Cloud Infrastructures

05 - Introduction to Docker

Juan Ángel Lorenzo del Castillo

CY Cergy Paris Université

ING3 - Ingénierie du Cloud Computing

2020-2021

juan-angel.lorenzo-del-castillo@cyu.fr

CY TECH

Acknowledgements

Disclaimer

Most of the content of these slides is either taken or inspired by the official Docker documentation, from courses I took with the Docker guys, or from my own experience.

So (for the biggest part of it) I am not claiming any authorship of this course.

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

4 Creating and managing containers

5 Docker images

6 Exercise

CY TECH

CC BY SA

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

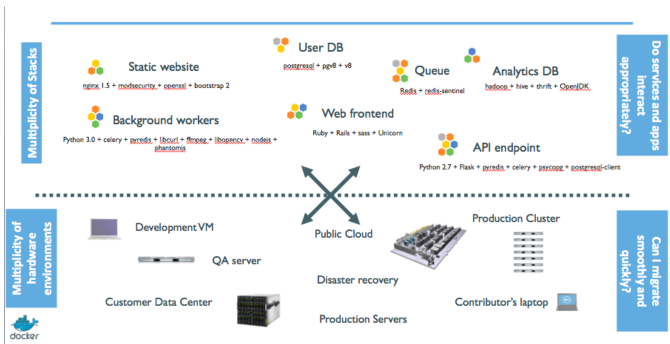
4 Creating and managing containers

5 Docker images

6 Exercise

Why containers

The deployment problem in the software industry



Source : Docker

Before

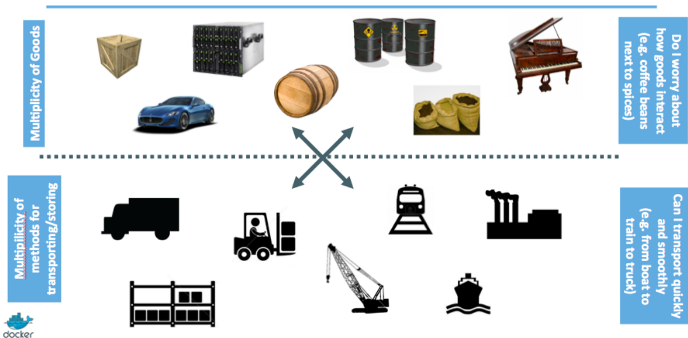
- monolithic applications
- long development cycles
- single environment
- slowly scaling up

Now

- decoupled services
- fast, iterative improvements
- multiple environments
- quickly scaling out

Why containers

An analogy (from ancient history)



Source : Docker

Why containers

Intermodal shipping containers

Source : Docker

Why containers

This spawned a shipping container ecosystem

- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

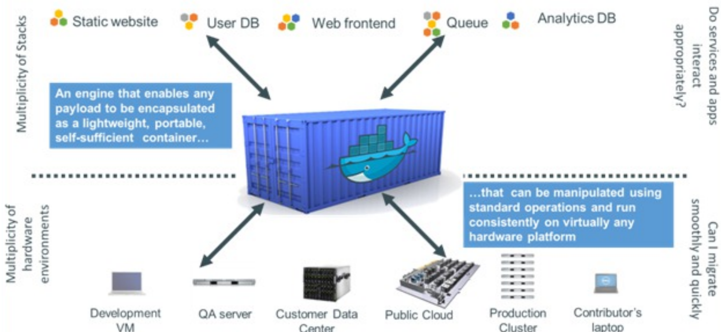
→ massive globalization

- 5000 ships deliver 200M containers per year

Source : Docker

Why containers

A shipping container system for applications



Source : Docker

How a container works

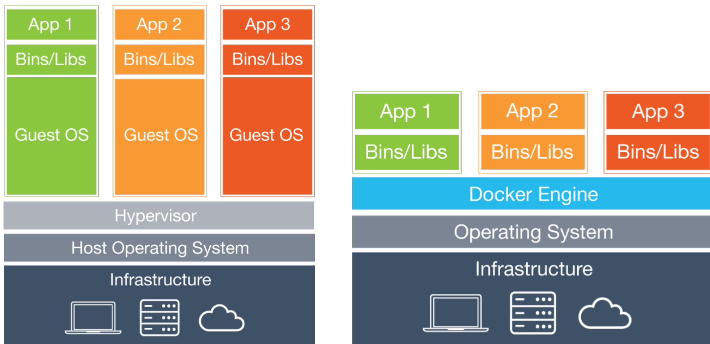
Isolates a running application into an execution context.

