Docker

Jérôme Gurhem - February 2023

Contents

The origin of Docker

Expectations

- Ease of use
- Portability
- Reproducibility
- Isolated environment
- Version control

Portability

Can easily run on multiple computer environment

- Abstraction layer between the operating system and the application
- Ability to run everywhere without modifications in your application
- Ease of use

Reproductibility

The results obtained by an application should be achieved again reliably

- Create a snapshot of an **application** and its **exact** dependencies
- Allows to re-execute snaphots on other platforms
- Avoid to install the whole stack locally to run an application
- Especially in the case of complex dependencies
- Facilitate application debugging
- Reduce time to reproduce and fix errors

Isolated environment

Applications only access what they need

- Environment isolation (users, files, networking)
- Snapshots are isolated from each other and the host
- Provides multiple isolated environments on a single host
- Each environment can have its own IP
- Multiple Linux distributions can run on the same host
- Application can access all the required resources (network, file system) in an isolated way

Version control

Track and manage changes of an application and its dependencies

- Version an applications and its exact installation
- Easily refer to versions
- Faciliate rolling back to a previous version
- Alleviate the cost of updating an application and its dependencies

Security

Protect hardware and data from threats

- Separate individual components from a larger application
- Isolation makes sure that an application cannot access the data of another application
- You cannot rely only on an external tool for security !

Note

Isolation does not provide encryption nor authentication for your application !

First solution : bare metal machines

Applications are installed on a server

- Portability
  - ❌ Not easy to install an application on another system
- Reproducibility
  - ❌ No control over dependencies therefore unwanted behavior can occur
- Isolated environment
  - ❌ Every application run on the same machine
- Version control
  - ❌ Not possible to version a server

Second solution : virtualization

Applications are managed through virtualization and hypervisors

- Portability
  - ✔ It is the responsibility of the hypervisor to run the VM
- Reproducibility
  - ✔ VMs can be saved and copied
- Isolated environment
  - ✔ Each VMs are isolated
- Version control
  - ✔ VM snapshots can be versioned

Hypervisor

A software that creates, run and manage virtual machines on a computer.

The issue

Everything is great ! Right ?

- ❌ Performances are terrible due to the emulation of the whole guest operating system by the hypervisor
- ❌ The images are huge

Third solution : containerization

Use cases

Why do we use containers ?

- Portability ✔
- Reproducibility ✔
- Isolated environment ✔
- Version control ✔
- Lightweight ✔

When do we use containers ?

- Development, to create the same and pre-configured environment for all developers
- Production, to run applications on a server

What is a container ?

Learn More

- Software package
  - Standardized
  - Provides isolation to run an application with its dependencies
  - Have the ability to interact with the host and other containers
- Applications and their dependencies are completely abstracted from other containers and the host
  - Each container has its own environment

What is an image ?

Learn More

- Lightweight, stand-alone and executable software package that contains everything an application needs to run
  - A container is a running instance of an image
- Contains an application, its dependencies (runtime and librairies) and all the tools they need
  - Blueprint to create a container

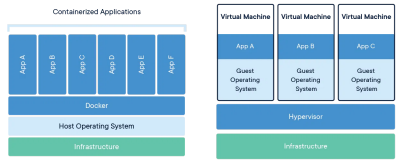
Image vs Container

- Image -> Application files + dependencies
- Container -> Running application

Containerization vs Virtualization

- Containerization abstract applications
- Containers share the host kernel with binaries and libs that are needed
- No emulation
- Small size and overhead

- Virtualizatio abstract hardware
- Each VM h stem and its dependenc
- Hardware is emulated by the hypervisor
- Large size and overhead



Other interests for Docker

- Docker project and container format are Open-Source
- Heavy use of containers in the industry
- Lots of cloud services are based on containers
- Micro-services architectures
- Containers orchestration

Containers orchestration

Automation of the operational effort to run containerized applications. (ex: Docker Compose, Docker Swarm, AWS ECS, Kubernetes, …)

Docker drawbacks

- Lack of cross-platform support (Windows containers cannot run on Linux)
- Poor support for graphical interfaces (Docker was designed for command line applications)

Alternatives

Podman, Apptainer/Singularity

Note

VM support for graphical interfaces is great.

