Orchestrating Docker

Manage and deploy Docker services to containerize applications efficiently

Shrikrishna Holla
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 "Unboxing Docker"
- A synopsis of the book’s content
- More information on Orchestrating Docker

About the Author

Shrikrishna Holla is a full-stack developer based in Bangalore and Chennai, India. He loves biking, listening to music, and occasionally, sketching. You can find him frequently in hackathons, wearing a hoodie and sipping Red Bull, preparing for an all-nighter.

He currently works as a product developer for Freshdesk, a cloud-based customer support platform.

You can get in touch with him on Twitter (@srikrishnaholla) or find him at the Docker IRC channel (#docker on Freenode) with the shrikrishna handle.

I would like to thank the creators of Docker, without whom this book wouldn't have seen the light of the day. To my editors, Parita and Larissa, it has been a long journey, but you have been extremely supportive and helpful week after week. To my parents, you have been, are, and will always be my inspiration—the final ray of light in the darkest dungeon. To my sisters, for the soothing words of advice whenever I've had the blues. To all my teachers, who helped me to choose my path. To my friends, who help me forget all my worries. To the countless people who have given me encouragement, suggestions, and feedback, I couldn't have done this without you. To my readers, thank you for trusting me with your learning.
Orchestrating Docker

Get started with Docker, the Linux containerizing technology that has revolutionized application sandboxing. With this book, you will be able to learn how to use Docker to make your development faster and your deployment of applications simpler.

This guide will show you how to build your application in sandboxed Docker containers and make them run everywhere-your development machine, your private server, or even on the cloud, with a fraction of the cost of a virtual machine. Build a PaaS, deploy a cluster, and so on, all on your development setup.

What This Book Covers

Chapter 1, Unboxing Docker, teaches you how to get Docker running in your environment.

Chapter 2, Docker CLI and Dockerfile, helps you to acclimatize to the Docker command-line tool and start building your own containers by writing Dockerfiles.

Chapter 3, Configuring Docker Containers, shows you how to control your containers and configure them to achieve fine-grained resource management.

Chapter 4, Automation and Best Practices, covers various techniques that help manage containers-co-ordinating multiple services using supervisor, service discovery, and knowledge about Docker's security.

Chapter 5, Friends of Docker, shows you the world surrounding Docker. You will be introduced to open source projects that use Docker. Then you can build your own PaaS and deploy a cluster using CoreOS.
Docker is a lightweight containerization technology that has gained widespread popularity in recent years. It uses a host of the Linux kernel's features such as namespaces, cgroups, AppArmor profiles, and so on, to sandbox processes into configurable virtual environments.

In this chapter, you will learn how to install Docker on various systems, both in development and in production. For Linux-based systems, since a kernel is already available, installation is as simple as the `apt-get install` or `yum install` commands. However, to run Docker on non-Linux operating systems such as OSX and Windows, you will need to install a helper application developed by Docker Inc., called Boot2Docker. This will install a lightweight Linux VM on VirtualBox, which will make Docker available through port 2375, assigned by the Internet Assigned Numbers Authority (IANA).

At the end of this chapter, you will have installed Docker on your system, be it in development or production, and verified it.

This chapter will cover the following points:

- Introducing Docker
- Installing Docker
- Ubuntu (14.04 and 12.04)
- Mac OSX and Windows
- OpenStack
- Inception: building Docker in Docker
- Verifying installation: Hello World output
- Introducing Docker
Docker was developed by DotCloud Inc. (Currently Docker Inc.), as the framework they built their **Platform as a Service (PaaS)** upon. When they found increasing developer interest in the technology, they released it as open source and have since announced that they will completely focus on the Docker technology's development, which is good news as it means continual support and improvement for the platform.

