STATUS         PORTS                    NAMES
d035ec1af08d   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   4 minutes ago   Up 4 minutes   0.0.0.0:8080->8080/tcp   strange_cannon
afs-MacBook-Air:~ alexandrefonte$
```

Playing with Docker Images

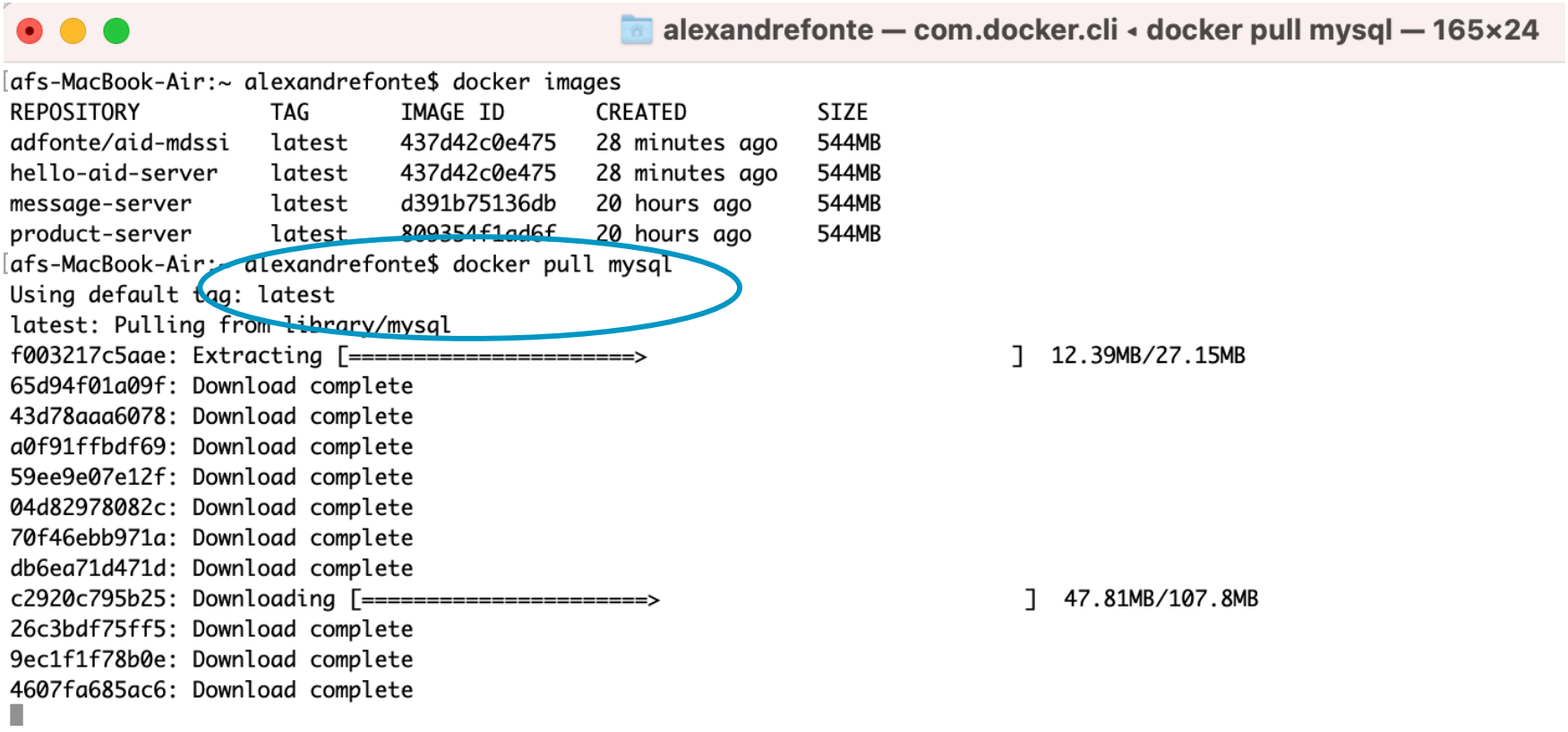
- Commands related with Images
  - Multiple Tags

```
alexandrefonte — -bash — 165×24
afs-MacBook-Air:~ alexandrefonte$ docker pull mysql:oracle
oracle: Pulling from library/mysql
e4430e06691f: Pull complete
99677fe408e0: Pull complete
911aa3c638bf: Pull complete
2a50ae0f3675: Pull complete
6e229cecb99f: Pull complete
c1790c22e04b: Pull complete
d99e6dd61db9: Pull complete
45eb6c638025: Pull complete
73dc8a66483d: Pull complete
52190d1e6a3f: Pull complete
Digest: sha256:041a3f0b3d58771afc64762b46289385f2b9126a14ab7b5895e83b87fd26f1bc
Status: Downloaded newer image for mysql:oracle
docker.io/library/mysql:oracle
afs-MacBook-Air:~ alexandrefonte$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
adfonte/aid-mdssi   latest    437d42c0e475   33 minutes ago   544MB
hello-aid-server    latest    437d42c0e475   33 minutes ago   544MB
message-server      latest    d391b75136db   20 hours ago     544MB
product-server      latest    809354f1ad6f   21 hours ago     544MB
mysql               oracle    0ca4d64ca878   8 days ago       435MB
mysql               latest    667cc8fb158c   9 days ago       521MB
afs-MacBook-Air:~ alexandrefonte$
```

```
ADF-iMac:~ alexandrefonte$ docker tag adfonte/hello-server:latest adfonte/hello-server:v1
ADF-iMac:~ alexandrefonte$ docker image ls
REPOSITORY                TAG       IMAGE ID       CREATED              SIZE
adfonte/hello-server      latest    4c9e9fc7d794   About an hour ago    544MB
adfonte/hello-server      v1        4c9e9fc7d794   About an hour ago    544MB
hello-server              latest    4c9e9fc7d794   About an hour ago    544MB
ADF-iMac:~ alexandrefonte$
```

Playing with Docker Images

- Command related with Images
  - **`docker pull`** *the lastest image*



Playing with Docker Images

- Command related with Images
  - **`docker push`** *the lastest image*

Precisa de login prévio.
Exemplo:

docker login -u adfonte

alexandrefonte — -bash — 80×24

