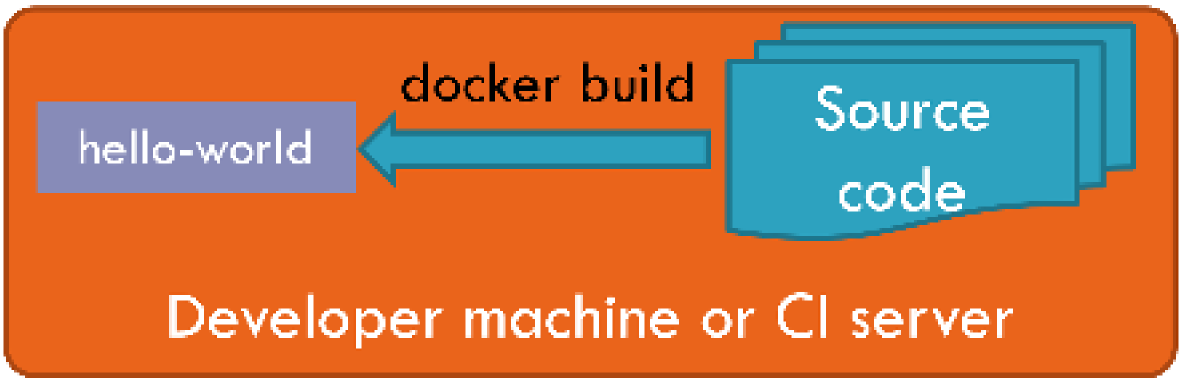
Creating a Simple Image

Up until now, we have been using ready-made images from Docker Hub. In this lesson you will learn how to make your first image.

As we saw earlier, containers are created from images. Up until now, we've been using images created by others. It's high time we learned how to create our own images. Inside our images, we can stuff our programs and their dependencies so that multiple containers can be created from those images and live happily ever after.

Creating a Simple Image



Using `docker build` to create an image

A Docker image is created using the *docker build* command and a *Dockerfile* file. The *Dockerfile* file contains instructions on how the image should be built.

> The *Dockerfile* file can have any name. Naming it *Dockerfile* makes it easier for others to understand its purpose when they see that file in your project. It also means we don't need to state the file name when using the *docker build* command.

using the *docker build* command.

I’d like to create a basic image for a container that displays a “hello world” message when its run.

For this, I create a file named *Dockerfile* that describes how my image should be built. A *Dockerfile* file always begins with a *FROM* instruction because every image is based on another base image. This is a powerful feature since it allows you to extend images that may already be complex.

As I only need a simple text output, I can use a Debian Linux image. Here’s my *Dockerfile* file:

```
FROM debian:8
```

This is not enough. While I do get a Debian Linux basis, I am not running any command that could display “hello world.” This can be achieved using the *CMD* instruction. The *CMD* instruction specifies which executable is run when a container is created using your image and provides optional arguments.

Here’s an improved *Dockerfile* file that creates a Debian Linux-based image and instructs it to greet our users when a container spawns:

Dockerfile

```
FROM debian:8

CMD ["echo", "Hello world"]
```

Note that both the program to run and its arguments are provided as a JSON array of strings.

In order to create an image from my *Dockerfile* file, I need to run the *docker build* command. To do this, I type the following command in my terminal in the folder where the *Dockerfile* file lives:

```
docker build -t hello .
```

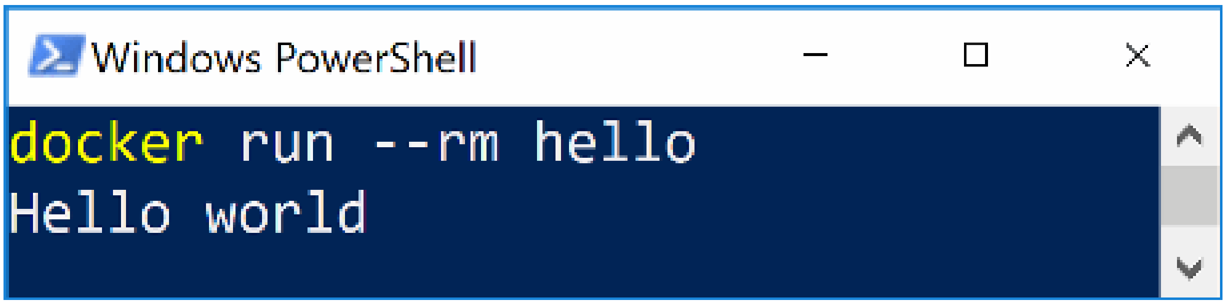
The `-t` switch is used in front of the desired image. An image can be created without a name, it would have an auto-generated unique ID, so it is an optional parameter on the *docker build* command.

> Note the dot at the end of the command above. It specifies which path is used as the build context (more about that later), and where the *Dockerfile* is expected to be found. Should my *Dockerfile* have another name or live elsewhere, I can add a `-f` switch in order to provide the file path.

The *docker build* command just created an image named *hello*. That image is stored locally on my computer, and I can run it as I would any other image:

```
docker run --rm hello
```

As expected, the *docker run* command above simply prints the message:



```
docker run --rm hello
Hello world
```

From here, you may want to publish your image for others to run containers based on it. We'll see how to do that in the *Publish Docker Images* chapter but for now, I want to tell you more about creating images.

Just to make things crystal clear, here's what I did:

- Create an image:
    - Create a file named *Dockerfile*
    - Run a *docker build* command
- Run a container from the image created

Again, I'd like to expand on what this means; running a container is the virtual equivalent of starting a brand-new machine and then trashing it. In order to print that “Hello world” message, we essentially got a new computer, had it execute an *echo* command, and then trashed it. Docker makes fire-and-forget computing cheap. Of course, this is overkill for such a simple purpose, but it remains true even when we install frameworks or move files around inside our containers; it’s a fantastic feature. Most of your Docker power will come when you understand how easily you can create and trash isolated virtual computers.

In the next lesson, we will learn how to include files in our images.