Content Update Program
FREE...See Details Inside

NEGUS LIVE LINUX SERIES

Docker Containers

Build and Deploy with Kubernetes, Flannel, Cockpit, and Atomic

Christopher Negus

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

NEGUS LIVE LINUX SERIES

Content Update Program

Docker Containers

This book is part of Prentice Hall and InformIT's exciting new Content Update Program, which provides automatic content updates for major technology improvements!

- As significant updates are made to the Docker technology, sections of this book will be updated or new sections will be added to match the updates to the technology.
- The updates will be delivered to you via a free Web Edition of this book, which can be accessed with any Internet connection.
- This means your purchase is protected from immediately outdated information!

For more information on InformIT's Content Update program, see the inside back cover or go to informit.com/CUP.

If you have additional questions, please email our Customer Service department at informit@custhelp.com.

Docker Containers

This page intentionally left blank

Docker Containers

Build and Deploy with Kubernetes, Flannel, Cockpit, and Atomic

Christopher Negus
with William Henry

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com

Library of Congress Control Number: 2015948006

ISBN-13: 978-0-134-13656-1
ISBN-10: 0-134-13656-X
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing: December 2015

As always, I dedicate this book to my wife, Sheree.
—Christopher Negus

This page intentionally left blank

Contents

This page intentionally left blank

Preface

Docker is a containerization technology at the center of a new wave for building, packaging, and deploying applications. It has the potential to impact every aspect of computing, from the application development process to how applications are deployed and scaled up and out across massive data centers.

Despite its great popularity, Docker is still a fairly new project, with many people still not really knowing exactly what Docker is. If you are one of those people, this book can help you take that first step, while also opening your eyes to the huge potential that containerization promises for you down the road. My goals for leading you into the world of containerization with this book can be summed up in these ways:

- **Hands-on learning**: I often say this in my books, but I believe that the best way to learn how technology works is to get it and use it. To that end, I let you choose from among several popular Linux systems, show you how to install Docker on the one you choose, and provide working examples of using Docker for everything from running a simple container to building and managing your own container images. That learning then extends into tools and techniques for orchestrating and managing containers.
- **How Docker can benefit you**: I explain the benefits of creating and running applications in containers, instead of installing software packages (in formats such as RPM or Deb) and running uncontained applications directly from your hard disk. Beyond running applications, I also describe how containerization can benefit software developers and system administrators.

- **Essential qualities of Docker**: I describe how Docker uses technologies such as Linux Containers (LXC) to keep containers separate from other applications running on a host computer or selectively tap into the host system. These qualities include how Docker uses name spaces, metadata, and separate file systems to both manage and secure containerized applications.

To get started, you don't need to know anything about Docker or containerization; you can treat this book as your introduction to Docker. However, this book is also intended to offer an entry into more advanced Docker-related topics, such as orchestration and container development.

As you progress through the book, you see specific ways to run containers, investigate them, stop and start them, save them, and generally manage them. As you begin creating your own containers, I discuss techniques to help you make container images that build and run efficiently. I even step you through build files (which are called Dockerfiles) that others have created to make their own containers.

A knowledge of Linux Containers in general, or Docker containers specifically, is not needed to start using this book. That said, however, there are other technologies you will use both within your Docker containers and outside those containers to work with them. Understanding some of those technologies will make your experience with Docker that much more fruitful.

Knowledge to Help You with Docker

To get the most out of working with Docker containers, it helps to know something about the operating environment in which Docker will be running. Docker is built on Linux technology and is specifically integrated with advanced features, including Linux Containers (LXC) for managing Linux name spaces and Cgroups for managing container access to system resources (such as CPU and memory).

Even your most basic interactions with Docker containers rely on underlying Linux technologies. You may have heard that you can run Docker on your Windows or Mac systems. But adding Docker to those systems always relies on your adding a Linux virtual machine. In other words, there are no Docker containers without Linux. Likewise, each container itself is typically built from a base image created from a specific Linux distribution.

So if you have no experience working with Linux systems, you might find it useful to learn about some of the following aspects of Linux and related technologies:

- **Command shell**: There are graphical interfaces available for working with Docker. However, most of the examples of Docker in this book are done from a Linux command line shell. Knowing how to get around in a Linux shell makes it much more efficient to work with Docker.
- **Software packages**: Docker is itself a mechanism for delivering software packaged and delivered together as a bundled application. To build the container images themselves, however, most Docker base images are set up to allow you to install software packages from the specific Linux distribution on which they were based.

  So, for example, for an Ubuntu base image, you should understand how to install Deb packages with tools such as apt-get. For Fedora, Red Hat Enterprise Linux, or CentOS Docker images, the `yum`, `dnf`, and `rpm` commands are useful. When you use these base images to build your own Docker containers, those images are usually enabled to automatically grab the packages you request from online software repositories. Understanding how to get and install packages in your chosen Linux distribution is important for your success with Docker.
- **File ownership and permissions**: Every file in a Linux system, as well as within a container, is owned by a particular user and group and has certain permissions set to allow access to those files. At times, you want to grant access to files and directories (folders) from the host within the container. Some of those might be special files, such as devices or sockets, that the application needs to run. Processes also run as a particular user. Understanding how those permissions work can be critical to getting a container working properly.