```
afs-Air-2:~ alexandrefonte$ docker tag adfonte/hello-server adfonte/hello-server
:v1
afs-Air-2:~ alexandrefonte$ docker image ls
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
kalilinux/kali-rolling    latest    9a07550f0b56   3 months ago     118MB
mysql                     5.7       d410f4167eea   3 months ago     495MB
adfonte/hello-server      latest    d09ee31d0506   11 months ago    544MB
adfonte/hello-server      v1        d09ee31d0506   11 months ago    544MB
adfonte/aid-mdssi         latest    437d42c0e475   12 months ago    544MB
afs-Air-2:~ alexandrefonte$ docker push adfonte/hello-server:v1
The push refers to repository [docker.io/adfonte/hello-server]
03ac875a1fc3: Layer already exists
575b1387983f: Layer already exists
0f3078493dc2: Layer already exists
cac2fff6ae3d: Layer already exists
08fa02ce37eb: Layer already exists
a037458de4e0: Layer already exists
bafdbe68e4ae: Layer already exists
a13c519c6361: Layer already exists
v1: digest: sha256:36937376b835ac730b6978374f604eed434257845065aeb9ee5a0ad41c029
adb size: 2007
```

Playing with Docker Containers

- Commands related with Containers

```
afs-MacBook-Air:~ alexandrefonte$ docker container

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.
```

Playing with Docker Containers

- Commands related with Containers (Cont.)
  - **`docker container run`** or **`Docker run`** creates a Container
  - **`–docker container pause`** `<id container>`
  - **`–docker container unpause`** `<id container>`
  - **`–docker container inspect`** `<id container>`

alexandrefonte — -bash — 195×29

```
2022-04-04 13:08:53.000  INFO 1 --- [           main] c.i.r.w.r.RestfulWebServicesApplication  : Started RestfulWebServicesApplication in 10.087 seconds (JVM running for 11.156)
^CADF-iMac:~ alexandrefonte$ dockecontainer unpause b0e48SE
b0e48
ADF-iMac:~ alexandrefonte$ docker container inspect b0e48
[
    {
        "Id": "b0e4882e67c82bdbbe4a418bf8eab935b01276364a04c8b80e97df00c58392a7",
        "Created": "2022-04-04T13:08:41.3399175Z",
        "Path": "sh",
        "Args": [
            "-c",
            "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 3685,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2022-04-04T13:08:41.8255284Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:f8049a029560296f2c8d98a9668672ed9db1fc85ec0054f9fd9956ae79bf8827",
        "ResolvConfPath": "/var/lib/docker/containers/b0e4882e67c82bdbbe4a418bf8eab935b01276364a04c8b80e97df00c58392a7/resolv.conf",
        "HostnamePath": "/var/lib/docker/containers/b0e4882e67c82bdbbe4a418bf8eab935b01276364a04c8b80e97df00c58392a7/hostname",
```

Playing with Docker Containers

- Command related with Container (Cont.)
  - **docker container prune**

```
ADF-iMac:~ alexandrefonte$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
5b926cdff7a9a382fdaac6caeef23076dac9442e9742855c09ea854d8dbe3e91
ec4a6b13f5337cdc9db6b3aec4171323cfd8336a68f32d13487e104f13f4e43e
418900a331879f9c2397ee150ff498a9ace12b41b01b4892536260da763b02f1
b46a91a50e3529075ab5c499de806dc401ab2c1ba9e24b73b8a283ba1e40aeed

Total reclaimed space: 0B
ADF-iMac:~ alexandrefonte$
```

  - **docker container stop** => sigterm => Graceful shutdown

```
afs-MacBook-Air:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                       COMMAND                  CREATED          STATUS                          PORTS                    NAMES
d4e15c186c5a   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   16 minutes ago   Exited (143) 13 minutes ago                              gallant_dhawan
d035ec1af08d   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   17 minutes ago   Up 17 minutes                   0.0.0.0:8080->8080/tcp   strange_cannon
3d24f42c431e   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   25 minutes ago   Exited (130) 17 minutes ago                              zealous_northcutt
5a309ffd8f59   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   25 minutes ago   Created                                                  loving_tereshkova
afs-MacBook-Air:~ alexandrefonte$ docker container stop d035
d035
afs-MacBook-Air:~ alexandrefonte$
```

Playing with Docker Containers

- Command related with Container (Cont.)
  - **docker container kill**

```
afs-MacBook-Air:~ alexandrefonte$ docker container ls
CONTAINER ID   IMAGE               COMMAND                  CREATED          STATUS          PORTS                  NAMES
6cc5cbf98bd4   tutum/hello-world   "/bin/sh -c 'php-fpm…"   18 minutes ago   Up 18 minutes   0.0.0.0:80->80/tcp   hopeful_mcclintock
afs-MacBook-Air:~ alexandrefonte$ docker kill 6cc
6cc
afs-MacBook-Air:~ alexandrefonte$
```

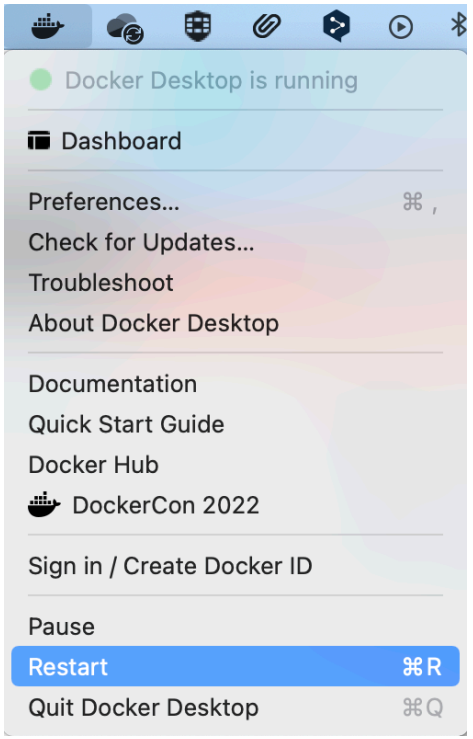
Playing with Docker Containers

Container is restarted if it stops. However, If you manually stop it, It is restarted only if docker daemon restarts

- Command related with Container
  - -By default, container do not restart automatically
  - **-docker container run – p …. --restart=always**