Commands


boot2docker-vm [Ru
/boot2docker/tls/ca.pem --ca-key=/var/lib/b
/lib/boot2docker/tls/server.pem --key=/var/l
rg=Boot2Docker
2015/12/25 21:41:55 Generating a server cert
Generating client cert
2015/12/25 21:41:57 no --host parameters, ma
Finished boot2docker init script...
login[1975]: root login on 'tty1'
Boot2Docker version 1.9.1, build master : ce
Docker version 1.9.1, build a34a1d5
root@boot2docker:~# $_

Usual commands

Learn More

- How to build an image ?
- How to run a container ?

```
# Getting help !
docker COMMAND --help

# Example :
docker run --help
docker ps --help
docker images --help
```

Push / Pull docker images

Learn More

```
docker pull image:tag
docker push image:tag
```

- Pull or push the image from/to the remote registry
- Tag is not necessary

```
# Example:
docker pull ubuntu:22.04
```

Note

Pushing an image with an existing tag will override it.

Ex 1: Pull docker images

The fist docker command

1. Pull the `hello-world` image
2. Pull the `ubuntu` image
3. Pull the `fedora` image
4. Pull the `ubuntu` image with the version 22.O4

List local images

Learn More

```
docker images
```

```
$ docker images
REPOSITORY                                TAG                 IMAGE ID       CREATED         SIZE
dockerhubaneo/armonik_pollingagent        test                b352dc9ebb5d   4 days ago      227MB
<none>                                    <none>              1f5ed6475c91   4 days ago      1.72GB
dockerhubaneo/armonik_control             test                aed0776a823d   4 days ago      227MB
<none>                                    <none>              6bce402dd842   4 days ago      1.63GB
```

Delete local images

Learn More

- remove an image

```
docker rmi image:tag
```

- remove all image with tag `<none>` (dangling images)

```
docker rmi -f $(docker images -a -q -f dangling=true)
# or
docker image prune
```

- remove all unused images, all unused containers (running or stopped) and all unused networks

```
docker system prune -a
```

Note

How to filter images for `docker images`.

Ex 2: Remove local images

Some cleanup is necessary !

1. List local images
2. What do you observe for the ubuntu images ?
3. Remove the `ubuntu` image
4. Remove all the remaining images

Run a docker image

Learn More

```
docker run -d --rm --name <NAME> -e ENV_VAR=value image[:TAG]
```

- `-d` : Run in detached mode (background)
- `--rm` : Delete the container after its execution
- `--name` : Give a name to the container
- `-e` : Set an environment variable in the container

```
docker run -u myuser -it image[:TAG]
```

- `-i` : Interactive mode, redirect STDIN inside the running container
- `-t` : Allocate a pseudo terminal
- `-it` : They are usually used in conjunction for interactive sessions
- `-u`, `--user` : Change the user

List running/terminated containers

Learn More

```
docker ps
docker ps -a -q
```

- `-a` : Show all containers (default shows only running)
- `-q` : Only display container IDs

```
# Example remove all terminated containers
docker rm $(docker ps -a -q)

# Example remove all running and terminated containers
docker rm -f $(docker ps -a -q)

# Easier and newer command
docker container prune
```

- `-f` : Force removal of running containers

Start a container

Learn More

```
docker start CONTAINER
```

This command can start a container.

Note

For this command to restart properly an exited container, the container has to be run with the `-it` options.

And the counterpart command to stop a container.

```
docker stop CONTAINER
```

Ex 3: Run a container

Hello world in the container !

1. Pull the `hello-world` image
2. List local images
3. Run the `hello-world` image
4. List all containers
5. Remove unused containers

Run a docker image with file sharing

Learn More

```
docker run --entrypoint=bash image[:TAG]
```

- `--entrypoint` : Override entrypoint specified in the Dockerfile

```
docker run -v /host/path:/container/path image[:TAG]
```

- `-v` : Allow the container to acess a folder on the host machine (absolute path)

```
docker run -p HOSTPORT:CONTAINERPORT image[:TAG]
```

- `-p` : Expose CONTAINERPORT in the container as HOSTPORT on the host machine

Get image/container informations

Learn More

```
docker inspect NAME|ID
```

Note

Allows to find the ENTRYPOINT or CMD that will be executed during run.

