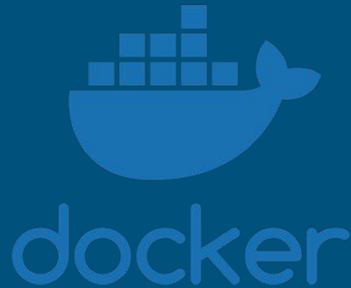


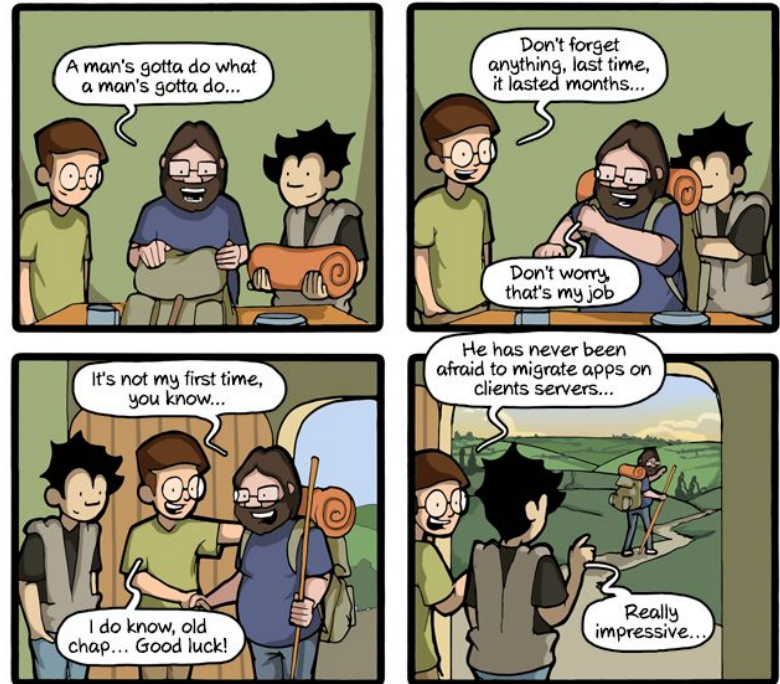
Lesson 06

Containerization



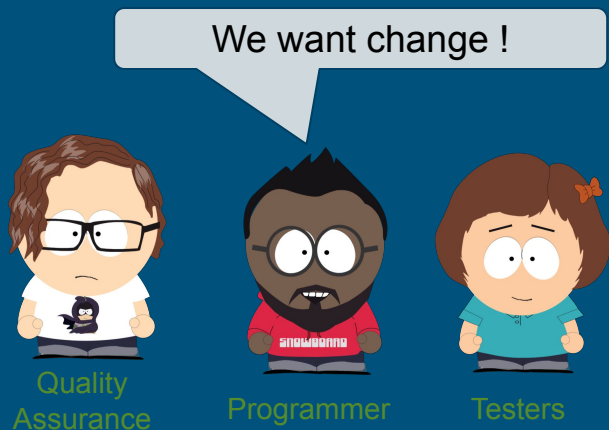
DEVOPS - ITI 4
HEI 2021-2022

Why use container ?

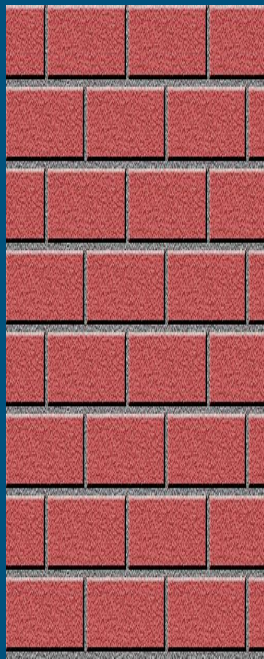


CommitStrip.com

Previously, in Introduction Lesson



DEV



Wall of Confusion



OPS

Case study in IT

DEV : Hello :-)

DEV : I need you to install on the servers
the latest version of the latest framework
that seems too good !