```
afs-MacBook-Air:~ alexandrefonte$ docker images
REPOSITORY          TAG       IMAGE ID       CREATED         SIZE
docker-aid          latest    81504b630f92   25 hours ago    578MB
adfonte/aid-mdssi   latest    437d42c0e475   33 hours ago    544MB
hello-aid-server    latest    437d42c0e475   33 hours ago    544MB
message-server      latest    d391b75136db   2 days ago      544MB
product-server      latest    809354f1ad6f   2 days ago      544MB
mysql               oracle    0ca4d64ca878   9 days ago      435MB
mysql               latest    667ee8fb158e   11 days ago     521MB
httpd               latest    118b6abfbf55   11 days ago     144MB
tutum/hello-world   latest    31e17b0746e4   6 years ago     17.8MB
afs-MacBook-Air:~ alexandrefonte$ docker container run -p8080:8080 adfonte/aid-mdssi:latest --restart=always

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.4)
```



More about restart policies:
https://docs.docker.com/config/containers/start-containers-automatically/

Playing with Docker Commands - stats, system

- **`docker events`**

```
ADF-iMac:~ alexandrefonte$ docker events
^CADF-iMac:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                                   COMMAND                  CREATED         STATUS         PORTS                    NAMES
845ce4281b49   in28min/todo-rest-api-h2:1.0.0.RELEASE   "sh -c 'java $JAVA_O…"   4 minutes ago   Up 3 minutes   0.0.0.0:5001->5000/tcp   frosty_grothendieck
ADF-iMac:~ alexandrefonte$ docker container stop 845
845
ADF-iMac:~ alexandrefonte$
```

alexandrefonte — com.docker.cli ◂ docker events — 80×24

```
Last login: Mon Apr  4 12:19:39 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
ADF-iMac:~ alexandrefonte$ docker events
2022-04-04T14:31:28.813280900+01:00 container kill 845ce4281b495223d892ad5e486c1
3007ad79ef6d055017bf9ea460a0e14f545 (image=in28min/todo-rest-api-h2:1.0.0.RELEAS
E, name=frosty_grothendieck, signal=15)
2022-04-04T14:31:29.137578500+01:00 container die 845ce4281b495223d892ad5e486c13
007ad79ef6d055017bf9ea460a0e14f545 (exitCode=143, image=in28min/todo-rest-api-h2
:1.0.0.RELEASE, name=frosty_grothendieck)
2022-04-04T14:31:29.205253600+01:00 network disconnect 83afcf1157f77edba5939c303
995c6cedc5f17258d4c500ac6c670f0e563727d (container=845ce4281b495223d892ad5e486c1
3007ad79ef6d055017bf9ea460a0e14f545, name=bridge, type=bridge)
2022-04-04T14:31:29.210857400+01:00 volume unmount 7b04ab6e1d573912ed22da0e0facf
ced39430fb39eee75707696bb7d5a98926d (container=845ce4281b495223d892ad5e486c13007
ad79ef6d055017bf9ea460a0e14f545, driver=local)
2022-04-04T14:31:29.222972200+01:00 container stop 845ce4281b495223d892ad5e486c1
3007ad79ef6d055017bf9ea460a0e14f545 (image=in28min/todo-rest-api-h2:1.0.0.RELEAS
E, name=frosty_grothendieck)
```

Playing with Docker Commands - stats, system

- **`docker container top <container id>`**

```
^Cafs-MacBook-Air:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                      COMMAND                  CREATED          STATUS                      PORTS                NAMES
16e24102a347   adfonte/aid-mdssi:latest   "java -jar hello-aid…"   7 minutes ago    Exited (130) 6 minutes ago                       naughty_bell
6cc5cbf98bd4   tutum/hello-world          "/bin/sh -c 'php-fpm…"   12 minutes ago   Up 12 minutes               0.0.0.0:80->80/tcp   hopeful_mcclintock
e4d02a3c2bcd   httpd:latest               "httpd-foreground"       25 minutes ago   Exited (0) 24 minutes ago                        zealous_ptolemy
f773db77544f   tutum/hello-world:latest   "/bin/sh -c 'php-fpm…"   26 minutes ago   Exited (0) 25 minutes ago                        peaceful_bose
afs-MacBook-Air:~ alexandrefonte$ docker top 6cc
UID                 PID                 PPID                C                   STIME               TTY                 TIME                CMD
root                3038                3010                0                   19:53               ?                   00:00:00            nginx: master process
f;
root                3072                3038                0                   19:53               ?                   00:00:00            php-fpm: master proce
m.conf)
nobody              3073                3072                0                   19:53               ?                   00:00:00            php-fpm: pool www
nobody              3074                3072                0                   19:53               ?                   00:00:00            php-fpm: pool www
root                3076                3038                0                   19:53               ?                   00:00:00            tail -F /var/log/ngin
root                3077                3038                0                   19:53               ?                   00:00:00            nginx: worker process
afs-MacBook-Air:~ alexandrefonte$
```

Playing with Docker Commands - stats, system

- **`docker container stats`** *`<container id>`*

```
[^Cafs-MacBook-Air:~ alexandrefonte$ docker container ls -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                      PORTS                    NAMES
16e24102a347   adfonte/aid-mdssi:latest       "java -jar hello-aid…"   7 minutes ago    Exited (130) 6 minutes ago                           naughty_bell
6cc5cbf98bd4   tutum/hello-world              "/bin/sh -c 'php-fpm…"   12 minutes ago   Up 12 minutes               0.0.0.0:80->80/tcp       hopeful_mcclintock
e4d02a3c2bcd   httpd:latest                   "httpd-foreground"       25 minutes ago   Exited (0) 24 minutes ago                            zealous_ptolemy
f773db77544f   tutum/hello-world:latest       "/bin/sh -c 'php-fpm…"   26 minutes ago   Exited (0) 25 minutes ago                            peaceful_bose
afs-MacBook-Air:~ alexandrefonte$ docker stats 6cc
```

alexandrefonte — com.docker.cli ◂ docker stats 6cc — 180×25