I mentioned only a few of the more obvious features you need to know about to work effectively with Docker containers. You will run into many other Linux-related features as you continue to explore how to make the best use of the Docker containers you use and create yourself.

If you are not familiar with Linux, I strongly recommend you take a class or get a book that gives you at least the basics of Linux to help you get going with Docker containers. My humble suggestion would be to pick up the ***Linux Bible***, Ninth Edition, written by this author (http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118999878.html). It will not only help you specifically with the technology you need to build Docker containers, but will also help you to generally work in a Linux environment as you develop Docker container images.

What This Book Covers

This book is meant to be used from beginning to end by someone just starting up with Docker containers. Later, it can serve as reference material to remind you of different options and features associated with Docker containers. The book is organized into five parts.

Part I: Getting Going with Containers

In Part I, you learn what you need to know to start working with Docker containers. Chapter 1, "Containerizing Applications with Docker," describes what containers are and how they differ from applications that are not contained. In Chapter 2, "Setting Up a Container Run-Time Environment," you learn how to install Docker on different general-purpose Linux systems, such as Fedora and Ubuntu, as well as how to install Docker on specialized container-oriented Linux systems, such as CoreOS and Project Atomic. In Chapter 3, "Setting Up a Private Docker Registry," we complete a basic container setup by showing you how to configure a private Docker registry to hold your own Docker images.

Part II: Working with Individual Containers

Most of the coverage in this part relates to using the `docker` command to work directly with individual containers. In Chapter 4, "Running Container Images," I show you how to run your first container images. To help you find and get container images, Chapter 5, "Finding, Pulling, Saving, and Loading Container Images," describes how to search for container images from the Docker registry and then pull the image you want, save it to a file, and load it into another Docker system.

In Chapter 6, "Tagging Images," you learn how to tag images, to better identify what the image contains and to use that information to push images to registries. In Chapter 7, "Investigating Containers," I show you how to look inside a Docker container or container image to see the details of how that container or image works. In Chapter 8, "Starting, Stopping, and Restarting Containers," you learn just that—how to stop, start, and restart containers.

In Chapter 9, "Configuring Container Storage," you learn how to configure storage, primarily by mounting directories from the host inside your containers. To learn how to configure networking for containers, Chapter 10, "Configuring Container Networking," describes how to configure both the default networking used (or not used) by the Docker service in general, as well as ways someone running containers can set network interfaces for individual containers.

Docker caches a lot of data, for possible reuse. In Chapter 11, “Cleaning Up Containers,” I show you how to clean out cached data left behind when you created or ran Docker images. In Chapter 12, “Building Docker Images,” you learn how to build your own Docker containers, including how to build containers that build and run efficiently.

Part III: Running Containers in Cloud Environments

In Chapter 13, “Using Super Privileged Containers,” I describe how to run what are referred to as super privileged containers (SPCs). To illustrate how SPCs work, I show you how you can get several images that can perform different administrative tasks on an RHEL Atomic system. In Chapter 14, “Managing Containers in the Cloud with Cockpit,” I describe how to manage containers across multiple hosts in your cloud or local environment using the Cockpit web-based container management tool.

Part IV: Managing Multiple Containers

In this part, I get into the area of orchestration. For Chapter 15, “Orchestrating Containers with Kubernetes,” I describe how to use Kubernetes master and node services all on one system to be able to try out Kubernetes. In Chapter 16, “Creating a Kubernetes Cluster,” I go beyond the all-in-one Kubernetes system to describe how to set up a Kubernetes cluster. With that cluster in place, you can deploy applications in container pods to be managed on different node computers from the master computer.

Part V: Developing Containers

In the short time that Docker has been around, techniques have already been developed to make building containers more efficient. In Chapter 17, “Developing Docker Containers,” I describe some tips and a few tricks for developing Docker containers. Finally, in Chapter 18, “Exploring Sample Dockerfile Files,” I show you various Dockerfile files I have come across to illustrate what different people have done to overcome obstacles to building their own containers.

So if you are ready now, step right up and start reading Chapter 1. I hope you enjoy the book!

This page intentionally left blank

Acknowledgments

The help I have had producing this book has been extraordinary. In my day job, I have the pleasure of working directly with people at Red Hat who take the fine work being done on projects like Docker, Kubernetes, and Atomic and extend and integrate those projects together into operating systems that are ready for the most stringent enterprise environments. So, in general, I want to thank developers, testers, and other writers on the Red Hat Enterprise Linux Atomic, OpenShift, and Linux container teams for helping me learn on a daily basis what it takes to make Linux Containers ready for the enterprise.

