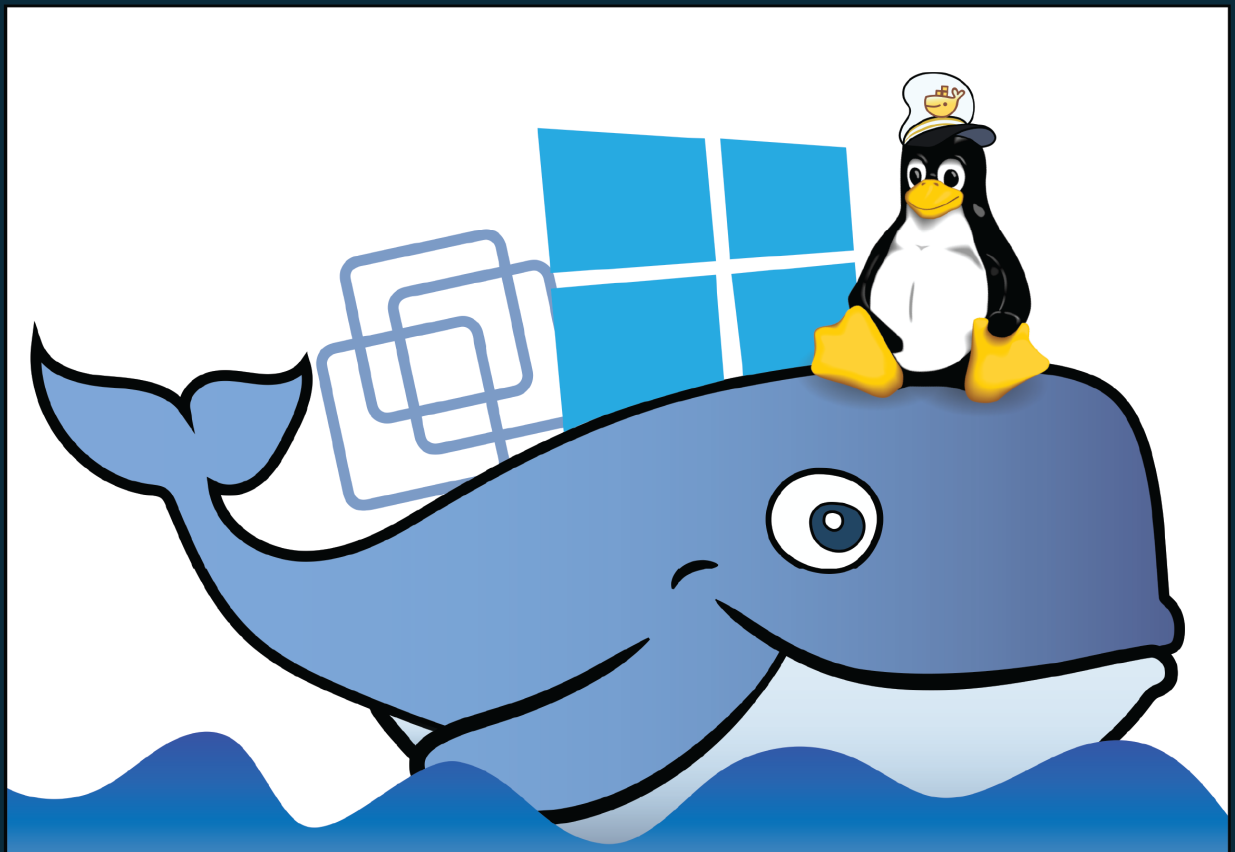
Docker
For Sysadmins
Linux Windows VMware



Hand-crafted for Infrastructure Ops

Nigel Poulton

Docker for Sysadmins: Linux Windows VMware

Getting started with Docker from the perspective of sysadmins and VM admins

Nigel Poulton

This book is for sale at http://leanpub.com/dockerforsysadmins

This version was published on 2016-10-11

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Huge thanks to my wife and kids for putting up with a geek in the house who genuinely thinks he's a bunch of software running inside of a container on top of midrange biological hardware. It can't be easy living with me!

Massive thanks as well to everyone who watches my Pluralsight videos. I love connecting with you and really appreciate all the feedback I've gotten over the years. This was one of the major reasons I decided to write this book! I hope it'll be an amazing tool to help you drive your careers even further forward.

Contents

0: About the book

This is a book about Docker, **hand-crafted for system administrators**. No prior knowledge required!

But what about developers and DevOps?