Like virtualisation, but without virtualisation:

- Agnostic about the content and the transporter.
- Isolation et automation.
- Principe of consistent and repeatable infrastructure.
- Very low overhead compared to a VM.
- A super chroot (*chroot on steroids*).
- It has been an asset for Solaris for many years.
- Technology used by Google in their **Borg** datacentre scheduler.

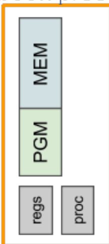
CC BY SA

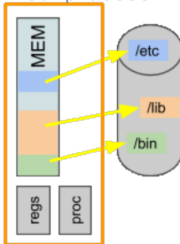
Differences between a VM and a container

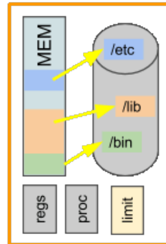


Source : Docker

Differences between a process and a container







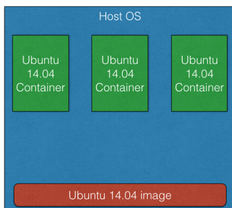
Source : rightscale.com

- With its own process space
- With its own network interface
- Without an own `/sbin/init`
- Isolated processes
- Kernel shared with the host
- Identical environments with the same package versions and configurations across all containers

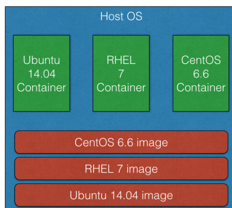
CY TECH
CC BY SA

Typical use cases for containers

1. Operating system container
   - Virtual environment sharing the kernel with the host's OS
   - But isolated in the host's userspace
   - It may run several processes and services
   - Practical to run different flavours of Linux in the same machine
2. Application container
   - A single service or application per container
   - **Microservices :** Split a large application into multiple smaller services
   - Instead of updating the whole applications, only the concerned services will be updated



Identical OS containers



Different flavoured OS containers

Source : risingstack.com

CC BY SA

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

4 Creating and managing containers

5 Docker images

6 Exercise

CY TECH

CC BY SA

Contributions of Docker

Formats and APIs

Before Docker

- No standardised exchange format (containers were not portable).
- Containers are hard to use for developers.
- As a result, they are hidden from the end users.
- No re-usable components, APIs, tools.
- *Analogy:* Shipping containers are not just steel boxes. They are steel boxes with a standard size, same hooks and holes.

After Docker

- Standardised container format.
- Make containers easy to use for developers.
- Emphasis on re-usable components, APIs, ecosystem of standard tools.
- Improvement over ad-hoc, in-house, specific tools.

CC BY SA

Contributions of Docker

Shipping

Before Docker

- Ship packages: deb, rpm, gem, jar, homebrew...
- Dependency hell.
- "Works on my machine."
- Base deployment often done from scratch and unreliable.

After Docker

- Ship container images with all their dependencies.
- Images are bigger, but they are broken down into layers (e.g: JRE, Tomcat, JAR app, etc.).
- Only ship layers that have changed.
- Save disk, network, memory usage.

Contributions of Docker

Devs vs. Ops

Before Docker

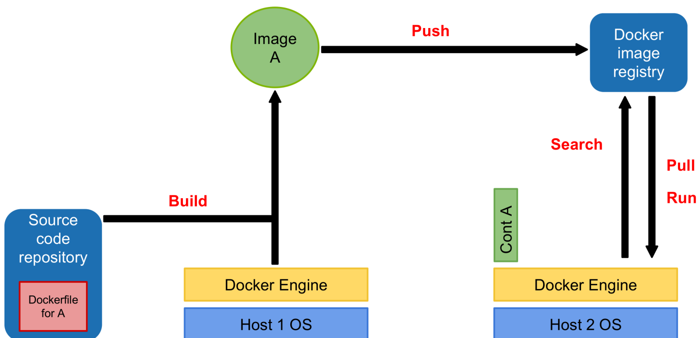
- Drop a tarball with instructions.
- Dev environment very different from production.
- Ops don't always have a dev environment themselves, and when they do, it can differ from the devs'.
- Ops have to sort out differences and make it work... or bounce it back to devs. Shipping code causes frictions and delays.

After Docker

- Drop a container image or a Compose file.
- Ops can always run that container image or Compose file.
- Ops still have to adapt to prod environment, but at least they have a reference point.
- Ops have tools allowing to use the same image in dev and prod.
- Devs can be empowered to make releases themselves more easily.

