Introducton to Data Engineering 6

Paul Blondel

UTSEUS, Shanghai

March 30st, 2021

'Our mission is to build tools of mass innovation.'

Solomon Hykes, Docker creator

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

Let's first understand what is and what is not Docker before going any further...

- You are already familiar with the concept of a virtual machine (VM).
-
  - Consists in simulating a computer on a physical machine
  - The hypervisor allocate some of the resources of the machine (memory, CPU)
  - VMs can be totally different from each other (one can be Windows, one Linux, etc)
- Interest of VMs is mainly flexibility and optimization of the use of hardware resources.
- The organization in VMs makes it easier to reallocate and change emulated physical devices (disk, memory, network, etc.)
- However: VMs are quite resource-intensive.
  - You have to run a complete operating system each time...

Docker is different because...

- It offers a much lighter solution (based on the ability of the Linux system):
  - Can isolate spaces to which part of the host machine's resources are allocated.
- These spaces (also named containers) partition the host system into sub-systems, within which the naming (of processes, users, network ports) is purely local.
  - Example: you can run a web server process on port 80 in container A, another web server process on port 80 in container B, without conflict or confusion. All names are somehow interpreted with respect to a given container (notion of namespace).
- Linux containers are much lighter in resource consumption than VMs, since they run within a single operating system. Docker exploits this specificity of the Linux system to offer a light and flexible virtualization mode (which we called "pseudo-server" in the preamble).