As for having a direct impact on the book, there are a few people from Red Hat I want to call out individually. First, William Henry wrote two chapters in this book on storage and networking. I was fortunate that he was available to write those critical chapters. Beyond his work here, William has made significant contributions to Docker-related projects. In fact, William wrote dozens of `docker` command man pages that are delivered with the Docker software itself. Having William around to participate in helping develop the content of the book was priceless as well.

Another important contributor to this book from Red Hat is Scott Collier. Scott's public contributions to the general knowledge about Docker have included blogs on setting up Docker and Kubernetes, as well as sharing many sample Dockerfiles through the Fedora Cloud initiative. For this book, Scott was generous with his time, helping me sort through technology and examples illustrated throughout the book.

Because I wrote this book outside of work hours (which is why it took me longer than I had hoped), I often relied on interactions with my publisher (Pearson) during evenings and weekends. So, thanks to editors Chris Zahn and Elaine Wiley for reviewing my content, occasionally responding on Sunday nights, and compressing their schedules to help me meet mine. Also from Pearson, my dear friend Debra Williams Cauley, who developed this project with me, has shown extraordinary patience as I sought to balance a tight schedule with my desire to take the time to write the exact book I wanted to write.

Finally, I'd like to thank my family. When someone writes a book he must almost, by necessity, neglect his family for some amount of time. I'm so proud of you all. Despite my drifting off to write, my son Seth managed to do a great imitation of Zac Efron in ***High School Musical*** by having the lead in his school play while also playing on his high school soccer team. My son Caleb found his niche, settling in on his little organic farm in Maine. And my wife, Sheree, continues to amaze younger generations with her fitness and Spartan runs. Your love and support are what keeps me going.

About the Author

Christopher Negus is a bestselling author of Linux books, a certified Linux instructor and examiner, Red Hat Certified Architect, and principal technical writer for Red Hat. At the moment, projects Chris is working on include Red Hat OpenStack Platform High Availability, Red Hat Enterprise Linux Atomic Enterprise, Kubernetes, and Linux Containers in Docker format.

As an author, Chris has written dozens of books about Linux and open source software. His ***Linux Bible***, Ninth Edition, released in 2015, is consistently among the top-selling Linux books today. During the dotcom days, Chris's ***Red Hat Linux Bible*** sold more than 250,000 copies in eight editions and was twice voted best Linux book of the year. Other books authored or coauthored by Chris include the ***Linux Toolbox*** series, ***Linux Toys*** series, ***Fedora and Red Hat Enterprise Linux Bible*** series, and ***Linux Troubleshooting Bible*** with Wiley Publishing.

With Prentice Hall, Chris helped produce the Negus Software Solution Series. For that series, Chris wrote ***Live Linux CDs*** and coauthored ***The Official Damn Small Linux Book***. That series also includes books on web development, Google Apps, and virtualization.

Chris joined Red Hat in 2008 as an RHCE instructor. For that role, he became a Red Hat Certified Instructor (RHCI) and Red Hat Certified Examiner (RHCX). In 2014, Chris became a Red Hat Certified Architect (RHCA), with certifications in Virtualization Administration, Deployment and Systems Management, Cluster and Storage Management, and Server Hardening. In 2011, Chris shifted from his Linux instructor role back to being a full-time writer for Red Hat, which he continues to do today.

Early in his career, Chris worked at UNIX System Laboratories and AT&T Bell Labs with the organizations that produced the UNIX operating system. During that time, Chris wrote the first official UNIX System V Desktop system manual and cowrote the ***Guide to the UNIX Desktop***. For eight years, Chris worked closely with developers of the UNIX system, from UNIX System V Release 2.0 through Release 4.2.

CHAPTER 3

Setting Up a Private Docker Registry

IN THIS CHAPTER:

- Create a private Docker registry in Fedora or Ubuntu
- Use the docker-registry package
- Use the registry container image
- Understand the Docker image namespace

One of the foundations of Docker is the ability to request to use an existing container image and then, if it is not already on your system, grab it from somewhere and download it to your system. By default, "somewhere" is the Docker Hub Registry (`https://hub.docker.com`). However, there are ways to configure other locations from which you can pull docker images. These locations are referred to as *registries*.

By setting up your own private registry, you can keep your private images to yourself. You can also save time by pushing and pulling your images locally, instead of having them go out over the Internet.

Setting up a private registry is simple. It requires getting the service (by installing a package or using the `registry` Docker container image), starting the service, and making sure the proper port is open so the service is accessible. Using registries requires a bit more explanation than setting up one, especially when you consider that features are added to Docker every day that are changing how Docker uses and searches registries for images.

In particular, the way that Docker uses the image namespace is changing to be more adaptable. If your location is disconnected from the Internet, with the Docker hub inaccessible, features are being developed to allow you to use a different default registry. Likewise, new features let you add registries to your search order, much the same way you can have an Internet browser look at different DNS servers.

This chapter describes how to set up a private Docker registry on several different Linux systems. The first examples are simply to help you get a Docker registry up and running quickly to begin testing or learning how to use registries. After that, I describe some techniques for making a Docker registry more production ready.

