Jaokim Verona, Michael Duffy,
Paul Swartout

Learning DevOps: Continuously Deliver Better Software

Learning Path

Learn to use some of the most exciting and powerful tools to deliver world-class quality software with continuous delivery and DevOps

Packt>

Learning DevOps: Continuously Deliver Better Software

Learn to use some of the most exciting and powerful tools to deliver world-class quality software with continuous delivery and DevOps

A course in three modules

Packt

BIRMINGHAM - MUMBAI

Learning DevOps: Continuously Deliver Better Software

Published on: September 2016

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.
ISBN 978-1-78712-661-9

www.packtpub.com

Credits

Authors
Joakim Verona
Michael Duffy
Paul Swartout

Reviewers
Per Hedman
Max Manders
Jon Auman
Tom Geudens
Sami Rönkä
Diego Woitasen
Adam Strawson

Content Development Editor
Aishwarya Pandere

Production Coordinator
Nilesh Mohite

Preface

The Learning DevOps: Continuously Deliver Better Software, is a course that will harness the power of DevOps to boost your skill set and make your IT organization perform better. It will aid developers, systems administrators who are keen to employ DevOps techniques to help with the day-to-day complications of managing complex infrastructures.

What this learning path covers

Module 1, Practical DevOps, describes how DevOps can assist us in the emerging field of the Internet of Things.

Module 2, DevOps Automation Cookbook, covers recipes that allow you to automate the build and configuration of the most basic building block in your infrastructure servers..

Module 3, Continuous Delivery and DevOps – A Quickstart Guide - Second Edition, provides some insight into how you can take CD and DevOps techniques and experience beyond the traditional software delivery process.

What you need for this learning path

Module 1: This module contains many practical examples. To work through the examples, you need a machine preferably with a GNU/Linux-based operating system,

such as Fedora.

Module 2: For this book, you will require the following software:

A server running Ubuntu 14.04 or greater.

A desktop PC running a modern Web Browser

A good Text editor or IDE.

Module 3: There are many tools mentioned within the book that will help you no end. These include technical tools such as Jenkins, GIT, Docker, Vagrant, IRC, Sonar, and Graphite, and nontechnical tools and techniques such as Scrum, Kanban, agile, and TDD.

You might have some of these (or similar) tools in place, or you might be looking at implementing them, which will help. However, the only thing you'll really need to enjoy and appreciate this book is the ability to read and an open mind.

Who this learning path is for

This course is for developers who wish to take on larger responsibilities and understand how the infrastructure that builds today's enterprises works. It is also for operations personnel who would like to better support their developers. Anyone who wants to understand how to painlessly and regularly ship quality software can take up this course.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this course—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the course's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt course, we have a number of things to help you to get the most from your purchase.

Downloading the example code

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

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the course in the **Search** box.
5. Select the course for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this course from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the course's webpage at the Packt Publishing website. This page can be accessed by entering the course's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the course is also hosted on GitHub at `https://github.com/PacktPublishing/Learning-DevOps-Continuously-Deliver-Better-Software`. We also have other code bundles from our rich catalog of books, videos, and courses available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our courses—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this course. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your course, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the course in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this course, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

Module 1

Practical DevOps

Harness the power of DevOps to boost your skill set and make your IT organization perform better

1
Introduction to DevOps and Continuous Delivery

Welcome to *Practical DevOps*!

The first chapter of this book will deal with the background of DevOps and setting the scene for how DevOps fits into the wider world of Agile systems development.

An important part of DevOps is being able to explain to coworkers in your organization what DevOps is and what it isn't.

The faster you can get everyone aboard the DevOps train, the faster you can get to the part where you perform the actual technical implementation!

In this chapter, we will cover the following topics:

- Introducing DevOps
- How fast is fast?
- The Agile wheel of wheels
- The cargo cult Agile fallacy
- DevOps and ITIL

Introducing DevOps

DevOps is, by definition, a field that spans several disciplines. It is a field that is very practical and hands-on, but at the same time, you must understand both the technical background and the nontechnical cultural aspects. This book covers both the practical and soft skills required for a best-of-breed DevOps implementation in your organization.

The word "DevOps" is a combination of the words "development" and "operation". This wordplay already serves to give us a hint of the basic nature of the idea behind DevOps. It is a practice where collaboration between different disciplines of software development is encouraged.

The origin of the word DevOps and the early days of the DevOps movement can be tracked rather precisely: Patrick Debois is a software developer and consultant with experience in many fields within IT. He was frustrated with the divide between developers and operations personnel. He tried getting people interested in the problem at conferences, but there wasn't much interest initially.

In 2009, there was a well-received talk at the O'Reilly Velocity Conference: "10+ Deploys per Day: Dev and Ops Cooperation at Flickr." Patrick then decided to organize an event in Ghent, Belgium, called DevOpsDays. This time, there was much interest, and the conference was a success. The name "DevOpsDays" struck a chord, and the conference has become a recurring event. DevOpsDays was abbreviated to "DevOps" in conversations on Twitter and various Internet forums.

The DevOps movement has its roots in Agile software development principles. The Agile Manifesto was written in 2001 by a number of individuals wanting to improve the then current status quo of system development and find new ways of working in the software development industry. The following is an excerpt from the Agile Manifesto, the now classic text, which is available on the Web at `http://agilemanifesto.org/`:

> "*Individuals and interactions over processes and tools*
>
> *Working software over comprehensive documentation*
>
> *Customer collaboration over contract negotiation*
>
> *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more.*"

In light of this, DevOps can be said to relate to the first principle, "Individuals and interactions over processes and tools."

This might be seen as a fairly obviously beneficial way to work—why do we even have to state this obvious fact? Well, if you have ever worked in any large organization, you will know that the opposite principle seems to be in operation instead. Walls between different parts of an organization tend to form easily, even in smaller organizations, where at first it would appear to be impossible for such walls to form.

DevOps, then, tends to emphasize that interactions between individuals are very important, and that technology might possibly assist in making these interactions happen and tear down the walls inside organizations. This might seem counterintuitive, given that the first principle favors interaction between people over tools, but my opinion is that any tool can have several effects when used. If we use the tools properly, they can facilitate all of the desired properties of an Agile workplace.

A very simple example might be the choice of systems used to report bugs. Quite often, development teams and quality assurance teams use different systems to handle tasks and bugs. This creates unnecessary friction between the teams and further separates them when they should really focus on working together instead. The operations team might, in turn, use a third system to handle requests for deployment to the organization's servers.

An engineer with a DevOps mindset, on the other hand, will immediately recognize all three systems as being workflow systems with similar properties. It should be possible for everyone in the three different teams to use the same system, perhaps tweaked to generate different views for the different roles. A further benefit would be smaller maintenance costs, since three systems are replaced by one.

Another core goal of DevOps is automation and Continuous Delivery. Simply put, automating repetitive and tedious tasks leaves more time for human interaction, where true value can be created.

How fast is fast?

The turnaround for DevOps processes must be fast. We need to consider time to market in the larger perspective, and simply stay focused on our tasks in the smaller perspective. This line of thought is also held by the Continuous Delivery movement.

As with many things Agile, many of the ideas in DevOps and Continuous Delivery are in fact different names of the same basic concepts. There really isn't any contention between the two concepts; they are two sides of the same coin.

DevOps engineers work on making enterprise processes faster, more efficient, and more reliable. Repetitive manual labor, which is error prone, is removed whenever possible.

It's easy, however, to lose track of the goal when working with DevOps implementations. Doing nothing faster is of no use to anyone. Instead, we must keep track of delivering increased business value.

For instance, increased communication between roles in the organization has clear value. Your product owners might be wondering how the development process is going and are eager to have a look. In this situation, it is useful to be able to deliver incremental improvements of code to the test environments quickly and efficiently. In the test environments, the involved stake holders, such as product owners and, of course, the quality assurance teams, can follow the progress of the development process.

Another way to look at it is this: If you ever feel yourself losing focus because of needless waiting, something is wrong with your processes or your tooling. If you find yourself watching videos of robots shooting balloons during compile time, your compile times are too long!

The same is true for teams idling while waiting for deploys and so on. This idling is, of course, even more expensive than that of a single individual.

While robot shooting practice videos are fun, software development is inspiring too! We should help focus creative potential by eliminating unnecessary overhead.

Homemade Death Ray Laser DRONE BOT!!! Remote Controlled!!

styropyro

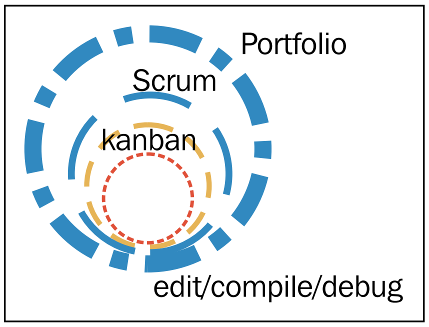
Published on Dec 15, 2013

I present my most terrifying laser creation as of yet...the remote controlled death ray drone bot! This beast packs a potent 2W blue laser that fries anything in it's path. It can walk around and shoot it's death ray at the touch of a button.

A death ray laser robot versus your team's productivity

The Agile wheel of wheels

There are several different cycles in Agile development, from the Portfolio level through to the Scrum and Kanban cycles and down to the Continuous Integration cycle. The emphasis on which cadence work happens in is a bit different depending on which Agile framework you are working with. Kanban emphasizes the 24-hour cycle and is popular in operations teams. Scrum cycles can be between two to four weeks and are often used by development teams using the Scrum Agile process. Longer cycles are also common and are called **Program Increments**, which span several Scrum Sprint cycles, in Scaled Agile Framework.



The Agile wheel of wheels

DevOps must be able to support all these cycles. This is quite natural given the central theme of DevOps: cooperation between disciplines in an Agile organization.

The most obvious and measurably concrete benefits of DevOps occur in the shorter cycles, which in turn make the longer cycles more efficient. Take care of the pennies, and the pounds will take care of themselves, as the old adage goes.

Here are some examples of when DevOps can benefit Agile cycles:

- Deployment systems, maintained by DevOps engineers, make the deliveries at the end of Scrum cycles faster and more efficient. These can take place with a periodicity of two to four weeks.

  In organizations where deployments are done mostly by hand, the time to deploy can be several days. Organizations that have these inefficient deployment processes will benefit greatly from a DevOps mindset.

- The Kanban cycle is 24 hours, and it's therefore obvious that the deployment cycle needs to be much faster than that if we are to succeed with Kanban.

  A well-designed DevOps Continuous Delivery pipeline can deploy code from being committed to the code repository to production in the order of minutes, depending on the size of the change.

Beware the cargo cult Agile fallacy

Richard Feynman was awarded the Nobel Prize for his work in the field of quantum physics in 1965. He noticed a common behavior among scientists, in which they went though all the motions of science but missed some central, vital ingredient of the scientific process. He called this behavior "cargo cult science," since it was reminiscent of the cargo cults in the Melanesian South Sea islands. These cargo cults where formed during the Second World War when the islanders watched great planes land with useful cargo. After the war stopped, the cargo also stopped coming. The islanders started simulating landing strips, doing everything just as they had observed the American military do, in order for the planes to land.

A cargo cult Agile aeroplane

We are not working in an Agile or DevOps-oriented manner simply because we have a morning stand-up where we drink coffee and chat about the weather. We don't have a DevOps pipeline just because we have a Puppet implementation that only the operations team knows anything about.

It is very important that we keep track of our goals and continuously question whether we are doing the right thing and are still on the right track. This is central to all Agile thinking. It is, however, something that is manifestly very hard to do in practice. It is easy to wind up as followers of the cargo cults.

When constructing deployment pipelines, for example, keep in mind why we are building them in the first place. The goal is to allow people to interact with new systems faster and with less work. This, in turn, helps people with different roles interact with each other more efficiently and with less turnaround.

If, on the other hand, we build a pipeline that only helps one group of people achieve their goals, for instance, the operations personnel, we have failed to achieve our basic goal.

While this is not an exact science, it pays to bear in mind that Agile cycles, such as the sprint cycle in the Scrum Agile method, normally have a method to deal with this situation. In Scrum, this is called the sprint retrospective, where the team gets together and discusses what went well and what could have gone better during the sprint. Spend some time here to make sure you are doing the right thing in your daily work.

A common problem here is that the output from the sprint retrospective isn't really acted upon. This, in turn, may be caused by the unfortunate fact that the identified problems were really caused by some other part of the organization that you don't communicate well with. Therefore, these problems come up again and again in the retrospectives and are never remedied.

If you recognize that your team is in this situation, you will benefit from the DevOps approach since it emphasizes cooperation between roles in the organization.

To summarize, try to use the mechanisms provided in the Agile methods in your methods themselves. If you are using Scrum, use the sprint retrospective mechanism to capture potential improvements. This being said, don't take the methods as gospel. Find out what works for you.

DevOps and ITIL

This section explains how DevOps and other ways of working coexist and fit together in a larger whole.

DevOps fits well together with many frameworks for Agile or Lean enterprises. Scaled Agile Framework, or SAFe® , specifically mentions DevOps. There is nearly never any disagreement between proponents of different Agile practices and DevOps since DevOps originated in the Agile environments. The story is a bit different with ITIL, though.

ITIL, which was formerly known as Information Technology Infrastructure Library, is a practice used by many large and mature organizations.

ITIL is a large framework that formalizes many aspects of the software life cycle. While DevOps and Continuous Delivery hold the view that the changesets we deliver to production should be small and happen often, at first glance, ITIL would appear to hold the opposite view. It should be noted that this isn't really true. Legacy systems are quite often monolithic, and in these cases, you need a process such as ITIL to manage the complex changes often associated with large monolithic systems.

If you are working in a large organization, the likelihood that you are working with such large monolithic legacy systems is very high.

In any case, many of the practices described in ITIL translate directly into corresponding DevOps practices. ITIL prescribes a configuration management system and a configuration management database. These types of systems are also integral to DevOps, and several of them will be described in this book.

Summary

This chapter presented a brief overview of the background of the DevOps movement. We discussed the history of DevOps and its roots in development and operations, as well as in the Agile movement. We also took a look at how ITIL and DevOps might coexist in larger organizations. The cargo cult anti-pattern was explored, and we discussed how to avoid it. You should now be able to answer where DevOps fits into a larger Agile context and the different cycles of Agile development.

We will gradually move toward more technical and hands-on subjects. The next chapter will present an overview of what the technical systems we tend to focus on in DevOps look like.

2
A View from Orbit

The DevOps process and Continuous Delivery pipelines can be very complex. You need to have a grasp of the intended final results before starting the implementation.

This chapter will help you understand how the various systems of a Continuous Delivery pipeline fit together, forming a larger whole.

In this chapter, we will read about:

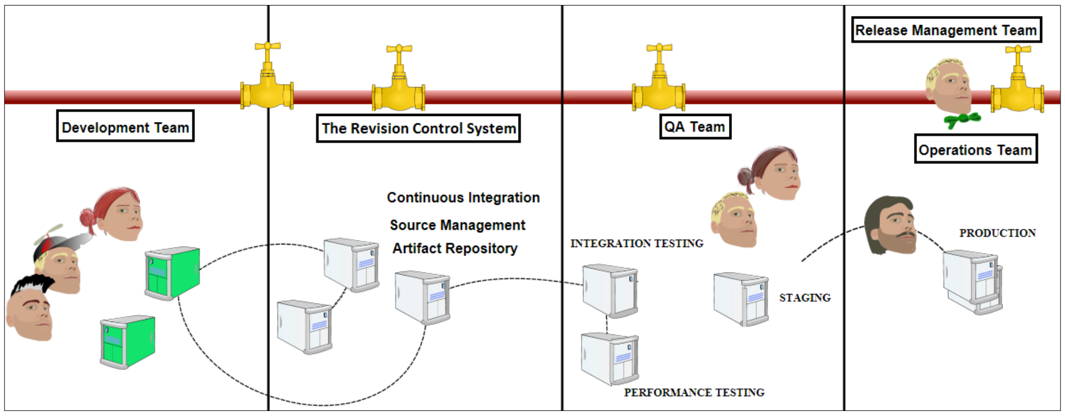
- An overview of the DevOps process, a Continuous Delivery pipeline implementation, and the participants in the process
- Release management
- Scrum, Kanban, and the delivery pipeline
- Bottlenecks

The DevOps process and Continuous Delivery – an overview

There is a lot of detail in the following overview image of the Continuous Delivery pipeline, and you most likely won't be able to read all the text. Don't worry about this just now; we are going to delve deeper into the details as we go along.

For the time being, it is enough to understand that when we work with DevOps, we work with large and complex processes in a large and complex context.

An example of a Continuous Delivery pipeline in a large organization is introduced in the following image:



While the basic outline of this image holds true surprisingly often, regardless of the organization. There are, of course, differences, depending on the size of the organization and the complexity of the products that are being developed.

The early parts of the chain, that is, the developer environments and the Continuous Integration environment, are normally very similar.

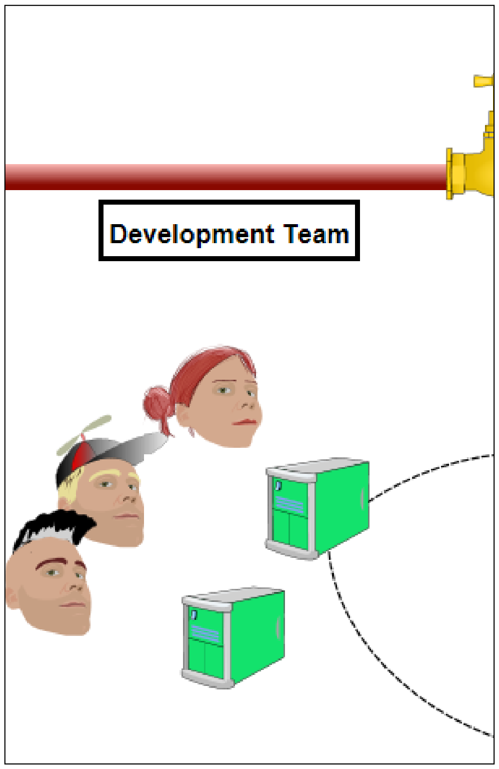
The number and types of testing environments vary greatly. The production environments also vary greatly.

In the following sections, we will discuss the different parts of the Continuous Delivery pipeline.

The developers

The developers (on the far left in the figure) work on their workstations. They develop code and need many tools to be efficient.

The following detail from the previous larger Continuous Delivery pipeline overview illustrates the development team.



Ideally, they would each have production-like environments available to work with locally on their workstations or laptops. Depending on the type of software that is being developed, this might actually be possible, but it's more common to simulate, or rather, mock, the parts of the production environments that are hard to replicate. This might, for example, be the case for dependencies such as external payment systems or phone hardware.

When you work with DevOps, you might, depending on which of its two constituents you emphasized on in your original background, pay more or less attention to this part of the Continuous Delivery pipeline. If you have a strong developer background, you appreciate the convenience of a prepackaged developer environment, for example, and work a lot with those. This is a sound practice, since otherwise developers might spend a lot of time creating their development environments. Such a prepackaged environment might, for instance, include a specific version of the Java Development Kit and an integrated development environment, such as Eclipse. If you work with Python, you might package a specific Python version, and so on.

Keep in mind that we essentially need two or more separately maintained environments. The preceding developer environment consists of all the development tools we need. These will not be installed on the test or production systems. Further, the developers also need some way to deploy their code in a production-like way. This can be a virtual machine provisioned with Vagrant running on the developer's machine, a cloud instance running on AWS, or a Docker container: there are many ways to solve this problem.

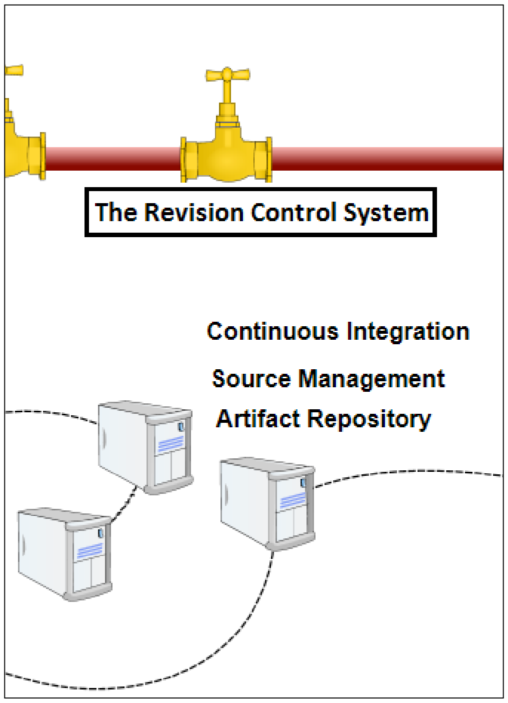
My personal preference is to use a development environment that is similar to the production environment. If the production servers run Red Hat Linux, for instance, the development machine might run CentOS Linux or Fedora Linux. This is convenient because you can use much of the same software that you run in production locally and with less hassle. The compromise of using CentOS or Fedora can be motivated by the license costs of Red Hat and also by the fact that enterprise distributions usually lag behind a bit with software versions.

If you are running Windows servers in production, it might also be more convenient to use a Windows development machine.

The revision control system

The revision control system is often the heart of the development environment. The code that forms the organization's software products is stored here. It is also common to store the configurations that form the infrastructure here. If you are working with hardware development, the designs might also be stored in the revision control system.

The following image shows the systems dealing with code, Continuous Integration, and artifact storage in the Continuous Delivery pipeline in greater detail:



For such a vital part of the organization's infrastructure, there is surprisingly little variation in the choice of product. These days, many use Git or are switching to it, especially those using proprietary systems reaching end-of-life.

Regardless of the revision control system you use in your organization, the choice of product is only one aspect of the larger picture.

You need to decide on directory structure conventions and which branching strategy to use.

If you have a great deal of independent components, you might decide to use a separate repository for each of them.

Since the revision control system is the heart of the development chain, many of its details will be covered in *Chapter 5*, *Building the Code*.

The build server

The build server is conceptually simple. It might be seen as a glorified egg timer that builds your source code at regular intervals or on different triggers.

The most common usage pattern is to have the build server listen to changes in the revision control system. When a change is noticed, the build server updates its local copy of the source from the revision control system. Then, it builds the source and performs optional tests to verify the quality of the changes. This process is called Continuous Integration. It will be explored in more detail in *Chapter 5*, *Building the Code*.

Unlike the situation for code repositories, there hasn't yet emerged a clear winner in the build server field.

In this book, we will discuss Jenkins, which is a widely used open source solution for build servers. Jenkins works right out of the box, giving you a simple and robust experience. It is also fairly easy to install.

The artifact repository

When the build server has verified the quality of the code and compiled it into deliverables, it is useful to store the compiled binary artifacts in a repository. This is normally not the same as the revision control system.

In essence, these binary code repositories are filesystems that are accessible over the HTTP protocol. Normally, they provide features for searching and indexing as well as storing metadata, such as various type identifiers and version information about the artifacts.

In the Java world, a pretty common choice is Sonatype Nexus. Nexus is not limited to Java artifacts, such as Jars or Ears, but can also store artifacts of the operating system type, such as RPMs, artifacts suitable for JavaScript development, and so on.

Amazon S3 is a key-value datastore that can be used to store binary artifacts. Some build systems, such as Atlassian Bamboo, can use Amazon S3 to store artifacts. The S3 protocol is open, and there are open source implementations that can be deployed inside your own network. One such possibility is the Ceph distributed filesystem, which provides an S3-compatible object store.

Package managers, which we will look at next, are also artifact repositories at their core.

Package managers

Linux servers usually employ systems for deployment that are similar in principle but have some differences in practice.

Red Hat-like systems use a package format called RPM. Debian-like systems use the `.deb` format, which is a different package format with similar abilities. The deliverables can then be installed on servers with a command that fetches them from a binary repository. These commands are called package managers.

On Red Hat systems, the command is called `yum`, or, more recently, `dnf`.
On Debian-like systems, it is called `aptitude`/`dpkg`.

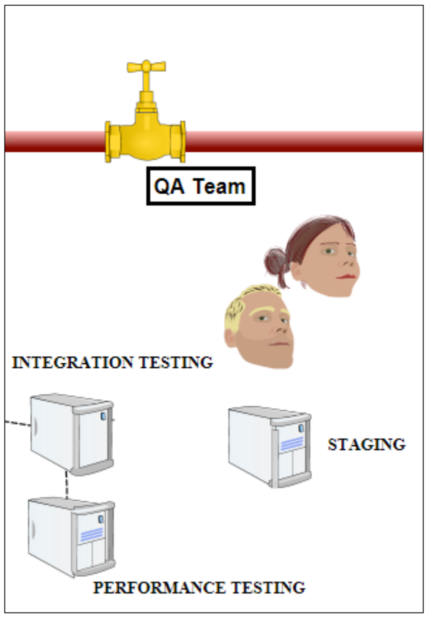
The great benefit of these package management systems is that it is easy to install and upgrade a package; dependencies are installed automatically.

If you don't have a more advanced system in place, it would be feasible to log in to each server remotely and then type `yum upgrade` on each one. The newest packages would then be fetched from the binary repository and installed. Of course, as we will see, we do indeed have more advanced systems of deployment available; therefore, we won't need to perform manual upgrades.

Test environments

After the build server has stored the artifacts in the binary repository, they can be installed from there into test environments.

The following figure shows the test systems in greater detail:



Test environments should normally attempt to be as production-like as is feasible. Therefore, it is desirable that the they be installed and configured with the same methods as production servers.

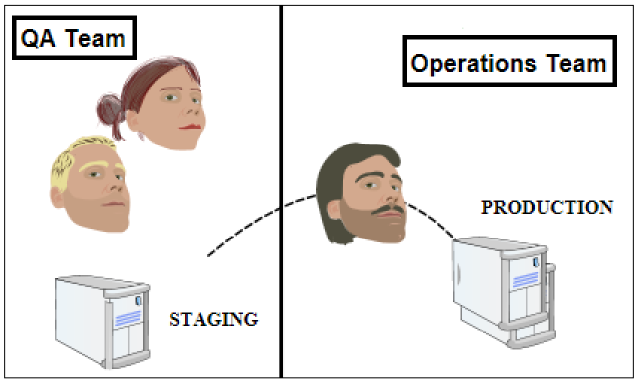
Staging/production

Staging environments are the last line of test environments. They are interchangeable with production environments. You install your new releases on the staging servers, check that everything works, and then swap out your old production servers and replace them with the staging servers, which will then become the new production servers. This is sometimes called the blue-green deployment strategy.

The exact details of how to perform this style of deployment depend on the product being deployed. Sometimes, it is not possible to have several production systems running in parallel, usually because production systems are very expensive.

At the other end of the spectrum, we might have many hundreds of production systems in a pool. We can then gradually roll out new releases in the pool. Logged-in users stay with the version running on the server they are logged in to. New users log in to servers running new versions of the software.

The following detail from the larger Continuous Delivery image shows the final systems and roles involved:



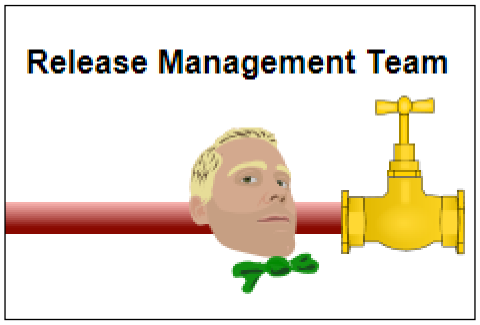
Not all organizations have the resources to maintain production-quality staging servers, but when it's possible, it is a nice and safe way to handle upgrades.

Release management

We have so far assumed that the release process is mostly automatic. This is the dream scenario for people working with DevOps.

This dream scenario is a challenge to achieve in the real world. One reason for this is that it is usually hard to reach the level of test automation needed in order to have complete confidence in automated deploys. Another reason is simply that the cadence of business development doesn't always the match cadence of technical development. Therefore, it is necessary to enable human intervention in the release process.

A faucet is used in the following figure to symbolize human interaction—in this case, by a dedicated release manager.



How this is done in practice varies, but deployment systems usually have a way to support how to describe which software versions to use in different environments.

The integration test environments can then be set to use the latest versions that have been deployed to the binary artifact repository. The staging and production servers have particular versions that have been tested by the quality assurance team.

Scrum, Kanban, and the delivery pipeline

How does the Continuous Delivery pipeline that we have described in this chapter support Agile processes such as Scrum and Kanban?

Scrum focuses on sprint cycles, which can occur biweekly or monthly. Kanban can be said to focus more on shorter cycles, which can occur daily.

The philosophical differences between Scrum and Kanban are a bit deeper, although not mutually exclusive. Many organizations use both Kanban and Scrum together.

From a software-deployment viewpoint, both Scrum and Kanban are similar. Both require frequent hassle-free deployments. From a DevOps perspective, a change starts propagating through the Continuous Delivery pipeline toward test systems and beyond when it is deemed ready enough to start that journey. This might be judged on subjective measurements or objective ones, such as "all unit tests are green."

Our pipeline can manage both the following types of scenarios:

- The build server supports the generation of the objective code quality metrics that we need in order to make decisions. These decisions can either be made automatically or be the basis for manual decisions.
- The deployment pipeline can also be directed manually. This can be handled with an issue management system, via configuration code commits, or both.

So, again, from a DevOps perspective, it doesn't really matter if we use Scrum, Scaled Agile Framework, Kanban, or another method within the lean or Agile frameworks. Even a traditional Waterfall process can be successfully managed—DevOps serves all!

Wrapping up – a complete example

So far, we have covered a lot of information at a cursory level.

To make it more clear, let's have a look at what happens to a concrete change as it propagates through the systems, using an example:

- The development team has been given the responsibility to develop a change to the organization's system. The change revolves around adding new roles to the authentication system. This seemingly simple task is hard in reality because many different systems will be affected by the change.
- To make life easier, it is decided that the change will be broken down into several smaller changes, which will be tested independently and mostly automatically by automated regression tests.
- The first change, the addition of a new role to the authentication system, is developed locally on developer machines and given best-effort local testing. To really know if it works, the developer needs access to systems not available in his or her local environment; in this case, an LDAP server containing user information and roles.
- If test-driven development is used, a failing test is written even before any actual code is written. After the failing test is written, new code that makes the test pass is written.

- The developer checks in the change to the organization's revision control system, a Git repository.
- The build server picks up the change and initiates the build process. After unit testing, the change is deemed fit enough to be deployed to the binary repository, which is a Nexus installation.
- The configuration management system, Puppet, notices that there is a new version of the authentication component available. The integration test server is described as requiring the latest version to be installed, so Puppet goes ahead and installs the new component.
- The installation of the new component now triggers automated regression tests. When these have been finished successfully, manual tests by the quality assurance team commence.
- The quality assurance team gives the change its seal of approval. The change moves on to the staging server, where final acceptance testing commences.
- After the acceptance test phase is completed, the staging server is swapped into production, and the production server becomes the new staging server. This last step is managed by the organization's load-balancing server.

The process is then repeated as needed. As you can see, there is a lot going on!

Identifying bottlenecks

As is apparent from the previous example, there is a lot going on for any change that propagates through the pipeline from development to production. It is important for this process to be efficient.

As with all Agile work, keep track of what you are doing, and try to identify problem areas.

When everything is working as it should, a commit to the code repository should result in the change being deployed to integration test servers within a 15-minute time span.

When things are not working well, a deploy can take days of unexpected hassles. Here are some possible causes:

- Database schema changes.
- Test data doesn't match expectations.
- Deploys are person dependent, and the person wasn't available.

- There is unnecessary red tape associated with propagating changes.
- Your changes aren't small and therefore require a lot of work to deploy safely. This might be because your architecture is basically a monolith.

We will examine these challenges further in the chapters ahead.

Summary

In this chapter, we delved further into the different types of systems and processes you normally work with when doing DevOps work. We gained a deeper, detailed understanding of the Continuous Delivery process, which is at the core of DevOps.

Next, we will look into how the DevOps mindset affects software architecture in order to help us achieve faster and more precise deliveries.

3
How DevOps Affects Architecture

Software architecture is a vast subject, and in this book, we will focus on the aspects of architecture that have the largest effects on Continuous Delivery and DevOps and vice versa.

In this chapter, we will see:

- Aspects of software architecture and what it means to us while working with our DevOps glasses on
- Basic terminology and goals
- Anti-patterns, such as the monolith
- The fundamental principle of the separation of concerns
- Three-tier systems and microservices

We finally conclude with some practical issues regarding database migration.

It's quite a handful, so let's get started!

Introducing software architecture

We will discuss how DevOps affects the architecture of our applications rather than the architecture of software deployment systems, which we discuss elsewhere in the book.

Often while discussing software architecture, we think of the non-functional requirements of our software. By non-functional requirements, we mean different characteristics of the software rather than the requirements on particular behaviors.

A functional requirement could be that our system should be able to deal with credit card transactions. A non-functional requirement could be that the system should be able to manage several such credit cards transactions per second.

Here are two of the non-functional requirements that DevOps and Continuous Delivery place on software architecture:

- We need to be able to deploy small changes often
- We need to be able to have great confidence in the quality of our changes

The normal case should be that we are able to deploy small changes all the way from developers' machines to production in a small amount of time. Rolling back a change because of unexpected problems caused by it should be a rare occurrence.

So, if we take out the deployment systems from the equation for a while, how will the architecture of the software systems we deploy be affected?

The monolithic scenario

One way to understand the issues that a problematic architecture can cause for Continuous Delivery is to consider a counterexample for a while.

Let's suppose we have a large web application with many different functions. We also have a static website inside the application. The entire web application is deployed as a single Java EE application archive. So, when we need to fix a spelling mistake in the static website, we need to rebuild the entire web application archive and deploy it again.

While this might be seen as a silly example, and the enlightened reader would never do such a thing, I have seen this anti-pattern live in the real world. As DevOps engineers, this could be an actual situation that we might be asked to solve.

Let's break down the consequences of this tangling of concerns. What happens when we want to correct a spelling mistake? Let's take a look:

1. We know which spelling mistake we want to correct, but in which code base do we need to do it? Since we have a monolith, we need to make a branch in our code base's revision control system. This new branch corresponds to the code that we have running in production.
2. Make the branch and correct the spelling mistake.
3. Build a new artifact with the correction. Give it a new version.
4. Deploy the new artifact to production.

Okay, this doesn't seem altogether too bad at first glance. But consider the following too:

- We made a change in the monolith that our entire business critical system comprises. If something breaks while we are deploying the new version, we lose revenue by the minute. Are we really sure that the change affects nothing else?
- It turns out that we didn't really just limit the change to correcting a spelling mistake. We also changed a number of version strings when we produced the new artifact. But changing a version string should be safe too, right? Are we sure?

The point here is that we have already spent considerable mental energy in making sure that the change is really safe. The system is so complex that it becomes difficult to think about the effects of changes, even though they might be trivial.

Now, a change is usually more complex than a simple spelling correction. Thus, we need to exercise all aspects of the deployment chain, including manual verification, for all changes to a monolith.

We are now in a place that we would rather not be.

Architecture rules of thumb

There are a number of architecture rules that might help us understand how to deal with the previous undesirable situation.

The separation of concerns

The renowned Dutch computer scientist Edsger Dijkstra first mentioned his idea of how to organize thought efficiently in his paper from 1974, *On the role of scientific thought*.

He called this idea "the separation of concerns". To this date, it is arguably the single most important rule in software design. There are many other well-known rules, but many of them follow from the idea of the separation of concerns. The fundamental principle is simply that we should consider different aspects of a system separately.

The principle of cohesion

In computer science, cohesion refers to the degree to which the elements of a software module belong together.

Cohesion can be used as a measure of how strongly related the functions in a module are.

It is desirable to have strong cohesion in a module.

We can see that strong cohesion is another aspect of the principle of the separation of concerns.

Coupling

Coupling refers to the degree of dependency between two modules. We always want low coupling between modules.

Again, we can see coupling as another aspect of the principle of the separation of concerns.

Systems with high cohesion and low coupling would automatically have separation of concerns, and vice versa.

Back to the monolithic scenario

In the previous scenario with the spelling correction, it is clear that we failed with respect to the separation of concerns. We didn't have any modularization at all, at least from a deployment point of view. The system appears to have the undesirable features of low cohesion and high coupling.

If we had a set of separate deployment modules instead, our spelling correction would most likely have affected only a single module. It would have been more apparent that deploying the change was safe.

How this should be accomplished in practice varies, of course. In this particular example, the spelling corrections probably belong to a frontend web component. At the very least, this frontend component can be deployed separately from the backend components and have their own life cycle.

In the real world though, we might not be lucky enough to always be able to influence the different technologies used by the organization where we work. The frontend might, for instance, be implemented using a proprietary content management system with quirks of its own. Where you experience such circumstances, it would be wise to keep track of the cost such a system causes.

A practical example

Let's now take a look at the concrete example we will be working on for the remainder of the book. In our example, we work for an organization called Matangle. This organization is a software as a service (SaaS) provider that sells access to educational games for schoolchildren.

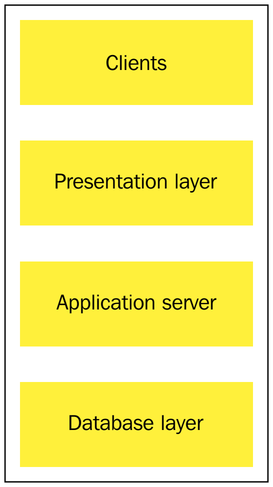
As with any such provider, we will, with all likelihood, have a database containing customer information. It is this database that we will start out with.

The organization's other systems will be fleshed out as we go along, but this initial system serves well for our purposes.

Three-tier systems

The Matangle customer database is a very typical three-tier, **CRUD** (**create, read, update,** and **delete**) type of system. This software architecture pattern has been in use for decades and continues to be popular. These types of systems are very common, and it is quite likely that you will encounter them, either as legacy systems or as new designs.

In this figure, we can see the separation of concerns idea in action:



The three tiers listed as follows show examples of how our organization has chosen to build its system.

The presentation tier

The presentation tier will be a web frontend implemented using the React web framework. It will be deployed as a set of JavaScript and static HTML files. The React framework is fairly recent. Your organization might not use React but perhaps some other framework such as Angular instead. In any case, from a deployment and build point of view, most JavaScript frameworks are similar.

The logic tier

The logic tier is a backend implemented using the Clojure language on the Java platform. The Java platform is very common in large organizations, while smaller organizations might prefer other platforms based on Ruby or Python. Our example, based on Clojure, contains a little bit of both worlds.

The data tier

In our case, the database is implemented with the PostgreSQL database system. PostgreSQL is a relational database management system. While arguably not as common as MySQL installations, larger enterprises might prefer Oracle databases. PostgreSQL is, in any case, a robust system, and our example organization has chosen PostgreSQL for this reason.

From a DevOps point of view, the three-tier pattern looks compelling, at least superficially. It should be possible to deploy changes to each of the three layers separately, which would make it simple to propagate small changes through the servers.

In practice, though, the data tier and logic tier are often tightly coupled. The same might also be true for the presentation tier and logic tier. To avoid this, care must be taken to keep the interfaces between the tiers lean. Using well-known patterns isn't necessarily a panacea. If we don't take care while designing our system, we can still wind up with an undesirable monolithic system.

Handling database migrations

Handling changes in a relational database requires special consideration.

A relational database stores both data and the structure of the data. Upgrading a database schema offers other challenges then the ones present when upgrading program binaries. Usually, when we upgrade an application binary, we stop the application, upgrade it, and then start it again. We don't really bother about the application state. That is handled outside of the application.

When upgrading databases, we do need to consider state, because a database contains comparatively little logic and structure, but contains much state.

In order to describe a database structure change, we issue a command to change the structure.

The database structures before and after a change is applied should be seen as different versions of the database. How do we keep track of database versions?

Liquibase is a database migration system that, at its core, uses a tried and tested method. There are many systems like this, usually at least one for every programming language. Liquibase is well-known in the Java world, and even in the Java world, there are several alternatives that work in a similar manner. Flyway is another example for the Java platform.

Generally, database migration systems employ some variant of the following method:

- Add a table to the database where a database version is stored.
- Keep track of database change commands and bunch them together in versioned changesets. In the case of Liquibase, these changes are stored in XML files. Flyway employs a slightly different method where the changesets are handled as separate SQL files or occasionally as separate Java classes for more complex transitions.
- When Liquibase is being asked to upgrade a database, it looks at the metadata table and determines which changesets to run in order to make the database up-to-date with the latest version.

As previously stated, many database version management systems work like this. They differ mostly in the way the changesets are stored and how they determine which changesets to run. They might be stored in an XML file, like in the case of Liquibase, or as a set of separate SQL files, as with Flyway. This later method is more common with homegrown systems and has some advantages. The Clojure ecosystem also has at least one similar database migration system of its own, called Migratus.

Rolling upgrades

Another thing to consider when doing database migrations is what can be referred to as rolling upgrades. These kinds of deployments are common when you don't want your end user to experience any downtime, or at least very little downtime.

Here is an example of a rolling upgrade for our organization's customer database.

When we start, we have a running system with one database and two servers. We have a load balancer in front of the two servers.

We are going to roll out a change to the database schema, which also affects the servers. We are going to split the customer name field in the database into two separate fields, first name and surname.

This is an incompatible change. How do we minimize downtime? Let's look at the solution:

1. We start out by doing a database migration that creates the two new name fields and then fills these new fields by taking the old name field and splitting the field into two halves by finding a space character in the name. This was the initial chosen encoding for names, which wasn't stellar. This is why we want to change it.

   This change is so far backward compatible, because we didn't remove the name field; we just created two new fields that are, so far, unused.

2. Next, we change the load balancer configuration so that the second of our two servers is no longer accessible from the outside world. The first server chugs along happily, because the old name field is still accessible to the old server code.
3. Now we are free to upgrade server two, since nobody uses it.

   After the upgrade, we start it, and it is also happy because it uses the two new database fields.

4. At this point, we can again switch the load balancer configuration such that server one is not available, and server two is brought online instead. We do the same kind of upgrade on server one while it is offline. We start it and now make both servers accessible again by reinstating our original load balancer configuration.

Now, the change is deployed almost completely. The only thing remaining is removing the old name field, since no server code uses it anymore.

As we can see, rolling upgrades require a lot of work in advance to function properly. It is far easier to schedule upgrades during natural downtimes, if your organization has any. International organizations might not have any suitable natural windows to perform upgrades, and then rolling upgrades might be the only option.

Hello world in Liquibase

This is a simple "hello world" style example for the Liquibase relational database changeset handler.

To try the example out, unpack the source code bundle and run Maven, the Java build tool.

```
cd ch3-liquibase-helloworld
mvn liquibase:update
```

The changelog file

The following is an example of what a Liquibase changelog file can look like.

It defines two changesets, or migrations, with the numerical identifiers 1 and 2:

- Changeset 1 creates a table called customer, with a column called name
- Changeset 2 adds a column called address to the table called customer

```
<?xml version="1.0" encoding="utf-8"?>
<databaseChangeLog xmlns="http://www.liquibase.org/xml/ns/
dbchangelog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.0.xsd">

  <changeSet id="1" author="jave">

    <createTable tableName="customer">
      <column name="id" type="int"/>
      <column name="name" type="varchar(50)"/>
    </createTable>
  </changeSet>
```

```
    <changeSet id="2" author="jave">
      <addColumn  tableName="customer">
          <column name="address" type="varchar(255)"/>
      </addColumn>
    </changeSet>

  </databaseChangeLog>
```

The pom.xml file

The `pom.xml` file, which is a standard Maven project model file, defines things such as the JDBC URL we need so that we can connect to the database we wish to work with as well as the version of the Liquibase plugin.

An H2 database file, `/tmp/liquidhello.h2.db`, will be created. H2 is a convenient in-memory database suitable for testing.

This is the pom.xml file for the "liquibase hello world" example:

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>se.verona.liquibasehello</groupId>
  <artifactId>liquibasehello</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.liquibase</groupId>
        <artifactId>liquibase-maven-plugin</artifactId>
        <version>3.0.0-rc1</version>
        <configuration>
          <changeLogFile>src/main/resources/db-changelog.xml
          </changeLogFile>
          <driver>org.h2.Driver</driver>
          <url>jdbc:h2:liquidhello</url>
        </configuration>
        <dependencies>
          <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.3.171</version>
          </dependency>
        </dependencies>
      </plugin>
    </plugins>
  </build>
</project>
```

If you run the code and everything works correctly, the result will be an H2 database file. H2 has a simple web interface, where you can verify that the database structure is indeed what you expect.

Manual installation

Before we can automate something, we need to understand the corresponding manual process.

Throughout this book, it is assumed that we are using a Red Hat based Linux distribution, such as Fedora or CentOS. Most Linux distributions are similar in principle, except that the command set used for package operations will perhaps differ a bit.

For the exercises, you can either use a physical server or a virtual machine installed in VirtualBox.

First we need the PostgreSQL relational database. Use this command:

```
dnf install postgresql
```

This will check whether there is a PostgreSQL server installed already. Otherwise, it will fetch the PostgreSQL packages from a remote yum repository and install it. So, on reflection, many of the potentially manual steps involved are already automated. We don't need to compile the software, check software versions, install dependencies, and so on. All of this is already done in advance on the Fedora project's build servers, which is very convenient.

For our own organization's software though, we will need to eventually emulate this behavior ourselves.

We will similarly also need a web server, in this case, NGINX. To install it, use the following command:

```
dnf install nginx
```

The `dnf` command replaces `yum` in Red Hat derived distributions. It is a compatible rewrite of `yum` that keeps the same command line interface.

The software that we are deploying, the Matangle customer relationship database, doesn't technically need a separate database and web server as such. A web server called HTTP Kit is already included within the Clojure layer of the software.

Often, a dedicated web server is used in front of servers built on Java, Python, and so on. The primary reason for this is, again, an issue with the separation of concerns; this time, it is not for the separation of logic but for non-functional requirements such as performance, load balancing, and security. A Java based web server might be perfectly capable of serving static content these days, but a native C-based web server such as Apache `httpd` or NGINX still has superior performance and more modest memory requirements. It is also common to use a frontend web server for SSL acceleration or load balancing, for instance.

Now, we have a database and a web server. At this point, we need to build and deploy our organization's application.

On your development machine, perform the following steps in the book's unpacked source archive directory:

```
cd ch3/crm1
```

```
lein build
```

We have now produced a Java archive that can be used to deploy and run the application.

Try out the application:

