This book will show you how monitoring containers and keeping a keen eye on the working of applications helps improve the overall performance of the applications that run on Docker.

The book covers monitoring containers using Docker's native monitoring functions, various plugins, as well as third-party tools that help in monitoring. We'll start with how to obtain detailed statistics for active containers, resources consumed, and container behavior. We also show you how to use these stats to improve the overall performance of the system. Next, you will learn how to use Sysdig to both view your containers' performance metrics in real time and record sessions to query later. By the end of this book, you will have a complete knowledge of how to implement monitoring for your containerized applications and make the most of the metrics you are collecting.

Who this book is written for

This book is for DevOps engineers and system administrators who manage Docker containers and want to better manage these containers using expert techniques and methods, and also improve the way they maintain their applications built on Docker.

What you will learn from this book

- Discover the tools built into Docker to gain an insight into your container's performance
- Augment Docker's built-in tools with modern tools such as cAdvisor from Google, Sysdig by Draios, and Soundcloud's Prometheus
- Integrate the monitoring of your containers with more traditional monitoring solutions, such as Zabbix
- Take advantage of the various SaaS offerings from third parties to move monitoring away from your local infrastructure and into the cloud
- Discover the various ways to ship your application's logs from the container to a central logging service
- Get the most out of your application and resources with the right implementation of your monitoring method

Monitor your Docker containers and their apps using various native and third-party tools with the help of this exclusive guide!
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Introduction to Docker Monitoring'
- A synopsis of the book’s content
- More information on Monitoring Docker
About the Author

Russ McKendrick is an experienced solutions architect who has been working in IT and IT-related industries for the better part of 23 years. During his career, he has had varied responsibilities in a number of industries, ranging from looking after entire IT infrastructures to providing first line, second line, and senior support in client facing, and internal teams for corporate organizations.

He works almost exclusively with Linux, using open source systems and tools on various platforms ranging from dedicated hardware and virtual machines to public clouds.
Preface

With the increase in the adoption of Docker containers, the need to monitor which containers are running, what resources they are consuming, and how it affects the overall performance of the system, has become a time-related need. Monitoring Docker will teach you how monitoring containers and keeping a keen eye on the working of applications help to improve the overall performance of the applications that run on Docker.

This book will cover monitoring containers using Docker's native monitoring functions, various plugins, and also third-party tools that help in monitoring. The book will first cover how to obtain detailed stats for the active containers, resources consumed, and container behavior. This book will also show the readers how to use these stats to improve the overall performance of the system.

What this book covers

Chapter 1, Introduction to Docker Monitoring, discusses how different it is to monitor containers compared to more traditional servers such as virtual machines, bare metal machines, and cloud instances (Pets versus Cattle and Chickens versus Snowflakes). This chapter also details the operating systems covered in the examples later in this book and also gives a little information on how to get a local test environment up and running using vagrant, so that installation instructions and practical examples can be easily followed.

Chapter 2, Using the Built-in Tools, helps you learn about the basic metrics you can get out of the vanilla Docker installation and how you can use them. Also, we will understand how to get real-time statistics on our running containers, how to use commands that are familiar to us, and how to get information on the processes that are launched as part of each container.
Preface

Chapter 3, *Advanced Container Resource Analysis*, introduces cAdvisor from Google, which adds a lot more precision to the basic tools provided by Docker. You will also learn how to install cAdvisor and start collecting metrics.

Chapter 4, *A Traditional Approach to Monitoring Containers*, looks at a traditional tool for monitoring services. By the end of this chapter, you should know your way around Zabbix and the various ways you can monitor your containers.

Chapter 5, *Querying with Sysdig*, describes Sysdig as "an open source, system-level exploration tool to capture system state and activity from a running Linux instance, then save, filter, and analyze it." In this chapter, you will learn how to use Sysdig to both view your containers' performance metrics in real time and also record sessions to query later.

Chapter 6, *Exploring Third Party Options*, walks you through a few of the Software as a Service (SaaS) options that are available, why you would use them, and how to install their clients on the host server.