Ex 4: The `ubuntu` image

Let's play with `ubuntu`

1. Run the `ubuntu` image.
2. Run the `ubuntu` image and delete the container afterwards.
3. Run the `ubuntu` image while connecting to it. You can now run commands on ubuntu !
4. Find the entrypoint command
5. Change the entrypoint to echo some text.
6. Echo some text without changing the entrypoint. How does it work ?
7. Use both method at the same time. What happens ?

Execute a command in a running container

Learn More

```
docker exec CONTAINER COMMAND
docker exec mycontainer echo test
```

- `-e` : Set an environment variable in the container
- `-d` : Run in detached mode (background)
- `-i` : Interactive mode, redirect STDIN inside the running container
- `-t` : Allocate a pseudo terminal (usually used with -i)
- `-u`, `--user` : Change the user

Ex 5: Running in the background

1. Run the `ubuntu` image in background. Do not forget to give it a name for easier reference.
2. List running containers.
3. Execute a command in the running container.
4. Connect to the running container.

Show logs from a running container

Learn More

```
docker logs CONTAINER
```

- `CONTAINER` : Name given during launch or id of the container
- `-f` : Show logs as they are generated
- `-t` : Show timestamps

List processes running in a container

Learn More

```
docker top CONTAINER [ps OPTIONS]
```

Note

Also shows the cpu and memory usage of those processes

Copy files into/from a running container

Learn More

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH
docker cp [OPTIONS] SRC_PATH CONTAINER:DEST_PATH
```

- `CONTAINER` : Name given during launch or id of the container
- `SRC_PATH` : Path to the source of the copy
- `DEST_PATH` : Path to the destination of the copy

Ex 6: Run a nginx server

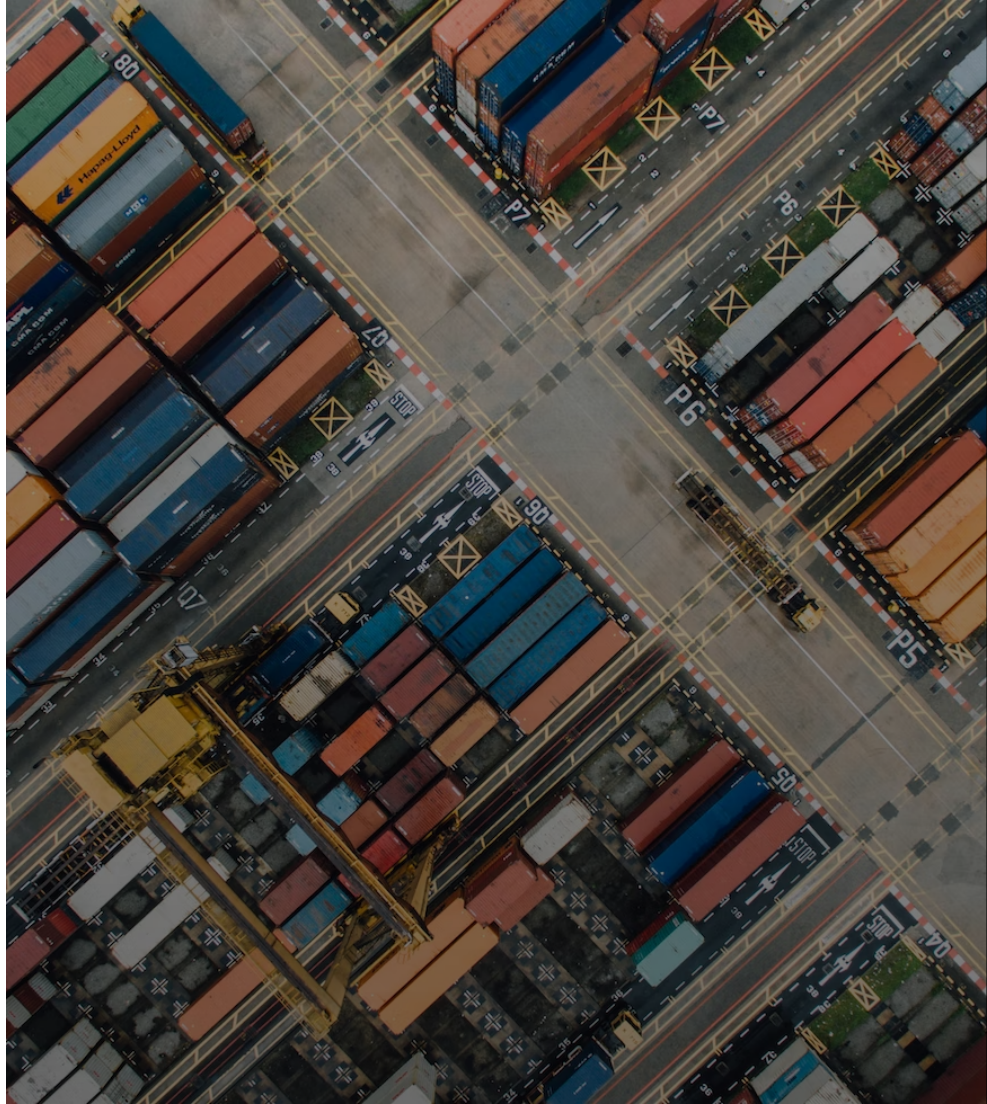
1. Pull the `nginx` image.
2. Run a nginx server on background with port 80 from container exposed host port 8080.
3. Check if it works by connecting to http://localhost:8080/
4. Listen to logs and refresh the webpage.
5. Find the pid of nginx master process.
6. Connect to the container and change the `/usr/share/nginx/html/index.html` file to serve a different content.
7. Copy this file on the host, modify it and copy the new version in the container.

Change image name and/or tag

Learn More

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]
```

