Docker Basics

by Felix Kolwa

Prerequisites

- Basic command line of your operating system in this case linux
- Basic bash knowledge

Introduction

This article will be covering Docker including the following topics:

- Installing Docker
- Raising containers from the command line
- Managing containers and reading log output
- Using Dockerfiles
- Scouting docker hub for fun and profit

What is Docker

Maybe this sounds familiar. You have been assigned a task in which you had to deploy a complex software onto an existing infrastructure. As you know there are a lot of variables to this which might be out of your control. The operating system, pre existing dependencies, probably even interfering software. Even if the environment is perfect at the moment of the deployment what happens after you are done? Living systems constantly change. New software is introduced while old and outdated software and libraries are getting removed. Parts of the system that you rely on today might be gone tomorrow.

This is where virtualization comes in. It used to be best practice to create isolated virtual computer systems, so called virtual machines (VMs), which simulate independent systems with their own operating systems and libraries. Using these VMs you can run any kind of software in a separated and clean environment without the fear of collisions with other parts of the system. You can emulate the exact hardware you need, install the OS you want and include all the software you are dependent on at just the right version. It offers great flexibility. It also means that these VMs are very demanding on your host system. The hardware has to be strong enough to create virtual hardware for your virtual systems. They also have to be created and installed for every virtual system that you are using. Even though they might run on the same host sharing resources between them is just as inconvenient as with real machines.

Introducing the container approach and one of their main competitors, Docker. Simply put, Docker enables you to isolate your software into containers. The only thing you need is a running instance of Docker on your host. Even better: All the necessary resources like OS and libraries cannot only be deployed with your software, they can even be shared between individual instances of your containers running on the same system! This is a big improvement above regular VMs. Sounds too good to be true?

Well, even though Docker comes with everything you need, it is still up to you to assure consistency and reproducibility of your own containers. In the following article I will slowly introduce you to Docker and give you the basic knowledge necessary to be part of the containerized world.

Getting Docker

Before we can start creating containers we first have to get Docker running on our system. Docker is available for Linux, Mac and just recently for Windows 10. Just choose the version that is right for you and come back right here once you are done:

- Linux(Ubuntu)
- Windows 10
- Mac

Please notice that the official documentation contains instructions for multiple Linux distributions, so just choose the one that fits your needs.

Even though the workflow is very similar for all platforms, the rest of the article will assume that you are running an Unix environment. Commands and scripts can vary when you are running on Windows 10.

Your first container

Got Docker installed and ready to go? Great! Let's get our hands on creating the first container. Most tutorials will start off by running the tried and true „Hello World" example but chances are you already did it when you were installing Docker.

So lets start something from scratch! Open your shell and type the following:

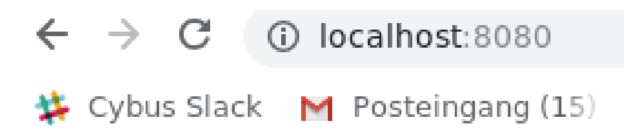
```
docker run -p 8080:80 httpd
```

If everything went well you will get a response like this:

```
Unable to find image ‚httpd:latest' locally
 latest: Pulling from library/httpd
 f17d81b4b692: Pull complete
 06fe09255c64: Pull complete
 0baf8127507d: Pull complete
 07b9730387a3: Pull complete
 6dbdee9d6fa5: Pull complete
 Digest: sha256:90b34f4370518872de4ac1af696a90d982fe99b0f30c9be994964f49a6e2f421
 Status: Downloaded newer image for httpd:latest
  AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.2. Set the ‚ServerName' directive globally to suppress this
message
```

```
  AH00558: httpd: Could not reliably determine the server‘s fully qualified domain
name, using 172.17.0.2. Set the ‚ServerName‘ directive globally to suppress this
message
  [Mon Nov 12 09:15:49.813100 2018] [mpm_event:notice] [pid 1:tid 140244084212928]
AH00489: Apache/2.4.37 (Unix) configured -- resuming normal operations
  [Mon Nov 12 09:15:49.813536 2018] [core:notice] [pid 1:tid 140244084212928] AH00094:
Command line: ‚httpd -D FOREGROUND‘
```

Now there is a lot to go through but first open a browser and head over to `localhost:8080`.