There have been many tools and technologies aimed at making distributed applications possible, even easy to set up, but none of them have as wide an appeal as Docker does, which is primarily because of its cross-platform nature and friendliness towards both system administrators and developers. It is possible to set up Docker in any OS, be it Windows, OSX, or Linux, and Docker containers work the same way everywhere. This is extremely powerful, as it enables a write-once-run-anywhere workflow. Docker containers are guaranteed to run the same way, be it on your development desktop, a bare-metal server, virtual machine, data center, or cloud. No longer do you have the situation where a program runs on the developer's laptop but not on the server.

The nature of the workflow that comes with Docker is such that developers can completely concentrate on building applications and getting them running inside the containers, whereas sysadmins can work on running the containers in deployment. This separation of roles and the presence of a single underlying tool to enable it simplifies the management of code and the deployment process.

But don't virtual machines already provide all of these features?

**Virtual Machines (VMs)** are fully virtualized. This means that they share minimal resources amongst themselves and each VM has its own set of resources allocated to it. While this allows fine-grained configuration of the individual VMs, minimal sharing also translates into greater resource usage, redundant running processes (an entire operating system needs to run!), and hence a performance overhead.

Docker, on the other hand, builds on a container technology that isolates a process and makes it believe that it is running on a standalone operating system. The process still runs in the same operating system as its host, sharing its kernel. It uses a layered copy-on-write filesystem called **Another Unionfs (AUFS)**, which shares common portions of the operating system between containers. Greater sharing, of course, can only mean less isolation, but vast improvements in Linux process's resource management solutions such as namespaces and cgroups have allowed Docker to achieve VM-like sandboxing of processes and yet maintain a very small resource footprint.
Let's take a look at the following image:

This is a Docker vs VM comparison. Containers share the host's resources with other containers and processes, and virtual machines have to run an entire operating system for every instance.

**Installing Docker**

Docker is available in the standard repositories of most major Linux distributions. We will be looking at the installation procedures for Docker in Ubuntu 14.04 and 12.04 (Trusty and Precise), Mac OSX, and Windows. If you are currently using an operating system not listed above, you can look up the instructions for your operating system at [https://docs.docker.com/installation/#installation](https://docs.docker.com/installation/#installation).

**Installing Docker in Ubuntu**

Docker is supported by Ubuntu from Ubuntu 12.04 onwards. Remember that you still need a 64-bit operating system to run Docker. Let's take a look at the installation instructions for Ubuntu 14.04.

**Installing Docker in Ubuntu Trusty 14.04 LTS**

Docker is available as a package in the Ubuntu Trusty release's software repositories under the name of `docker.io`:

```bash
$ sudo apt-get update
$ sudo apt-get -y install docker.io
```

That's it! You have now installed Docker onto your system. However, since the command has been renamed `docker.io`, you will have to run all Docker commands with `docker.io` instead of `docker`. 

---
Unboxing Docker

The package is named docker.io because it conflicts with another KDE3/GNOME2 package called docker. If you rather want to run commands as docker, you can create a symbolic link to the /usr/local/bin directory. The second command adds autocomplete rules to bash:

$ sudo ln -s /usr/bin/docker.io /usr/local/bin/docker
$ sudo sed -i '$acomplete -F _docker docker' \> /etc/bash_completion.d/docker.io

Installing Docker in Ubuntu Precise 12.04 LTS

Ubuntu 12.04 comes with an older kernel (3.2), which is incompatible with some of the dependencies of Docker. So we will have to upgrade it:

$ sudo apt-get update
$ sudo apt-get -y install linux-image-generic-lts-raring linux-headers-generic-lts-raring
$ sudo reboot

The kernel that we just installed comes with AUFS built in, which is also a Docker requirement.