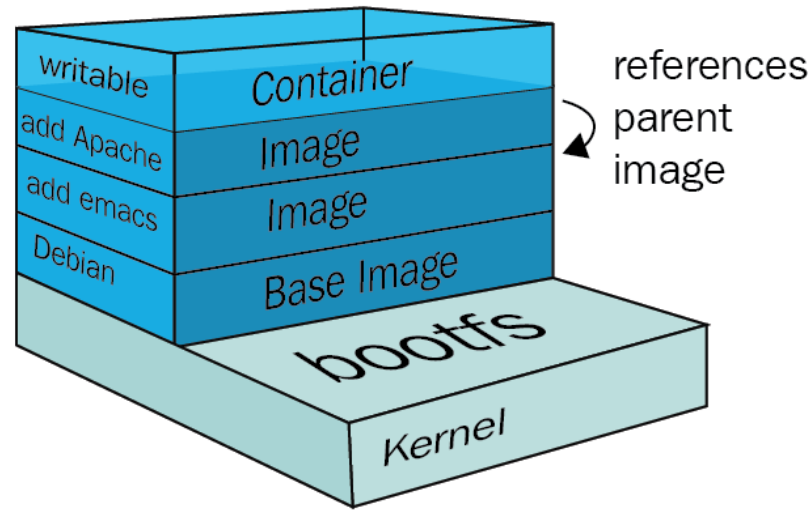
Image building



Dockerfile

The definition

- Set of instructions that are used to build an image
- Each instruction creates a new layer
- Modifications are added on top of the previous ones
  - Each layer is read-only
  - The last layer is writeable
  - The image is built from the first layer to the last one
  - The image is built from the cache if the layer is not modified



Dockerfile

An example

```
FROM python:3.9-slim

WORKDIR /app

COPY ./app .

RUN pip install -r requirements.txt

ENV NAME Aneo

EXPOSE 80

CMD ["python", "app.py"]
```

Dockerfile

Learn More

The more useful instructions !

- **FROM** : Specify the base docker image.
- **WORKDIR** : Change working directory.
- **RUN** : Execute a command and create a layer.
- **COPY** : Copy files from the host to the image during build.
- **ENTRYPOINT** : Command always executed when running the image.
- **CMD** : Default parameters to the command executed when running the image. Added after the entrypoint. Is overwriten by the command added in the command line during the run of the image.

Entrypoint example

The difference between ENTRYPOINT and CMD explained

If we consider the following ENTRYPOINT and CMD,

```
ENTRYPOINT ["python", "app.py"]
CMD ["-p", "8080"]
```

If we use `docker run myapp`, the command executed in the container will be:

```
python app.py -p 8080
```

But, if we use `docker run myapp --port 8443`, the command executed in the container will be:

```
python app.py --port 8443
```

Dockerfile

Some more useful instructions !