It works!

Yeah, we just did that!
What we just achieved is we set up and started a simple http server locally on port 8080 within less than 25 typed characters. But what did we write exactly? Let's analyze the command a bit closer:

- `docker` – This states that we want to use the Docker command line interface (CLI).
- `run` – The first actual command. It states that we want to run a command in a new container.
- `-p 8080:80` – The publish flag. Here we declare what Docker internal port (our container) we want to publish to the host (the pc you are sitting at). the first number declares the port on the host (8080) and the second the port on the Docker container (80).
- `httpd` – The image we want to use. This contains the actual server logic and all dependencies.

IMAGES
Okay, so what is an image and where does it come from? Quick answer: An image is a template that contains instructions for creating a container. Images can be hosted locally or online. Our httpd image was hosted on the Docker Hub. We will talk more about the official docker registry in the Exploring the Docker Hub part of this lesson.

HELP
The Docker CLI contains a thorough manual. So whenever you want more details about a certain command just add --help behind the command and you will get the man page regarding the command.

Great! Now that we understand what we did we can take a look at the output.

```
Unable to find image ‚httpd:latest‘ locally
 latest: Pulling from library/httpd
 f17d81b4b692: Pull complete
 06fe09255c64: Pull complete
 0baf8127507d: Pull complete
 07b9730387a3: Pull complete
 6dbdee9d6fa5: Pull complete
 Digest: sha256:90b34f4370518872de4ac1af696a90d982fe99b0f30c9be994964f49a6e2f421
 Status: Downloaded newer image for httpd:latest
```

The `httpd` image we used was not found locally so Docker automatically downloaded the image and all dependencies for us. It also provides us with a `digest` for our just created container. This string starting with `sha256` can be very useful for us! Imagine that you create software that is based upon a certain image. By binding the image to this digest you make sure that you are always pulling and using the same version of the container and thus ensuring reproducibility and improving stability of your software.
While the rest of the output is internal output from our small webserver you might have noticed that the command prompt did not return to input once the container started. This is because we are currently running the container in forefront. All output that our container generates will be visible in our shell window while we are running it. You can try this by reloading the webpage of our webserver. Once the connection is reestablished the container should log something similar to this:

```
172.17.0.1 - - [12/Nov/2018:09:17:12 +0000] „GET / HTTP/1.1“ 304 -
```

You might have also noticed that the ip address is not the one from your local machine. This is because Docker creates containers in their own Docker network. Explaining Docker networks is out of scope for this tutorial so I will just redirect you to the official documentation about Docker networks for the time being.

For now, stop the container and return to the command prompt by pressing ctrl+c while the shell window is in focus.

Managing containers

Detaching containers

Now that we know how to run a container it is clear that having them run in an active window isn't always practical. Let's start the container again but this time we will add a few things to the command:

```
docker run --name serverInBackground -d -p 8080:80 httpd
```

When you run the command you will notice two things: First the command will execute way faster then the first time. This is because the image that we are using was already downloaded the last time and is currently hosted locally on our machine. Second, there is no output anymore besides a strange string of characters. This string is the ID of our container. It can be used to refer to its running instance.

So what are those two new flags?

- `--name` – This is a simple one. It attaches a human readable name to our container instance. While the container ID is nice to work with on a deeper level, attaching an actual name to it makes it easier to distinguish between running containers for us as human beings. Just keep in mind that IDs are unique and your attached name might not!
- `-d` – This stands for detach and makes our container run in the background. It also provides us with the container ID.

SHARING RESOURCES

If you want to you can execute the above command with different names and ports as many times as you wish. While you can have multiple containers running httpd they will all be sharing the same image. No need to download or copy what already have on your host.