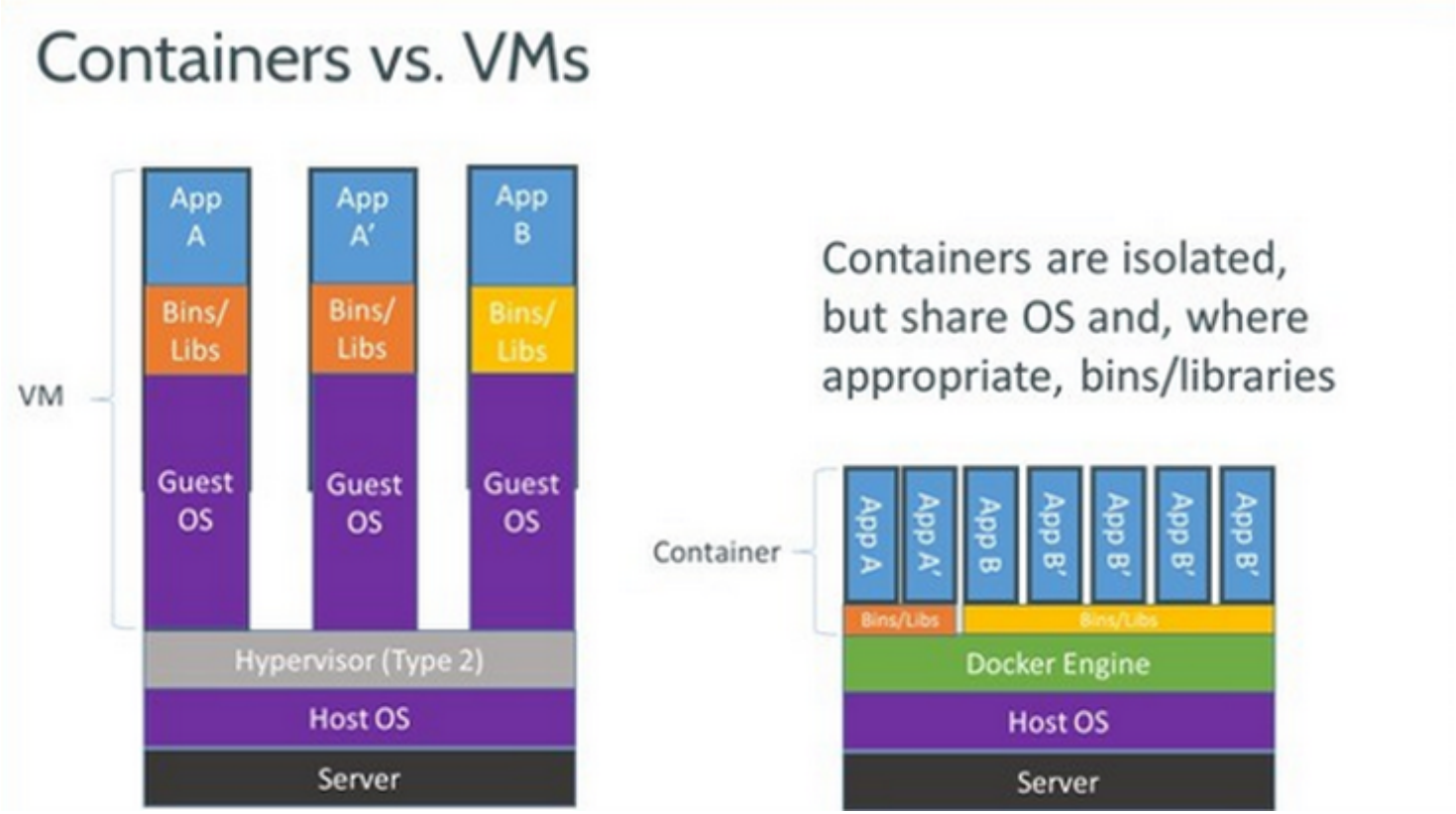

Figure: Differences between VMs and Docker containers

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

Thanks to Docker...

- Less dependency on infrastructure for testing apps
  - you don't have to worry if you forgot or not to install all the libraries, and if your colleague's environement (MacOS) or the production environnent (Ubuntu) are compatible or not : you just pop up a docker container image !
- Easy to deploy apps with diff version (one container can have one version, another another version, etc.)
- Easy to test any software (if the app crashes: it is contained in the container: just need to pop a new container)
- Easy to productionalize apps
- Running self dependent apps rather than having issues on version installed
- Easy to scale (add new containers with containing the same app)
- Cheaper than VM (lighter, takes less memory, faster, etc).

Docker (or, more precisely, the docker engine) is a program that allows us to create containers and install ready-to-use environments, the images. A little vocabulary: in all that follows,

- host system is the main operating system managing your machine; it is for example Windows, or Mac OS.
- Docker engine is the program that manages the containers
- A container is an autonomous part of the host system, behaving as an independent machine
- The Docker client is the utility through which we transmit to the engine the commands to manage these containers. It can be either from the command line (Docker CLI) or from kitematic.

- A Docker container can therefore be seen as an autonomous sub-system, mobilizing very few resources because most of the system tasks are delegated to the system in which it is instantiated.
- It is therefore virtually a way to multiply at low cost pseudo-machines in which we could install "by hand" various software. Docker goes a step further by offering pre-configured installations, packaged in such a way that they can be placed very easily in a container. They are called images.
- You can find images with pre-configured data servers (Oracle, Postgres, MySQL), Web servers (Apache, njinx), NoSQL servers (mongodb, cassandra), search engines (ElasticSearch, Solr). The installation of an image is very simple, and considerably relieves the sometimes tedious tasks of direct installation
- An image is placed in a container. The same image can be placed in several containers and thus obtain a distributed system[1]

[1]Which is very convenient for Data Engineering!