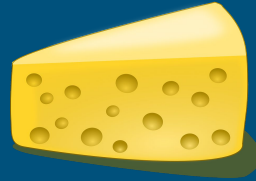
DEV : :(:(:(

OPS : Hi :-)

OPS : Ok, I check if it's possible

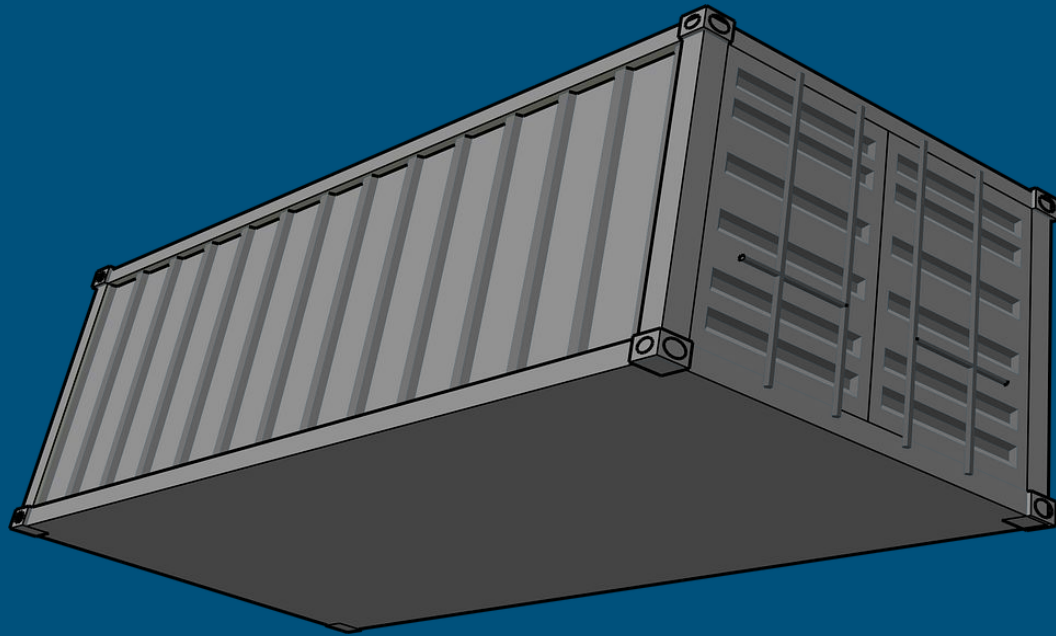
OPS : Sorry, I can't, it's not in our
standards :/

Case study in Logistic Sector



How to transport different resources with same vehicles ?

Solution



Containerization

Container

- From an external point of view : Standardized element
- From an internal point of view : Contains everything you want

Containerization in IT



What is docker ?



Overview

In a few words

- Containerization System
- Free software
- Created by Solomon Hykes in 2013

Usage

- Sandbox
- Development and tests
- Production deployment

Key concepts

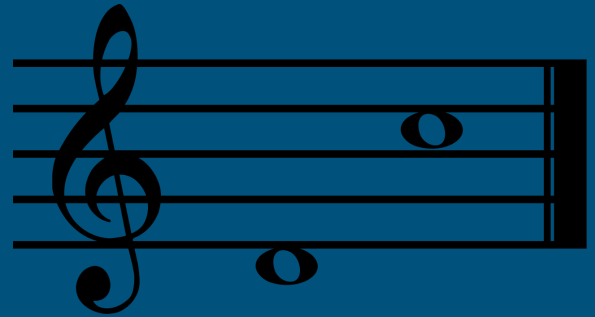
Image

- Template file
- Snapshot / Fixed point
- Layered structure

Container

- Instantiates an image
- Runs / Constantly evolves
- Interacts with other elements (host, others containers, web ...)