Now let's wrap up the installation:

$ curl -s https://get.docker.io/ubuntu/ | sudo sh

This is a curl script for easy installation. Looking at the individual pieces of this script will allow us to understand the process better:

1. First, the script checks whether our Advanced Package Tool (APT) system can deal with https URLs, and installs apt-transport-https if it cannot:
   # Check that HTTPS transport is available to APT
   if [ ! -e /usr/lib/apt/methods/https ]; then apt-get update apt-get install -y apt-transport-https
   fi

2. Then it will add the Docker repository to our local key chain:
   $ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
   --recv-keys 36A1D7869245C8950F966B92D8576A8BA88D21E9

   You may receive a warning that the package isn't trusted. Answer yes to continue the installation.
3. Finally, it adds the Docker repository to the APT sources list, and updates and installs the lxc-docker package:

```
$ sudo sh -c "echo deb https://get.docker.io/ubuntu docker main
> /etc/apt/sources.list.d/docker.list"
$ sudo apt-get update
$ sudo apt-get install lxc-docker
```

Docker versions before 0.9 had a hard dependency on LXC (Linux Containers) and hence couldn’t be installed on VMs hosted on OpenVZ. But since 0.9, the execution driver has been decoupled from the Docker core, which allows us to use one of numerous isolation tools such as LXC, OpenVZ, systemd-nspawn, libvirt-lxc, libvirt-sandbox, qemu/kvm, BSD Jails, Solaris Zones, and even chroot! However, it comes by default with an execution driver for Docker’s own containerization engine, called libcontainer, which is a pure Go library that can access the kernel’s container APIs directly, without any other dependencies.

To use any other containerization engine, say LXC, you can use the -e flag, like so: $ docker -d -e lxc.

Now that we have Docker installed, we can get going at full steam! There is one problem though: software repositories like APT are usually behind times and often have older versions. Docker is a fast-moving project and a lot has changed in the last few versions. So it is always recommended to have the latest version installed.

**Upgrading Docker**

You can upgrade Docker as and when it is updated in the APT repositories. An alternative (and better) method is to build from source. The tutorial for this method is in the section titled *Inception: Docker in Docker*. It is recommended to upgrade to the newest stable version as the newer versions might contain critical security updates and bug fixes. Also, the examples in this book assume a Docker version greater than 1.0, whereas Ubuntu’s standard repositories package a much older version.
Mac OSX and Windows

Docker depends on the Linux kernel, so we need to run Linux in a VM and install and use Docker through it. Boot2Docker is a helper application built by Docker Inc. that installs a VM containing a lightweight Linux distribution made specifically to run Docker containers. It also comes with a client that provides the same Application Program Interface (API) as that of Docker, but interfaces with the `docker` daemon running in the VM, allowing us to run commands from within the OSX/Windows terminal. To install Boot2Docker, carry out the following steps:

1. Download the latest release of Boot2Docker for your operating system from http://boot2docker.io/.

2. The installation image is shown as follows:

![Install Docker for Mac OS X](image)

3. Run the installer, which will install VirtualBox and the Boot2Docker management tool.

Run `boot2docker`. The first run will ask you for a Secure Shell (SSH) key passphrase. Subsequent runs of the script will connect you to a shell session in the virtual machine. If needed, the subsequent runs will initialize a new VM and start it.

Alternately, to run Boot2Docker, you can also use the terminal command `boot2docker`:

```
$ boot2docker init # First run
```
$ boot2docker start
$ export DOCKER_HOST=tcp://$(boot2docker ip 2>/dev/null):2375

You will have to run boot2docker init only once. It will ask you for an SSH key passphrase. This passphrase is subsequently used by boot2docker ssh to authenticate SSH access.

Once you have initialized Boot2Docker, you can subsequently use it with the boot2docker start and boot2docker stop commands.

DOCKER_HOST is an environment variable that, when set, indicates to the Docker client the location of the docker daemon. A port forwarding rule is set to the boot2Docker VM's port 2375 (where the docker daemon runs). You will have to set this variable in every terminal shell you want to use Docker in.

Bash allows you to insert commands by enclosing subcommands within ``` or $(). These will be evaluated first and the result will be substituted in the outer commands.

If you are the kind that loves to poke around, the Boot2Docker default user is docker and the password is tcuser.