```
CONTAINER ID   NAME                CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O       PIDS
6cc5cbf98bd4   hopeful_mcclintock  0.03%     4.816MiB / 3.843GiB   0.12%     1.23kB / 0B   0B / 8.19kB     6
```

Limits to 512MB of RAM and 5% of CPU

```
docker run -p 80:80 -m 521m --cpu-quota 5000 -d --restart=always tutum/hello-world:latest
```

alexandrefonte — com.docker.cli ◂ docker stats 90bd — 180×25

```
CONTAINER ID   NAME          CPU %     MEM USAGE / LIMIT   MEM %     NET I/O     BLOCK I/O       PIDS
90bd09fbdc83   keen_wiles    0.04%     4.883MiB / 521MiB   0.94%     946B / 0B   0B / 8.19kB     6
```

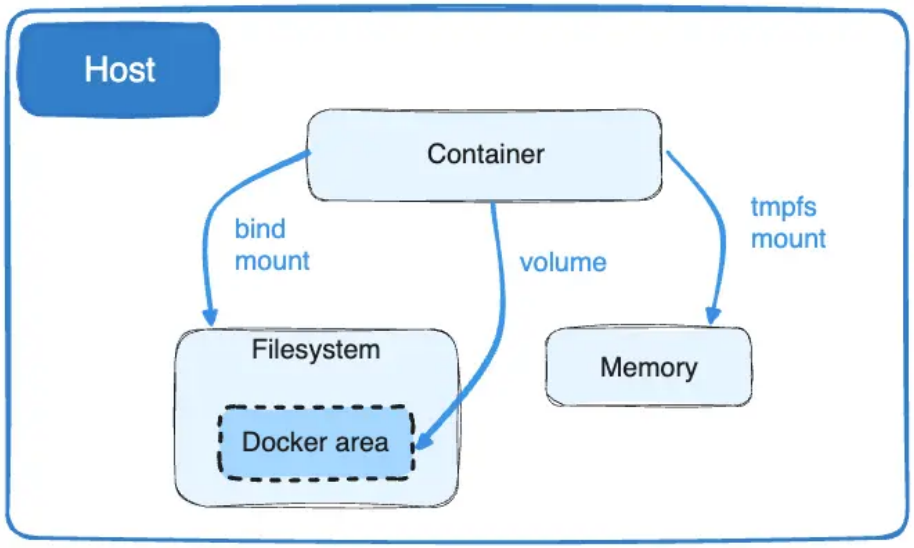
Playing with Docker Commands - stats, system

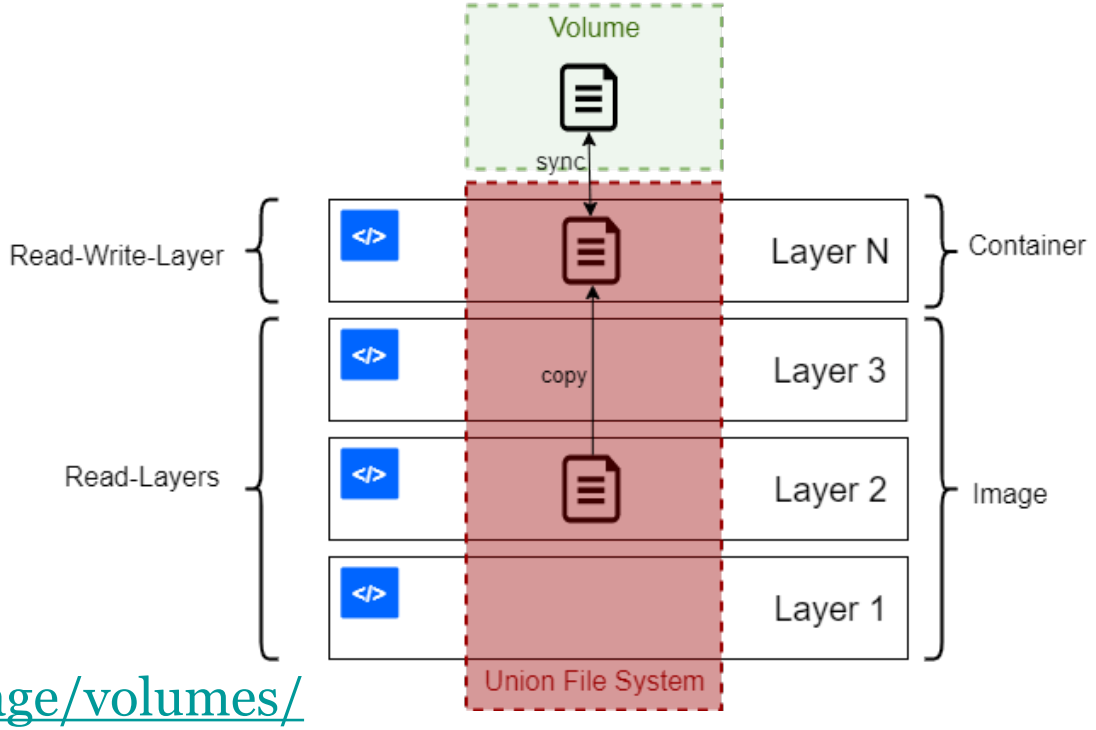
- **docker system df**
  - To show docker disk usage