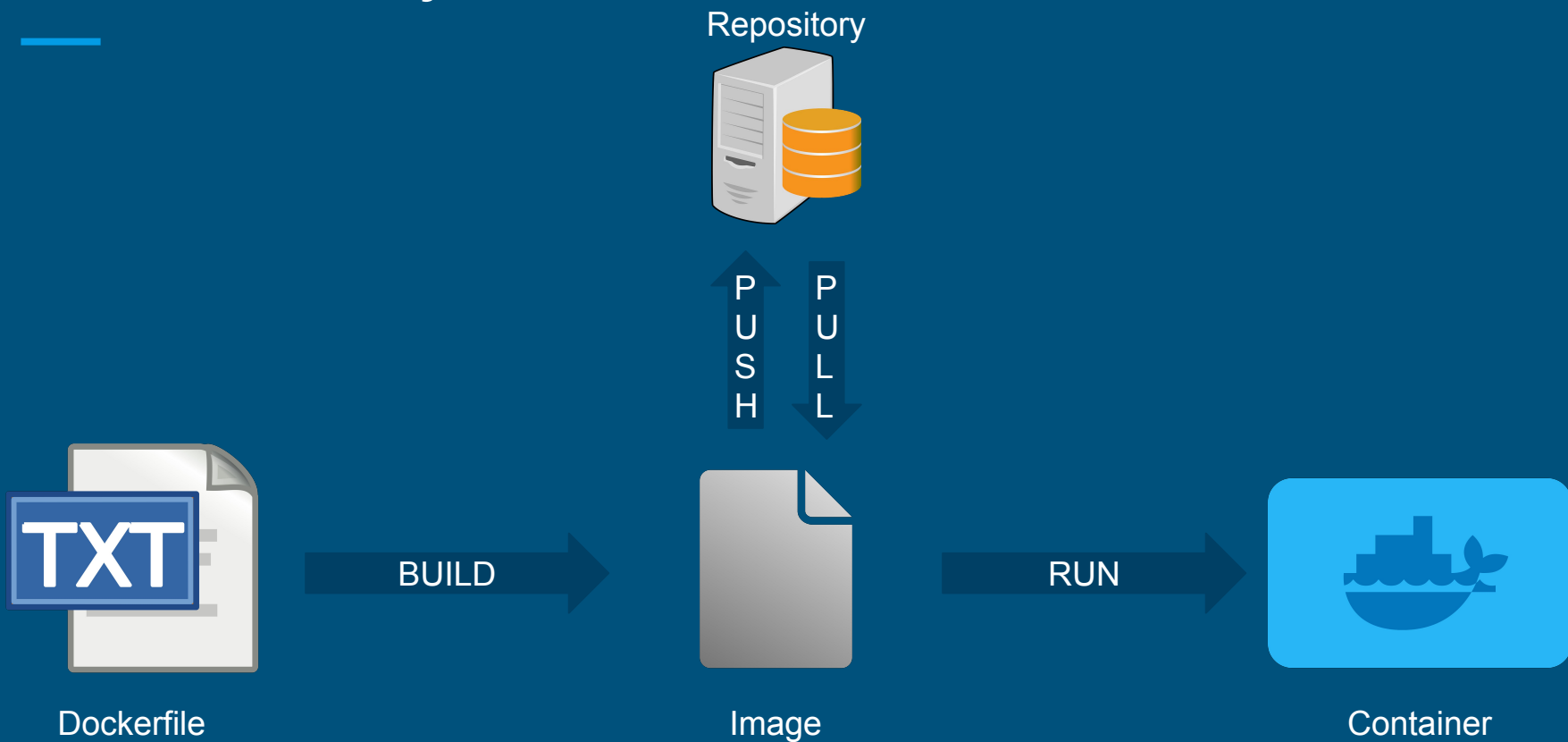
A concert is a
container whose
image is the score



An object is a
container whose
image is the class



Docker lifecycle



How to use Docker images ?