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

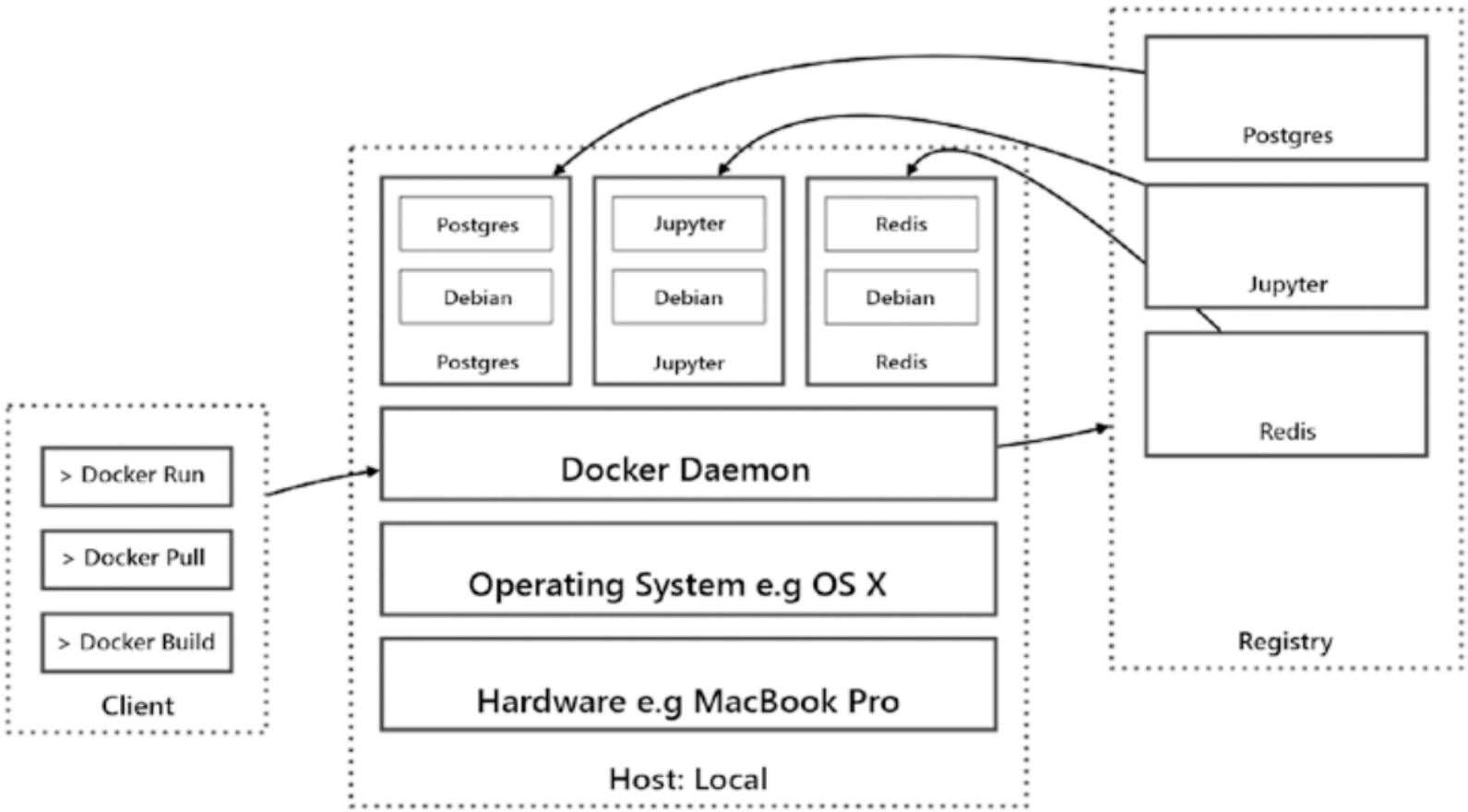

Figure: The docker ecosystem

[1] The Docker Engine is often called Docker Daemon and vice-versa.

Let's describe briefly the Docker ecosystem :
You begin looking at Docker by focusing on the ecosystem of the container. Later, you will leverage Docker's tools for composing larger systems with the containers you have built. In the immediate ecosystem of the Docker container, it is important to keep track of the following concepts:

- The Docker client (mostly command line)
- The Host
- The Docker Engine (or Docker Daemon)
- Docker images
- Docker containers
- The Docker Registry
- Docker Compose (not in the diagram)

- The Docker Client: The Docker client is a command line interface used to give instructions to the Docker engine regardless of the details of the engine's implementation on your system. This is similar to the client-server architecture of the Web to interact with a remote server. In the case of Docker, the Docker client talks to the Docker engine that performs the work of containers and containerization. We will be using the Docker command line.