If you're a developer with no interest in operations then this book is not for you. If you're into DevOps then I think you'll get a lot from the book.

To keep things short… the book is **not** about showing you how to develop microservice apps with Docker. The book **is** about how the core Docker plumbing works. You'll learn the *how* and the *why* - the *commands* and the *deep-dives*. I really want to set you on your way to being as good at Docker as you already are at Linux, Windows or VMware.

Why should I read this book or care about Docker?

Docker is coming and there's no hiding from it. Developers are all over it. In IT Ops, we need to get ready to support *Dockerized* apps in our business critical production environments.

Isn't Docker just for developers?

Hell no!!!

All of those *Dockerized* apps that developers are creating need a solid Docker infrastructure to run on. And that's where IT Ops comes into the picture… IT Ops will be asked to build and run high performance and highly available Docker infrastructures to support business applications. If we're not skilled-up on Docker, we're going to struggle.

Why this Docker book and not another one?

At the time I decided to write the first edition of this book, *so many* of the Docker books already out there were terrible! They were a shocking mix of *badly written*, full of *technical inaccuracies*, or massively *out of date*. And sometimes they were all three! It's honestly not my intention to offend people, but go and read some of the reviews on Amazon. ***Some*** of the Docker books out there are a shameful waste of trees and paper!

So I decided to write something that was *well written*, *technically accurate*, and *kept up to date*. I want you to love this book.

If you buy the book and think it's bad, call me out on Twitter[1], give the book bad reviews, do whatever you feel necessary. And I'll try and fix it. But I'm confident you won't need to do any of that.

Should I buy the book if I've already watched your video courses?

If you like my video courses[2] you'll probably like the book. If you don't like my video courses you probably won't like the book.

How the book is organized

I've divided the book into two sections:

- The general info stuff
- The technical stuff

The general info stuff covers things like - Who is Docker, Inc. What is the Docker project. What is the OCI. Why do we even have containers... Not the coolest part of

[1] https://twitter.com/nigelpoulton

[2] https://app.pluralsight.com/library/search?q=nigel+poulton

the book, but the kind of stuff that's important if you want a good rounded knowledge of Docker and containers. It's only a short section and you probably *should* read it.

The technical stuff is what the book is all about! This is where you'll find everything you need to start working with Docker. It gets into the detail of *images*, *containers* and the increasingly important topic of *orchestration*. You'll get the theory so that you know how it all fits together, and you'll get commands and examples to show you how it all works in practice.

Every chapter in the *technical stuff* section is divided into three parts:

- The TLDR
- The deep dive
- The commands

The TLDR will give you two or three paragraphs that you could use to explain the topic at the coffee machine.

> **TLDR** or TL;DR, is a modern acronym meaning "too long; didn't read". It's normally used to indicate something that was too long to bother reading. I'm using it here in the book to indicate a short section that you can read if you're in a hurry and haven't got time to read the longer *deep dive* that immediately follows it.

The deep dive is where we'll explain how everything works and go through the examples.

The Commands lists out all of the commands you've learned in an easy to read list with brief reminders of what each one does.

I think you'll love that format.

Other stuff about the book

Here are just a few other things I want you to know about the book.

Text wrapping

I've tried really hard to get the commands and outputs to fit on a single line without wrapping! So instead of getting this...

```
$ docker service ps uber-service
ID                         NAME                IMAGE                     NOD\
E              DESIRED STATE  CURRENT STATE               ERROR
7zi85ypj7t6kjdkevreswknys  uber-service.1      nigelpoulton/tu-demo:v2  ip-\
172-31-12-203  Running        Running about an hour ago
0v5a97xatho0dd4x5fwth87e5   \_ uber-service.1   nigelpoulton/tu-demo:v1  ip-\
172-31-12-207  Shutdown       Shutdown about an hour ago
31xx0df6je8aqmkjqn8w1q9cf  uber-service.2      nigelpoulton/tu-demo:v2  ip-\
172-31-12-203  Running        Running about an hour ago
```

... you *should* get this.

```
$ docker service ps web-fe
ID          NAME      IMAGE             NODE  DESIRED  CURRENT
817f...f6z  web-fe.1  nigelpoulton/...  mgr2  Running  Running 5 mins
a1dh...mzn  web-fe.2  nigelpoulton/...  wrk1  Running  Running 5 mins
cc0j...ar0  web-fe.3  nigelpoulton/...  wrk2  Running  Running 5 mins
```

> For best results you might want to flip your reading device onto its side.