Listing containers

So now that we started our container make sure that it is actually running. Last time we opened our browser and accessed the webpage hosted on the server. This time let's take another approach. Type the following in the command prompt:

```
docker ps
```

The output should look something like this:

```
CONTAINER ID    IMAGE    COMMAND              CREATED            STATUS          PORTS
NAMES
  018acb9dbbbd     httpd     „httpd-foreground"    11 minutes ago       Up 11 minutes
0.0.0.0:8080->80/tcp    serverInBackground
```

- `ps` – The ps command prints all running container instances including information about ID, image, ports and even name. The output can be filtered by adding flags to the command. To learn more just type `docker ps --help`.

Inspecting containers

Another important ability is to get low level information about the configuration of a certain container. You can get these information by typing:

```
docker inspect serverInBackground
```

Notice that is doesn't matter if you use the attached name or the container ID. Both will give you the same result.

The output of this command is huge and includes everything from information about the image itself to network configuration.

HINT

You can execute the same command using a image id to inspect the template configuration of the image.

To learn more about inspecting docker containers please refer to the official documentation.

Crawling inside the container

We can even go in deeper and interact with the internals of the container. Say we want to try changes to our running container without having to shut it down and restart it every time. So how do we approach this?

Like a lot of Docker images, httpd is based upon a Linux image itself. In this case httpd is running a slim version of Debian in the background. So being a Linux system we can access a shell inside the container. This gives us a working environment that we are already familiar with. Let's jump in and try it:

```
docker exec -it -u 0 serverInBackground bash
```

There are a few new things to talk about:

- `exec` – This allows us to execute a command inside a running container.
- `-it` – These are actually two flags. `-i -t` would have the same result. While i stands for interactive (we need it to be interactive if we want to use the shell) t stands for TTY and creates a pseudo version of the Teletype Terminal. A simple text based terminal.
- `-u 0` – This flag specifies the UID of the user we want to log in as. 0 opens the connection as root user.
- `serverInBackground` – The container name (or ID) that we want the command to run in.

- `bash` – At the end we define what we actually want to run in the container. In our case this is the bash environment. Notice that bash is installed in this image. This might not always be the case! To be safe you can add `sh` instead of `bash`. This will default back to a very stripped down shell environment by default.

When you execute the command you will see a new shell inside the container. Try moving around in the container and use commands you are familiar with. You will notice that you are missing a lot of capabilities. This has to be expected on a distribution that is supposed to be as small as possible. Thankfully httpd includes the `apt` packaging manager so you can expend the capabilities. When you are done, You can exit the shell again by typing `exit`.

Getting log output

Sometimes something inside your containers just won't work and you can't find out why by blindly stepping through your configuration. This is where the Docker logs come in.

To see logs from a running container just type this:

```
docker logs serverInBackground -f --tail 10
```

Once again there are is a new command and a few new flags for us:

- `logs` – This command fetches the logs printed by a specific container.
- `-f` – Follow the log output. This is very handy for debugging. With this flag you get a real time update of the container logs while they happen.
- `--tail` – Chances are your container is running for days if not months. Printing all the logs is rarely necessary if not even bad practice. By using the
- `tail` flag you can specify the amount of lines to be printed from the bottom of the file.

You can quit the log session by pressing `ctrl+c` while the shell is in focus.

Stopping a detached container

If you have to shutdown a running container the most graceful way is to stop it. The command is pretty straight forward:

```
docker stop serverInBackground
```

This will try to shutdown the container and kill it, if it is not responding. Keep in mind that the stopped container is not gone! You can restart the container by simply writing

```
docker start serverInBackground
```

Killing the container – a last resort

Sometimes if something went really wrong, your only choice is to take down a container as quickly as possible.

```
docker kill serverInBackground
```

NOTE
Even though this will get the job done, killing a container might lead to unwanted side effects due to not shutting it down correctly.

