Running Your First Image

To test Docker is installed correctly, try running:

$ **docker run debian echo "Hello World"**

This may take a little while, depending on your Internet connection, but eventually

you will get something similar to the following:

We can ask Docker to give us a shell inside a container with the following command:

$ **docker run -i -t debian /bin/bash**

root@622ac5689680:/# **echo "Hello from**

First, let’s launch a new container; but this

time, we’ll give it a new hostname with the -h flag:

$ **docker run -h CONTAINER -i -t debian /bin/bash**

We’ve moved the */bin* directory and made the container pretty useless, at least temporarily.

2 Before we get rid of this container, let’s see what the ps, inspect, and diff

commands tell us about it. Open a new terminal (leave the container session running),

and try running docker ps from the host. You will see something like this:

CONTAINER ID IMAGE COMMAND ... NAMES

00723499fdbf debian "/bin/bash" ... stupefied\_turing

This tells us a few details about all the currently running containers. Most of the output

should be self-explanatory, but note that Docker has given the container a readable

name that can be used to identify it from the host, in this case

"stupefied\_turing“.3 We can get more information on a given container by running

docker inspect with the name or ID of the container:

$ **docker inspect stupefied\_turing**

[

{

"Id": "00723499fdbfe55c14565dc53d61452519deac72e18a8a6fd7b371ccb75f1d91",

"Created": "2015-09-14T09:47:20.2064793Z",

"Path": "/bin/bash",

"Args": [],

"State": {

"Running": true,

...

There is a lot of valuable output here, but it’s not exactly easy to parse. We can use

grep or the --format argument (which takes a Go template4) to filter for the information

we’re interested in. For example:

$ **docker inspect stupefied\_turing | grep IPAddress**

"IPAddress": "172.17.0.4",

"SecondaryIPAddresses": null,

$ **docker inspect --format {{.NetworkSettings.IPAddress}} stupefied\_turing**

172.17.0.4

Both give us the IP address of the running container. But for now, let’s move

The last thing I want to show you before we’re finished with this container is docker

logs. If you run this command with the name of your container, you will get a list of

everything that’s happened inside the container:

$ **docker logs stupefied\_turing**

root@CONTRAINER:/# mv /bin /basket

To get rid of the

container, use the docker rm command:

$ **docker rm stupefied\_turing**

stupefied\_turing

22 | Chapter 3: First Steps

OK, let’s see how we can build a new, useful container we actually want to keep.5

We’re going to create a Dockerized cowsay application. If you don’t know what cowsay

is, I suggest you brace yourself. Start by launching a container and installing some

packages:

$ **docker run -it --name cowsay --hostname cowsay debian bash**

root@cowsay:/# **apt-get update**

...

Reading package lists... Done

root@cowsay:/# **apt-get install -y cowsay fortune**

Give it a whirl!

root@cowsay:/# **/usr/games/fortune | /usr/games/cowsay**

Excellent. Let’s keep this container.6 To turn it into an image, we can just use the

docker commit command. It doesn’t matter if the container is running or stopped. To

do this, we need to give the command the name of the container (“cowsay”) a name

for the image (“cowsayimage”) and the name of the repository to store it in (“test”):

root@cowsay:/# exit

exit

$ **docker commit cowsay test/cowsayimage**

d1795abbc71e14db39d24628ab335c58b0b45458060d1973af7acf113a0ce61d

The returned value is the unique ID of our image. Now we have an image with cowsay

installed that we can run:

$ **docker run test/cowsayimage /usr/games/cowsay "Moo"**

Building Images from Dockerfiles

A Dockerfile is simply a text file that contains a set of steps that can be used to create

a Docker image. Start by creating a new folder and file for this example:

$ **mkdir cowsay**

$ **cd cowsay**

$ **touch Dockerfile**

And insert the following contents into *Dockerfile*:

**FROM** debian:wheezy

**RUN** apt-get update && apt-get install -y cowsay fortune

We can now build the image by running the docker build command inside the same

directory:

$ **ls**

Dockerfile

$ **docker build -t test/cowsay-dockerfile .**

Then we can run the image in the same way as before:

$ **docker run test/cowsay-dockerfile /usr/games/cowsay "Moo"**

Connecting Containers to the World

Say you’re running a web server inside a container. How do you provide the outside

world with access? The answer is to “publish” ports with the -p or -P commands. This

command forwards ports on the host to the container. For example:

$ **docker run -d -p 8000:80 nginx**

af9038e18360002ef3f3658f16094dadd4928c4b3e88e347c9a746b131db5444

$ **curl localhost:8000**

<!DOCTYPE html>

<html>

<head>

<title>Welcome to nginx!</title>

...

The -p 8000:80 argument has told Docker