- The Host: The host is a machine on which you will run the Docker daemon/engine. Locally, the host will depend upon your Docker configuration. If you are running Docker for Linux, Docker for Mac, or Docker for Windows, the host will be your system itself.

- The Docker Engine: The Docker engine is a persistent process that manages containers. It is running as a background service or daemon on the host. In fact, the engine is occasionally referred to as the Docker daemon. The Docker engine does the core work of Docker: building, running, and distributing your Docker containers. In this text, you will interact with the engine directly but will do so through the Docker client.
- Docker images and docker containers: The Docker engine has several methods for building our own images. These include the Docker client and via a domain-specific language (DSL) known as the Dockerfile. We can also download images that other people have created. Docker containers are instances of Docker images. They are stand-alone, containing everything a single instance of an application needs to run (OS, dependencies, source, metadata), and can be run, started, stopped, moved, and deleted. They are also isolated and secure. It is helpful to think of Docker images and containers in terms of object-oriented programming. An image is a defined "class" of container that we might create. A container is then an "instance" or "object" of that class. The Docker engine will manage multiple

- The Docker Registry: Docker registries hold images. These are public or private stores from which you upload or download images. For the purposes of this book, you will use the public Docker registry at Docker Hub.3 Docker Hub is the source of the official images of the major open source technologies you will be using including Jupyter, PostgreSQL, Redis, and MongoDB.
- Docker Compose: Docker Compose is a tool for assembling microservices and applications consisting of multiple containers. These microservices and applications are designed using a single YAML file and can be built, run, and scaled, each via a single command. Docker Compose is particularly useful for the data scientist in building standalone computational systems comprised of Jupyter and one or more data stores (e.g. Redis).

How to use the Docker client to run containers?

To communicate with the Docker engine, we can use a command line client program named simply docker. The simplest image is a Hello world, we instantiate it with the following command:

```
$ docker run hello-world
```

The run is the command to instantiate a new image in a Docker container. Here is what you should get on the first run:

```
$ docker run hello-world

Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
4276590986f6: Pull complete
Status: Downloaded newer image for hello-world:lates

Hello from Docker!
```

The list of the available images can be obtained using the following command:

```
$ docker images
```

Here is the type of output you should get now:

```
REPOSITORY          TAG                 IMAGE ID
hello-world         latest              4276590986f6
```

You can instantiate an image and give it a name using the following command:

```
$ docker run --name 'name-container' <options>
```

Let's try now to run a basic container with a materialistic ubuntu system:

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

For more information about Docker-run:

```
$ man docker-run
```

The list of containers and their status can be obtained using:

```
$ docker ps -a
```

Example of output you could get:

```
CONTAINER ID  IMAGE         COMMAND       CREATED         STATUS                      PORTS  NAMES
d1c2291dc9f9  mysql:latest  "docker..."   16 minutes ago  Exited (1) 9 minutes ago    3306   db
ec5215871db3  hello-world   "/hello"      19 minutes ago  Exited (0) 19 minutes ago           my test
```

Note the first field, CONTAINER ID, which tells us the identifier by which we can pass instructions to the container. Here are the most useful ones, assuming the container is d1c2291dc9f9. First of all we can stop it with the stop command.

```
$ docker stop d1c2291dc9f9
```

Stopping a container does not mean that it no longer exists, but that it is no longer active. You can restart it with the start command:

```
$ docker start id_container
```

To remove it, it is the docker rm command. To inspect the system/network configuration of a container, Docker provides the inspect command:

```
$ docker inspect id_container
```

We obtain a large JSON document. Among all the information given, the IP address of the container is particularly interesting. We get it with:

```
$ docker inspect <id_container> | grep "IPAddress"
```

In case your container contains a server (here for example a http server is installed) :
Docker provides a publishing mechanism to indicate on which port a container is listening. You simply indicate with the –publish (or -p) option how to associate a container port with a host system port. Example:
–detach (or -d) indicates that the container is launched in the background, which avoids blocking the terminal

```
$ docker run --name web2 --publish 81:80 --detach httpd:latest
```

Or simply:

```
$ docker run --name web2 -p 81:80  -d httpd
```

The -p option indicates that port 80 of the container is forwarded to port 81 of the host machine.