Removing a container
As we already mentioned, stopping a container does not remove it. To show that a stopped container is still managed in the background just type the following:

```
docker container ls -a
```

- `container` – This accesses the container interaction.
- `ls` – Outputs a list of containers according to the filters supplied.
- `-a` – Outputs all containers, even those not running.

```
CONTAINER ID     IMAGE     COMMAND                  CREATED                  STATUS
PORTS     NAMES
ee437314785f    httpd    „httpd-foreground“  About a minute ago   Exited (0) 8 seconds
ago              serverInBackground
```

As you can see even though we stopped the container it is still there. To get rid of it we have to remove it. Just run this command:

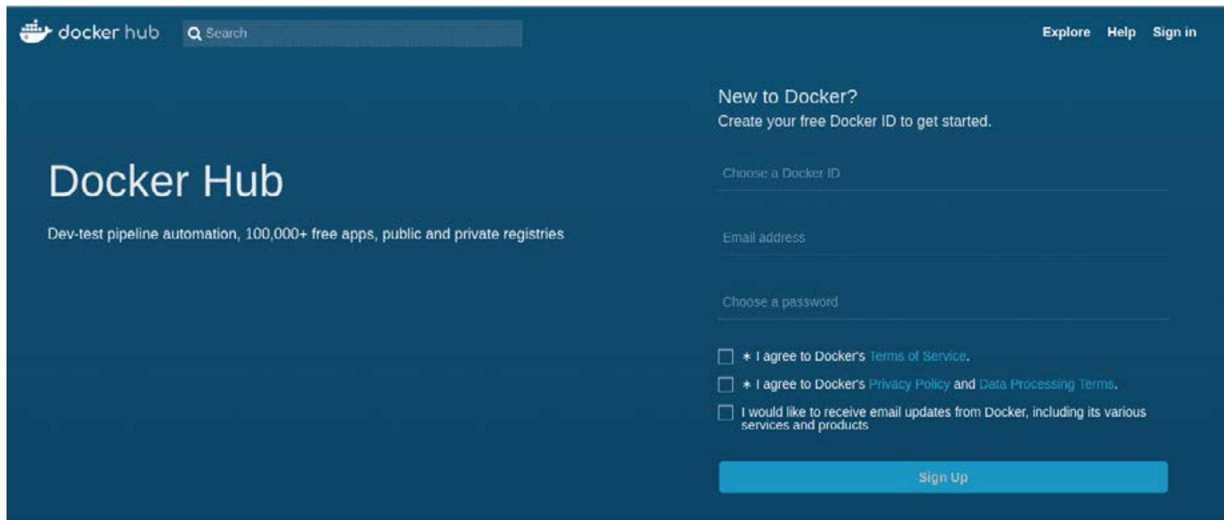
```
docker rm serverInBackground
```

When you now run `docker container ls -a` again you will notice that the container tagged `serverInBackground` is gone. Keep in mind that this only removes the stopped container! The image you used to create the container will still be there.
Removing the image
The time might come in which you don't need a certain image anymore. You can remove a image the same way you remove a container. To get the id of the image you want to remove you can run the docker image ls command from earlier. Once you know what you want to remove type the following command:

```
docker rmi <IMAGE-ID>
```

This will the image if it is not needed anymore by running docker instances.

Exploring the Docker Hub



You might have asked yourself where this mysterious httpd image comes from or how I know which Linux distro it is based on. Every image you use has to be hosted somewhere. This can either be done locally on your machine or a dedicated repository in your company or even online through a hosting service. The official Docker Hub is one of those repositories. Head over to the Docker Hub and take a moment to browse the site. When creating your own containers it is always a good idea not to reinvent the wheel. There are thousands of images out there spreading from small web servers (like our httpd image) to full fledged operating systems ready at your disposal. Just type a keyword in the search field at the top of the page (web server for example) and take a stroll through the offers available or just check out the httpd repo. Most of these images hosted here offer help regarding dependencies or installation. Some of them even include information about something called a Dockerfile..

Writing a Dockerfile

While creating containers from the command line is pretty straight forward, there are certain situations in which you don't want to configure these containers by hand. Luckily enough we have another option, the Dockerfile. If you have already taken a look at the example files provided for httpd you might have an idea about what you can expect.