In doing this I've had to trim some of the output from some commands, but I don't think you're missing anything important. However, despite all of this, if you're reading on a small enough device, you're still going to get some wrapping :-(

But you didn't include something I really hoped you would...

I know the book doesn't cover *everything* about Docker. But it's not supposed to! I've written the book to get you up to speed as quickly as possible while still spending

the time to learn how it all fits together. If the book was 1,000 printed pages it **would not** help you get up to speed quickly!

However, I will add sections to the book if I think they're important and fundamental enough. Please use the book's feedback pages and hit me up on Twitter[3] with ideas of what you think should be included in the next version of the book.

Right, that's enough waffling. Let's crack on!

[3] https://twitter.com/nigelpoulton

4: The big picture

In the next few chapters we're going to get into the details of things like images, containers, and orchestration. But before we do that, I think it's a good idea to show you the big picture first.

In this chapter we'll download an image, start a new container, log in to the new container, run a command inside of it, and then destroy it. This will give you a good idea of what Docker is all about and how some of the major components fit together.

But don't worry if some of the stuff we do here is totally new to you. We're not trying to make you experts by the end of this chapter. All we're doing here is giving you a *feel* of things - setting you up so that when we get into the details in later chapters, you have an idea of how the pieces fit together.

All you need to follow along with the exercises in this chapter is a single Docker host. This can be any of the options we just installed in the previous chapter, though if you are using *Docker for Windows* you should be running it in "Linux Container" mode. It doesn't matter if this Docker host is a VM on your laptop, an instance in the public cloud, or bare metal server in your data center. All it needs, is to be running Docker with a connection to the internet.

Engine check

When you install Docker you get two major components:

- the Docker client
- the Docker daemon (sometimes called server)

The daemon implements the Docker Remote API[4]. In a default Linux installation the client talks to the daemon via a local IPC/Unix socket at `/var/run/docker.sock`. You can test that the client and daemon are operating and can talk to each other with the `docker version` command.

[4] https://docs.docker.com/engine/reference/api/docker_remote_api/

```
$ docker version
Client:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.1
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   23cf638
 Built:        Thu Aug 18 05:33:38 2016
 OS/Arch:      linux/amd64
```

As long as you get a response back from the `Client` and `Server` components you should be good to go. If you get an error response from the Server component, try the command again with `sudo` in front of it: `sudo docker version`. If it works with `sudo` you will need to prefix the remainder of the commands in this chapter with `sudo`.

Images

Now let's look at *images*.

Right now, the best way to think of a Docker image is as an object that contains an operating system and an application. It's not massively different from a virtual machine template. A virtual machine template is essentially a stopped virtual machine. In the Docker world, an image is effectively a stopped container.

Run the `docker images` command on your Docker host.

```
$ docker images
REPOSITORY    TAG         IMAGE ID       CREATED       SIZE
```

If you are working from a freshly installed Docker host it will have no images and will look like the output above.

Getting images onto your Docker host is called "pulling". Pull the `ubuntu:latest` image to your Docker host with the command below.

```
$ docker pull ubuntu:latest
latest: Pulling from library/ubuntu

952132ac251a: Pull complete
82659f8f1b76: Pull complete
c19118ca682d: Pull complete
8296858250fe: Pull complete
24e0251a0e2c: Pull complete
Digest: sha256:f4691c96e6bbaa99d...a2128ae95a60369c506dd6e6f6ab
Status: Downloaded newer image for ubuntu:latest
```

Run the `docker images` command again to see the `ubuntu:latest` image you just pulled.

```
$ docker images
REPOSITORY      TAG      IMAGE ID        CREATED         SIZE
ubuntu          latest   bd3d4369aebc    11 days ago     126.6 MB
```

We'll get into the details of where the image is stored and what's inside of it in the next chapter. For now it's enough to understand that it contains enough of an operating system (OS), as well as all the code to run whatever application it's designed for. The `ubuntu` image that we've pulled has a stripped down version of the Ubuntu Linux OS including a few of the common Ubuntu utilities.

It's worth noting as well that each image gets it's own unique ID. When working with the image, as we will do in the next step, you can refer to it using either its `ID` or name.

Containers

Now that we have an image pulled locally on our Docker host, we can use the `docker run` command to launch a container from it.

```
$ docker run -it ubuntu:latest /bin/bash
root@6dc20d508db0:/#
```

Look closely at the output from the command above. You should notice that your shell prompt has changed. This is because your shell is now attached to the shell of the new container - you are literally inside of the new container!