Later in the chapter, I tell you how to adapt the way your local Docker service uses Docker registries, including how to replace Docker.io as the default registry and add other registries to the search path.

NOTE

Having a local registry in place is not required to use Docker. However, as you build, save, and reuse images throughout this book, you may find it handy to have a way to store your images (especially private ones) without pushing them out to the public Docker Hub Registry. That said, you can skip this chapter for now if you want to learn more about using containers before you jump into setting up a Docker registry.

Getting and Starting a Private Docker Registry

You can run a Docker registry on your Linux system in a number of different ways to store your own Docker images. For Linux distributions that include a docker-registry package (such as Fedora and Red Hat Enterprise Linux), you can install that package and start up the service. For other distributions, you can run the official `registry` container image from Docker.io to provide the service.

See the section later in the chapter that corresponds to the Linux system you are using for instructions on installing and running a Docker registry on that system. For Fedora, I illustrate how to use the docker-registry package, while for Ubuntu I show how to use the `registry` container.

Here are a few general things you should know about setting up a Docker registry:

- **Install anywhere**: Like most servers, the Docker registry does not need to be installed on client systems (that is, where you run your `docker` commands). You can install it on any Linux system that your clients can reach over a network. That way, multiple Docker clients can access your Docker registry.

- **Open port**: If your Docker registry is not on the client, you must be sure that TCP port 5000 is not being blocked by the firewall where the Docker registry is running.
- **Provide space**: If you push a lot of images to your registry, space can fill up quickly. For the docker-registry package, stored images are contained in the `/var/lib/docker-registry` directory. Make sure you configure enough space in that directory to meet your needs, or you can configure a different directory, if you want.

Setting Up a Docker Registry in Fedora

Follow these instructions to install and start up a Docker registry on a Fedora system. At the moment, this procedure creates a version 1 Docker registry from the docker-registry RPM package. Although this procedure was tested on Fedora, the same basic procedures should work for the following Linux distributions:

- Fedora 22 or later
- Red Hat Enterprise Linux 7.1 or later
- CentOS 7.1 or later

The docker-registry package is not included in the Atomic project Fedora, RHEL, and CentOS distributions. So you must use the `registry` container, described later for setting up a Docker registry in Ubuntu, to get that feature on an Atomic Linux system.

> **NOTE**
>
> During the following procedure, you are going to use image tags to identify the registry where you intend an image to be stored. For a more in-depth look at tags, refer to Chapter 6, "Tagging Images." To get `docker-registry` to work, you may need to edit the `usr/lib/system/docker-registry.service` and remove `--debug`.

1. **Install docker-registry**: When you install the docker-registry package in Fedora, it pulls in more than a dozen dependent packages as well. To install those packages, type the following:

```
# yum install docker-registry
...
Transaction Summary
===============================================
```

```
Install  1 Package (+15 Dependent packages)
Total download size: 6.8 M
Installed size: 39 M
Is this ok [y/d/N]: y
```

2. **List docker-registry contents**: Use the `rpm` command to list the contents of the docker-registry file in Fedora. There are nearly 200 files (mostly python code in the package). This command shows you only documentation and configuration files (I describe how to configure them later):

```
# rpm -ql docker-registry | grep -E "(/etc)|(/usr/share)|(systemd)"
/etc/docker-registry.yml
/etc/sysconfig/docker-registry
/usr/lib/systemd/system/docker-registry.service
/usr/share/doc/docker-registry
/usr/share/doc/docker-registry/AUTHORS
/usr/share/doc/docker-registry/CHANGELOG.md
/usr/share/doc/docker-registry/LICENSE
/usr/share/doc/docker-registry/README.md
```

3. **Open firewall**: If your Fedora system is running a firewall that blocks incoming connections, you may need to open TCP port 5000 to allow access to the Docker registry service. Assuming you are using the firewall service in Fedora, run these commands to open the port on the firewall (immediately and permanently) and see that the port has been opened:

```
# firewall-cmd --zone=public --add-port=5000/tcp
# firewall-cmd --zone=public --add-port=5000/tcp --permanent
# firewall-cmd --zone=public --list-ports
5000/tcp
```

4. **Start the docker-registry service**: If you want to do any special configuration for your Docker registry, refer to the next sections before starting the service. For a simple docker-registry installation, however, you can simply start the service and begin using it, as follows (as the status shows, the docker-registry service is active and enabled):

```
# systemctl start docker-registry
# systemctl enable docker-registry
Created symlink from
  /etc/systemd/system/multi-user.target.wants/docker-registry.
service
  to /usr/lib/systemd/system/docker-registry.service.
# systemctl status docker-registry
docker-registry.service - Registry server for Docker
 Loaded: loaded (/usr/lib/systemd/system/docker-registry.
service;enabled)
 Active: active (running) since Mon 2015-05-25 12:02:14 EDT; 42s ago
```

```
Main PID: 5728 (gunicorn)
   CGroup: /system.slice/docker-registry.service
        ├─5728 /usr/bin/python /usr/bin/gunicorn --access-logfile
             - --max-requests 100 --graceful-timeout 3600-t 36...
...
```