So go ahead and create a new file called ‚Dockerfile' (mind the capitalization). We will add some content to this file:

```
FROM httpd:2.4
COPY ./html/ /usr/local/apache2/htdocs/
```

This is a very barebone Dockerfile. It basically just says two things:

- `FROM`– Use the provided image with the specified version for this container.
- `COPY`– Copy the content from the first path on the host machine to the second path in the container.

So what the Dockerfile currently says is: Use the image known as httpd in version 2.4, copy all the files from the sub folder „./html' to ‚/usr/local/apache2/htdocs/' and create a new image containing all my changes.
For extra credit: Remember the digest from before? You can use the digest to pin our new image to the httpd image version we used in the beginning. The syntax for this is:

```
FROM <IMAGENAME>@<DIGEST-STRING>
```

Now, it would be nice to have something that can actually be copied over. Create a folder called `html` and create a small index.html file in there. If you don't feel like writing one on your own just use mine:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>That‘s one small step for the web,</h1>
    <p>one giant leap for containerization.</p>
  </body>
</html>
```

Open a shell window in the exact location where you placed your Dockerfile and html folder and type the following command:

```
docker build . -t my-new-server-image
```

- `build`– The command for building images from Dockerfiles
- `.` – The `build` command expects a path as second parameter. The dot refers to the current location of the shell prompt.
- `-t` – The tag flag sets a name for the image that it can be referred by.

The shell output should look like this:

```
Sending build context to Docker daemon  3.584kB
 Step 1/2 : FROM httpd:2.4
 ---> 55a118e2a010
 Step 2/2 : COPY ./html/ /usr/local/apache2/htdocs/
 ---> Using cache
 ---> 867a4993670a
 Successfully built 867a4993670a
 Successfully tagged my-new-server-image:latest
```

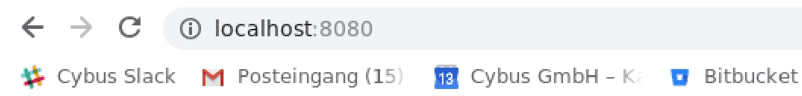
You can make sure that your newly created image is hosted on your local machine by running

```
docker image ls
```

This will show you all images hosted on your machine.
We can finally run our modified httpd image by simply typing:

```
docker run --name myModifiedServer -d -p 8080:80 my-new-server-image
```

This command should look familiar by now. The only thing we changed is that we are not using the httpd image anymore. Instead we are referring to our newly created ‚my-new-server-image'.
Let's see if everything is working by opening the Server in a browser.



That's one small step for the web,

one giant leap for containerization.

I think it is time for us to pat ourselves on the back. We did good today!

Summary

By the time you reached these lines you should be able to create, monitor and remove containers from preexisting images as well as create new ones using Dockerfiles. You should also have a basic understanding of how to inspect and debug running containers.

Where to go from here

As was to be expected from a basic lesson there is still a lot to cover. A good place to start is the Docker documentation itself. Another topic we didn't even touch is Docker Compose, which provides an elegant way to orchestrate groups of containers.

LEARN MORE

- Official Docker site
- Official documentation
- Dockerfiles
- Docker networks
- Docker Compose
- Docker Hub

Cybus is a specialist for secure IIoT Edge software, headquartered in Germany. Cybus Connectware serves smart factories as a universal Edge and DevOps hub. Machine builders and providers of IIoT services use the Cybus Connectware as a software-based gateway. As early as 2017, Cybus published the first secure industrial connector for machine data according to today's DIN SPEC 27070 standard. Industry analyst Gartner named Cybus a worldwide „Cool Vendor". Today, the company counts medium-sized and large companies from numerous industrial sectors such as mechanical engineering, automotive and aviation among its customers.

Cybus GmbH · Osterstraße 124 · 20255 Hamburg · Germany · www.cybus.io · hello@cybus.io · (+49) 40 228 58 68 51