Let's examine that `docker run` command. `docker run` tells the Docker daemon to start a new container. The `-it` flags tell the daemon to make the container interactive and to attach our current shell to the shell of the container (we'll get more specific about this in the chapter on containers). Next, the command tells Docker that we want the container to be based on the `ubuntu:latest` image, and we tell it to run the `/bin/bash` process inside the container.

Run the following `ps` command from inside of the container to list all running processes.

```
root@6dc20d508db0:/# ps -elf
F S UID    PID  PPID   NI ADDR SZ WCHAN  STIME TTY  TIME CMD
4 S root     1     0    0 -  4560 wait   13:38 ?    00:00:00 /bin/bash
0 R root     9     1    0 -  8606 -      13:38 ?    00:00:00 ps -elf
```

As you can see from the output of the `ps` command, there are only two processes running inside of the container:

- PID 1. This is the `/bin/bash` process that we told the container to run with the `docker run` command.
- PID 9. This is the `ps -elf` process that we ran to list the running processes.

The presence of the `ps -elf` process in the output above could be a bit misleading as it is a short-lived process that dies as soon as the `ps` command exits. This means that the only long-running process inside of the container is the `/bin/bash` process.

Press `Ctrl-PQ` to exit the container. This will land you back in the shell of your Docker host. You can verify this by looking at your shell prompt.

Now that you are back at the shell prompt of you Docker host, run the `ps -elf` command again.

```
$ ps -elf
F S UID        PID  PPID    NI ADDR SZ WCHAN  TIME CMD
4 S root         1     0     0 -  9407 -      00:00:03 /sbin/init
1 S root         2     0     0 -     0 -      00:00:00 [kthreadd]
1 S root         3     2     0 -     0 -      00:00:00 [ksoftirqd/0]
1 S root         5     2   -20 -     0 -      00:00:00 [kworker/0:0H]
1 S root         7     2     0 -     0 -      00:00:00 [rcu_sched]
<Snip>
0 R ubuntu   22783 22475     0 -  9021 -      00:00:00 ps -elf
```

Notice how many more processes are running on your Docker host compared to the single long-running process inside of the container.

In a previous step you pressed `Ctrl-PQ` to exit your shell from the container. Doing this from inside of a container will exit you from the container without killing it. You can see all of the running containers on your system using the `docker ps` command.

```
$ docker ps
CNTNR ID  IMAGE           COMMAND    CREATED     STATUS     NAMES
0b3...41  ubuntu:latest   /bin/bash  7 mins ago  Up 7 mins  tiny_poincare
```

The output above shows a single running container. This is the container that you created earlier. The presence of your container in this output proves that it's still running. You can also see that it was created 7 minutes ago and has been running for 7 minutes.

Attaching to running containers

You can attach your shell to running containers with the `docker exec` command. As the container from the previous steps is still running let's connect back to it.

> **Note:** The example below references a container called "tiny_poincare". The name of your container will be different, so remember to substitute "tiny_poincare" with the name or ID of the container running on your Docker host.

```
$ docker exec -it tiny_poincare bash
root@6dc20d508db0:/#
```

Notice that your shell prompt has changed again. You are back inside the container.

The format of the `docker exec` command is: `docker exec -options <container-name or container-id> <command>`. In our example we used the `-it` options to attach our shell to the container's shell. We referenced the container by name and told it to run the bash shell.

Exit the container again by pressing `Ctrl-PQ`.

Your shell prompt should be back to your Docker host.

Run the `docker ps` command again to verify that your container is still running.

```
$ docker ps
CNTNR ID  IMAGE          COMMAND    CREATED     STATUS     NAMES
0b3...41  ubuntu:latest  /bin/bash  9 mins ago  Up 9 mins  tiny_poincare
```

Stop the container and kill it using the `docker stop` and `docker rm` commands.

```
$ docker stop tiny_poincare
tiny_poincare
$
$ docker rm tiny_poincare
tiny_poincare
```

Verify that the container was successfully deleted by running another `docker ps` command.

```
$ docker ps
CONTAINER ID    IMAGE    COMMAND    CREATED    STATUS    PORTS    NAMES
```

Congratulations! You've downloaded a Docker image, launched a container from that image, executed a command inside of the container (`ps -elf`) and then stopped and deleted the container. This *big picture* view should help you with the up-coming chapters where we will dig deeper into images and containers.