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

- Every Docker image is defined as a stack of layers, each defining fundamental, stateless changes to the image:
  - The first layer might be a minimalist operating system image (Debian, Ubuntu, etc. Docker image)
  - The next layer is the installation of dependencies necessary for your application to run, and all the way up to the source code of your application.
- The file named "Dockerfile" permits to define the layers of the image and thus build the image:
  - The file uses a domain-specific language to tell the Docker daemon how to sequentially build a Docker image.
  - All the instructions contained in Dockerfile are read and executed one by one to build the image

When writing a Dockerfile, one should follows the following best practices:

- Containers should be ephemeral or stateless, in that they can be reinstantiated with a minimum of set up and configuration.
- Avoid installing unnecessary packages.
- Each container should have only one purpose (ex: a database, a web server, etc.).
- The numbers of layers should be minimized.

About statelessness:

- An application may run more than one container at a time
  - All these containers should be stateless.
- If any information is to be persisted beyond the termination of the container, this information should be written to a stateful backing service such as a database (like MongoDB or Postgres) or a key-value store (like Redis).
- Indeed: We should be able to shut down a container and remove it from our system, then start an identical container using the same image, and run command with no effect to our work.

The first thing you have to do is to create a file named "Dockerfile", then define in it the image you are going to use as a base, thanks to the FROM statement. In our case, we will use a Ubuntu base image[2].

```
FROM ubuntu
```

The FROM statement can only be used once in a Dockerfile!
You can add the name and coordinate of the maintainer of the Dockerfile with:

```
MAINTAINER Firstname Lastname (mail@mymail.com)
```

[2] To find the list of the docker images you can use as base images you can go here: *https* : *//hub.docker.com/*

Then you can use the RUN instruction to execute command lines in your container, for example updating the Ubuntu system and installing mongodb:

```
RUN apt-get update
RUN apt-get install mongodb
```

Notes

apt-get is a Ubuntu/Debian package manager, you can use it to search, install, update or remove software. The package manager your can use depend on the base image you use (for instance, of your base image is archlinux, you have to use pacman).

The ENV instruction permits to set or update environment variables (like PATH, etc).

Then use the ADD statement to copy or upload files to the image. In our case, we use it to add the sources of our local application in the /app/ folder of the image.

```
ADD . /PATH_IN_CONTAINER/
```

Then use the WORKDIR command to change the current directory. The command is equivalent to a cd command on the command line. All the commands that follow will be executed from the defined directory.

```
WORKDIR /PATH_IN_CONTAINER
```

Now that the source code and dependencies are present in your container, we need to tell our image some final information.

```
EXHIBIT 27017
VOLUME /PATH_IN_CONTAINER_TO_SOME_REPOSITORY
```

The EXPOSE statement is used to indicate the port on which your application is listening. The VOLUME statement is used to indicate which directory you want to share with your host.

We will conclude with the instruction that must always be present, and place it in the last line for better understanding: CMD. It allows our container to know which command it should execute when it starts.

```
CMD sudo systemctl start mongodb
```

Summary of the Dockerfile:

```
FROM ubuntu
MAINTAINER Firstname Lastname (mail@mymail.com)
RUN apt-get update
RUN apt-get install mongodb
ADD . /app/
WORKDIR /app
EXPOSE 27017
VOLUME /app/logs
CMD  sudo systemctl start mongodb
```

You can now create your first own Docker image!
Type this command in the repository containing the Dockerfile:

```
$ docker build -t my-mongodb-image .
```

The -t argument allows you to give a name to your Docker image. This makes it easier to find your image later. The . is the directory where the Dockerfile is located; in our case, at the root of our project.

Now you can launch your container with the docker run command :

```
$ docker run -d -p 27017:27017 my-mongodb-image
```

1. What is Docker?
2. Why using Docker?
3. How Docker works?
4. Let's see how to use Docker!
5. Let's make your own Docker image!
   - Best practices
   - How to create a Dockerfile?

sqsq