Docker Basics — AL — ESIR

Erwan Bousse

April 13, 2015

Abstract

This lab session aims at introducing *Docker* through the notion of *images* and *containers*. We will go through basic commands and deploy a simple web application on a machine using two containers.

1 About Docker

Docker is a free-software project that aims at providing facilities to deploy applications using *containers*. A container is a GNU/Linux-based operating system that can be started on top of some existing running Linux kernel. The idea is to put into a container 100% of the dependencies of a particular application (libraries, tools, etc.) so that this application can be deployed in any environment without suffering from side effects — except the Linux kernel, which is shared with the host. Docker containers use their own virtual network, which can be accessed from the host OS in order to create a bridge with the physical network. Figure 1 illustrates such situation.

To define containers, Docker relies on the notion of *image*. If we compare images and containers with object-oriented programming, an image is to a class what a container is to an object. An image defines the self-contained application, and can be "instantiated" (possibly multiple times) into a container to run the application. Docker provides a repository of images called Docker Hub Registry[1], which can be used both to store your own images and to use existing images of other users or groups. Note that when you use the command `docker pull`, you use this repository by default to retrieve images.

The same way a constructor of a class requires arguments, creating an container requires arguments: a starting command (not mandatory if the image was created with an *entry point*), a name (not mandatory), links to other containers (not mandatory), and any other option used. Creating a container is done using the command `run` of Docker. Each time you do `docker run <image name> [<starting command>]`, a container is instantiated and the command is executed, until it stops when the command is over. Running `docker ps -a` allows you to see all the containers you have, even those that finished their job. You might have to regularly clean up and delete useless containers using `docker rm <container id>`.

[1] `https://registry.hub.docker.com/`

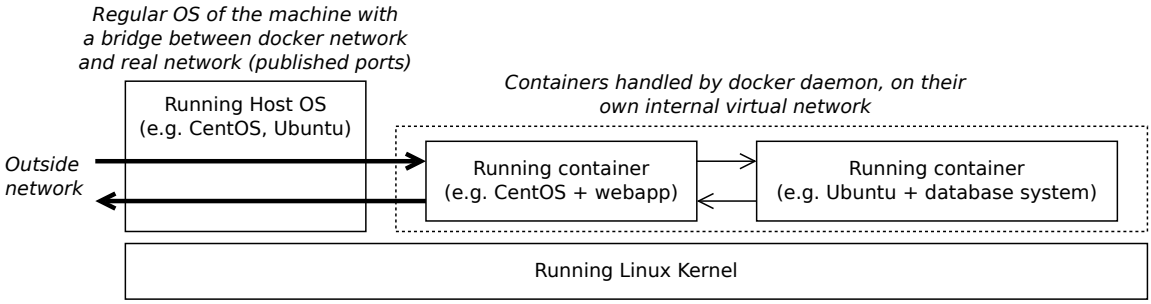

Figure 1: Overview of Docker on top of a running Linux Kernel

2 Preparing Your Server

Since you don't have a `root` access on computers for lab sessions, we will have to rely on a virtual machine to play with Docker.

1. Go to `http://vm.istic.univ-rennes1.fr/`, log in, and click on "Demander une VM" in the top-right corner. Give your machine a name, write your teacher name, choose a validity of only 1 month, and select the OS "Ubuntu 14".
2. Validate. While waiting for the machine to be ready, you can do Section 3 "Official Docker Tutorial", since it doesn't require a real machine ☺.
3. When the machine is ready, connect using `ssh` (GNU/Linux) or PuTTY (Windows) to your machine as the user `root` with the password `projet`. The hostname of your VM should be: `<name of your VM>.istic.univ-rennes1.fr`.
4. Once logged in, change the password of `root` with the command: `passwd`.
5. Install docker with the command: `apt-get update && apt-get install docker.io`
6. Done! You can now pull images, create containers, and whatsoever, using the `docker` command.

3 Official Docker Tutorial

Before playing with Docker on a real (virtual) machine, you must first go through the steps 1–7 of the official Docker tutorial available here: `https://www.docker.com/tryit/`. You don't have to do Step 8 since it just asks you to create an account. Please read carefully all the given explanations! Also keep in mind what was explained in the first section, and don't forget to ask questions if you don't understand what's happening!

Note that everything related to commits and creating a new image from an existing container will not be used in the following exercise, but is important to know and may be used in the next lab session.

4 Exercise: Deploying a Web Application

In this part we are going to use two existing docker images to deploy a web application on the machine. The first image is called `redis`, which contains a Redis[2] database system. The second image is called `mdms`, and contains a very simple web application. Note that the image `mdms` is pre-configured to communicate in the docker virtual network with a host named `mdms-redis`, containing a Redis database.

If you are stuck, go have a look here `http://docs.docker.com/reference/commandline/cli/` or use the `docker` command for which you want more information using the `--help` option.

1. First, download the official Redis Docker image, from the repository named `redis` (no username is required since this is an "Official Repo" of a company or a group).

2. You now have to create a first container using this image, and to give this container a name you choose. Write that name down because you will need it later. Also be aware that:

   - The image already has a predefined entry point, which starts a redis server. Therefore you don't need to give a command to the container.
   - The database server needs to run *in the background*, so that we can start a second container from the same terminal. Find the option to do so by simply calling `docker run` to see all available options, and use the option.

   Once you figured out the correct command to create the container, create it!

3. Now, download the image that contains the webapp called MDMS that we want to deploy. On the Docker Hub Registry, the user that created this image is named `maxleiko`, and the repository of the image is called `mdms`.

4. In order to create a container with the `mdms` image, take into account the following points:

   - As explained previously, the `mdms` image is configured to talk to a machine named `mdms-redis`. Therefore we will have to create a container with a *link* to the Redis container, using the option `--link <container name>:<alias>`. The *alias* is the name through which our new container will be able to reach our Redis container.
   - We need to be able to access to this container from the physical network, so that the web page is accessible from your current computer. To do so, we have to publish the port `8080` of the container to the host, so that a connection to the port `80` of the host reaches directly the port `8080` of the container. This can be achieved with the option `-p <hostPort>:<containerPort>`
   - This time we want to follow the output of the process, and we want to keep a hand on it. Therefore we don't want to put the application in the background. However, to be able to interact with the process (for instance `CTRL+C`), we need to do two things: to allocate a pseudo terminal to the process using the `-t` option, and to allow interactions with this terminal using the `-i` option. If you forget these options, you will have to create a second SSH connection to your virtual machine in order to ask docker to stop the container.

   Again, once you figured out the correct command to create the container, have a try!

[2] `http://redis.io/`

5. Once everything is running, use your web browser to go to `http://<name of your VM>.istic.univ-rennes1.fr`. A web page should appear. You can login with "admin:admin" to play with the app and create some notes using markdown.