Container life cycle

docker pull

- *docker pull* pulls/downloads an image from a repository

```
$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cab9fde470971e499788
Status: Downloaded newer image for hello-world:latest
```

docker create

- *docker create* creates a new container from an image

```
$ docker create hello-world  
f780a721a66fffc6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0
```

Container ID

docker start

- *docker start* starts an existing container
 - *-i* (*--interactif*) : Print container STDIN in console

Container ID

```
$ docker start f780a721a66ffcd6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0  
f780a721a66ffcd6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0
```

```
$ docker start -i f780a721a66ffcd6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0
```

```
Hello from Docker!
```

docker run



- *docker run* pull, create and start an image
 - `-i` (`--interactif`) : Print container STDIN in console
 - `-t` (`--tty`) : Create a pseudo terminal in container
 - `-d` (`--detach`) : Run container in background
 - `--name` : Assign a name to the container
 - `--rm` : Automatically remove the container when it exits

```
$ docker run -ti centos  
[root@4a38d0b048fc /]#
```



console in the
container

```
$ docker run -d centos  
f83beda8fa970986a82b71534eabe9fe68a4ec01ff38c9cbec0f87317a9256c9  
  
$
```



console out
the container

docker stop

- *docker stop* stops a running container

Container ID

```
$ docker stop f780a721a66fffc6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0  
f780a721a66fffc6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0
```

docker rm

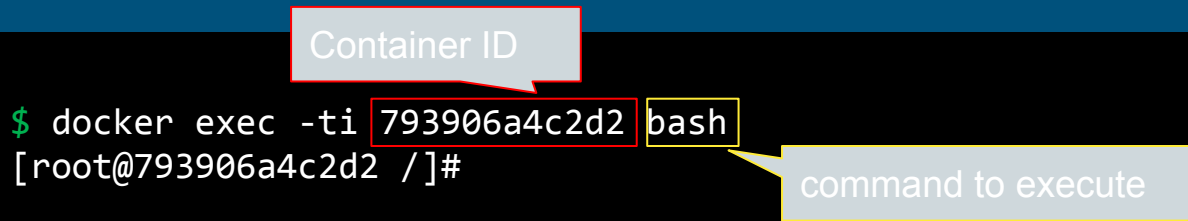
- *docker rm* removes an existing container

Container ID

```
$ docker rm f780a721a66fffc6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0  
f780a721a66fffc6752d54574b54da59bec39c301f0052ecbd18eb6e8ac08d0
```

docker exec

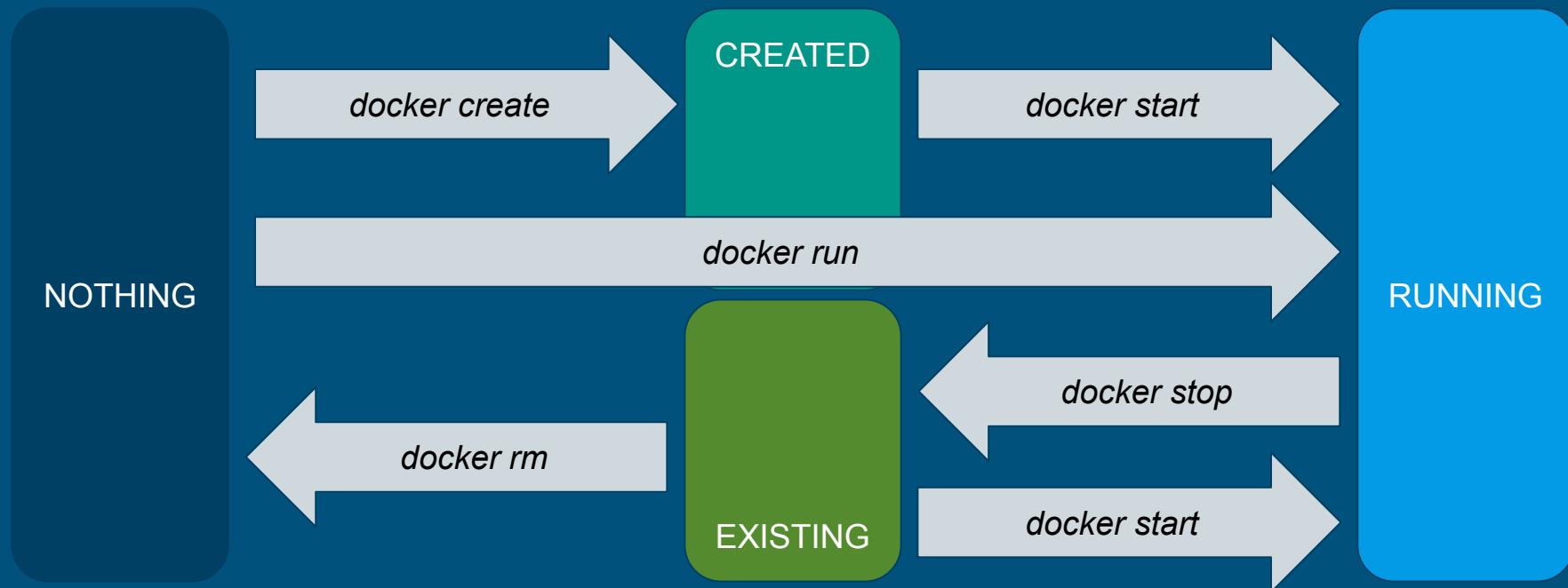
- *docker exec* executes a command in a running container