```
ADF-iMac:~ alexandrefonte$ docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          2         1         663.7MB   520.7MB (78%)
Containers      4         1         0B        0B
Local Volumes   10        4         65.54kB   32.77kB (50%)
Build Cache     0         0         0B        0B
ADF-iMac:~ alexandrefonte$
```

Docker Volumes

- When we start a new container, Docker adds a read-write layer on the top of the image layers allowing the container to run as though on a standard Linux file system.
- **However, when the container is stopped or deleted, that read-write layer is lost.**

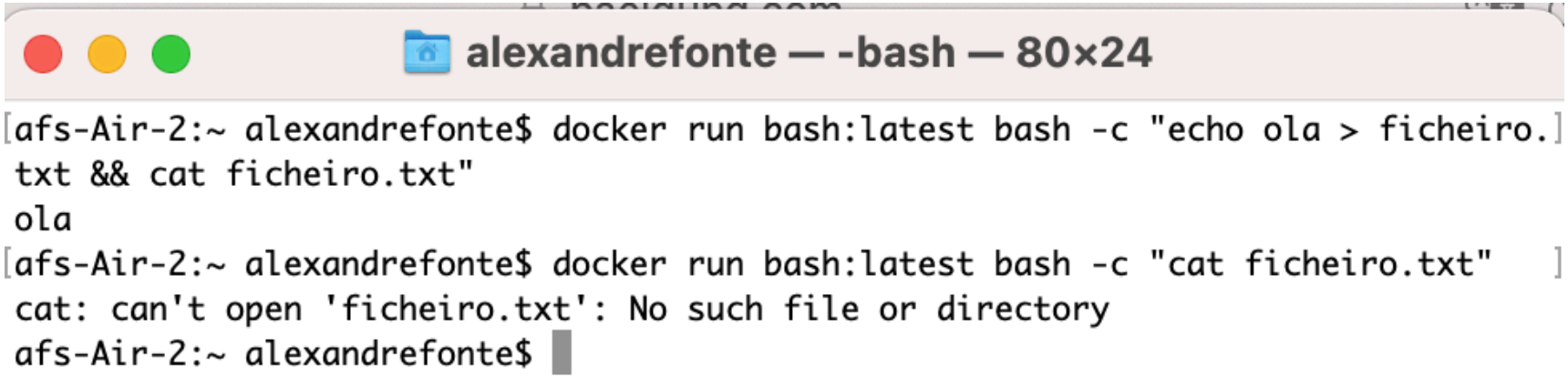




fonte1: https://docs.docker.com/storage/volumes/
fonte2: https://www.baeldung.com/ops/docker-volumes

Docker Volumes

- Exemplo - Ilustração do Problema



Docker Volumes - Solução criar um volume

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers.
- The volume itself has a lifecycle that's longer than the container's, allowing it to persist until no longer needed.
- Volumes can be shared between containers.

Docker Volumes - Comandos principais

- Criar um volume

```
$ docker volume create data_volume
data_volume
```

- Listar os volumes

```
$ docker volume ls
DRIVER              VOLUME NAME
local       data_volume
local   d7fb659f9b2f6c6fd7b2c796a47441fa77c8580a080e50fb0b1582c8f602ae2f
```

- Remover os volumes

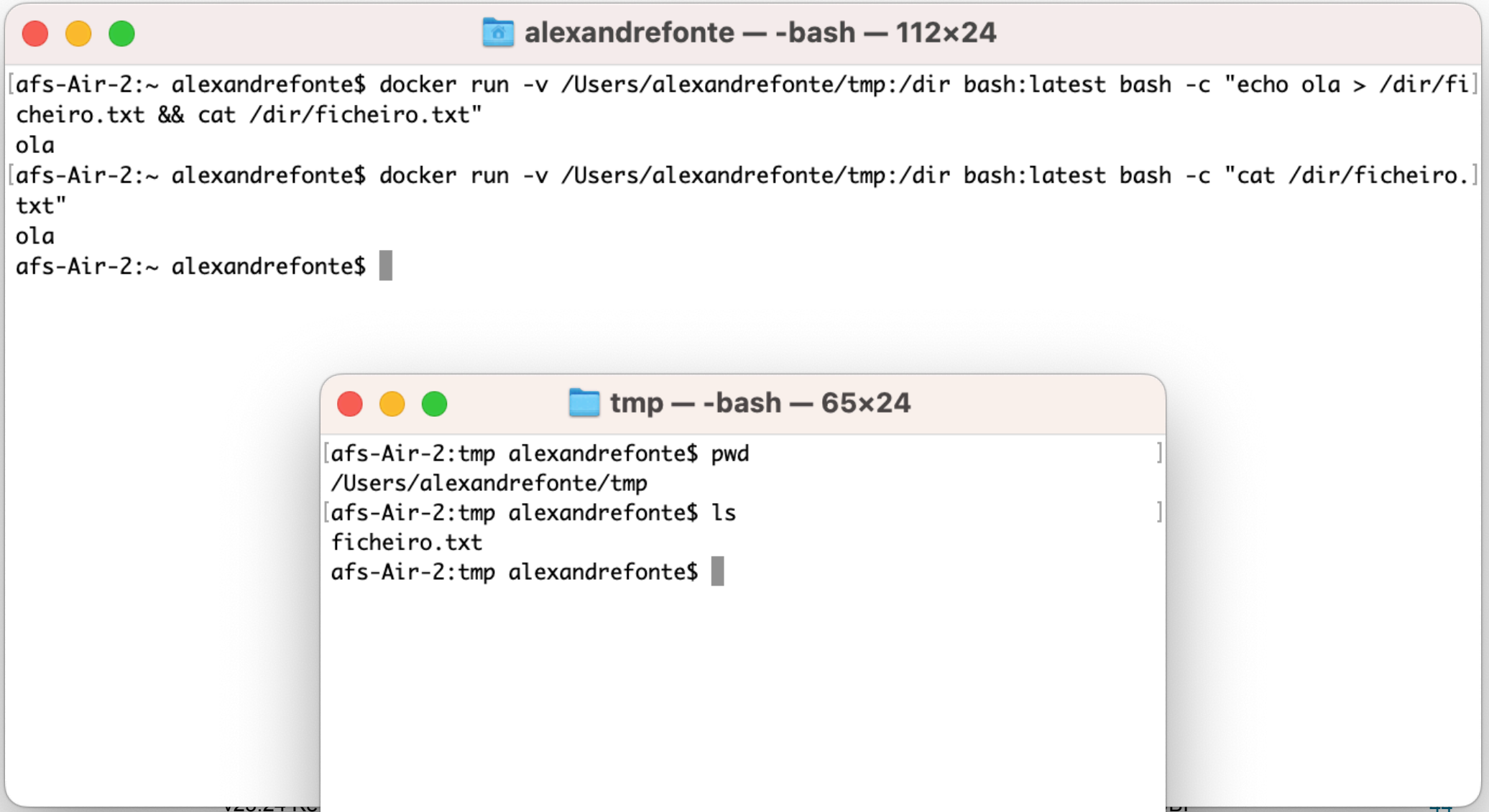
```
$ docker volume rm data_volume
data_volume
```