```
lein run
```

Point a browser to the URL presented in the console to see the web user interface.

How do we deploy the application properly on our servers? It would be nice if we could use the same commands and mechanisms as when we installed our databases and webservers. We will see how we do that in *Chapter 7*, *Deploying the Code*. For now, just running the application from a shell will have to suffice.

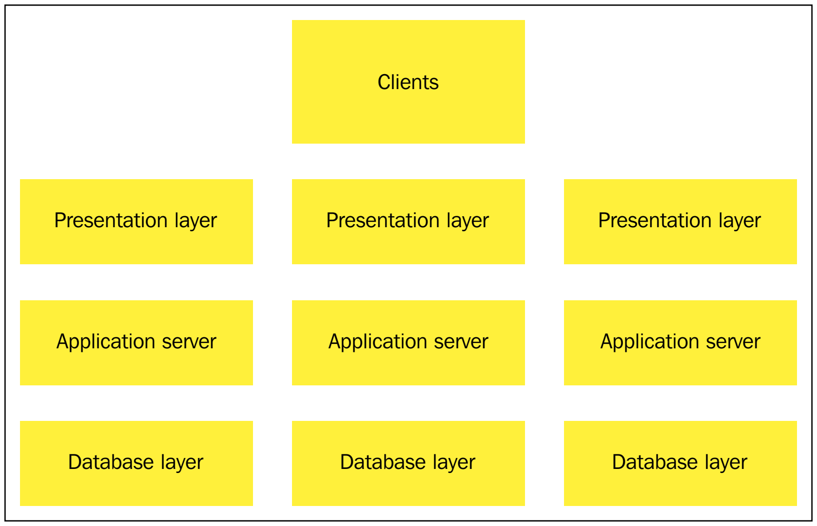
Microservices

Microservices is a recent term used to describe systems where the logic tier of the three-tier pattern is composed of several smaller services that communicate with language-agnostic protocols.

Typically, the language-agnostic protocol is HTTP based, commonly JSON REST, but this is not mandatory. There are several possibilities for the protocol layer.

This architectural design pattern lends itself well to a Continuous Delivery approach since, as we have seen, it's easier to deploy a set of smaller standalone services than a monolith.

Here is an illustration of what a microservices deployment might look like:



We will evolve our example towards the microservices architecture as we go along.

Interlude – Conway's Law

In 1968, Melvin Conway introduced the idea that the structure of an organization that designs software winds up copied in the organization of the software. This is called Conway's Law.

The three-tier pattern, for instance, mirrors the way many organizations' IT departments are structured:

- The database administrator team, or DBA team for short
- The backend developer team
- The frontend developer team
- The operations team

Well, that makes four teams, but we can see the resemblance clearly between the architectural pattern and the organization.

The primary goal of DevOps is to bring different roles together, preferably in cross-functional teams. If Conway's Law holds true, the organization of such teams would be mirrored in their designs.

The microservice pattern happens to mirror a cross-functional team quite closely.

How to keep service interfaces forward compatible

Service interfaces must be allowed to evolve over time. This is natural, since the organization's needs change over time, and service interfaces reflect that to a degree.

How can we accomplish this? One way is to use a pattern that is sometimes called **Tolerant Reader**. This simply means that the consumer of a service should ignore data that it doesn't recognize.

This is a method that lends itself well to REST implementations.

> SOAP, which is an XML schema-based method to define services, is more rigorous. With SOAP, you normally don't change existing interfaces. Interfaces are seen as contracts that should be constant. Instead of changing the interface, you define a new schema version. Existing consumers either have to implement the new protocol and then be deployed again, or the producer needs to provide several versioned endpoints in parallel. This is cumbersome and creates undesirable tighter coupling between providers and consumers.

While the DevOps and Continuous Delivery ideas don't actually do much in the way of mandating how things should be done, the most efficient method is usually favored.

In our example, it can be argued that the least expensive method is to spread the burden of implementing changes over both the producer and consumer. The producer needs to be changed in any case, and the consumer needs to accept a onetime cost of implementing the Tolerant Reader pattern. It is possible to do this with SOAP and XML, but it is less natural than with a REST implementation. This is one of the reasons why REST implementations are more popular in organizations where DevOps and Continuous Delivery have been embraced.

How to implement the Tolerant Reader pattern in practice varies with the platform used. For JsonRest, it's usually sufficient to parse the JSON structure into the equivalent language-specific structure. Then, you pick the parts you need for your application. All other parts, both old and new, are ignored. The limitation of this method is that the producer can't remove the parts of the structure that the consumer relies on. Adding new parts is okay, because they will be ignored.

This again puts a burden on the producer to keep track of the consumer's wishes.

Inside the walls of an organization, this isn't necessarily a big problem. The producer can keep track of copies of the consumer's latest reader code and test that they still work during the producer's build phase.

For services that are exposed to the Internet at large, this method isn't really usable, in which case the more rigorous approach of SOAP is preferable.

Microservices and the data tier

One way of viewing microservices is that each microservice is potentially a separate three-tier system. We don't normally implement each tier for each microservice though. With this in mind, we see that each microservice can implement its own data layer. The benefit would be a potential increase of separation between services.

It is more common in my experience, though, to put all of the organization's data into a single database or at least a single database type. This is more common, but not necessarily better.

There are pros and cons to both scenarios. It is easier to deploy changes when the systems are clearly separate from each other. On the other hand, data modeling is easier when everything is stored in the same database.

DevOps, architecture, and resilience

We have seen that the microservice architecture has many desirable properties from a DevOps point of view. An important goal of DevOps is to place new features in the hands of our user faster. This is a consequence of the greater amount of modularization that microservices provide.

Those who fear that microservices could make life uninteresting by offering a perfect solution without drawbacks can take a sigh of relief, though. Microservices do offer challenges of their own.

We want to be able to deploy new code quickly, but we also want our software to be reliable.

Microservices have more integration points between systems and suffer from a higher possibility of failure than monolithic systems.

Automated testing is very important with DevOps so that the changes we deploy are of good quality and can be relied upon. This is, however, not a solution to the problem of services that suddenly stop working for other reasons. Since we have more running services with the microservice pattern, it is statistically more likely for a service to fail.

We can partially mitigate this problem by making an effort to monitor the services and take appropriate action when something fails. This should preferably be automated.

In our customer database example, we can employ the following strategy:

- We use two application servers that both run our application
- The application offers a special monitoring interface via JsonRest
- A monitoring daemon periodically polls this monitoring interface
- If a server stops working, the load balancer is reconfigured such that the offending server is taken out of the server pool

This is obviously a simple example, but it serves to illustrate the challenges we face when designing resilient systems that comprise many moving parts and how they might affect architectural decisions.

Why do we offer our own application-specific monitoring interface though? Since the purpose of monitoring is to give us a thorough understanding of the current health of the system, we normally monitor many aspects of the application stack. We monitor that the server CPU isn't overloaded, that there is sufficient disk and memory space available, and that the base application server is running. This might not be sufficient to determine whether a service is running properly, though. For instance, the service might for some reason have a broken database access configuration. A service-specific health probing interface would attempt to contact the database and return the status of the connection in the return structure.

It is, of course, best if your organization can agree on a common format for the probing return structure. The structure will also depend on the type of monitoring software used.

Summary

In this chapter, we took a look at the vast subject of software architecture from the viewpoint of DevOps and Continuous Delivery.

We learned about the many different faces that the rule of the separation of concerns can take. We also started working on the deployment strategy for one component within our organization, the Matangle customer database.

We delved into details, such as how to install software from repositories and how to manage database changes. We also had a look at high level subjects such as classic three-tier systems and the more recent microservices concept.

Our next stop will deal with how to manage source code and set up a source code revision control system.

4
Everything is Code

Everything is code, and you need somewhere to store it. The organization's source code management system is that place.

Developers and operations personnel share the same central storage for their different types of code.

There are many ways of hosting the central code repository:

- You can use a software as a service solution, such as GitHub, Bitbucket, or GitLab. This can be cost-effective and provide good availability.
- You can use a cloud provider, such as AWS or Rackspace, to host your repositories.

Some types of organization simply can't let their code leave the building. For them, a private in-house setup is the best choice.

In this chapter, we will explore different options, such as Git, and web-based frontends to Git, such as Gerrit and GitLab.

This exploration will serve to help you find a Git hosting solution that covers your organization's needs.

In this chapter, we will start to experience one of the challenges in the field of DevOps—there are so many solutions to choose from and explore! This is particularly true for the world of source code management, which is central to the DevOps world.

Therefore, we will also introduce the software virtualization tool Docker from a user's perspective so that we can use the tool in our exploration.

The need for source code control

Terence McKenna, an American author, once said that everything is code.

While one might not agree with McKenna about whether everything in the universe can be represented as code, for DevOps purposes, indeed nearly everything can be expressed in codified form, including the following:

- The applications that we build
- The infrastructure that hosts our applications
- The documentation that documents our products

Even the hardware that runs our applications can be expressed in the form of software.

Given the importance of code, it is only natural that the location that we place code, the source code repository, is central to our organization. Nearly everything we produce travels through the code repository at some point in its life cycle.

The history of source code management

In order to understand the central need for source code control, it can be illuminating to have a brief look at the development history of source code management. This gives us an insight into the features that we ourselves might need. Some examples are as follows:

- Storing historical versions of source in separate archives.

  This is the simplest form, and it still lives on to some degree, with many free software projects offering tar archives of older releases to download.

- Centralized source code management with check in and check out.

  In some systems, a developer can lock files for exclusive use. Every file is managed separately. Tools like this include RCS and SCCS.

> Normally, you don't encounter this class of tool anymore, except the occasional file header indicating that a file was once managed by RCS.

- A centralized store where you merge before you commit. Examples include **Concurrent Versions System** (**CVS**) and **Subversion**.

Subversion in particular is still in heavy use. Many organizations have centralized workflows, and Subversion implements such workflows well enough for them.

- A decentralized store.

  On each step of the evolutionary ladder, we are offered more flexibility and concurrency as well as faster and more efficient workflows. We are also offered more advanced and powerful guns to shoot ourselves in the foot with, which we need to keep in mind!

Currently, Git is the most popular tool in this class, but there are many other similar tools in use, such as Bazaar and Mercurial.

Time will tell whether Git and its underlying data model will fend off the contenders to the throne of source code management, who will undoubtedly manifest themselves in the coming years.

Roles and code

From a DevOps point of view, it is important to leverage the natural meeting point that a source code management tool is. Many different roles have a use for source code management in its wider meaning. It is easier to do so for technically-minded roles but harder for other roles, such as project management.

Developers live and breathe source code management. It's their bread and butter.

Operations personnel also favor managing the descriptions of infrastructure in the form of code, scripts, and other artifacts, as we will see in the coming chapters. Such infrastructural descriptors include network topology, versions of software that should be installed on particular servers, and so on.

Quality assurance personnel can store their automated tests in codified form in the source code repository. This is true for software testing frameworks such as Selenium and Junit, among others.

There is a problem with the documentation of the manual steps needed to perform various tasks, though. This is more of a psychological or cultural problem than a technical one.

While many organizations employ a wiki solution such as the wiki engine powering Wikipedia, there is still a lot of documentation floating around in the Word format on shared drives and in e-mails.

This makes documentation hard to find and use for some roles and easy for others. From a DevOps viewpoint, this is regrettable, and an effort should be made so that all roles can have good and useful access to the documentation in the organization.

It is possible to store all documentation in the wiki format in the central source code repository, depending on the wiki engine used.

Which source code management system?

There are many source code management (SCM) systems out there, and since SCM is such an important part of development, the development of these systems will continue to happen.

Currently, there is a dominant system, however, and that system is Git.

Git has an interesting story: it was initiated by Linus Torvalds to move Linux kernel development from BitKeeper, which was a proprietary system used at the time. The license of BitKeeper changed, so it wasn't practical to use it for the kernel anymore.

Git therefore supports the fairly complicated workflow of Linux kernel development and is, at the base technical level, good enough for most organizations.

The primary benefit of Git versus older systems is that it is a distributed version control system (DVCS). There are many other distributed version control systems, but Git is the most pervasive one.

Distributed version control systems have several advantages, including, but not limited to, the following:

- It is possible to use a DVCS efficiently even when not connected to a network. You can take your work with you when traveling on a train or an intercontinental flight.
- Since you don't need to connect to a server for every operation, a DVCS can be faster than the alternatives in most scenarios.
- You can work privately until you feel your work is ready enough to be shared.
- It is possible to work with several remote logins simultaneously, which avoids a single point of failure.

Other distributed version control systems apart from Git include the following:

- **Bazaar**: This is abbreviated as bzr. Bazaar is endorsed and supported by Canonical, which is the company behind Ubuntu. Launchpad, which is Canonical's code hosting service, supports Bazaar.
- **Mercurial**: Notable open source projects such as Firefox and OpenJDK use Mercurial. It originated around the same time as Git.

Git can be complex, but it makes up for this by being fast and efficient. It can be hard to understand, but that can be made easier by using frontends that support different tasks.

A word about source code management system migrations

I have worked with many source code management systems and experienced many transitions from one type of system to another.

Sometimes, much time is spent on keeping all the history intact while performing a migration. For some systems, this effort is well spent, such as for venerable free or open source projects.

For many organizations, keeping the history is not worth the significant expenditure in time and effort. If an older version is needed at some point, the old source code management system can be kept online and referenced. This includes migrations from Visual SourceSafe and ClearCase.

Some migrations are trivial though, such as moving from Subversion to Git. In these cases, historic accuracy need not be sacrificed.

Choosing a branching strategy

When working with code that deploys to servers, it is important to agree on a branching strategy across the organization.

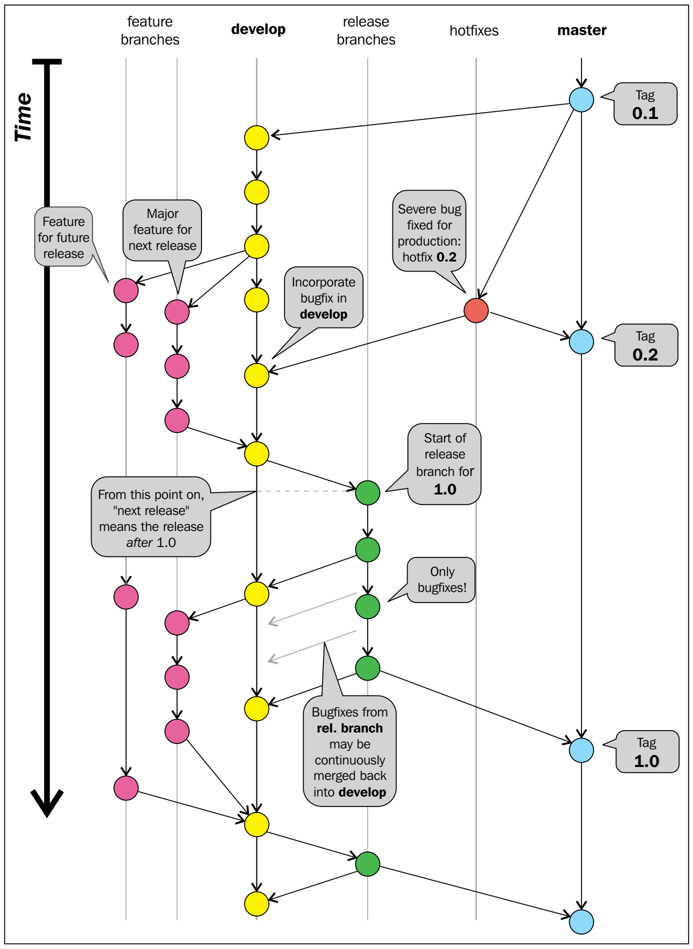
A branching strategy is a convention, or a set of rules, that describes when branches are created, how they are to be named, what use branches should have, and so on.

Branching strategies are important when working together with other people and are, to a degree, less important when you are working on your own, but they still have a purpose.

Most source code management systems do not prescribe a particular branching strategy and neither does Git. The SCM simply gives you the base mechanics to perform branching.

With Git and other distributed version control systems, it is usually pretty cheap to work locally with feature branches. A feature, or topic, branch is a branch that is used to keep track of ongoing development regarding a particular feature, bug, and so on. This way, all changes in the code regarding the feature can be handled together.

There are many well-known branching strategies. Vincent Driessen formalized a branching strategy called **Git flow**, which has many good features. For some, Git flow is too complex, and in those cases, it can be scaled down. There are many such scaled-down models available. This is what Git flow looks like:


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Git flow looks complex, so let's have a brief look at what the branches are for:

- The **master** branch only contains finished work. All commits are tagged, since they represent releases. All releases happen from **master**.
- The **develop** branch is where work on the next release happens. When work is finished here, **develop** is merged to master.
- We use separate **feature branches** for all new features. **Feature branches** are merged to develop.
- When a devastating bug is revealed in production, a **hotfix** branch is made where a bug fix is created. The **hotfix** branch is then merged to **master**, and a new release for production is made.

Git flow is a centralized pattern, and as such, it's reminiscent of the workflows used with Subversion, CVS, and so on. The main difference is that using Git has some technical and efficiency-related advantages.

Another strategy, called the forking pattern, where every developer has a central repository, is rarely used in practice within organizations, except when, for instance, external parties such as subcontractors are being employed.

Branching problem areas

There is a source of contention between Continuous Delivery practices and branching strategies. Some Continuous Delivery methods advocate a single master branch and all releases being made from the master branch. Git flow is one such model.

This simplifies some aspects of deployment, primarily because the branching graph becomes simpler. This in turn leads to simplified testing, because there is only one branch that leads to the production servers.

On the other hand, what if we need to perform a bug fix on released code, and the master has new features we don't want to release yet? This happens when the installation cadence in production is slower than the release cadence of the development teams. It is an undesirable state of affairs, but not uncommon.

There are two basic methods of handling this issue:

- **Make a bug fix branch and deploy to production from it**: This is simpler in the sense that we don't disrupt the development flow. On the other hand, this method might require duplication of testing resources. They might need to mirror the branching strategy.

- **Feature toggling**: Another method that places harder requirements on the developers is feature toggling. In this workflow, you turn off features that are not yet ready for production. This way you can release the latest development version, including the bug fix, together with new features, which will be disabled and inactive.

There are no hard rules, and dogmatism is not helpful in this case.

It is better to prepare for both scenarios and use the method that serves you best in the given circumstances.

Here are some observations to guide you in your choice:

Feature toggling is good when you have changes that are mostly backward compatible or are entirely new. Otherwise, the feature toggling code might become overly complex, catering to many different cases. This, in turn, complicates testing.

Bug fix branches complicate deployment and testing. While it is straightforward to branch and make the bug fix, what should the new version be called, and where do we test it? The testing resources are likely to already be occupied by ongoing development branches.

Some products are simple enough for us to create testing environments on the fly as needed, but this is rarely the case with more complex applications. Sometimes, testing resources are very scarce, such as third party web services that are difficult to copy or even hardware resources.

Artifact version naming

Version numbers become important when you have larger installations.

The following list shows the basic principles of version naming:

- Version numbers should grow monotonically, that is, become larger
- They should be comparable to each other, and it should be easy to see which version is newer
- Use the same scheme for all your artifacts

This usually translates to a version number with three or four parts:

- The first is major—changes here signal major changes in the code
- The second is for minor changes, which are backward API compatible
- The third is for bug fixes
- The fourth can be a build number

While this might seem simple, it is a sufficiently complex area to have created a standardization effort in the form of SemVer, or Semantic Versioning. The full specification can be read at `http://semver.org`.

It is convenient that all installable artifacts have a proper release number and a corresponding tag in the source code management system.

Some tools don't work this way. Maven, the Java build tool, supports a snapshot release number. A snapshot can be seen as a version with a fuzzy last part. Snapshots have a couple of drawbacks, though; for instance, you can't immediately see which source code tag the artifact corresponds to. This makes debugging broken installations harder.

The snapshot artifact strategy further violates a basic testing tenet: the exact same binary artifact that was tested should be the one that is deployed to production.

When you are working with snapshots, you instead test the snapshot until you are done testing, then you release the artifact with a real version number, and that artifact is then deployed. It is no longer the same artifact.

> While the basic idea of snapshots is, of course, that changing a version number has no practical consequences, Murphy's law states that there are, of course, consequences. It is my experience that Murphy is correct in this case, and snapshot versioning creates more harm than good. Instead, use build numbers.

Choosing a client

One of the nice aspects of Git is that it doesn't mandate the use of a particular client. There are several to choose from, and they are all compatible with each other. Most of the clients use one of several core Git implementations, which is good for stability and quality.

Most current development environments have good support for using Git.

In this sense, choosing a client is not something we actually need to do. Most clients work well enough, and the choice can be left to the preferences of those using the client. Many developers use the client integrated in their development environments, or the command-line Git client. When working with operations tasks, the command-line Git client is often preferred because it is convenient to use when working remotely through an SSH shell.

The one exception where we actually have to make a choice is when we assist people in the organization who are new to source code management in general and Git in particular.

In these cases, it is useful to maintain instructions on how to install and use a simple Git client and configure it for our organization's servers.

A simple instruction manual in the organization's wiki normally solves this issue nicely.

Setting up a basic Git server

It's pretty easy to set up a basic Git server. While this is rarely enough in a large organization, it is a good exercise before moving on to more advanced solutions.

Let's first of all specify an overview of the steps we will take and the bits and pieces we will need to complete them:

1. A client machine with two user accounts. The `git` and `ssh` packages should be installed.

   The SSH protocol features prominently as a base transport for other protocols, which is also is the case for Git.

   You need your SSH public keys handy. If you don't have the keys for some reason, use `ssh-keygen` to create them.

   We need two users, because we will simulate two users talking to a central server. Make keys for both test users.

2. A server where the SSH daemon is running.

   This can be the same machine as the one where you simulate the two different client users, or it can be another machine.

3. A Git server user.

   We need a separate Git user who will handle the Git server functionality.

   Now, you need to append the public keys for your two users to the `authorized_keys` file, which is located in the respective user's `.ssh` directory. Copy the keys to this account .

   > Are you starting to get the feeling that all of this is a lot of hassle? This is why we will have a look at solutions that simplify this process later.

4. A bare Git repository.

   Bare Git repositories are a Git peculiarity. They are just Git repositories, except there is no working copy, so they save a bit of space. Here's how to create one:

   ```
   cd /opt/git
   mkdir project.git
   cd project.git
   git init --bare
   ```

5. Now, try cloning, making changes, and pushing to the server.

   Let's review the solution:

   - This solution doesn't really scale very well.

     If you have just a couple of people requiring access, the overhead of creating new projects and adding new keys isn't too bad. It's just a onetime cost. If you have a lot of people in your organization who need access, this solution requires too much work.

   - Solving security issues involve even more hassle.

     While I am of the perhaps somewhat controversial opinion that too much work is spent within organizations limiting employee access to systems, there is no denying that you need this ability in your setup.

     In this solution, you would need to set up separate Git server accounts for each role, and that would be a lot of duplication of effort. Git doesn't have fine-grained user access control out of the box.

Shared authentication

In most organizations, there is some form of a central server for handling authentication. An LDAP server is a fairly common choice. While it is assumed that your organization has already dealt with this core issue one way or the other and is already running an LDAP server of some sort, it is comparatively easy to set up an LDAP server for testing purposes.

One possibility is using 389 Server, named after the port commonly used for the LDAP server, together with the phpLDAPadmin web application for administration of the LDAP server.

Having this test LDAP server setup is useful for the purposes of this book, since we can then use the same LDAP server for all the different servers we are going to investigate.

Hosted Git servers

Many organizations can't use services hosted within another organization's walls at all.

These might be government organizations or organizations dealing with money, such as bank, insurance, and gaming organizations.

The causes might be legal or, simply nervousness about letting critical code leave the organization's doors, so to speak.

If you have no such qualms, it is quite reasonable to use a hosted service, such as GitHub or GitLab, that offers private accounts.

Using GitHub or GitLab is, at any rate, a convenient way to get to learn to use Git and explore its possibilities.

Both vendors are easy to evaluate, given that they offer free accounts where you can get to know the services and what they offer. See if you really need all the services or if you can make do with something simpler.

Some of the features offered by both GitLab and GitHub over plain Git are as follows:

- Web interfaces
- A documentation facility with an inbuilt wiki
- Issue trackers
- Commit visualization

- Branch visualization
- The pull request workflow

While these are all useful features, it's not always the case that you can use the facilities provided. For instance, you might already have a wiki, a documentation system, an issue tracker, and so on that you need to integrate with.

The most important features we are looking for, then, are those most closely related to managing and visualizing code.

Large binary files

GitHub and GitLab are pretty similar but do have some differences. One of them springs from the fact that source code systems like Git traditionally didn't cater much for the storage of large binary files. There have always been other ways, such as storing file paths to a file server in plain text files.

But what if you actually have files that are, in a sense, equivalent to source files except that they are binary, and you still want to version them? Such file types might include image files, video files, audio files, and so on. Modern websites make increasing use of media files, and this area has typically been the domain of content management systems (CMSes). CMSes, however nice they might be, have disadvantages compared to DevOps flows, so the allure of storing media files in your ordinary source handling system is strong. Disadvantages of CMSes include the fact that they often have quirky or nonexistent scripting capabilities. Automation, another word that should have really been included in the "DevOps" portmanteau, is therefore often difficult with a CMS.

You can, of course, just check in your binary to Git, and it will be handled like any other file. What happens then is that Git operations involving server operations suddenly become sluggish. And then, some of the main advantages of Git—efficiency and speed—vanish out the window.

Solutions to this problem have evolved over time, but there are no clear winners yet. The contenders are as follows:

- **Git LFS**, supported by GitHub
- **Git Annex**, supported by GitLab but only in the enterprise edition

Git Annex is the more mature of these. Both solutions are open source and are implemented as extensions to Git via its plugin mechanism.

There are several other systems, which indicates that this is an unresolved pain point in the current state of Git. The Git Annex has a comparison between the different breeds at `http://git-annex.branchable.com/not/`.

> If you need to perform version control of your media files, you should start by exploring Git Annex. It is written in Haskell and is available through the package system of many distributions.
>
> It should also be noted that the primary benefit of this type of solution is the ability to version media files together with the corresponding code. When working with code, you can examine the differences between versions of code conveniently. Examining differences between media files is harder and less useful.
>
> In a nutshell, Git Annex uses a tried and tested solution to data logical problems: adding a layer of indirection. It does this by storing symlinks to files in the repository. The binary files are then stored in the filesystem and fetched by local workspaces using other means, such as rsync. This involves more work to set up the solution, of course.

Trying out different Git server implementations

The distributed nature of Git makes it possible to try out different Git implementations for various purposes. The client-side setup will be similar regardless of how the server is set up.

You can also have several solutions running in parallel. The client side is not unduly complicated by this, since Git is designed to handle several remotes if need be.

Docker intermission

In *Chapter 7*, *Deploying the Code*, we will have a look at a new and exciting way to package our applications, called **Docker**.

In this chapter, we have a similar challenge to solve. We need to be able to try out a couple of different Git server implementations to see which one suits our organization best.

We can use Docker for this, so we will take this opportunity to peek at the possibilities of simplified deployments that Docker offers us.

Since we are going to delve deeper into Docker further along, we will cheat a bit in this chapter and claim that Docker is used to download and run software. While this description isn't entirely untrue, Docker is much more than that. To get started with Docker, follow these steps:

1. To begin with, install Docker according to the particular instructions for your operating system. For Red Hat derivatives, it's a simple `dnf install docker-io` command.

   The `io` suffix might seem a bit mysterious, but there was already a Docker package that implemented desktop functionality, so `docker-io` was chosen instead.

2. Then, the `docker` service needs to be running:

   ```
   systemctl enable docker
   systemctl start docker
   ```

3. We need another tool, Docker Compose, which, at the time of writing, isn't packaged for Fedora. If you don't have it available in your package repositories, follow the instructions on this page `https://docs.docker.com/compose/install/`

Docker Compose is used to automatically start several Docker-packaged applications, such as a database server and a web server, together, which we need for the GitLab example.

Gerrit

A basic Git server is good enough for many purposes.

Sometimes, you need precise control over the workflow, though.

One concrete example is merging changes into configuration code for critical parts of the infrastructure. While my opinion is that it's core to DevOps to not place unnecessary red tape around infrastructure code, there is no denying that it's sometimes necessary. If nothing else, developers might feel nervous committing changes to the infrastructure and would like for someone more experienced to review the code changes.

Gerrit is a Git-based code review tool that can offer a solution in these situations. In brief, Gerrit allows you to set up rules to allow developers to review and approve changes made to a codebase by other developers. These might be senior developers reviewing changes made by inexperienced developers or the more common case, which is simply that more eyeballs on the code is good for quality in general.

Gerrit is Java-based and uses a Java-based Git implementation under the hood.

Gerrit can be downloaded as a Java WAR file and has an integrated setup method. It needs a relational database as a dependency, but you can opt to use an integrated Java-based H2 database that is good enough for evaluating Gerrit.

An even simpler method is using Docker to try out Gerrit. There are several Gerrit images on the Docker registry hub to choose from. The following one was selected for this evaluation exercise: `https://hub.docker.com/r/openfrontier/gerrit/`

To run a Gerrit instance with Docker, follow these steps:

1. Initialize and start Gerrit:

   ```
   docker run -d -p 8080:8080 -p 29418:29418 openfrontier/gerrit
   ```

2. Open your browser to `http://<docker host url>:8080`

   Now, we can try out the code review feature we would like to have.

Installing the git-review package

Install `git-review` on your local installation:

```
sudo dnf install git-review
```

This will install a helper application for Git to communicate with Gerrit. It adds a new command, `git-review`, that is used instead of `git push` to push changes to the Gerrit Git server.

The value of history revisionism

When we work with code together with other people in a team, the code's history becomes more important than when we work on our own. The history of changes to files becomes a way to communicate. This is especially important when working with code review and code review tools such as Gerrit .

The code changes also need to be easy to understand. Therefore, it is useful, although perhaps counterintuitive, to edit the history of the changes in order to make the resulting history clearer.

As an example, consider a case where you made a number of changes and later changed your mind and removed them. It is not useful information for someone else that you made a set of edits and then removed them. Another case is when you have a set of commits that are easier to understand if they are a single commit. Adding commits together in this way is called **squashing** in the Git documentation.

Another case that complicates history is when you merge from the upstream central repository several times, and merge commits are added to the history. In this case, we want to simplify the changes by first removing our local changes, then fetching and applying changes from the upstream repository, and then finally reapplying our local changes. This process is called **rebasing**.

Both squashing and rebasing apply to Gerrit.

The changes should be clean, preferably one commit. This isn't something that is particular to Gerrit; it's easier for a reviewer to understand your changes if they are packaged nicely. The review will be based on this commit.

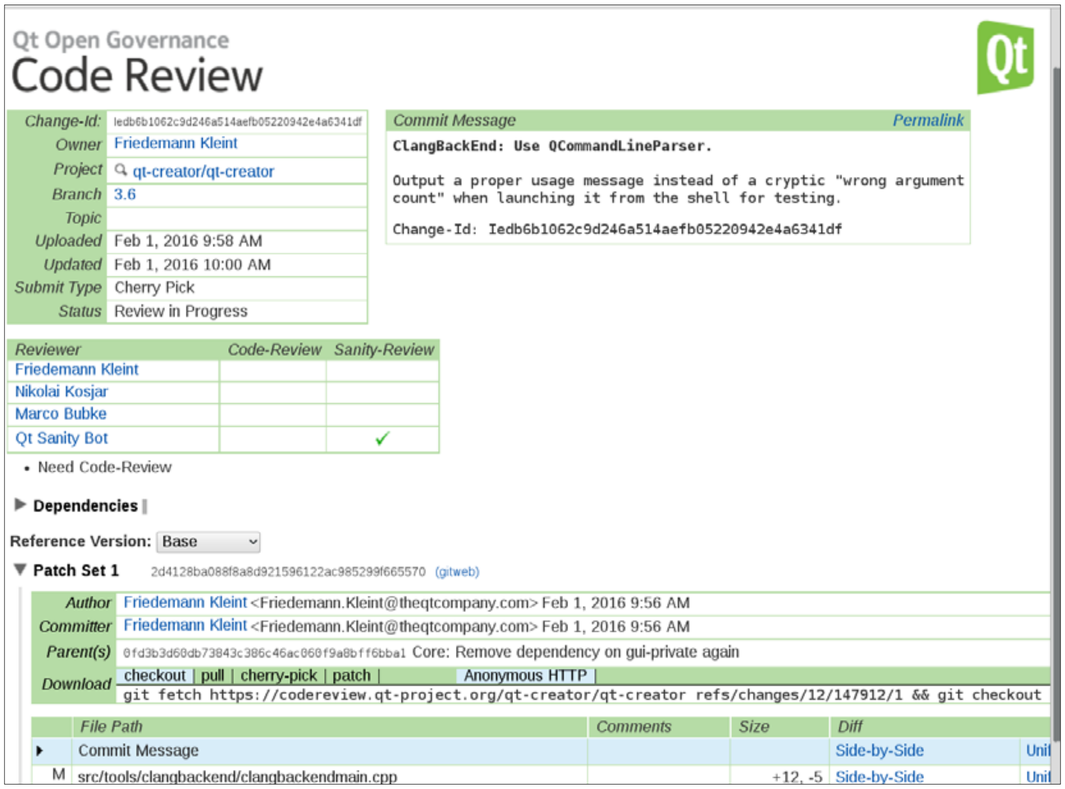
1. To begin with, we need to have the latest changes from the Git/Gerrit server side. We rebase our changes on top of the server-side changes:

   ```
   git pull --rebase origin master
   ```

2. Then, we polish our local commits by squashing them:

   ```
   git rebase -i origin/master
   ```

Now, let's have a look at the Gerrit web interface:



Qt Open Governance

Code Review

Change-Id:	Iedb6b1062c9d246a514aefb05220942e4a6341df
Owner	Friedemann Kleint
Project	qt-creator/qt-creator
Branch	3.6
Topic	
Uploaded	Feb 1, 2016 9:58 AM
Updated	Feb 1, 2016 10:00 AM
Submit Type	Cherry Pick
Status	Review in Progress

Commit Message *Permalink*

```
ClangBackEnd: Use QCommandLineParser.

Output a proper usage message instead of a cryptic "wrong argument
count" when launching it from the shell for testing.

Change-Id: Iedb6b1062c9d246a514aefb05220942e4a6341df
```

Reviewer	*Code-Review*	*Sanity-Review*
Friedemann Kleint		
Nikolai Kosjar		
Marco Bubke		
Qt Sanity Bot		✓

- Need Code-Review

Dependencies

Reference Version: Base

Patch Set 1 2d4128ba088f8a8d921596122ac985299f665570 (gitweb)

Author	Friedemann Kleint <Friedemann.Kleint@theqtcompany.com> Feb 1, 2016 9:56 AM
Committer	Friedemann Kleint <Friedemann.Kleint@theqtcompany.com> Feb 1, 2016 9:56 AM
Parent(s)	0fd3b3d60db73843c386c46ac060f9a8bff6bba1 Core: Remove dependency on gui-private again
Download	checkout | pull | cherry-pick | patch | Anonymous HTTP | `git fetch https://codereview.qt-project.org/qt-creator/qt-creator refs/changes/12/147912/1 && git checkout`

	File Path	*Comments*	*Size*	*Diff*	
	Commit Message			Side-by-Side	Unif
M	src/tools/clangbackend/clangbackendmain.cpp		+12, -5	Side-by-Side	Unif

We can now approve the change, and it is merged to the master branch.

There is much to explore in Gerrit, but these are the basic principles and should be enough to base an evaluation on.

Are the results worth the hassle, though? These are my observations:

- Gerrit allows fine-grained access to sensitive codebases. Changes can go in after being reviewed by authorized personnel.

  This is the primary benefit of Gerrit. If you just want to have mandatory code reviews for unclear reasons, don't. The benefit has to be clear for everyone involved. It's better to agree on other more informal methods of code review than an authoritative system.

- If the alternative to Gerrit is to not allow access to code bases at all, even read-only access, then implement Gerrit.

Some parts of an organization might be too nervous to allow access to things such as infrastructure configuration. This is usually for the wrong reasons. The problem you usually face isn't people taking an interest in your code; it's the opposite.

Sometimes, sensitive passwords are checked in to code, and this is taken as a reason to disallow access to the source. Well, if it hurts, don't do it. Solve the problem that leads to there being passwords in the repositories instead.

The pull request model

There is another solution to the problem of creating workflows around code reviews: the pull request model, which has been made popular by GitHub.

In this model, pushing to repositories can be disallowed except for the repository owners. Other developers are allowed to fork the repository, though, and make changes in their fork. When they are done making changes, they can submit a pull request. The repository owners can then review the request and opt to pull the changes into the master repository.

This model has the advantage of being easy to understand, and many developers have experience in it from the many open source projects on GitHub.

Setting up a system capable of handling a pull request model locally will require something like GitHub or GitLab, which we will look at next.

GitLab

GitLab supports many convenient features on top of Git. It's a large and complex software system based on Ruby. As such, it can be difficult to install, what with getting all the dependencies right and so on.

There is a nice Docker Compose file for GitLab available at `https://registry.hub.docker.com/u/sameersbn/gitlab/`. If you followed the instructions for Docker shown previously, including the installation of `docker-compose`, it's now pretty simple to start a local GitLab instance:

```
mkdir gitlab
cd gitlab
wget https://raw.githubusercontent.com/sameersbn/docker-gitlab/master/
docker-compose.yml
docker-compose up
```

The `docker-compose` command will read the `.yml` file and start all the required services in a default demonstration configuration.

If you read the startup log in the console window, you will notice that three separate application containers have been started: `gitlab postgresql1`, `gitlab redis1`, and `gitlab gitlab1`.

The GitLab container includes the Ruby base web application and Git backend functionality. Redis is distributed key-value store, and PostgreSQL is a relational database.

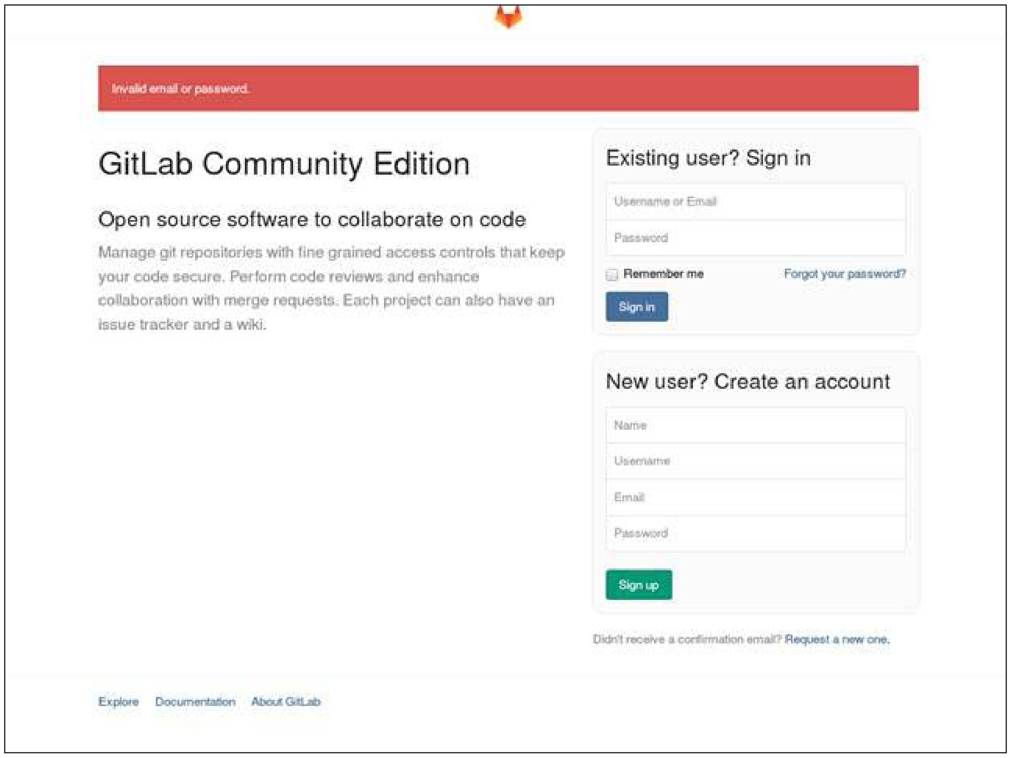
If you are used to setting up complicated server functionality, you will appreciate that we have saved a great deal of time with `docker-compose`.

The `docker-compose.yml` file sets up data volumes at `/srv/docker/gitlab`.

To log in to the web user interface, use the administrator password given with the installation instructions for the GitLab Docker image. They have been replicated here, but beware that they might change as the Docker image author sees fit:

- Username: root
- Password: 5iveL!fe

Here is a screenshot of the GitLab web user interface login screen:



Invalid email or password.

GitLab Community Edition

Open source software to collaborate on code

Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Existing user? Sign in

Username or Email

Password

Remember me

Forgot your password?

Sign in

New user? Create an account

Name

Username

Email

Password

Sign up

Didn't receive a confirmation email? Request a new one.

Explore Documentation About GitLab

Try importing a project to your GitLab server from, for instance, GitHub or a local private project.

Have a look at how GitLab visualizes the commit history and branches.

While investigating GitLab, you will perhaps come to agree that it offers a great deal of interesting functionality.

When evaluating features, it's important to keep in mind whether it's likely that they will be used after all. What core problem would GitLab, or similar software, solve for you?

It turns out that the primary value added by GitLab, as exemplified by the following two features, is the elimination of bottlenecks in DevOps workflows:

- The management of user ssh keys
- The creation of new repositories

These features are usually deemed to be the most useful.

Visualization features are also useful, but the client-side visualization available with Git clients is more useful to developers.

Summary

In this chapter, we explored some of the options available to us for managing our organization's source code. We also investigated areas where decisions need to be made within DevOps, version numbering, and branching strategies.

After having dealt with source code in its pure form, we will next work on building the code into useful binary artifacts.

5
Building the Code

You need a system to build your code, and you need somewhere to build it.

Jenkins is a flexible open source build server that grows with your needs.
Some alternatives to Jenkins will be explored as well.

We will also explore the different build systems and how they affect our DevOps work.

Why do we build code?

Most developers are familiar with the process of building code. When we work in the field of DevOps, however, we might face issues that developers who specialize in programming a particular component type won't necessarily experience.

For the purposes of this book, we define software building as the process of molding code from one form to another. During this process, several things might happen:

- The compilation of source code to native code or virtual machine bytecode, depending on our production platform.
- Linting of the code: checking the code for errors and generating code quality measures by means of static code analysis. The term "Linting" originated with a program called Lint, which started shipping with early versions of the Unix operating system. The purpose of the program was to find bugs in programs that were syntactically correct, but contained suspicious code patterns that could be identified with a different process than compiling.
- Unit testing, by running the code in a controlled manner.
- The generation of artifacts suitable for deployment.

It's a tall order!

Not all code goes through each and every one of these phases. Interpreted languages, for example, might not need compilation, but they might benefit from quality checks.

The many faces of build systems

There are many build systems that have evolved over the history of software development. Sometimes, it might feel as if there are more build systems than there are programming languages.

Here is a brief list, just to get a feeling for how many there are:

- For Java, there is Maven, Gradle, and Ant
- For C and C++, there is Make in many different flavors
- For Clojure, a language on the JVM, there is Leiningen and Boot apart from Maven
- For JavaScript, there is Grunt
- For Scala, there is sbt
- For Ruby, we have Rake
- Finally, of course, we have shell scripts of all kinds

Depending on the size of your organization and the type of product you are building, you might encounter any number of these tools. To make life even more interesting, it's not uncommon for organizations to invent their own build tools.

As a reaction to the complexity of the many build tools, there is also often the idea of standardizing a particular tool. If you are building complex heterogeneous systems, this is rarely efficient. For example, building JavaScript software is just easier with Grunt than it is with Maven or Make, building C code is not very efficient with Maven, and so on. Often, the tool exists for a reason.

Normally, organizations standardize on a single ecosystem, such as Java and Maven or Ruby and Rake. Other build systems besides those that are used for the primary code base are encountered mainly for native components and third-party components.

At any rate, we cannot assume that we will encounter only one build system within our organization's code base, nor can we assume only one programming language.

I have found this rule useful in practice: *it should be possible for a developer to check out the code and build it with minimal surprises on his or her local developer machine*.

This implies that we should standardize the revision control system and have a single interface to start builds locally.

If you have more than one build system to support, this basically means that you need to wrap one build system in another. The complexities of the build are thus hidden and more than one build system at the same time are allowed. Developers not familiar with a particular build can still expect to check it out and build it with reasonable ease.

Maven, for example, is good for declarative Java builds. Maven is also capable of starting other builds from within Maven builds.

This way, the developer in a Java-centric organization can expect the following command line to always build one of the organization's components:

```
mvn clean install
```

One concrete example is creating a Java desktop application installer with the Nullsoft NSIS Windows installation system. The Java components are built with Maven. When the Java artifacts are ready, Maven calls the NSIS installer script to produce a self-contained executable that will install the application on Windows.

While Java desktop applications are not fashionable these days, they continue to be popular in some domains.

The Jenkins build server

A build server is, in essence, a system that builds software based on various triggers. There are several to choose from. In this book, we will have a look at Jenkins, which is a popular build server written in Java.

Jenkins is a fork of the Hudson build server. Kohsuke Kawaguchi was Hudson's principal contributor, and in 2010, after Oracle acquired Hudson, he continued work on the Jenkins fork. Jenkins is clearly the more successful of the two strains today.

Jenkins has special support for building Java code but is in no way limited to just building Java.

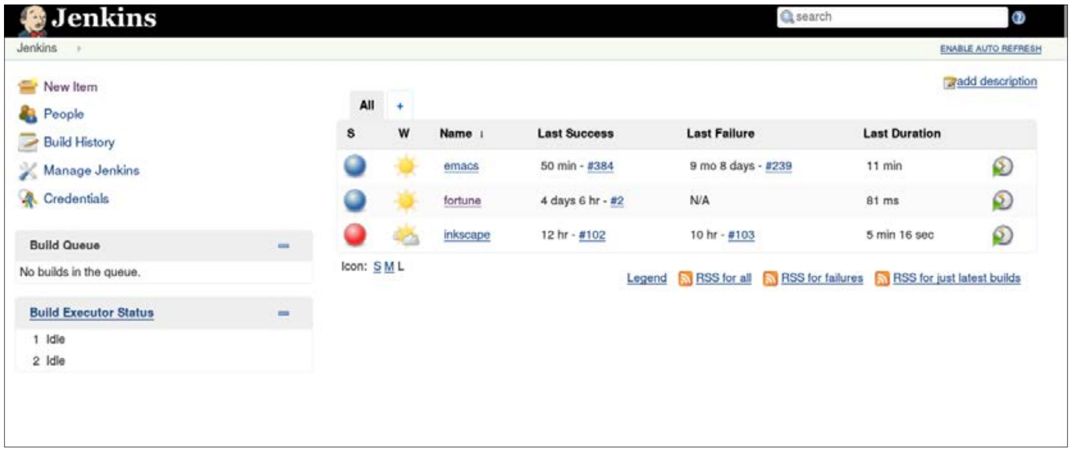
Setting up a basic Jenkins server is not particularly hard at the outset. In Fedora, you can just install it via `dnf`:

```
dnf install jenkins
```

Jenkins is handled as a service via `systemd`:

```
systemctl start jenkins
```

You can now have a look at the web interface at `http://localhost:8080`:



Jenkins

Jenkins › ENABLE AUTO REFRESH

New Item
People
Build History
Manage Jenkins
Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

add description

All +

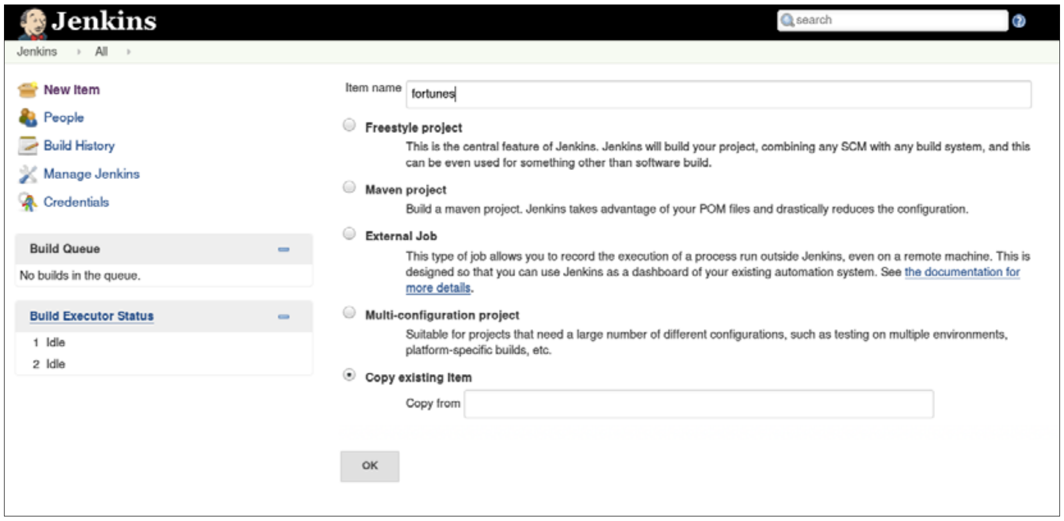
S	W	Name ↓	Last Success	Last Failure	Last Duration
🔵	☀	emacs	50 min - #384	9 mo 8 days - #239	11 min
🔵	☀	fortune	4 days 6 hr - #2	N/A	81 ms
🔴	⛅	inkscape	12 hr - #102	10 hr - #103	5 min 16 sec

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

The Jenkins instance in the screenshot has a couple of jobs already defined. The fundamental entity in Jenkins is the job definition, and there are several types to choose from. Let's create a simple job in the web interface. To keep it simple, this job will just print a classic Unix `fortune` quote:

1. Create a job of the type **Freestyle project**:



Jenkins

Jenkins › All ›

New Item
People
Build History
Manage Jenkins
Credentials

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Item name: fortunes

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

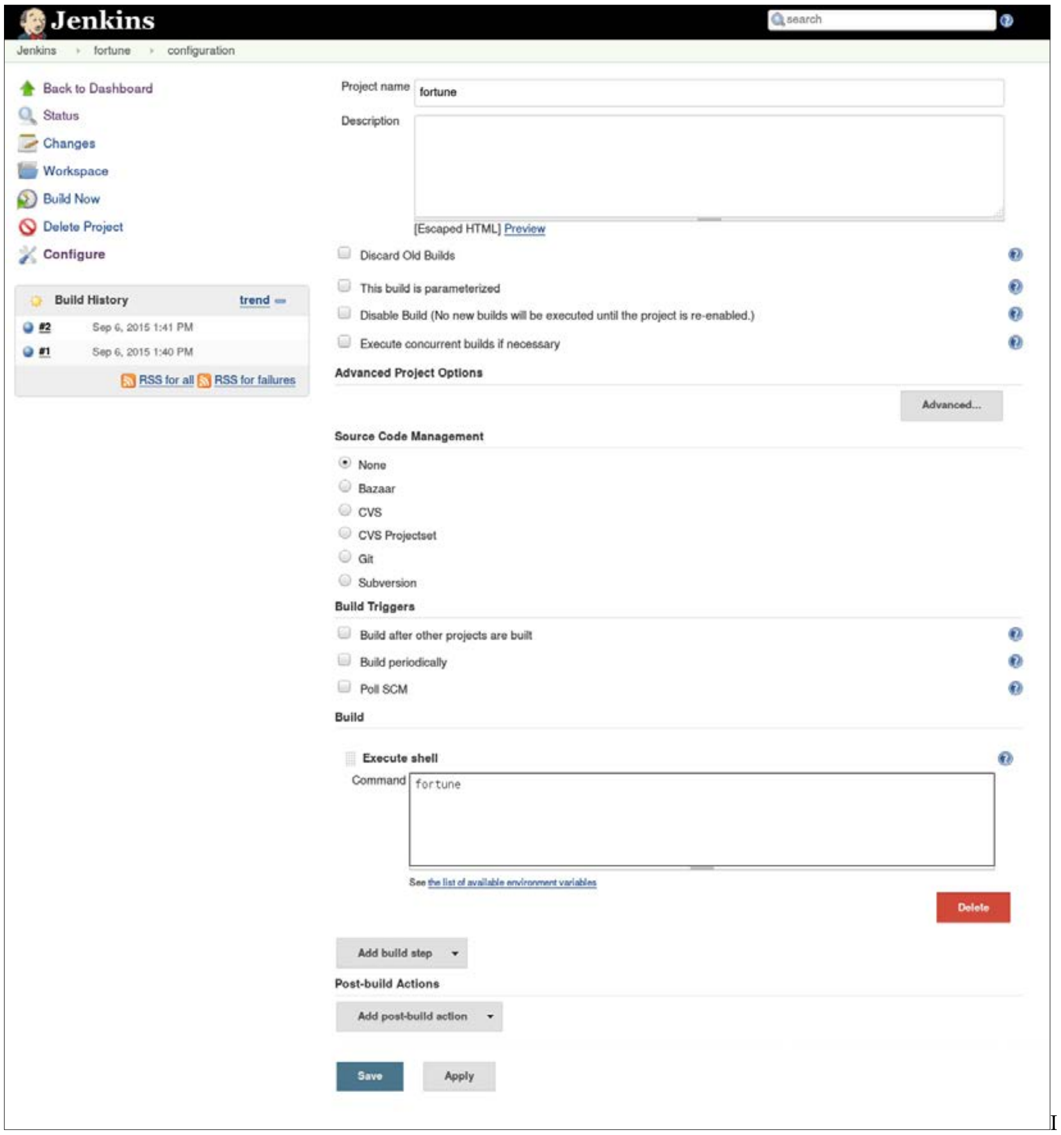
External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

[✓ **Copy existing Item**]
Copy from

OK

2. Add a **shell build** step.
3. In the shell entry (Command), type `fortune`:



Jenkins › fortune › configuration

Project name: fortune

Description:

[Escaped HTML] Preview

- [ ] Discard Old Builds
- [ ] This build is parameterized
- [ ] Disable Build (No new builds will be executed until the project is re-enabled.)
- [ ] Execute concurrent builds if necessary

Advanced Project Options

Source Code Management

- (●) None
- ( ) Bazaar
- ( ) CVS
- ( ) CVS Projectset
- ( ) Git
- ( ) Subversion

Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] Poll SCM

Build

Execute shell

Command:

```
fortune
```

See the list of available environment variables

Post-build Actions

Build History: #2 Sep 6, 2015 1:41 PM; #1 Sep 6, 2015 1:40 PM

Whenever you run the job, a `fortune` quote will be printed in the job log.

Jobs can be started manually, and you will find a history of job executions and can examine each job log. This keeps a history of previous executions, which is very handy when you are trying to figure out which change broke a build and how to fix it.

If you don't have the `fortune` command, install it with `dnf install fortune-mod`, or you might opt to simply run the `date` command instead. This will just output the date in the build log instead of classic quotes and witticisms.

Managing build dependencies

In the previous, simple example, we printed a fortune cookie to the build log.

While this exercise can't be compared in complexity with managing real builds, we at least learned to install and start Jenkins, and if you had issues installing the `fortune` utility, you got a glimpse of the dark underbelly of managing a Continuous Integration server: managing the build dependencies.

Some build systems, such as the Maven tool, are nice in the way that the Maven POM file contains descriptions of which build dependencies are needed, and they are fetched automatically by Maven if they aren't already present on the build server. Grunt works in a similar way for JavaScript builds. There is a build description file that contains the dependencies required for the build. Golang builds can even contain links to GitHub repositories required for completing the build.

C and C++ builds present challenges in a different way. Many projects use GNU Autotools; among them is Autoconf, which adapts itself to the dependencies that are available on the host rather than describing which dependencies they need. So, to build Emacs, a text editor, you first run a configuration script that determines which of the many potential dependencies are available on the build system.

> If an optional dependency is missing, such as image libraries for image support, the optional feature will not be available in the final executable. You can still build the program, but you won't get features that your build machine isn't prepared for.

While this is a useful feature if you want your software to work in many different configurations depending on which system it should run on, it's not often the way we would like our builds to behave in an enterprise setting. In this case, we need to be perfectly sure which features will be available in the end. We certainly don't want bad surprises in the form of missing functionality on our production servers.

The RPM (short for Red Hat Package Manager) system, which is used on systems derived from Red Hat, offers a solution to this problem. At the core of the RPM system is a build descriptor file called a spec file, short for specification file. It lists, among other things, the build dependencies required for a successful build and the build commands and configuration options used. Since a spec file is essentially a macro-based shell script, you can use it to build many types of software. The RPM system also has the idea that build sources should be pristine. The spec file can adapt the source code by patching the source before building it.

The final artifact

After finishing the build using the RPM system, you get an RPM file, which is a very convenient type of deployment artifact for operating systems based on Red Hat. For Debian-based distributions, you get a `.deb` file.

The final output from a Maven build is usually an enterprise archive, or EAR file for short. This contains Java Enterprise applications.

It is final deployment artifacts such as these that we will later deploy to our production servers.

In this chapter, we concern ourselves with building the artifacts required for deployment, and in *Chapter 7*, *Deploying the Code*, we talk about the final deployment of our artifacts.

However, even when building our artifacts, we need to understand how to deploy them. At the moment, we will use the following rule of thumb: OS-level packaging is preferable to specialized packaging. This is my personal preference, and others might disagree.

Let's briefly discuss the background for this rule of thumb as well as the alternatives.

As a concrete example, let's consider the deployment of a Java EAR. Normally, we can do this in several ways. Here are some examples:

- Deploy the EAR file as an RPM package through the mechanisms and channels available in the base operating system
- Deploy the EAR through the mechanisms available with the Java application server, such as JBoss, WildFly, and Glassfish

It might superficially look like it would be better to use the mechanism specific to the Java application server to deploy the EAR file, since it is specific to the application server anyway. If Java development is all you ever do, this might be a reasonable supposition. However, since you need to manage your base operating system anyway, you already have methods of deployment available to you that are possible to reuse.

Also, since it is quite likely that you are not just doing Java development but also need to deploy and manage HTML and JavaScript at the very least, it starts to make sense to use a more versatile method of deployment.

Nearly all the organizations I have experience of have had complicated architectures comprising many different technologies, and this rule of thumb has served well in most scenarios.

The only real exception is in mixed environments where Unix servers coexist with Windows servers. In these cases, the Unix servers usually get to use their preferred package distribution method, and the Windows servers have to limp along with some kind of home-brewed solution. This is just an observation and not a condoning of the situation.

Cheating with FPM

Building operating system deliverables such as RPMs with a spec file is very useful knowledge. However, sometimes you don't need the rigor of a real spec file. The spec file is, after all, optimized for the scenario where you are not yourself the originator of the code base.

There is a Ruby-based tool called FPM, which can generate source RPMs suitable for building, directly from the command line.

The tool is available on GitHub at `https://github.com/jordansissel/fpm`.

On Fedora, you can install FPM like this:

```
yum install rubygems
yum install ruby
yum install ruby-devel gcc
gem install fpm
```

This will install a shell script that wraps the FPM Ruby program.

One of the interesting aspects of FPM is that it can generate different types of package; among the supported types are RPM and Debian.

Here is a simple example to make a "hello world" shell script:

```
#!/bin/sh
echo 'Hello World!'
```

We would like the shell script to be installed in `/usr/local/bin`, so create a directory in your home directory with the following structure:

```
$HOME/hello/usr/local/bin/hello.sh
```

Make the script executable, and then package it:

```
chmod a+x usr/local/bin/hello.sh
fpm -s dir -t rpm -n hello-world -v 1 -C installdir usr
```

This will result in an RPM with the name `hello-world` and version `1`.

To test the package, we can first list the contents and then install it:

```
rpm -qivp hello-world.rpm
rpm -ivh hello-world.rpm
```

The shell script should now be nicely installed in `/usr/local/bin`.

FPM is a very convenient method for creating RPM, Debian, and other package types. It's a little bit like cheating!

Continuous Integration

The principal benefit of using a build server is achieving Continuous Integration. Each time a change in the code base is detected, a build that tests the quality of the newly submitted code is started.

Since there might be many developers working on the code base, each with slightly different versions, it's important to see whether all the different changes work together properly. This is called **integration testing**. If integration tests are too far apart, there is a growing risk of the different code branches diverging too much, and merging is no longer easy. The result is often referred to as "merge hell". It's no longer clear how a developer's local changes should be merged to the master branch, because of divergence between the branches. This situation is very undesirable. The root cause of merge hell is often, perhaps surprisingly, psychological. There is a mental barrier to overcome in order to merge your changes to the mainline. Part of working with DevOps is making things easier and thus reducing the perceived costs associated with doing important work like submitting changes.

Continuous Integration builds are usually performed in a more stringent manner than what developers do locally. These builds take a longer time to perform, but since performant hardware is not so expensive these days, our build server is beefy enough to cope with these builds.

If the builds are fast enough to not be seen as tedious, developers will be enthused to check in often, and integration problems will be found early.

Continuous Delivery

After the Continuous Integration steps have completed successfully, you have shiny new artifacts that are ready to be deployed to servers. Usually, these are test environments set up to behave like production servers.

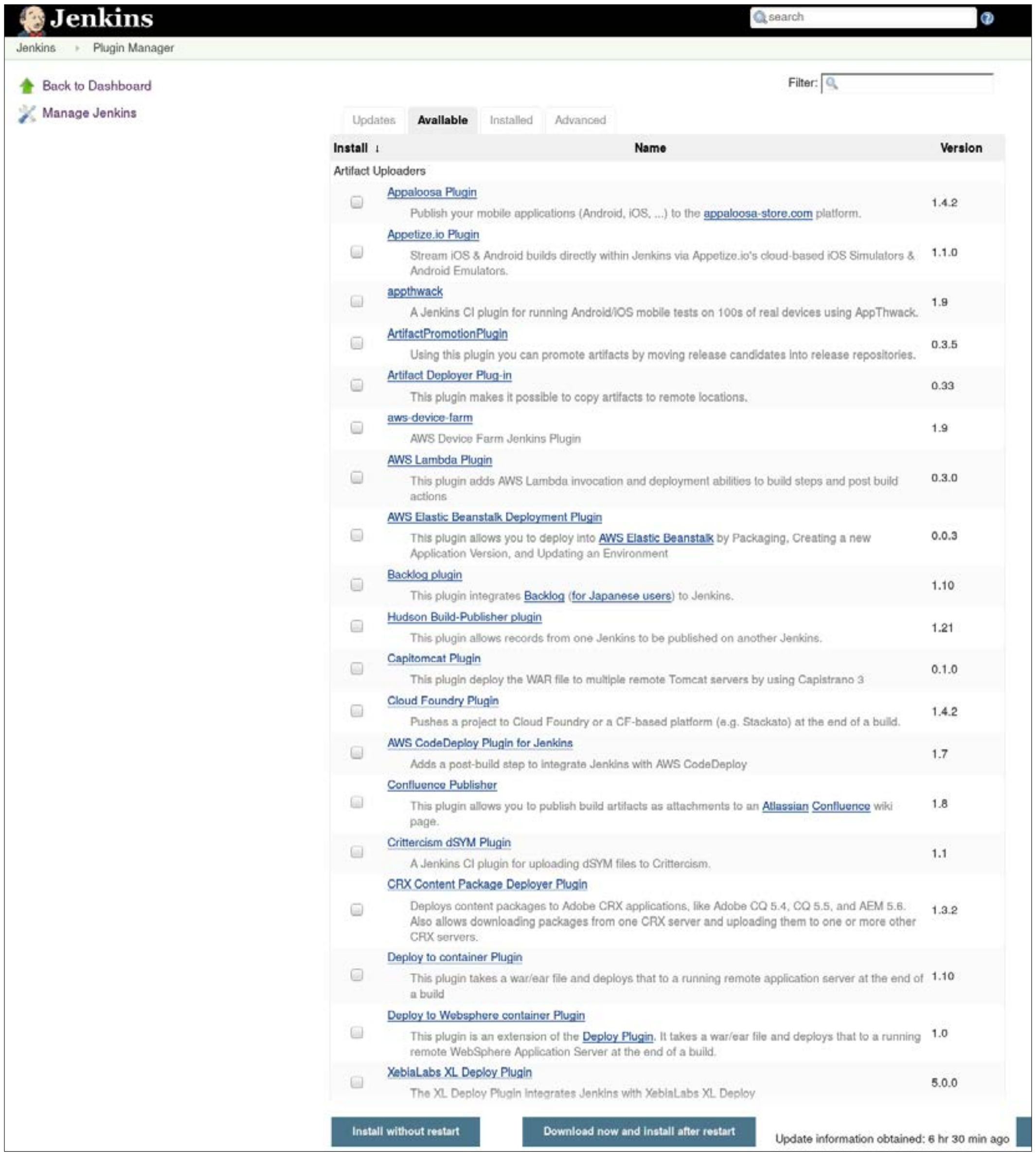
We will discuss deployment system alternatives later in the book.

Often, the last thing a build server does is to deploy the final artifacts from the successful build to an artifact repository. From there, the deployment servers take over the responsibility of deploying them to the application servers. In the Java world, the Nexus repository manager is fairly common. It has support for other formats besides the Java formats, such as JavaScript artifacts and Yum channels for RPMs. Nexus also supports the Docker Registry API now.

Using Nexus for RPM distributions is just one option. You can build Yum channels with a shell script fairly easily.

Jenkins plugins

Jenkins has a plugin system to add functionality to the build server. There are many different plugins available, and they can be installed from within the Jenkins web interface. Many of them can be installed without even restarting Jenkins. This screenshot shows a list of some of the available plugins:



Install ↓	Name	Version
	Artifact Uploaders	
☐	Appaloosa Plugin Publish your mobile applications (Android, iOS, ...) to the appaloosa-store.com platform.	1.4.2
☐	Appetize.io Plugin Stream iOS & Android builds directly within Jenkins via Appetize.io's cloud-based iOS Simulators & Android Emulators.	1.1.0
☐	appthwack A Jenkins CI plugin for running Android/iOS mobile tests on 100s of real devices using AppThwack.	1.9
☐	ArtifactPromotionPlugin Using this plugin you can promote artifacts by moving release candidates into release repositories.	0.3.5
☐	Artifact Deployer Plug-in This plugin makes it possible to copy artifacts to remote locations.	0.33
☐	aws-device-farm AWS Device Farm Jenkins Plugin	1.9
☐	AWS Lambda Plugin This plugin adds AWS Lambda invocation and deployment abilities to build steps and post build actions	0.3.0
☐	AWS Elastic Beanstalk Deployment Plugin This plugin allows you to deploy into AWS Elastic Beanstalk by Packaging, Creating a new Application Version, and Updating an Environment	0.0.3
☐	Backlog plugin This plugin integrates Backlog (for Japanese users) to Jenkins.	1.10
☐	Hudson Build-Publisher plugin This plugin allows records from one Jenkins to be published on another Jenkins.	1.21
☐	Capitomcat Plugin This plugin deploy the WAR file to multiple remote Tomcat servers by using Capistrano 3	0.1.0
☐	Cloud Foundry Plugin Pushes a project to Cloud Foundry or a CF-based platform (e.g. Stackato) at the end of a build.	1.4.2
☐	AWS CodeDeploy Plugin for Jenkins Adds a post-build step to integrate Jenkins with AWS CodeDeploy	1.7
☐	Confluence Publisher This plugin allows you to publish build artifacts as attachments to an Atlassian Confluence wiki page.	1.8
☐	Crittercism dSYM Plugin A Jenkins CI plugin for uploading dSYM files to Crittercism.	1.1
☐	CRX Content Package Deployer Plugin Deploys content packages to Adobe CRX applications, like Adobe CQ 5.4, CQ 5.5, and AEM 5.6. Also allows downloading packages from one CRX server and uploading them to one or more other CRX servers.	1.3.2
☐	Deploy to container Plugin This plugin takes a war/ear file and deploys that to a running remote application server at the end of a build	1.10
☐	Deploy to Websphere container Plugin This plugin is an extension of the Deploy Plugin. It takes a war/ear file and deploys that to a running remote WebSphere Application Server at the end of a build.	1.0
☐	XebiaLabs XL Deploy Plugin The XL Deploy Plugin integrates Jenkins with XebiaLabs XL Deploy	5.0.0

Update information obtained: 6 hr 30 min ago

Among others, we need the Git plugin to poll our source code repositories.

Our sample organization has opted for Clojure for their build, so we will install the Leingingen plugin.

The host server

The build server is usually a pretty important machine for the organization. Building software is processor as well as memory and disk intensive. Builds shouldn't take too long, so you will need a server with good specifications for the build server—with lots of disk space, processor cores, and RAM.

The build server also has a kind of social aspect: it is here that the code of many different people and roles integrates properly for the first time. This aspect grows in importance if the servers are fast enough. Machines are cheaper than people, so don't let this particular machine be the area you save money on.

Build slaves

To reduce build queues, you can add build slaves. The master server will send builds to the slaves based on a round-robin scheme or tie specific builds to specific build slaves.

The reason for this is usually that some builds have certain requirements on the host operating system.

Build slaves can be used to increase the efficiency of parallel builds. They can also be used to build software on different operating systems. For instance, you can have a Linux Jenkins master server and Windows slaves for components that use Windows build tools. To build software for the Apple Mac, it's useful to have a Mac build slave, especially since Apple has quirky rules regarding the deployment of their operating system on virtual servers.

There are several methods to add build slaves to a Jenkins master; see `https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds`.

In essence, there must be a way for the Jenkins master to issue commands to the build slave. This command channel can be the classic SSH method, and Jenkins has a built-in SSH facility. You can also start a Jenkins slave by downloading a Java JNLP client from the master to the slave. This is useful if the build slave doesn't have an SSH server.

A note on cross-compiling

While it's possible to use Windows build slaves, sometimes it's actually easier to use Linux to build Windows software. C compilers such as GCC can be configured to perform cross-compilation using the MinGW package.

Whether or not this is easier very much depends on the software being built.

A big system usually comprises many different parts, and some of the parts might contain native code for different platforms.

Here are some examples:

- Native android components
- Native server components coded in C for efficiency
- Native client components, also coded in C or C++ for efficiency

The prevalence of native code depends a bit on the nature of the organization you are working with. Telecom products often have a lot of native code, such as codecs and hardware interface code. Banking systems might have high-speed messaging systems in native code.

An aspect of this is that it is important to be able to build all the code that's in use conveniently on the build server. Otherwise, there's a tendency for some code to be only buildable on some machine collecting dust under somebody's desk. This is a risk that needs to be avoided.

What your organization's systems need, only you can tell.

Software on the host

Depending on the complexity of your builds, you might need to install many different types of build tool on your build server. Remember that Jenkins is mostly used to trigger builds, not perform the builds themselves. That job is delegated to the build system used, such as Maven or Make.

In my experience, it's most convenient to have a Linux-based host operating system. Most of the build systems are available in the distribution repositories, so it's very convenient to install them from there.

To keep your build server up to date, you can use the same deployment servers that you use to keep your application servers up to date.

Triggers

You can either use a timer to trigger builds, or you can poll the code repository for changes and build if there were changes.

It can be useful to use both methods at the same time:

- Git repository polling can be used most of the time so that every check in triggers a build.
- Nightly builds can be triggered, which are more stringent than continuous builds, and thus take a longer time. Since these builds happen at night when nobody is supposed to work, it doesn't matter if they are slow.
- An upstream build can trigger a downstream build.

You can also let the successful build of one job trigger another job.

Job chaining and build pipelines

It's often useful to be able to chain jobs together. In its simplest form, this works by triggering a second job in the event that the first job finishes successfully. Several jobs can be cascaded this way in a chain. Such a build chain is quite often good enough for many purposes. Sometimes, a nicer visualization of the build steps as well as greater control over the details of the chain is desired.

In Jenkins terminology, the first build in a chain is called the upstream build, and the second one is called the downstream build.

While this way of chaining builds is often sufficient, there will most likely be a need for greater control of the build chain eventually. Such a build chain is often called a pipeline or workflow.

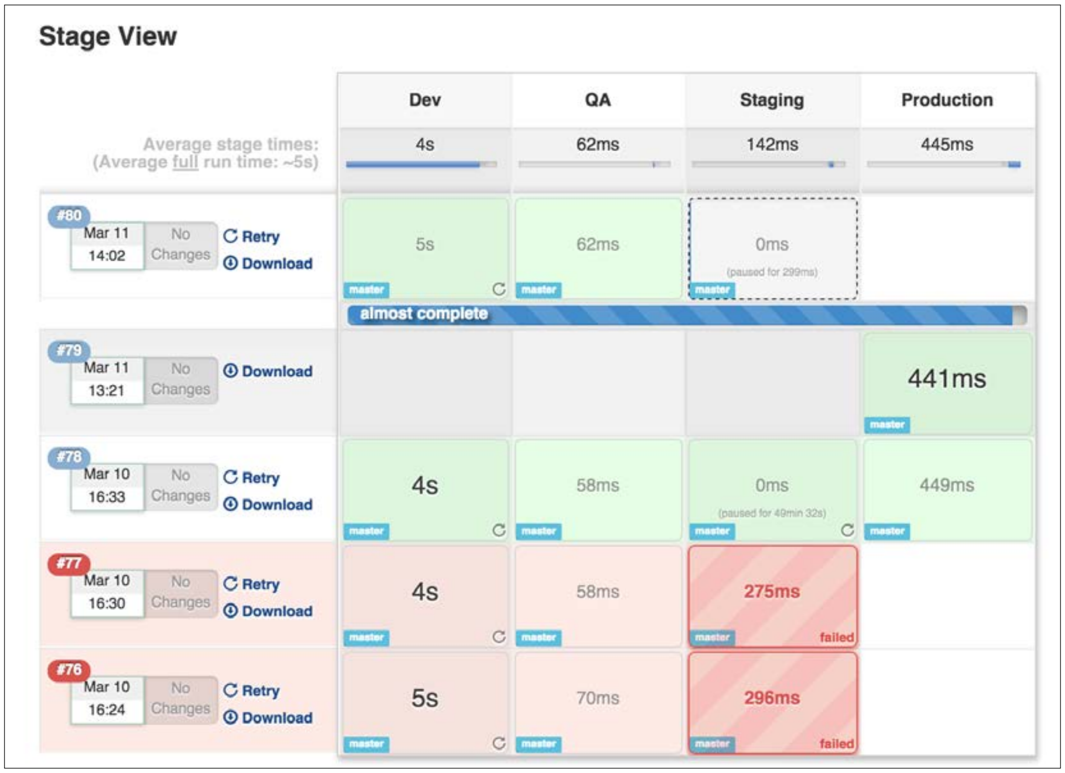
There are many plugins that create improved pipelines for Jenkins, and the fact that there are several shows that there is indeed a great desire for improvements in this area.

Two examples are the multijob plugin and the workflow plugin.

The workflow plugin is the more advanced and also has the advantage that it can be described in a Groovy DSL rather than fiddling about in a web UI.

The workflow plugin is promoted by CloudBees, who are the principal contributors to Jenkins today.

An example workflow is illustrated here:



Stage View

	Dev	QA	Staging	Production
Average stage times: (Average full run time: ~5s)	4s	62ms	142ms	445ms
#80 Mar 11 14:02 No Changes Retry Download	5s master	62ms master	0ms (paused for 299ms) master	
almost complete				
#79 Mar 11 13:21 No Changes Download				441ms master
#78 Mar 10 16:33 No Changes Retry Download	4s master	58ms master	0ms (paused for 49min 32s) master	449ms master
#77 Mar 10 16:30 No Changes Retry Download	4s master	58ms master	275ms master failed	
#76 Mar 10 16:24 No Changes Retry Download	5s master	70ms master	296ms master failed	

When you have a look at the Groovy build script that the workflow plugin uses, you might get the idea that Jenkins is basically a build tool with a web interface, and you would be more or less correct.

A look at the Jenkins filesystem layout

It is often useful to know where builds wind up in the filesystem.

In the case of the Fedora package, the Jenkins jobs are stored here:

```
/var/lib/jenkins/jobs
```

Each job gets its own directory, and the job description XML is stored in this directory as well as a directory for the build called workspace. The job's XML files can be backed up to another server in order to be able to rebuild the Jenkins server in the event of a catastrophic failure. There are dedicated backup plugins for this purpose as well.

Builds can consume a lot of space, so it may sometimes happen that you need to clean out this space manually.

This shouldn't be the normal case, of course. You should configure Jenkins to only leave the number of builds you have space for. You can also configure your configuration management tool to clear out space if needed.

Another reason you might need to delve into the filesystem is when a build mysteriously fails, and you need to debug the cause of the failure. A common cause of this is when the build server state does not meet expectations. For a Maven build, broken dependencies could be polluting the local repository on the build server, for example.

Build servers and infrastructure as code

While we are discussing the Jenkins file structure, it is useful to note an impedance mismatch that often occurs between GUI-based tools such as Jenkins and the DevOps axiom that infrastructure should be described as code.

One way to understand this problem is that while Jenkins job descriptors are text file-based, these text files are not the primary interface for changing the job descriptors. The web interface is the primary interface. This is both a strength and weakness.

It is easy to create ad-hoc solutions on top of existing builds with Jenkins. You don't need to be intimately familiar with Jenkins to do useful work.

On the other hand, the out-of-the-box experience of Jenkins lacks many features that we are used to from the world of programming. Basic features like inheritance and even function definitions take some effort to provide in Jenkins.

The build server feature in GitLab, for example, takes a different approach. Build descriptors are just code right from the start. It is worth checking out this feature in GitLab if you don't need all the possibilities that Jenkins offers.

Building by dependency order

Many build tools have the concept of a build tree where dependencies are built in the order required for the build to complete, since parts of the build might depend on other parts.

In Make-like tools, this is described explicitly; for instance, like this:

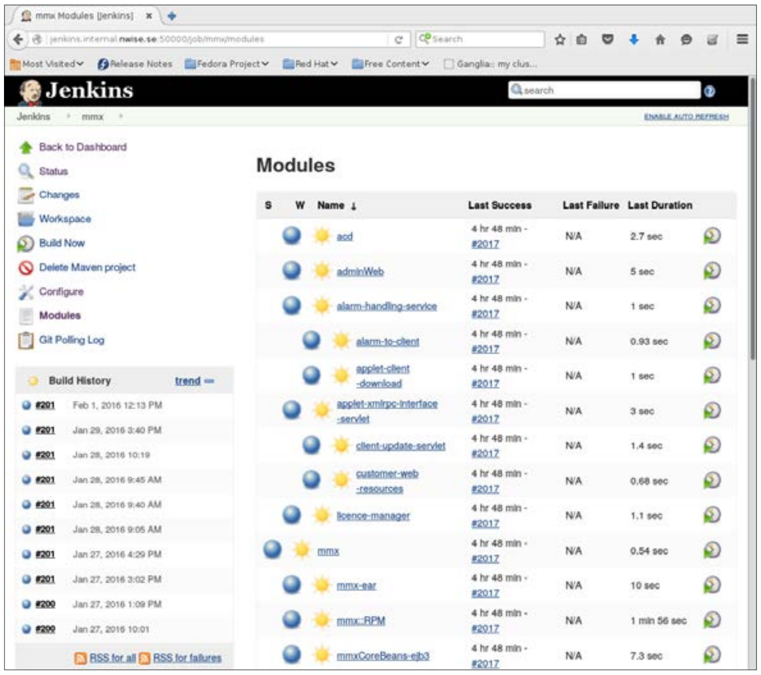
```
a.out : b.o c.o
b.o : b.c
c.o : c.c
```

So, in order to build `a.out`, `b.o` and `c.o` must be built first.

In tools such as Maven, the build graph is derived from the dependencies we set for an artifact. Gradle, another Java build tool, also creates a build graph before building.

Jenkins has support for visualizing the build order for Maven builds, which is called the **reactor** in Maven parlance, in the web user interface.

This view is not available for Make-style builds, however.



#201	Feb 1, 2016 12:13 PM
#201	Jan 29, 2016 3:40 PM
#201	Jan 28, 2016 10:19
#201	Jan 28, 2016 9:45 AM
#201	Jan 28, 2016 9:40 AM
#201	Jan 28, 2016 9:05 AM
#201	Jan 27, 2016 4:29 PM
#201	Jan 27, 2016 3:02 PM
#200	Jan 27, 2016 1:09 PM
#200	Jan 27, 2016 10:01

Modules

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	acd	4 hr 48 min - #2017	N/A	2.7 sec
●	☀	adminWeb	4 hr 48 min - #2017	N/A	5 sec
●	☀	alarm-handling-service	4 hr 48 min - #2017	N/A	1 sec
●	☀	alarm-to-client	4 hr 48 min - #2017	N/A	0.93 sec
●	☀	applet-client -download	4 hr 48 min - #2017	N/A	1 sec
●	☀	applet-xmlrpc-interface -servlet	4 hr 48 min - #2017	N/A	3 sec
●	☀	client-update-servlet	4 hr 48 min - #2017	N/A	1.4 sec
●	☀	customer-web -resources	4 hr 48 min - #2017	N/A	0.68 sec
●	☀	licence-manager	4 hr 48 min - #2017	N/A	1.1 sec
●	☀	mmx	4 hr 48 min - #2017	N/A	0.54 sec
●	☀	mmx-ear	4 hr 48 min - #2017	N/A	10 sec
●	☀	mmx::RPM	4 hr 48 min - #2017	N/A	1 min 56 sec
●	☀	mmxCoreBeans-ejb3	4 hr 48 min - #2017	N/A	7.3 sec

Build phases

One of the principal benefits of the Maven build tool is that it standardizes builds.

This is very useful for a large organization, since it won't need to invent its own build standards. Other build tools are usually much more lax regarding how to implement various build phases. The rigidity of Maven has its pros and cons. Sometimes, people who get started with Maven reminisce about the freedom that could be had with tools such as Ant.

You can implement these build phases with any tool, but it's harder to keep the habit going when the tool itself doesn't enforce the standard order: building, testing, and deploying.

We will examine testing in more detail in a later chapter, but we should note here that the testing phase is very important. The Continuous Integration server needs to be very good at catching errors, and automated testing is very important for achieving that goal.

Alternative build servers

While Jenkins appears to be pretty dominant in the build server scene in my experience, it is by no means alone. Travis CI is a hosted solution that is popular among open source projects. Buildbot is a buildserver that is written in, and configurable with, Python. The Go server is another one, from ThoughtWorks. Bamboo is an offering from Atlassian. GitLab also supports build server functionality now.

Do shop around before deciding on which build server works best for you.

When evaluating different solutions, be aware of attempts at vendor lock-in. Also keep in mind that the build server does not in any way replace the need for builds that are well behaved locally on a developer's machine.

Also, as a common rule of thumb, see if the tool is configurable via configuration files. While management tends to be impressed by graphical configuration, developers and operations personnel rarely like being forced to use a tool that can only be configured via a graphical user interface.

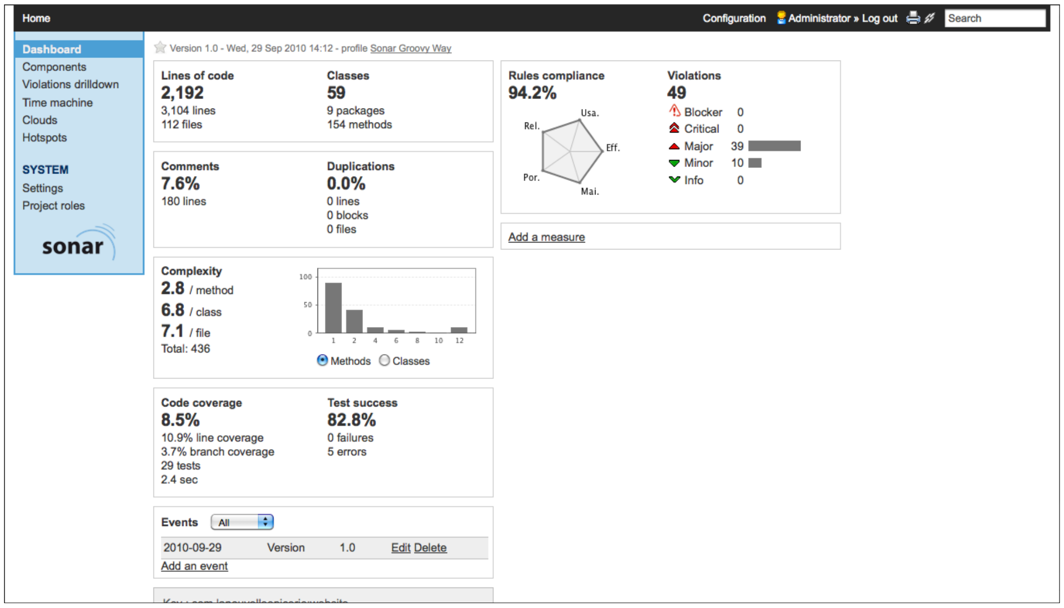
Collating quality measures

A useful thing that a build server can do is the collation of software quality metrics. Jenkins has some support for this out of the box. Java unit tests are executed and can be visualized directly on the job page.

Another more advanced option is using the Sonar code quality visualizer, which is shown in the following screenshot. Sonar tests are run during the build phase and propagated to the Sonar server, where they are stored and visualized.

A Sonar server can be a great way for a development team to see the fruits of their efforts at improving the code base.

The drawback of implementing a Sonar server is that it sometimes slows down the builds. The recommendation is to perform the Sonar builds in your nightly builds, once a day.



Version 1.0 - Wed, 29 Sep 2010 14:12 - profile Sonar Groovy Way

Lines of code
2,192
3,104 lines
112 files

Classes
59
9 packages
154 methods

Comments
7.6%
180 lines

Duplications
0.0%
0 lines
0 blocks
0 files

Complexity
2.8 / method
6.8 / class
7.1 / file
Total: 436

Code coverage
8.5%
10.9% line coverage
3.7% branch coverage
29 tests
2.4 sec

Test success
82.8%
0 failures
5 errors

Rules compliance
94.2%

Violations
49

Blocker	0
Critical	0
Major	39
Minor	10
Info	0

Events All

2010-09-29	Version	1.0	Edit Delete

About build status visualization

The build server produces a lot of data that is amenable to visualization on a shared display. It is useful to be immediately aware that a build has failed, for instance.

The easiest thing is to just hook up a monitor in a kiosk-like configuration with a web browser pointing to your build server web interface. Jenkins has many plugins that provide a simplified job overview suitable for kiosk displays. These are sometimes called **information radiators**.

It is also common to hook up other types of hardware to the build status, such as lava lamps or colorful LED lamps.

In my experience, this kind of display can make people enthusiastic about the build server. Succeeding with having a useful display in the long run is more tricky than it would first appear, though. The screen can be distracting. If you put the screen where it's not easily seen in order to circumvent the distraction, the purpose of the display is defeated.

A lava lamp in combination with a screen placed discreetly could be a useful combination. The lava lamp is not normally lit, and thus not distracting. When a build error occurs, it lights up, and then you know that you should have a look at the build information radiator. The lava lamp even conveys a form of historical record of the build quality. As the lava lamp lights up, it grows warm, and after a while, the lava moves around inside the lamp. When the error is corrected, the lamp cools down, but the heat remains for a while, so the lava will move around for a time proportional to how long it took to fix the build error.

Taking build errors seriously

The build server can signal errors and code quality problems as much as it wants; if developer teams don't care about the problems, then the investment in the notifications and visualization is all for nought.

This isn't something that can be solved by technical means alone. There has to be a process that everybody agrees on, and the easiest way for a consensus to be achieved is for the process to be of obvious benefit to everyone involved.

Part of the problem is organizations where everything is on fire all the time. Is a build error more important than a production error? If code quality measures estimate that it will take years to improve a code base's quality, is it worthwhile to even get started with fixing the issues?

How do we solve these kinds of problems?

Here are some ideas:

- Don't overdo your code quality metrics. Reduce testing until reports show levels that are fixable. You can add tests again after the initial set of problems is taken care of.
- Define a priority for problems. Fix production issues first. Then fix build errors. Do not submit new code on top of broken code until the issues are resolved.

Robustness

While it is desirable that the build server becomes one of the focal points in your Continuous Delivery pipeline, also consider that the process of building and deployment should not come to a standstill in the event of a breakdown of the build server. For this reason, the builds themselves should be as robust as possible and repeatable on any host.

This is fairly easy for some builds, such as Maven builds. Even so, a Maven build can exhibit any number of flaws that makes it non-portable.

A C-based build can be pretty hard to make portable if one is not so fortunate as to have all build dependencies available in the operating system repositories. Still, robustness is usually worth the effort.

Summary

In this chapter, we took a whirlwind tour through the systems that build our code. We had a look at constructing a Continuous Integration server with Jenkins. We also examined a number of problems that might arise, because the life of a DevOps engineer is always interesting but not always easy.

In the next chapter, we will continue our efforts to produce code of the highest quality by studying how we can integrate testing in our workflow.

6
Testing the Code

If we are going to release our code early and often, we ought to be confident of its quality. Therefore, we need automated regression testing.

In this chapter, some frameworks for software testing are explored, such as **JUnit** for unit testing and **Selenium** for web frontend testing. We will also find out how these tests are run in our Continuous Integration server, Jenkins, thus forming the first part of our Continuous Delivery pipeline.

Testing is very important for software quality, and it's a very large subject in itself.

We will concern ourselves with these topics in this chapter:

- How to make manual testing easier and less error prone
- Various types of testing, such as unit testing, and how to perform them in practice
- Automated system integration testing

We already had a look at how to accumulate test data with Sonar and Jenkins in the previous chapter, and we will continue to delve deeper into this subject.

Manual testing

Even if test automation has larger potential benefits for DevOps than manual testing, manual testing will always be an important part of software development. If nothing else, we will need to perform our tests manually at least once in order to automate them.

Acceptance testing in particular is hard to replace, even though there have been attempts to do so. Software requirement specifications can be terse and hard to understand even for the people developing the features that implement those requirements. In these cases, quality assurance people with their eyes on the ball are invaluable and irreplaceable.

The things that make manual testing easier are the same things that make automated integration testing easier as well, so there is a synergy to achieve between the different testing strategies.

In order to have happy quality assurance personnel, you need to:

- Manage test data, primarily the contents of backend databases, so that tests give the same results when you run them repeatedly
- Be able to make rapid deployments of new code in order to verify bug fixes

Obvious as this may seem, it can be hard in practice. Maybe you have large production databases that can't just be copied to test environments. Maybe they contain end-user data that needs to be protected under law. In these cases, you need to de-identify the data and wash it of any personal details before deploying it to test environments.

Each organization is different, so it is not possible to give generally useful advice in this area other than the KISS rule: "Keep it simple, stupid."

Pros and cons with test automation

When you talk with people, most are enthusiastic about the prospect of test automation. Imagine all the benefits that await us with it:

- Higher software quality
- Higher confidence that the software releases we make will work as intended
- Less of the monotonous tedium of laborious manual testing.

All very good and desirable things!

In practice, though, if you spend time with different organizations with complex multi-tiered products, you will notice people talking about test automation, but you will also notice a suspicious absence of test automation in practice. Why is that?

If you just compile programs and deploy them once they pass compilation, you will likely be in for a bad experience. Software testing is completely necessary for a program to work reliably in the real world. Manual testing is too slow to achieve Continuous Delivery. So, we need test automation to succeed with Continuous Delivery. Therefore, let's further investigate the problem areas surrounding test automation and see if we can figure out what to do to improve the situation:

- Cheap tests have lower value.

  One problem is that the type of test automation that is fairly cheap to produce, unit testing, typically has lower perceived value than other types of testing. Unit testing is still a good type of testing, but manual testing might be perceived as exposing more bugs in practice. It might then feel unnecessary to write unit tests.

- It is difficult to create test cradles that are relevant to automated integration testing.

  While it is not very difficult to write test cradles or test fixtures for unit tests, it tends to get harder as the test cradle becomes more production-like. This can be because of a lack of hardware resources, licensing, manpower, and so on.

- The functionality of programs vary over time and tests must be adjusted accordingly, which takes time and effort.

  This makes it seem as though test automation just makes it harder to write software without providing a perceived benefit.

  This is especially true in organizations where developers don't have a close relationship with the people working with operations, that is, a non DevOps oriented organization. If someone else will have to deal with your crappy code that doesn't really work as intended, there is no real cost involved for the developers. This isn't a healthy relationship. This is the central problem DevOps aims to solve. The DevOps approach bears this repeating rule: help people with different roles work closer together. In organizations like Netflix, an Agile team is entirely responsible for the success, maintenance, and outages of their service.

- It is difficult to write robust tests that work reliably in many different build scenarios.

  A consequence of this is that developers tend to disable tests in their local builds so that they can work undisturbed with the feature they have been assigned. Since people don't work with the tests, changes that affect the test outcomes creep in, and eventually, the tests fail.

The build server will pick up the build error, but nobody remembers how the test works now, and it might take several days to fix the test error. While the test is broken, the build displays will show red, and eventually, people will stop caring about build issues. Someone else will fix the problem eventually.

- It is just hard to write good automated tests, period.

  It can indeed be hard to create good automated integration tests. It can also be rewarding, because you get to learn all the aspects of the system you are testing.

  These are all difficult problems, especially since they mostly stem from people's perceptions and relationships.

There is no panacea, but I suggest adopting the following strategy:

- Leverage people's enthusiasm regarding test automation
- Don't set unrealistic goals
- Work incrementally

Unit testing

Unit testing is the sort of testing that is normally close at heart for developers. The primary reason is that, by definition, unit testing tests well-defined parts of the system in isolation from other parts. Thus, they are comparatively easy to write and use.

Many build systems have built-in support for unit tests, which can be leveraged without undue difficulty.

With Maven, for example, there is a convention that describes how to write tests such that the build system can find them, execute them, and finally prepare a report of the outcome. Writing tests basically boils down to writing test methods, which are tagged with source code annotations to mark the methods as being tests. Since they are ordinary methods, they can do anything, but by convention, the tests should be written so that they don't require considerable effort to run. If the test code starts to require complicated setup and runtime dependencies, we are no longer dealing with unit tests.

Here, the difference between unit testing and functional testing can be a source of confusion. Often, the same underlying technologies and libraries are reused between unit and functional testing.

This is a good thing as reuse is good in general and lets you benefit from your expertise in one area as you work on another. Still, it can be confusing at times, and it pays to raise your eyes now and then to see that you are doing the right thing.

JUnit in general and JUnit in particular

You need something that runs your tests. JUnit is a framework that lets you define unit tests in your Java code and run them.

JUnit belongs to a family of testing frameworks collectively called **xUnit**. The **SUnit** is the grandfather of this family and was designed by Kent Beck in 1998 for the Smalltalk language.

While JUnit is specific to Java, the ideas are sufficiently generic for ports to have been made in, for instance, C#. The corresponding test framework for C# is called, somewhat unimaginatively, **NUnit**. The N is derived from .NET, the name of the Microsoft software platform.

We need some of the following nomenclature before carrying on. The nomenclature is not specific to JUnit, but we will use JUnit as an example to make it easier to relate to the definitions.

- **Test runner:** A test runner runs tests that are defined by an xUnit framework.

  JUnit has a way to run unit tests from the command line, and Maven employs a test runner called Surefire. A test runner also collects and reports test results. In the case of Surefire, the reports are in XML format, and these reports can be further processed by other tools, particularly for visualization.

- **Test case:** A test case is the most fundamental type of test definition.

  How you create test cases differs a little bit among JUnit versions. In earlier versions, you inherited from a JUnit base class; in recent versions, you just need to annotate the test methods. This is better since Java doesn't support multiple inheritance and you might want to use your own inheritance hierarchies rather than the JUnit ones. By convention, Surefire also locates test classes that have the `Test` suffix in the class name.

- **Test fixtures:** A test fixture is a known state that the test cases can rely on so that the tests can have well-defined behavior. It is the responsibility of the developer to create these. A test fixture is also sometimes known as a test context.

With JUnit, you usually use the `@Before` and `@After` annotations to define test fixtures. `@Before` is, unsurprisingly, run before a test case and is used to bring up the environment. `@After` likewise restores the state if there is a need to.

Sometimes, `@Before` and `@After` are more descriptively named **Setup** and **Teardown**. Since annotations are used, the method can have the names that are the most intuitive in that context.

- **Test suites:** You can group test cases together in test suites. A test suite is usually a set of test cases that share the same test fixture.
- **Test execution:** A test execution runs the tests suites and test cases.

  Here, all the previous aspects are combined. The test suites and test cases are located, the appropriate test fixtures are created, and the test cases run. Lastly, the test results are collected and collated.

- **Test result formatter:** A test result formatter formats test result output for human consumption. The format employed by JUnit is versatile enough to be used by other testing frameworks and formatters not directly associated with JUnit. So, if you have some tests that don't really use any of the xUnit frameworks, you can still benefit by presenting the test results in Jenkins by providing a test result XML file. Since the file format is XML, you can produce it from your own tool, if need be.
- **Assertions:** An assertion is a construct in the xUnit framework that makes sure that a condition is met. If it is not met, it is considered an error, and a test error is reported. The test case is also usually terminated when the assertion fails.

JUnit has a number of assertion methods available. Here is a sample of the available assertion methods:

- To check whether two objects are equal:

  ```
  assertEquals(str1, str2);
  ```

- To check whether a condition is true:

  ```
  assertTrue (val1 < val2);
  ```

- To check whether a condition is false:

  ```
  assertFalse(val1 > val2);
  ```

A JUnit example

JUnit is well supported by Java build tools. It will serve well as an example of JUnit testing frameworks in general.

If we use Maven, by convention, it will expect to find test cases in the following directory:

```
/src/test/java
```

Mocking

Mocking refers to the practice of writing simulated resources to enable unit testing. Sometimes, the words "fake" or "stub" are used. For example, a middleware system that responds with JSON structures from a database would "mock" the database backend for its unit tests. Otherwise, the unit tests would require the database backend to be online, probably also requiring exclusive access. This wouldn't be convenient.

Mockito is a mocking framework for Java that has also been ported to Python.

Test Coverage

When you hear people talk about unit testing, they often talk about test coverage. Test coverage is the percentage of the application code base that is actually executed by the test cases.

In order to measure unit test code coverage, you need to execute the tests and keep track of the code that has or hasn't been executed.

Cobertura is a test coverage measurement utility for Java that does this. Other such utilities include jcoverage and Clover.

Cobertura works by instrumenting the Java bytecode, inserting code fragments of its own into already compiled code. These code fragments are executed while measuring code coverage during execution of test cases

Its usually assumed that a hundred percent test coverage is the ideal. This might not always be the case, and one should be aware of the cost/benefit trade-offs.

A simple counterexample is a simple getter method in Java:

```
private int positiveValue;
void setPositiveValue(int x){
  this.positiveValue=x;
}

int getPositiveValue(){
  return positiveValue;
}
```

If we write a test case for this method, we will achieve a higher test coverage. On the other hand, we haven't achieved much of anything, in practice. The only thing we are really testing is that our Java implementation doesn't have bugs.

If, on the other hand, the setter is changed to include validation to check that the value is not a negative number, the situation changes. As soon as a method includes logic of some kind, unit testing is useful.

Automated integration testing

Automated integration testing is similar in many ways to unit testing with respect to the basic techniques that are used. You can use the same test runners and build system support. The primary difference with unit testing is that less mocking is involved.

Where a unit test would simply mock the data returned from a backend database, an integration test would use a real database for its tests. A database is a decent example of the kind of testing resources you need and what types of problems they could present.

Automated integration testing can be quite tricky, and you need to be careful with your choices.

If you are testing, say, a read-only middleware adapter, such as a SOAP adapter for a database, it might be possible to use a production database copy for your testing. You need the database contents to be predictable and repeatable; otherwise, it will be hard to write and run your tests.

The added value here is that we are using a production data copy. It might contain data that is hard to predict if you were to create test data from scratch. The requirements are the same as for manual testing. With automated integration testing, you need, well, more automation than with manual testing. For databases, this doesn't have to be very complicated. Automated database backup and restore are well-known operations.

Docker in automated testing

Docker can be quite convenient when building automated test rigs. It adds some of the features of unit testing but at a functional level. If your application consists of several server components in a cluster, you can simulate the entire cluster with a set of containers. Docker provides a virtual network for the cluster that makes clear how the containers interact at the network level.

Docker also makes it easy to restore a container to a known state. If you have your test database in a Docker container, you can easily restore the database to the same state as before the tests taking place. This is similar to restoring the environment in the `After` method in unit tests.

The Jenkins Continuous Integration server has support for the starting and stopping of containers, which can be useful when working with Docker test automation.

Using Docker Compose to run the containers you need is also a useful option.

Docker is still young, and some aspects of using Docker for test automation can require glue code that is less than elegant.

A simple example could be firing up a database container and an application server container that communicate with each other. The basic process of starting the containers is simple and can be done with a shell script or Docker Compose. But, since we want to run tests on the application server container that has been started, how do we know whether it was properly started? In the case of the WildFly container, there is no obvious way to determine the running state apart from watching the log output for occurrences of strings or maybe polling a web socket. In any case, these types of hacks are not very elegant and are time consuming to write. The end result can be well worth the effort, though.

Arquillian

Arquillian is an example of a test tool that allows a level of testing closer to integration testing than unit testing together with mocking allows. Arquillian is specific to Java application servers such as WildFly. Arquillian is interesting because it illustrates the struggle of reaching a closer approximation of production systems during testing. You can reach such approximations in any number of ways, and the road to there is filled with trade-offs.

There is a "hello world" style demonstration of Arquillian in the book's source code archive.

Performance testing

Performance testing is an essential part of the development of, for instance, large public web sites.

Performance testing presents similar challenges as integration testing. We need a testing system that is similar to a production system in order for the performance test data to be useful to make a forecast about real production system performance.

The most commonly used performance test is load testing. With load testing, we measure, among other things, the response time of a server while the performance testing software generates synthetic requests for the server.

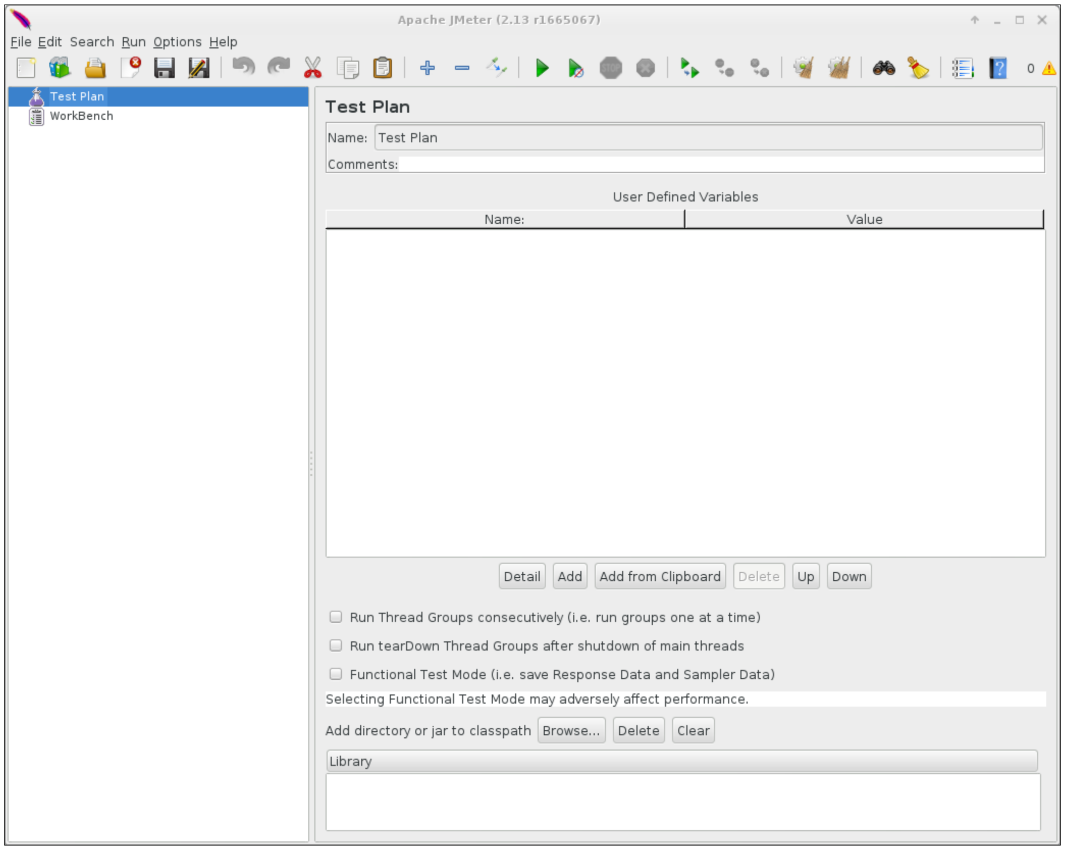
Apache JMeter is an example of a an open source application for measuring performance. While it's simpler than its proprietary counterparts, such as LoadRunner, JMeter is quite useful, and simplicity is not really a bad thing.

JMeter can generate simulated load and measure response times for a number of protocols, such as HT, LDAP, SOAP, and JDBC.

There is a JMeter Maven plugin, so you can run JMeter as part of your build.

JMeter can also be used in a Continuous Integration server. There is a plugin for Jenkins, called the performance plugin, that can execute JMeter test scenarios.

Ideally, the Continuous Integration server will deploy code that has been built to a test environment that is production-like. After deployment, the performance tests will be executed and test data collected, as shown in this screenshot:


Apache JMeter (2.13 r1665067)
File Edit Search Run Options Help
Test Plan
WorkBench
Test Plan
Name: Test Plan
Comments:
User Defined Variables
Name:
Value
Detail
Add
Add from Clipboard
Delete
Up
Down
Run Thread Groups consecutively (i.e. run groups one at a time)
Run tearDown Thread Groups after shutdown of main threads
Functional Test Mode (i.e. save Response Data and Sampler Data)
Selecting Functional Test Mode may adversely affect performance.
Add directory or jar to classpath
Browse...
Delete
Clear
Library

Automated acceptance testing

Automated acceptance testing is a method of ensuring that your testing is valid from the end user's point of view.

Cucumber is a framework where test cases are written in plaintext and associated with test code. This is called **behavior-driven development**. The original implementation of Cucumber was written in Ruby, but ports now exist for many different languages.

The appeal of Cucumber from a DevOps point of view is that it is intended to bring different roles together. Cucumber feature definitions are written in a conversational style that can be achieved without programming skills. The hard data required for test runs is then extracted from the descriptions and used for the tests.

While the intentions are good, there are difficulties in implementing Cucumber that might not immediately be apparent. While the language of the behavior specifications is basically free text, they still need to be somewhat spartan and formalized; otherwise, it becomes difficult to write matching code that extracts the test data from the descriptions. This makes writing the specifications less attractive to the roles that were supposed to write them in the first place. What then happens is that programmers write the specifications, and they often dislike the verbosity and resort to writing ordinary unit tests.

As with many things, cooperation is of the essence here. Cucumber can work great when developers and product owners work together on writing the specifications in a way that works for everyone concerned.

Now, let's look at a small "hello world" style example for Cucumber.

Cucumber works with plaintext files called feature files, which look like this:

```
Feature: Addition
  I would like to add numbers with my pocket calculator

  Scenario: Integer numbers
    * I have entered 4 into the calculator
    * I press add
    * I have entered 2 into the calculator
    * I press equal
    * The result should be 6 on the screen
```

The feature description is implementation-language neutral. Describing the Cucumber test code is done in a vocabulary called **Gherkin**.

If you use the Java 8 lambda version of Cucumber, a test step could look somewhat like this:

```
Calculator calc;
public MyStepdefs() {
  Given("I have entered (\\d+) into the calculator", (Integer i) -> {
    System.out.format("Number entered: %n\n", i);
    calc.push(i);
  });
```

```
    When("I press (\\w+)", (String op) -> {
      System.out.format("operator entered: %n\n", op);
      calc.op(op);
    });
    Then("The result should be (\\d+)", (Integer i) -> {
      System.out.format("result : %n\n", i);
      assertThat(calc.result(),i);
    });
```

The complete code can, as usual, be found in the book's source archive.

This is a simple example, but it should immediately be apparent both what the strengths and weaknesses with Cucumber are. The feature descriptions have a nice human-readable flair. However, you have to match the strings with regular expressions in your test code. If your feature description changes even slightly in wording, you will have to adjust the test code.

Automated GUI testing

Automating GUI testing has many desirable properties, but it is also difficult. One reason is that user interfaces tend to change a lot during the development phase, and buttons and controls move around in the GUI.

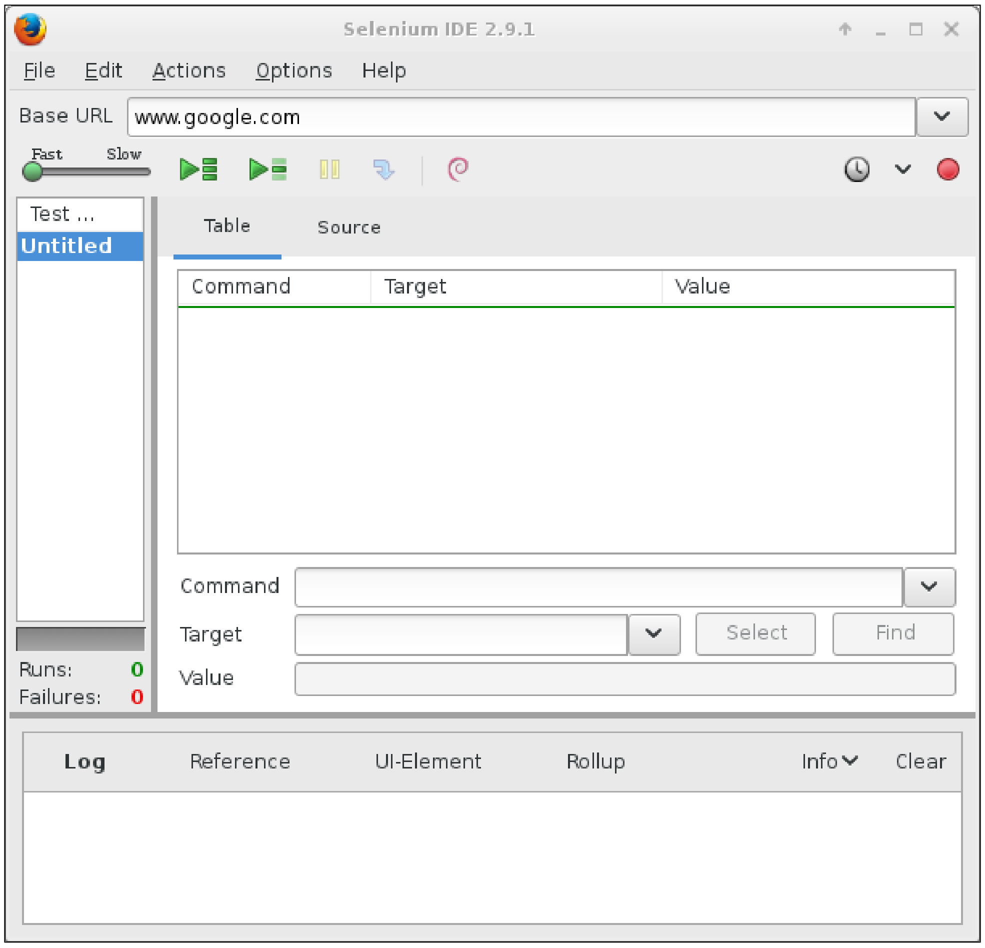
Older generations of GUI testing tools often worked by synthesizing mouse events and sending them to the GUI. When a button moved, the simulated mouse click went nowhere, and the test failed. It then became expensive to keep the tests updated with changes in the GUI.

Selenium is a web UI testing toolkit that uses a different, more effective, approach. The controllers are instrumented with identifiers so that Selenium can find the controllers by examining the document object model (DOM) rather than blindly generating mouse clicks.

Selenium works pretty well in practice and has evolved over the years.

Another method is employed by the Sikuli test framework. It uses a computer vision framework, OpenCV, to help identify controllers even if they move or change appearances. This is useful for testing native applications, such as games.

The screenshot included below is from the Selenium IDE.



Integrating Selenium tests in Jenkins

Selenium works by invoking a browser, pointing it to a web server running your application, and then remotely controlling the browser by integrating itself in the JavaScript and DOM layers.

When you develop the tests, you can use two basic methods:

- Record user interactions in the browser and later save the resulting test code for reuse
- Write the tests from scratch using Selenium's test API

Many developers prefer to write tests as code using the Selenium API at the outset, which can be combined with a test-driven development approach.

Regardless of how the tests are developed, they need to run in the integration build server.

This means that you need browsers installed somewhere in your test environment. This can be a bit problematic since build servers are usually headless, that is, they are servers that don't run user interfaces.

It's possible to wrap a browser in a simulated desktop environment on the build server.

A more advanced solution is using Selenium Grid. As the name implies, Selenium Grid provides a server that gives a number of browser instances that can be used by the tests. This makes it possible to run a number of tests in parallel as well as to provide a set of different browser configurations.

You can start out with the single browser solution and later migrate to the Selenium Grid solution when you need it.

There is also a convenient Docker container that implements Selenium Grid.

JavaScript testing

Since there usually are web UI implementations of nearly every product these days, the JavaScript testing frameworks deserve special mention:

- Karma is a test runner for unit tests in the JavaScript language
- Jasmine is a Cucumber-like behavior testing framework
- Protractor is used for AngularJS

Protractor is a different testing framework, similar to Selenium in scope but optimized for AngularJS, a popular JavaScript user interface framework. While it would appear that new web development frameworks come and go everyday, it's interesting to note why a test framework like Protractor exists when Selenium is available and is general enough to test AngularJS applications too.

First of all, Protractor actually uses the Selenium web driver implementation under the hood.

You can write Protractor tests in JavaScript, but you can use JavaScript for writing test cases for Selenium as well if you don't like writing them in a language like Java.

The main benefit turns out to be that Protractor has internalized knowledge about the Angular framework, which a general framework like Selenium can't really have.

AngularJS has a model/view setup that is particular to it. Other frameworks use other setups, since the model/view setup isn't something that is intrinsic to the JavaScript language—not yet, anyway.

Protractor knows about the peculiarities of Angular, so it's easier to locate controllers in the testing code with special constructs.

Testing backend integration points

Automated testing of backend functionality such as SOAP and REST endpoints is normally quite cost effective. Backend interfaces tend to be fairly stable, so the corresponding tests will also require less maintenance effort than GUI tests, for instance.

The tests can also be fairly easy to write with tools such as soapUI, which can be used to write and execute tests. These tests can also be run from the command line and with Maven, which is great for Continuous Integration on a build server.

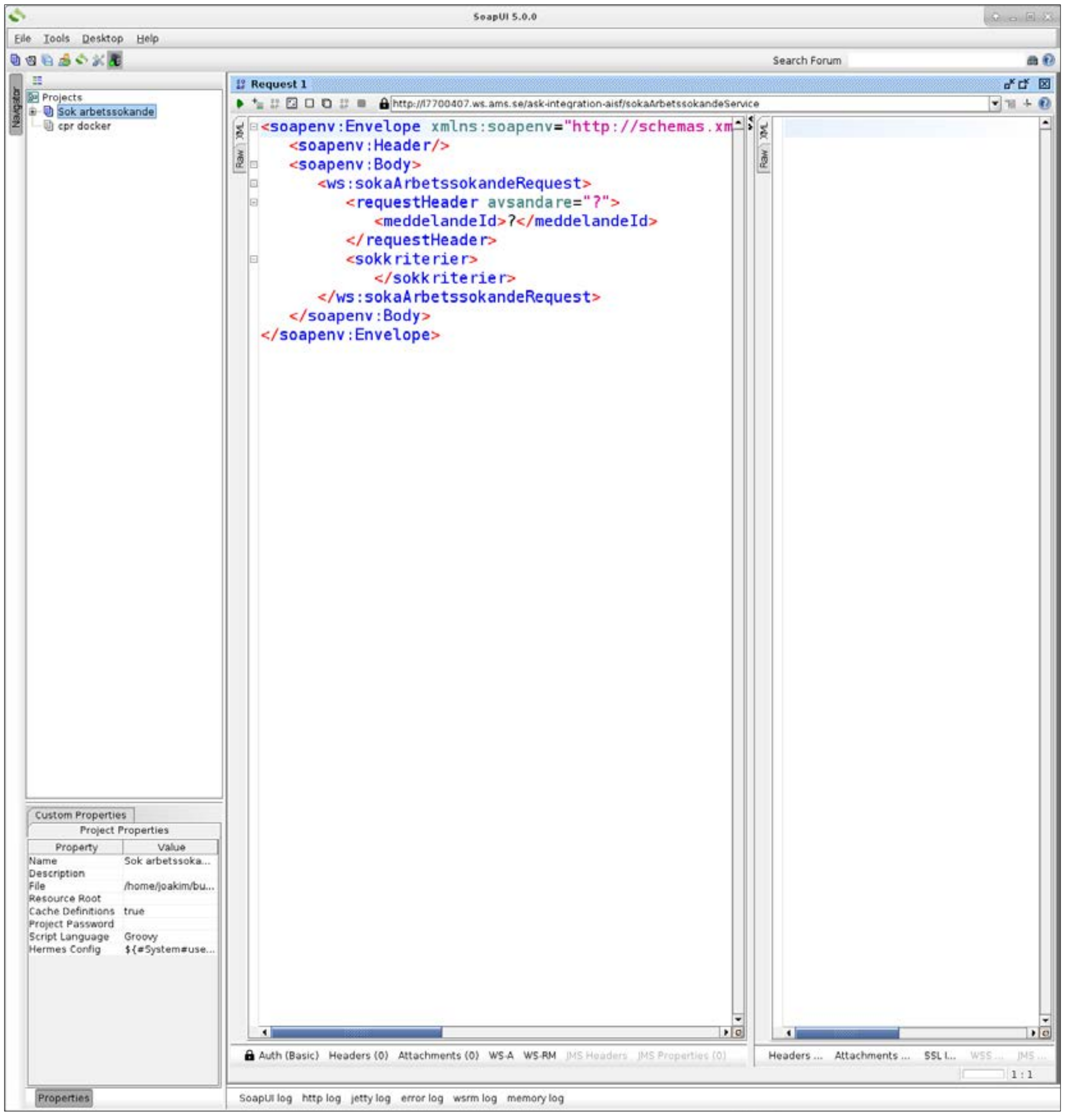
The soapUI is a good example of a tool that appeals to several different roles. Testers who build test cases get a fairly well-structured environment for writing tests and running them interactively. Tests can be built incrementally.

Developers can integrate test cases in their builds without necessarily using the GUI. There are Maven plugins and command-line runners.

The command line and Maven integration are useful for people maintaining the build server too.

Furthermore, the licensing is open source with some added features in a separate, proprietary version. The open source nature makes the builds more reliable. It is very stress-inducing when a build fails because a license has unexpectedly reached its end or a floating license has run out.

The soapUI tool has its share of flaws, but in general, it is flexible and works well. Here's what the user interface looks like:


SoapUI 5.0.0
File Tools Desktop Help
Search Forum
Projects
Sok arbetssokande
cpr docker
Request 1
http://l7700407.ws.ams.se/ask-integration-aisf/sokaArbetssokandeService
Auth (Basic) Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Properties (0)
Headers ... Attachments ... SSL I... WSS ... JMS ...
SoapUI log http log jetty log error log wsrm log memory log
Properties

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xm
   <soapenv:Header/>
   <soapenv:Body>
      <ws:sokaArbetssokandeRequest>
         <requestHeader avsandare="?">
            <meddelandeId>?</meddelandeId>
         </requestHeader>
         <sokkriterier>
            </sokkriterier>
      </ws:sokaArbetssokandeRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Custom Properties

Project Properties

Property	Value
Name	Sok arbetssoka...
Description	
File	/home/joakim/bu...
Resource Root	
Cache Definitions	true
Project Password	
Script Language	Groovy
Hermes Config	${#System#use...

The soapUI user interface is fairly straightforward. There is a tree view listing test cases on the left. It is possible to select single tests or entire test suites and run them. The results are presented in the area on the right.

It is also worth noting that the test cases are defined in XML. This makes it possible to manage them as code in the source code repository. This also makes it possible to edit them in a text editor on occasion, for instance, when we need to perform a global search and replace on an identifier that has changed names—just the way we like it in DevOps!

Test-driven development

Test-driven development (TDD) has an added focus on test automation. It was made popular by the Extreme programming movement of the nineties.

TDD is usually described as a sequence of events, as follows:

- **Implement the test**: As the name implies, you start out by writing the test and write the code afterwards. One way to see it is that you implement the interface specifications of the code to be developed and then progress by writing the code. To be able to write the test, the developer must find all relevant requirement specifications, use cases, and user stories.

  The shift in focus from coding to understanding the requirements can be beneficial for implementing them correctly.

- **Verify that the new test fails:** The newly added test should fail because there is nothing to implement the behavior properly yet, only the stubs and interfaces needed to write the test. Run the test and verify that it fails.
- **Write code that implements the tested feature:** The code we write doesn't yet have to be particularly elegant or efficient. Initially, we just want to make the new test pass.
- **Verify that the new test passes together with the old tests:** When the new test passes, we know that we have implemented the new feature correctly. Since the old tests also pass, we haven't broken existing functionality.
- **Refactor the code:** The word "refactor" has mathematical roots. In programming, it means cleaning up the code and, among other things, making it easier to understand and maintain. We need to refactor since we cheated a bit earlier in the development.

TDD is a style of development that fits well with DevOps, but it's not necessarily the only one. The primary benefit is that you get good test suites that can be used in Continuous Integration tests.

REPL-driven development

While REPL-driven development isn't a widely recognized term, it is my favored style of development and has a particular bearing on testing. This style of development is very common when working with interpreted languages, such as Lisp, Python, Ruby, and JavaScript.

When you work with a Read Eval Print Loop (REPL), you write small functions that are independent and also not dependent on a global state.

The functions are tested even as you write them.

This style of development differs a bit from TDD. The focus is on writing small functions with no or very few side effects. This makes the code easy to comprehend rather than when writing test cases before functioning code is written, as in TDD.

You can combine this style of development with unit testing. Since you can use REPL-driven development to develop your tests as well, this combination is a very effective strategy.

A complete test automation scenario

We have looked at a number of different ways of working with test automation. Assembling the pieces into a cohesive whole can be daunting.

In this section, we will have a look at a complete test automation example, continuing from the user database web application for our organization, Matangle.

You can find the source code in the accompanying source code bundle for the book.

The application consists of the following layers:

- A web frontend
- A JSON/REST service interface
- An application backend layer
- A database layer

The test code will work through the following phases during execution:

- Unit testing of the backend code
- Functional testing of the web frontend, performed with the Selenium web testing framework
- Functional testing of the JSON/REST interface, executed with soapUI

All the tests are run in sequence, and when all of them succeed, the result can be used as the basis for a decision to see whether the application stack is deemed healthy enough to deploy to a test environment, where manual testing can commence.

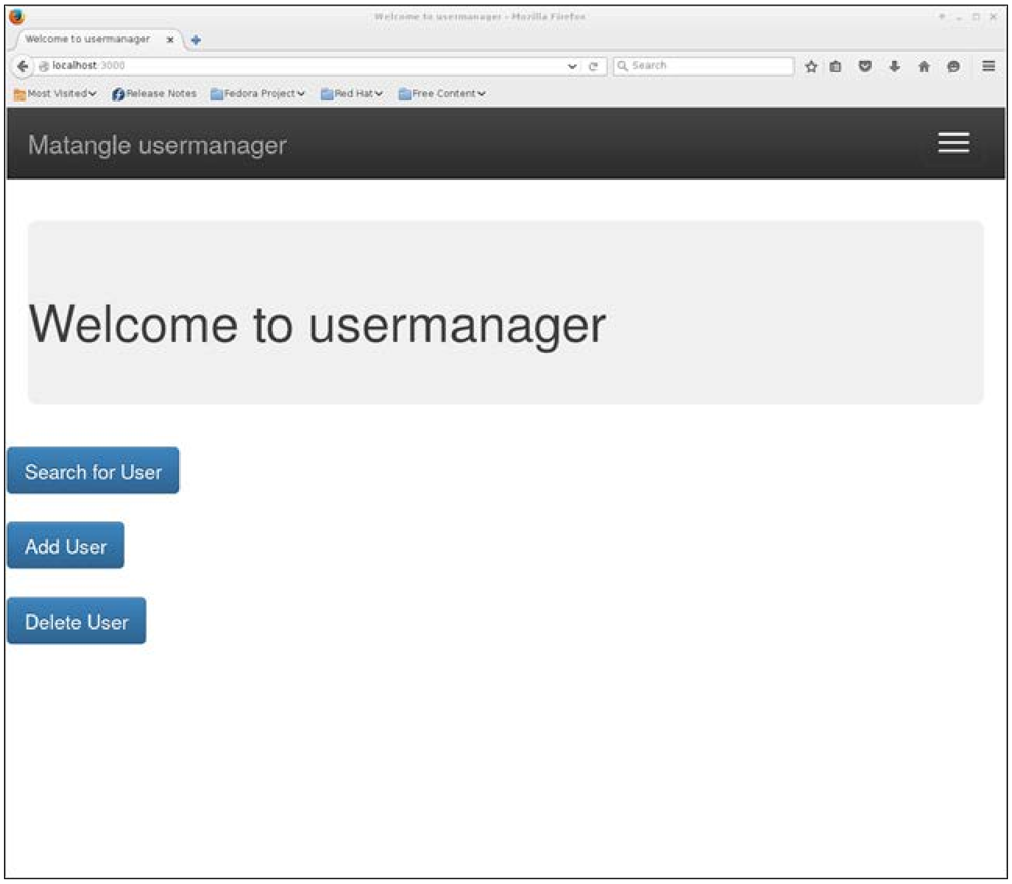
Manually testing our web application

Before we can automate something usefully, we need to understand the details of what we will be automating. We need some form of a test plan.

Below, we have a test plan for our web application. It details the steps that a human tester needs to perform by hand if no test automation is available. It is similar to what a real test plan would look like, except a real plan would normally have many more formalities surrounding it. In our case, we will go directly to the details of the test in question:

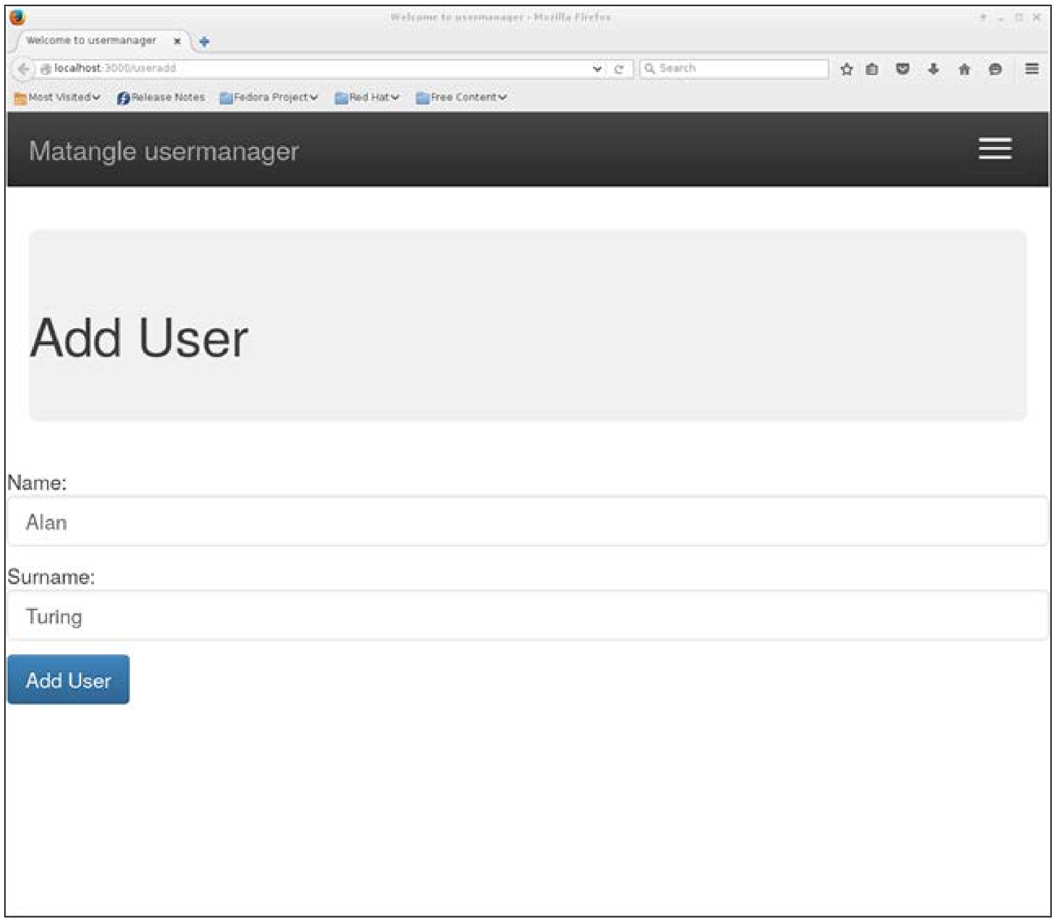
1. Start a fresh test. This resets the database backend to a known state and sets up the testing scenario so that manual testing can proceed from a known state.

   The tester points a browser to the application's starting URL:



Welcome to usermanager

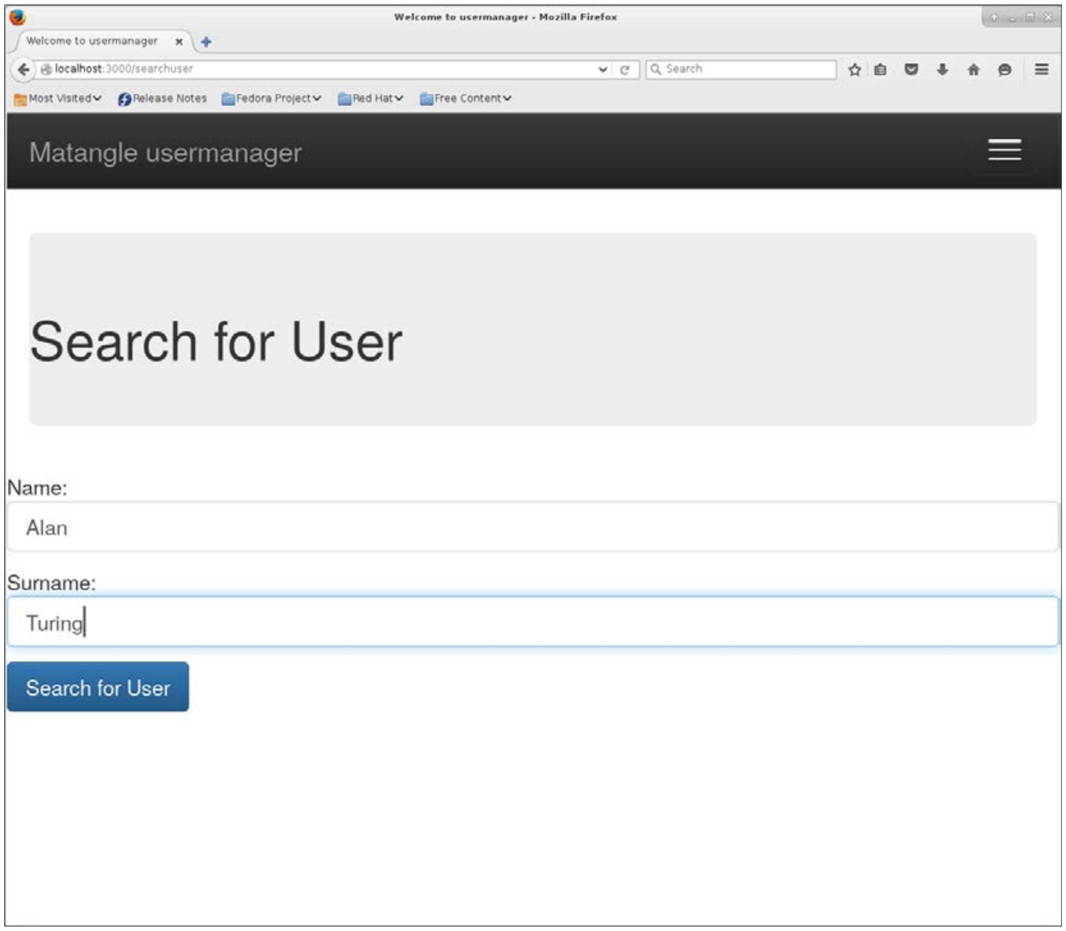
2. Click on the **Add User** link.
3. Add a user:



Enter a username—`Alan Turing`, in our test case.

4. Save the new user. A success page will be shown.

5. Verify that the user was added properly by performing a search:



Click on the **Search User** link. Search for **Alan Turing**. Verify that **Alan** is present in the result list.

While the reader is probably less than impressed with the application's complexity at this point, this is the level of detail we need to work with if we are going to be able to automate the scenario, and it is this complexity that we are studying here.

Running the automated test

The test is available in a number of flavors in the source bundle.

To run the first one, you need a Firefox installation.

Choose the one called `autotest_v1`, and run it from the command line:

```
autotest_v1/bin/autotest.sh
```

If all goes well, you will see a Firefox window open, and the test you previously performed by hand will be done automatically. The values you filled in and the links you clicked on by hand will all be automated.

This isn't foolproof yet, because maybe the Firefox version you installed isn't compatible, or something else is amiss with the dependencies. The natural reaction to problems like these is dependency management, and we will look at a variant of dependency management using Docker shortly.

Finding a bug

Now we will introduce a bug and let the test automation system find it.

As an exercise, find the string "Turing" in the test sources. Change one of the occurrences to "Tring" or some other typographical error. Just change one; otherwise, the verification code will believe there is no error and that everything is alright!

Run the tests again, and notice that the error is found by the automatic test system.

Test walkthrough

Now we have run the tests and verified that they work. We have also verified that they are able to discover the bug we created.

What does the implementation look like? There is a lot of code, and it would not be useful to reprint it in the book. It is useful, though, to give an overview of the code and have a look at some snippets of the code.

Open the `autotest_v1/test/pom.xml` file. It's a Maven project object model file, and it's here that all the plugins used by the tests are set up. Maven POM files are declarative XML files and the test steps are step-by-step imperative instructions, so in the latter case, Java is used.

There's a property block at the top, where dependency versions are kept. There is no real need to break out the versions; it has been used in this case to make the rest of the POM file less version-dependent:

```
<properties>
  <junit.version>XXX</junit.version>
  <selenium.version>XXX</selenium.version>
  <cucumber.version>XXX</cucumber.version>
  ...
</properties>
```

Here are the dependencies for JUnit, Selenium and Cucumber:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
</dependency>

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>${selenium.version}</version>
</dependency>

<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-core</artifactId>
    <version>${cucumber.version}</version>
</dependency>

<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>${cucumber.version}</version>
</dependency>

<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>${cucumber.version}</version>
</dependency>
```

To define the tests according to the Cucumber method, we need a feature file that describes the test steps in a human-readable language. This feature file corresponds to our previous test plan for manual tests:

```
Feature: Manage users
  As an Administrator
  I want to be able to
  - Create a user
  - Search for the user
  - Delete the user

  Scenario: Create a named user
     Given a user with the name 'Alan'
     And the surname 'Turing'
     When the administrator clicks 'Add User'
     Then the user should be added

  Scenario: Search for a named user
     Given a user with the name 'Alan'
     And the surname 'Turing'
     When the administrator clicks 'Search for User'
     Then the user 'Alan Turing' should be shown

  Scenario: Delete a named user
     Given a user with the name 'Alan'
     And the surname 'Turing'
     When the administrator clicks 'Delete User'
     Then the user should be deleted
```

The feature file is mostly plaintext with small elements of machine-readable markup. It's up to the corresponding test code to parse the plaintext of the scenarios with regexes.