A terminal window with a black background and white text. The command `$ docker exec -ti 793906a4c2d2 bash` is entered. A red box highlights the container ID `793906a4c2d2`, with a callout box labeled "Container ID" pointing to it. A yellow box highlights the command `bash`, with a callout box labeled "command to execute" pointing to it. The prompt `[root@793906a4c2d2 /]#` is shown on the next line.

```
$ docker exec -ti 793906a4c2d2 bash
[root@793906a4c2d2 /]#
```

Summary



State of the container

docker ps



- *docker ps* list running containers
 - -a (--all) : Show all containers
 - -q (--quiet) : Only display numerics IDS

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
793906a4c2d2   centos    "/bin/bash"             3 hours ago   Up 3 hours                   festive_poitras
```

```
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
6c76d7f80c6a   centos    "/bin/bash"             3 hours ago   Exited (127) 3 hours ago   flamboyant_swartz
793906a4c2d2   centos    "/bin/bash"             3 hours ago   Up 3 hours                   festive_poitras
1c58a888fc94   centos    "/bin/bash"             3 hours ago   Exited (0) 3 hours ago   serene_hoover
72170e3b2c01   centos    "/bin/bash"             3 hours ago   Exited (0) 3 hours ago   compassionate_joliot
e248e53d6c3b   hello-world  "/hello"                3 hours ago   Exited (0) 3 hours ago   keen_jennings
d5ee93590282   hello-world  "bash"                  3 hours ago   Created                          clever_benz
```

docker logs

- *docker logs* fetches the logs of a container

```
$ docker logs e248e53d6c3b
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

docker inspect

- *docker inspect* displays low-level information on an container

```
$ docker inspect e248e53d6c3b
[
  {
    "Id": "e248e53d6c3b41ba240e2f785625f6e6b78d272a9785f786e43bb879befddde1",
    "Created": "2018-10-01T16:02:54.8629553Z",
    "Path": "/hello",
    "Args": [],
    "State": {
      "Status": "exited",
      "Running": false,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 0,
      "ExitCode": 0,
```


docker stats

- *docker stats* displays a live stream of containers resource usage statistics

```
$ docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
3aa414f4c1c9	jee-courses	0.00%	288.6MiB / 1.75GiB	16.11%	106MB / 234MB	78.7MB / 0B	29
8dc0b7793925	middle-devweb	0.00%	159.4MiB / 1.75GiB	8.90%	170MB / 437MB	19.7MB / 287kB	42
01a81525d957	middle-devops	0.00%	158.6MiB / 1.75GiB	8.85%	1.61MB / 36.8MB	98.2MB / 287kB	42
3348a1225f4a	front-all	0.00%	18.11MiB / 1.75GiB	1.01%	2.02GB / 1.96GB	9.18MB / 0B	109
7c345a7404fc	back-all	0.00%	90.13MiB / 1.75GiB	5.03%	227MB / 275MB	63.2MB / 235MB	30

How to create Docker images ?



Dockerfile

Overview

- Text file
- Contains instructions to create an image

FROM

- *FROM image:version*
- Initializes image creation
- Must be first instruction in a dockerfile and appear only once
- Pull an image in a repository