Chapter 7, *Collecting Application Logs from within the Container*, looks at how we can get the content of the log files for the applications running within our containers to a central location so that they are available even if you have to destroy and replace a container.

Chapter 8, *What Are the Next Steps?*, looks at the next steps you can take in monitoring your containers by talking about the benefits of adding alerting to your monitoring. Also, we will cover some different scenarios and look at which type of monitoring is appropriate for each of them.
Introduction to Docker Monitoring

Docker has been a recent but very important addition to a SysAdmins toolbox.

Docker describes itself as an open platform for building, shipping, and running distributed applications. This means that developers can bundle their code and pass it to their operations team. From here, they can deploy safe in the knowledge that it will be done so in a way that introduces consistency with the environment in which the code is running.

When this process is followed, it should make the age-old developers versus operations argument of "it worked on my local development server"—a thing of the past. Since before its "production ready" 1.0 release back in June 2014, there had been over 10,000 Dockerized applications available. By the end of 2014, that number had risen to over 71,000. You can see how Docker grew in 2014 by looking at the infographic that was published by Docker in early 2015, which can be found at https://blog.docker.com/2015/01/docker-project-2014-a-whirlwind-year-in-review/.

While the debate is still raging about how production ready the technology is, Docker has gained an impressive list of technology partners, including RedHat, Canonical, HP, and even Microsoft.

Companies such as Google, Spotify, Soundcloud, and CenturyLink, have all open sourced tools that support Docker in some way, shape, or form and there has also been numerous independent developers who have released apps that provide additional functionality to the core Docker product set. Also, all the companies have sprung up around the Docker ecosystem.
This book assumes that you have had some level of experience building, running, and managing Docker containers, and that you would now like to start to metrics from your running applications to further tune them, or that you would like to know when a problem occurs with a container so that you can debug any ongoing issues.

If you have never used Docker before, you may want to try one of the excellent books that serve and introduce you to all the things that Docker provides, books such as *Learning Docker*, Packt Publishing, or Docker's own introduction to containers, which can be found at their documentation pages, as follows:

- **Official Docker docs**: [https://docs.docker.com/](https://docs.docker.com/)

Now, we have a brought ourselves up to speed with what Docker is; the rest of this chapter will cover the following topics:

- How different is it to monitor containers versus more traditional servers such as virtual machines, bare metal machine, and cloud instances (Pets, Cattle, Chickens, and Snowflakes).
- What are the minimum versions of Docker you should be running?
- How to follow instructions on bringing up an environment locally using Vagrant in order to follow the practical exercises in this book

**Pets, Cattle, Chickens, and Snowflakes**

Before we start discussing the various ways in which you can monitor your containers, we should get an understanding of what a SysAdmins world looks like these days and also where containers fit into it.

A typical SysAdmin will probably be looking after an estate of servers that are hosted in either an on-site or third-party data center, some may even manage instances hosted in a public cloud such as Amazon Web Services or Microsoft Azure, and some SysAdmins may juggle all their server estates across multiple hosting environments.

Each of these different environments has its own way of doing things, as well as performing best practices. Back in February 2012, Randy Bias gave a talk at Cloudscaling that discussed architectures for open and scalable clouds. Towards the end of the slide deck, Randy introduced the concept of Pets versus Cattle (which he attributes to Bill Baker, who was then an engineer at Microsoft).
Pets versus Cattle is now widely accepted as a good analogy to describe modern hosting practices.

**Pets**

Pets are akin to traditional physical servers or virtual machines, as follows:

- Each pet has a name; for example, myserver.domain.com.
- When they're not well, you take them to the vet to help them get better. You employ SysAdmins to look after them.
- You pay close attention to them, sometimes for years. You take backups, patch them, and ensure that they are fully documented.

**Cattle**

Cattle, on the other hand, represent more modern cloud computing instances, as follows:

- You've got too many to name, so you give them numbers; for example, the URL could look something like ip123123123123.eu.public-cloud.com.
- When they get sick, you shoot them and if your herd requires it, you replace anything you've killed: A server crashes or shows signs that it is having problems, you terminate it and your configuration automatically replaces it with an exact replica.
- You put them in a field and watch them from far and you don't expect them to live long. Rather than monitoring the individual instances, you monitor the cluster. When more resources are needed, you add more instances and once the resource is no longer required, you terminate the instances to get you back to your base configuration.

