Docker
Imagen Contenedor

Raúl Estrada

Octubre 2020

1. Overview

Docker is a tool for creating, deploying, and running applications easily. It allows us to package our applications with all the dependencies, and distribute them as individual bundles. Docker guarantees that our application will run in the same way on every Docker instance.

When we start using Docker, there are **two main concepts** we need to be clear on — **images and containers**.

In this tutorial, we'll learn what they are and how they differ.

2. Docker Images

An image is a file that represents a packaged application with all the dependencies needed to run correctly. In other words, we could say that a **Docker image is like a Java class**.

Images are built as a series of layers. Layers are assembled on top of one another. So, what is a layer? Simply put, a layer is an image.

Let's say we want to create a Docker image of a Hello World Java application. The first thing we need to think about is what does our application need.

To start, it is a Java application, so we will need a JVM. OK, this seems easy, but what does a JVM need to run? It needs an Operating System. Therefore, **our Docker image will have an Operating System layer, a JVM, and our Hello World application**.

A major advantage of Docker is its large community. If we want to build on to an image, we can go to Docker Hub and search if the image we need is available.

Let's say we want to create a database, using the PostgreSQL database. We don't need to create a new PostgreSQL image from scratch. We just go to Docker Hub, search for *postgres*, which is the Docker official image name for PostgresSQL, choose the version we need, and run it.

Every image we create or pull from Docker Hub is stored in our filesystem and is identified by its name and tag. It can also be **identified by its image id**.

Using the *docker images* command, we can view a list of images we have available in our filesystem:

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
postgres            11.6                d3d96b1e5d48        4 weeks ago         332MB
mongo               latest              9979235fc504        6 weeks ago         364MB
rabbitmq            3-management        44c4867e4a8b        8 weeks ago         180MB
mysql               8.0.18              d435eee2caa5        2 months ago        456MB
jboss/wildfly       18.0.1.Final        bfc71fe5d7d1        2 months ago        757MB
flyway/flyway       6.0.8               0c11020ffd69        3 months ago        247MB
java                8-jre               e44d62cf8862        3 years ago         311MB
```

3. Running Docker Images

An image is run using the *docker run* command with the image name and tag. Let's say we want to run the postgres 11.6 image:

```
1 docker run -d postgres:11.6
```

Notice we provided the *-d* option. This tells Docker to run the image in the background — also known as the detached mode.

Using the *docker ps* command we can check if our image is running we should use this command:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
3376143f0991        postgres:11.6       "docker-entrypoint.s…"   3 minutes ago       Up 3 minutes
5432/tcp            tender_heyrovsky
```

Notice the *CONTAINER ID* in the output above. Let's take a look at what a container is and how it is related to an image.

4. Docker Containers

A container is an instance of an image. Each container can be identified by its ID. Going back to our Java development analogy, we could say that **a container is like an instance of a class**.

Docker defines seven states for a container: *created*, *restarting*, *running*, *removing*, *paused*, *exited*, and *dead*. This is important to know. Since a container is just an instance of the image, it doesn't need to be running.

Now let's think again about the *run* command we have seen above. We have said it is used to run images, but that is not totally accurate. The truth is that the *run* command is used to *create* and *start* a new container of the image.

One big advantage is that containers are like lightweight VMs. Their behaviors are completely isolated from each other. This means that we can run multiple containers of the same image, having each one in a different state with different data and different IDs.

Being able to run multiple containers of the same image at the same time is a great advantage because it allows us an easy way of scaling applications. For example, let's think about microservices. If every service is packaged as a Docker image, then that means that new services can be deployed as containers on demand.

5. Containers Lifecycle

Earlier, we mentioned the seven states of a container, Now, let's see how we can use the *docker* command-line tool to process the different lifecycle states.

Starting up a new container requires us to *create* it and then *start* it. This means that it has to go through the create state before it can be running. We can do this by creating and starting the container explicitly:

```
docker container create <image_name>:<tag>
docker container start <container_id>
```

Or we can easily do this with the *run* command:

```
docker run <image_name>:<tag>
```

We can pause a running container and then put it on running state again:

```
docker pause <container_id>
docker unpause <container_id>
```

A paused container will show "Paused" as the status when we check the processes:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
PORTS               NAMES
9bef2edcad7b        postgres:11.6       "docker-entrypoint.s…"   5 minutes ago       Up 4 minutes (Paused)
5432/tcp            tender_heyrovsky
```

We can also stop a running container and then rerun it:

```
docker stop <container_id>
docker start <container_id>
```

And finally, we can remove a container:

```
docker container rm <container_id>
```

Only containers in the stopped or created state can be removed.

For more information regarding the Docker commands, we can refer to the Docker Command Line Reference.

6. Conclusion

In this article, we discussed Docker images and containers and how they differ. Images describe the applications and how they can be run. Containers are the image instances, where multiple containers of the same image can be run, each in a different state.

We have also talked about the containers' lifecycle and learned the basic commands to manage them.

Now that we know the basics, it's time to learn more about the exciting world of Docker and to start increasing our knowledge!