```
FROM centos:latest
```

```
FROM java:latest
```

LABEL

- *LABEL key=value*
- Adds metadata to an image

```
LABEL maintainer="denis.cauchois@junia.com"
```

```
LABEL description="my first image"
```

```
LABEL version="1.0"
```

ARG

- *ARG variable*
- Defines a variable whose value will be given during the build of the image

```
ARG maintainer
```

```
LABEL maintainer=${maintainer}
```

ENV

- *ENV variable value*
- Defines a variable whose value will be given during the run of the container

```
ENV version 1.0.0
```


COPY

- *COPY source destination*
- Copies files or directories from source (host) to destination (image)

```
COPY ./src/bin/startup /bin/ #Copy file startup in /bin/
```

```
COPY ./src/conf/* /etc/httpd/conf/ #Copy all files in conf in /etc/httpd/conf/
```

RUN

- *RUN command with arguments*
- Executes any commands during the build of the image

```
#Install java
```

```
RUN yum install -y java
```

```
#Download hello-world jar
```

```
RUN curl https://search.maven.org/remotecontent?filepath=io/elasticsearch/hello-world/0.0.1/hello-world-0.0.1.jar -o helloworld.jar
```

CMD

- *CMD ["executable","param1",...]*
- *CMD ["param1",...]*
- *CMD command param1 ...*
- Must appear only once
- Provides defaults values or command to execute container

```
CMD ["/bin/echo", "Hello World"]
```

```
$ docker run -ti my-container  
Hello World
```

```
$ docker run -ti my-container bash  
[root@793906a4c2d2 /]#
```

overrides CMD

ENTRYPOINT

- *ENTRYPOINT* ["command","param1",...]
- Must appear only once
- Allows you to configure a container that will run as an executable

```
ENTRYPOINT ["/bin/echo", "Hello"]  
CMD ["World"]
```

```
$ docker run -ti my-echo  
Hello World
```

```
$ docker run -ti my-echo Denis  
Hello Denis
```

overrides CMD

USER

- *USER user*
- Sets the user to use when running the container and to the execute *RUN*, *CMD* and *ENTRYPOINT*

```
USER www
```

HEALTHCHECK

- *HEALTHCHECK CMD command*
- Check container still still working as expected

```
HEALTHCHECK CMD /bin/checkApplicationIsUp
```

Image's build



docker images

- *docker images* list all images

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-hello-world	latest	f0cb6bc46c95	5 minutes ago	643MB
centos	latest	5182e96772bf	2 months ago	200MB
java	8	d23bdf5b1b1b	20 months ago	643MB

docker build



- *docker build* builds an image from a Dockerfile
 - `--build-arg` : Set arguments values
 - `-t` (`--tag`) : Name and optionally a tag in the 'name:tag' format

```
$ docker build -t my-hello-world .  
Sending build context to Docker daemon 23.55kB  
.....  
Successfully built f0cb6bc46c95  
Successfully tagged my-hello-world:latest
```

A lot of lines ...

How to take control with docker ?



Docker Network

Overview



Container 1



Container 2



Container 3

Docker Area

HOST

Type

	Communication with other containers in the Docker Area	Communication with host
<i>none</i>	Forbidden	Forbidden
<i>bridge</i> <u>(default)</u>	Authorized	Forbidden
<i>host</i>	Authorized	Authorized
customized	Only with the containers in the same network	Forbidden

Command to use network in a container

- during create / run step of the container
- `--network` links container to the network

```
$ docker create --network my-network --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

- `--expose` exposes container's port in the network

```
$ docker create --expose 8080 --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

- `-p` (`--publish`) exposes container's port to the host

```
$ docker create -p 8080:8081 --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

host port

container port

Command to manage network

- `docker network create` creates a new network (with driver bridge)

```
$ docker network create my-network
251d26612f6f0c84568a8c32109bdd4bf55490f4223d4bca5dc6cee92a0624c2
```