**Chickens**

Next up is a term that is a good way of describing how containers fit into the Pets versus Cattle world; in a blog post title "Cloud Computing: Pets, Cattle and ... Chickens?" on ActiveState, Bernard Golden describes containers as Chickens:

- They're more efficient than cattle when it comes to resource use. A container can boot in seconds where a instance or server can take minutes; it also uses less CPU power than a typical virtual machine or cloud instance.

You can view the original slide deck at http://www.slideshare.net/randybias/architectures-for-open-and-scalable-clouds.
Introduction to Docker Monitoring

- There are many more chickens than cattle. You can quite densely pack containers onto your instances or servers.
- Chickens tend to have a shorter lifespan than cattle and pets. Containers lend themselves to running micro-services; these containers may only be active for a few minutes.

The original blog post can be found at http://www.activestate.com/blog/2015/02/cloud-computing-pets-cattle-and-chickens.

Snowflakes
The final term is not animal-related and it describes a type of server that you defiantly don't want to have in your server estate, a Snowflake. This term was penned by Martin Fowler in a blog post titled "SnowflakeServer". Snowflakes is a term applied to "legacy" or "inherited" servers:

- Snowflakes are delicate and are treated with kid gloves. Typically, the server has been in the data center since you started. No one knows who originally configured it and there is no documentation of it; all you know is that it is important.
- Each one is unique and is impossible to exactly reproduce. Even the most hardened SysAdmin fears to reboot the machine incase it doesn't boot afterwards, as it is running end-of-life software that can not easily be reinstalled.

Martin's post can be found at http://martinfowler.com/bliki/SnowflakeServer.html.

So what does this all mean?
Depending on your requirements and the application you want to deploy, your containers can be launched onto either pet or cattle style servers. You can also create a clutch of chickens and have your containers run micro-services.

Also, in theory, you can replace your feared snowflake servers with a container-based application that meets all the end-of-life software requirements while remaining deployable on a modern supportable platform.

Each of the different styles of server has different monitoring requirements, in the final chapter we will look at Pets, Cattle, Chickens, and Snowflakes again and discuss the tools we have covered in the coming chapters. We will also cover best practices you should take into consideration when planning your monitoring.
Chapter 1

Docker

While Docker hit its version 1.0 milestone over a year ago, it is still in its infancy; with each new release comes new features, bug fixes, and even support for some early functionality that is being depreciated.

Docker itself is now a collection of several smaller projects; these include the following:

• Docker Engine
• Docker Machine
• Docker Compose
• Docker Swarm
• Docker Hub
• Docker Registry
• Kitmatic

In this book, we will be using Docker Engine, Docker Compose, and the Docker Hub.

Docker Engine is the core component of the Docker project and it provides the main bulk of the Docker functionality. Whenever Docker or the `docker` command is mentioned in this book, I will be referring to Docker Engine.

The book assumes you have Docker Engine version 1.71 or later installed; older versions of Docker Engine may not contain the necessary functionality required to run the commands and software covered in the upcoming chapters.

Docker Compose started its life as a third-party orchestration tool called Fig before being purchased by Docker in 2014. It is described as a way of defining a multi-container application using YAML (http://yaml.org). Simply put, this means that you quickly deploy complex applications using a single command that calls a human readable configuration file.

We assume that you have Docker Compose 1.3.3 or later installed; the `docker-compose.yml` files mentioned in this book have been written with this version in mind.
Introduction to Docker Monitoring

Finally, the majority of the images we will be deploying during this book will be sourced from the Docker Hub (https://hub.docker.com/), which not only houses a public registry containing over 40,000 public images but also 100 official images. The following screenshot shows the official repositories listing on the Docker Hub website:

You can also sign up and use the Docker Hub to host your own public and private images.