CC BY SA

Docker for software management



Source: Treeptik

What is Docker

- **Docker:** the platform
  - Docker Engine.
  - Docker Hub.
  - Docker Compose.
  - Docker Machine.
  - Docker Swarm.
  - Kitematic.
  - Docker Cloud.
  - Docker Datacenter.
  - etc.

- **Docker Inc**: the company
  - Docker Inc. used to be dotCloud Inc.
  - dotCloud Inc. used to be a French company.
  - Docker Inc. is the primary sponsor and contributor to the Docker Project:
    - Hires maintainers and contributors.
    - Provides infrastructure for the project.
    - Runs the Docker Hub.
  - HQ in San Francisco.
  - Backed by more than 100M in venture capital.

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

4 Creating and managing containers

5 Docker images

6 Exercise

CY TECH
CC BY SA

Install Docker

- By default, Docker should be already installed and running on your EISTI laptop. Check it out with:

```
$ docker version  # you might need to prefix it with sudo
```

  if you get a sensible answer, there is nothing else to do.

- Otherwise, if you want to install Docker on a Linux machine, you have other options:
  - Installing from distros packages

```
$ sudo yum install docker          # Red Hat and derivatives
$ sudo apt-get install docker.io   # Debian and derivatives
```

  - Installation script From Docker. It works on Ubuntu, Debian, Fedora and Gentoo.

```
$ curl -s https://get.docker.com/ | sudo sh
```


cc BY SA

Configure Docker (I)

- The Docker engine runs a client and a server.
- The Docker user is `root` equivalent
- It provides root-level access to the host.
- If your user is not in the Docker group, you will need to prefix every command with `sudo`; e.g. `sudo docker version`.
- To avoid this, add your user to the Docker group:
  - Add the Docker group

    ```
    $ sudo groupadd docker
    ```

  - Add your user to the group

    ```
    $ sudo gpasswd —a $USER docker
    ```

  - Restart the Docker daemon

    ```
    $ sudo service docker restart
    ```


CC BY SA

Configure Docker (II)

Issues

- Docker provides a network in a private subnet address range for container communication purposes.
- It happens to be the same network as one of EISTI private subnets.
- To avoid conflicts it has been changed in your laptops to a subnet in the `10.42.0.0/24` range.
- You can check it by inspecting Docker's network configuration:

```
$ ip addr show
...
6: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue ...
    link/ether 02:42:66:7d:24:7c brd ff:ff:ff:ff:ff:ff
    inet 10.42.0.1/24 scope global docker0
```

- If your address range is not the previous one, and you do not want Alex to have a word with you, make sure to change it (see next slide).

CC BY SA

Configure Docker (III)

- Changing Docker's default subnet to a EISTI's friendly one (**only if needed!**):
- **Solution 1:**
  - Create a file `/etc/docker/daemon.json` as root

    ```
    $ sudo touch /etc/docker/daemon.json
    ```

  - Using your favourite editor, add the following lines to the file

    ```
    {
        "bip": "10.42.0.1/24",
        "dns": ["193.48.70.2","194.57.186.254","193.55.155.254"]
    }
    ```

  - Restart the Docker daemon:

    ```
    $ sudo service docker restart
    ```

- If, for any reason, this solution does not work properly, try the steps on the next slide.

CC BY SA

Configure Docker (III)

- Changing Docker's default subnet to a EISTI's friendly one (**only if needed!**):
- **Solution 2:**
  - Stop the Docker daemon

```
$ sudo service docker stop
```

  - Disable the docker's bridge network interface (in our example, `docker0`):

```
$ sudo ip link set docker0 down
```

  - Delete the bridge interface:

```
$ sudo brctl delbr docker0
```

  - Recreate all symlinks:

```
$ sudo systemctl enable docker
```

  - Now, two options depending on your Linux version:
    1. Edit the file `/etc/systemd/system/multi-user.target.wants/docker.service` Add `"--bip=10.42.0.1/24"` (without the quotes), or
    2. Edit the file `/etc/default/docker`. Add the line `DOCKER_OPTS="--bip=10.42.0.1/24"` (including the quotes).
  - Finally, recreate all dependencies and restart the Docker daemon:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

Table of Contents

1 Introduction
2 Docker
3 Docker configuration
4 Creating and managing containers
5 Docker images
6 Exercise

Docker architecture

- Docker is a client-server application:
  - The **Docker Engine** (or "daemon"): receives and processes incoming Docker API requests.
  - The **Docker client**: talks to the Docker daemon via the Docker API. The client includes a CLI (Command Line Interface) to send commands to the Docker daemon.
  - The **Docker Hub Registry**: collection of public images. The Docker daemon talks to it via the registry API.