- `docker network ls` lists current network

```
$ docker network ls
```

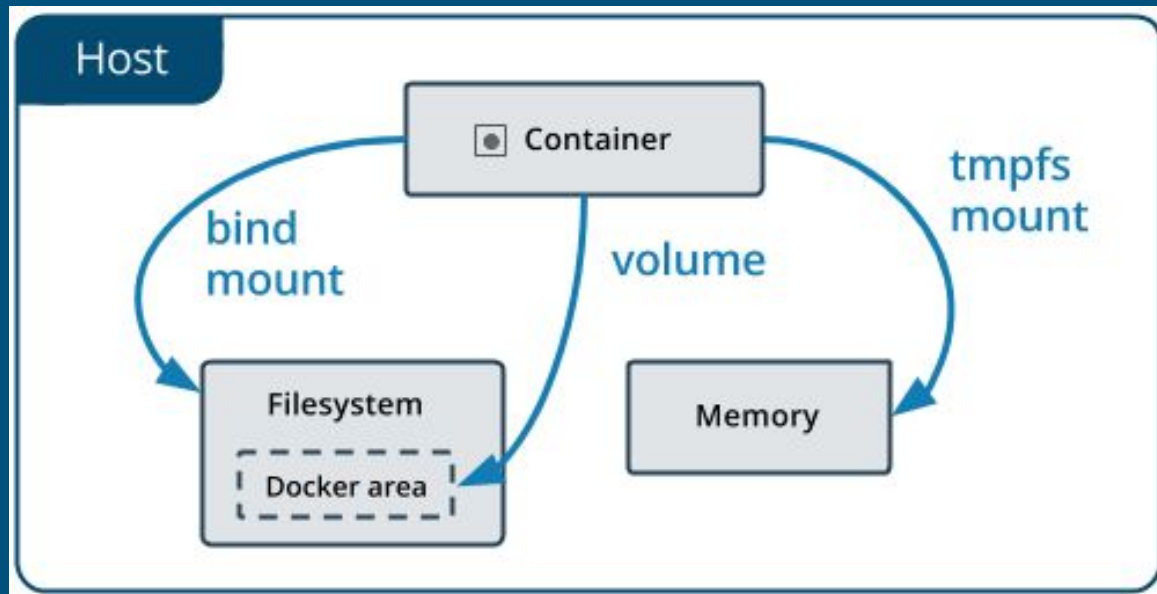
NETWORK ID	NAME	DRIVER	SCOPE
4deaa1549b43	bridge	bridge	local
e90b32784055	host	host	local
251d26612f6f	my-network	bridge	local
ad2f71169c8e	none	null	local

- `docker network rm` deletes an existing network

```
$ docker network rm my-network
my network
```

Docker File System

Overview



Command to use storage in a container

- during create / run step of the container
- `-v` (`--volume`) mounts a volume :

- Volume bind to the host

host directory

container directory

```
$ docker create -v /var/data:/data --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

- Volume bind to the Docker Area

volume in Docker Area

container directory

```
$ docker create -v my-volume:/data --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

- `--tmpfs` mounts a temporary volume in memory

container directory

```
$ docker create --tmpfs /tmp --name centos centos  
97d5089e7af2bfe1d35dbfeb1b5d7158738e8f533726aebb4d897ad116df7e87
```

Command to manage volume in Docker Area

- *docker volume create* creates a volume

```
$ docker volume create my-volume
251d26612f6f0c84568a8c32109bdd4bf55490f4223d4bca5dc6cee92a0624c2
```

- *docker volume ls* lists volumes

```
$ docker volume ls
DRIVER      VOLUME NAME
local       my-volume
```

- *docker volume rm* deletes a volume

```
$ docker volume rm my-volume
my volume
```