The boot2Docker management tool provides several commands:

$ boot2docker

Usage: boot2docker [options] {help|init|up|ssh|save|down|poweroff|reset |restart|config|status|info |ip|delete|download|version} [args]

When using boot2Docker, the DOCKER_HOST environment variable has to be available in the terminal session for Docker commands to work. So, if you are getting the Post http:///var/run/docker.sock/v1.12/containers/create: dial unix /var/run/docker.sock: no such file or directory error, it means that the environment variable is not assigned. It is easy to forget to set this environment variable when you open a new terminal. For OSX users, to make things easy, add the following line to your .bashrc or .bash_profile shells:

alias setdockerhost='export DOCKER_HOST=tcp://$(boot2docker ip 2>/dev/null):2375'
Now, whenever you open a new terminal or get the above error, just run the following command:

```
$ setdockerhost
```

This image shows how the terminal screen will look like when you have logged into the Boot2Docker VM.

### Upgrading Boot2Docker

1. Download the latest release of the Boot2Docker Installer for OSX from http://boot2docker.io/.
2. Run the installer, which will update VirtualBox and the Boot2Docker management tool.

To upgrade your existing virtual machine, open a terminal and run the following commands:

```
$ boot2docker stop
$ boot2docker download
```

### OpenStack

OpenStack is a piece of free and open source software that allows you to set up a cloud. It is primarily used to deploy public and private Infrastructure as a Service (IaaS) solutions. It consists of a pool of interrelated projects for the different components of a cloud setup such as compute schedulers, keychain managers, network managers, storage managers, dashboards, and so on.
Docker can act as a hypervisor driver for OpenStack Nova Compute. Docker support for OpenStack was introduced with the Havana release.

But... how?

Nova's Docker driver embeds a tiny HTTP server that talks to the Docker Engine's internal **Representational State Transfer (REST)** API (you will learn more on this later) through a **UNIX TCP** socket.

Docker has its own image repository system called Docker-Registry, which can be embedded into Glance (OpenStack's image repository) to push and pull Docker images. Docker-Registry can be run either as a **docker** container or in a standalone mode.

**Installation with DevStack**

If you are just setting up OpenStack and taking up the DevStack route, configuring the setup to use Docker is pretty easy.

Before running the DevStack route's stack.sh script, configure the virtual driver option in the localrc file to use Docker:

```
VIRT_DRIVER=docker
```

Then run the Docker installation script from the devstack directory. The socat utility is needed for this script (usually installed by the stack.sh script). If you don't have the socat utility installed, run the following:

```
$ apt-get install socat
$ ./tools/docker/install_docker.sh
```

Finally, run the stack.sh script from the devstack directory:

```
$ ./stack.sh
```

**Installing Docker for OpenStack manually**

Docker can also be installed manually if you already have OpenStack set up or in case the DevStack method doesn't work out:

1. Firstly, install Docker according to one of the Docker installation procedures. If you are co-locating the docker registry alongside the Glance service, run the following command:

   ```
   $ sudo yum -y install docker-registry
   ```
In the /etc/sysconfig/docker-registry folder, set the REGISTRY_PORT and SETTINGS_FLAVOR registries as follows:

$ export SETTINGS_FLAVOR=openstack
$ export REGISTRY_PORT=5042

In the docker registry file, you will also need to specify the OpenStack authentication variables. The following commands accomplish this:

$ source /root/keystonerc_admin
$ export OS_GLANCE_URL=http://localhost:9292

By default, /etc/docker-registry.yml sets the local or alternate storage_path path for the openstack configuration under /tmp. You may want to alter the path to a more permanent location:

openstack:
  
storage: glance
  
storage_alternate: local
  
storage_path: /var/lib/docker-registry

2. In order for Nova to communicate with Docker over its local socket, add nova to the docker group and restart the compute service to pick up the change:

$ usermod -G docker nova
$ service openstack-nova-compute restart

3. Start Redis (used by the Docker Registry), if it wasn't started already:

$ sudo service redis start
$ sudo chkconfig redis on

4. Finally, start the registry:

$ sudo service docker-registry start
$ sudo chkconfig docker-registry on

**Nova configuration**

Nova needs to be configured to use the virt Docker driver.