It is also possible to localize the feature files to the language used within your own team. This can be useful since the feature files might be written by people who are not accustomed to English.

The feature needs actual concrete code to execute, so you need some way to bind the feature to the code.

You need a test class with some annotations to make Cucumber and JUnit work together:

```
@RunWith(Cucumber.class)
@Cucumber.Options(
        glue = "matangle.glue.manageUser",
        features = "features/manageUser.feature",
        format = {"pretty", "html:target/Cucumber"}
)
```

In this example, the names of the Cucumber test classes have, by convention, a Step suffix.

Now, you need to bind the test methods to the feature scenarios and need some way to pick out the arguments to the test methods from the feature description. With the Java Cucumber implementation, this is mostly done with Java annotations. These annotations correspond to the keywords used in the feature file:

```
@Given(".+a user with the name '(.+)'")
    public void addUser(String name) {
```

In this case, the different inputs are stored in member variables until the entire user interface transaction is ready to go. The sequence of operations is determined by the order in which they appear in the feature file.

To illustrate that Cucumber can have different implementations, there is also a Clojure example in the book's source code bundle.

So far, we have seen that we need a couple of libraries for Selenium and Cucumber to run the tests and how the Cucumber feature descriptor is bound to methods in our test code classes.

The next step is to examine how Cucumber tests execute Selenium test code.

Cucumber test steps mostly call classes with Selenium implementation details in classes with a View suffix. This isn't a technical necessity, but it makes the test step classes more readable, since the particulars of the Selenium framework are kept in a separate class.

The Selenium framework takes care of the communication between the test code and the browser. View classes are an abstraction of the web page that we are automating. There are member variables in the view code that correspond to HTML controllers. You can describe the binding between test code member variables and HTML elements with annotations from the Selenium framework, as follows:

```
@FindBy(id = "name") private WebElement nameInput;
@FindBy(id = "surname") private WebElement surnameInput;
```

The member variable is then used by the test code to automate the same steps that the human tester followed using the test plan. The partitioning of classes into view and step classes also makes the similarity of the step classes to a test plan more apparent. This separation of concerns is useful when people involved with testing and quality assurance work with the code.

To send a string, you use a method to simulate a user typing on a keyboard:

```
nameInput.clear();
nameInput.sendKeys(value);
```

There are a number of useful methods, such as `click()`, which will simulate a user clicking on the control.

Handling tricky dependencies with Docker

Because we used Maven in our test code example, it handled all code dependencies except the browser. While you could clearly deploy a browser such as Firefox to a Maven-compatible repository and handle the test dependency that way if you put your mind to it, this is normally not the way this issue is handled in the case of browsers. Browsers are finicky creatures and show wildly differing behavior in different versions. We need a mechanism to run many different browsers of many different versions.

Luckily, there is such a mechanism, called Selenium Grid. Since Selenium has a pluggable driver architecture, you can essentially layer the browser backend in a client server architecture.

To use Selenium Grid, you must first determine how you want the server part to run. While the easiest option would be to use an online provider, for didactic reasons, it is not the option we will explore here.

There is an `autotest_seleniumgrid` directory, which contains a wrapper to run the test using Docker in order to start a local Selenium Grid. You can try out the example by running the wrapper.

The latest information regarding how to run Selenium Grid is available on the project's GitHub page.

Selenium Grid has a layered architecture, and to set it up, you need three parts:

- A `RemoteWebDriver` instance in your testing code. This will be the interface to Selenium Grid.
- Selenium Hub, which can be seen as a proxy for browser instances.

- Firefox or Chrome grid nodes. These are the browser instances that will be proxied by Hub.

The code to set up the `RemoteWebDriver` could look like this:

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setPlatform(Platform.LINUX);
capabilities.setBrowserName("Firefox");
        capabilities.setVersion("35");
    driver = new RemoteWebDriver(
                new URL("http://localhost:4444"),
                capabilities);
```

The code asks for a browser instance with a particular set of capabilities. The system will do its best to oblige.

The code can only work if there is a Selenium Grid Hub running with a Firefox node attached.

Here is how you start Selenium Hub using its Docker packaging:

```
docker run -d -p 4444:4444 --name selenium-hub selenium/hub
```

And here is how you can start a Firefox node and attach it to Hub:

```
docker run -d --link selenium-hub:hub selenium/node-firefox
```

Summary

This concludes the test code walkthrough. When you read the code, you might want to use only a subset of the ideas that are illustrated. Maybe the Cucumber method isn't really a good fit for you, or you value concise and succinct code over the layered abstractions used in the example. That is a natural and sound reaction. Adopt the ideas so they work for your team. Also, have a look at the other flavors of the testing code available in the source bundle when deciding on what works for you!

Software testing is a vast subject that can fill volumes. In this chapter, we surveyed some of the many different types of testing available. We also looked at concrete ways of working with automated software testing in a Continuous Integration server. We used Jenkins and Maven as well as JUnit and JMeter. Although these tools are Java oriented, the concepts translate readily to other environments.

Now that we have built and tested our code, we will start working with deploying our code in the next chapter.

7 Deploying the Code

Now that the code has been built and tested, we need to deploy it to our servers so that our customers can use the newly developed features!

There are many competing tools and options in this space, and the one that is right for you and your organization will depend on your needs.

We will explore Puppet, Ansible, Salt, PalletOps, and others, by showing how to deploy sample applications in different scenarios. Any one of these tools has a vast ecosystem of complementing services and tools, so it is no easy subject to get a grip on.

Throughout the book, we have encountered aspects of some of the different deployment systems that already exist. We had a look at RPMs and `.deb` files and how to build them with the `fpm` command. We had a look at various Java artifacts and how Maven uses the idea of a binary repository where you can deploy your versioned artifacts.

In this chapter, we will focus on installing binary packages and their configuration with a configuration management system.

Why are there so many deployment systems?

There is a bewildering abundance of options regarding the installation of packages and configuring them on actual servers, not to mention all the ways to deploy client-side code.

Let's first examine the basics of the problem we are trying to solve.

We have a typical enterprise application, with a number of different high-level components. We don't need to make the scenario overly complex in order to start reasoning about the challenges that exist in this space.

In our scenario, we have:

- A web server
- An application server
- A database server

If we only have a single physical server and these few components to worry about that get released once a year or so, we can install the software manually and be done with the task. It will be the most cost-effective way of dealing with the situation, even though manual work is boring and error prone.

It's not reasonable to expect a conformity to this simplified release cycle in reality though. It is more likely that a large organization has hundreds of servers and applications and that they are all deployed differently, with different requirements.

Managing all the complexity that the real world displays is hard, so it starts to make sense that there are a lot of different solutions that do basically the same thing in different ways.

Whatever the fundamental unit that executes our code is, be it a physical server, a virtual machine, some form of container technology, or a combination of these, we have several challenges to deal with. We will look at them now.

Configuring the base OS

The configuration of the base operating system must be dealt with somehow.

Often, our application stack has subtle, or not so subtle, requirements on the base operating system. Some application stacks, such as Java, Python, or Ruby, make these operating system requirements less apparent, because these technologies go to a great length to offer cross-platform functionality. At other times, the operating system requirements are apparent to a greater degree, such as when you work with low-level mixed hardware and software integrations, which is common in the telecom industry.

There are many existing solutions that deal with this fundamental issue. Some systems work with a bare metal (or bare virtual machine) approach, where they install the desired operating system from scratch and then install all the base dependencies that the organization needs for their servers. Such systems include, for example, Red Hat Satellite and Cobbler, which works in a similar way but is more lightweight.

Cobbler allows you to boot a physical or virtual machine over the network using `dhcpd`. The DHCP server can then allow you to provide a netboot-compliant image. When the netboot image is started, it contacts Cobbler to retrieve the packages that will be installed in order to create the new operating system. Which packages are installed can be decided on the server from the target machine's network MAC address for instance.

Another method that is very popular today is to provide base operating system images that can be reused between machines. Cloud systems such as AWS, Azure, or OpenStack work this way. When you ask the cloud system for a new virtual machine, it is created using an existing image as a base. Container systems such as Docker also work in a similar way, where you declare your base container image and then describe the changes you want to formulate for your own image.

Describing clusters

There must be a way to describe clusters.

If your organization only has a single machine with a single application, then you might not need to describe how a cluster deployment of your application would look like. Unfortunately (or fortunately, depending on your outlook), the reality is normally that your applications are spread out over a set of machines, virtual or physical.

All the systems we work with in this chapter support this idea in different ways. Puppet has an extensive system that allows machines to have different roles that in turn imply a set of packages and configurations. Ansible and Salt have these systems as well. The container-based Docker system has an emerging infrastructure for describing sets of containers connected together and Docker hosts that can accept and deploy such cluster descriptors.

Cloud systems such as AWS also have methods and descriptors for cluster deployments.

Cluster descriptors are normally also used to describe the application layer.

Delivering packages to a system

There must be a way to deliver packages to a system.

Much of an application can be installed as packages, which are installed unmodified on the target system by the configuration management system. Package systems such as RPM and `deb` have useful features, such as verifying that the files provided by a package are not tampered with on a target system, by providing checksums for all files in the package. This is useful for security reasons as well as debugging purposes. Package delivery is usually done with operating system facilities such as `yum` package channels on Red Hat based systems, but sometimes, the configuration management system can also deliver packages and files with its own facilities. These facilities are often used in tandem with the operating system's package channels.

There must be a way to manage configurations that is independent of installed packages.

The configuration management system should, obviously, be able to manage our applications' configurations. This is complex because configuration methods vary wildly between applications, regardless of the many efforts that have been made to unify the field.

The most common and flexible system to configure applications relies on text-based configuration files. There are several other methods, such as using an application that provides an API to handle configuration (such as a command-line interface) or sometimes handling configuration via database settings.

In my experience, configuration systems based on text files create the least amount of hassle and should be preferred for in-house code at least. There are many ways to manage text-based configurations. You can manage them in source code handling systems such as Git. There's a host of tools that can ease the debugging of broken configuration, such as `diff`. If you are in a tight spot, you can edit configurations directly on the servers using a remote text editor such as Emacs or Vi.

Handling configurations via databases is much less flexible. This is arguably an anti-pattern that usually occurs in organizations where the psychological rift between developer teams and operations teams are too wide, which is something we aim to solve with DevOps. Handling configurations in databases makes the application stack harder to get running. You need a working database to even start the application.

Managing configuration settings via imperative command-line APIs is also a dubious practice for similar reasons but can sometimes be helpful, especially if the API is used to manage an underlying text-based configuration. Many of the configuration management systems, such as Puppet, depend on being able to manage declarative configurations. If we manage the configuration state via other mechanisms, such as command-line imperative API, Puppet loses many of its benefits.

Even managing text-based configuration files can be a hassle. There are many ways for applications to invent their own configuration file formats, but there are a set of base file formats that are popular. Such file formats include XML, YML, JSON, and INI.

Usually, configuration files are not static, because if they were, you could just deploy them with your package system like any piece of binary artifact.

Normally, the application configuration files need to be based on some kind of template file that is later instantiated into a form suitable for the machine where the application is being deployed.

An example might be an application's database connector descriptor. If you are deploying your application to a test environment, you want the connector descriptor to point to a test database server. Vice versa, if you are deploying to a production server, you want your connector to point to a production database server.

As an aside, some organizations try to handle this situation by managing their DNS servers, such that an example database DNS alias `database.yourorg.com` resolves to different servers depending on the environment. The domain `yourorg.com` should be replaced with your organization's details of course, and the database server as well.

Being able to use different DNS resolvers depending on the environment is a useful strategy. It can be difficult for a developer, however, to use the equivalent mechanism on his or her own development machine. Running a private DNS server on a development machine can be difficult, and managing a local host file can also prove cumbersome. In these cases, it might be simpler to make the application have configurable settings for database hosts and other backend systems at the application level.

Many times, it is possible to ignore the details of the actual configuration file format altogether and just rely on the configuration system's template managing system. These usually work by having a special syntax for placeholders that will be replaced by the configuration management system when creating the concrete configuration file for a concrete server, where the application will be deployed. You can use the exact same basic idea for all text-based configuration files, and sometimes even for binary files, even though such hacks should be avoided if at all possible.

The XML format has tools and infrastructure that can be useful in managing configurations, and XML is indeed a popular format for configuration files. For instance, there is a special language, XSLT, to transform XML from one structural form to another. This is very helpful in some cases but used less in practice than one might expect. The simple template macro substitution approach gets you surprisingly far and has the added benefit of being applicable on nearly all text-based configuration formats. XML is also fairly verbose, which also makes it unpopular in some circles. YML can be seen as a reaction to XML's verbosity and can accomplish much of the same things as XML, with less typing.

Another useful feature of some text configuration systems that deserves mention is the idea that a base configuration file can include other configuration files. An example of this is the standard Unix `sudo` tool, which has its base configuration in the `/etc/sudoers` file, but which allows for local customization by including all the files that have been installed in the directory `/etc/sudoers.d`.

This is very convenient, because you can provide a new `sudoer` file without worrying too much about the existing configuration. This allows for a greater degree of modularization, and it is a convenient pattern when the application allows it.

Virtualization stacks

Organizations that have their own internal server farms tend to use virtualization a lot in order to encapsulate the different components of their applications.

There are many different solutions depending on your requirements.

Virtualization solutions provide virtual machines that have virtual hardware, such as network cards and CPUs. Virtualization and container techniques are sometimes confused because they share some similarities.

You can use virtualization techniques to simulate entirely different hardware than the one you have physically. This is commonly referred to as emulation. If you want to emulate mobile phone hardware on your developer machine so that you can test your mobile application, you use virtualization in order to emulate a device. The closer the underlying hardware is to the target platform, the greater the efficiency the emulator can have during emulation. As an example, you can use the QEMU emulator to emulate an Android device. If you emulate an Android x86_64 device on an x86_64-based developer machine, the emulation will be much more efficient than if you emulate an ARM-based Android device on an x86_64-based developer machine.

With server virtualization, you are usually not really interested in the possibility of emulation. You are interested instead in encapsulating your application's server components. For instance, if a server application component starts to run amok and consume unreasonable amounts of CPU time or other resources, you don't want the entire physical machine to stop working altogether.

This can be achieved by creating a virtual machine with, perhaps, two cores on a machine with 64 cores. Only two cores would be affected by the runaway application. The same goes for memory allocation.

Container-based techniques provide similar degrees of encapsulation and control over resource allocation as virtualization techniques do. Containers do not normally provide the emulation features of virtualization, though. This is not an issue since we rarely need emulation for server applications.

The component that abstracts the underlying hardware and arbitrates hardware resources between different competing virtual machines is called a **hypervisor**. The hypervisor can run directly on the hardware, in which case it is called a bare metal hypervisor. Otherwise, it runs inside an operating system with the help of the operating system kernel.

VMware is a proprietary virtualization solution, and exists in desktop and server hypervisor variants. It is well supported and used in many organizations. The server variant changes names sometimes; currently, it's called VMware ESX, which is a bare metal hypervisor.

KVM is a virtualization solution for Linux. It runs inside a Linux host operating system. Since it is an open source solution, it is usually much cheaper than proprietary solutions since there are no licensing costs per instance and is therefore popular with organizations that have massive amounts of virtualization.

Xen is another type of virtualization which, amongst other features, has **paravirtualization**. Paravirtualization is built upon the idea that that if the guest operating system can be made to use a modified kernel, it can execute with greater efficiency. In this way, it sits somewhere between full CPU emulation, where a fully independent kernel version is used, and container-based virtualization, where the host kernel is used.

VirtualBox is an open source virtualization solution from Oracle. It is pretty popular with developers and sometimes used with server installations as well but rarely on a larger scale. Developers who use Microsoft Windows on their developer machines but want to emulate Linux server environments locally often find VirtualBox handy. Likewise, developers who use Linux on their workstations find it useful to emulate Windows machines.

What the different types of virtualization technologies have in common is that they provide APIs in order to allow the automation of virtual machine management. The `libvirt` API is one such API that can be used with several different underlying hypervisors, such as KVM, QEMU, Xen, and LXC

Executing code on the client

Several of the configuration management systems described here allow you to reuse the node descriptors to execute code on matching nodes. This is sometimes convenient. For example, maybe you want to run a directory listing command on all HTTP servers facing the public Internet, perhaps for debugging purposes.

In the Puppet ecosystem, this command execution system is called Marionette Collective, or MCollective for short.

A note about the exercises

It is pretty easy to try out the various deployment systems using Docker to manage the base operating system, where we will do our experiments. It is a time-saving method that can be used when developing and debugging the deployment code specific to a particular deployment system. This code will then be used for deployments on physical or virtual machines.

We will first try each of the different deployment systems that are usually possible in the local deployment modes. Further down the line, we will see how we can simulate the complete deployment of a system with several containers that together form a virtual cluster.

We will try to use the official Docker images if possible, but sometimes there are none, and sometimes the official image vanishes, as happened with the official Ansible image. Such is life in the fast-moving world of DevOps, for better or for worse.

It should be noted, however, that Docker has some limitations when it comes to emulating a full operating system. Sometimes, a container must run in elevated privilege modes. We will deal with those issues when they arise.

It should also be noted that many people prefer Vagrant for these types of tests. I prefer to use Docker when possible, because it's lightweight, fast, and sufficient most of the time.

[Keep in mind that actually deploying systems in production will require more attention to security and other details than we provide here.]

The Puppet master and Puppet agents

Puppet is a deployment solution that is very popular in larger organizations and is one of the first systems of its kind.

Puppet consists of a client/server solution, where the client nodes check in regularly with the Puppet server to see if anything needs to be updated in the local configuration.

The Puppet server is called a **Puppet master**, and there is a lot of similar wordplay in the names chosen for the various Puppet components.

Puppet provides a lot of flexibility in handling the complexity of a server farm, and as such, the tool itself is pretty complex.

This is an example scenario of a dialogue between a Puppet client and a Puppet master:

1. The Puppet client decides that it's time to check in with the Puppet master to discover any new configuration changes. This can be due to a timer or manual intervention by an operator at the client. The dialogue between the Puppet client and master is normally encrypted using SSL.
2. The Puppet client presents its credentials so that the Puppet master can know exactly which client is calling. Managing the client credentials is a separate issue.
3. The Puppet master figures out which configuration the client should have by compiling the Puppet catalogue and sending it to the client. This involves a number of mechanisms, and a particular setup doesn't need to utilize all possibilities.

   It is pretty common to have both a role-based and concrete configuration for a Puppet client. Role-based configurations can be inherited.

4. The Puppet master runs the necessary code on the client side such that the configuration matches the one decided on by the Puppet master.

In this sense, a Puppet configuration is declarative. You declare what configuration a machine should have, and Puppet figures out how to get from the current to the desired client state.

There are both pros and cons of the Puppet ecosystem:

- Puppet has a large community, and there are a lot of resources on the Internet for Puppet. There are a lot of different modules, and if you don't have a really strange component to deploy, there already is, with all likelihood, an existing module written for your component that you can modify according to your needs.
- Puppet requires a number of dependencies on the Puppet client machines. Sometimes, this gives rise to problems. The Puppet agent will require a Ruby runtime that sometimes needs to be ahead of the Ruby version available in your distribution's repositories. Enterprise distributions often lag behind in versions.
- Puppet configurations can be complex to write and test.

Ansible

Ansible is a deployment solution that favors simplicity.

The Ansible architecture is agentless; it doesn't need a running daemon on the client side like Puppet does. Instead, the Ansible server logs in to the Ansible node and issues commands over SSH in order to install the required configuration.

While Ansible's agentless architecture does make things simpler, you need a Python interpreter installed on the Ansible nodes. Ansible is somewhat more lenient about the Python version required for its code to run than Puppet is for its Ruby code to run, so this dependence on Python being available is not a great hassle in practice.

Like Puppet and others, Ansible focuses on configuration descriptors that are idempotent. This basically means that the descriptors are declarative and the Ansible system figures out how to bring the server to the desired state. You can rerun the configuration run, and it will be safe, which is not necessarily the case for an imperative system.

Let's try out Ansible with the Docker method we discussed earlier.

We will use the `williamyeh/ansible` image, which has been developed for the purpose, but it should be possible to use any Ansible Docker image or different ones altogether, to which we just add Ansible later.

1. Create a Dockerfile with this statement:
   ```
   FROM williamyeh/ansible:centos7
   ```
2. Build the Docker container with the following command:
   ```
   docker build .
   ```

This will download the image and create an empty Docker container that we can use.

Normally, you would, of course, have a more complex Dockerfile that can add the things we need, but in this case, we are going to use the image interactively, so we will instead mount the directory with Ansible files from the host so that we can change them on the host and rerun them easily.

3. Run the container.

   The following command can be used to run the container. You will need the hash from the previous `build` command:

   ```
   docker run -v `pwd`/ansible:/ansible  -it <hash> bash
   ```

   Now we have a prompt, and we have Ansible available. The `-v` trick is to make parts of the host filesystem visible to the Docker guest container. The files will be visible in the `/ansible` directory in the container.

The `playbook.yml` file is as follows:

```
---
- hosts: localhost
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
```

This playbook doesn't do very much, but it demonstrates some concepts of Ansible playbooks.

Now, we can try to run our Ansible playbook:

```
cd /ansible
ansible-playbook -i inventory playbook.yml    --connection=local --sudo
```

The output will look like this:

```
PLAY [localhost] ***********************************************************
******


GATHERING FACTS ************************************************************
******
ok: [localhost]
```

```
TASK: [ensure apache is at the latest version] **************************
******
ok: [localhost]


PLAY RECAP **************************************************************
******
localhost                  : ok=2    changed=0    unreachable=0
failed=0
```

Tasks are run to ensure the state we want. In this case, we want to install Apache's `httpd` using `yum`, and we want `httpd` to be the latest version.

To proceed further with our exploration, we might like to do more things, such as starting services automatically. However, here we run into a limitation with the approach of using Docker to emulate physical or virtual hosts. Docker is a container technology, after all, and not a full-blown virtualization system. In Docker's normal use case scenarios, this doesn't matter, but in our case, we need make some workarounds in order to proceed. The main problem is that the `systemd init system` requires special care to run in a container. Developers at Red Hat have worked out methods of doing this. The following is a slightly modified version of a Docker image by Vaclav Pavlin, who works with Red Hat:

```
FROM fedora
RUN yum -y update; yum clean all
RUN yum install  ansible sudo
RUN systemctl mask systemd-remount-fs.service dev-hugepages.mount sys-
fs-fuse-connections.mount systemd-logind.service getty.target console-
getty.service
RUN cp /usr/lib/systemd/system/dbus.service /etc/systemd/system/; sed
-i 's/OOMScoreAdjust=-900//' /etc/systemd/system/dbus.service

VOLUME ["/sys/fs/cgroup", "/run", "/tmp"]
ENV container=docker

CMD ["/usr/sbin/init"]
```

The environment variable `container` is used to tell the `systemd` init system that it runs inside a container and to behave accordingly.

We need some more arguments for `docker run` in order to enable `systemd` to work in the container:

```
docker run -it --rm -v /sys/fs/cgroup:/sys/fs/cgroup:ro  -v `pwd`/
ansible:/ansible <hash>
```

The container boots with `systemd`, and now we need to connect to the running container from a different shell:

```
docker exec -it <hash> bash
```

Phew! That was quite a lot of work just to get the container more lifelike! On the other hand, working with virtual machines, such as VirtualBox, is even more cumbersome in my opinion. The reader might, of course, decide differently.

Now, we can run a slightly more advanced Ansible playbook inside the container, as follows:

```
---
- hosts: localhost
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

This example builds on the previous one, and shows you how to:

- Install a package
- Write a template file
- Handle the running state of a service

The format is in a pretty simple YML syntax.

PalletOps

PalletOps is an advanced deployment system, which combines the declarative power of Lisp with a very lightweight server configuration.

PalletOps takes Ansible's agentless idea one step further. Rather than needing a Ruby or Python interpreter installed on the node that is to be configured, you only need `ssh` and a `bash` installation. These are pretty simple requirements.

PalletOps compiles its Lisp-defined DSL to Bash code that is executed on the slave node. These are such simple requirements that you can use it on very small and simple servers—even phones!

On the other hand, while there are a number of support modules for Pallet called **crates**, there are fewer of them than there are for Puppet or Ansible.

Deploying with Chef

Chef is a Ruby-based deployment system from Opscode.

It is pretty easy to try out Chef; for fun, we can do it in a Docker container so we don't pollute our host environment with our experiments:

```
docker run -it ubuntu
```

We need the `curl` command to proceed with downloading the chef installer:

```
apt-get -y install curl
curl -L https://www.opscode.com/chef/install.sh | bash
```

The Chef installer is built with a tool from the Chef team called **omnibus**. Our aim here is to try out a Chef tool called `chef-solo`. Verify that the tool is installed:

```
chef-solo -v
```

This will give output as:

```
Chef: 12.5.1
```

The point of `chef-solo` is to be able to run configuration scripts without the full infrastructure of the configuration system, such as the client/server setup. This type of testing environment is often useful when working with configuration systems, since it can be hard to get all the bits and pieces in working order while developing the configuration that you are going to deploy.

Chef prefers a file structure for its files, and a pre-rolled structure can be retrieved from GitHub. You can download and extract it with the following commands:

```
curl -L  http://github.com/opscode/chef-repo/tarball/master -o master.tgz
tar -zxf master.tgz
mv chef-chef-repo* chef-repo
rm master.tgz
```

You will now have a suitable file structure prepared for Chef cookbooks, which looks like the following:

```
./cookbooks
./cookbooks/README.md
./data_bags
./data_bags/README.md
./environments
./environments/README.md
./README.md
./LICENSE
./roles
./roles/README.md
./chefignore
```

You will need to perform a further step to make everything work properly, telling `chef` where to find its cookbooks as:

```
mkdir .chef
echo "cookbook_path [ '/root/chef-repo/cookbooks' ]" > .chef/knife.rb
```

Now we can use the `knife` tool to create a template for a configuration, as follows:

```
knife cookbook create phpapp
```

Deploying with SaltStack

SaltStack is a Python-based deployment solution.

There is a convenient dockerized test environment for Salt, by Jackson Cage. You can start it with the following:

```
docker run -i -t --name=saltdocker_master_1 -h master -p 4505 -p 4506 \
   -p 8080 -p 8081 -e SALT_NAME=master -e SALT_USE=master \
   -v `pwd`/srv/salt:/srv/salt:rw jacksoncage/salt
```

This will create a single container with both a Salt master and a Salt minion.

We can create a shell inside the container for our further explorations:

```
docker exec -i -t saltdocker_master_1 bash
```

We need a configuration to apply to our server. Salt calls configurations "states", or Salt states.

In our case, we want to install an Apache server with this simple Salt state:

```
top.sls:
base:
  '*':
    - webserver

webserver.sls:
apache2:               # ID declaration
  pkg:                # state declaration
    - installed       # function declaration
```

Salt uses `.yml` files for its configuration files, similar to what Ansible does.

The file `top.sls` declares that all matching nodes should be of the type `webserver`. The `webserver` state declares that an `apache2` package should be installed, and that's basically it. Please note that this will be distribution dependent. The Salt Docker test image we are using is based on Ubuntu, where the Apache web server package is called `apache2`. On Fedora for instance, the Apache web server package is instead simply called `httpd`.

Run the command once to see Salt in action, by making Salt read the Salt state and apply it locally:

```
salt-call --local state.highstate -l debug
```

The first run will be very verbose, especially since we enabled the debug flag!

Now, let's run the command again:

```
salt-call --local state.highstate -l debug
```

This will also be pretty verbose, and the output will end with this:

```
local:
----------
          ID: apache2
    Function: pkg.installed
      Result: True
     Comment: Package apache2 is already installed.
     Started: 22:55:36.937634
    Duration: 2267.167 ms
```

```
     Changes:


Summary
------------
Succeeded: 1
Failed:    0
------------
Total states run:     1
```

Now, you can quit the container and restart it. This will clean the container from the Apache instance installed during the previous run.

This time, we will apply the same state but use the message queue method rather than applying the state locally:

```
salt-call state.highstate
```

This is the same command as used previously, except we omitted the `–local flag`. You could also try running the command again and verify that the state remains the same.

Salt versus Ansible versus Puppet versus PalletOps execution models

While the configuration systems we explore in this chapter share a fair number of similarities, they differ a lot in the way code is executed on the client nodes:

- With Puppet, a Puppet agent registers with the Puppet master and opens a communication channel to retrieve commands. This process is repeated periodically, normally every thirty minutes.

  Thirty minutes isn't fast. You can, of course, configure a lower value for the time interval required for the next run. At any rate, Puppet essentially uses a pull model. Clients must check in to know whether changes are available.

- Ansible pushes changes over SSH when desired. This is a push model.
- Salt uses a push model, but with a different implementation. It employs a ZeroMQ messaging server that the clients connect to and listen for notifications about changes. This works a bit like Puppet, but faster.

Which method is best is an area of contention between developer communities. Proponents of the message queue architecture believe that it is faster and that speed matters. Proponents of the plain SSH method claim that it is fast enough and that simplicity matters. I lean toward the latter stance. Things tend to break, and the likelihood of breakage increases with complexity.

Vagrant

Vagrant is a configuration system for virtual machines. It is geared towards creating virtual machines for developers, but it can be used for other purposes as well.

Vagrant supports several virtualization providers, and VirtualBox is a popular provider for developers.

First, some preparation. Install `vagrant` according to the instructions for your distribution. For Fedora, the command is this:

```
yum install 'vagrant*'
```

This will install a number of packages. However, as we are installing this on Fedora, we will experience some problems. The Fedora Vagrant packages use `libvirt` as a virtual machine provider rather than VirtualBox. That is useful in many cases, but in this case, we would like to use VirtualBox as a provider, which requires some extra steps on Fedora. If you use some other distribution, the case might be different.

First, add the VirtualBox repository to your Fedora installation. Then we can install VirtualBox with the `dnf` command, as follows:

```
dnf install VirtualBox
```

VirtualBox is not quite ready to be used yet, though. It needs special kernel modules to work, since it needs to arbitrate access to low-level resources. The VirtualBox kernel driver is not distributed with the Linux kernel. Managing Linux kernel drivers outside of the Linux source tree has always been somewhat inconvenient compared to the ease of using kernel drivers that are always installed by default. The VirtualBox kernel driver can be installed as a source module that needs to be compiled. This process can be automated to a degree with the `dkms` command, which will recompile the driver as needed when there is a new kernel installed. The other method, which is easier and less error-prone, is to use a kernel module compiled for your kernel by your distribution. If your distribution provides a kernel module, it should be loaded automatically. Otherwise, you could try `modprobe vboxdrv`. For some distributions, you can compile the driver by calling an `init.d` script as follows:

```
sudo /etc/init.d/vboxdrv setup
```

Now that the Vagrant dependencies are installed, we can bring up a Vagrant virtual machine.

The following command will create a Vagrant configuration file from a template. We will be able to change this file later. The base image will be `hashicorp/precise32`, which in turn is based on Ubuntu.

```
vagrant init hashicorp/precise32
```

Now, we can start the machine:

```
vagrant up
```

If all went well, we should have a `vagrant` virtual machine instance running now, but since it is headless, we won't see anything.

Vagrant shares some similarities with Docker. Docker uses base images that can be extended. Vagrant also allows this. In the Vagrant vocabulary, a base image is called a **box**.

To connect to the headless `vagrant` instance we started previously, we can use this command:

```
vagrant ssh
```

Now we have an `ssh` session, where we can work with the virtual machine. For this to work, Vagrant has taken care of a couple of tasks, such as setting up keys for the SSH communication channel for us.

Vagrant also provides a configuration system so that Vagrant machine descriptors can be used to recreate a virtual machine that is completely configured from source code.

Here is the Vagrantfile we got from the earlier stage. Comments are removed for brevity.

```
Vagrant.configure(2) do |config|
  config.vm.box = "hashicorp/precise32"
end
```

Add a line to the Vagrantfile that will call the bash script that we will provide:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise32"
  config.vm.provision :shell, path: "bootstrap.sh"
end
```

The `bootstrap.sh` script will look like this:

```
#!/usr/bin/env bash
apt-get update
apt-get install -y apache2
```

This will install an Apache web server in the Vagrant-managed virtual machine.

Now we know enough about Vagrant to be able to reason about it from a DevOps perspective:

- Vagrant is a convenient way of managing configurations primarily for virtual machines based on VirtualBox. It's great for testing.
- The configuration method doesn't really scale up to clusters, and it's not the intended use case, either.
- On the other hand, several configuration systems such as Ansible support Vagrant, so Vagrant can be very useful while testing our configuration code.

Deploying with Docker

A recent alternative for deployment is Docker, which has several very interesting traits. We have already used Docker several times in this book.

You can make use of Docker's features for test automation purposes even if you use, for instance, Puppet or Ansible to deploy your products.

Docker's model of creating reusable containers that can be used on development machines, testing environments, and production environments is very appealing.

At the time of writing, Docker is beginning to have an impact on larger enterprises, but solutions such as Puppet are dominant.

While it is well known how to build large Puppet or Ansible server farms, it's not yet equally well known how to build large Docker-based server clusters.

There are several emerging solutions, such as these:

- **Docker Swarm:** Docker Swarm is compatible with Docker Compose, which is appealing. Docker Swarm is maintained by the Docker community.
- **Kubernetes**: Kubernetes is modeled after Google's Borg cluster software, which is appealing since it's a well-tested model used in-house in Google's vast data centers. Kubernetes is not the same as Borg though, which must be kept in mind. It's not clear whether Kubernetes offers scaling the same way Borg does.

Comparison tables

Everyone likes coming up with new words for old concepts. While the different concepts in various products don't always match, it's tempting to make a dictionary that maps the configuration systems' different terminology with each other.

Here is such a terminology comparison chart:

System	**Puppet**	**Ansible**	**Pallet**	**Salt**
Client	Agent	Node	Node	Minion
Server	Master	Server	Server	Master
Configuration	Catalog	Playbook	Crate	Salt State

Also, here is a technology comparison chart:

System	**Puppet**	**Ansible**	**Pallet**	**Chef**	**Salt**
Agentless	No	Yes	Yes	Yes	Both
Client dependencies	Ruby	Python, sshd, bash	sshd, bash	Ruby, sshd, bash	Python
Language	Ruby	Python	Clojure	Ruby	Python

Cloud solutions

First, we must take a step back and have a look at the landscape. We can either use a cloud provider, such as AWS or Azure, or we can use our own internal cloud solution, such as VMware or OpenStack. There are valid arguments for both external and internal cloud providers or even both, depending on your organization.

Some types of organizations, such as government agencies, must store all data regarding citizens within their own walls. Such organizations can't use external cloud providers and services and must instead build their own internal cloud equivalents.

Smaller private organizations might benefit from using an external cloud provider but can't perhaps afford having all their resources with such a provider. They might opt to have in-house servers for normal loads and scale out to an external cloud provider during peak loads.

Many of the configuration systems we have described here support the management of cloud nodes as well as local nodes. PalletOps supports AWS and Puppet supports Azure, for instance. Ansible supports a host of different cloud services.

AWS

Amazon Web Services allows us to deploy virtual machine images on Amazon's clusters. You can also deploy Docker images. Follow these steps to set up AWS:

1. Sign up for an account with AWS. Registration is free of charge, but a credit card number is required even for the free of charge tier
2. Some identity verification will need to happen, which can be done via an automated challenge-response phone call.
3. When the user verification process is complete, you will be able to log in to AWS and use the web console.

In my opinion, the AWS web console does not represent the epitome of web interface usability, but it gets the job done. There are a host of options, and in our case, we are interested in the virtual machine and Docker container options.

4. Go to **EC2 network and security**. Here you can create management keys that will be required later.

   As a first example, let's create the default container example provided by AWS, console-sample-app-static. To log in to the generated server, you need first to create an SSH key pair and upload your public key to AWS. Click through all the steps and you will get a small sample cluster. The final resource creation step can be slow, so it's the perfect opportunity to grab a cup of coffee!

5. Now, we can view the details of the cluster and choose the web server container. You can see the IP address. Try opening it in a web browser.

Now that we have a working account on AWS, we can manage it with the configuration management system of our choosing.

Azure

Azure is a cloud platform from Microsoft. It can host both Linux and Microsoft virtual machines. While AWS is the service people often default to, at least in the Linux space, it never hurts to explore the options. Azure is one such option that is gaining market share at the moment.

Creating a virtual machine on Azure for evaluation purposes is comparable to creating a virtual machine on AWS. The process is fairly smooth.

Summary

In this chapter, we explored some of the many options available to us when deploying the code we built. There are a lot many options, and that is for a reason. Deployment is a difficult subject, and you will likely spend a lot of time figuring out which option suits you best.

In the next chapter, we will explore the topic of monitoring our running code.

8 Monitoring the Code

In the previous chapter, we explored methods of deploying our code.

Now that the code has been safely deployed to your servers with the deployment solution of your choice, you need to watch over it to make sure it's running properly. You can spend a lot of time preparing for the many modes of failure that you may envision during development. In the end, your software will probably break for other reasons altogether than those you prepared for. If your system breaks, it can be very expensive for your organization, either in lost revenue or in terms of lost credibility, which, in the end, might amount to the same thing. You need to know as quickly as possible what has gone wrong in order to deal with the situation.

Given the potential negative impact of service downtime, there are many alternative solutions that approach the problem domain of watching over your deployed code from different angles and viewpoints.

In this chapter, we will have a look at several of the options that are available to us.

Nagios

In this section, we will explore the Nagios monitoring solution for overall server health.

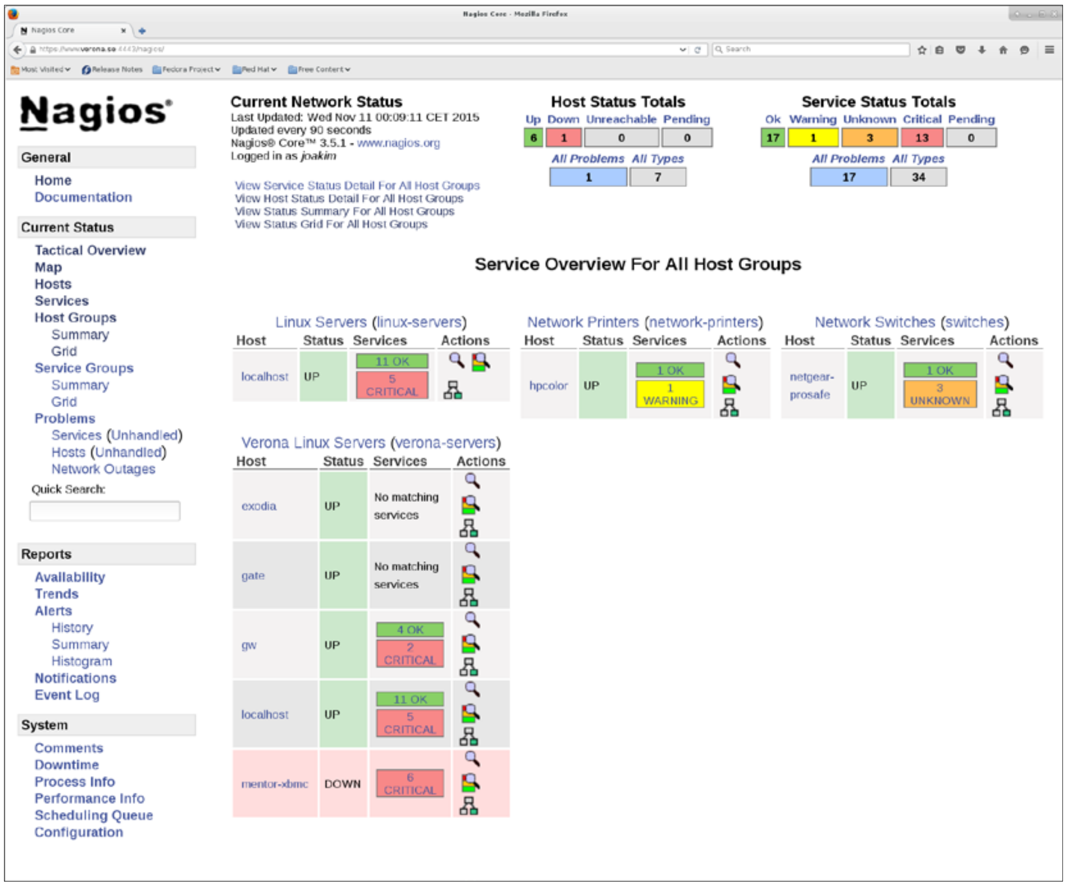
Nagios has been around since 1999 and has a large community that provides plugins and extensions for various needs. There are extensive resources available for Nagios on the web, and commercial support if your organization requires it.

Since it has been around so long and many organizations use it, Nagios is the standard in network monitoring against which other solutions compare themselves. As such, it is the natural starting point for our journey through the monitoring landscape.

The name Nagios is a recursive acronym, which is a tradition in hacker circles. It stands for "Nagios Ain't Gonna Insist On Sainthood". Originally, the product was called NetSaint, but the name was rejected in favor of the Nagios acronym during a trademark dispute. The word *agios* also means *angel* in Greek, so it is a pretty clever acronym overall.

Here are some screenshots from a small demo Nagios installation. Nagios provides many views, and here we see two of them:

- **Service Overview For All Host Groups**:



Nagios

General
Home
Documentation

Current Status
Tactical Overview
Map
Hosts
Services
Host Groups
Summary
Grid
Service Groups
Summary
Grid
Problems
Services (Unhandled)
Hosts (Unhandled)
Network Outages
Quick Search:

Reports
Availability
Trends
Alerts
History
Summary
Histogram
Notifications
Event Log

System
Comments
Downtime
Process Info
Performance Info
Scheduling Queue
Configuration

Current Network Status
Last Updated: Wed Nov 11 00:09:11 CET 2015
Updated every 90 seconds
Nagios® Core™ 3.5.1 - www.nagios.org
Logged in as *joakim*

View Service Status Detail For All Host Groups
View Host Status Detail For All Host Groups
View Status Summary For All Host Groups
View Status Grid For All Host Groups

Host Status Totals

Up	Down	Unreachable	Pending
6	1	0	0

All Problems	*All Types*
1	7

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
17	1	3	13	0

All Problems	*All Types*
17	34

Service Overview For All Host Groups

Linux Servers (linux-servers)

Host	Status	Services	Actions
localhost	UP	11 OK 5 CRITICAL	

Network Printers (network-printers)

Host	Status	Services	Actions
hpcolor	UP	1 OK 1 WARNING	

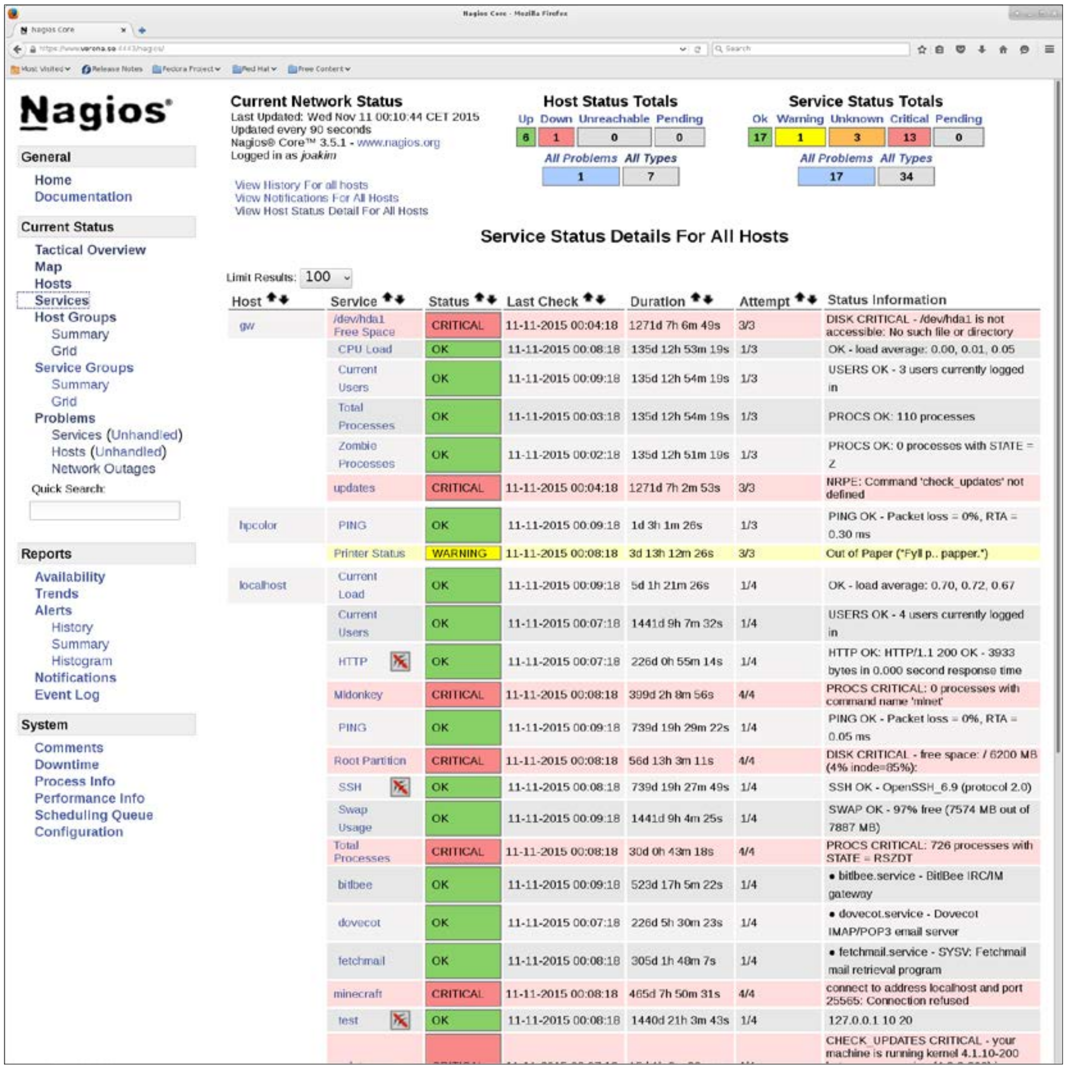
Network Switches (switches)

Host	Status	Services	Actions
netgear-prosafe	UP	1 OK 3 UNKNOWN	

Verona Linux Servers (verona-servers)

Host	Status	Services	Actions
exodia	UP	No matching services	
gate	UP	No matching services	
gw	UP	4 OK 2 CRITICAL	
localhost	UP	11 OK 5 CRITICAL	
mentor-xbmc	DOWN	6 CRITICAL	

- **Service Status Details For All Hosts**:



Current Network Status
Last Updated: Wed Nov 11 00:10:44 CET 2015
Updated every 90 seconds
Nagios® Core™ 3.5.1 - www.nagios.org
Logged in as *joakim*

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
6	1	0	0

All Problems	All Types
1	7

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
17	1	3	13	0

All Problems	All Types
17	34

Service Status Details For All Hosts

Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
gw	/dev/hda1 Free Space	CRITICAL	11-11-2015 00:04:18	1271d 7h 6m 49s	3/3	DISK CRITICAL - /dev/hda1 is not accessible: No such file or directory
	CPU Load	OK	11-11-2015 00:08:18	135d 12h 53m 19s	1/3	OK - load average: 0.00, 0.01, 0.05
	Current Users	OK	11-11-2015 00:09:18	135d 12h 54m 19s	1/3	USERS OK - 3 users currently logged in
	Total Processes	OK	11-11-2015 00:03:18	135d 12h 54m 19s	1/3	PROCS OK: 110 processes
	Zombie Processes	OK	11-11-2015 00:02:18	135d 12h 51m 19s	1/3	PROCS OK: 0 processes with STATE = Z
	updates	CRITICAL	11-11-2015 00:04:18	1271d 7h 2m 53s	3/3	NRPE: Command 'check_updates' not defined
hpcolor	PING	OK	11-11-2015 00:09:18	1d 3h 1m 26s	1/3	PING OK - Packet loss = 0%, RTA = 0.30 ms
	Printer Status	WARNING	11-11-2015 00:08:18	3d 13h 12m 26s	3/3	Out of Paper ("Fyll p.. papper.")
localhost	Current Load	OK	11-11-2015 00:09:18	5d 1h 21m 26s	1/4	OK - load average: 0.70, 0.72, 0.67
	Current Users	OK	11-11-2015 00:07:18	1441d 9h 7m 32s	1/4	USERS OK - 4 users currently logged in
	HTTP	OK	11-11-2015 00:07:18	226d 0h 55m 14s	1/4	HTTP OK: HTTP/1.1 200 OK - 3933 bytes in 0.000 second response time
	Mldonkey	CRITICAL	11-11-2015 00:08:18	399d 2h 8m 56s	4/4	PROCS CRITICAL: 0 processes with command name 'mlnet'
	PING	OK	11-11-2015 00:09:18	739d 19h 29m 22s	1/4	PING OK - Packet loss = 0%, RTA = 0.05 ms
	Root Partition	CRITICAL	11-11-2015 00:08:18	56d 13h 3m 11s	4/4	DISK CRITICAL - free space: / 6200 MB (4% inode=85%):
	SSH	OK	11-11-2015 00:08:18	739d 19h 27m 49s	1/4	SSH OK - OpenSSH_6.9 (protocol 2.0)
	Swap Usage	OK	11-11-2015 00:09:18	1441d 9h 4m 25s	1/4	SWAP OK - 97% free (7574 MB out of 7887 MB)
	Total Processes	CRITICAL	11-11-2015 00:08:18	30d 0h 43m 18s	4/4	PROCS CRITICAL: 726 processes with STATE = RSZDT
	bitlbee	OK	11-11-2015 00:09:18	523d 17h 5m 22s	1/4	● bitlbee.service - BitlBee IRC/IM gateway
	dovecot	OK	11-11-2015 00:07:18	226d 5h 30m 23s	1/4	● dovecot.service - Dovecot IMAP/POP3 email server
	fetchmail	OK	11-11-2015 00:08:18	305d 1h 48m 7s	1/4	● fetchmail.service - SYSV: Fetchmail mail retrieval program
	minecraft	CRITICAL	11-11-2015 00:08:18	465d 7h 50m 31s	4/4	connect to address localhost and port 25565: Connection refused
	test	OK	11-11-2015 00:08:18	1440d 21h 3m 43s	1/4	127.0.0.1 10 20
						CHECK_UPDATES CRITICAL - your machine is running kernel 4.1.10-200

The views indicate some example problems, as follows:

- A host is unreachable
- A disk in one of the hosts is getting full

A host has pending updates available. Let's now examine the basics of Nagios in particular and monitoring in general with a simple scenario:

- We have a Nagios host that monitors the well-being of a number of other physical and virtual hosts.
- One of the hosts being monitored, a web server, suddenly vanishes from the network altogether. Nagios knows this, since it is set up to ping the web server every five minutes and take action in the event that the server does not respond to the pings in a timely fashion.
- Nagios determines what to do when this server is down, which in this case is to send an e-mail to everyone in a predetermined group (in this case, the administrators of the organization).
- Joakim, who happens to be around the Matangle office, is the first to respond to the e-mail, and goes down to the quite unprofessional server room in the cellar. The web server has stopped working because its power supply unit is full of dust, which accumulated there while renovations were being done in a nearby room—sawing plaster generates a lot of dust. Joakim curses and vows that he would move the servers to better hosting, if he only had the resources.

You likely take better care of your servers than Joakim did in this scenario, but the point is that unforeseen things happen, and Nagios was helpful in resolving the situation. Also, while employing a proper data center for your servers might prevent things such as dust in your equipment and people tripping over network cables, there are still any number of weird and unforeseen modes of failure present in the unforgiving world of complex server deployments.

Nagios can monitor services as well as hosts and can utilize different methods to do so. The simplest methods work on the Nagios server and query a service on a remote host.

Nagios differentiates between two basic classes of checks:

- **Active checks**: These are initiated by Nagios. Nagios can execute code in plugins regularly. The outcome of the code execution determines whether the check fails or not. Active checks are the most common and are useful and easy to set up with many types of services, such as HTTP, SSH, and databases.
- **Passive checks**: These originate from a system other than Nagios, and the system notifies Nagios rather than Nagios polling it. These are useful in special situations, for instance, if the service being monitored is behind a firewall or if the service is asynchronous and it would be inefficient to monitor it with polling.

Configuring Nagios is relatively straightforward if you are used to file-based configuration, but there is a lot of configuring to do in order to get basic functionality.

We are going to explore Nagios with the Docker Hub image `cpuguy83/nagios`. The image uses Nagios version 3.5, which is the same version available in the Fedora repositories at the time of writing. There are later versions of Nagios available, but many still seem to prefer the version 3 series. The `cpuguy83` image takes the approach of downloading Nagios tarballs from SourceForge and installing them inside the image. If you prefer, you can use this same approach to install the Nagios version of your choosing. There is an alternative Docker file in the book sources, and in case you use it, you will need to build the image locally.

This statement starts a Nagios server container:

```
docker run -e     NAGIOSADMIN_USER=nagiosadmin -e NAGIOSAMDIN_PASS=nagios
-p 80:30000 cpuguy83/nagios
```

Verify that the Nagios web interface is available on port 30000; enter the username and password defined above to log in.

The out-of-the-box configuration of the Nagios user interface allows you to browse, amongst other things, hosts and services, but the only host defined at the outset is the localhost, which in this case is the container running Nagios itself. We will now configure another host to monitor, in the form of a separate Docker container. This will allow us to bring the monitored container down by force and verify that the Nagios container notices the outage.

This command will start a second container running `nginx` in the default configuration:

```
docker run -p 30001:80 nginx
```

Verify that the container is running by going to `port 30001`. You will see a simple message, `Welcome to nginx!`. This is more or less all that this container does out of the box, which is good enough for our test. You can, of course, use a physical host instead if you prefer.

To run the containers together, we can either link them on the command line or use a Docker compose file. Here is a Docker compose file for this scenario, which is also available in the book sources:

```
nagios:
 image: mt-nagios
 build:
   - mt-nagios
 ports:
  -  80:30000
 environment:
   - NAGIOSADMIN_USER=nagiosadmin
   - NAGIOSAMDIN_PASS=nagios
 volumes:
  ./nagios:/etc/nagios
nginx:
 image: nginx
```

The Nagios configuration is mounted in the `./nagios` directory as a Docker volume.

The configuration required for Nagios to monitor the `nginx` container is available in the sources and is also included here:

```
define host {
    name        regular-host
    use         linux-server
    register       0
    max_check_attempts   5
}

define host{
    use             regular-host
    host_name       client1
    address         192.168.200.15
    contact_groups  admins
    notes           test client1
}
hostgroups.cfg
define hostgroup {
    hostgroup_name  test-group
    alias           Test Servers
    members         client1
}

services.cfg
#+BEGIN_SRC sh
define service {
    use                     generic-service
    hostgroup_name          test-group
    service_description     PING
    check_command           check_ping!200.0,20%!600.0,60%
}
```

Let's see what happens when we bring the `nginx` container down.

Find out the hash of the `nginx` container with `docker ps`. Then kill it with `docker kill`. Verify that the container is really gone with `docker ps` again.

Now, wait a while and reload the Nagios web user interface. You should see that Nagios has alerted you of the situation.

This will look similar to the earlier screenshots that showed faulty services.

Now, you would like to have an e-mail when NGINX goes down. It is, however, not straightforward to make an example that works in a foolproof way, simply since e-mail is more complicated these days due to spam. You need to know your mail server details, such as the SMTP server details. Here is a skeleton you need to fill out with your particular details:

You can create a file `contacts.cfg` for handling e-mail, with the following contents:

```
define contact{
    contact_name                    matangle-admin
    use                             generic-contact
    alias                           Nagios Admin
    email                           pd-admin@matangle.com
}

define contactgroup{
    contactgroup_name       admins
    alias                   Nagios Administrators
    members                 matange-admin
}
```

If you don't change this configuration, the mail will wind up at `pd-admin@matangle.com`, which will be ignored.

Munin

Munin is used to graph server statistics such as memory usage, which is useful in order to understand overall server health. Since Munin graphs statistics over time, you can see resource allocation trends, which can help you find problems before they get serious. There are several other applications like Munin that create graphs; however, like Nagios, Munin is a good starting point.

It is designed to be easy to use and set up. The out-of-the-box experience gives you many graphs with little work.

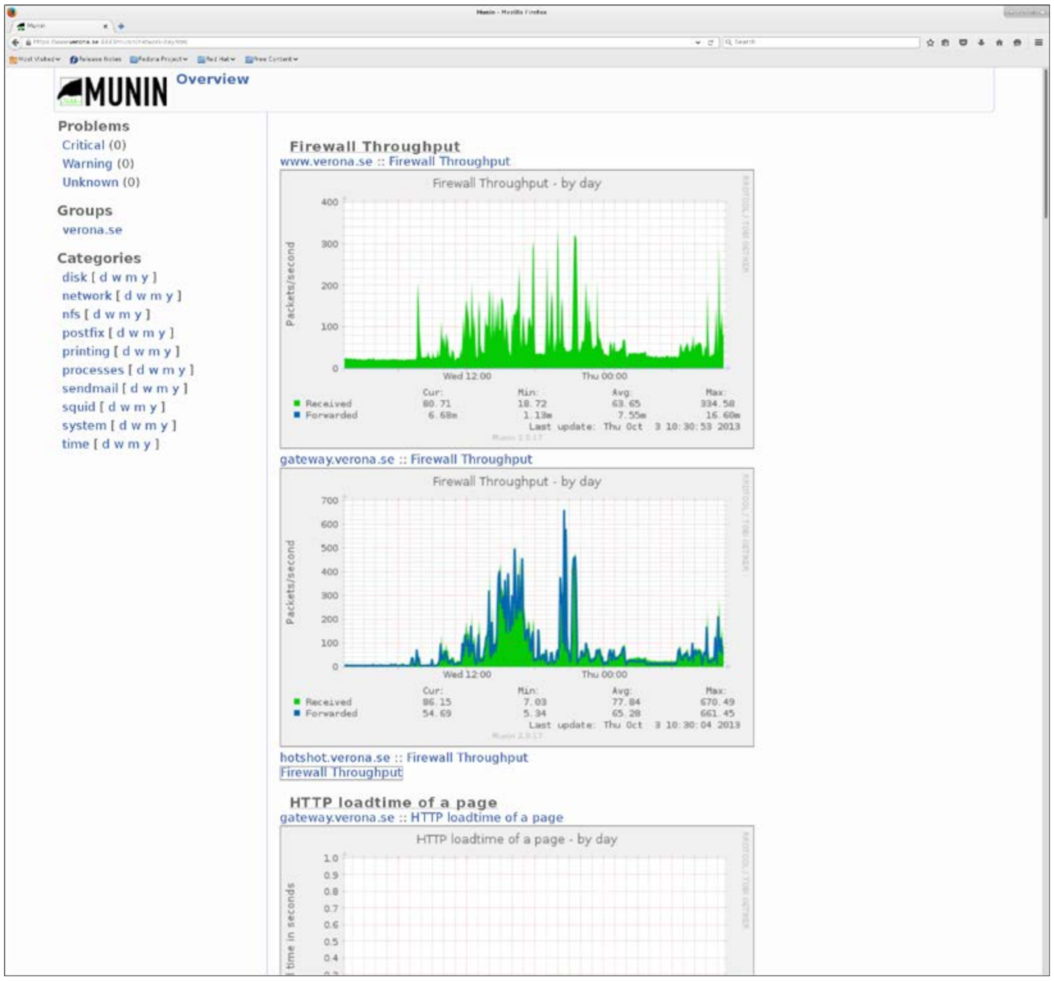
In the legends of the Norse, Hugin and Munin were two pitch-black ravens. They tirelessly flew around the world of Midgard and collected information. After their journeys far and wide, they returned to the god king Odin to sit on his shoulders and tell him all about their experiences. The name Hugin derives from the word for thought and Munin from the word for memory.

While Nagios focuses on the high-level traits of the health of a service (whether the service or host is alive or not in binary terms), Munin keeps track of statistics that it periodically samples, and draws graphs of them.

Munin can sample a lot of different types of statistics, from CPU and memory load to the number of active users in your children's Minecraft server. It is also easy to make plugins that get you the statistics you want from your own services.

An image can say a lot and convey lots of information at a glance, so graphing the memory and processor load of your servers can give you an early warning if something is about to go wrong.

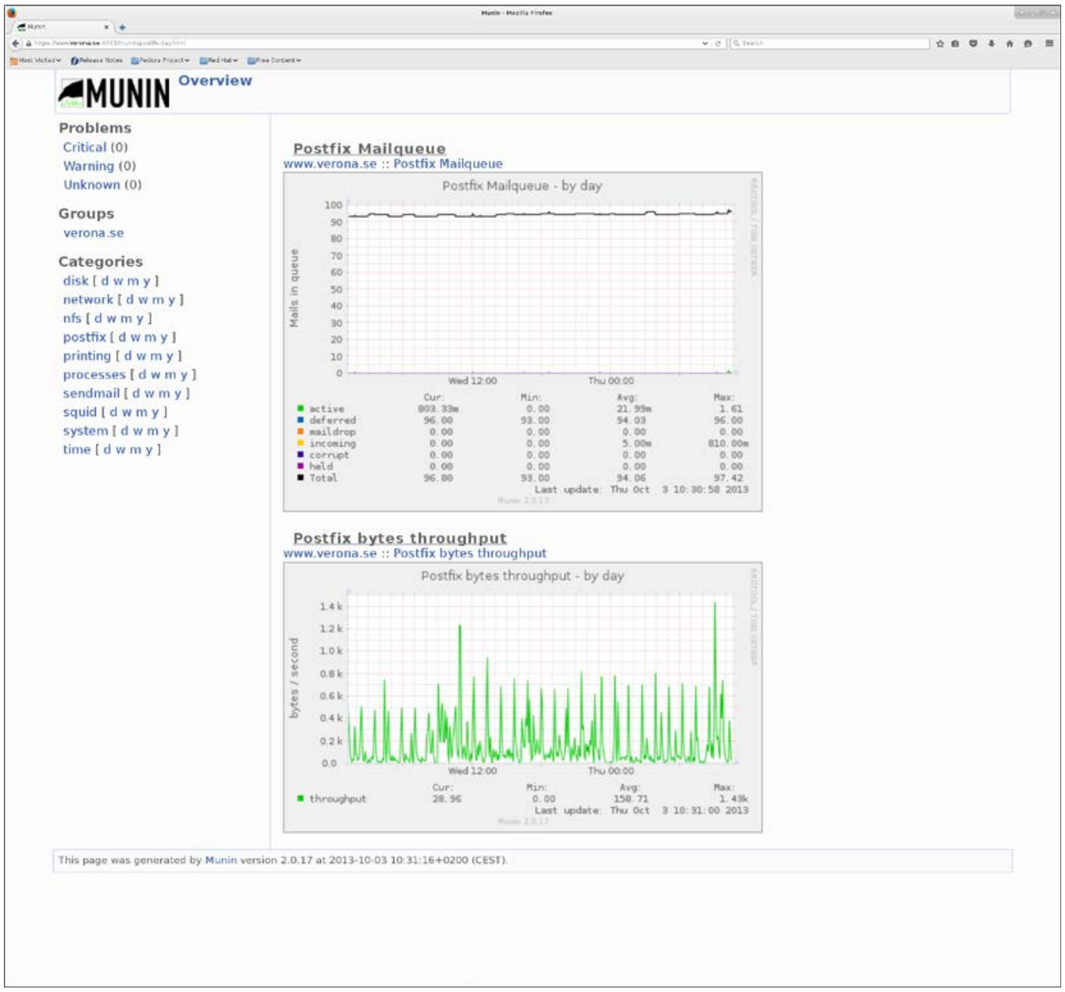
Here is an example screenshot of Munin, displaying metrics for a firewall installation at the Matangle headquarters:



```
              Cur:      Min:      Avg:       Max:
■ Received    80.71     18.72     63.65      334.58
■ Forwarded    6.68m     1.13m     7.55m      16.60m
                   Last update:  Thu Oct  3 10:30:53 2013
```

```
              Cur:      Min:      Avg:       Max:
■ Received    86.15      7.03     77.84      670.49
■ Forwarded   54.69      5.34     65.28      661.45
                   Last update:  Thu Oct  3 10:30:04 2013
```

And here are some metrics for the organization's `smtpd`, a Postfix installation:



```
            Cur:       Min:      Avg:        Max:
■ active    803.33m    0.00      21.99m      1.61
■ deferred  96.00      93.00     94.03       96.00
■ maildrop  0.00       0.00      0.00        0.00
■ incoming  0.00       0.00      5.00m       810.00m
■ corrupt   0.00       0.00      0.00        0.00
■ held      0.00       0.00      0.00        0.00
■ Total     96.80      93.00     94.06       97.42
                  Last update: Thu Oct  3 10:30:58 2013
```

```
              Cur:       Min:      Avg:        Max:
■ throughput  28.96      0.00      158.71      1.43k
                  Last update: Thu Oct  3 10:31:00 2013
```

This page was generated by Munin version 2.0.17 at 2013-10-03 10:31:16+0200 (CEST).

Like any of the monitoring systems we are exploring in this chapter, Munin also has a network-oriented architecture, as explained here. It is similar in design to Nagios. The main components of Munin are as follows:

- There is a central server, the Munin master, which is responsible for gathering data from the Munin nodes. The Munin data is stored in a database system called **RRD**, which is an acronym for **Round-robin Database**. The RRD also does the graphing of the gathered data.
- The Munin node is a component that is installed on the servers that will be monitored. The Munin master connects to all the Munin nodes and runs plugins which return data to the master.

To try it out, we will again use a Docker container running the service we are exploring, Munin:

```
docker run -p 30005:80 lrivallain/munin:latest
```

Munin will take a while to run the first time, so wait a while before checking the web interface. If you don't like to wait, you can run the `munin-update` command by hand in the container, as shown here. It polls all the Munin nodes for statistics explicitly.

```
docker run exec -it <hash> bash
su - munin --shell=/bin/bash
/usr/share/munin/munin-update
```

Now you should be able to see the graphs created during the first update. If you let the stack run for a while, you can see how the graphs develop.

It is not very hard to write a Munin plugin that monitors a statistic specific to your application stack. You can make a shell script that Munin calls in order to get the statistics you want to track.

Munin itself is written in Perl, but you can write Munin plugins in most languages as long as you conform to the very simple interface.

The program should return some metadata when called with a `config` argument. This is so that Munin can put proper labels on the graphs.

Here is an example graph configuration:

```
graph_title Load average
graph_vlabel load
load.label load
```

To emit data you simply print it to `stdout`.

```
printf "load.value "
cut -d' ' -f2  /proc/loadavg
```

Here is an example script that will graph the machine's load average:

```
#!/bin/sh

case $1 in
   config)
        cat <<'EOM'
graph_title Load average
graph_vlabel load
load.label load
EOM
        exit 0;;
esac

printf "load.value "
cut -d' ' -f2  /proc/loadavg
```

This system is pretty simple and reliable, and you can probably easily implement it for your application. All you need to do is be able to print your statistics to `stdout`.

Ganglia

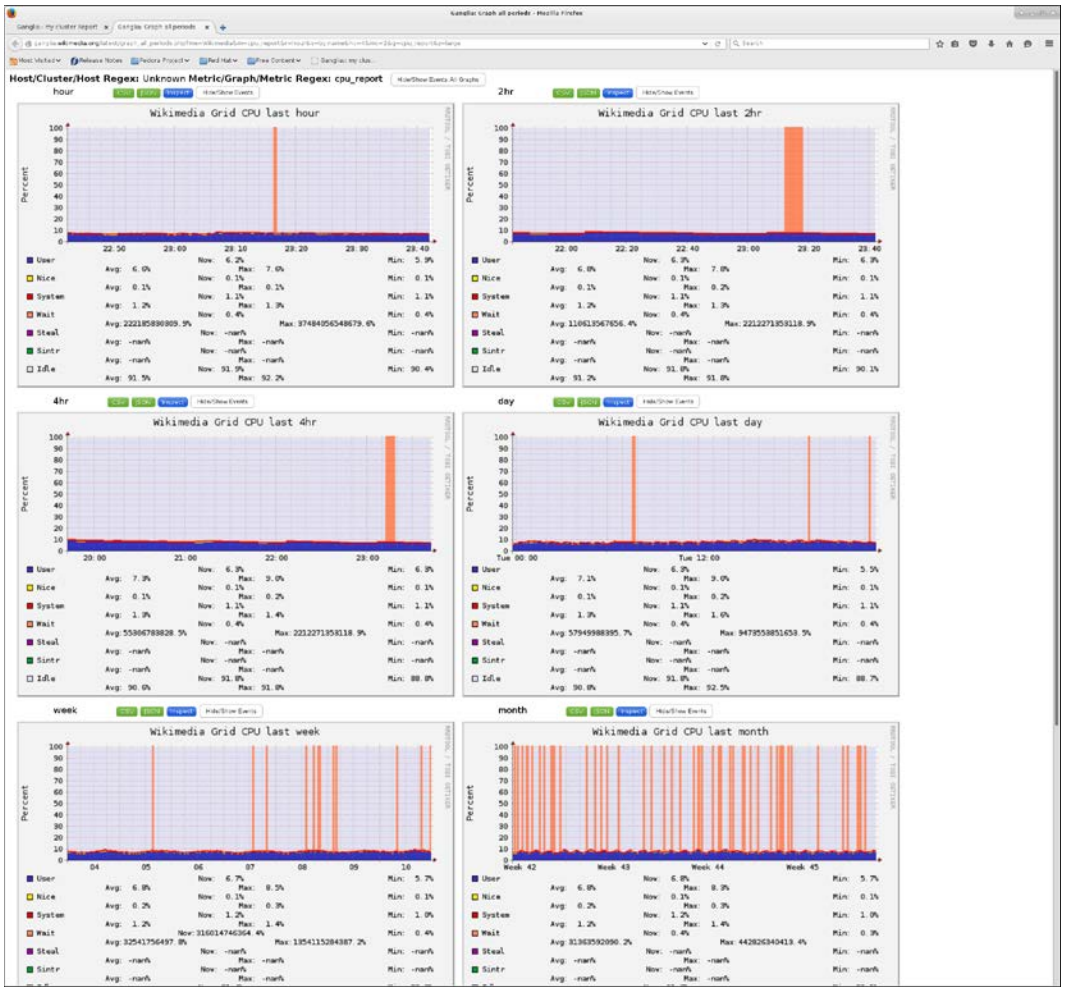
Ganglia is a graphing and monitoring solution for large clusters. It can aggregate information in convenient overview displays.

The word "ganglia" is the plural form of ganglion, which is a nerve cell cluster in anatomy. The analogy implies that Ganglia can be the sensory nerve cell network in your cluster.

Like Munin, Ganglia also uses RRDs for database storage and graphing, so the graphs will look similar to the previous Munin graphs. Code reuse is a good thing!

Ganglia has an interesting online demonstration at `http://ganglia.wikimedia.org/latest/`.

Wikimedia serves media content for Wikipedia, which is pretty busy. The demonstration thus gives us a good overview of Ganglia's capabilities, which would be hard to get in any easy way in your own network.


Host/Cluster/Host Regex: Unknown Metric/Graph/Metric Regex: cpu_report
hour
2hr
4hr
day
week
month
Wikimedia Grid CPU last hour
Wikimedia Grid CPU last 2hr
Wikimedia Grid CPU last 4hr
Wikimedia Grid CPU last day
Wikimedia Grid CPU last week
Wikimedia Grid CPU last month
Percent
User
Nice
System
Wait
Steal
Sintr
Idle

The first page shows an overview of the available data in graph format. You can drill down in the graphs to get other perspectives.

If you drill down one of the application cluster graphs for CPU load, for example, you get a detailed view of the individual servers in the cluster. You can click on a server node to drill down further.

If you have the kind of cluster scale that Wikimedia has, you can clearly see the attraction of Ganglia overviews and drill-down views. You have both a view from orbit and the details easily accessible together.

Ganglia consists of the following components:

- **Gmond**: This is an acronym for **Ganglia monitoring daemon**. Gmond is a service that collects information about a node. Gmond will need to be installed on each server that you want Ganglia to monitor.
- **Gmetad**: This stands for **Ganglia meta daemon**. Gmetad is a daemon which runs on the master node, collecting the information that all the Gmond nodes gather. Gmetad daemons can also work together to spread the load across the network. If you have a large enough cluster, the topology does indeed start to look like a nerve cell network!
- **RRD**: A round-robin database, the same tool Munin uses on the master node to store data and visualizations for Ganglia in time series that are suitable for graphing.
- **A PHP-based web frontend:** This displays the data that the master node has collected and RRD has graphed for us.

Compared to Munin, Ganglia has an extra layer, the meta daemon. This extra layer allows Ganglia to scale by distributing the network load between nodes.

Ganglia has a grid concept, where you group clusters with a similar purpose together. You can, for example, put all your database servers together in a grid. The database servers don't need to have any other relationship; they can serve different applications. The grid concept allows you to view the performance metrics of all your database servers together.

You can define grids in any manner you like: it can be convenient to view servers by location, by application, or any other logical grouping you need.

You can try Ganglia out locally as well, again, using a Docker image.

The `wookietreiber/ganglia` Docker image from Docker Hub offers some inbuilt help:

```
docker run wookietreiber/ganglia --help

Usage: docker run wookietreiber/ganglia [opts]

Run a ganglia-web container.

    -? | -h | -help | --help            print this help
    --with-gmond                        also run gmond inside the
container
    --without-gmond                     do not run gmond inside the
container
    --timezone arg                      set timezone within the
container,
                                        must be path below /usr/share/
zoneinfo,
                                        e.g. Europe/Berlin
```

In our case, we will run the image with the following command-line arguments:

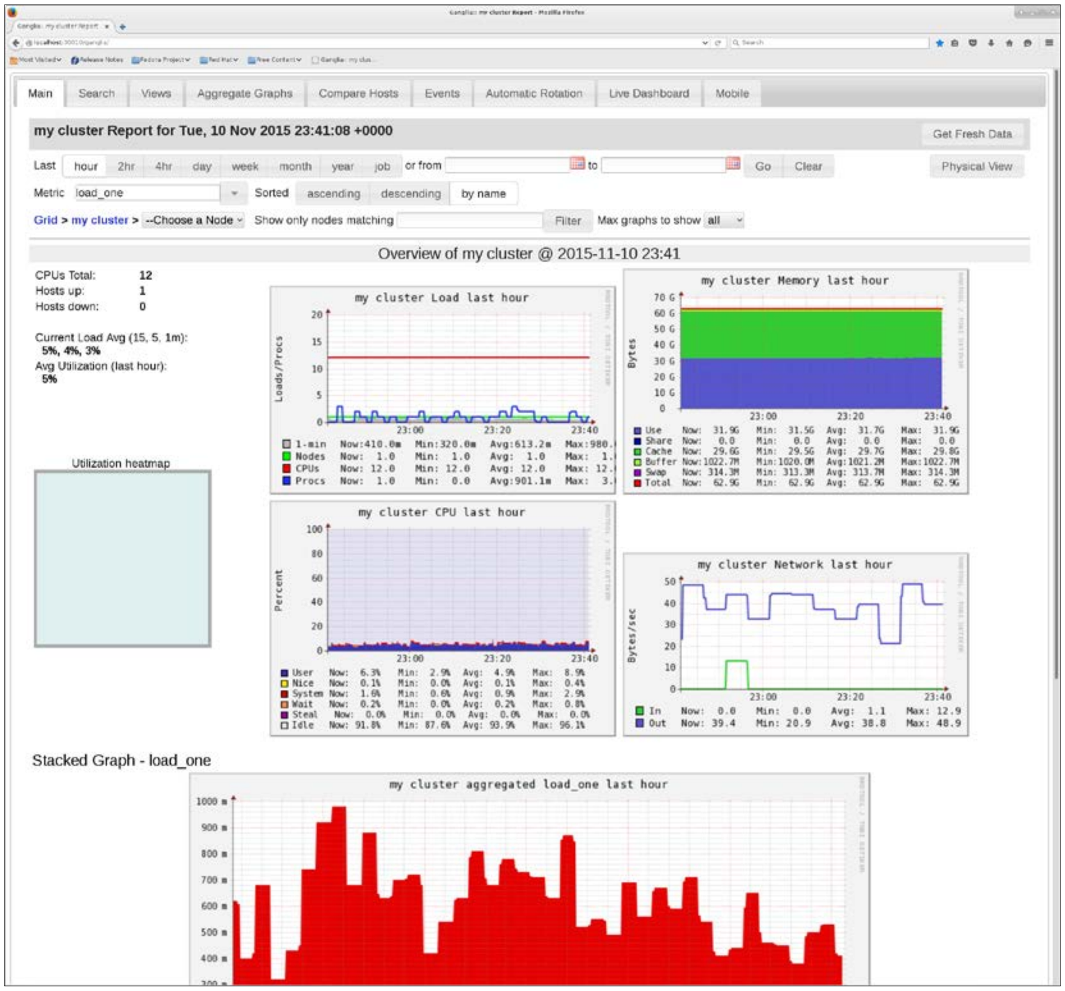
```
docker run -p 30010:80 wookietreiber/ganglia
```

This image runs Gmond as well as Gmetad. This means that it will monitor itself out of the box. We will add a separate container running Gmetad in a moment.

Now that the Ganglia container is running, you can check out the web interface served by the container:

`http://localhost:30010/ganglia/`

The browser should show a view similar to the following screenshot:



my cluster Report for Tue, 10 Nov 2015 23:41:08 +0000

Last hour 2hr 4hr day week month year job or from to

Metric load_one Sorted ascending descending by name

Grid > my cluster > --Choose a Node Show only nodes matching Max graphs to show all

Overview of my cluster @ 2015-11-10 23:41

CPUs Total: 12
Hosts up: 1
Hosts down: 0

Current Load Avg (15, 5, 1m):
5%, 4%, 3%
Avg Utilization (last hour):
5%

```
my cluster Load last hour
1-min  Now:410.0m  Min:320.0m  Avg:613.2m  Max:980.
Nodes  Now:  1.0   Min:  1.0   Avg:  1.0   Max:  1.
CPUs   Now: 12.0   Min: 12.0   Avg: 12.0   Max: 12.
Procs  Now:  1.0   Min:  0.0   Avg:901.1m  Max:  3.
```

```
my cluster Memory last hour
Use    Now:  31.9G  Min:  31.5G  Avg:  31.7G  Max:  31.9G
Share  Now:   0.0   Min:   0.0   Avg:   0.0   Max:   0.0
Cache  Now:  29.6G  Min:  29.5G  Avg:  29.7G  Max:  29.8G
Buffer Now:1022.7M  Min:1020.0M  Avg:1021.2M  Max:1022.7M
Swap   Now: 314.3M  Min: 313.3M  Avg: 313.7M  Max: 314.3M
Total  Now:  62.9G  Min:  62.9G  Avg:  62.9G  Max:  62.9G
```

```
my cluster CPU last hour
User   Now:  6.3%  Min:  2.9%  Avg:  4.9%  Max:  8.9%
Nice   Now:  0.1%  Min:  0.0%  Avg:  0.1%  Max:  0.4%
System Now:  1.6%  Min:  0.6%  Avg:  0.9%  Max:  2.9%
Wait   Now:  0.2%  Min:  0.0%  Avg:  0.2%  Max:  0.8%
Steal  Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
Idle   Now: 91.8%  Min: 87.6%  Avg: 93.9%  Max: 96.1%
```

```
my cluster Network last hour
In   Now:  0.0  Min:  0.0  Avg:  1.1  Max: 12.9
Out  Now: 39.4  Min: 20.9  Avg: 38.8  Max: 48.9
```

Stacked Graph - load_one

my cluster aggregated load_one last hour

Ganglia Gmond nodes communicate between themselves on a multicast IP channel. This offers redundancy and ease of configuration in large clusters, and the multicast configuration is default. You can also configure Ganglia in unicast mode.

Since we will only run with two communicating containers, we will not actually benefit from the multicast configuration in this case.

Again, we will use Docker Compose to start the containers together so that the separate Gmond and Gmetad instances can communicate. We can start three containers that all run `gmond`. The Gmetad daemon will in the default configuration discover them and list them together under `My Cluster`:

```
gmetad:
  image: wookietreiber/ganglia
  ports:
    - "30010:80"
gmond:
  image: wookietreiber/ganglia
gmond2:
  image: wookietreiber/ganglia
```

When you access `http://localhost:30010/ganglia/`, you should find the new `gmond` instance being monitored.

Graphite

While Munin is nice because it is robust and fairly easy to start using, the graphs it provides are only updated once in a while, normally every fifth minute. There is therefore a niche for a tool that does graphing that is closer to real time. Graphite is such a tool.

The Graphite stack consists of the following three major parts. It is similar to both Ganglia and Munin but uses its own component implementations.

- The Graphite Web component, which is a web application that renders a user interface consisting of graphs and dashboards organized within a tree-like browser widget
- The Carbon metric processing daemon, which gathers the metrics
- The Whisper time series database library

As such, the Graphite stack is similar in utility to both Munin and Ganglia. Unlike Munin and Ganglia though, it uses its own time series library, Whisper, rather than an RRD.

There are several prepackaged Docker images for trying out Graphite. We can use the `sitespeedio/graphite` image from the Docker Hub, as follows:

```
docker run -it \ -p 30020:80 \ -p 2003:2003 \ sitespeedio/graphite
```

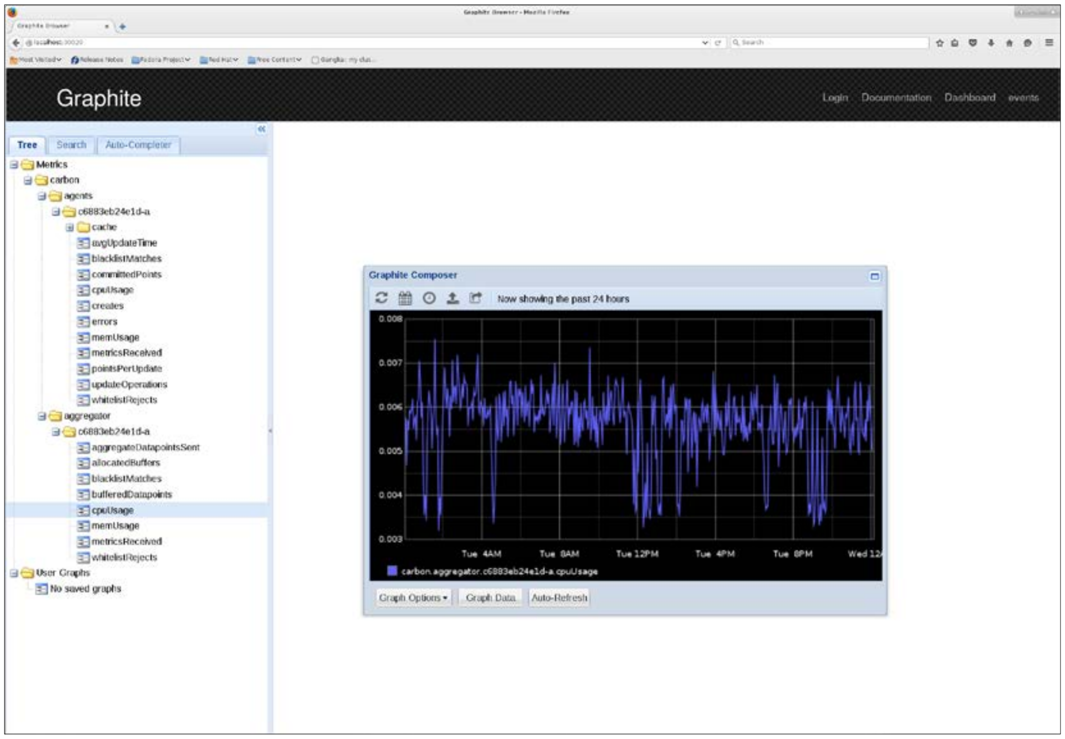
This starts a Docker container running Graphite with HTTP Basic authentication.

You can now view the user interface of Graphite. Enter "guest" for both the username and password. You can change the username and password by providing a `.htpasswd` file for the image.

If you don't change the port in the Docker statement above, this URL should work:

```
http://localhost:30020/
```

The browser window should display a view similar to the following screenshot:



Graphite has a customizable user interface, where you can organize the graphs in which you are interested together in dashboards. There is also a completion widget that lets you find named graphs by typing the first few characters of a graph name.

Log handling

Log handling is a very important concept, and we will explore some of the many options, such as the **ELK** (**Elasticsearch**, **Logstash** and **Kibana**) stack.

Traditionally, logging just consisted of using simple print statements in code to trace events in the code. This is sometimes called **printf-style debugging**, because you use traces to see how your code behaves rather than using a regular debugger.

Here is a simple example in C syntax. The idea is that we want to know when we enter the function *fn(x)* and what value the argument *x* has:

```
void fn(char *x){
  printf("DEBUG entering fn, x is %s\n", x);
 ...
}
```

From the debug traces in the console, you can determine whether the program being developed is behaving as expected.

You would, of course, also like to see whether something serious is wrong with your program and report that with a higher priority:

```
printf("ERROR x cant be an empty string\n");
```

There are several problems with this style of debugging. They are useful when you want to know how the program behaves, but they are not so useful when you have finished developing and want to deploy your code.

Today, there are a great number of frameworks that support logging in many different ways, based on the time-proven pattern above.

The logging frameworks add value to printf-style logging primarily by defining standards and offering improved functionality such as these:

- Different log priorities, such as Debug, Warning, Trace, and Error.
- Filtering log messages of different priorities. You might not always be interested in debug traces, but you are probably always interested in error messages.

- Logging to different destinations, such as files, databases, or network daemons. This includes the ELK stack that we will visit later.
- The rotating and archiving of log files. Old log files can be archived.

Every now and then, a new logging framework pops up, so the logging problem domain seems far from exhausted even to this day. This tendency is understandable, since properly done logs can help you determine the exact cause of the failure of a network service that is no longer running or, of course, any complicated service that you are not constantly supervising. Logging is also hard to do right, since excessive logging can kill the performance of your service, and too little doesn't help you determine the cause of failures. Therefore, logging systems go to great lengths to strike a balance between the various traits of logging.

Client-side logging libraries

Log4j is a popular logging framework for Java. There are several ports for other languages, such as:

- **Log4c** for the C language
- **Log4js** for the JavaScript language
- **Apache log4net**, a port to the Microsoft .NET framework

Several other ports exist, with many different logging frameworks that share many of their concepts.

Since there are many logging frameworks for the Java platform alone, there are also some wrapper logging frameworks, such as **Apache Commons Logging** or **Simple Logging Facade for Java** (**SLF4J**). These are intended to allow you to use a single interface and swap out the underlying logging framework implementation if needed.

Logback is meant to be the successor of log4j and is compatible with the ELK stack.

Log4j is the first of the Java logging frameworks, and essentially gives you the equivalent of the previous `printf` statements but with many more bells and whistles.

Log4j works with three primary constructs:

- Loggers
- Appenders
- Layouts

The logger is the class you use to access logging methods. There's a logging method to call for each log severity. Loggers are also hierarchical.

These concepts are easiest to explain with some example code:

```
Logger  logger = Logger.getLogger("se.matangle");
```

This gets us a logger specific to our organization. In this way, we can make out our own logger messages from messages that originate with other organizations' code that we might be using. This becomes very useful in a Java enterprise environment, where you might have dozens of libraries that all use log4j for their logging and you want to be able to configure different log levels for each library.

We can use several loggers at the same time for greater granularity of our logs. Different software components can have different loggers.

Here is an example from a video recording application:

```
Logger  videologger = Logger.getLogger("se.matangle.video");
logger.warn("disk is dangerously low")
```

We can use several different log levels, adding greater specificity to our logs:

```
videologger.error("video encoding source running out prematurely")
```

The place where the log message winds up in the end is called an **appender** in log4j terminology. There are a number of appenders available, such as the console, files, network destinations, and logger daemons.

Layouts control how our log messages will be formatted. They allow for escape sequences with printf-like formatting.

For example, the class `PatternLayout` configured to use the conversion pattern `%r [%t] %-5p %c - %m%n` will create the following example output:

```
176 [main] INFO  se.matangle - User found
```

Here's what the fields in the pattern stand for:

- The first field is the number of milliseconds elapsed since the start of the program
- The second field is the thread making the log request
- The third field is the level of the log statement
- The fourth field is the name of the logger associated with the log request
- The text after the - is the message of the statement

Log4j endeavors to make the configuration of the logging external to the application. By externalizing the logging configuration, a developer can make the logging work locally by configuring logs to go to a file or to the console. Later on, when the application is deployed to a production server, the administrator can configure the log appender to be something else, such as the ELK stack we are going to discuss later. This way, the code doesn't need to be changed, and we can modify the behavior and destination of the logging at deployment time.

An application server such as WildFly offers its own configuration system that plugs in to the log4j system.

If you don't use an application server, newer versions of log4j support many different configuration formats. Here is an XML-style file, which will look for `log4j2-test.xml` in the classpath:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level
%logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

The ELK stack

The ELK stack consists of the following components:

- **Logstash**: This is an open source logging framework similar to log4j. The server component of Logstash processes incoming logs.
- **Elasticsearch**: This stores all of the logs and indexes them, as the name implies, for searching.
- **Kibana**: This is the web interface for the searching and visualization of logs.

To see how it works, we will follow a log message all the way from where it originates inside the code, through the intermediary network layers, to the end destination at the network operator's screen:

1. Deep down in the code, a Java exception occurs. The application unexpectedly can't map a user ID present in an import file to the user IDs in the database. The following error is logged with log4j:

   ```
   logger.error("cant find imported user id in database")
   ```

2. The log4j system is configured to log errors to a Logstash backend, as follows:

   ```
   log4j.appender.applog=org.apache.log4j.net.SocketAppender
   log4j.appender.applog.port=5000
   log4j.appender.applog.remoteHost=master
   log4j.appender.applog.DatePattern='.'yyyy-MM-dd
   log4j.appender.applog.layout=org.apache.log4j.PatternLayout
   log4j.appender.applog.layout.ConversionPattern=%d %-5p [%t] %c %M
   - %m%n
   log4j.rootLogger=warn, applog
   log4j.logger.nl=debug
   ```

3. The Logstash system is configured as follows to listen to the ports described by the log4j system. The Logstash daemon must be started separately from your application as shown:

   ```
   input {
      log4j {
       mode => server
       host => "0.0.0.0"
       port => 5000
       type => "log4j"
     }
   }
   output {
   stdout { codec => rubydebug }
   stdout { }
     elasticsearch {
       cluster => "elasticsearch"
     }
   }
   ```

4. The Elasticsearch engine receives the log output and makes it searchable.
5. The administrator can open the Kibana GUI application and watch the log appear in real time or also search for historical data.

You might think that this is a lot of complexity, and you would be correct. But it's interesting to note that log4j can be configured to support this complex scenario as well as much simpler scenarios, so you can't really go wrong using log4j or one of its compatible competitors.

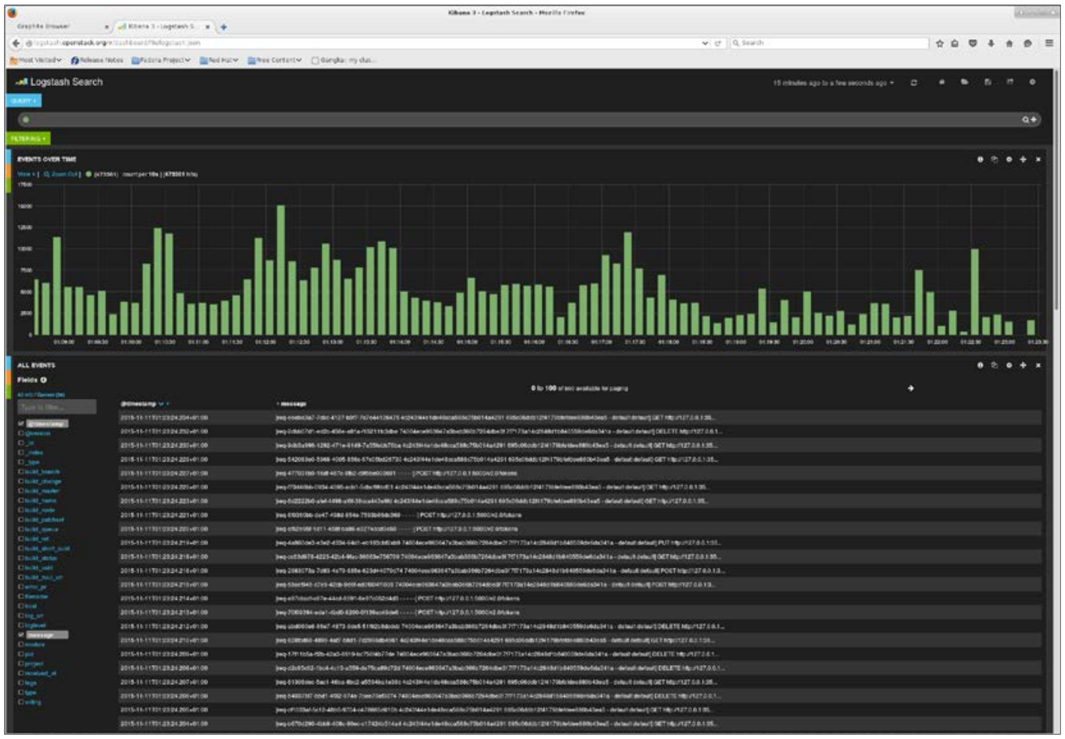
You can start out with not using any network logging and just use plain log files. This is good enough for many situations. If and when you need the extra power of the ELK stack, you can add it later. It is useful to keep the log files around as well. If the advanced logging facilities fail for some reason, you can always resort to using the traditional Unix `grep` utility on the log files.

Kibana has many features to help you analyze your log data. You can graph the data and filter log data for the particular patterns that you are looking for.

To try all of this out in practice, we can use the official Kibana and Elasticsearch Docker images available on Docker Hub:

```
docker run -d elasticsearch &&
docker run --link some-elasticsearch:elasticsearch -d kibana
```

If all goes well, we will be able to access the Kibana user interface, which looks like this:



Summary

In this chapter, we had a look at some of the many options available for monitoring our deployed code: Nagios to monitor the states of hosts and services; Munin, Ganglia, and Graphite to graph statistics from our hosts; and log4j and the ELK stack to keep track of log information.

In the next chapter, we will look at tools that help with workflows within the development organization, such as issue trackers.

9
Issue Tracking

In the previous chapter, we looked at how we can keep an eye on our deployed code with monitoring and log handling solutions.

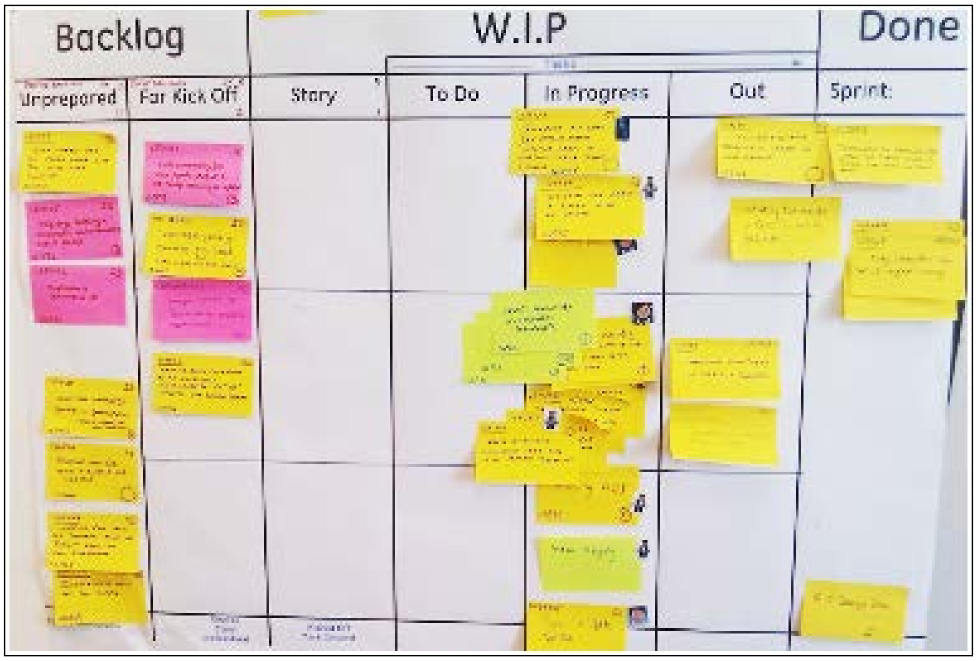
In this chapter, we will look at systems that handle development workflows within an organization, such as issue tracking software. Such systems are an important aid while implementing Agile processes.

What are issue trackers used for?

From an Agile process standpoint, issue trackers are used to help with the minutiae and details of the Agile process. The entities handled by the issue tracker might represent work items, bugs, and issues. Most Agile processes include an idea of how to manage tasks that an Agile team are to perform, in the form of Post-it notes on a board or an electronic equivalent.

When working in an Agile setting, it is common to have a board with issues on handwritten Post-it notes. This is a central concept in the Kanban method, since Kanban actually means *signboard* in Japanese. The board gives a nice overview of the work in progress and is pretty easy to manage because you just move the Post-it notes around on the board to represent state changes in the workflow.

It is also pretty easy to change your Kanban board by just rewriting the various markers, such as lanes, that you have written on your board:



Physical boards are also common with Scrum teams.

On the other hand, computerized issue trackers, which are mostly web-based, offer much better detail and can be used when working remotely. These issue trackers also help you remember steps in your process, among other benefits.

Ideally, one would like both a physical board and an issue tracker, but it is a lot of work to keep them synchronized. There is no easy way to solve this basic contention. Some people use a projector to show the issue tracker as if it was a board, but it does not have the same tactile feel to it, and gathering around a projected image or monitor is not the same thing as gathering around a physical board. Boards also have the advantage of being always available and convenient for reference when team members want to discuss tasks.

Technically, issue trackers are usually implemented as databases of objects representing state machines, with various degrees of sophistication. You create an issue with a web interface or by other means, such as e-mail or automation; then, the issue goes through various states due to human interaction, and in the end, the issue is archived in a closed state for future reference. Sometimes, the flow progresses due to an interaction from another system. A simple example might be that a task is no longer deemed valid because it is overdue and is automatically closed.

Some examples of workflows and issues

Although the term "issue" is used throughout this chapter, some systems use the terms "ticket" or "bug". Technically, they are the same thing. An issue might also represent a to-do item, enhancement request, or any other type of work item. It might feel slightly counterintuitive that, technically, an enhancement is basically the same thing as a bug, but if you see the enhancement as a missing feature, it starts to make sense.

An issue has various metadata associated with it, depending on what it represents and what the issue tracker supports. The most basic type of issue is very simple, yet useful. It has the following basic attributes:

- **Description**: This is a free-form textual description of the issue
- **Reporter**: This represents the person who opened the issue
- **Assigned**: This is the person who should work on the item.

Also, it has two states: open and closed.

This is usually the minimum that an issue tracker provides. If you compare this with a Post-it note, the only extra metadata is the reporter, and you could probably remove that as well and the tracker would still be functional. With a Post-it note, you handle open and closed states simply by putting the Post-it on the board or removing it.

When project managers first encounter an electronic issue tracker, there is often a tendency to go completely overboard with adding complexity to the state machines and attribute storage of the tracker. That said, there is clearly a lot of useful information that can be added to an issue tracker without overdoing it.

Apart from the previously mentioned basic information, these additional attributes are usually quite useful:

- **Due date**: The date on which the issue is expected to be resolved.
- **Milestone**: A milestone is a way to group issues together in useful work packages that are larger than a single issue. A milestone can represent the output from a Scrum sprint, for example. A milestone usually also has a due date, and if you use the milestone feature, the due dates on individual issues are normally not used.
- **Attachments**: It might be convenient to be able to attach screenshots and documents to an issue, which might be useful for the developer working on the issue or the tester verifying it.
- **Work estimates**: It can be useful to have an estimate of the expected work expenditure required to resolve the issue. This can be used for planning purposes. Likewise, a field with the actual time spent on the issue can also be useful for various calculations. On the other hand, it is always tricky to do estimates, and in the end, it might be more trouble than it's worth. Many teams do quite well without this type of information in their issue tracker.

The following are also some useful states that can be used to model an Agile workflow better than the plain open and closed states. The open and closed states are repeated here for clarity:

- **Open**: The issue is reported, and no one is working on it as of yet
- **In progress**: Somebody is assigned to the issue and working on resolving it
- **Ready for test**: The issue is completed and is now ready for verification. It is unassigned again
- **Testing**: Someone is assigned to work on testing the implementation of the issue
- **Done**: The task is marked as ready and unassigned once more. The done state is used to mark issues to keep track of them until the end of a sprint, for example, if the team is working with the Scrum method
- **Closed**: The issue is no longer monitored, but it is still kept around for reference

In the best case scenario, issues progress from one state to the next in an orderly fashion. In a more realistic scenario, there is a likelihood that a task goes from testing to open instead of done, since testing might reveal that the issue wasn't really properly fixed.

What do we need from an issue tracker?

What do we need from an issue tracker apart from it supporting the basic workflows described previously? There are many concerns, some of them not immediately apparent. Some things to consider are listed as follows:

- What scale do we need?

  Most tools work well on scales of up to about 20 people, but beyond that, we need to consider performance and licensing requirements. How many issues do we need to be able to track? How many users need access to the issue tracker? These are some of the questions we might have.

- How many licenses do we need?

  In this regard, free software gains the upper hand, because proprietary software can have unintuitive pricing. Free software can be free of charge, with optional support licensing.

  Most of the issue trackers mentioned in this chapter are free software, with the exception of Jira.

- Are there performance limitations?

  Performance is not usually a limiting factor, since most trackers use a production-ready database such as PostgreSQL or MariaDB as the backend database. Most issue trackers described in this chapter behave well at the scales usually associated with an organization's in-house issue tracker. Bugzilla has been proven in installations that handle a large number of issues and face the public Internet.

  Nevertheless, it is good practice to evaluate the performance of the systems you intend to deploy. Perhaps some peculiarity of your intended use triggers some performance issue. An example might be that most issue trackers use a relational database as a backend, and relational databases are not the best for representing hierarchical tree structures if one needs to do recursive queries. In normal use cases, this is not a problem, but if you intend to use deeply nested trees of issues on a large scale, problems might surface.

- What support options are available, and what quality are they?

  It is hard to determine the quality of support before actually using it, but here are some ideas to help with evaluation:

  - For open source projects, support is dependent on the size of the user community. Also, there often are commercial support providers available.
  - Commercial issue trackers can have both paid and community support.

- Can we use an issue tracker hosted off-site or not?

  There are many companies that host issue trackers. These include Atlassian's Jira tracker, which can both be installed on customer premises or hosted by Atlassian. Another example of a hosted issue tracker is Trello from Trello, Inc.

  Deploying issue trackers in your own network is usually not very hard. They are among the simpler of the systems described in this book with regard to installation complexity. Normally, you need a database backend and a web application backend layer. Usually, the hardest part is getting integration with authentication servers and mail servers to work. Nevertheless, even if hosting your own issue tracker installation isn't really hard, there is no denying that using a hosted tracker is easier.

  Some organizations can't leak data about their work outside their own networks for legal or other reasons. For these organizations, hosted issue trackers are not an option. They must deploy issue trackers inside their own networks.

- Does the issue workflow system adapt to our needs?

  Some systems, such as Jira, allow highly flexible state machines to define the flow and integrated editors to edit it. Others have more minimalistic flows. The right solution for you depends on your needs. Some people start out with wanting a very complex flow and wind up realizing they only need the open and closed states. Others realize they need complex flows to support the complexities of their process.

  A configurable flow is useful in large organizations, because it can encapsulate knowledge about processes that might not be readily apparent. For instance, when a bug has been squashed, Mike in the quality assurance department should verify the fix. This is not so useful in a small organization, because everyone knows what to do anyway. However, in a large organization, the opposite is often true. It is useful to get some help in deciding who should take care of a task next and what to do.

- Does the issue tracker support our choice of the Agile method?

  Most issue trackers mentioned in this chapter have underlying data models flexible enough to represent any sort of Agile process. All you really need, in a sense, is a configurable state machine. Issue trackers might, of course, add additional features to provide better support for particular Agile methods and workflows.

  These extra features that support Agile methods usually boil down to two primary classes: visualization and reporting. While these are nice to have, my opinion is that too much focus is placed on these features, especially by inexperienced team leaders. Visualization and reporting do add value, but not as much as one might believe at the outset.

> Keep this in mind when selecting features for your issue tracker. Make sure that you will actually use that impressive report you have been wanting.

- Does the system integrate easily with other systems in our organization?

  Examples of systems that an issue tracker might integrate with include code repositories, single sign on, e-mail systems, and so on. A broken build might cause the build server to automatically generate a ticket in the issue tracker and assign it to the developer that broke the build, for example.

  Since we are primarily concerned with development workflows here, the main integration we need is with the code repository system. Most of the issue trackers we explore here have some form of integration with a code repository system.

- Is the tracker extendable?

  It is often useful for an issue tracker to have some form of API as an extension point. APIs can be used to integrate to other systems and also customize the tracker so that it fits our organization's processes.

  Perhaps you need an HTML-formatted display of currently open issues to show on a publicly viewable monitor. This can be accomplished with a good API. Perhaps you want the issue tracker to trigger deploys in your deployment system. This can also be accomplished with a good issue tracker API.

  A good API opens up many possibilities that would otherwise be closed for us.

- Does the tracker offer multi-project support?

  If you have many teams and projects, it might be useful to have separate issue trackers for each project. Thus, a useful feature of an issue tracker is to be able to partition itself into several subprojects, each having an isolated issue tracker inside the system.

  As usual, trade-offs are involved. If you have many separate issue trackers, how do you move an issue from one tracker to another? This is needed when you have several teams working on different sets of tasks; for instance, development teams and quality assurance teams. From a DevOps perspective, it is not good if the tools pull the teams further apart rather than bringing them closer together.

  So, while it is good for each team to have its own issue tracker within the main system, there should also be good support for handling all the tasks of several teams together as a whole.

- Does the issue tracker support multiple clients?

  While not usually a feature deemed as a critical acceptance requirement for choosing an issue tracker, it can be convenient to access the issue tracker via multiple clients and interfaces. It is very convenient for developers to be able to access the issue tracker while working inside their integrated development environment, for instance.

  Some organizations that work with free software prefer a completely e-mail based workflow, such as Debian's debugs. Such requirements are rare in an enterprise setting, though.

Problems with issue tracker proliferation

As it is technically very easy to set up an issue tracker these days, it often happens that teams set up their own issue trackers to handle their own issues and tasks. It happens for many other types of tools such as editors, but editors are among the personal tools of a developer, and their primary use case isn't the sharing of collaboration surfaces with other people. Issue tracker proliferation is therefore a problem, while editor proliferation isn't.

A contributing cause of the issue tracker proliferation problem is that a large organization might standardize a type of tracker that few actually like to use. A typical reason is that only the needs of one type of team, such as the quality assurance team or the operations team, is being considered while deciding on an issue tracker. Another reason is that only a limited amount of licenses were bought for the issue tracker, and the rest of the team will have to make do on their own.

It is also quite common that the developers use one kind of tracker, the quality assurance team another, and the operations team yet another completely different and incompatible system.

So, while it is suboptimal for the organization as a whole to have many different issue tracker systems that do the same thing, it might be perceived as a win for a team that gets to choose its own tools in this area.

Again, one of the central ideas in DevOps is to bring different teams and roles closer together. Proliferation of mutually incompatible collaboration tools is not helpful in this regard. This mostly happens in large organizations, and there is usually no simple solution to the problem.

If you use a free software tracker, you at least get rid of the limited license problem. Finding a system that pleases everyone in every regard is, of course, much harder. It might help to limit the scope of selection criteria so that rather than the management, the issue tracker tool pleases the people who will actually be using it all day long. This usually also means putting less focus on reporting spiffy burn down charts and more on actually getting things done.

All the trackers

Next, we will explore a selection of different issue tracker systems. They are all easy to try out before you commit to an actual deployment. Most are free, but some proprietary alternatives are also mentioned.

All the trackers mentioned here are present on the comparison page on Wikipedia at `https://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems`.

Since we can only explore a selection of issue trackers, the following were chosen because they exhibit differences due to different design choices. Bugzilla was designed for large-scale public-facing trackers. Trac was designed for simplicity and tool integration. Redmine as a fully featured project management tool with an issue tracker. The GitLab tracker was chosen for simplicity, and Git integration and Jira for usability.

We begin our issue tracker exploration with Bugzilla, since it is one of the earliest issue trackers and is still popular. This means that its flows are pretty mature since they have proven themselves over a long period of time.

Bugzilla

Bugzilla can be said to be the grandfather of all the issue tracker systems described in this chapter. Bugzilla has been around since 1998 and is in use by many high-profile organizations, such as Red Hat, the Linux kernel project, and the Mozilla project. If you have ever reported a bug for one of these products, you have likely already encountered Bugzilla.

Bugzilla is free software maintained by the Mozilla project, which also produces the well-known Firefox browser. Bugzilla is written in Perl.

Bugzilla focuses, unsurprisingly, on tracking bugs rather than handling other issue types. Handling other types of tasks, such as providing a wiki, needs to be done separately.

Many organizations use Bugzilla as a public-facing tool where issues can be reported. As a result, Bugzilla has good security features.

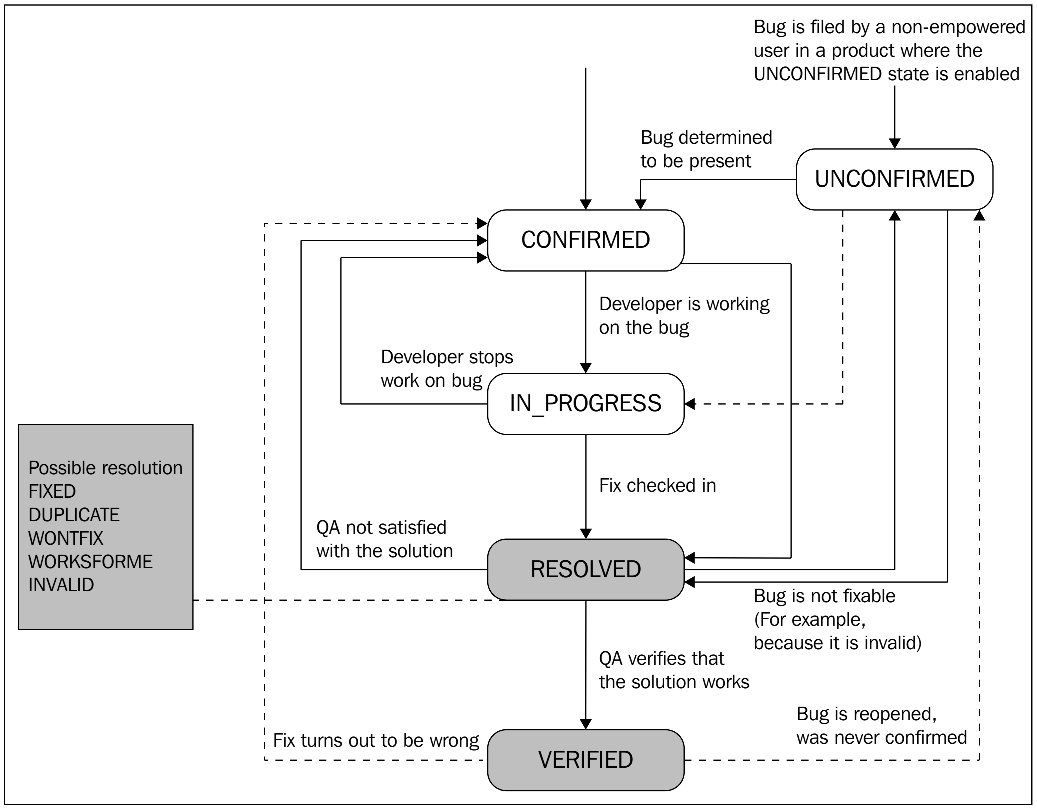
There is a demonstration installation where one can test the latest features of the development version of Bugzilla at `https://landfill.bugzilla.org/bugzilla-tip/`.

Bugzilla supports customizations. It is possible to add custom fields to issues and customize the workflow.

Bugzilla has both the XML-RPC and JSON REST APIs.

Since Bugzilla has been around for a long time, there are many extensions and plugins available, including alternative clients.

The following figure shows the Bugzilla bug state machine, or workflow:


Bug is filed by a non-empowered user in a product where the UNCONFIRMED state is enabled
Bug determined to be present
UNCONFIRMED
CONFIRMED
Developer is working on the bug
Developer stops work on bug
IN_PROGRESS
Fix checked in
Possible resolution
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
QA not satisfied with the solution
RESOLVED
Bug is not fixable (For example, because it is invalid)
QA verifies that the solution works
Bug is reopened, was never confirmed
Fix turns out to be wrong
VERIFIED

This workflow is configurable, but the default states are as follows:

- **Unconfirmed**: A new bug by a non-powered user begins in the unconfirmed state
- **Confirmed**: If the bug is confirmed to be worthy of investigation, it is put in the confirmed state
- **In progress**: When a developer starts working on resolving the bug, the in progress state is used
- **Resolved**: When developers believe that they have finished fixing the bug, they put it in the resolved state
- **Verified**: When the quality assurance team agrees that the bug is fixed, it is put in the verified state

Let's try out Bugzilla now.

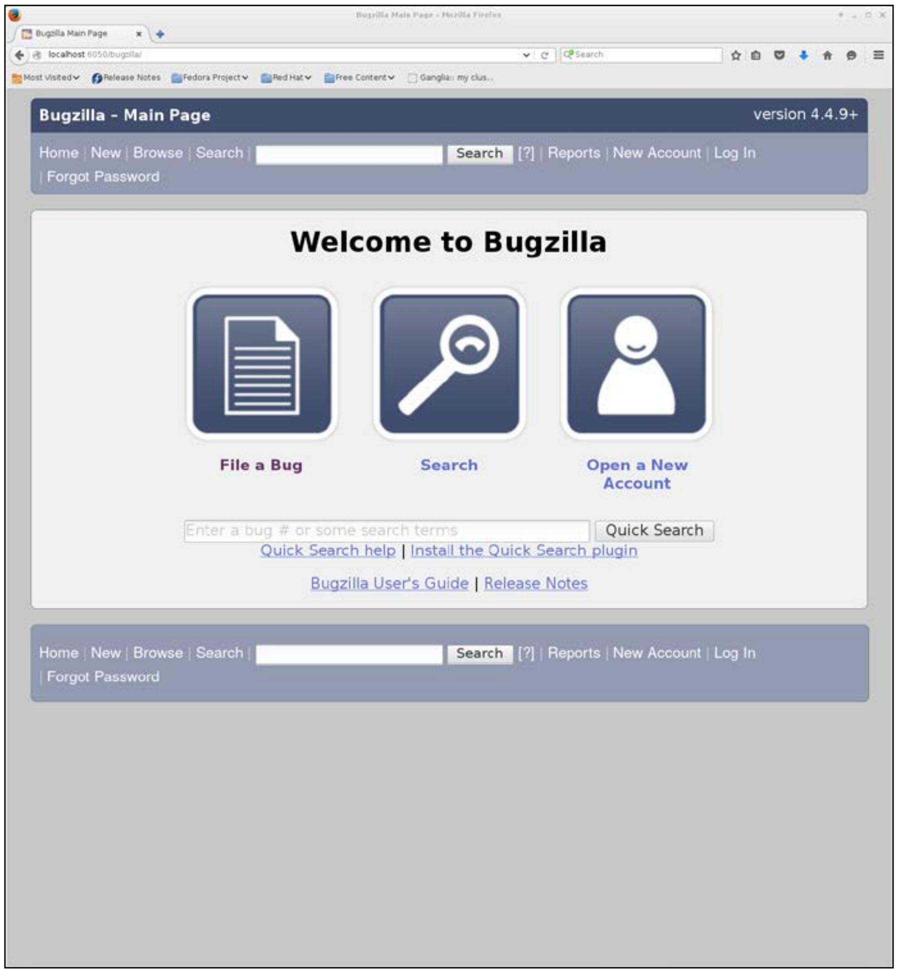
There are several Docker hub images available for Bugzilla, and it is most likely also available in the repositories of your Linux distribution.

This starts Bugzilla in a separate container, complete with a suitable MySQL configuration. The port is arbitrarily chosen:

```
docker run -p 6050:80 dklawren/docker-bugzilla
```

Try accessing the interface now, at `http://localhost:6050`.

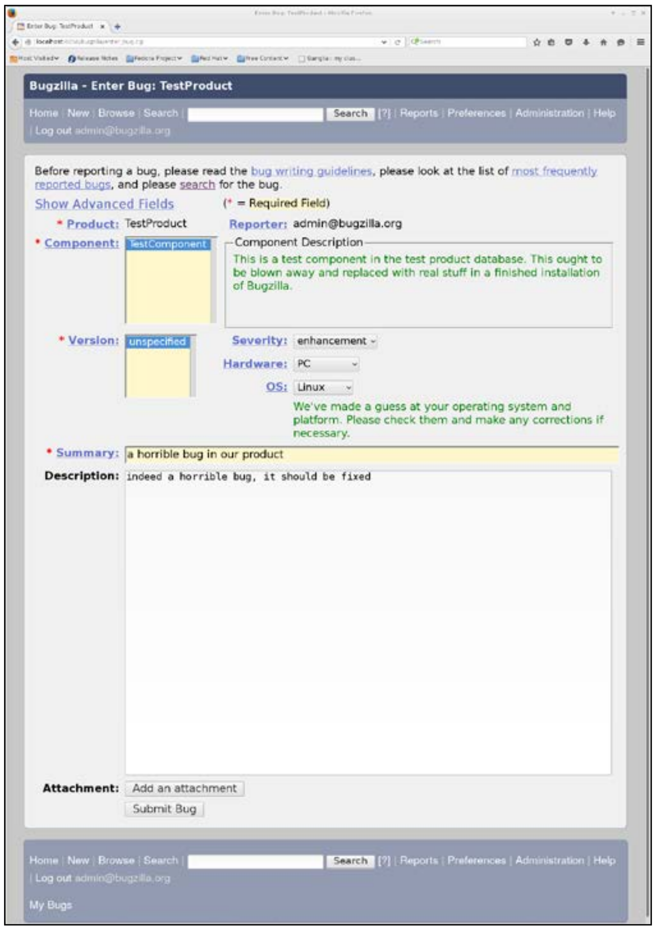
You will be greeted by this view:


Bugzilla - Main Page
version 4.4.9+
Home | New | Browse | Search | Search [?] | Reports | New Account | Log In | Forgot Password
Welcome to Bugzilla
File a Bug
Search
Open a New Account
Enter a bug # or some search terms
Quick Search
Quick Search help | Install the Quick Search plugin
Bugzilla User's Guide | Release Notes

Bugzilla requires you to have an account in order to submit bugs. You can either use the administrator account or make a new one.

You can log in using the administrator username `admin@bugzilla.org`, and the password is initially `password`. Now see the following steps to create and resolve a bug:

1. Choose **Open a New Account**. This will prompt you for your e-mail address. A confirmation mail with a link will be sent to the address. Ensure the port number is correct in the link; otherwise, create it. Your account should be created now.
2. Try creating a new bug with the **File a Bug** option. Bugzilla needs you to provide the product and component for the bug. There is a **TestProduct** and **TestComponent** option available; we can use them to report our bug:



Bugzilla - Enter Bug: TestProduct

Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

(* = Required Field)

* Product: TestProduct
Reporter: admin@bugzilla.org
* Component: TestComponent

Component Description
This is a test component in the test product database. This ought to be blown away and replaced with real stuff in a finished installation of Bugzilla.

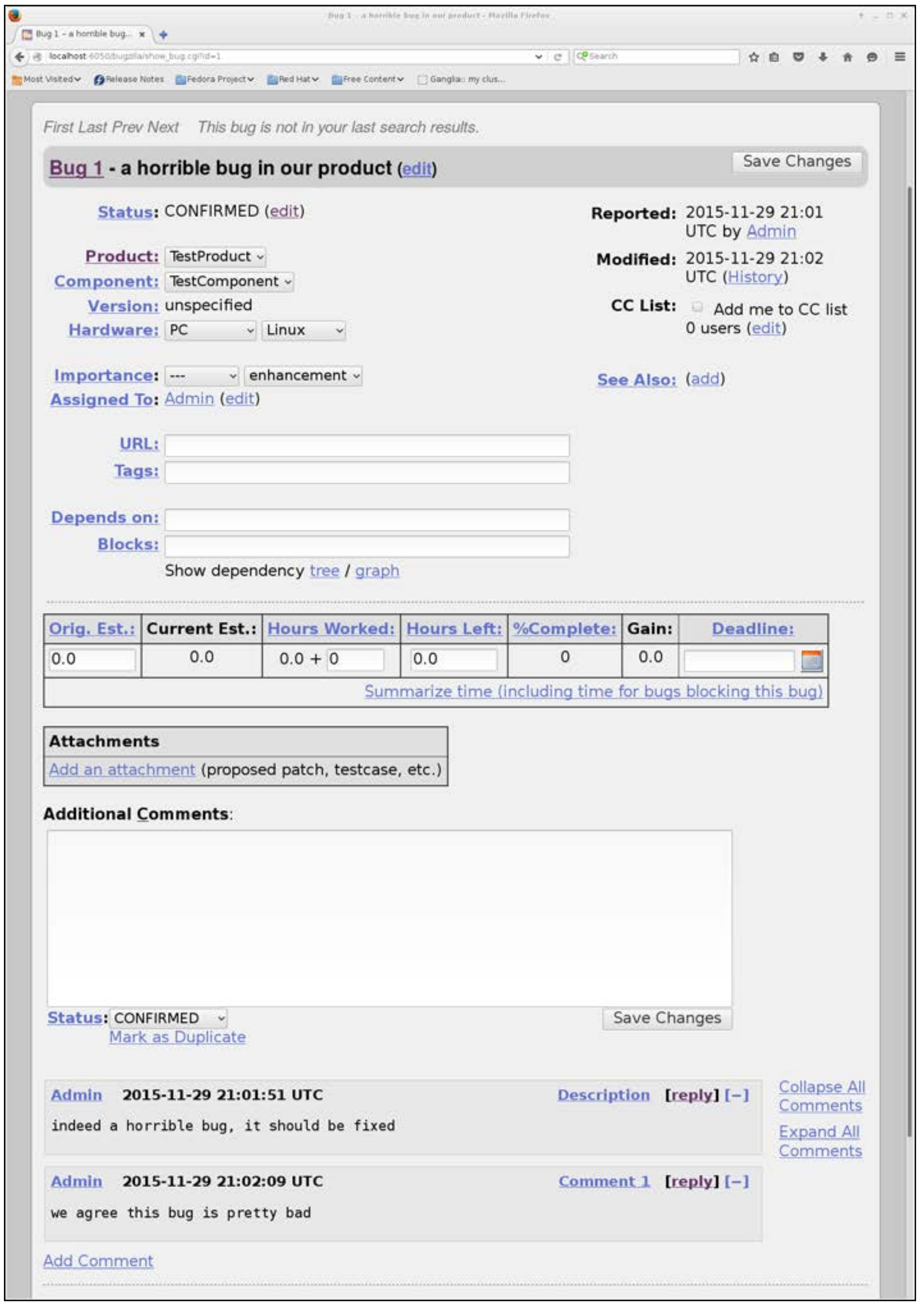
* Version: unspecified
Severity: enhancement
Hardware: PC
OS: Linux

We've made a guess at your operating system and platform. Please check them and make any corrections if necessary.

* Summary: a horrible bug in our product
Description: indeed a horrible bug, it should be fixed

Home | New | Browse | Search | Search [?] | Reports | Preferences | Administration | Help
| Log out admin@bugzilla.org
Show Advanced Fields
Attachment: Add an attachment
Submit Bug
My Bugs

3. Now that the bug is open, we can comment on it. Every time we comment on a bug, we can also move it to a new state, depending on our access level. Try adding some comments as shown in the following screenshot:


Bug 1 - a horrible bug in our product - Mozilla Firefox
Bug 1 - a horrible bug...
localhost:6050/bugzilla/show_bug.cgi?id=1
Search
Most Visited
Release Notes
Fedora Project
Red Hat
Free Content
Ganglia:: my clus...

First Last Prev Next   This bug is not in your last search results.

Bug 1 - a horrible bug in our product (edit)   Save Changes

Status:	CONFIRMED (edit)	Reported:	2015-11-29 21:01 UTC by Admin
Product:	TestProduct	Modified:	2015-11-29 21:02 UTC (History)
Component:	TestComponent	CC List:	Add me to CC list 0 users (edit)
Version:	unspecified		
Hardware:	PC Linux		
Importance:	--- enhancement	See Also:	(add)
Assigned To:	Admin (edit)		
URL:			
Tags:			
Depends on:			
Blocks:			
	Show dependency tree / graph		

Orig. Est.:	Current Est.:	Hours Worked:	Hours Left:	%Complete:	Gain:	Deadline:
0.0	0.0	0.0 + 0	0.0	0	0.0	

Summarize time (including time for bugs blocking this bug)

Attachments
Add an attachment (proposed patch, testcase, etc.)

Additional Comments:

Status: CONFIRMED   Save Changes
Mark as Duplicate

Admin 2015-11-29 21:01:51 UTC   Description [reply] [−]

```
indeed a horrible bug, it should be fixed
```

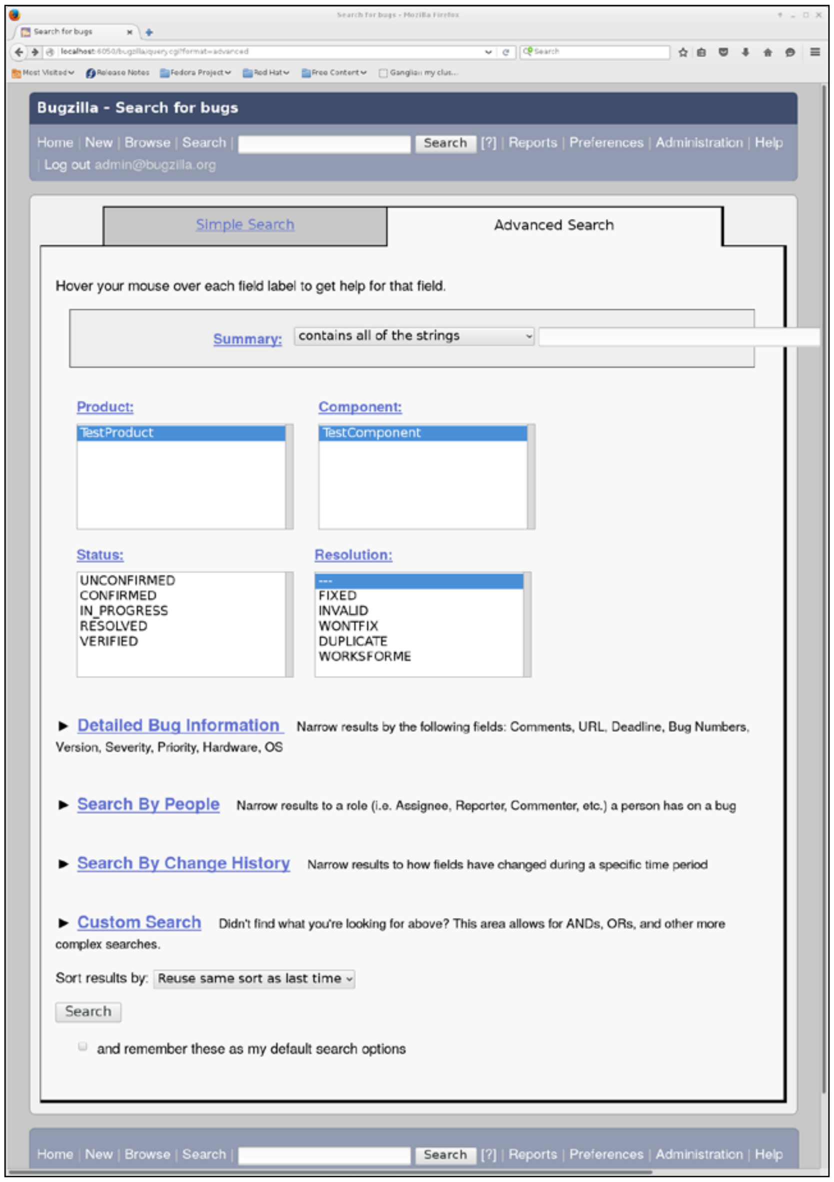
Admin 2015-11-29 21:02:09 UTC   Comment 1 [reply] [−]

```
we agree this bug is pretty bad
```

Collapse All Comments
Expand All Comments

Add Comment

4. Try searching for the bug. There is a simple search page and an advanced one. The advanced page is depicted in the following screenshot. It lets you search for most field combinations:



Bugzilla - Search for bugs

Home | New | Browse | Search | Search [?] | Reports | Preferences | Administration | Help | Log out admin@bugzilla.org

Simple Search | Advanced Search

Hover your mouse over each field label to get help for that field.

Summary: contains all of the strings

Product: TestProduct

Component: TestComponent

Status: UNCONFIRMED, CONFIRMED, IN_PROGRESS, RESOLVED, VERIFIED

Resolution: ---, FIXED, INVALID, WONTFIX, DUPLICATE, WORKSFORME

► Detailed Bug Information Narrow results by the following fields: Comments, URL, Deadline, Bug Numbers, Version, Severity, Priority, Hardware, OS

► Search By People Narrow results to a role (i.e. Assignee, Reporter, Commenter, etc.) a person has on a bug

► Search By Change History Narrow results to how fields have changed during a specific time period

► Custom Search Didn't find what you're looking for above? This area allows for ANDs, ORs, and other more complex searches.

Sort results by: Reuse same sort as last time

Search

☐ and remember these as my default search options

Home | New | Browse | Search | Search [?] | Reports | Preferences | Administration | Help

5. Eventually, when you are done, mark the bug as resolved.

As you can see, Bugzilla has all the features that can be expected from an issue tracker. It is geared towards a workflow that is optimized for working on a large number of bugs in a number of products and components. It is fairly easy to get started with Bugzilla if you are content with how it works out of the box. Customization requires some additional effort.

Trac

Trac is an issue tracker that is fairly easy to set up and test. The main attraction of Trac is that it is small and integrates a number of systems together in one concise model. Trac was one of the earliest issue trackers that showed that an issue tracker, a wiki, and a repository viewer could be beneficially integrated together.

Trac is written in Python and published under a free software license by Edgewall. It was originally licensed under the GPL, but since 2005, it has been licensed under a BSD-style license.

Trac is highly configurable through the use of a plugin architecture. There are many plugins available, a number of which are listed on `https://trac-hacks.org`.

Trac develops at a conservative pace, which many users are quite happy with. Trac also has an XML-RPC API, which can be used to read and modify issues.

Trac traditionally used to integrate only with Subversion, but now features Git support.

Trac also has an integrated wiki feature, which makes it a fairly complete system.

There are several Docker hub Trac images available, and Trac is also usually available in the repositories of Linux distributions.

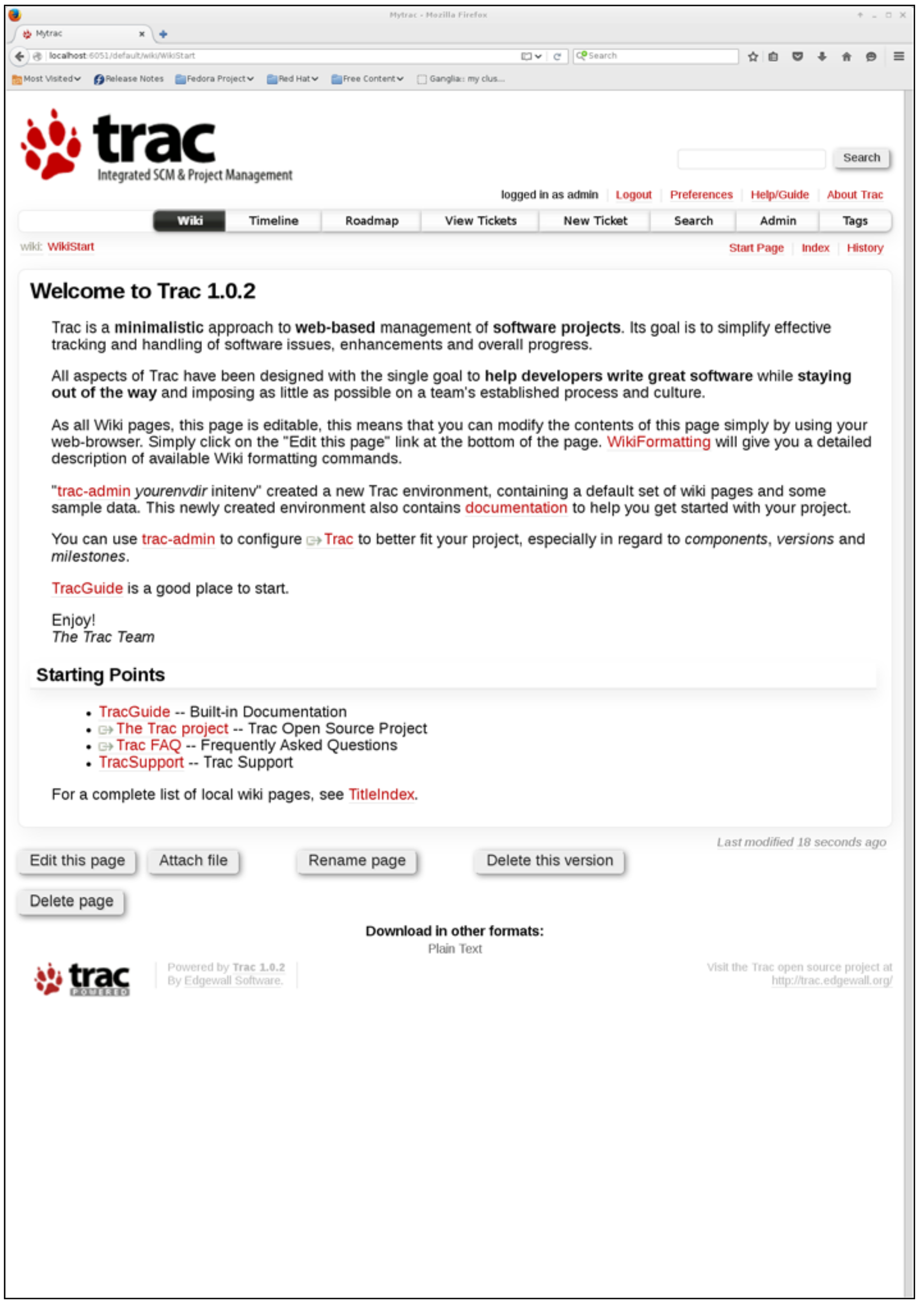
There is a Dockerfile for Trac available in this book's code bundle, which we can also use. Take a look at the following command:

```
docker run -d -p 6051:8080 barogi/trac:1.0.2
```

This will make a Trac instance available on `port 6051`. Let's try it out:

- Accessing `http://localhost:6051/` shows a list of available projects. At the outset, only the default project is available.

- The menu row at the right shows the basic features available by default, as can be seen in this screenshot:



Welcome to Trac 1.0.2

Trac is a **minimalistic** approach to **web-based** management of **software projects**. Its goal is to simplify effective tracking and handling of software issues, enhancements and overall progress.

All aspects of Trac have been designed with the single goal to **help developers write great software** while **staying out of the way** and imposing as little as possible on a team's established process and culture.

As all Wiki pages, this page is editable, this means that you can modify the contents of this page simply by using your web-browser. Simply click on the "Edit this page" link at the bottom of the page. WikiFormatting will give you a detailed description of available Wiki formatting commands.

"trac-admin *yourenvdir* initenv" created a new Trac environment, containing a default set of wiki pages and some sample data. This newly created environment also contains documentation to help you get started with your project.

You can use trac-admin to configure Trac to better fit your project, especially in regard to *components*, *versions* and *milestones*.

TracGuide is a good place to start.

Enjoy!
The Trac Team

Starting Points

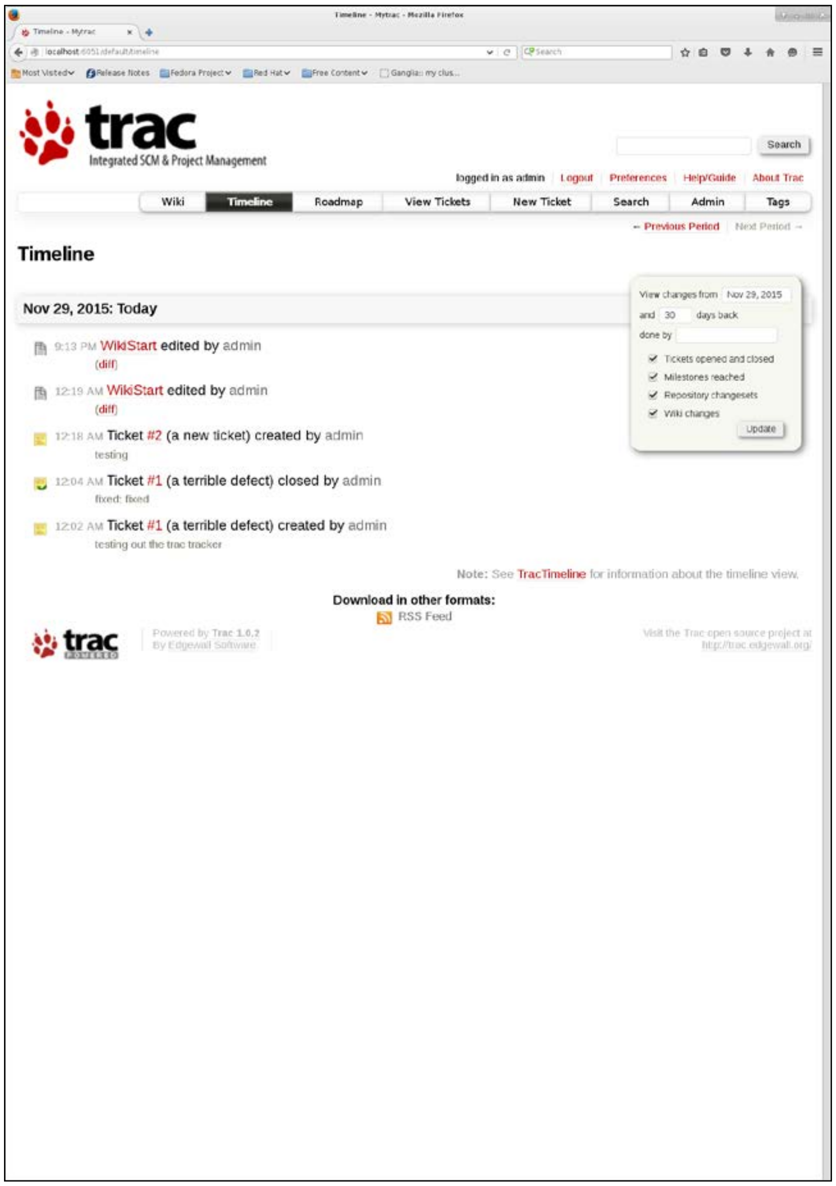
- TracGuide -- Built-in Documentation
- The Trac project -- Trac Open Source Project
- Trac FAQ -- Frequently Asked Questions
- TracSupport -- Trac Support

For a complete list of local wiki pages, see TitleIndex.

Last modified 18 seconds ago

Download in other formats:
Plain Text

- **Wiki** is a traditional wiki, where you can write documents useful for development, for instance.
- **Timeline** will show events that have occurred in Trac:


Timeline - Mytrac - Mozilla Firefox
localhost:6051/default/timeline
Most Visited Release Notes Fedora Project Red Hat Free Content Ganglia:: my clus...
trac
Integrated SCM & Project Management
Search
logged in as admin | Logout | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | View Tickets | New Ticket | Search | Admin | Tags
← Previous Period | Next Period →

Timeline

Nov 29, 2015: Today

9:13 PM WikiStart edited by admin
(diff)

12:19 AM WikiStart edited by admin
(diff)

12:18 AM Ticket #2 (a new ticket) created by admin
testing

12:04 AM Ticket #1 (a terrible defect) closed by admin
fixed: fixed

12:02 AM Ticket #1 (a terrible defect) created by admin
testing out the trac tracker

View changes from Nov 29, 2015
and 30 days back
done by

✓ Tickets opened and closed
✓ Milestones reached
✓ Repository changesets
✓ Wiki changes

Update

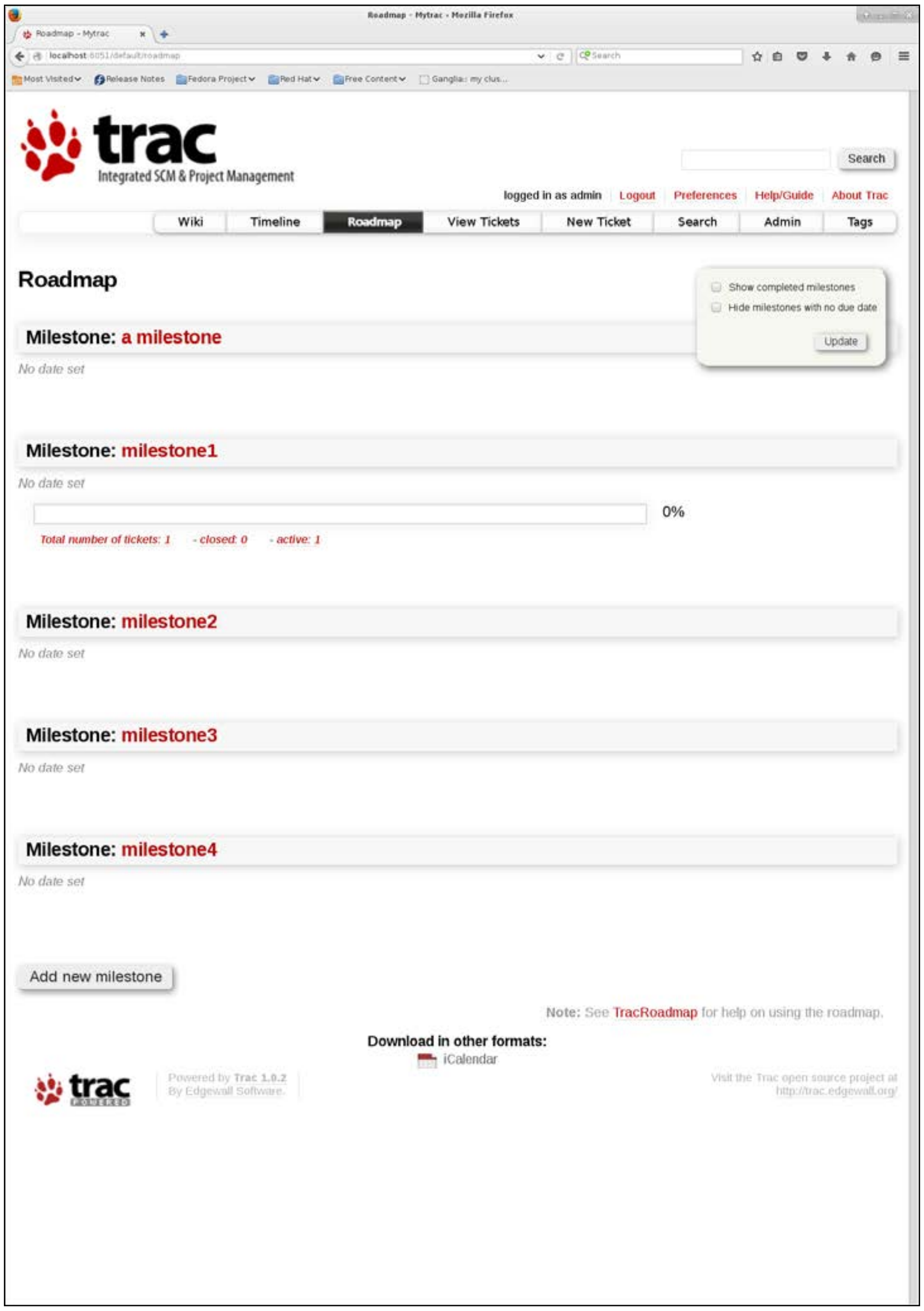
Note: See TracTimeline for information about the timeline view.

Download in other formats:
RSS Feed

Powered by Trac 1.0.2
By Edgewall Software.

Visit the Trac open source project at
http://trac.edgewall.org/

- **Roadmap** will show tickets organized into milestones:



Roadmap

Milestone: a milestone

No date set

Milestone: milestone1

No date set

0%

Total number of tickets: 1 - closed: 0 - active: 1

Milestone: milestone2

No date set

Milestone: milestone3

No date set

Milestone: milestone4

No date set

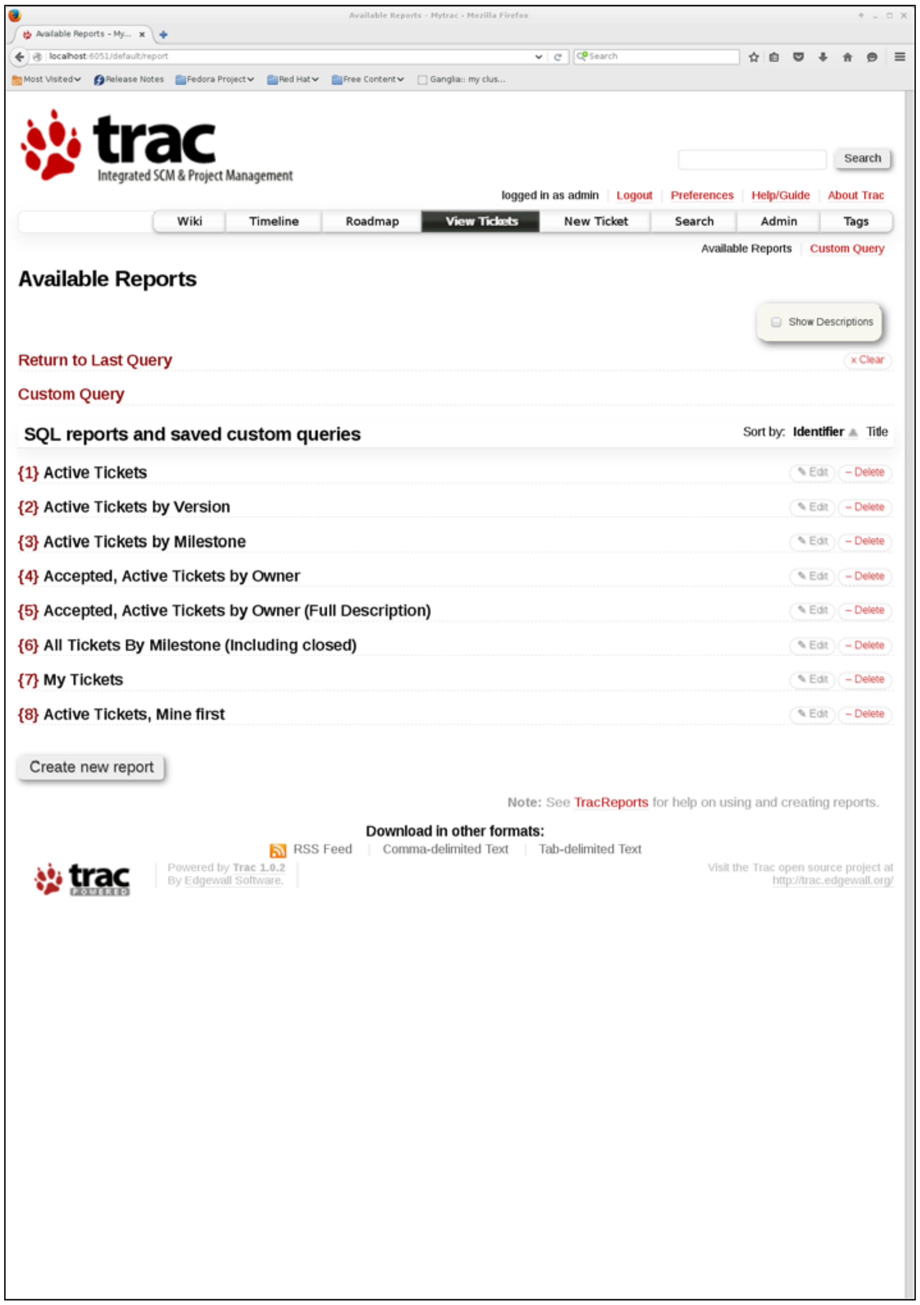
Note: See TracRoadmap for help on using the roadmap.

Download in other formats:

Powered by Trac 1.0.2
By Edgewall Software.

Visit the Trac open source project at
http://trac.edgewall.org/

- **View Tickets**, as the name implies, shows tickets. There are several possible reports, and you can create new ones. This is a very useful feature that lets different teams and roles see the information that is relevant to them:


Available Reports - Mytrac - Mozilla Firefox
localhost:8051/default/report
trac
Integrated SCM & Project Management
Search
logged in as admin | Logout | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | View Tickets | New Ticket | Search | Admin | Tags
Available Reports | Custom Query
Show Descriptions
× Clear
Sort by: Identifier Title
Edit
Delete
Create new report
RSS Feed | Comma-delimited Text | Tab-delimited Text
Powered by Trac 1.0.2
By Edgewall Software.
Visit the Trac open source project at http://trac.edgewall.org/

Available Reports

Return to Last Query

Custom Query

SQL reports and saved custom queries

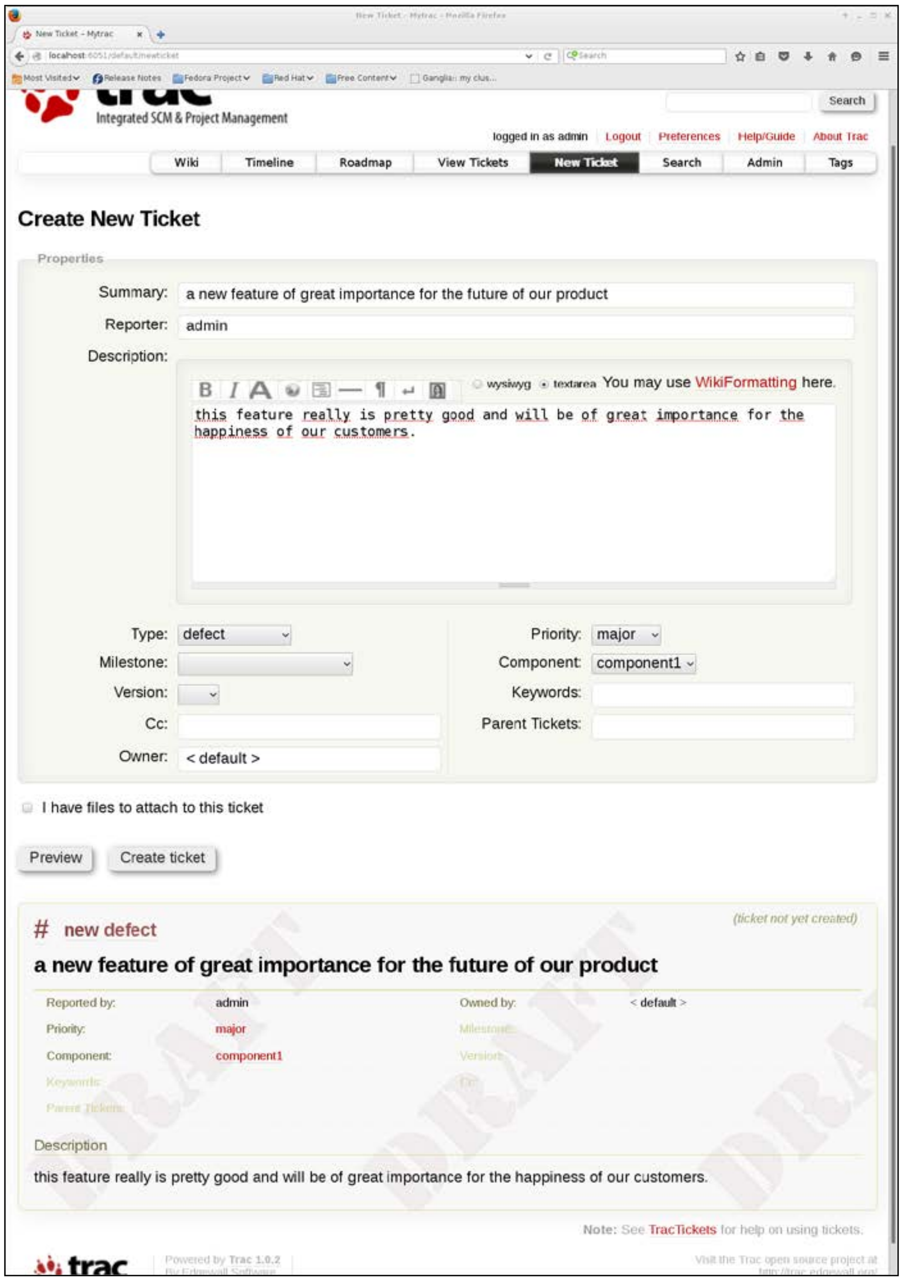
{1} Active Tickets
{2} Active Tickets by Version
{3} Active Tickets by Milestone
{4} Accepted, Active Tickets by Owner
{5} Accepted, Active Tickets by Owner (Full Description)
{6} All Tickets By Milestone (Including closed)
{7} My Tickets
{8} Active Tickets, Mine first

Note: See TracReports for help on using and creating reports.

Download in other formats:

Now, try logging in with both the username and password as `admin`.

You now get a new menu bar entry called **New Ticket**. Try creating a ticket. As you type, a live preview will be updated in the lower part of the window:



New Ticket – Mytrac – Mozilla Firefox

localhost:8051/default/newticket

Integrated SCM & Project Management

logged in as admin | Logout | Preferences | Help/Guide | About Trac

Wiki | Timeline | Roadmap | View Tickets | New Ticket | Search | Admin | Tags

Create New Ticket

Properties

Summary: a new feature of great importance for the future of our product

Reporter: admin

Description: wysiwyg ◉ textarea You may use WikiFormatting here.

```
this feature really is pretty good and will be of great importance for the
happiness of our customers.
```

Type:	defect	Priority:	major
Milestone:		Component:	component1
Version:		Keywords:	
Cc:		Parent Tickets:	
Owner:	< default >		

I have files to attach to this ticket

Preview | Create ticket

(ticket not yet created)

new defect

a new feature of great importance for the future of our product

Reported by:	admin	Owned by:	< default >
Priority:	major	Milestone:	
Component:	component1	Version:	
Keywords:		Cc:	
Parent Tickets:			

Description

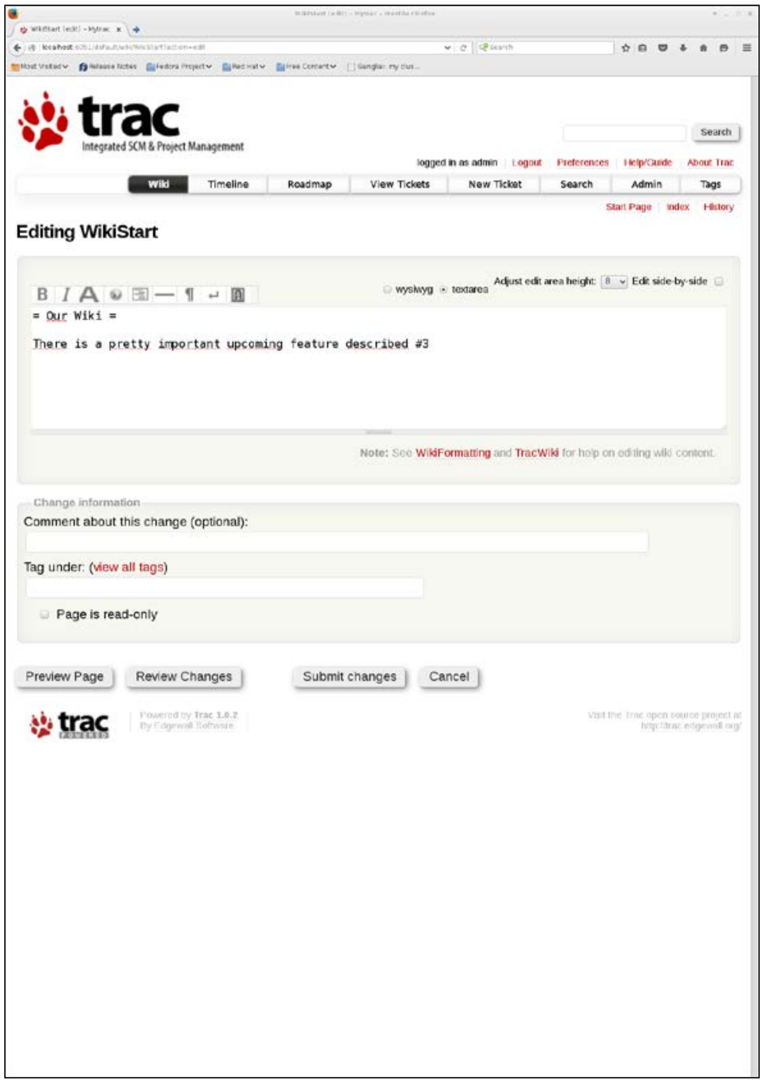
this feature really is pretty good and will be of great importance for the happiness of our customers.

Note: See TracTickets for help on using tickets.

Powered by Trac 1.0.2

The ticket is now in the "new" state. We can change the state with the **Modify** button. Try resolving the ticket and commenting on it. The next state will be "closed".

Since one of the main selling points of Trac is the tight integration of a Wiki, issue tracker, and repository, let's try the integration out. Edit the main Wiki page. Trac uses a simple wiki syntax. Create a new heading, using an equals sign to mark it as shown in the following screenshot:


trac
Integrated SCM & Project Management
Search
logged in as admin | Logout | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | View Tickets | New Ticket | Search | Admin | Tags
Start Page | Index | History
Editing WikiStart
wysiwyg textarea Adjust edit area height: 8 Edit side-by-side
Note: See WikiFormatting and TracWiki for help on editing wiki content.
Change information
Comment about this change (optional):
Tag under: (view all tags)
Page is read-only
Preview Page
Review Changes
Submit changes
Cancel

```
= Our Wiki =

There is a pretty important upcoming feature described #3
```

When we submit the changes, the issue identifier becomes clickable, and we can visit the issue.

If you mention a bug in the commit message of a code change, Trac will make it clickable as well in its changeset browser. This is a simple and convenient feature inspired by wikis.

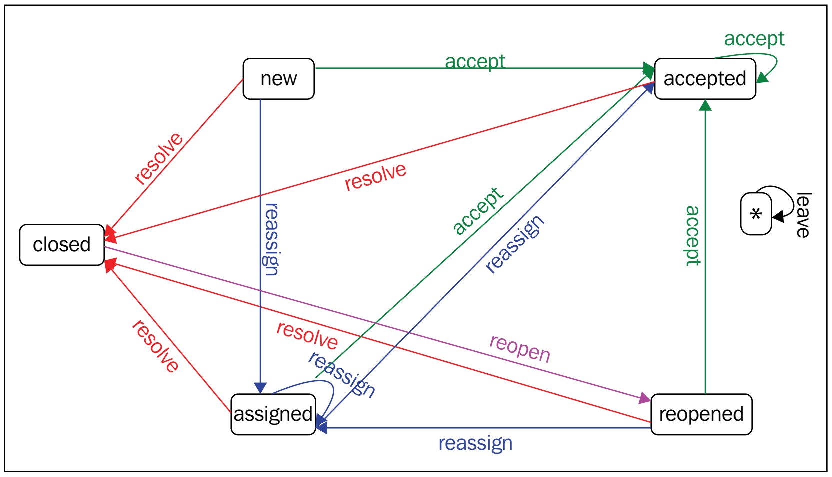
The Admin interface allows customization of the Trac instance. You can manage many of the entities in Trac directly in the interface, but it's not possible to manage every aspect of Trac from here. You need to configure some of the following aspects in the filesystem:

- Users
- Components
- Milestones
- Priorities
- Resolutions
- Severity
- Ticket types

Trac has a configurable state machine. By default, new installations get a state machine only slightly more complex than the minimal open/closed state machine. The state machine is as follows:

- New
- Assigned
- Accepted

- Closed
- Reopened



The Trac distribution comes with a set of example workflow configurations in the form of `.ini` files that describe workflow states and transitions. It's not possible to edit the workflow in the GUI at present.

This concludes our look at the Trac issue tracker. It is attractive, if you like the simplicity and tool integration.

Redmine

Redmine is a popular issue tracker written in Ruby on Rails. Its design shares many similarities with Trac.

Redmine has two forks as well: ChiliProject and OpenProject. While ChiliProject's development has stalled, OpenProject continues to be developed.

Redmine is free software. It is similar to Trac in the sense that it integrates several tools together into a coherent whole. It is also web based, like the other issue trackers we explore here.

Redmine has a number of additional features when compared with the out-of-the-box experience of Trac. Some of them are as follows:

- Multiple projects can be managed from within the web interface
- Gantt charts and calendars can be used

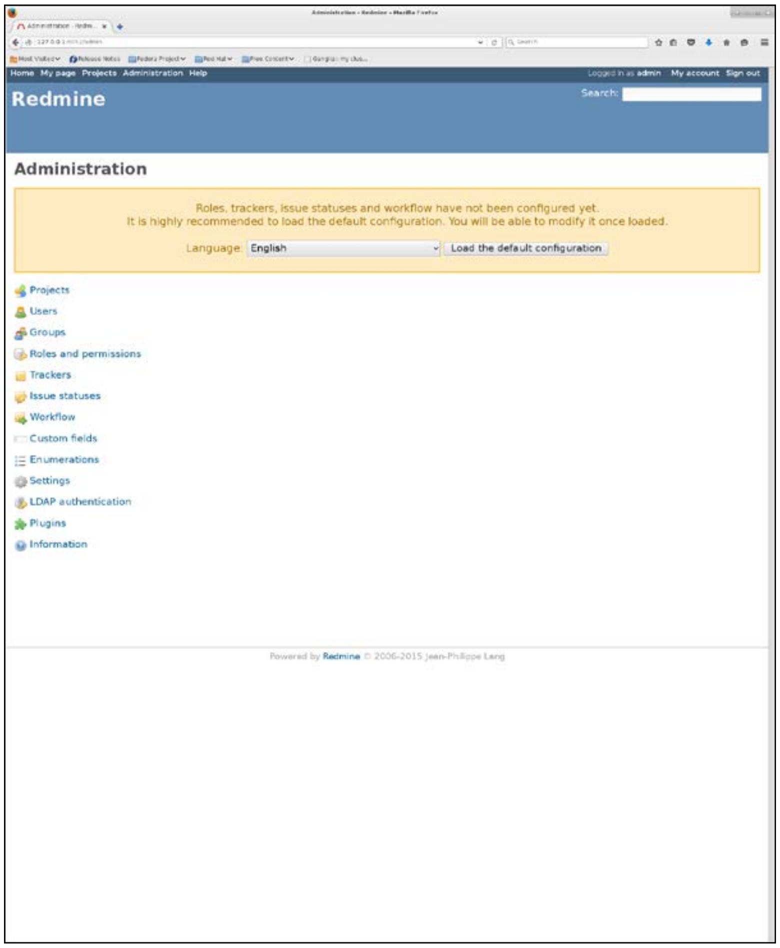
There is a Dockerfile for Redmine available in this book's code bundle, and there is also an official Redmine image available at the Docker hub. Take a look at this command:

```
docker run -d -p  6052:3000 redmine
```

This will start a Redmine instance available on the host `port 6052`. This instance will use a SQLite database, so it will not scale to production usage. You can configure the Redmine container to use a Postgres or MariaDB container as a database if and when you decide to do a production installation.

You can log in with the default username and password, both `admin`.

You need to initialize the database manually before proceeding. Use the **Administration** link at the top of the page. Follow the instructions on the page:



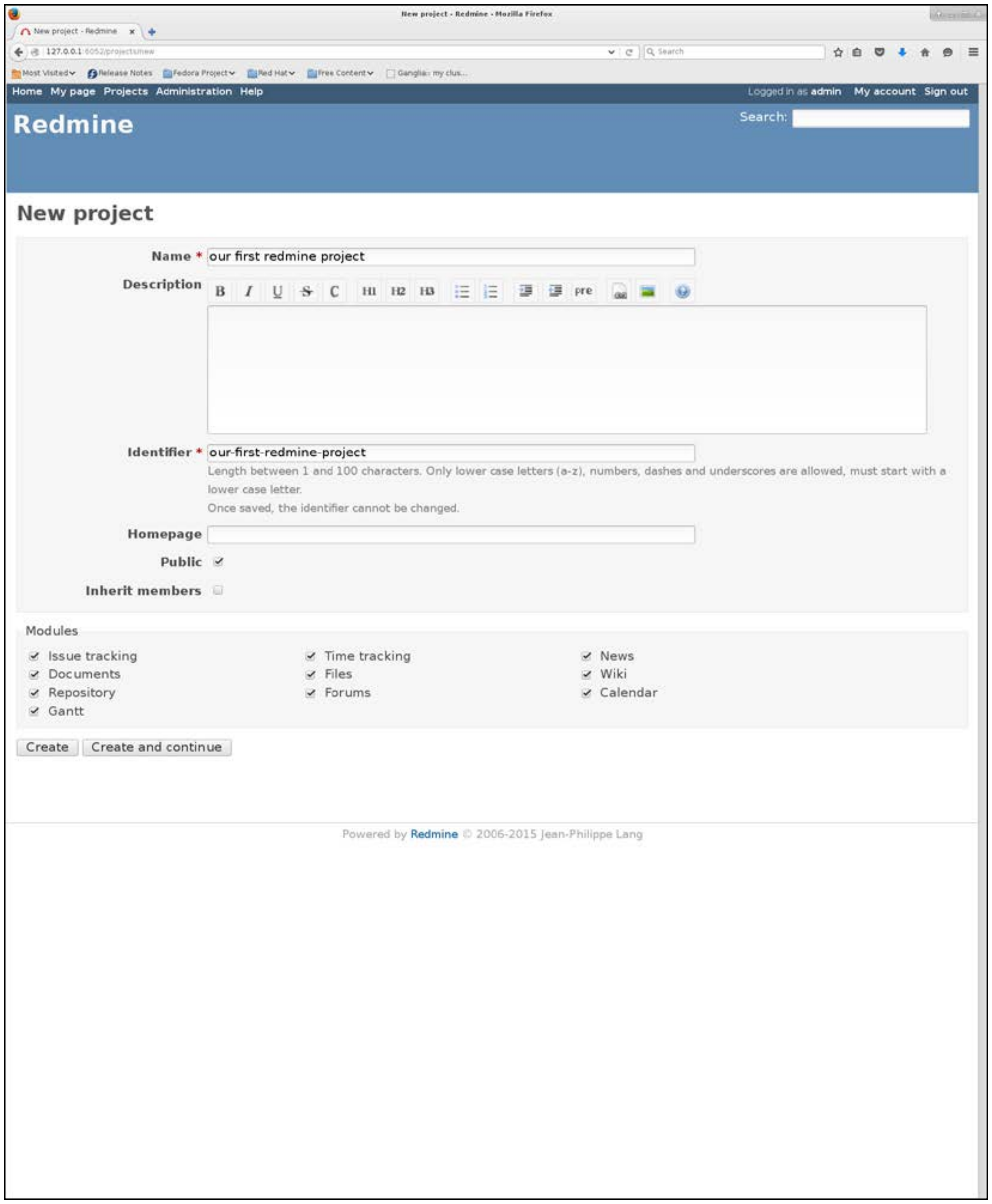
Administration

Roles, trackers, issue statuses and workflow have not been configured yet.
It is highly recommended to load the default configuration. You will be able to modify it once loaded.

Language: English

Powered by Redmine © 2006-2015 Jean-Philippe Lang

You can create a new project from within the user interface. If you don't follow the previous database initialization step first, the new project will not have the necessary configuration, and you won't be able to create issues. Redmine's **New project** view is included in the screenshot below:


New project - Redmine - Mozilla Firefox

127.0.0.1:6052/projects/new

Home My page Projects Administration Help — Logged in as admin My account Sign out

Redmine — Search:

New project

Name *	our first redmine project
Description	
Identifier *	our-first-redmine-project Length between 1 and 100 characters. Only lower case letters (a-z), numbers, dashes and underscores are allowed, must start with a lower case letter. Once saved, the identifier cannot be changed.
Homepage	
Public	✓
Inherit members	☐

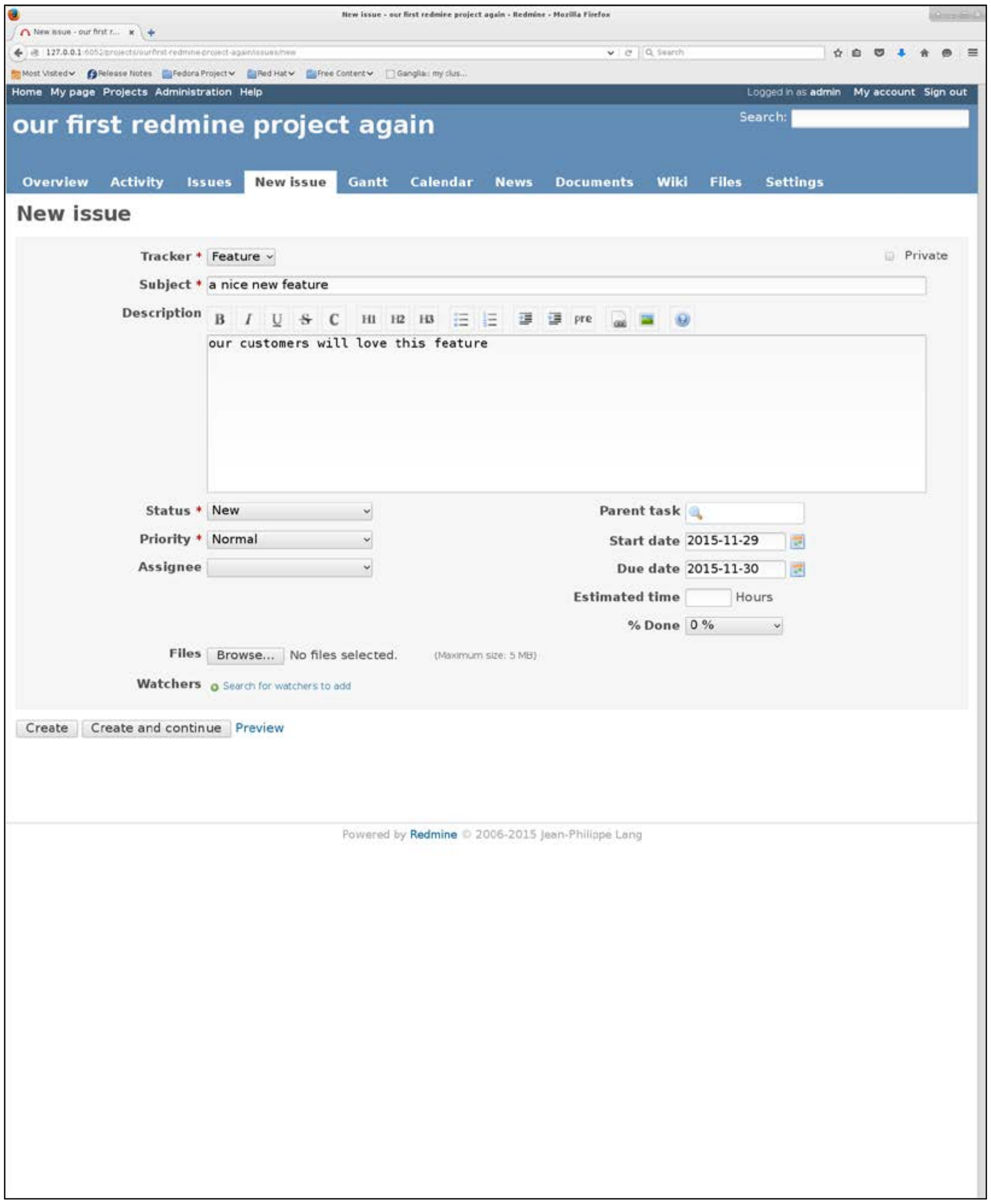
Modules

✓ Issue tracking	✓ Time tracking	✓ News
✓ Documents	✓ Files	✓ Wiki
✓ Repository	✓ Forums	✓ Calendar
✓ Gantt		

Create | Create and continue

Powered by Redmine © 2006-2015 Jean-Philippe Lang

Now that we have created a project, we can now create an issue. With Redmine, you choose a class for your issue: by default, **Bug**, **Feature**, and **Support**. Let's make a new feature issue:


New issue - our first redmine project again - Redmine - Mozilla Firefox
Home My page Projects Administration Help
Logged in as admin My account Sign out
Search:
Overview Activity Issues New issue Gantt Calendar News Documents Wiki Files Settings
Browse...
Search for watchers to add
Create
Create and continue
Preview
Powered by Redmine © 2006-2015 Jean-Philippe Lang

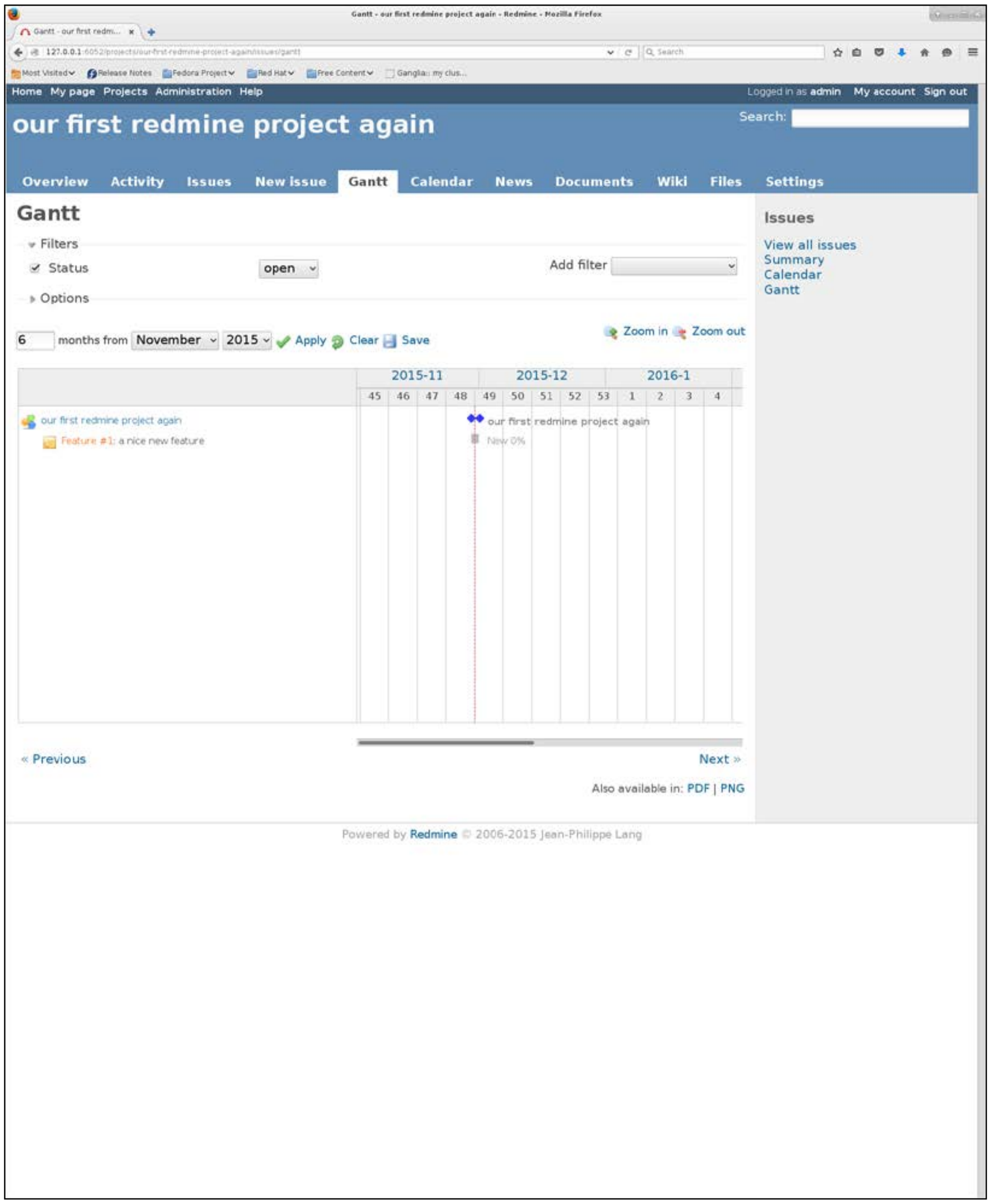
our first redmine project again

New issue

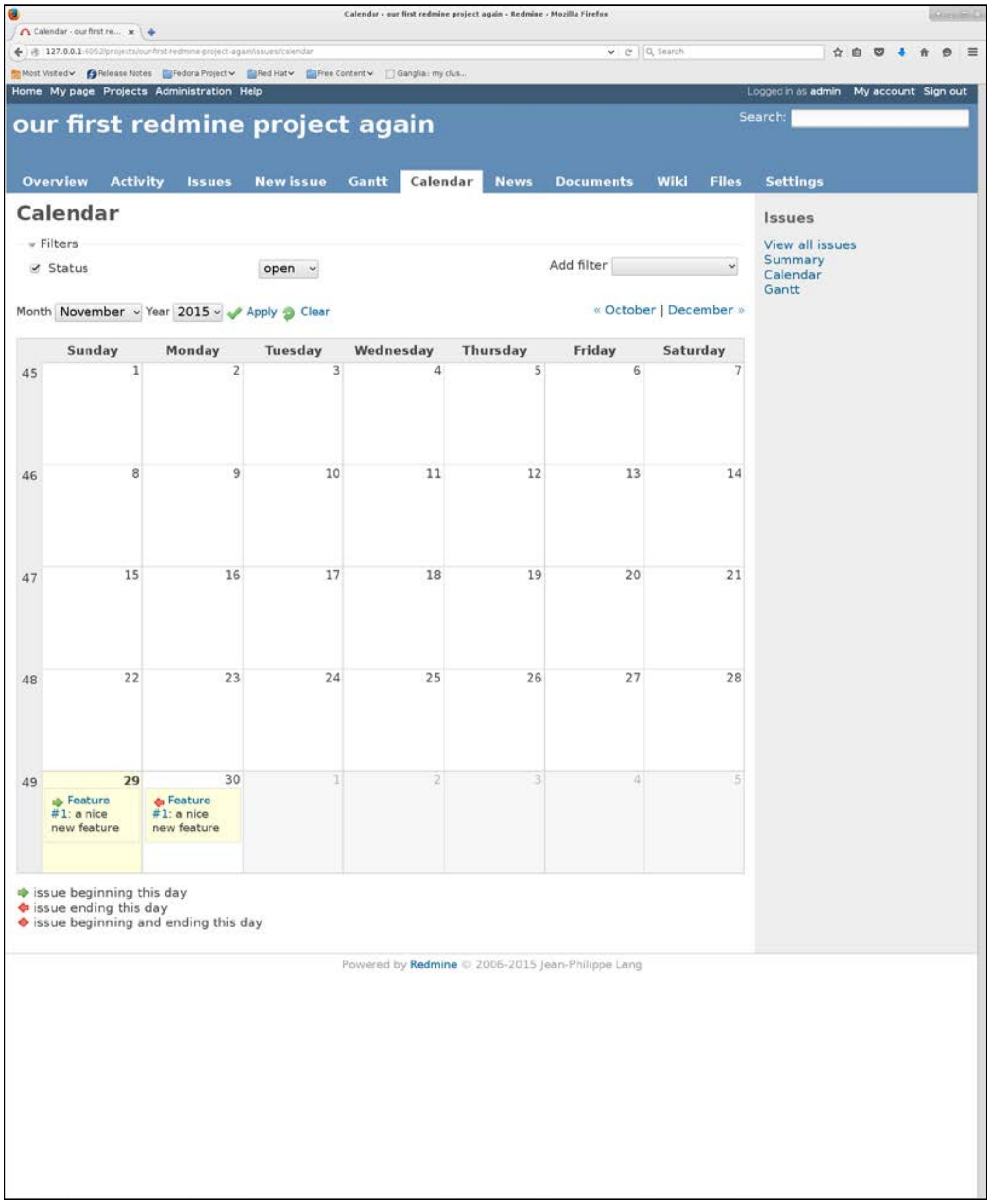
Tracker *	Feature
Private	☐
Subject *	a nice new feature
Description	our customers will love this feature
Status *	New
Priority *	Normal
Assignee	
Parent task	
Start date	2015-11-29
Due date	2015-11-30
Estimated time	Hours
% Done	0 %
Files	No files selected. (Maximum size: 5 MB)
Watchers	

There is also a **Gantt** view and a **Calendar** view, among many other features.

Here is the **Gantt** view:


Home My page Projects Administration Help
Logged in as admin My account Sign out
our first redmine project again
Search:
Overview Activity Issues New issue Gantt Calendar News Documents Wiki Files Settings
Gantt
Filters
Status open
Add filter
Options
6 months from November 2015 Apply Clear Save
Zoom in Zoom out
2015-11 2015-12 2016-1
45 46 47 48 49 50 51 52 53 1 2 3 4
our first redmine project again
Feature #1: a nice new feature
New 0%
« Previous
Next »
Also available in: PDF | PNG
Issues
View all issues
Summary
Calendar
Gantt
Powered by Redmine © 2006-2015 Jean-Philippe Lang

And here is the **Calendar** view:


Calendar - our first redmine project again - Redmine - Mozilla Firefox
Home My page Projects Administration Help
Logged in as admin My account Sign out
our first redmine project again
Search:
Overview Activity Issues New issue Gantt Calendar News Documents Wiki Files Settings
Issues
View all issues
Summary
Calendar
Gantt
Apply Clear
« October | December »
Powered by Redmine © 2006-2015 Jean-Philippe Lang

Calendar

Filters

Status: open

Month: November Year: 2015

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
45	1	2	3	4	5	6	7
46	8	9	10	11	12	13	14
47	15	16	17	18	19	20	21
48	22	23	24	25	26	27	28
49	29 Feature #1: a nice new feature	30 Feature #1: a nice new feature	1	2	3	4	5

- issue beginning this day
- issue ending this day
- issue beginning and ending this day

A Redmine issue has the following default state machine:

- New
- In Progress
- Resolved
- Feedback
- Closed
- Rejected

In conclusion, Redmine is a nice tracker that has many nice features and builds on the experience from Trac.

The GitLab issue tracker

GitLab's issue tracker is, as one might expect, very well integrated with the Git repository management system. GitLab's issue tracker is nice looking but is not, at the time of writing, very flexible. It might be deemed good enough for teams that prefer simplicity in their tools.

GitLab has a quick development pace, so the feature set might be expected to change. The GitLab issue tracker tracks issues per repository, so if you have many repositories, it might be difficult to get an overview of all the issues at the same time.

The suggested solution for this concern is creating a separate project where issues concerning several repositories are gathered.

While the GitLab issue tracker lacks many of the features seen in competing systems, it does have a nice, flexible API, and there is a command-line client for the API, so you can try it out from the convenience of your shell.

Testing the GitLab API is quite easy:

First, install the GitLab CLI

Then, set the following two environment variables that describe the endpoint of the GitLab API and the authorization token:

- `GITLAB_API_PRIVATE_TOKEN` = <*token from your project*>
- `GITLAB_API_ENDPOINT` = `http://gitlab.matangle.com:50003/api/v3`

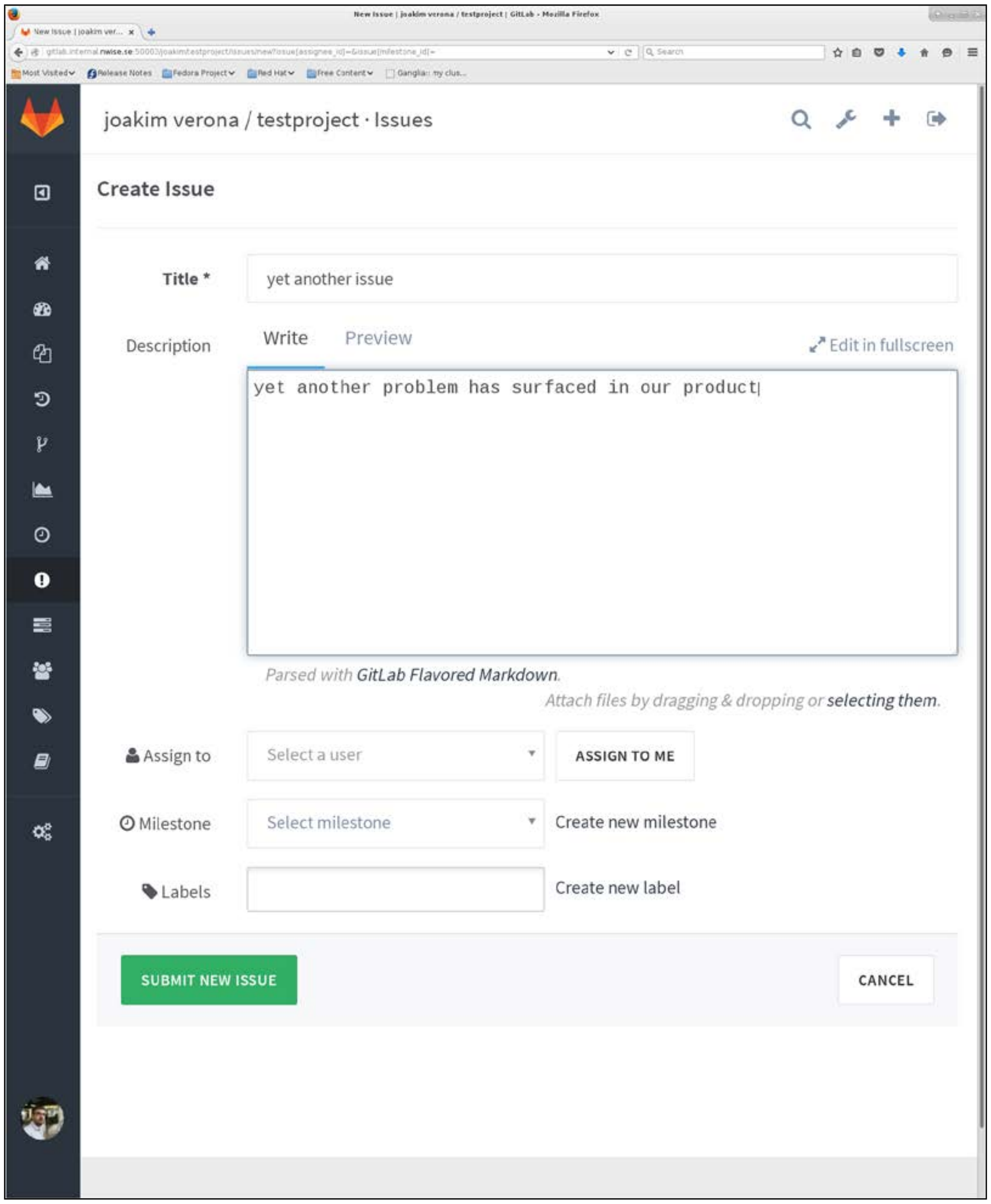
Now you can list issues and so on. There is an inbuilt help system:

```
  gitlab help Issues
+--------------+
|    Issues    |
+--------------+
| close_issue  |
+--------------+
| create_issue |
+--------------+
| edit_issue   |
+--------------+
| issue        |
+--------------+
| issues       |
+--------------+
| reopen_issue |
+--------------+
```

GitLab issues only have two conventional states: open and closed. This is pretty spartan, but the idea is to instead use other types of metadata, such as labels, to describe issues. A GitLab issue tracker label is made up of text and a background color that can be associated with an issue. More than one label can be associated.

There is also the possibility of embedding a to-do list in the Markdown format in the issue description. While labels and to-do lists can together be used to create flexible views, this does not really allow for the traditional use case of issue tracker state machines. You have to remember what you need to do yourself: the GitLab issue tracker does not help you remember process flows. Small teams might find the lack of ceremony in the GitLab tracker refreshing, and it's certainly easy to set up once you have already set up GitLab—you get it installed already.

The user interface is quite intuitive. Here is the **Create Issue** view:


New Issue | joakim verona / testproject | GitLab - Mozilla Firefox
joakim verona / testproject · Issues
Create Issue
Title *
yet another issue
Description
Write
Preview
Edit in fullscreen
yet another problem has surfaced in our product
Parsed with GitLab Flavored Markdown.
Attach files by dragging & dropping or selecting them.
Assign to
Select a user
ASSIGN TO ME
Milestone
Select milestone
Create new milestone
Labels
Create new label
SUBMIT NEW ISSUE
CANCEL

Here are some of the features of the interface:

- **Assign to**: People who can be assigned as members of the project.
- **Labels**: You can have a default set of labels, create your own, or mix.
  Here are the default labels:
    - **Bug**
    - **Confirmed**
    - **Critical**
    - **Discussion**
    - **Documentation**
    - **Enhancement**
    - **Suggestion**
    - **Support**

Here are some of the attributes that can be used for an issue:

- **Milestones**: You can group issues together in milestones. The milestones describe when issues are to be closed in a timeline fashion.
- **GitLab Flavored Markdown support**: GitLab supports Markdown and some extra markup that makes linking together different entities in GitLab easier.
- **Attach files**

After the complexity of the Bugzilla, Trac, and Redmine interfaces, the no-nonsense approach of the GitLab issue tracker can feel quite refreshing!

Jira

Jira is a flexible bug tracker by Atlassian, written in Java. It is the only proprietary issue tracker reviewed here.

While I prefers **FLOSS** (**Free/Libre/Open Source Software**), Jira is very adaptable and free of charge for small teams of up to 15 people. Unlike Trac and GitLab, the different modules such as the wiki and repository viewer are separate products. If you need a Wiki, you deploy one separately, and Atlassian's own Confluence Wiki is the one that integrates with the least amount of friction. Atlassian also provides a code browser called **Fisheye** for code repository integration and browsing.

For small teams, Jira licensing is cheap, but it quickly becomes more expensive.

By default, Jira's workflow is much like the Bugzilla workflow:

- Open
- In Progress
- Resolved
- Closed
- Reopened

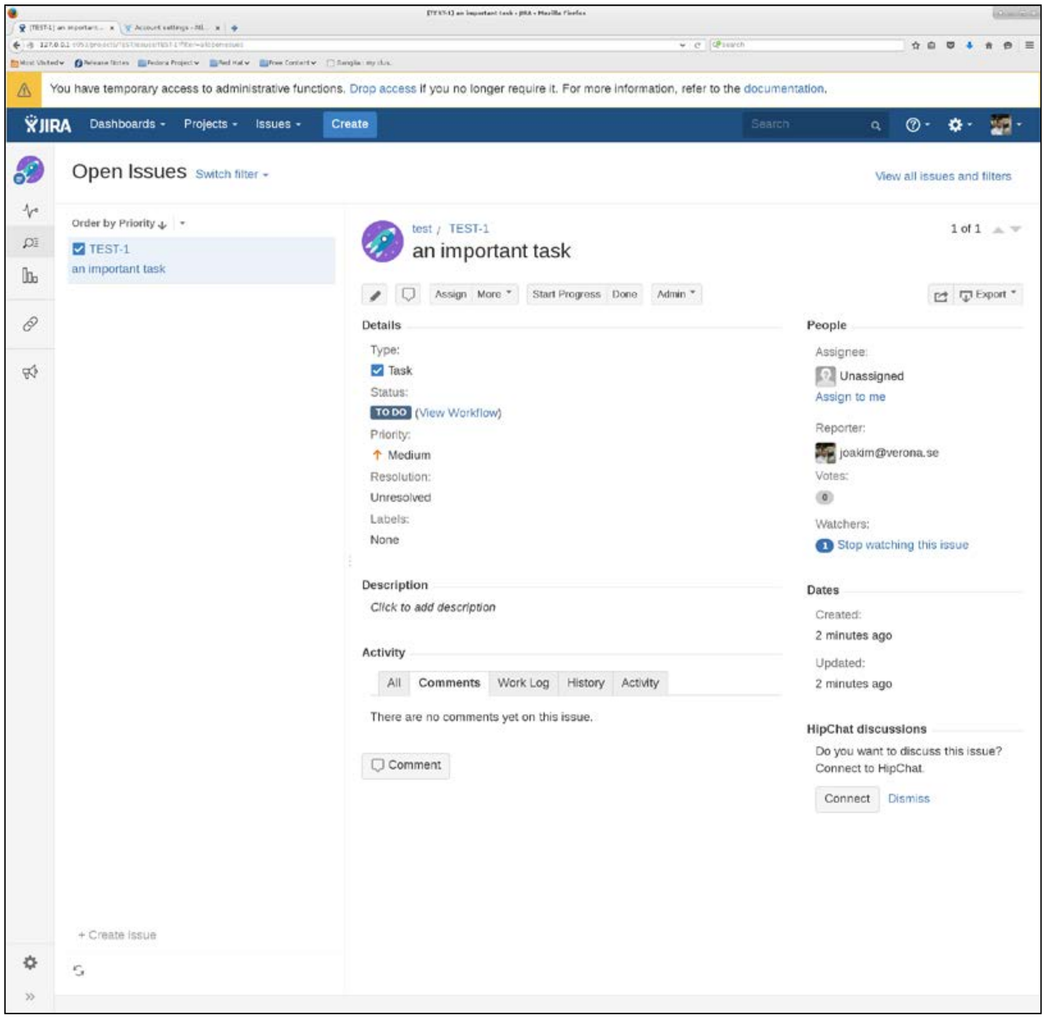
There are several Docker Hub images available; one is `cptactionhank/atlassian-jira`—but please remember, Jira is not free software.

Let's try Jira now:

```
docker run -p 6053:8080 cptactionhank/atlassian-jira:latest
```

You can access the interface on `port 6053`. Jira is somewhat more convoluted to try out, because you need to create an Atlassian account before starting. Log in with your account when you are done.

Jira starts out with a simple tutorial, where you create a project and an issue in the project. If you follow along, the resulting view will look like this:



You have temporary access to administrative functions. Drop access if you no longer require it. For more information, refer to the documentation.

Open Issues

TEST-1
an important task

test / TEST-1
an important task

1 of 1

Details

Type:
Task
Status:
TO DO (View Workflow)
Priority:
Medium
Resolution:
Unresolved
Labels:
None

Description
Click to add description

Activity

There are no comments yet on this issue.

People

Assignee:
Unassigned
Assign to me

Reporter:
joakim@verona.se

Votes:
0

Watchers:
1 Stop watching this issue

Dates

Created:
2 minutes ago

Updated:
2 minutes ago

HipChat discussions

Do you want to discuss this issue? Connect to HipChat.

Jira has many features, many of them available through plugins available in an inbuilt app store. If you can live with Jira not being free software, it is probably the issue tracker with the glossiest look of those reviewed here. It is also fairly complex to configure and administer.

Summary

In this chapter, we looked at various systems that can implement issue tracking workflow support in our organization. As usual, there are many solutions to choose from, especially in this area, where the tools are of the type that are used every day.

The next chapter will deal with something slightly more esoteric: DevOps and the Internet of Things.

10
The Internet of Things and DevOps

In the previous chapter, we explored some of the many different tool options, such as issue trackers, available to us to help us manage workflows.

This chapter will be forward looking: how can DevOps assist us in the emerging field of the Internet of Things?

The Internet of Things, or IoT for short, presents challenges for DevOps. We will explore what these challenges are.

Since IoT is not a clearly defined term, we will begin with some background.

Introducing the IoT and DevOps

The phrase **Internet of Things** was coined in the late 1990s, allegedly by the British entrepreneur Kevin Ashton, while he was working with RFID technology. Kevin became interested in using RFID to manage supply chains while working at Proctor and Gamble.

RFID, or **Radio Frequency ID**, is the technology behind the little tags you wear on your key chain and use to open doors, for instance. RFID tags are an example of interesting things that can, indirectly in this case, be connected to the Internet. RFID tags are not limited to opening doors, of course, and the form factor does not have to be limited to a tag on a key chain.

An RFID tag contains a small chip, about 2 millimeters squared, and a coil. When placed near a reader, the coil is charged with electricity by induction, and the chip is given power long enough for it to transmit a unique identifier to the reader's hardware. The reader, in turn, sends the tag's identification string to a server that decides, for example, whether it is going to open the lock associated with the reader or not. The server is likely connected to the Internet so that the identifiers associated with the RFID tags can be added or deleted remotely depending on changing circumstances regarding who has access to the locked door. Several different systems work in symbiosis to achieve the desired outcome.

Other interesting IoT technologies include passive QR codes that can be scanned by a camera and provide information to a system. The newer Bluetooth low energy technology provides intelligent active sensors. Such sensors can operate up to a year on a lithium cell.

Below, two RFID tags for operating locks are depicted:

The term "Internet of Things" is pretty vague. When is it a "thing" and when is it a computer? Or is it a little bit of both?

Let's take a small computer, such as the Raspberry Pi; it is a **System on a Chip** (**SoC**), mounted on a board the size of a credit card. It's small, but is still a full computer and powerful enough to run Java application servers, web servers, and Puppet agents.

Compared to traditional computers, IoT devices are constrained in various ways. Often, they are embedded devices that are placed in difficult-to-access, constrained locations. This, then, would seem to be the defining characteristic of the IoT from a DevOps point of view: the devices might be constrained in different regards. We cannot use every technique that we use on servers or desktop machines. The constraints could be limited memory, processing power, or access, to name a few.

Here are some example types of IoT devices, which you have probably either encountered already or soon will:

- **Smartwatches**: Today, a smartwatch can have a Bluetooth and Wi-Fi connection and automatically detect available upgrades. It can download new firmware and upgrade itself on user interaction. There are many smartwatches available today, from the robust Pebble to Android Wear and Apple Watch. There already are smartwatches with mobile network connections as well, and eventually, they will become commonplace.
- **Keyboards**: A keyboard can have upgradeable firmware. A keyboard is actually a good example of an IoT device, in a sense. It provides many sensors. It can run software that provides sensor readouts to more powerful machines that in the end are Internet-connected. There are fully programmable keyboards with open firmware, such as the Ergodox, which interfaces an Arduino-style board to a keyboard matrix (`http://ergodox.org/`).
- **Home automation systems**: These are Internet-connected and can be controlled with smartphones or desktop computers. The Telldus TellStick is an example of such a device that lets you control remote power relays via an Internet-connected device. There are many other similar systems.
- **Wireless surveillance cameras**: These can be monitored with a smartphone interface. There are many providers for different market segments, such as Axis Communications.

- **Biometric sensors**: Such sensors include fitness sensors, for example, pulse meters, body scales, and accelerometers placed on the body. The Withings body scale measures biometrics and uploads it to a server and allows you to read statistics through a website. There are accelerometers in smartwatches and phones as well that keep track of your movements.
- **Bluetooth-based key finders**: You can activate these from your smartphone if you lose your keys.
- **Car computers**: These do everything from handling media and navigation to management of physical control systems in the car, such as locks, windows, and doors.
- **Audio and video systems**: These include entertainment systems, such as networked audio players, and video streaming hardware, such as Google Chromecast and Apple TV.
- **Smartphones**: The ubiquitous smartphones are really small computers with 3G or 4G modems and Wi-Fi that connects them to the wider Internet.
- **Wi-Fi capable computers embedded in memory cards**: These make it possible to convert an existing DSLR camera into a Wi-Fi capable camera that can automatically upload images to a server.
- **Network routers in homes or offices**: These are in fact, small servers that often can be upgraded remotely from the ISP side.
- **Networked printers**: These are pretty intelligent these days and can interact with cloud services for easier printing from many different devices.

The list can, of course, go on, and all of these devices are readily available and used in many households today.

It is interesting to note that many of the devices mentioned are very similar from a technical viewpoint, even if they are used for completely different purposes. Most smartphones use variants of the ARM CPU architecture, which can be licensed across manufacturers together with a few variants of peripheral chips. MIPS processors are popular among router hardware vendors and Atmel RISC processors among embedded hardware implementers, for example.

The basic ARM chips are available to developers and hobbyists alike for easy prototyping. The Raspberry Pi is ARM-based and can be used as a prototyping device for professional use. Arduino similarly makes Atmel-architecture devices available for easy prototyping of hardware.

This makes a perfect storm of IoT innovation possible:

- The devices are cheap and easy to acquire even in small runs.
- The devices are simple to develop for and get prototyping platforms
- The development environments are similar, which makes them easier to learn. A lot of the time, the GNU Compiler Collection, or GCC, is used.
- There is a lot of support available in forums, mailing lists, documentation, and so on.

For powerful devices such as the Raspberry Pi, we can use the same methods and practices that we use for servers. Pi devices can be servers, just less powerful than a traditional server. For IoT devices, agentless deployment systems are a better fit than systems that require an agent.

Tinier devices, such as the Atmel embedded CPUs used in Arduinos, are more constrained. Typically, you compile new firmware and deploy them to the device during reboot, when special bootstrap loader code runs. The device then connects to the host via USB.

During development, one can automate the upload of firmware by connecting a separate device that resets the original device and puts it into loader mode. This might work during development, but it's not cost-effective in a real deployment scenario since it affects costs. These are the types of problems that might affect DevOps when working with the IoT. In the development environment, we might be able to use, more or less, the methods that we are used to from developing server applications, perhaps with some extra hardware. From a quality assurance perspective, though, there is a risk involved in deploying on hardware that is different to that used in testing.

The future of the IoT according to the market

According to the research group Gartner, there will be approximately 6.4 billion things connected to the Internet at the end of 2016: a 30 percent increase since 2015. Further away, at the end of the year 2020, there will be an estimated 21 billion Internet-connected devices. The consumer market will be responsible for the greatest number of devices, and enterprises for the greatest spending.

The next generation of wireless networks, which are tentatively called 5G mobile networks, is expected to reach the market in 2020, which isn't too far away at the time of writing. The upcoming mobile networks will have capabilities suitable for IoT devices and even be way ahead of today's 4G networks.

Some of the projected abilities of the new mobile network standards include:

- The ability to connect many more devices than what is possible with today's networks. This will enable the deployment of massive sensor networks with hundreds of thousands of sensors.
- Data rates of several tens of megabits per second for tens of thousand of users. In an office environment, every node will be offered a capacity of one gigabit per second.
- Very low latencies, enabling real-time interactive applications.

Regardless of how the 5G initiatives turn out in practice, it is safe to assume that mobile networks will evolve quickly, and many more devices will be connected directly to the Internet.

Let's look at what some industry giants have to say:

- We begin with Ericsson, a leading Swedish provider of mobile network hardware. The following is a quote from Ericsson's IoT website:

  > "**Ericsson and the Internet of Things**
  >
  > *More than 40 percent of global mobile traffic runs on Ericsson networks. We are one of the industry's most innovative companies with over 35,000 granted patents, and we are driving the evolution of the IoT by lowering thresholds for businesses to create new solutions powered by modern communications technology; by breaking barriers between industries; and by connecting companies, people and society. Going forward, we see two major opportunity areas:*
  >
  > **Transforming industries***: The IoT is becoming a key factor in one sector after another, enabling new types of services and applications, altering business models and creating new marketplaces.*
  >
  > **Evolving operator roles***: In the new IoT paradigm, we see three different viable roles that operators can take. An operator's choice on which roles to pursue will depend on factors like history, ambition, market preconditions and their future outlook."*

Ericsson also contributed to a nice visual dictionary of IoT concepts, which they refer to as the "comic book":
`http://www.alexandra.dk/uk/services/Publications/Documents/IoT_Comic_Book.pdf`

- The networking giant Cisco estimates that the Internet of Things will consist of 50 billion devices connected to the Internet by 2020.

  Cisco prefer to use the term "Internet of Everything" rather than Internet of Things. The company envisions many interesting applications not previously mentioned here. Some of these are:

  - Smart trash bins, where embedded sensors allow the remote sensing of how full a bin is. The logistics of waste management can then be improved.
  - Parking meters that change rates based on the demand.
  - Clothes that notify the user if he or she gets sick.

  All of these systems can be connected and managed together.

- IBM estimate, somewhat more conservatively perhaps, 26 billion devices connected to the Internet by 2020. They provide some examples of applications that are already deployed or in the works:

  - Improved logistics at car manufacturers' using wide sensor net deployments
  - Smarter logistics in health care situations in hospitals
  - Again, smarter logistics and more accurate predictions regarding train travel

Different market players have slightly different predictions for the coming years. At any rate, it seems clear that the Internet of Things will grow nearly exponentially in the coming years. Many new and exciting applications and opportunities will present themselves. It's clear that many fields will be affected by this growth, including, naturally, the field of DevOps.

Machine-to-machine communication

Many of the IoT devices will primarily connect to other machines. As a scale comparison, let's assume every human has a smartphone. That's about 5 billion devices. The IoT will eventually contain at least 10 times more devices—50 billion devices. When this will happen differs a bit among the predictions, but we will get there in the end.

One of the driving forces of this growth is machine-to-machine communication.

Factories are already moving to a greater degree of automation, and this tendency will only grow as more and more options become available. Logistics inside factories can increase greatly with wide sensor net deployments and big data analytics for continuously improving processes.

Modern cars use processors for nearly every possible function, from lights and dashboard functions to window lifts. The next step will be connecting cars to the Internet and, eventually, self-driving cars that will communicate all sensor data to centralized coordinating servers.

Many forms of travel can benefit from the IoT.

IoT deployment affects software architecture

An IoT network consists of many devices, which might not be at the same firmware revision. Upgrades might be spread out in time because the hardware might not be physically available and so on. This makes compatibility at the interface level important. Since small networked sensors might be memory and processor constrained, versioned binary protocols or simple REST protocols may be preferred. Versioned protocols are also useful in order to allow things with different hardware revisions to communicate at different versioned end points.

Massive sensor deployments can benefit from less talkative protocols and layered message queuing architectures to handle events asynchronously.

IoT deployment security

Security is a difficult subject, and having lots of devices that are Internet-connected rather that on a private network does not make the situation easier. Many consumer hardware devices, such as routers, have interfaces that are intended to be used for upgrades but are also easy to exploit for crackers. A legitimate service facility thus becomes a backdoor. Increasing the available surface increases the number of potential attack vectors.

Perhaps you recognize some of these anti-patterns from development:

- A developer leaves a way in the code to enable him or her to later submit code that will be evaluated in the server application context. The idea is that you as a developer don't really know what kind of hot fixes will be necessary and whether an operator will be available when the fix needs to be deployed. So why not leave a "backdoor" in the code so that we can deploy code directly if needed? There are many problems here, of course. The developers don't feel that the usual agreed-upon deployment process is efficient enough, and as an end result, crackers could probably easily figure out how to use the backdoor as well. This anti-pattern is more common than one might think, and the only real remedy is code review.

  Leaving open doors for crackers is never good, and you can imagine the calamity if it was possible to exploit a backdoor in a self-driving car or a heat plant, for instance.

- Unintended exploits, such as SQL injection, mostly occur because developers might not be aware of the problem.

  The remedy is having knowledge about the issue and coding in such a way as to avoid the problem.

Okay, but what about DevOps and the IoT again?

Let's take a step back. So far, we have discussed the basics of the Internet of Things, which is basically our ordinary Internet but with many more nodes than what we might normally consider possible. We have also seen that in the next couple of years, the number of devices with Internet capability in some form or another will keep on growing exponentially. Much of this growth will be in the machine-to-machine parts of the Internet.

But is DevOps, with its focus on fast deliveries, really the right fit for large networks of critical embedded devices?

The classic counterexamples would be DevOps in a nuclear facility or in medical equipment such as pacemakers. But just making faster releases isn't the core idea of DevOps. It's to make faster, more correct releases by bringing people working with different disciplines closer together.

This means bringing production-like testing environments closer to the developers and the people working with them closer together as well.

Described like this, it really does seem like DevOps can be of use for traditionally conservative industries.

The challenges shouldn't be underestimated though:

- The life cycles of embedded hardware devices can be longer than those of traditional client-server computers. Consumers can't be expected to upgrade during every product cycle. Likewise, industrial equipment might be deployed in places that make them expensive to change.
- There are more modes of failure for IoT devices than desktop computers. This makes testing harder.
- In industrial and corporate sectors, traceability and auditability are important. This is the same as for deployments on servers, for instance, but there are many more IoT endpoints than there are servers.
- In traditional DevOps, we can work with small changesets and deploy them to a subset of our users. If the change somehow doesn't work, we can make a fix and redeploy. If a web page renders badly for a known subset of our users and can be fixed quickly, there is only a small potential risk involved. On the other hand, if even a single IoT device controlling something such as a door or an industrial robot fails, the consequences can be devastating.

There are great challenges for DevOps in the IoT field, but the alternatives aren't necessarily better. DevOps is also a toolbox, and you always need to think about whether the tool you pick out of the box really is the right one for the job at hand.

We can still use many of the tools in the DevOps toolbox; we just need to make sure we are doing the right thing and not just implementing ideas without understanding them.

Here are some suggestions:

- Failures and fast turnarounds are okay as long as you are in your testing lab
- Make sure your testing lab is production-like
- Don't just have the latest versions in your lab; accommodate older versions as well

A hands-on lab with an IoT device for DevOps

So far, we have mostly discussed in the abstract sense about DevOps and the IoT and the future of the IoT.

To get a feel for what we can do in hands-on terms, let's make a simple IoT device that connects to a Jenkins server and presents a build status display. This way, we get to try out an IoT device and combine it with our DevOps focus!

The status display will just present a blinking LED in the event that a build fails. The project is simple but can be expanded upon by the creative reader. The IoT device selected for this exercise is quite versatile and can do much more than make an LED blink!

The project will help to illustrate some of the possibilities as well as challenges of the Internet of Things.

The **NodeMCU Amica** is a small programmable device based on the ESP8266 chip from Espressif. Apart from the base ESP8266 chip, the Amica board has added features that make development easier.

Here are some of the specifications of the design:

- There is a 32-bit RISC CPU, the Tensilica Xtensa LX106, running at 80 MHZ.
- It has a Wi-Fi chip that will allow it to connect to our network and our Jenkins server.
- The NodeMCU Amica board has a USB socket to program the firmware and connect a power adapter. The ESP8266 chip needs a USB-to-serial adapter to be connected to the USB interface, and this is provided on the NodeMCU board.
- The board has several input/output ports that can be connected to some kind of hardware to visualize the build status. Initially, we will keep it simple and just use the onboard LED that is connected to one of the ports on the device.
- The NodeMCU contains default firmware that allows it to be programmed in the Lua language. Lua is a high-level language that allows for rapid prototyping. Incidentally, it is popular in game programming, which might offer a hint about Lua's efficiency.

- The device is fairly cheap, considering the many features it offers:

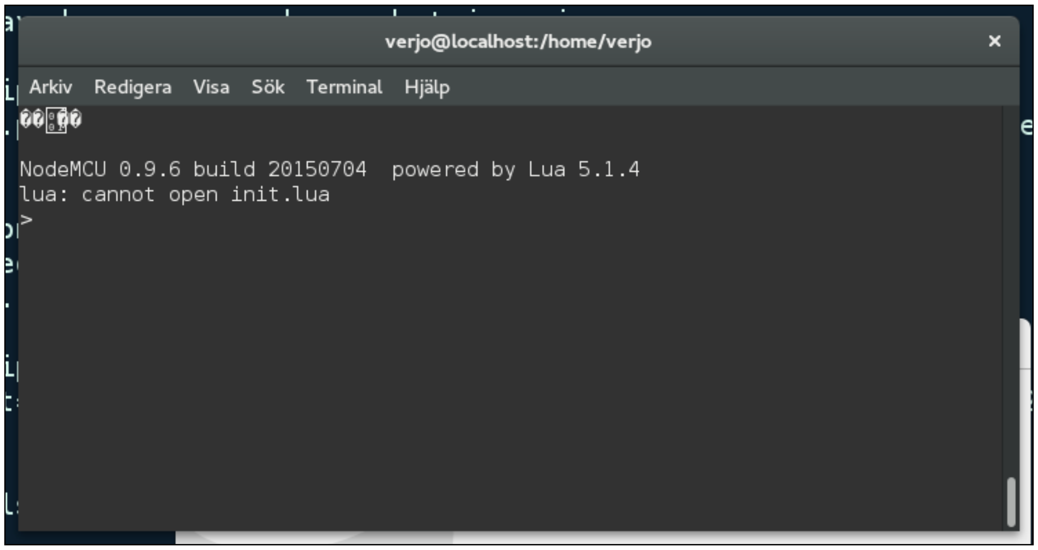
There are many options to source the NodeMCU Amica, both from electronics hobbyist stores and Internet resellers.

While the NodeMCU is not difficult to source, the project is fairly simple from a hardware point of view and might also be undertaken with an Arduino or a Raspberry Pi in practice if those devices turn out to be simpler to gain access to.

Here are some hints for getting started with the NodeMCU:

- The NodeMCU contains firmware that provides an interactive Lua interpreter that can be accessed over a serial port. You can develop code directly over the serial line. Install serial communication software on your development machine. There are many options, such as Minicom for Linux and Putty for Windows.
- Use the serial settings 9600 baud, eight bits, no parity, and one stop bit. This is usually abbreviated to 9600 8N1.
- Now that we have a serial terminal connection, connect the NodeMCU to a USB port, switch to the terminal, and verify that you get a prompt in the terminal window.

If you are using Minicom, the window will look like this:


verjo@localhost:/home/verjo
Arkiv Redigera Visa Sök Terminal Hjälp

```
NodeMCU 0.9.6 build 20150704  powered by Lua 5.1.4
lua: cannot open init.lua
>
```

Before starting with the code, depending on how your particular NodeMCU was set up at the factory, it might be required to burn a firmware image to the device. If you get a prompt in the previous step, you don't need to burn the firmware image. You might then want to do it later if you require more features in your image.

To burn a new firmware image, if needed, first download it from the firmware source repository releases link. The releases are provided at `https://github.com/nodemcu/nodemcu-firmware/releases`

Here is an example `wget` command to download the firmware. The released firmware versions are available in integer and float flavor depending on your needs when it comes to mathematical functions. The integer-based firmware versions are normally sufficient for embedded applications:

```
wget https://github.com/nodemcu/nodemcu-firmware/releases/
download/0.9.6-dev_20150704/nodemcu_integer_0.9.6-dev_20150704.bin
```

You can also build the firmware image directly from the sources on GitHub locally on your development machine, or you can use an online build service that builds a firmware for you according to your own specifications.

The online build service is at `http://nodemcu-build.com/`. It is worth checking out. If nothing else, the build statistics graphs are quite intriguing.

Now that you have acquired a suitable firmware file, you need to install a firmware burning utility so that the firmware image file can be uploaded to the NodeMCU:

```
git clone https://github.com/themadinventor/esptool.git
```

Follow the installation instructions in the repository's `README` file.

If you'd rather not do the system-wide installation suggested in the `README`, you can install the `pyserial` dependency from your distribution and run the utility from the git-clone directory.

Here's an example command to install the `pyserial` dependency:

```
sudo dnf install pyserial
```

The actual firmware upload command takes a while to complete, but a progress bar is displayed so that you know what is going on.

The following command line is an example of how to upload the 0.9.6 firmware that was current at the time of writing:

```
sudo python ./esptool.py --port /dev/ttyUSB0 write_flash 0x00000 nodemcu_
integer_0.9.6-dev_20150704.bin
```

If you get gibberish on the serial console while connecting the NodeMCU, you might need to provide additional arguments to the firmware burn command:

```
sudo esptool.py --port=/dev/ttyUSB0 write_flash 0x0 nodemcu_
integer_0.9.6-dev_20150704.bin  -fs 32m -fm dio -ff 40m
```

The `esptool` command also has some other functionalities that can be used to validate the setup:

```
sudo ./esptool.py read_mac

Connecting...