Our first container

- Open a CLI terminal and type

```
$ docker run -it ubuntu /bin/bash
```

  - It runs a brand new container.
  - `ubuntu` : The image to be used to create the container. If the ubuntu image is not available locally it will be downloaded from the Docker Hub.
  - `-it` : shorthand for `-i -t`.
    - `-i` (interactive) tells Docker to connect to the container's stdin (the command line).
    - `-t` tells Docker that we want a pseudo-terminal.
  - `/bin/bash` : we want to run a bash prompt on the container.

CC BY SA

Our first container

```
$ docker run -it ubuntu /bin/bash
root@0bc82356b52d9:/# cat /etc/issue
Ubuntu 14.04.2 LTS
root@0bc82356b52d9:/# dpkg -l | wc -l
136
```

- Perform the following actions:
  - Check the ubuntu version of our container and compare it with your host's.
  - Check the kernel version of our container and compare it with your host's.
  - Compare the number of packages installed in our container with those in your host.
  - Compare the filesystem tree in your container and in your host (use `df -h`). If you create a file in your container filesystem, does it also exist in your host?
  - Install and run *figlet*:

```
root@0bc82356b52d9:/# apt-get update
...
root@0bc82356b52d9:/# apt-get install figlet
...
root@0bc82356b52d9:/# figlet Hola mundo!
```


CC BY SA

Our first container

- Type `exit`. Your container is now in a *stopped* state. It still exists on disk, but all compute resources have been freed up.
  - From the host's command line, list the active containers with `docker ps`. Then, run `docker ps -a`. Why does the output differ?
  - If you start a new container with `docker run`, we will start a brand new container from the *ubuntu* image, in which `figlet` will not be there.
- To come back to our stopped container, from our host we need to:
  1. Find out the container ID:

     ```
     $ docker ps -a
     CONTAINER ID        IMAGE
     0bc82356b52d9       ubuntu
     ```

  2. Start the container

     ```
     $ docker start 0bc82356b52d9
     ```

  3. Attach the container

     ```
     $ docker attach 0bc82356b52d9
     ```

  We will see later how to create a custom image that contains our modifications to the base image.

CC BY SA

Background containers

A non-interactive container can be run in the background.

- Run the following container:

```
$ docker run jpetazzo/clock
```

- It executes a non-interactive container that runs forever.
- To stop it, press ˆC.
- Now start the container in the background, with the -d flag (daemon).

```
$ docker run —d jpetazzo/clock
50c95fe2e63e3d491aa18d76ecc4180cb91d73785c12bc4c5ac65a2634567442
```

- Docker gives the container ID.
- The container keeps running in the background. Check it with `docker ps` (BTW, test `ps` with the options `−q` and `−l`).
- Docker keeps logging the container output.

CC BY SA

Background containers

Container logs.

- Check the container's output using a prefix of the container ID

```
$ docker logs 50c95
Mon Jul 24 18:44:25 UTC 2017
Mon Jul 24 18:44:26 UTC 2017
```

- To view only the tail of the logs:

```
$ docker logs --tail 3 50c95
```

- To follow the logs in our background container in real time (like `tail -f` in Linux):

```
$ docker logs --tail 1 --follow 50c95
```


CC BY SA

Background containers

Stopping and killing background containers.

1. Killing a container using the `docker kill` command
   - It stops the container immediately, by using the KILL signal.
2. Stopping a container using the `docker stop` command
   - It sends a TERM signal and, after 10 seconds, if the container has not stopped, it sends KILL.