Edit the /etc/nova/nova.conf configuration file according to the following options:

[DEFAULT]
compute_driver = docker.DockerDriver
Alternatively, if you want to use your own Docker-Registry, which listens on a port different than 5042, you can override the following option:

```ini
docker_registry_default_port = 5042
```

### Glance configuration

Glance needs to be configured to support the Docker container format. Just add Docker to the list of container formats in the Glance configuration file:

```
[DEFAULT]
container_formats = ami,ari,aki,bare,ovf,docker
```

Leave the default formats in order to not break an existing glance installation.

### Docker-OpenStack flow

Once you configured Nova to use the `docker` driver, the flow is the same as that in any other driver:

```bash
$ docker search hipache
Found 3 results matching your query ("hipache")
NAME                     DESCRIPTION
samalba/hipache           https://github.com/dotcloud/hipache
```

Then tag the image with the Docker-Registry location and push it:

```bash
$ docker pull samalba/hipache
$ docker tag samalba/hipache localhost:5042/hipache
$ docker push localhost:5042/hipache
```

The push refers to a repository:

```bash
[localhost:5042/hipache] (len: 1)
Sending image list
Pushing repository localhost:5042/hipache (1 tags)
Push 100% complete```
In this case, the Docker-Registry (running in a docker container with a port mapped on 5042) will push the images to Glance. From there, Nova can reach them and you can verify the images with the Glance Command-Line Interface (CLI):

$ glance image-list

Only images with a docker container format will be bootable. The image basically contains a tarball of the container filesystem.

You can boot instances with the nova boot command:

$ nova boot --image "docker-busybox:latest" --flavor m1.tiny test

The command used will be the one configured in the image. Each container image can have a command configured for the run. The driver does not override this command.

Once the instance is booted, it will be listed in nova list:

$ nova list

You can also see the corresponding container in Docker:

$ docker ps

**Inception: Build Docker in Docker**

Though installing from standard repositories is easier, they usually contain older versions, which means that you might miss critical updates or features. The best way to remain updated is to regularly get the latest version from the public GitHub repository. Traditionally, building software from a source has been painful and done only by people who actually work on the project. This is not so with Docker. From Docker 0.6, it has been possible to build Docker in Docker. This means that upgrading Docker is as simple as building a new version in Docker itself and replacing the binary. Let's see how this is done.

**Dependencies**

You need to have the following tools installed in a 64-bit Linux machine (VM or bare-metal) to build Docker:

- Git
- Make
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is used here to clone the Docker public source code repository. Check out git-scm.org for more details.

The make utility is a software engineering tool used to manage and maintain computer programs. Make provides most help when the program consists of many component files. A Makefile file is used here to kick off the Docker containers in a repeatable and consistent way.

Building Docker from source

To build Docker in Docker, we will first fetch the source code and then run a few make commands that will, in the end, create a docker binary, which will replace the current binary in the Docker installation path.

Run the following command in your terminal:

$ git clone https://git@github.com/dotcloud/docker

This command clones the official Docker source code repository from the Github repository into a directory named docker:

$ cd docker
$ sudo make build

This will prepare the development environment and install all the dependencies required to create the binary. This might take some time on the first run, so you can go and have a cup of coffee.

If you encounter any errors that you find difficult to debug, you can always go to #docker on freenode IRC. The developers and the Docker community are very helpful.

Now we are ready to compile that binary:

$ sudo make binary

This will compile a binary and place it in the ./bundles/<version>-dev/binary/ directory. And voila! You have a fresh version of Docker ready.

Before replacing your existing binary though, run the tests:

$ sudo make test
If the tests pass, then it is safe to replace your current binary with the one you've just compiled. Stop the docker service, create a backup of the existing binary, and then copy the freshly baked binary in its place:

```
$ sudo service docker stop
$ alias wd='which docker'
$ sudo cp $(wd) $(wd)_
$ sudo cp $(pwd)/bundles/<version>-dev/binary/docker-<version>-dev $(wd)
$ sudo service docker start
```

Congratulations! You now have the up-to-date version of Docker running.

OSX and Windows users can follow the same procedures as SSH in the boot2Docker VM.

**Verifying Installation**

To verify that your installation is successful, run the following command in your terminal console:

```
$ docker run -i -t ubuntu echo Hello World!
```

The docker run command starts a container with the ubuntu base image. Since this is the first time you are starting an ubuntu container, the output of the container will be something like this:

```
Unable to find image 'ubuntu' locally
Pulling repository ubuntu
e54ca5efa2e9: Download complete
511136ea3c5a: Download complete
d7ac5e4f1812: Download complete
2f4b4d6a4a06: Download complete
83ff768040a0: Download complete
6c37f792ddac: Download complete

Hello World!
```
When you issue the `docker run ubuntu` command, Docker looks for the `ubuntu` image locally, and if it’s not found, it will download the `ubuntu` image from the public `docker` registry. You will also see it say **Pulling dependent layers**.

This means that it is downloading filesystem layers. By default, Docker uses AUFS, a layered copy-on-write filesystem, which means that the container image's filesystem is a culmination of multiple read-only filesystem layers. And these layers are shared between running containers. If you initiate an action that will write to this filesystem, it will create a new layer that will be the difference of the underlying layers and the new data. Sharing of common layers means that only the first container will take up a considerable amount of memory and subsequent containers will take up an insignificant amount of memory as they will be sharing the read-only layers. This means that you can run hundreds of containers even on a relatively low-powered laptop.

Once the image has been completely downloaded, it will start the container and echo *Hello World!* in your console. This is another salient feature of the Docker containers. Every container is associated with a command and it should run that command. Remember that the Docker containers are unlike VMs in that they do not virtualize the entire operating system. Each `docker` container accepts only a single command and runs it in a sandboxed process that lives in an isolated environment.

**Useful tips**

The following are two useful tips that might save you a lot of trouble later on. The first shows how to give the docker client non-root access, and the second shows how to configure the Ubuntu firewall rules to enable forwarding network traffic.

You do not need to follow these if you are using Boot2Docker.
Giving non-root access

Create a group called docker and add your user to that group to avoid having to add the sudo prefix to every docker command. The reason you need to run a docker command with the sudo prefix by default is that the docker daemon needs to run with root privileges, but the docker client (the commands you run) doesn't. So, by creating a docker group, you can run all the client commands without using the sudo prefix, whereas the daemon runs with the root privileges:

$ sudo groupadd docker  # Adds the docker group
$ sudo gpasswd -a $(whoami) docker  # Adds the current user to the group
$ sudo service docker restart

You might need to log out and log in again for the changes to take effect.

UFW settings

Docker uses a bridge to manage network in the container. Uncomplicated Firewall (UFW) is the default firewall tool in Ubuntu. It drops all forwarding traffic. You will need to enable forwarding like this:

$ sudo vim /etc/default/ufw
# Change:
# DEFAULT_FORWARD_POLICY="DROP"
# to
DEFAULT_FORWARD_POLICY="ACCEPT"

Reload the firewall by running the following command:

$ sudo ufw reload

Alternatively, if you want to be able to reach your containers from other hosts, then you should enable incoming connections on the docker port (default 2375):

$ sudo ufw allow 2375/tcp

Downloading the example code

You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you
Summary

I hope this introductory chapter got you hooked to Docker. The upcoming chapters will take you into the Docker world and try to dazzle you with its awesomeness.

In this chapter, you learned some history and some basics on Docker and how it works. We saw how it is different from and advantageous over VM.

Then we proceeded to install Docker on our development setup, be it Ubuntu, Mac, or Windows. Then we saw how to replace OpenStack's hypervisor with Docker. Later, we built Docker from source, within Docker! Talk about eating your own dog food!

Finally, we downloaded our first image and ran our first container. Now you can pat your self on the back and proceed to the next chapter, where we will cover the primary Docker commands in depth and see how we can create our own images.
Where to buy this book
You can buy Orchestrating Docker from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.