5. **Get an image**: A common image used to test Docker is the hello-world image available from the Docker Hub Registry. Run that image as follows (which pulls that image to the local system and runs it):

   ```
   # docker run --name myhello hello-world
   Unable to find image 'hello-world:latest' locally
   latest: Pulling from docker.io/hello-world
   91c95931e552: Download complete
   a8219747be10: Download complete
   Hello from Docker.
   docker.io/hello-world:latest: The image you are pulling has been
   verified.
   ...
   ```

6. **Allow access to registry**: The docker clients in Fedora and Red Hat Enterprise Linux require that you either obtain a certificate from the registry or you identify the registry as insecure. For this example, you can identify the registry as insecure by editing the `/etc/sysconfig/docker` file and creating the following lines in that file:

   ```
   ADD_REGISTRY='--add-registry localhost:5000'
   INSECURE_REGISTRY='--insecure-registry localhost:5000'
   ```

   After that, restart the local Docker service:

   ```
   # systemctl restart docker
   ```

7. **Tag the image**: Use `docker tag` to give the image a name that you can use to push it to the Docker registry on the local system:

   ```
   # docker tag hello-world localhost:5000/hello-me:latest
   ```

8. **Push the image**: To push the hello-world to the local Docker registry, type the following:

   ```
   # docker push localhost:5000/hello-me:latest
   The push refers to a repository [localhost:5000/hello-me] (1 tags)
   ...
   Pushing tag for rev [91c95931e552] on
        {http://localhost:5000/v1/repositories/hello-me/tags/latest}
   ```

9. **Pull the image**: To make sure you can retrieve the image from the registry, in the second Terminal, remove the image from your system, then try to retrieve it from your local registry:

```
# docker rm myhello
# docker rmi hello-world localhost:5000/hello-me:latest
# docker pull localhost:5000/hello-me:latest
Pulling repository localhost:5000/hello-me
91c95931e552: Download complete
a8219747be10: Download complete
# docker images
REPOSITORY              TAG    IMAGE ID     CREATED     VIRTUAL SIZE
localhost:5000/hello-me latest 91c95931e552 5 weeks ago 910 B
```

In the example just shown, the image was successfully pushed to and pulled from the local repository. At this point, you have these choices:

- If you want to learn more about how the Docker registry works and possibly modify its behavior, skip to the "Configuring a Private Docker Registry" section later in this chapter.
- If you are ready to start using Docker containers, skip ahead to Chapter 4, "Running Container Images."

The next section describes how to set up a Docker registry in Ubuntu.

Setting Up a Docker Registry in Ubuntu

Instead of installing a Docker registry from a software package, you can download the `registry` container from the Docker Hub Registry and use that to provide the Docker registry service. This is a quick and easy way to try out a Docker registry, although the default registry doesn't scale well for a production environment and is more difficult to configure.

NOTE

Several versions of the registry are available. For this example, I use `registry:latest`, which results in an image of a version 1 Docker registry. By the time you try this, there may be a stable version 2 available. I recommend you refer here for information on running the version 2 Docker registry: `https://docs.docker.com/registry/`.

Although this procedure was tested on Ubuntu 14.04, the same basic procedure should work on any Linux system running the Docker service.

To get started here, install Docker as described in Chapter 2, "Setting Up a Container Run-Time Environment," and start up the Docker service. I suggest you open two Terminal windows (shells) to do this procedure. Open one where you plan to run the registry service, so you can watch it in progress as you start up and test it. Open another Terminal, from which you can push and pull images.

1. **Get the registry image:** Run the `docker pull` command as follows to pull the `registry` image from the Docker Hub Registry (see Chapter 5, "Finding, Pulling, Saving, and Loading Container Images," for a description of `docker pull`):

```
$ sudo docker pull registry:latest
Pulling repository registry
204704ce3137: Download complete
e9e06b06e14c: Download complete
...
```

2. **Run the registry image**: To try out the Docker registry, run the image in the foreground so you can watch messages produced as the container image is running (see Chapter 4 for a description of `docker run`). This command starts the latest `registry` image, exposes TCP port 5000 on the system so clients outside the container can use it, and runs it as a foreground process in the first terminal:

```
$ sudo docker run -p 5000:5000 registry:latest
[2015-05-25 21:33:35 +0000][1][INFO] Starting gunicorn 19.1.1
[2015-05-25 21:33:35 +0000][1][INFO] Listening at:
http://0.0.0.0:5000 (1)
[2015-05-25 21:33:35 +0000][1][INFO] Using worker: gevent
...
```

3. **Get an image**: To test that you can push and pull images, open a second Terminal window. A common image used to test Docker is the hello-world image available from the Docker Hub Registry. Run that image as follows (which pulls that image to the local system and runs it):

```
$ sudo docker run --name myhello hello-world
Pulling repository hello-world
91c95931e552: Download complete
a8219747be10: Download complete
Hello from Docker.
This message shows that your installation appears to be working
correctly.
...
```