**Launching a local environment**

Wherever possible, I will try to ensure that the practical exercises in this book will be able to be run on a local machine such as your desktop or laptop. For the purposes of this book, I will assume that your local machine is running either a recent version OS X or an up-to-date Linux distribution and has a high enough specification to run the software mentioned in this chapter.
The two tools we will be using to launch our Docker instances will also run on Windows; therefore, it should be possible to follow the instructions within this, although you may have to refer the usage guides for any changes to the syntax.

Due to the way in which Docker is architected, a lot of the content of this book will have you running commands and interacting with the command line on the virtual server that is acting as the host machine, rather than the containers themselves. Because of this, we will not be using either Docker Machine or Kitematic.

Both of these are tools provided by Docker to quickly bootstrap a Docker-enabled virtual server on your local machine, as unfortunately the host machines deployed by these tools contain a stripped down operating system that is optimized for running Docker with the smallest footprint as possible.

As we will be installing additional packages on the host machines, a stripped down "Docker only" operating system may not have the components available to meet the prerequisites of the software that we will be running in the later chapters; therefore, to ensure that there are no problems further on, we be running a full operating system.

Personally, I prefer a RPM-based operating system such as RedHat Enterprise Linux, Fedora, or CentOS, as I have been using them pretty much since the day I first logged into a Linux server.

However, as a lot of readers will be familiar with the Debian-based Ubuntu, I will be providing practical examples for both operating systems.

To ensure the experience is as consistent as possible, we will be installing Vagrant and VirtualBox to run the virtual machine that will act as a host to run our containers.

Vagrant, written by Mitchell Hashimoto, is a command line tool for creating and configuring reproducible and portable virtual machine environments. There have been numerous blog posts and articles that actually pitch Docker against Vagrant; however, in our case, the two technologies work quite well together in providing a repeatable and consistent environment.

Vagrant is available for Linux, OS X, and Windows. For details on how to install, go to the Vagrant website at https://www.vagrantup.com/.

VirtualBox is a great all round open source virtualization platform originally developed by Sun and now maintained by Oracle. It allows you to run both 32-bit and 64-bit guest operating systems on your local machine. Details on how to download and install VirtualBox can be found at https://www.virtualbox.org/; again, VirtualBox can be installed on Linux, OS X, and Windows.
Cloning the environment

The source for the environment along with the practical examples can be found on GitHub in the Monitoring Docker repository at https://github.com/russmckendrick/monitoring-docker.

To clone the repository on a terminal on your local machine, run the following commands (replacing the file path as needed):

```bash
mkdir ~/Documents/Projects
cd ~/Documents/Projects/
git clone https://github.com/russmckendrick/monitoring-docker.git
```

Once cloned, you should see a directory called monitoring-docker and then enter that directory, as follows:

```bash
cd ~/Documents/Projects/monitoring-docker
```

Running a virtual server

In the repository, you will find two folders containing the necessary Vagrant file to launch either a CentOS 7 or a Ubuntu 14.04 virtual server.

If you would like to use the CentOS 7 vagrant box, change the directory to vagrant-centos:

```bash
cd vagrant-centos
```

Once you are in the vagrant-centos directory, you will see that there is a Vagrant file; this file is all you need to launch a CentOS 7 virtual server. After the virtual server has been booted, the latest version of docker and docker-compose will be installed and the monitoring-docker directory will also be mounted inside the virtual machine using the mount point /monitoring-docker.

To launch the virtual server, simply type the following command:

```bash
vagrant up
```

This will download the latest version of the vagrant box from https://atlas.hashicorp.com/russmckendrick/boxes/centos71 and then boot the virtual server; it’s a 450 MB download so it may take several minutes to download; it only has to do this once.
Now that you have booted the virtual server, you can connect to it using the following command:

```
vagrant ssh
```

Once logged in, you should verify that `docker` and `docker-compose` are both available:

```
vagrant@docker:~$ docker --version
docker version 1.8.3, build d2c979
vagrant@docker:~$ docker-compose --version
docker-compose version 1.4.1
```