- **USER** : Change the user in the image (this also sets the default user when running the image).
- **ENV** : Set environment variable in the image.
- **EXPOSE** : Expose a port.
- **ARG** : Give an argument during build.

Image size optimization

Why a small image is better ?

- Takes more time to pull larger images
- During the image pulling, the machine is on but not computing
- The smaller the image, the faster the resources/services are producing value
- Reduces the load on the container registry (network and storage)

Image size optimization

How the image size increases ?

- Each additional instruction in the Dockerfile adds to the size of the image
- Make sure that each layer only adds the necessary things
- Remove temporary files in the same layer as their creation
- The image size increases even if the files are deleted in another layer

Image size optimization

How to reduce the image size ?

- Use a small sized base image
- Use .dockerignore to avoid copying unnecessary files
- Do not include superfluous dependencies
- Remove temporary data in the layer that creates them
- Use multi-stage building to reduce image size
- Tools to reduce images : Docker Slim

Multi-stage building

Learn More

Another way to reduce the size of the image

```
# first stage that is used to build the application
FROM ubuntu as build
RUN install tools needed to build the application (compiler)

COPY src/* /dest
WORKDIR /dest
RUN compile the code

# second stage that only holds the application and its runtime depdencies
FROM ubuntu as base
RUN install tools needed to execute the application (runtime)
COPY --from=build /dest/bin/ /usr/local/bin/

ENTRYPOINT ["/usr/local/bin/myapplication", "the", "options"]
```

Build docker image

Learn More

```
docker build -t myimage[:mytag] .
docker build -t myimage[:mytag] -f path/to/dockerfile context_path
```

- `-t` : Name and tag of the image that will be created during build. Name also includes the registry. Default registry is dockerhub
- `-f` : Path to the dockerfile
- `context_path` : Path to the folder from which to COPY input files

Notes

- Docker context is the root of the files that the build process will acces.
- `latest` is the default tag if it is not given.
- It is a special tag that represents the latest version of a image. You need to push it yourself.
- This tag should be used carefully as Docker does not update it locally when pulling an image as it already exists.

Registry

Docker images storage

- Stateless server application that stores and distributes Docker images
- Only stores layers
- Docker images references to multiple layers in the registries
- Ex: dockerhub, github registry, AWS ECR

Login to registry

Learn More

```
# Asks for password
docker login -u user --password-stdin [SERVER]

# Password is in the history
docker login -u user -p password

# Password can also be given like that
# Useful in pipelines and scripts
echo password | docker login -u user --password-stdin
```

- `-u` : Username
- `-p` : Password
- `--password-stdin` !: Take the password from stdin
- `SERVER` : registry to login into, default specified by daemon (dockerhub)

Warning

Commands are written in the bash history and thus the passwords !