```
$ docker volume prune
WARNING! This will remove all local volumes not used by at least one container.
Are you sure you want to continue? [y/N] y
Deleted Volumes:
data_volume
```

- Correr um contentor com um volume usar -v (slide anterior)

Docker Volumes - Comandos principais

- Exemplo em que associa um diretório do sistema de ficheiros a um directório do contentor



How to Dockerize Applications
Example of Spring Boot Rest Applications

- **Demo:** Create two Spring Boot Rest Apps, two images and run containers.

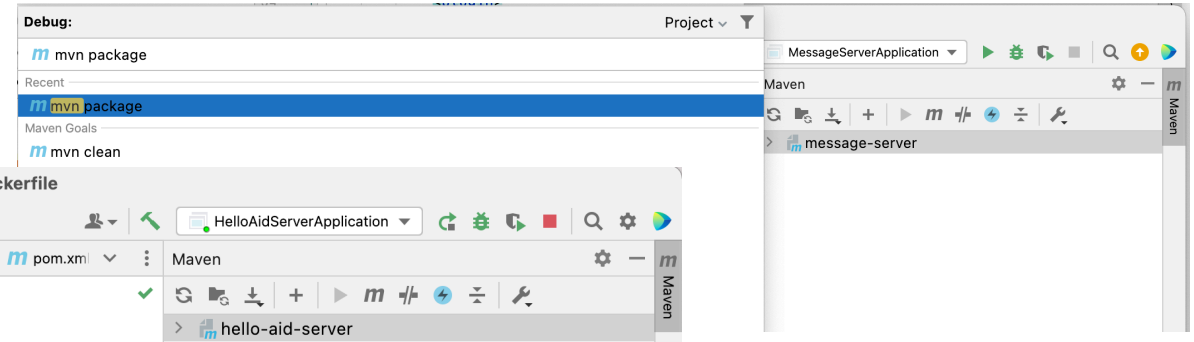
How to Dockerize Spring Boot Rest Applications

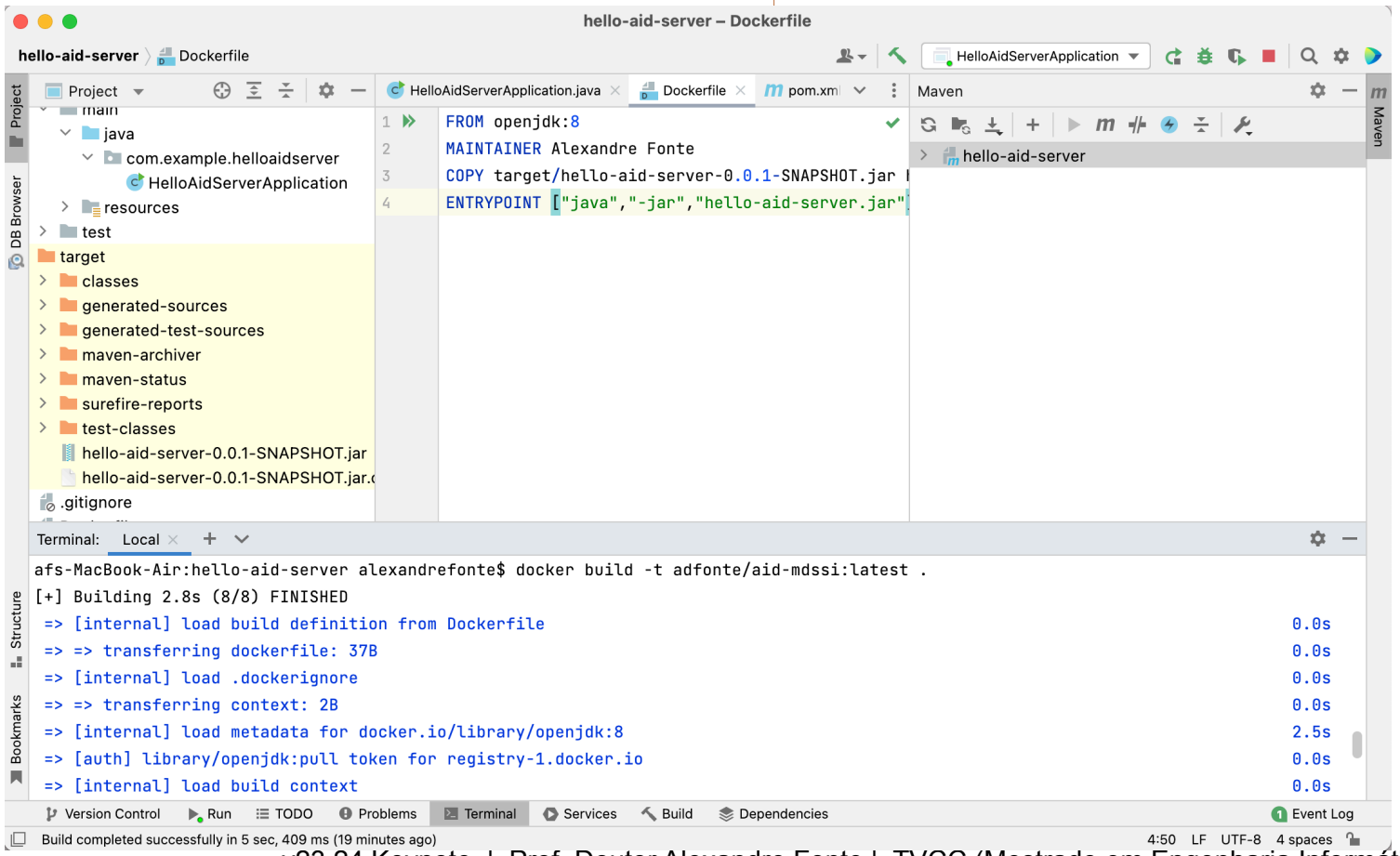
- **Step 1:** Create two Spring Boot Projects with Spring Boot Web Starter
  - -Artfact ID: message-server
  - -Artfact ID: product-server
- **Step 2:** Annotate main classes with @RestController and add a REST GET method that returns a friendly message such as:
  - -“A message from Docker!”
  - -"A brand new product”;
- **Step 3:** Configure message-server to use port 8080, and product-server to user 8081.
  - - Add to the application.properties:
    - •server.port=8080 or server.port=8081
- **Step 4:** Run and test if both Rest Services are running.

How to Dockerize Spring Boot Rest Applications

- **Step 5:** Stop both projects and Create their .Jar files for distribution, running the command:

  -man package





How to Dockerize Spring Boot Rest Applications

- **Step 6:** Create a Dockerfile similar to:

```
FROM openjdk:8
MAINTAINER Alexandre Fonte
COPY target/hello-server-0.0.1-SNAPSHOT.jar hello-server.jar
ENTRYPOINT ["java","-jar","hello-server.jar"]
```

This file contains the following information:

- **FROM**: As the base for our image, we'll take the Java-enabled Alpine Linux created in the previous section.
- **MAINTAINER**: The maintainer of the image.
- **COPY**: We let Docker copy our jar file into the image.
- **ENTRYPOINT**: This will be the executable to start when the container is booting. We must define them as JSON-Array because we'll use an ENTRYPOINT in combination with a CMD for some application arguments.

How to Dockerize Spring Boot Rest Applications

- **Step 7:** Open a terminal window on IntelliJ IDEA and build the project images given them a proper name and a TAG.

```
docker build -t message-server:latest .
```

- **Step 8:** Deploy/Run both images as daemons

How to Dockerize Spring Boot Rest Applications

- **Step 9 and following:** To Be continue …

Docker Compose

- Compose is a tool for defining and running multi-container Docker applications.
- Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file.
- With a single command, you create and start all the services from your configuration file.
- Using Compose is basically a three-step process:

  -Define your app's environment with a **`Dockerfile`** so it can be reproduced anywhere.

  -Define the services that make up your app in **`docker-compose.yml`** so they can be run together in an isolated environment.

  -Run **`docker compose up`** and the Docker compose command starts and runs your entire app.

Docker Compose - YAML file structure

- Compose allows us developers to easily handle multiple docker containers at once by applying many rules which are declared in a `compose.yml or docker-compose.yml` file.
- It consists of multiple layers that are split using tab stops or spaces instead of the braces we know in most programming languages.
- There are four main things almost every Compose-File should have which include:
  - The **version** of the compose file
  - The **services** which will be built
  - All used **volumes**
  - The **networks** which connect the different services

Docker Compose - YAML file structure

```yaml
version: '3.3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data: /var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: password
       MYSQL_DATABASE: db
       MYSQL_USER: user
       MYSQL_PASSWORD: password
     ports:
       - "3306:3306"
     expose:
       - "3306"
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: user
       WORDPRESS_DB_PASSWORD: password
       WORDPRESS_DB_NAME: db
     volumes:
       - wordpress:/var/www/html
   volumes:
     db_data:
     wordpress:
```

As you can see this file contains a whole Wordpress application including the MySQL database. Each of these services is treated as a separate container that can be swapped in and out when you need it.

Docker Compose - YAML file structure

- Services

  -**services** tag is the parent tag and contains all the containers

```
services:
  proxy:
    build: /path/to/dockerfile
  app:
    build:
  db:
    image: postgres
```

build: it builds the image from a dockerfile

- Images

  -The base image of a container can be defined by either using a preexisting image that is available on DockerHub or by building images using a Dockerfile.

```
version: '3.3'

services:
    alpine:
        image: alpine:latest
        stdin_open: true
        tty: true
        command: sh
```

```
version: '3.3'
services:
    app:
        container_name: website
        restart: always
        build: .
        ports:
            - '3000:3000'
    command:
        - 'npm run start'
```

Docker Compose - YAML file structure

- Ports
  - Exposing the ports in Compose works similarly as in the Dockerfile

Exposing ports without publishing them to the host machine:

-
```
expose:
 - "3000"
 - "8000"
```

Exposing the port to the host system:

```
ports:
  - "8000:80"  # host:container
```

Docker Compose - YAML file structure

- Volumes

```
version: '3.3'

services:
    alpine:
        image: alpine:latest
        stdin_open: true
        tty: true
        command: sh
        volumes:
          - my-named-global-volume:/my-volumes/named-global-volume
          - /tmp:/my-volumes/host-volume
          - /home:/my-volumes/readonly-host-volume:ro
    ...
    volumes:
  my-named-global-volume:
```

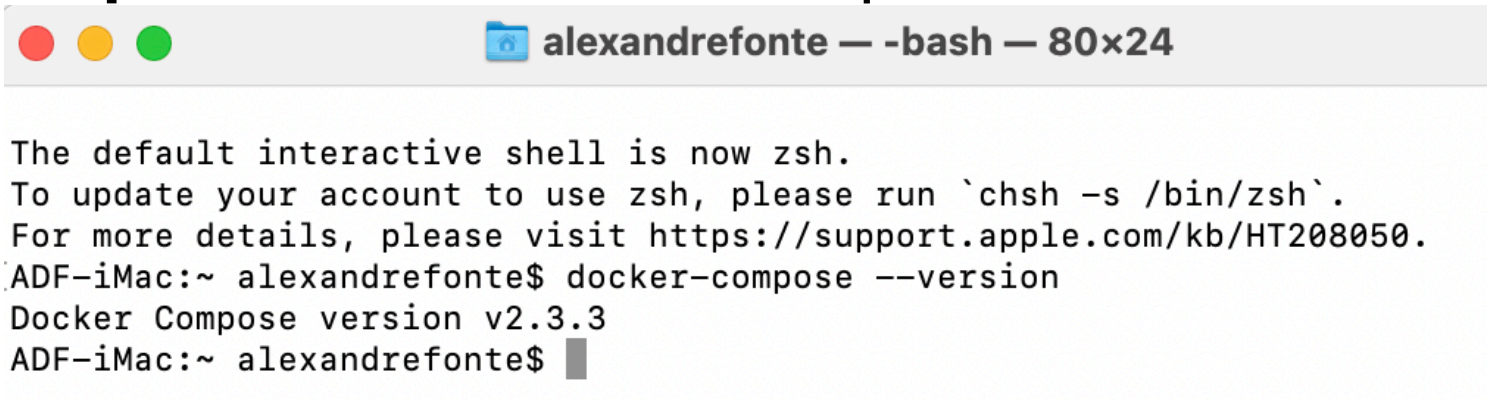
Here, container will have read/write access to the my-named-global-volume shared folder, no matter the different paths they've mapped it to.

The /tmp folder of the host's file system is mapped to the /my-volumes/host-volume folder of the container. This portion of the file system is writeable, which means that the container can not only read but also write (and delete) files in the host machine.

We can mount a volume in read-only mode by appending :ro to the rule, like for the /home folder (we don't want a Docker container erasing our users by mistake).

Playing with Docker Compose

- **Step 9:** Check Docker Compose version



- **Step 10:** Define `docker-compose.yaml` file similar to:

```
version: '2.0'
services:
  hello-server:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    ports:
      - “8080:8080"
  hello-server2:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    ports:
      - "8081:8080"
```

Playing with Docker Compose

- Run **`docker compose up -d`** to start your app

```
ADF-iMac:hello-aid-server alexandrefonte$ docker compose up -d
[+] Running 3/3
 ⠿ Network hello-aid-server_default                Created                                                    0.1s
 ⠿ Container hello-aid-server-hello-aid-server2-1  Started                                                    0.9s
 ⠿ Container hello-aid-server-hello-aid-server-1   Started                                                    1.0s
ADF-iMac:hello-aid-server alexandrefonte$ docker container ls
CONTAINER ID   IMAGE                       COMMAND                  CREATED         STATUS         PORTS                    NAMES
a6015bfcc104   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   4 seconds ago   Up 3 seconds   0.0.0.0:8080->8080/tcp   hello-aid-server-hello-aid-server-1
92968b1678f7   adfonte/aid-mdssi:latest    "java -jar hello-aid…"   4 seconds ago   Up 3 seconds   0.0.0.0:8081->8080/tcp   hello-aid-server-hello-aid-server2-1
ADF-iMac:hello-aid-server alexandrefonte$
```

- **Nota:** It is also created a virtual network by default
- Run **`docker compose down`** to stop your app

```
ADF-iMac:hello-aid-server alexandrefonte$ docker compose down
[+] Running 3/3
 ⠿ Container hello-aid-server-hello-aid-server-1   Removed                                                    0.6s
 ⠿ Container hello-aid-server-hello-aid-server2-1  Removed                                                    0.7s
 ⠿ Network hello-aid-server_default                Removed                                                    0.2s
ADF-iMac:hello-aid-server alexandrefonte$ docker container ls
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ADF-iMac:hello-aid-server alexandrefonte$
```

Playing with Docker Compose

- **Step 11:** Update de **`docker-compose.yaml`** file and run again **`docker compose up -d`** to start your app

```
version: '2.0'
services:
  hello-server:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    networks:
      - frontend
    ports:
      - "8080:8080"
  hello-server2:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    networks:
      - backend
    ports:
      - "8081:8080"
networks:
  frontend:
  backend:
```

Playing with Docker Compose

- **Step 12:** Update de **`docker-compose.yaml`** file and run again **`docker compose up -d`** to start your app
  - -Add a IP management section to the configuration of network
  - -Attribute a static address to 1st container by adding a ipv4_address section.
  - -Finally enable a Ipv6 configuration

```
version: '2.0'
services:
  hello-server:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    networks:
      frontend:
        ipv4_address: 172.16.1.10
    ports:
      - "8080:8080"
  hello-server2:
    image: adfonte/hello-server:latest
    mem_limit: 700m
    networks:
      - backend
    ports:
      - "8081:8080"
networks:
  frontend:
    ipam:
      config:
        - subnet: 172.16.1.0/24
          gateway: 172.16.1.254
  backend:
    ipam:
      config:
        - subnet: 172.16.2.0/24
          gateway: 172.16.2.254
```

Playing with Docker Compose

- Docker Networking

  -Docker networking is used to establish communication between Docker containers and the outside world via the host machine.

```
afs-MacBook-Air:~ alexandrefonte$ docker network ls
NETWORK ID     NAME                          DRIVER    SCOPE
887eda5510b6   bridge                        bridge    local
27202f2c150b   hello-server-backend           bridge    local
de5c44dbf7ca   hello-server-frontend          bridge    local
8cdef0ac9afc   host                          host      local
f282e351e8b3   none                          null      local
```

  -Docker supports different types of drivers:

  - bridge: the default network used when applications need to communicate.
  - host: uses the host's networking directly (its IP and ports). Don't get its own IP address. It removes network isolation between the container and the host machine where Docker is running !
  - None: disable all networking.

Playing with Docker Compose

- Docker Networking
  - **docker network inspect bridge**

```
ADF-iMac:hello-aid-server alexandrefonte$ docker network inspect de5
[
    {
        "Name": “hello-tvcc-server_frontend",
        "Id":
"de5c44dbf7ca2e2e2363add4520b16ef8aea350e5e3494f8d30dce8e26bfb167",
        "Created": "2022-04-23T10:45:24.481300612Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.16.1.0/24"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "4d903d10f44c742e517bb48742cda0c4ead70e50d64ccb8bbbea89d9855442f8":
{
                "Name": "hello-server-hello-server-1",
                "EndpointID": re Fonte | TVCC (Mestrado em Engenharia Informática), @UBI
```

Playing with Docker Compose

- Docker Networking (summary of main commands)

-docker container run --network host …

•To use host driver

-docker network inspect

•To inspect details of networks

-docker network disconnect *mynetwork* 0f8d7a833f42

•Disconnecting a Container with a given Id from the Network mynetwork

-docker network ls

•List Available Networks

-docker rm *network id*

•Removing a Network

-docker run -p 8080:8080 *image id*

•Public Networking

Questões