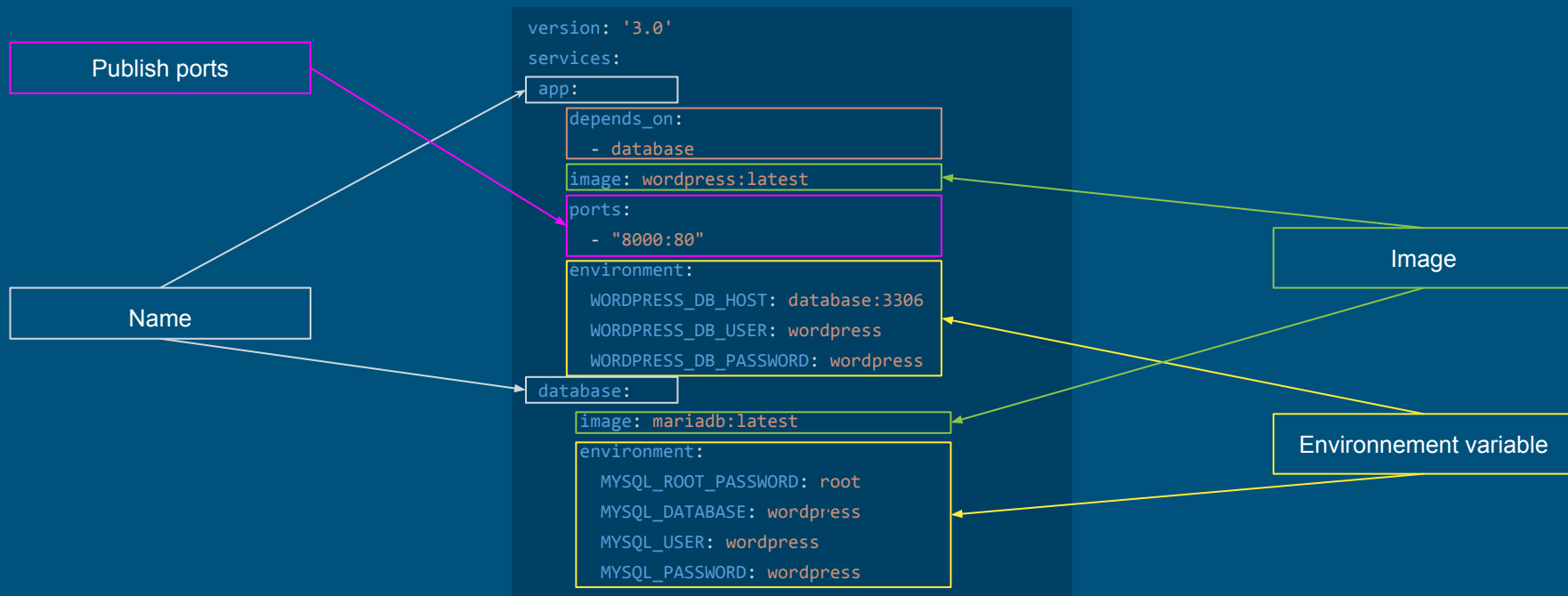
Docker compose



Overview

- Tool for defining and running multi-container Docker applications
- Configured by a YAML file

Example : docker-compose.yml



Command to manage docker-compose

- *docker-compose up -d* starts services

```
$ docker-compose up -d
Starting lesson-04_database_1 ... done
Starting lesson-04_app_1      ... done
```

- *docker-compose ps* lists services status

```
$ docker-compose ps
```

Name	Command	State	Ports
lesson-04_app_1	docker-entrypoint.sh apache2 ...	Up	0.0.0.0:8000->80/tcp
lesson-04_database_1	docker-entrypoint.sh mysqld	Up	3306/tcp

- *docker-compose stop* stops services

```
$ docker-compose stop
Stopping lesson-04_database_1 ... done
Stopping lesson-04_app_1      ... done
```

Question

What will happen?

```
FROM alpine:latest  
ENTRYPOINT ["echo","Hello"]  
CMD ["World"]
```

Dockerfile

```
$ docker build -t hello .  
  
$ docker run -ti hello Guys
```

Terminal

Thank you for your attention !



Links :

- Sources

- <http://www.commitstrip.com/>
- <https://giphy.com/>
- <https://docs.docker.com/>
- https://www.youtube.com/channel/UCOAhkxpryr_BKybt9wlw-NQ

- Examples :

- <https://gitlab.com/hei-devops/lesson/lesson-2020/lesson-06>
- <https://labs.play-with-docker.com/>