Summary

Dockerfile

```
FROM python:3.9-slim
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

- Each instruction creates a layer
- Additional layers increase image size
- Small images let applications start sooner

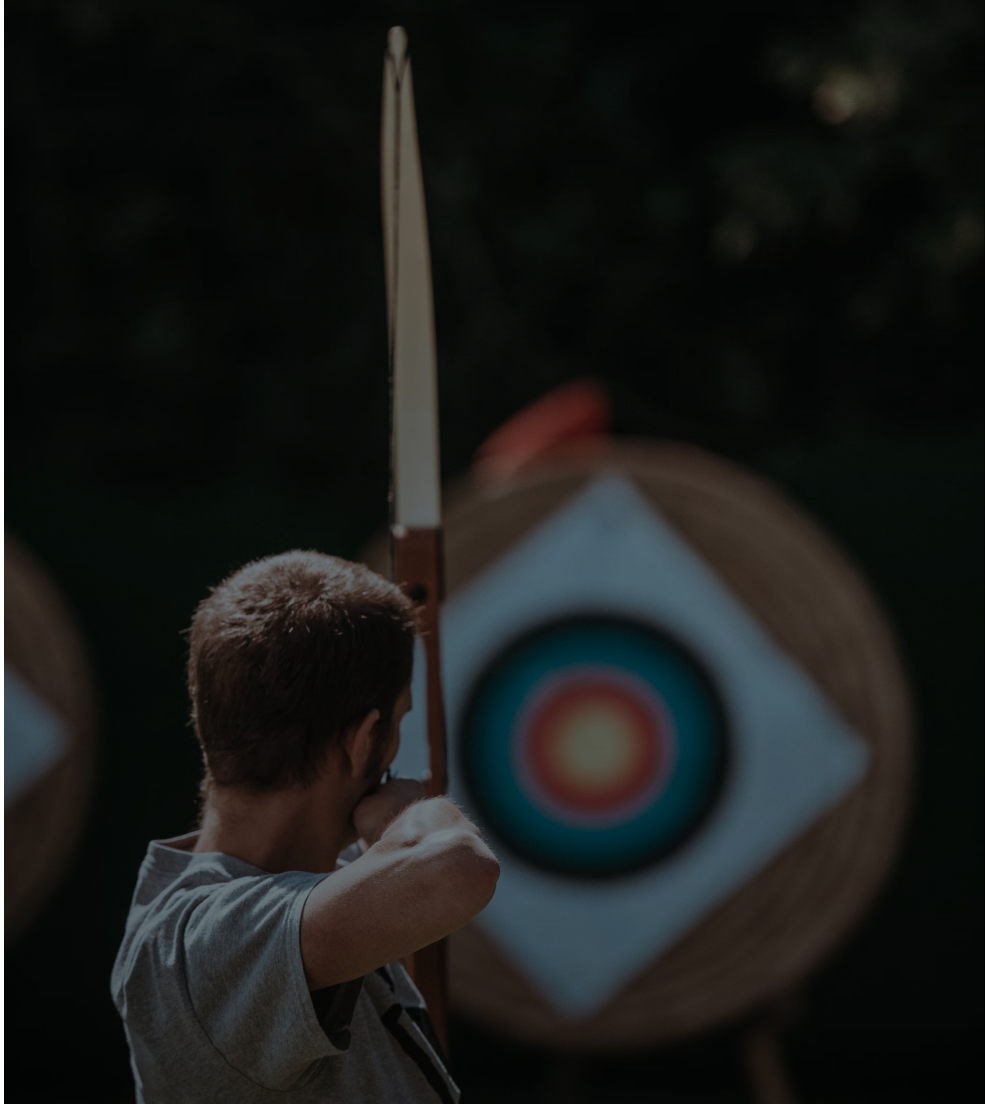
Cheatsheet

Cheatsheet2

Commands