4. **Tag the image**: Use `docker tag` to give the image a name that you can use to push it to the Docker registry on the local system:

```
$ sudo docker tag hello-world localhost:5000/hello-me:latest
```

5. **Push the image**: To push the hello-world to the local Docker registry, type the following:

```
$ sudo docker push localhost:5000/hello-me:latest
The push refers to a repository [localhost:5000/hello-me] (len: 1)
...
Pushing tag for rev [91c95931e552] on
     {http://localhost:5000/v1/repositories/hello-me/tags/latest}
```

6. **Check the Docker registry log messages**: If the image was pushed to the registry successfully, in the first Terminal you should see messages showing PUT commands succeeding. For example:

```
172.17.42.1 - - [25/May/2015:22:12:37 +0000] "PUT
/v1/repositories/hello-me/images HTTP/1.1" 204 - "-" "docker/1.0.1
go/go1.2.1 git-commit/990021a kernel/3.13.0-24-generic os/linux
arch/amd64"
```

7. **Pull the image**: To make sure you can retrieve the image from the registry, in the second Terminal remove the image from your system, and then try to retrieve it from your local registry:

```
$ sudo docker rm myhello
$ sudo docker rmi hello-world localhost:5000/hello-me:latest
$ sudo docker pull localhost:5000/hello-me:latest
Pulling repository localhost:5000/hello-me
91c95931e552: Download complete
a8219747be10: Download complete
# docker images
REPOSITORY              TAG    IMAGE ID     CREATED     VIRTUAL SIZE
localhost:5000/hello-me latest 91c95931e552 5 weeks ago 910 B
```

8. **Run the docker registry again**: Instead of running the `registry` image in the foreground, holding the Terminal open, you can have it run more permanently in the background (`-d`). To do that, close the running registry container and start a new image as follows:

```
$ sudo docker run -d -p 5000:5000 registry:latest
```

The Docker registry is running in the background now, ready to use. At this point, you have these choices:

- If you want to learn more about how the Docker registry works and possibly modify its behavior, skip to the "Configuring a Private Docker Registry" section later in this chapter.
- If you are ready to start using Docker containers, skip ahead to Chapter 4.

The next section describes how to set up a Docker registry in other Linux distributions.

Configuring a Private Docker Registry

The default registries that come in the docker-registry package or the `registry` container are fine if you just want to try out a Docker registry. If you want to use a registry in a production environment, however, you need a deeper understanding of how to configure your Docker registry to better suit your needs.

The following sections describe how to modify the Docker registry software for both the docker-registry package and using the registry container.

Configuring the docker-registry Package

To better understand how the docker-registry package software works, start with how the registry is set to run by default. When the docker-registry service starts up in Fedora or Red Hat Enterprise Linux, it runs the `gunicorn` process. There is one main `gunicorn` process and four additional `gunicorn` workers running, by default, to provide the service.

From a full `ps` output the `gunicorn` processes; you can see the options set for them:

```
# ps -ef | grep gunicorn
00:00:00 /usr/bin/python /usr/bin/gunicorn --access-logfile -
 --max-requests 100 --graceful-timeout 3600 -t 3600 -k gevent -b
0.0.0.0:5000 -w 4 docker_registry.wsgi:application
```

Here's what you can learn from this command line:

- **--access-logfile**: Access to the docker-registry service is logged to any file you set. In this case, however, the log file is set to a single hyphen (-), so access messages are simply sent to standard output (where they are picked up by the `systemd` journal and can be viewed by the `journalctl` command).
- **--max-requests 100**: Sets the maximum number of requests that a `gunicorn` daemon can accept to 100. After that, the worker is restarted.
- **--graceful-timeout 3600**: Gives the `gunicorn` worker 3600 seconds (6 minutes) to finish handling a request once it has been sent a restart signal. If it has not completed what it is doing by that time, it is killed.
- **-t 3600**: If the `gunicorn` worker is silent for more than 3600 seconds (6 minutes), it is killed and restarted.
- **-k gevent**: Sets the type of `gunicorn` worker to gevent (an asynchronous type of worker based on Greenlets).

- **-b 0.0.0.0:5000**: Sets the worker to bind on all IP addresses on the system (0.0.0.0) on port 5000. This allows docker clients to connect to the Docker registry through any external network interface on the system via TCP port 5000.
- **-w 4**: Sets the number of worker processes to 4 (above the original `gunicorn` process).
- **docker_registry.wsgi:application**: Runs the process with the Docker registry wsgi application.