Detaching and attaching to containers

- To detach from an interactive container, use the *detach* sequence: `^P^Q`
  - The detach sequence can be customised with `docker run --detach-keys`. For example, to stop the container with the sequence `^x j`:

```
$ docker run -ti --detach-keys ctrl-x,j jpetazzo/clock
```

- To attach to a running container (as explained before):

```
$ docker attach containerID
```

  - The container must be running
  - There can be multiple containers attached to the same container
  - It defaults to `^P^Q` if `--detach-keys` is not specified.
  - To try it on our last container:

```
$ docker attach $(docker ps -lq)
```


CC BY SA

Debugging inside the container

- `docker exec`
  - It allows users to run a new process in a container which is already running.
  - It is not meant to be used for production, but it is handy for development.
  - We can get a shell prompt inside an existing container this way.
  - For example:

```
$ docker run -d jpetazzo/clock
6dd2aa3aa787196899c984651ffd58504c33d5c3cf64f25e03620e419ea86f4e

$ docker exec -it 6dd2aa3aa787 echo hola
hola
```


cc BY SA

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

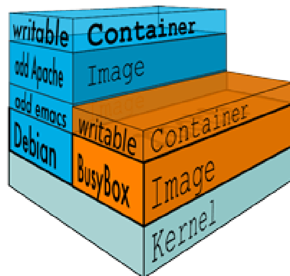
4 Creating and managing containers

5 Docker images

6 Exercise

Images and layers

- **Image** = collection of files + metadata
  - Those files form the root filesystem of a container.
  - An image is a read-only filesystem.
- Images are made of ***layers***, conceptually stacked on top of each other.
- Each layer can add, change and remove files. It is a *differential* from the previous layer.
- Images can share layers to optimise disk usage, transfer times, memory use, etc.
- **Container**: encapsulated set of processes running in a read-write copy of that filesystem.



Images and layers

If an image is read-only, how do we change it?

- We don't.
- We create a new container from that image.
- Then we make changes to that container.
- When we are satisfied with those changes, we transform them into a new layer.
- A new image is created by stacking the new layer on top of the old image.

Two methods to create an image:

- `docker commit`
  - Saves all the changes made to a container to a new layer
  - Creates a new image as a copy of the container
- `docker build`
  - Used with *Dockerfiles*
  - Performs a repeatable build sequence

cc BY SA

Images namespaces

- Root namespace
  e.g. `ubuntu, busybox, ...`
  - For official images. Put in the registry by Docker Inc. but generally authored and maintained by third parties.
  - We can find small images (busybox), distro images to be used as bases for further builds (ubuntu, fedora), or ready-to-use components and services (postgresql, redis).
- User namespace
  e.g. `jpetazzo/clock`
  - Images from Docker Hub users and organisations.
  - Example: `jpetazzo` is the Docker Hub user. `clock` is the image name.
- Self-hosted images
  e.g. `myregistry.example.com:5000/myprivate/image`
  - Images hosted on third-party registries.
  - Address, port and iname name.

Handling Images

- To list all our local images (images stored in our Docker host):

```
$ docker images
REPOSITORY          TAG       IMAGE ID        CREATED          SIZE
sameersbn/skype     1.0.1-3   4c06c9bb3da0    5 months ago     411.5 MB
ubuntu              latest    c73a085dc378    10 months ago    127.1 MB
jpetazzo/clock      latest    12068b93616f    2 years ago      2.433 MB
```

- To search for images on a remote registry (we cannot list all images, we search by keywords)

```
$ docker search wordpress
NAME                 DESCRIPTION                STARS   OFFICIAL   AUTOMATED
wordpress            The WordPress rich...      1826    [OK]
bitnami/wordpress    Bitnami Docker Ima...      42                 [OK]
...
```

  - *Official* images are those in the root namespace.
  - *Automated* images built automatically by the Docker Hub (build recipe always available).

Handling Images

- To download an image:
  - Implicitly, when we run `docker run` and the image is not found locally.
  - Explicitly, with `docker pull`

```
$ docker pull wordpress:cli-1.2
```

  - `cli-1.2` is the *version tag*: the exact version we want to download. By default, if nothing is specified, the version downloaded will be *latest*.
  - Tags ensure that the same version will be used everywhere (repeatability).
  - Docker downloads all the necessary layers to build the image.
- To remove a container or an image:
  - `docker rm containerID` removes a container, **not** an image.
  - `docker rmi imageID` removes an image.

Building images interactively

1. Create a container from a base image.
2. Install software manually in the container. To see the differences between the base image and your container:

```
$ docker diff containerID
```

3. Turn it into a new image with tag `image_name`:

```
$ docker commit containerID image_name
```

4. Run the new image:

```
$ docker run -it image_name
```

- We can also optionally use

```
$ docker tag imageID image_name
```

to tag an image.

CC BY SA

Table of Contents

1 Introduction

2 Docker

3 Docker configuration

4 Creating and managing containers

5 Docker images

6 Exercise

CY TECH

CC BY SA

Exercise 1: Managing and distributing containers

Hands On!

Go to TP1