```
# Build image from Dockerfile
docker build -t <image_name>

# List local images
docker images

# Delete image
docker rmi <image_name>

# Remove all unused images
docker image prune

# Create and run a container from an image, with a custom name:
docker run --rm --name <container_name> <image_name>

# Start or stop an existing container:
docker start|stop <container_name> (or <container-id>)

# Remove a stopped container:
docker rm <container_name>
```

More practice



Ex 7: Write a Dockerfile and build the image

1. Write a Dockerfile based on ubuntu
2. Install git
3. Install AWS CLI v2
4. Build the image
5. List local images
6. Change the tag of the created image
7. Remove dangling images
8. Create an new user and change to the created user
9. Build the image
10. Put an entrypoint for the aws cli
11. Build the image
12. Make sure the image is not too big

Ex 8: Make the AWS CLI work through Docker

The aws configure way v1

Note

There are multiple ways to do so !

1. Run the image.
2. Configure aws cli with `aws configure`.
3. Use aws cli commands.
4. What are the issues ?

Ex 8: Make the AWS CLI work through Docker

The aws configure way v2

1. Run the image in the background.
2. Exec `aws configure` in the running container.
3. Exec aws cli commands in the running container.
4. What are the issues ?

Alternative

Copy the AWS credential file into the running container.

Ex 8: Make the AWS CLI work through Docker

Set environment variables in the running container

1. Execute aws cli commands with the image while configuring the cli through environment variables.
2. What are the issues ?

Note

You can use `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.

Ex 8: Make the AWS CLI work through Docker

Mount aws credential file into the docker

1. Execute aws cli commands with the image while configuring the cli through environment variables.
2. What do you need to change if you want to be the `root` user ?
3. What are the issues ?

Solutions

Ex 1: Pull docker images

1. `docker pull hello-world`
2. `docker pull ubuntu`
3. `docker pull fedora`
4. `docker pull ubuntu:22.04`

Ex 2: Remove local images

1. `docker images`
2. They have the same image id. They are basically the same images but with different tags.
3. `docker rmi ubuntu`
4. `docker rmi $(docker images -q)` or `docker image prune`

Ex 3: Run a container

1. `docker pull hello-world`
2. `docker images`
3. `docker run hello-world`
4. `docker ps -a`
5. `docker rm $(docker ps -a -q)`

Ex 4: The `ubuntu` image

1. `docker run ubuntu`
2. `docker run --rm ubuntu`
3. `docker run -it ubuntu`
4. `docker inspect ubuntu | jq -r .[0].ContainerConfig.Cmd`
5. `docker run --entrypoint echo ubuntu some text`
6. `docker run ubuntu echo some text`. In this case, it is the command (CMD) that is changed.
7. `docker run --entrypoint echo ubuntu echo some text`. `echo echo some text`

Ex 5: Running in the background

1. `docker run -it -d --name myubuntu ubuntu bash`
2. `docker ps`
3. `docker exec myubuntu whoami`
4. `docker exec -it myubuntu bash`

Ex 6: Run a nginx server

1. `docker pull nginx`
2. `docker run -it -d -p --rm --name mynginx 8080:80 nginx`
3. It should work
4. `docker logs mynginx -f`
5. `docker top mynginx`
6. `docker exec -it mynginx bash`. Should not be easy to edit since there almost nothing installed in the image. User can still install packages but it increases the size of the running container.
7. `docker cp mynginx:/usr/share/nginx/html/index.html .`, edit, `docker cp index.html mynginx:/usr/share/nginx/html/index.html`

Ex 7: Write a Dockerfile and build the image

Dockerfile

```
FROM ubuntu

RUN apt-get update \
  && apt-get install -y git jq curl unzip mandoc \
  && apt-get clean \
  && rm -rf /var/cache/apt/list/*

RUN curl \
  "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zi
  -o "awscliv2.zip" \
  && unzip awscliv2.zip \
  && ./aws/install \
  && rm awscliv2.zip

RUN useradd -m -U aneo
USER aneo
ENTRYPOINT ["aws"]
```

Commandes

```
docker build -t my_awscli .
docker tag my_awscli my_awscli:v1

docker images
docker image prune
docker rmi $(docker images -q -f dangling=true)
```

Ex 8: Make the AWS CLI work through Docker

The aws configure way v1

1. `docker run -it my_awscli --name awscli bash`
2. `aws configure`
3. `aws sts get-caller-identity`
4. You cannot run aws cli commands with tools installed on your machine. You have to exit the container to do so.

Ex 8: Make the AWS CLI work through Docker

The aws configure way v2

1. `docker run -it -d my_awscli --name awscli bash`
2. `docker exec awscli configure`
3. `docker exec awscli sts get-caller-identity`
4. You need to have a running container in the background. Do not forget to set it up.

Alternative : `docker exec awscli mkdir /home/aneo/.aws && docker cp $HOME/.aws/credentials awscli:/home/aneo/.aws/credentials`

Ex 8: Make the AWS CLI work through Docker

Set environment variables in the running container

1. `docker run -e AWS_ACCESS_KEY_ID=xxx -e AWS_SECRET_ACCESS_KEY=xxx my_awscli sts get-caller-identity`

2. Your credentials may be shown on the command line. It is also a lengthy command.

Ex 8: Make the AWS CLI work through Docker

Mount aws credential file into the docker

1. ```
   docker run -v "$HOME/.aws/credentials":"/home/aneo/.aws/credentials" my_awscli sts get-caller-identity
   ```

2. ```
   docker run -v "$HOME/.aws/credentials":"/root/.aws/credentials" -u root my_awscli sts get-caller-identity
   ```

3. Mounting files can let the user inside the container modify files with higher permissions than your regular user.

References

- https://d2iq.com/blog/brief-history-containers
- https://blog.packagecloud.io/what-are-docker-image-layers
- https://docs.docker.com/registry/introduction/
- https://blog.runcloud.io/when-and-why-to-use-docker/
- https://docs.docker.com/get-started/docker_cheatsheet.pdf
- https://dev.to/kitarp29/reducing-docker-image-size-a67
- https://www.geeksforgeeks.org/why-should-you-use-docker-7-major-reasons/