What is Docker and Why is it So Popular?

Docker is going to revolutionize how we approach application development and deployment; trust me. Unfortunately, it's difficult to get started due to the fact that all the resources out there are either too technical, or are lacking in the intuition behind why Docker is amazing. I hope this post finds a nice balance between the two.

What is Docker?

Docker is a technology that wraps Linux's operating-system-level virtualization. This is an extremely verbose way of saying that Docker is a tool that quickly creates isolated environments for you to develop and deploy applications in. Another magical feature of Docker is that it allows you to run ANY linux-based operating system within the container — offering even greater flexibility. If you need to run one application with CentOS and another with Ubuntu — no problem!

Containers and Images

Docker uses the term Containers to represent an actively running environment that can run almost any piece of software; whether it be a web application or a service daemon.

Docker Images are the "building blocks" that act as the starting point for our containers. Images become containers, and containers can be made into images. Images are typically base operating system images (such as Ubuntu), but they can also be highly-customized images containing the base OS plus any additional dependencies you install on it.

Alright, enough talk. It's time to get our hands dirty.

Installation

You will need to be running either Linux or Mac to use Docker. Installation instructions can be found here. One thing to note: I am running Docker on Mac OS X. The only major difference between running Docker commands on a Linux machine versus a Mac machine, is that you have to sudo your commands on Linux. On Mac (like you will see below), you do not have to. There is a bit more set-up involved in running Docker on a Mac (you actually run commands through a Linux VM), but once set-up, they behave the same.

Hello Docker!

Let's start with a super basic Docker interaction and step through it piece-by-piece.

```
$ docker run ubuntu:14.04 /bin/echo 'Hello Docker!'
```

This command does a number of things:

1. The docker run takes a base image and a command to run inside a new container created from that image.
2. Here we are using the "ubuntu" base image and specifying a tag of "14.04″. Tags are a way to use a specific version of an image. For example, we could have run the above command with ubuntu:12.04. Docker images are downloaded if they do not exist on your local machine already, so the first time you run the above command it will take a bit longer.
3. Finally, we specify the command to run in the newly instantiated container. In this simple case, we are just echoing Hello Docker.

If run successfully, you should see 'Hello Docker!' printed to your terminal. Congrats! You just created a fresh Docker container and ran a simple command inside…feel the power!

An 'Interactive' Look at Docker

Now let's get fancy and poke around inside of a container. Issue the following command, and you will be given a shell inside a new container.

```
$ docker run -i -t ubuntu:14.04 /bin/bash
```

The –i and –t flags are a way of telling Docker that you want to “attach” to this container and directly interact with it.

Once inside the container, you can run any command you would normally run on a Ubuntu machine. Play around a bit and explore. Try installing software via apt-get, or running a simple shell-script. When you are done you can type exit to leave.

One thing to note: containers are only active as long as they are running a service or command. In the case above, the minute we exited from our container, it stopped. I will show you how to keep containers running shortly. This does, however, highlight an important paradigm shift. Docker containers are really best used for small, discrete services. Each container should run one service or command and cease to exist when it has completed its task.

A Daemonized Docker Service

Now that you know how to run commands inside of a Docker container, let’s create a small service that will run indefinitely in the background. This container will just print “Hello Docker” until you explicitly tell it to stop.

```
docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo
 Hello Docker; sleep 1; done"
```

Let’s step through this command:

1. As you know by now, this command runs a small shell command inside of a container running ubuntu 14.04.
2. The new addition is the –d flag, which tells Docker to run this container in the background as a daemon.
3. To verify this container is actually running, issue the command docker ps, which gives you some helpful information about our running containers.

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND               CREATED
   STATUS       PORTS NAMES
09e17400c7ee  ubuntu:14.04  /bin/sh -c 'while tr  2 minutes
```

```
 ago  Up 1 minute        sleepy_pare
```

You can view the output of your command by taking the NAME and issuing the following command:

```
$ docker logs sleepy_pare
Hello Docker
Hello Docker
Hello Docker
```

To stop the container and remove it, you can run docker rm –f sleepy_pare.

Congratulations! You have learned the basic concepts and usage of Docker. There is still a wealth of features to learn, but this should set a strong foundation for your own further exploration into the awesome world of Docker.

Want more Docker? Our dedicated Docker page **has got you covered. Featuring our latest titles and even more free content, make sure you check it out!**

From the 4th to the 10th of April save 50% on 20 of our very best cloud titles. From AWS to Azure. OpenStack and, of course, Docker, we're confident you'll find something useful. And if one's not enough, pick up any 5 featured titles for just $50! Find them here.

About the Author

Julian Gindi is a Washington DC-based software and infrastructure engineer. He currently serves as Lead Infrastructure Engineer at iStrategylabs where he does everything from system administration to designing and building deployment systems. He is most passionate about Operating System design and implementation, and in his free time contributes to the Linux Kernel.