To change the behavior of the docker-registry service, you can edit the `/etc/sysconfig/docker-registry` file. Here is how that file is set by default in Fedora:

```
# The Docker registry configuration file
DOCKER_REGISTRY_CONFIG=/etc/docker-registry.yml

# The configuration to use from DOCKER_REGISTRY_CONFIG file
SETTINGS_FLAVOR=local

# Address to bind the registry to
REGISTRY_ADDRESS=0.0.0.0

# Port to bind the registry to
REGISTRY_PORT=5000

# Number of workers to handle the connections
GUNICORN_WORKERS=4
```

In the `docker-registry` file, you can do such things as have the Docker registry listen only on a particular IP address (by default, REGISTRY_ADDRESS=0.0.0.0 listens on all addresses). You can change the port of the service to something other than TCP port 5000 or set the number of `gunicorn` workers to something other than 4.

The `/etc/docker-registry.yml` file is set as the Docker registry config file. SETTINGS_FLAVOR=local tells the config file to include common variables and then set the directory `/var/lib/docker-registry` for local storage use. In the `/etc/sysconfig/docker-registry` file, the common variables you can set include the following:

- **LOGLEVEL**: By default, the log level is set to info. This can also be set to debug, notice, warning, warn, err, error, crit, alert, emerg, or panic.
- **DEBUG**: Set to either true or false to have debugging turned on or off.
- **STANDALONE**: If set to true (the default), the registry acts as a standalone registry and doesn't query the Docker index.

- **INDEX_ENDPOINT**: If the local registry is not set to run in standalone, the default, the index endpoint is set to `https://index.docker.io`.
- **STORAGE_REDIRECT**: By default, this is disabled.
- **DISABLE_TOKEN_AUTH**: If the service is not in standalone, this variable is enabled to allow token authentication.
- **PRIVILEGED_KEY**: By default, no privileged key is set.
- **SEARCH_BACKEND**: By default, there is no search backend.
- **SQLALCHEMY_INDEX_DATABASE**: By default, the SQLite search backend database is set to: `sqlite:////tmp/docker-registry.db`.

If you want to use a setting flavor other than local, look in the `/etc/docker-registry.yml` file. Different setting flavors can be used for Ceph Object Gateway configuration, Google Cloud Storage configuration, OpenStack Swift Storage, and others.

Other variables you can set that can be picked up by the gunicorn process, include the following. Notice that some of these values show up on the `gunicorn` command line:

- **GUNICORN_GRACEFUL_TIMEOUT**: Sets the timeout for gracefully restarting workers (in seconds).
- **GUNICORN_SILENT_TIMEOUT**: Sets the timeout for restarting workers that have gone silent (in seconds).
- **GUNICORN_USER**: Runs the gunicorn process as the user set here, instead of running it with root user privileges.
- **GUNICORN_GROUP**: Runs the gunicorn process as the group set here, instead of running it with root group privileges.
- **GUNICORN_ACCESS_LOG_FILE**: Sets the name of the log file to direct messages to those that are related to clients trying to access the service. By default, messages are sent to the systemd journal through standard output.
- **GUNICORN_ERROR_LOG_FILE**: Sets the name of the log file to direct messages to those that are related to error conditions. By default, messages are sent to the systemd journal through standard output.
- **GUNICORN_OPTS**: Identifies any extra options you want to pass to the gunicorn process.

After you set or change `/etc/sysconfig/docker-registry` file variables, restart the `docker-registry` service for these features to take effect.

Configuring the `registry` Container

Instead of trying to configure the `registry` container image by modifying the contents of the running container, the creators of that container image suggest you rebuild the registry container image yourself. In particular, you probably want to add security measures to your registry and more flexible storage features.

So far, this book has not yet introduced you to the concepts you need to build your own containers. However, after you have become familiar with the process, if you decide you want to build a custom version 1 registry container, I recommend you refer to the `docker-registry` GitHub page:

```
https://github.com/docker/docker-registry
```

From the `docker-registry` GitHub page, you can find information on how to build a version 1 registry image and links to the Dockerfile used to build it (`https://github.com/docker/docker-registry/blob/master/Dockerfile`).

By the time you read this, Docker registry version 2 may be ready to use. Refer to the Docker registry 2.0 page (`https://docs.docker.com/registry`) for details on how to deploy and configure this newer version of the Docker registry.

Understanding the Docker Image Namespace

Similar to the way that the Internet uses the Domain Name System (DNS) to have a unique set of names refer to all the host computers in the world, Docker set out to make a namespace to allow a unique way to name every container image in the world. In that vision, a `docker run` *`someimage`* would result in the exact same *`someimage`* being pulled to the local system and run, no matter where your location or what type of Linux system you run it on.

For some potential Docker users, this presents problems. Some Docker installations are disconnected from the Internet. Security requirements of others allow them to search and pull images only from registries that they own themselves. These issues would prevent a pure Docker system from being installed in their environments.

There has been pressure to change some aspects of how the Docker image namespace works, so you can expect that story to evolve over time. As things stand today, however, you should know that a system running Docker purely from the upstream Docker Project code has the following attributes:

- **Search**: An unpatched Docker system today only searches the Docker Hub Registry when you run a `docker search` command.