Finally, you can try running the `hello-world` container using the following command:

```
docker run hello-world
```
If everything goes as expected, you should see the following output:

```bash
$ docker run hello-world
Hello from Docker.
This message shows that your installation appears to be working correctly.

To generate this message, Docker ran the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which ran the
   executable that produces the output you are currently reading.
4. The Docker daemon destroyed that container to free up server resources.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash
```

To try something more ambitious, you can run an Ubuntu container with the following command:

```bash
docker run -it ubuntu bash
```

Before we launch and enter the Ubuntu container, let's confirm that we are running the CentOS host machine by checking the release file that can be found in `/etc`:

```bash
vi /etc/*-release
```

```bash
CentOS Linux release 7.3.1903 (Core)
NAME="CentOS"
VERSION="7.3.1903"
ARCH='x86_64'
RELEASE='7.3.1903'
```

```bash
vi /etc/*-release
```

```bash
CentOS Linux release 7.3.1903 (Core)
NAME="CentOS"
VERSION="7.3.1903"
ARCH='x86_64'
RELEASE='7.3.1903'
```

```bash
vi /etc/*-release
```

```bash
CentOS Linux release 7.3.1903 (Core)
NAME="CentOS"
VERSION="7.3.1903"
ARCH='x86_64'
RELEASE='7.3.1903'
```

```bash
vi /etc/*-release
```
Now, we can launch the Ubuntu container. Using the same command, we can confirm that we are inside the Ubuntu container by viewing its release file:

```
[Image of terminal output showing container information]
```

To exit the container just type in `exit`. This will stop the container from running, as it has terminated the only running process within the container, which was bash, and returned you to the host CentOS machine.

As you can see here from our CentOS 7 host, we have launched and removed an Ubuntu container.

Both the CentOS 7 and Ubuntu Vagrant files will configure a static IP address on your virtual machine. It is 192.168.33.10; also, there is a DNS record for this IP address available at `docker.media-glass.es`. These will allow you to access any containers that expose themselves to a browser at either `http://192.168.33.10/` or `http://docker.media-glass.es/`.

```
The URL http://docker.media-glass.es/ will only work while the vagrant box is up, and you have a container running which serves Web pages.
```

You can see this in action by running the following command:

```
docker run -d -p 80:80 russmckendrick/nginx-php
```

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.
Introduction to Docker Monitoring

This will download and launch a container running NGINX. You can then go to http://192.168.33.10/ or http://docker.media-glass.es/ in your browser; you should see a forbidden page. This is because we have not yet given NGINX any content to serve (more on this will be covered later in the book):

![403 Forbidden](image)

For more examples and ideas, go to the website at http://docs.docker.com/userguide/.

**Halting the virtual server**

To log out of the virtual server and return to your local machine, you type `exit`.

You should now see your local machine's terminal prompt; however, the virtual server you booted will still be running in the background happily, using resources, until you either power it down using the following command:

```
vagrant halt
```

Terminate the virtual server altogether using `vagrant destroy`:

```
vagrant destroy
```
To check the current status of the virtual server, you can run the following command:

```
vagrant status
```

The result of the preceding command is given in the following output:

```
VM is powered off. To restart the VM, simply run "vagrant up"
```

Either powering the virtual server back on or creating it from scratch again, can be achieved by issuing the `vagrant up` command again.

The preceding details show how to use the CentOS 7 vagrant box. If you would prefer to launch an Ubuntu 14.04 virtual server, you can download and install the vagrant box by going into the `vagrant-ubuntu` directory using the following command:

```
cd ~/Documents/Projects/monitoring-docker/vagrant-ubuntu
vagrant up
```

From here, you will be able run `vagrant up` and follow the same instructions used to boot and interact with the CentOS 7 virtual server.

**Summary**

In this chapter, we talked about different types of server and also discussed how your containerized applications can fit into each of the categories. We have also installed VirtualBox and used Vagrant to launch either a CentOS 7 or Ubuntu 14.04 virtual server, with `docker` and `docker-compose` installed.

Our new virtual server environment will be used throughout the upcoming chapters to test the various different types of monitoring. In the next chapter, we will start our journey by using Docker's in-built functionality to explore metrics about our running containers.
Where to buy this book

You can buy Monitoring 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.