- **Blocking registries**: Docker does not have a feature to block the Docker Hub Registry. So pulling an image without identifying a specific registry causes Docker to search for that image on the Docker Hub Registry (if it's not already on the local system).
- **Changing the default registry**: Docker doesn't have a feature for changing your default registry to anything other than the Docker Hub Registry.
- **Push confirmation**: Docker does not ask you to confirm a push request before it begins pushing an image.

Changes to some of these features are being discussed in the Docker community. Patches to change how some of these features work are included in Red Hat Enterprise Linux, Fedora, Atomic project, and related Linux distributions. For example, the current version of the docker package in RHEL Atomic (docker-1.8) includes some of those features just mentioned.

For example, here are some settings from the `/etc/sysconfig/docker` file on an RHEL Atomic system that represent features that have not yet been added to the upstream Docker Project:

```
ADD_REGISTRY='--add-registry registry.access.redhat.com'
# BLOCK_REGISTRY='--block-registry'
# INSECURE_REGISTRY='--insecure-registry'
```

The ADD_REGISTRY variable lets you add a registry to use for `docker search` and `docker pull` commands. For users of Red Hat distributions, this puts Red Hat's own registry (`registry.access.redhat.com`) before the Docker Hub Registry, so the user can know he is searching and pulling from that registry first. A user could also replace that registry with his own registries or simply add his own registry in front of Red Hat's registry.

Using the ADD_REGISTRY variable to this file puts any registry you add at the front of the list searched. However, if a requested image is not found in any of the registries you add, the Docker Hub Registry still is searched next. To change that behavior, you need to use the BLOCK_REGISTRY variable.

By setting the BLOCK_REGISTRY variable, you can block access to any registry you choose. Of course, at the moment only the Docker Hub Registry is searched by default. So, to block the Docker Hub Registry from search and pull requests, you could use the following line:

```
BLOCK_REGISTRY='--block-registry docker.io'
```

With that set, any requests for images that could not be found in registries set with ADD_REGISTRY variables would fail to be found, even if they existed at the

Docker Hub Registry. In this way, only registries that you specifically included are searched for images by the users of this particular docker installation.

The INSECURE_REGISTRY='--insecure-registry' variable does not explicitly allow or disallow a registry. This is a specific case where someone wants to use the local Docker client to pull an image from a registry that provides HTTPS communication, but the client doesn't have a certificate from that registry to verify its authenticity. Uncommenting the variable and adding the name of the insecure registry to that line allows the `docker` command to pull from that registry without full authorization. For example:

```
INSECURE_REGISTRY='--insecure-registry myreg.example.com'
```

Again, this and other features just described are not part of the upstream Docker Project. But if you need these features for your installation, you can change how access to registries works by default in Docker using these features that are currently in Fedora, RHEL, CentOS, and related Atomic project systems.

Summary

Setting up a private Docker registry gives you the ability to push and pull images without using the public Docker Hub Registry. This chapter described two different ways of setting up a Docker registry for yourself.

For Linux distributions that have a docker-registry package available (such as Fedora and Red Hat Enterprise Linux), you can install that package and start up the docker-registry service using the `systemctl` command. As an alternative, any system running the Docker service can pull and run the `registry` image, available from the Docker Hub Registry, to offer a private Docker registry.

Besides describing how to set up your own Docker registry, the chapter included a description of how the Docker image namespace works, with the Docker Hub Registry as its centerpiece. Proposed modifications to that model have been implemented in Fedora and other Red Hat sponsored operating systems and are being discussed in the Docker community. These modifications give users the ability to change which registries are set up to be used with search and pull requests from the Docker service.

This page intentionally left blank

Index

Symbols

A

B

D

I

J-K

N

O

P

Q-R

This page intentionally left blank

NEGUS LIVE LINUX SERIES

Content Update Program

Docker Containers

Instructions to access your free copy of *Docker Containers* Web Edition as part of the Content Update Program:

If you purchased your book from informit.com, your free Web Edition can be found under the **Digital Purchases** tab on your account page.

If you have purchased your book at a retailer other than InformIT and/or have not registered your copy, follow these steps:

1. Go to **informit.com/register**.
2. Sign in or create a new account.
3. Enter ISBN: **9780134136561.**
4. Answer the questions as proof of purchase.
5. Click on the **Digital Purchases** tab on your Account page to access your free Web Edition.

More About the Content Update Program…

InformIT will be updating the *Docker Containers* Web Edition periodically, as the Docker technology evolves.

Registered users will receive an email alerting them of the changes each time the *Docker Containers* Web Edition has been updated. The email alerts will be sent to the email address used for your informit.com account.

When a new edition of this book is published, no further updates will be added to this book's Web Edition. However, you will continue to have access to your current Web Edition with its existing updates.

The Web Edition can be used on tablets that use modern mobile browsers. Simply log into your informit.com account and access the Web Edition from the **Digital Purchases** tab.

For more information about the Content Update Program, visit informit.com/CUP or email our Customer Service department at informit@custhelp.com.