MAC: 18:fe:34:00:d7:21
```

```
sudo ./esptool.py flash_id
Connecting...
Manufacturer: e0
Device: 4016
```

After uploading the firmware, reset the NodeMCU.

At this point, you should have a serial terminal with the NodeMCU greeting prompt. You achieve this state either using the factory-provided NodeMCU firmware or uploading a new firmware version to the device.

Now, let's try some "hello world" style exercises to begin with.

Initially, we will just blink the LED that is connected to the GPIO pin 0 of the NodeMCU Amica board. If you have another type of board, you need to figure out whether it has an LED and to which input/output pin it is connected in case it does. You can, of course, also wire an LED yourself.

Note that some variants of the board have the LED wired to the GPIO pin 3 rather than pin 0 as is assumed here.

You can either upload the program as a file to your NodeMCU if your terminal software allows it, or you can type in the code directly into the terminal.

The documentation for the NodeMCU library is available at `http://www.nodemcu.com/docs/` and provides many examples of usage of the functions.

You can first try to light the LED:

```
gpio.write(0, gpio.LOW)  -- turn led on
```

Then, turn off the LED using the following:

```
gpio.write(0, gpio.HIGH) -- turn led off
```

Now, you can loop the statements with some delays interspersed:

```
while 1 do                     -- loop forever
      gpio.write(0, gpio.HIGH) -- turn led off
      tmr.delay(1000000)       -- wait one second
      gpio.write(0, gpio.LOW)  -- turn led on
      tmr.delay(1000000)       -- wait one second
end
```

At this point, you should be able to verify a basic working setup. Typing code directly into a terminal is somewhat primitive though.

There are many different development environments for the NodeMCU that improve the development experience.

> I have a penchant for the Emacs editor and have used the NodeMCU Emacs mode. This mode, NodeMCU-mode, can be downloaded from GitHub. Emacs has an inbuilt facility to make serial connections. The reader should, of course, use the environment he or she feels most comfortable with.

We need some additional hints before being able to complete the lab.

To connect to a wireless network, use the following:

```
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
```

`SSID` and `password` need to be replaced with the appropriate strings for your network.

If the NodeMCU connects properly to your wireless network, this command will print the IP address it acquired from the network's `dhcpd` server:

```
print(wifi.sta.getip())
```

This snippet will connect to the HTTP server at `www.nodemcu.com` and print a return code:

```
conn=net.createConnection(net.TCP, false)
conn:on("receive", function(conn, pl) print(pl) end)
conn:connect(80,"121.41.33.127")
conn:send("GET / HTTP/1.1\r\nHost: www.nodemcu.com\r\n"
    .."Connection: keep-alive\r\nAccept: */*\r\n\r\n")
```

You might also need a timer function. This example prints `hello world` every 1000 ms:

```
tmr.alarm(1, 1000, 1, function()
    print("hello world")
end )
```

Here, Lua's functional paradigm shows through since we are declaring an anonymous function and send it as an argument to the timer function. The anonymous function will be called every 1000 milliseconds, which is every second.

To stop the timer, you can type:

```
tmr.stop(1)
```

Now, you should have all the bits and pieces to complete the labs on your own. If you get stuck, you can refer to the code in the book's source code bundle. Happy hacking!

Summary

In this final chapter, we learned about the emerging field of the Internet of Things and how it affects DevOps. Apart from an overview of the IoT, we also made a hardware device that connects to a build server and presents a build status.

The idea of going from the abstract to the concrete with practical examples and then back again to the abstract has been a running theme in this book.

In *Chapter 1, Introduction to DevOps and Continuous Delivery*, we learned about the background of DevOps and its origin in the world of Agile development.

In *Chapter 2, A View from Orbit*, we studied different aspects of a Continuous Delivery pipeline.

Chapter 3, How DevOps Affects Architecture, delved into the field of software architecture and how the ideas of DevOps might affect it.

In *Chapter 4, Everything is Code*, we explored how a development organization can choose to handle its vital asset—source code.

Chapter 5, Building the Code, introduced the concept of build systems, such as Make and Jenkins. We explored their role in a Continuous Delivery pipeline.

After we have built the code, we need to test it. This is essential for executing effective, trouble-free releases. We had a look at some of the testing options available in *Chapter 6, Testing the Code*.

In *Chapter 7*, *Deploying the Code*, we explored the many options available to finally deploy our built and tested code to servers.

When we have our code running, we need to keep it running. *Chapter 8*, *Monitoring the Code*, examined the ways in which we can ensure our code is running happily.

Chapter 9, *Issue Tracking*, dealt with some of the many different issue trackers available that can help us deal with the complexities of keeping track of development flows.

This is the last chapter in this book, and it has been a long journey!

I hope you enjoyed the trip as much as I have, and I wish you success in your further explorations in the vast field of DevOps!

Module 2

DevOps Automation Cookbook

Over 120 recipes covering key automation techniques through code management and virtualization offered by modern Infrastructure-as-a-Service solutions

1
Basic Command Line Tools

In this chapter, we will cover the following:

- Controlling network interfaces
- Monitoring network details with the IP command
- Monitoring connections using the ss command
- Gathering basic OS statistics
- Viewing historical resource usage with SAR
- Installing and configuring a Git client
- Creating an SSH key for Git
- Using `ssh-copy-id` to copy keys
- Creating a new Git repository
- Cloning an existing Git repository
- Checking changes into a Git repository
- Pushing changes to a Git remote
- Creating a Git branch

Introduction

Every Linux System Administrator should have a solid grasp of command-line tools, from the very basics of navigating the file system to the ability to run diagnostic tools to examine potential issues. The command line offers unparalleled power and flexibility along with the ability to chain together commands to form powerful one-line scripts. Although it's quicker to pick up and use the GUI tools as compared to their command line equivalents, few offer the combination of concision and power that a well used combination of command-line tools can bring.

For system administrators who utilize DevOps techniques, the command line offers the first step on the road to automation and offers powerful abilities that can be leveraged with full stack automation. Ansible, Puppet, and Chef are powerful tools, but sometimes it is easier to write a small bash script to undertake a task rather than writing a custom function within a configuration management tool. Despite automation, the command line will be a place where you will spend the majority of your time, and remember that no matter how attractive a point and click tool is, it's highly unlikely that you can automate it.

> Most operating systems have a command line, even if they are traditionally seen as the domain of the GUI. For instance, Windows users have the option of using the excellent PowersShell tool to both administer and control Windows servers.

In this chapter, we are going to cover some useful recipes that can help DevOps engineers in their day-to-day lives. These commands will cover a wide variety of topics, covering elements such as basic networking commands, performance metrics, and perhaps the most important of all, the basics of using the Git **Distributed Version Control Software** (**DVCS**). Depending on where you approach the DevOps role from, you may find that some of this chapter touches topics that you have already covered in depth or instance, seasoned Systems Administrators will find the items on the Net tools and system performance are familiar ground; however, these can be valuable introductions for a developer. Likewise, developers will probably find the section on Git to be nothing new, while, many Systems administrators may not be used to version control systems and will benefit hugely from the items in this section.

Controlling network interfaces

Networking is one of the core elements of server management, and can be one of the more complex to manage. This recipe will show you how to use the IP tool to discover the details of, and make changes to, the networking setup of your server.

Although these are ephemeral changes, the ability to apply them at the command line is very powerful; for instance, it allows you to script the addition and removal of IP addresses. When looking at command-line tools, it's a good idea to think of not only how they can help you now, but also how they could be used for automation in future.

Getting ready

The IP tools come preinstalled on the major Linux distributions (RHEL and Debian based), so additional configuration is no longer required.

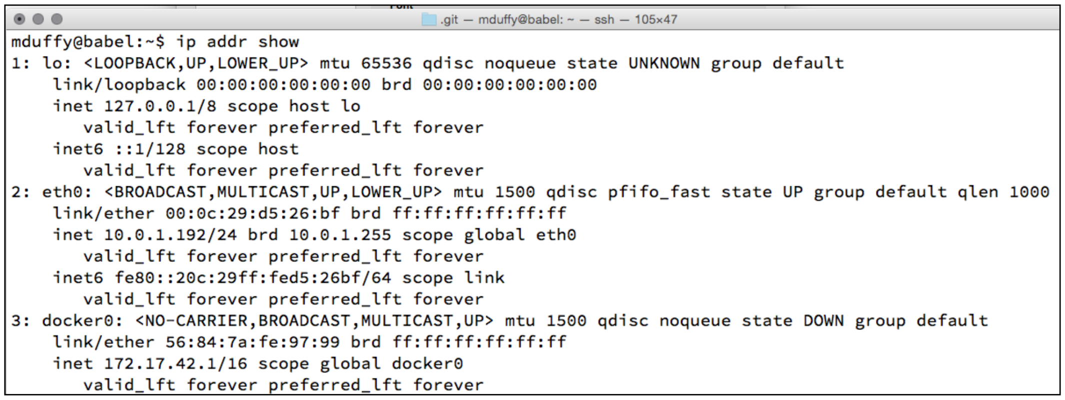
How to do it...

Let's configure the network interface as follows:

1. Sometimes you just want to know the IP address of the server you are working on; this can be found using the following command:

   `$ ip addr show`

   This should give you an output similar to to the following screenshot:



```
mdaffy@babel:~$ ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:d5:26:bf brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.192/24 brd 10.0.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fed5:26bf/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0
       valid_lft forever preferred_lft forever
```

   This output gives you the details of each interface, including the IP address, state, MAC address, and interface options.

2. To narrow this down to a single interface, you can use the following command:

   `$ ip addr show <interface>`

   Where `<interface>` is the name of the network interface you wish to see the details for. So, for example, to see the details of `eth0` you can use the following command:

   `$ ip addr show eth0`

3. To add a new IP4 address, you can use the `ip addr add` command. Note that you also need to supply a netmask in **Classless Inter-Domain Routing** (**CIDR**) format. The command looks like this:

   ```
   $ ip addr add  <IP address>/<CIDR> dev <device>
   ```

   For example, to add a new `RFC1918`-compliant address to the interface named `eth1`, you will use the following command:

   ```
   $ ip addr add 10.132.1.1/24 dev eth1
   ```

4. You can confirm that the new address is available on the interface using the `ip addr show` command:

   ```
   $ ip addr show eth1
   ```

5. Removing an IP address is a straightforward reversal of adding it. The command is the same as the command used to add addresses, but with the delete option:

   ```
   $ ip addr del <IP address>/<CIDR> dev <device>
   ```

6. So, to remove the address that we just added, we can use the following command:

   ```
   $ ip addr del 10.132.1.1/24 dev eth1
   ```

   The IP command can be used to control the physical interfaces; thus, allowing you to bring them up and down from the command line.

   It goes without saying that you need to be careful when using this command. Removing the only available interface on a remote server is both possible, and if you don't have access to a remote console, extremely inconvenient.

7. To bring an interface `down`, you can use the `ip link` set command, followed by the interface name, and then the desired state. For example, you can use the following command to enable the `eth2` interface:

   ```
   $ ip link set eth2 down
   ```

   Conversely, to bring it back up, you can use the following command:

   ```
   $ ip link set eth2 up
   ```

8. You can check the state of the interface using the `ip` command:

   ```
   $ ip addr eth2
   ```

See also

You can find further details on the `ip` command within its `man` pages. You can access these using the following command:

```
$ man 8 ip
```

Monitoring network details with the IP command

As well as allowing you to set your network interfaces, the `IP` command can also be used to check if they are functioning correctly. One of the first places to look when trying to figure out the main reason for any issues is the networking stack.

The following recipe will run you through how to use the `IP` command to check that your network interfaces are up, and also list some basic statistics.

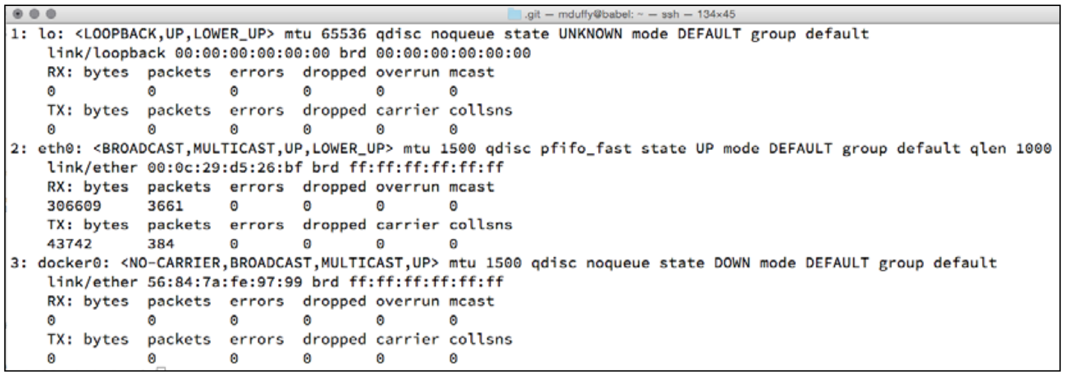
Getting ready

No additional configuration should be required as the IP tools come preinstalled in major Linux distributions (RHEL and Debian based).

How to do it...

Let's monitor network details with the following `IP` command:

1. To view basic network statistics on all interfaces, you can use the `ip -s link` command. When it is used without options, it should produce the following output:



```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:d5:26:bf brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    306609     3661     0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    43742      384      0       0       0       0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 56:84:7a:fe:97:99 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
```

This will list the interface name and its configured options; for example, if `multicast` is enabled, and its current state (`up` or `down`). The next line gives you its MAC address, and the following lines give you some interface statistics:

By default, the columns in the `ip -s link` command stand for the following:

`RX/TX: bytes`: The total number of bytes sent/received by this interface

`RX/TX: packets`: The total number of network packets sent/received by this interface

`RX/TX: errors`: The total number of transmission errors found on this interface

`RX/TX: Dropped`: Total number of dropped networking packets

`RX: mcast`: Recieved Multicast packets

`TX: collsns`: Network packet collisions

2. Sometimes, you may want to see the output only for a single interface. You can do this using the following command:

   ```
   ip -s  link ls <<interface>>
   ```

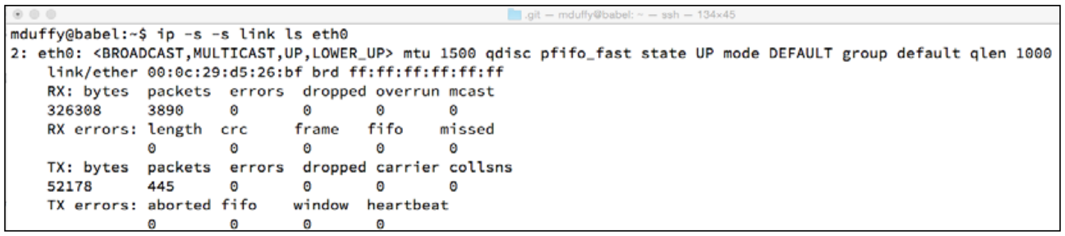
3. To see the statistics for the `eth0` interface, you can use the following command:

   ```
   ip -s link ls eth0
   ```

4. To see additional info, you can append an additional `-s` switch to the following command:

   ```
   ip -s -s link ls eth0
   ```

   This produces the following output:



```
mduffy@babel:~$ ip -s -s link ls eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode DEFAULT group default qlen 1000
    link/ether 00:0c:29:d5:26:bf brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    326308     3890     0       0       0       0
    RX errors: length   crc     frame   fifo    missed
               0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    52178      445      0       0       0       0
    TX errors: aborted  fifo    window  heartbeat
               0        0       0       0
```

   This expands on the previous listing and allows you to see the details of network errors.

Monitoring connections using the ss command

Alongside the `IP` command, we also have the `ss` command (Socket Statistics). This command is the perfect replacement for the `netstat` command and offers more functionality, it is also faster and gives results that are more accurate.

The following recipes offer some alternatives that should allow you to replace the venerable `netstat` command.

Getting ready

No additional configuration should be required as the IP tools come preinstalled in major Linux distributions (RHEL and Debian based).

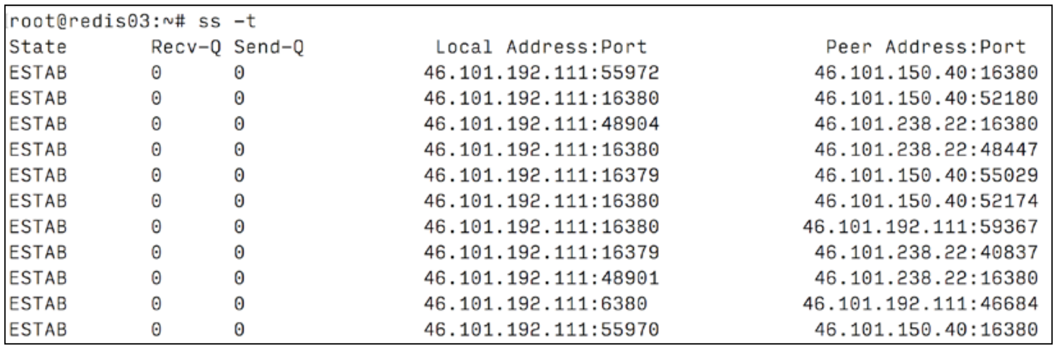
How to do it...

Let's monitor network details using the `ss` command:

1. You can use the following command to show established TCP connections:

   `ss -t`

   This should produce an output similar to the following screenshot:



```
root@redis03:~# ss -t
State       Recv-Q Send-Q          Local Address:Port              Peer Address:Port
ESTAB       0      0             46.101.192.111:55972             46.101.150.40:16380
ESTAB       0      0             46.101.192.111:16380             46.101.150.40:52180
ESTAB       0      0             46.101.192.111:48904             46.101.238.22:16380
ESTAB       0      0             46.101.192.111:16380             46.101.238.22:48447
ESTAB       0      0             46.101.192.111:16379             46.101.150.40:55029
ESTAB       0      0             46.101.192.111:16380             46.101.150.40:52174
ESTAB       0      0             46.101.192.111:16380            46.101.192.111:59367
ESTAB       0      0             46.101.192.111:16379             46.101.238.22:40837
ESTAB       0      0             46.101.192.111:48901             46.101.238.22:16380
ESTAB       0      0             46.101.192.111:6380             46.101.192.111:46684
ESTAB       0      0             46.101.192.111:55970             46.101.150.40:16380
```

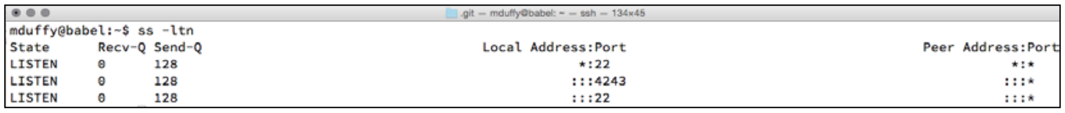
2. Alternatively, if you want to see UDP connections rather than TCP, then you can do so using the following command:

   `$ ss -u`

3. You can use the following command to see which ports are listening for connections on a server:

   `$ ss -ltn`

This displays the following listening ports in the output:



```
.git — mduffy@babel: ~ — ssh — 134x45
mduffy@babel:~$ ss -ltn
State      Recv-Q Send-Q                                  Local Address:Port                                  Peer Address:Port
LISTEN     0      128                                                 *:22                                               *:*
LISTEN     0      128                                            :::4243                                            :::*
LISTEN     0      128                                              :::22                                            :::*
```

The column we are interested in is the one titled **Local Address:Port**. This essentially lists the listening IP address and the TCP port it is listening on. If you see a *****, it means that the port is available on all interfaces configured on this server.

> The `n` in the `-ltn` option turns off `hostname` lookups. This makes the command run much faster, but you may want to omit it if you wish to see the `hostname` that an interface maps to.

4. Alternatively, you can use the same command to list all the listening UDP connections:

   ```
   $ ss -lun
   ```

   You can even combine the `t` and `u` flags to list out ALL listening ports, both UDP and TCP:

   ```
   $ ss -ltun
   ```

Gathering basic OS statistics

One of the most basic responsibilities of a DevOps engineer is to know how the various server instances under their control are performing. It forms a key part of DevOp techniques, driving infrastructure transparency and measuring the impact of changes, both prior to the change and after it has taken place.

There are many tools that are available for performance monitoring, from comprehensive **Application Performance Monitoring** (**APM**) tools through to focused monitoring for particular applications. We'll be covering these throughout the book; however, some of the best tools for basic server monitoring are already available with the operating system. Like most command-line tools, the performance monitoring tools that are shipped with the OS can be used standalone or can be chained with other commands to create complex tools.

Getting ready

The following tools are a part of the standard install of most Linux distributions and should be available with it, the exception being for `sysstat` tools, which generally need to be installed. To install `systat` tools on Ubuntu, issue the following command:

```
sudo apt-get install sysstat
```

This will make several performance-monitoring tools available, in particular `sar` and `mpstat`.

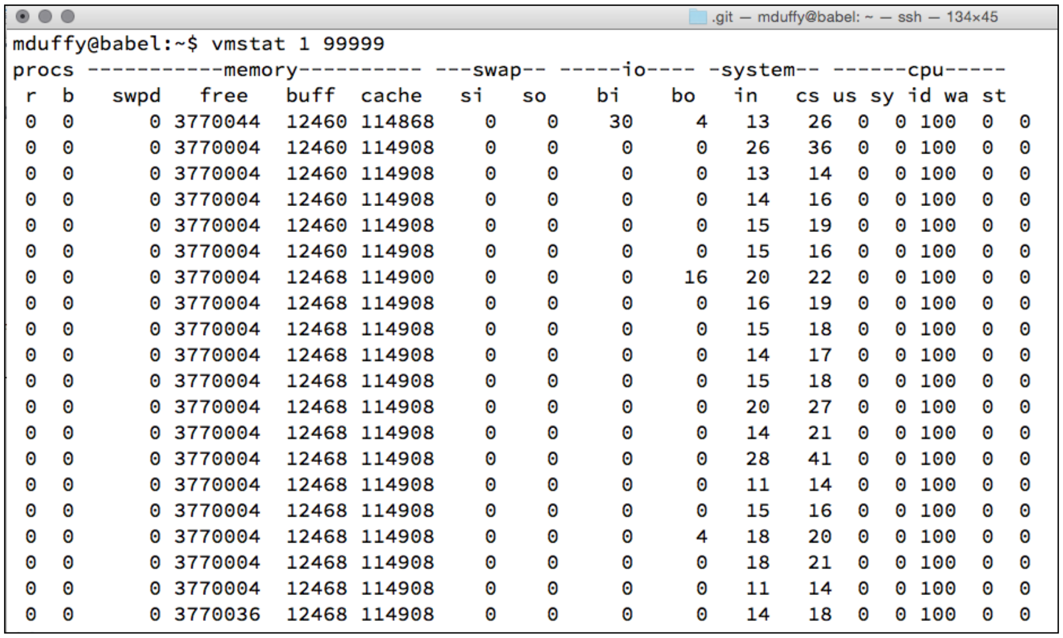
How to do it...

Let's gather basic OS statistics:

1. To gather basic information on your server, run the following command:

   ```
   vmstat 1 99999
   ```

   This should produce output similar to the following screenshot:



```
mduffy@babel:~$ vmstat 1 99999
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 3770044  12460 114868    0    0    30     4   13   26  0  0 100  0  0
 0  0      0 3770004  12460 114908    0    0     0     0   26   36  0  0 100  0  0
 0  0      0 3770004  12460 114908    0    0     0     0   13   14  0  0 100  0  0
 0  0      0 3770004  12460 114908    0    0     0     0   14   16  0  0 100  0  0
 0  0      0 3770004  12460 114908    0    0     0     0   15   19  0  0 100  0  0
 0  0      0 3770004  12460 114908    0    0     0     0   15   16  0  0 100  0  0
 0  0      0 3770004  12468 114900    0    0     0    16   20   22  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   16   19  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   15   18  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   14   17  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   15   18  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   20   27  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   14   21  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   28   41  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   11   14  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   15   16  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     4   18   20  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   18   21  0  0 100  0  0
 0  0      0 3770004  12468 114908    0    0     0     0   11   14  0  0 100  0  0
 0  0      0 3770036  12468 114908    0    0     0     0   14   18  0  0 100  0  0
```

   The command shows you the system statistics every second, for the number of times specified (`99999` in this instance).

By default, the columns in `vmstat` stand for the following:

`Procs – r`: Total number of processes waiting to run

`Procs – b`: Total number of busy processes

`Memory – swpd`: Used virtual memory

`Memory – free`: Free virtual memory

`Memory – buff`: Memory used as buffers

`Memory – cache`: Memory used as cache.

`Swap – si`: Memory swapped from disk (for every second)

`Swap – so`: Memory swapped to disk (for every second)

`IO – bi`: Blocks in (in other words) the blocks received from device (for every second)

`IO – bo`: Blocks out (in other words) the blocks sent to the device (for every second)

`System – in`: Interrupts per second

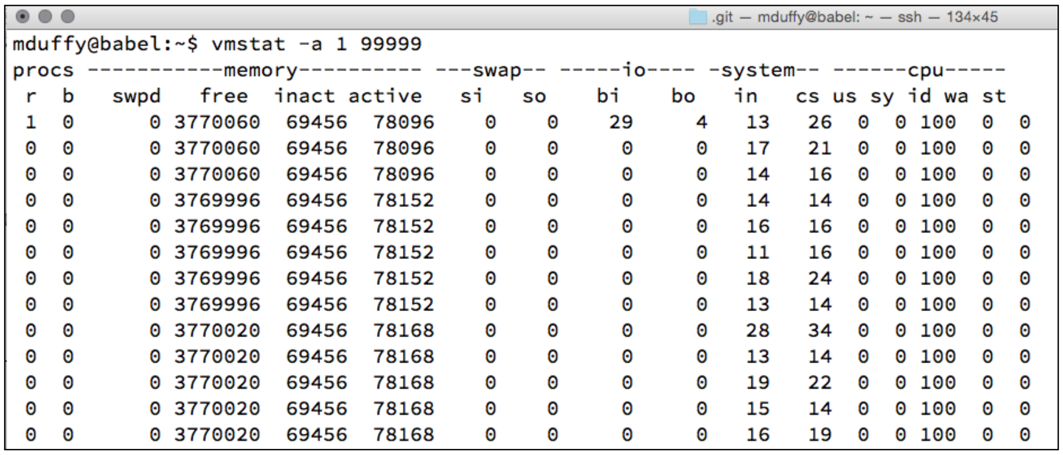
`System – cs`: Context switches

`CPU – us, sy, id, wa, st`: CPU user time, system time, idle time, and wait time

2. The `vmstat` command can also be useful to show memory usage information, particularly active and inactive memory. To show vmstat information for memory usage, you can issue the following command:

   `vmstat -a 1 999`

   This will give you an output similar to the following screenshot:



```
mduffy@babel:~$ vmstat -a 1 99999
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free  inact active   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 3770060  69456  78096    0    0    29     4   13   26  0  0 100  0  0
 0  0      0 3770060  69456  78096    0    0     0     0   17   21  0  0 100  0  0
 0  0      0 3770060  69456  78096    0    0     0     0   14   16  0  0 100  0  0
 0  0      0 3769996  69456  78152    0    0     0     0   14   14  0  0 100  0  0
 0  0      0 3769996  69456  78152    0    0     0     0   16   16  0  0 100  0  0
 0  0      0 3769996  69456  78152    0    0     0     0   11   16  0  0 100  0  0
 0  0      0 3769996  69456  78152    0    0     0     0   18   24  0  0 100  0  0
 0  0      0 3769996  69456  78152    0    0     0     0   13   14  0  0 100  0  0
 0  0      0 3770020  69456  78168    0    0     0     0   28   34  0  0 100  0  0
 0  0      0 3770020  69456  78168    0    0     0     0   13   14  0  0 100  0  0
 0  0      0 3770020  69456  78168    0    0     0     0   19   22  0  0 100  0  0
 0  0      0 3770020  69456  78168    0    0     0     0   15   14  0  0 100  0  0
 0  0      0 3770020  69456  78168    0    0     0     0   16   19  0  0 100  0  0
```

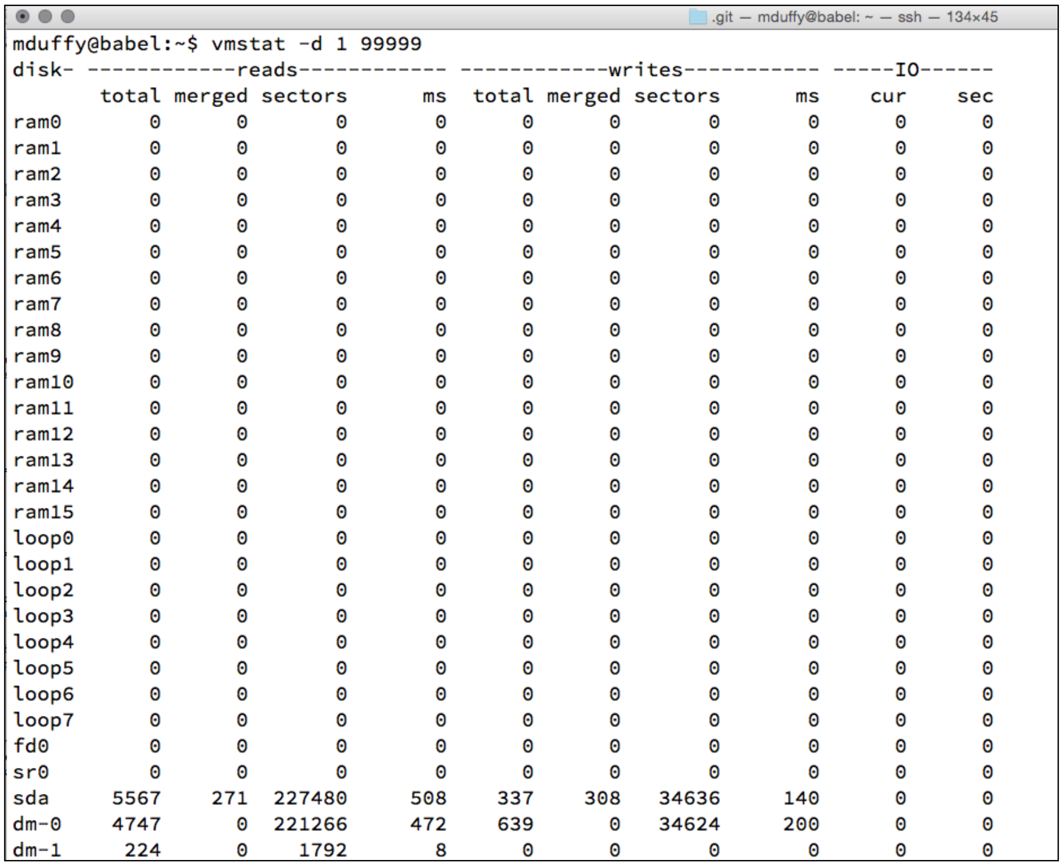
You can also reformat the output to be displayed in Mega Bytes using the following command:

```
vmstat -a -S M 1 999
```

3. The `vmstat` is not limited to gathering only CPU and RAM information; it can also be used to gather information for disks and other block devices. You can use the following command to gather basic disk statistics:

```
vmstat -d 1 99999
```

This should produce an output that looks something like the following screenshot:



```
mduffy@babel:~$ vmstat -d 1 99999
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors       ms  total merged sectors       ms    cur    sec
ram0       0      0       0        0      0      0       0        0      0      0
ram1       0      0       0        0      0      0       0        0      0      0
ram2       0      0       0        0      0      0       0        0      0      0
ram3       0      0       0        0      0      0       0        0      0      0
ram4       0      0       0        0      0      0       0        0      0      0
ram5       0      0       0        0      0      0       0        0      0      0
ram6       0      0       0        0      0      0       0        0      0      0
ram7       0      0       0        0      0      0       0        0      0      0
ram8       0      0       0        0      0      0       0        0      0      0
ram9       0      0       0        0      0      0       0        0      0      0
ram10      0      0       0        0      0      0       0        0      0      0
ram11      0      0       0        0      0      0       0        0      0      0
ram12      0      0       0        0      0      0       0        0      0      0
ram13      0      0       0        0      0      0       0        0      0      0
ram14      0      0       0        0      0      0       0        0      0      0
ram15      0      0       0        0      0      0       0        0      0      0
loop0      0      0       0        0      0      0       0        0      0      0
loop1      0      0       0        0      0      0       0        0      0      0
loop2      0      0       0        0      0      0       0        0      0      0
loop3      0      0       0        0      0      0       0        0      0      0
loop4      0      0       0        0      0      0       0        0      0      0
loop5      0      0       0        0      0      0       0        0      0      0
loop6      0      0       0        0      0      0       0        0      0      0
loop7      0      0       0        0      0      0       0        0      0      0
fd0        0      0       0        0      0      0       0        0      0      0
sr0        0      0       0        0      0      0       0        0      0      0
sda     5567    271  227480      508    337    308   34636      140      0      0
dm-0    4747      0  221266      472    639      0   34624      200      0      0
dm-1     224      0    1792        8      0      0       0        0      0      0
```

Sometimes, the output of `vmstat` can be slightly cluttered. You can widen the output using the -w options. This can be used on any `vmstat` command, such as the following:

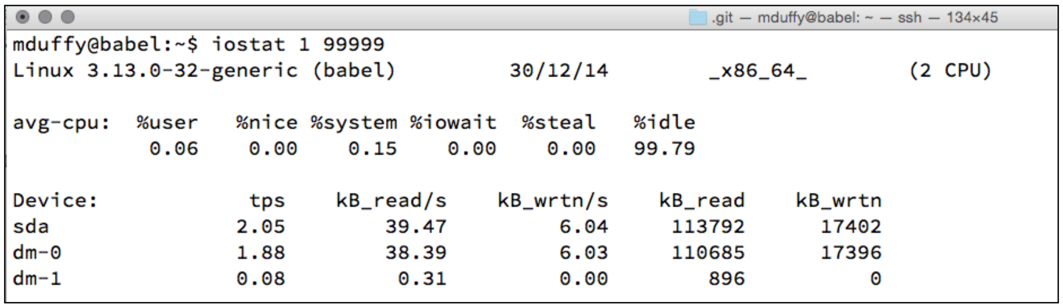
```
vmstat -d -w
```

4. Although `vmstat` is capable of displaying disk statistics, there is a tool that is better suited to this task in the shape of `iostat`. The `iostat` is able to display relatively detailed statistics of IO on a server in real time and can be used to reveal performance bottlenecks caused by disk devices.

   The following command will display basic statistics and just like `vmstat`, it will repeat the information every *n* seconds for *n* times, where *n* is a user-specified input:

   ```
   iostat 1 99999
   ```

   This will give you an output similar to the following screenshot:



```
mduffy@babel:~$ iostat 1 99999
Linux 3.13.0-32-generic (babel)        30/12/14        _x86_64_        (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.06    0.00    0.15    0.00    0.00   99.79

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               2.05        39.47         6.04     113792      17402
dm-0              1.88        38.39         6.03     110685      17396
dm-1              0.08         0.31         0.00        896          0
```

5. By default, `iostat` will show you the CPU information and disk information for all devices. You can drill into the information that `iostat` produces by using some simple options. For instance, you can show only information for device `sda`, and only disk statistics by using the following options:

   ```
   iostat -d -p sda 1 9999
   ```

Viewing historical resource usage with SAR

The tools that we have looked at so far are fantastic to analyze problems that are present now; but what about when you need to look at issues that occurred in the past? For that you can use the **System Activity Report** (**SAR**) tool. Using the `sar` tool, you will be able to look back over a period of time and see how the server has been running.

This recipe will demonstrate how to install and use the `sysstat` tools; thus, allowing you to examine historical system statistics.

Getting ready

For this recipe, you will need either a Debian or Red Hat based server.

How to do it...

Let's take a look at how to install and use `sysstat`, also allowing you to examine historical SAR:

1. Install the `sysstat` package using the following command for a Debian-based distribution:

   ```
   $ sudo apt-get install sysstat
   ```

   We can also use the following command for a RHEL-based distribution:

   ```
   $ sudo yum install sysstat
   ```

2. Edit the `/etc/default/sysstat` file with your favorite text editor and change the following value from:

   ```
   ENABLED="false"
   ```

   To:

   ```
   ENABLED="true"
   ```

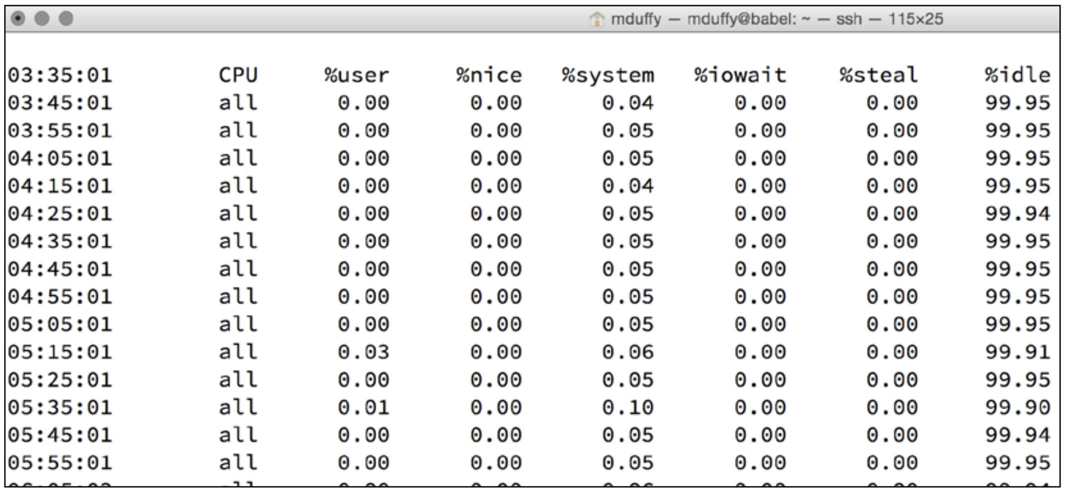
3. Restart the `sysstat` service using the following command:

   ```
   $ sudo service sysstat restart
   ```

4. By default, `sar` stats are collected every 10 minutes. The data is collected using a simple cron job configured within `/etc/cron.d/sysstat`. This job can be amended to collect the data as frequently as you require.
5. Use the following command to view basic CPU statistics, including wait times:

   ```
   sar -u
   ```

   This should produce the following output:



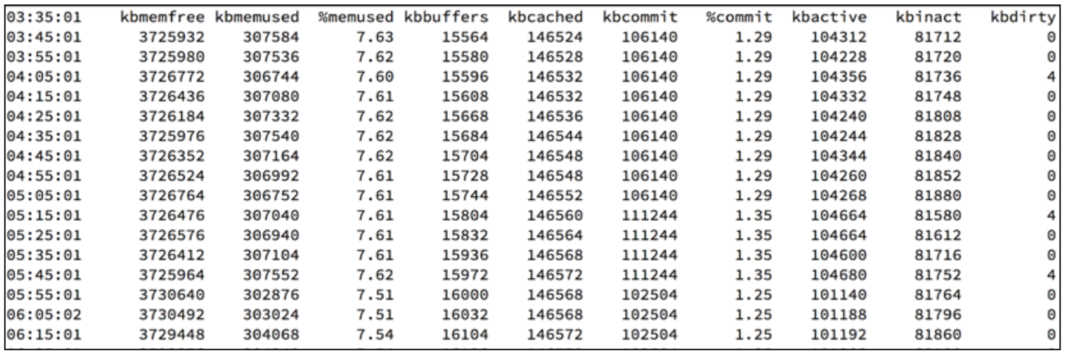
```
03:35:01        CPU     %user     %nice   %system   %iowait    %steal     %idle
03:45:01        all      0.00      0.00      0.04      0.00      0.00     99.95
03:55:01        all      0.00      0.00      0.05      0.00      0.00     99.95
04:05:01        all      0.00      0.00      0.05      0.00      0.00     99.95
04:15:01        all      0.00      0.00      0.04      0.00      0.00     99.95
04:25:01        all      0.00      0.00      0.05      0.00      0.00     99.94
04:35:01        all      0.00      0.00      0.05      0.00      0.00     99.95
04:45:01        all      0.00      0.00      0.05      0.00      0.00     99.95
04:55:01        all      0.00      0.00      0.05      0.00      0.00     99.95
05:05:01        all      0.00      0.00      0.05      0.00      0.00     99.95
05:15:01        all      0.03      0.00      0.06      0.00      0.00     99.91
05:25:01        all      0.00      0.00      0.05      0.00      0.00     99.95
05:35:01        all      0.01      0.00      0.10      0.00      0.00     99.90
05:45:01        all      0.00      0.00      0.05      0.00      0.00     99.94
05:55:01        all      0.00      0.00      0.05      0.00      0.00     99.95
```

> Note that most `sar` commands can also produce output in real time by adding a duration and repetition, much the same as the `vmstat` and `iostat` commands. For instance, `sar -u 1 30` will display the basic CPU statistics every second for 30 seconds.

6. Use the following command to view the available memory statistics:

   `sar -r`

   This should produce an output that looks similar to the following screenshot:

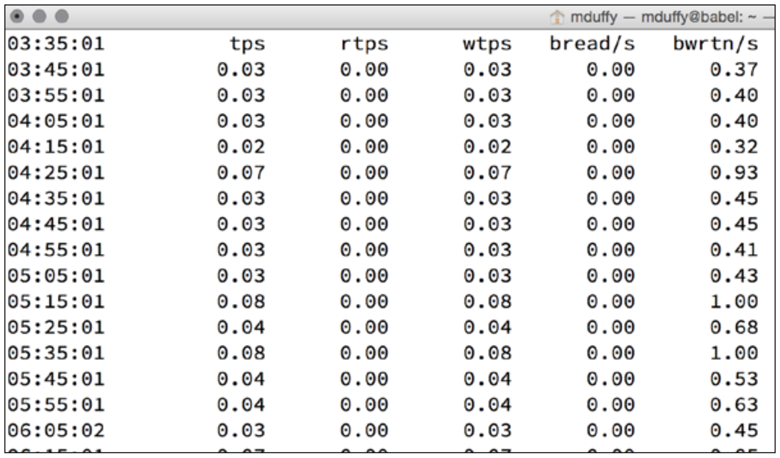


```
03:35:01    kbmemfree kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
03:45:01      3725932    307584      7.63     15564    146524    106140      1.29    104312     81712         0
03:55:01      3725980    307536      7.62     15580    146528    106140      1.29    104228     81720         0
04:05:01      3726772    306744      7.60     15596    146532    106140      1.29    104356     81736         4
04:15:01      3726436    307080      7.61     15608    146532    106140      1.29    104332     81748         0
04:25:01      3726184    307332      7.62     15668    146536    106140      1.29    104240     81808         0
04:35:01      3725976    307540      7.62     15684    146544    106140      1.29    104244     81828         0
04:45:01      3726352    307164      7.62     15704    146548    106140      1.29    104344     81840         0
04:55:01      3726524    306992      7.61     15728    146548    106140      1.29    104260     81852         0
05:05:01      3726764    306752      7.61     15744    146552    106140      1.29    104268     81880         0
05:15:01      3726476    307040      7.61     15804    146560    111244      1.35    104664     81580         4
05:25:01      3726576    306940      7.61     15832    146564    111244      1.35    104664     81612         0
05:35:01      3726412    307104      7.61     15936    146568    111244      1.35    104600     81716         0
05:45:01      3725964    307552      7.62     15972    146572    111244      1.35    104680     81752         4
05:55:01      3730640    302876      7.51     16000    146568    102504      1.25    101140     81764         0
06:05:02      3730492    303024      7.51     16032    146568    102504      1.25    101188     81796         0
06:15:01      3729448    304068      7.54     16104    146572    102504      1.25    101192     81860         0
```

7. Seeing the IO stats for individual block devices can be helpful when tracking down performance issues. You can use the following command to view these statistics with `sar`:

   `sar -b`

   This will produce an output similar to the following screenshot:



```
03:35:01          tps      rtps      wtps   bread/s   bwrtn/s
03:45:01         0.03      0.00      0.03      0.00      0.37
03:55:01         0.03      0.00      0.03      0.00      0.40
04:05:01         0.03      0.00      0.03      0.00      0.40
04:15:01         0.02      0.00      0.02      0.00      0.32
04:25:01         0.07      0.00      0.07      0.00      0.93
04:35:01         0.03      0.00      0.03      0.00      0.45
04:45:01         0.03      0.00      0.03      0.00      0.45
04:55:01         0.03      0.00      0.03      0.00      0.41
05:05:01         0.03      0.00      0.03      0.00      0.43
05:15:01         0.08      0.00      0.08      0.00      1.00
05:25:01         0.04      0.00      0.04      0.00      0.68
05:35:01         0.08      0.00      0.08      0.00      1.00
05:45:01         0.04      0.00      0.04      0.00      0.53
05:55:01         0.04      0.00      0.04      0.00      0.63
06:05:02         0.03      0.00      0.03      0.00      0.45
```

Installing and configuring a Git client

One key element in moving towards using DevOps techniques is the ability to manage and develop your infrastructure as code. Using version control is second nature to most developers; however, some System Administrators have not yet fully embraced version control. It is important that all DevOps engineers are both familiar with, and able to use a good version control system. Using version control, you can immediately pinpoint where, when and why the changes were introduced; it also allows you to experiment with alternative approaches using branches of existing code.

[Don't be tempted to think that version control is just for code. Version control can also be used to contain configuration items where they exist in the form of plain text (YAML, JSON, or INI files for instance). If you use version control to control changes, you can immediately gain a complete record of the changes made to that particular system.]

Getting ready

For this recipe, you need an Ubuntu 14.04 server.

How to do it…

Let's install and configure a Git repository:

1. Install the `git` client using the following command:

   ```
   sudo apt-get install git
   ```

2. Once the `git` client is installed, you need to configure it with your credentials:

   ```
   git config --global user.email "<Your Email address>"
   git config --global user.name "<Your actual name>"
   ```

Creating an SSH key for Git

Although you can maintain your code using local Git repositories, at some point you will want to either clone from, or push to, a remote Git repository. Although it is possible to use HTTP authentication, it can be both more secure, and certainly more convenient to use an SSH and a key to manage your authentication.

This recipe will show you how to generate an RSA SSH key that is suitable for use with Git, and also to authenticate against Linux servers.

Getting ready

For this recipe, you either need a Red Hat- or Debian-based Linux host.

How to do it

Let's create an SSH key for Git:

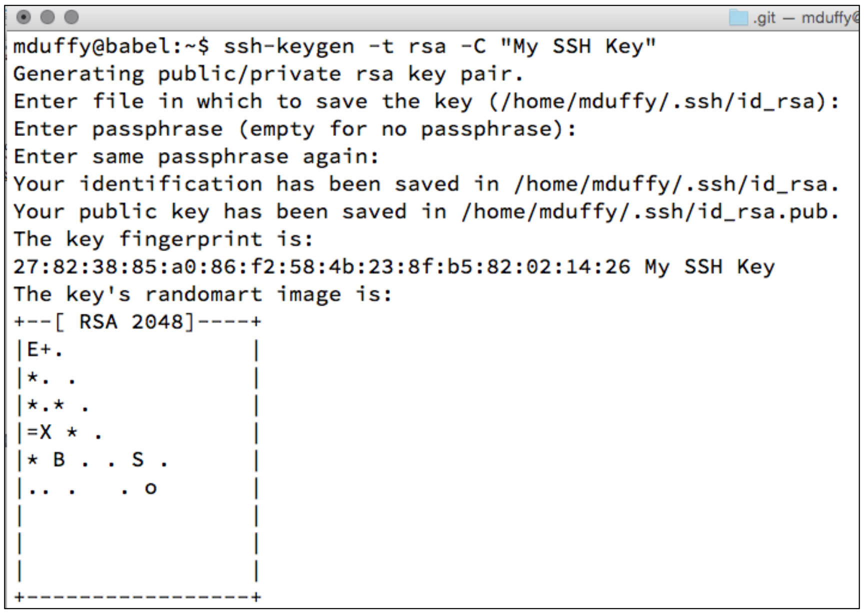
1. Create a new RSA key using the `ssh-keygen` command:

   ```
   ssh-keygen -t rsa -C "My SSH Key"
   ```

   Replace `"My SSH Key"` with an identifying text such as `My laptop`. This helps when you are managing multiple keys.

2. You will be prompted for a `passphrase` after running the preceding command; it's highly recommended that you create one to ensure the security of your key; otherwise, if you lose your private key, any scallywag who finds it can use it to access your systems. You can alleviate the tedium of typing in the password using an `ssh-agent` to store the details for the duration of a session.

   When you use the `ssh-keygen` command, you will see that it produces an output similar to the following screenshot:



```
mduffy@babel:~$ ssh-keygen -t rsa -C "My SSH Key"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mduffy/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mduffy/.ssh/id_rsa.
Your public key has been saved in /home/mduffy/.ssh/id_rsa.pub.
The key fingerprint is:
27:82:38:85:a0:86:f2:58:4b:23:8f:b5:82:02:14:26 My SSH Key
The key's randomart image is:
+--[ RSA 2048]----+
|E+.              |
|*.  .            |
|*.* .            |
|=X * .           |
|* B . . S .      |
|.. .    . o      |
|                 |
|                 |
|                 |
+-----------------+
```

How it works...

By default, the `ssh-keygen` command will create a new set of files in your home directory, within a hidden directory named `.ssh`. This includes both your public and private keys. Remember, never, ever share your private key. If you suspect that it has been shared at all, delete it and then revoke it from any system it was previously used with and create a new key pair.

Using ssh-copy-id to copy keys

Your SSH key can be used to authenticate yourself to a Linux server, and although you can manually copy SSH keys onto the servers you control, there are easier ways to manage them. Using the `ssh-copy-id` command allows you to easily copy your public key onto a server, which can be valuable when managing a great number of servers.

Getting ready

For this recipe, either you will need a Red Hat- or Debian-based Linux host.

How to do it...

Using `ssh-copy-id` only requires a single command to copy a public key to a target server. For instance, to copy my SSH key to a server called `testserver`, you can use the following command:

```
ssh-copy-id testserver
```

How it works...

The `ssh-copy-id` command logs onto a server using another authentication method (normally a password). It then checks the permissions of the user's `.ssh` directory and copies the new key into the `authorized_keys` file.

See also

You can find further details of the `ssh-copy-id` command from the Linux `man` pages; you can invoke them using the command `man ssh-copy-id`.

Creating a new Git repository

The very first step for any new project should be to create a Git repository to hold your source code so that you can track changes from the outset. Unlike centralized version control systems such as SVN, Git allows you to easily create and add to the new repository without needing a centralized server to hold it.

This recipe will show you how to create a new Git repository that is ready for content to be added.

Getting ready

For this recipe, you will need either a Red Hat- or Debian-based Linux host with a Git client installed.

How to do it...

To create a new Git repository, follow these steps:

1. Create a new directory to contain your project in:
   ```
   mkdir ~/projects/newproject
   ```
2. Use the `git init` command to initialize the new project:
   ```
   git init ~/projects/newproject
   ```

How it works...

The `git init` command creates a directory called `.git` within the directory of your project. This directory contains all the the data required for Git to track content. Any changes made to the configuration for this repository will be contained within this directory.

See also

You can find more details on how the `git init` command works at:

`https://git-scm.com/docs/git-init`

Cloning an existing Git repository

Quite often, you'll want to clone existing code to work on it. In fact, this is probably something you are going to do more often than creating a new repository. Much like developers, DevOps engineers spend more time collaborating on existing code rather than creating brand new code.

Getting ready

For this recipe, you need either a Red Hat- or Debian-based Linux host with a Git client installed.

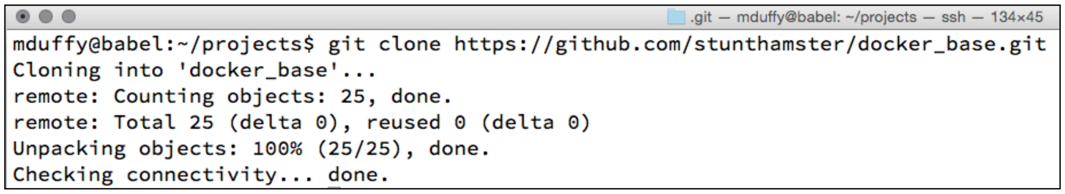
How to do it...

Let's start cloning an existing repository:

1. Change your directory into the one you want to clone the existing project into.
2. Use the `git clone` command to clone your chosen repository:

   ```
   $ git clone <GIT URL>
   ```

   This should give you an output similar to the following screenshot:



```
mduffy@babel:~/projects$ git clone https://github.com/stunthamster/docker_base.git
Cloning into 'docker_base'...
remote: Counting objects: 25, done.
remote: Total 25 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (25/25), done.
Checking connectivity... done.
```

3. Once it's cloned, you can pull any changes made by other users using the `git pull` command in the working directory:

   ```
   $ git pull
   ```

   This will connect you to the remote repository and pull any changes down to your local repository.

How it works...

The `git clone` command replicates the remote repository from a remote location to your local directory. This includes all branches and history; it's a complete copy of the repository. Once you've cloned it locally, you can branch, check in changes, and view history, all without the need to communicate with the remote again.

See also

You can find more options of how to use Git clone at `https://git-scm.com/docs/git-clone`.

Checking changes into a Git repository

Once you have worked on your code, you'll want to check your changes into your local repository. This is the first step in propagating your change further, as you need to update your local copy of the repository before you can push the changes for other users to view.

This recipe will tell you how to commit changes to your local Git repository.

Getting ready

For this recipe, you need either a Red Hat- or Debian-based Linux host.

How to do it...

Let's make changes into our local Git repository:

1. Change the directory into the one you want to use for your project.
2. Add any new files to the `git` staging area:

   ```
   git add .
   ```

   > This will add new files to the working folder (including folders and contents) to your commit. You can be more specific and add individual files if you wish.

3. Commit the new files and changes to the repository:

   ```
   git commit -am "An interesting and illuminating check in message"
   ```

The `'a'` option means '`all`'; this essentially means that you are committing all changes in this commit, and the `m` option means '`message`', and allows you to add a message explaining your commit.

How it works...

The preceding commands carry out two different tasks: the first adds new files to the change set, and the second adds any changes to the change set; it also commits them with an appropriate message. The changes exist within the Git staging area until you commit them. The best way to think of the staging area is as a buffer between the codebase and your changes. You can chuck away your Git stage at any point without affecting the branch you are currently working on.

See also

You can find more details on how to changes into Git at `https://git-scm.com/docs/git-add`.

Pushing changes to a Git remote

At some point, you're going to want to push your local repository to a remote repository. This can either be to ensure that you have a remote backup in case you accidentally drop your laptop into a car crusher, or ideally because you want to share your insanely good code with other people. Either way, it's a straightforward command.

> The most popular Git remote is probably Github. Github is a SAAS Git repository and offers a free account option for public repositories. If you want your code to be private, there are paid options available. You can find out more at `http://www.github.com`.

Getting ready

For this recipe, you need a Red Hat- or Debian-based Linux host.

How to do it...

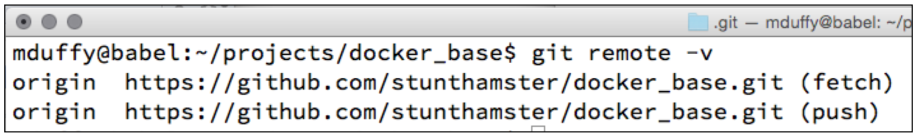
1. Configure your remote:

   ```
   git remote add origin << origin address >>
   ```

2. Verify the remote:

   ```
   git remote -v
   ```

This should produce output similar to the following screenshot:

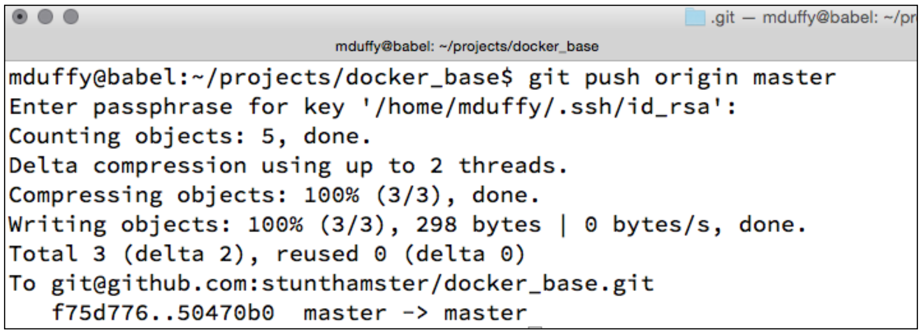


```
mduffy@babel:~/projects/docker_base$ git remote -v
origin  https://github.com/stunthamster/docker_base.git (fetch)
origin  https://github.com/stunthamster/docker_base.git (push)
```

3. Push your remote changes using the `git push` command:

   `git push origin master`

   This should produce an output similar to the following screenshot:



```
mduffy@babel:~/projects/docker_base$ git push origin master
Enter passphrase for key '/home/mduffy/.ssh/id_rsa':
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 298 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
To git@github.com:stunthamster/docker_base.git
   f75d776..50470b0  master -> master
```

It's possible that you may have several remotes configured for a single repository. In such a case, you can easily push to the correct remote by specifying it via a name, such as with the following command: `git push github master`.

How it works...

The `git push` command is essentially the opposite of the `git pull` command; it takes the contents of your local Git repository and pushes any changes that don't exist on the remote to. The `git pull` command, pushes any and all history as well, so what you have locally will also exist, in its entirety, on the remote.

See also

You can find more about pushing changes to a remote at `https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes`.

Creating a Git branch

Branching in Git is incredibly powerful and easy to use because operations are performed locally. It's not only easy but also highly recommended to operate on any major changes within a branch. You can use local branches to play with ideas, experiment, and generally mess around, all without affecting anyone else's changes. Once you've concluded your work, you can easily push the branch to the remote and issue a `pull` request to merge your changes into the main branch or if the experiment went nowhere, delete the branch without pushing the changes remotely.

Getting ready

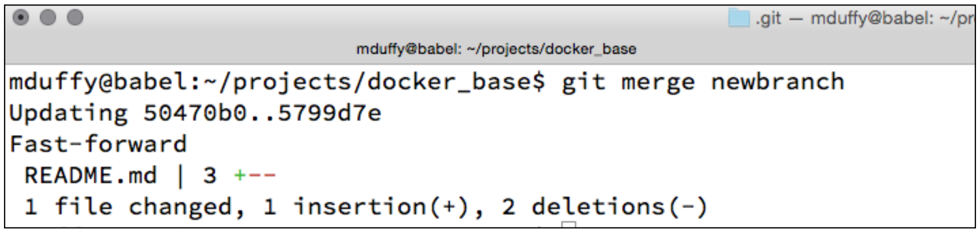
For this recipe, you need either a Red Hat- or Debian-based Linux host.

How to do it...

Let's create a Git branch:

1. Ensure that the repository is cloned locally using the `git clone` command and change your working directory into the checked out directory.
2. Issue the `branch` command to both create and switch to a new branch of the code:
   ```
   $ git checkout -b <branchname>
   ```
3. Go ahead and make some changes to your code, and when you are finished, use the `git commit` command to commit your changes. Remember, you are committing to your own branch, so your original code held in the master branch is still safe.
4. Once you've made your changes and are happy for them to be merged into the main code base, you need to switch back to the `master` branch. To switch back to a branch, you can use the `git checkout` command and the branch you wish to switch to; for instance, to switch back to the `master` branch, use the following:
   ```
   git checkout master
   ```
5. Once you have rechecked the `master` branch, you can merge your code using the `git merge` command. This will take the branch you specify and merge the code into the branch that you have currently checked out. You can issue a merge using the following command:
   ```
   $ git merge <branchname>
   ```

This should produce an output like the following screenshot:


.git — mduffy@babel: ~/pr
mduffy@babel: ~/projects/docker_base

```
mduffy@babel:~/projects/docker_base$ git merge newbranch
Updating 50470b0..5799d7e
Fast-forward
 README.md | 3 +--
 1 file changed, 1 insertion(+), 2 deletions(-)
```

You may run into a merge conflict occasionally; this essentially means that you have tried to insert a change that clashes with another developer's change. If this happens, you can invoke the `git-mergetool`, which will help you resolve the conflict by choosing which code is kept with the merge.

Once you have merged the branch, remember to both commit your changes and push them to a remote (if you have one).

6. Once you have finished with a branch, you can delete it using the following command:

   ```
   $ git branch -d <branchname>
   ```

   This will remove the branch from your Git repository.

How it works...

The `git checkout -b` command creates a new branch of the code from wherever you are in the current branch (you can easily branch from a branch). This essentially tracks any changes from the existing point of the branch, rather than copying all the existing code into the new branch; thus, making it relatively efficient to create branches from a space perspective. Any change that you make within the branch, stays within the branch until you merge them.

The `git merge` command takes the two branches and compares them for changes. As long as no merge conflicts are found, Git takes the changes from the first branch and copies them into the second.

See also

You can find more details of Git branching at `https://git-scm.com/docs/git-branch`.

2
Ad Hoc Tasks with Ansible

In this chapter, we are going to cover the following recipes:

- Installing an Ansible control node on Ubuntu
- Installing an Ansible control node on CentOS
- Creating an Ansible inventory
- Using the raw module to install `python-simplejson`
- Installing packages with Ansible
- Restarting services using Ansible
- Executing freeform commands with Ansible
- Managing users with Ansible
- Managing SSH keys with Ansible

Introduction

There is a growing number of automation tools available to DevOps Engineers, each with its individual strengths and weaknesses. Puppet, Chef, SaltStack, Ansible; the list seems to grow on a daily basis, as do the capabilities that they offer. Configuration management has become one of the core techniques that help define DevOps engineering, and is one of the key benefits of adding DevOps techniques to a team.

Configuration management is not a new concept, and there have been various tools to support automatic configuration management, with the granddaddy of them all being CFEngine. First developed by *Mark Burgess* in 1993 to solve the problems of managing his own infrastructure, CFEngine has since grown to be a fully featured commercial product used by a great number of companies.

CFEngine has inspired many features that more recent tools use, and *Mark Burgess* has written a great deal on the subject of configuration management and delivery of reliable infrastructure, and is influential in the growing discussion around the best techniques to use.

At its most basic, a configuration management tool should be able to deploy elements of an infrastructure using code to define units of configuration. It should allow an administrator the ability to run the tool multiple times and always end up with the same configuration, allowing for reliable software and configuration releases to multiple environments. Many tools have taken this a step further and embraced the concept of **idempotency**. This means that if you run the tool multiple times, it will only perform the steps required to bring a target node into a declared state and will not perform actions that have already been applied in previous executions. For example, an idempotent tool will not restart a service unless a configuration change indicates that it needs to be done.

Due to the wide variety of tools that are now available, we have a broad choice to pick from, and as with any other tool, its important to understand the strengths and weaknesses of each one. I have chosen Ansible primarily for it's ease of use, simplicity of deployment, and it's ability to be used not only for configuration management, but also for software deployments, allowing you to use a single tool to control various elements of your infrastructure. That is not to say that other configuration management tools do not have some unique features that Ansible does not; for instance, Ansible posses no reporting features unless you purchase a subscription of the commercial Ansible product, Ansible Tower. This feature is baked into Puppet with or without a commercial add-on.

Ansible is relatively unique amongst many configuration management tools in that it is designed without the concept of a centralized server to manage it. All operations from where the Ansible code is executed to the target node take place over SSH connections. This makes Ansible relatively simple to implement, as most networks already have a mechanism that gives SSH access to hosts, either from the corporate desktop, or quite often, from a designated jump server. Most users can be up and running using Ansible quickly if they use an existing communication layer; you don't even need to write any code, as you can use Ansible to run the ad-hoc tasks.

When you use Ansible to run ad-hoc tasks, you add a powerful tool to your system administration repertoire. Although you can use tools such as Csshx (`https://code.google.com/p/csshx/`) to control simultaneous terminals, it doesn't scale well beyond ten machines or so (unless your eyesight is far better than mine!).

Ansible ad-hoc tasks allow you to perform complex operations within a single line using the Ansible configuration language. This allows you to reduce the time it takes to run a command against multiple machines and use an inventory with groups; it also allows you to target the servers that you specifically want to run the command against.

Installing an Ansible control node on Ubuntu

Ansible has a very slim installation; there is no database, no custom daemons, no queues, or any other additional software required. You simply install a set of command-line tools that allow you to work with the Ansible code.

> You should put some thought into choosing your control machine. Although it's feasible to run Ansible straight from your laptop, it's probably not a good idea once you have more than one person working with the code base. Ideally, you can create a small server that you can use to run the Ansible code and then you can add safeguards around who can log in and use the tool.

Getting ready

For this recipe, you need an instance/install of Ubuntu 14.04.

How to do it...

There is a **Personal Package Archive** (**PPA**) that is available for installation of `ansible` on Ubuntu; you can use the following steps to install the latest stable release (1.9.4 at the time of writing):

1. First, you need to install the PPA repository on the Ansible node:

   ```
   $ sudo apt-add-repository ppa:ansible/ansible
   ```

   You may be prompted to add the repository; simply hit enter if you are.

2. Now you have the PPA repository installed, you need to update the apt repositories with the following command:

   ```
   $ sudo apt-get update
   ```

3. You can now install Ansible using the following command:

   ```
   $ sudo apt-get install ansible
   ```

4. You can test if Ansible is installed correctly using the version switch, as shown in the following example:

   ```
   $ ansible --version
   ```

   This should return the version of Ansible that you have installed.

See also

You can find out more about how to set up the Ansible control node using the Ansible documentation at `http://docs.ansible.com/intro_installation.html`.

Installing an Ansible control node on CentOS

Ansible can be installed on many different operating systems and can run equally well on a Red Hat-based Linux distribution as it can on Ubuntu. This recipe will show you how to install Ansible on a CentOS 7 server.

Getting ready

For this recipe, you need an instance of CentOS 7.

How to do it...

Let's install an Ansible control node on CentOS:

1. We need to install the **Extra Packages for Enterprise Linux** (**EPEL**) repository before we install Ansible. You can install it with the following command:

   ```
   $ sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-
   release-latest-7.noarch.rpm
   ```

2. Install Ansible using the following command:

   ```
   $ sudo yum install ansible
   ```

3. You can test if Ansible is installed correctly using the version switch, as shown in the following example:

   ```
   $ ansible --version
   ```

   This should return the version of Ansible that you have installed.

See also

You can find out more about how to set up the Ansible control node using the Ansible documentation at `http://docs.ansible.com/intro_installation.html`.

Creating an Ansible inventory

Every action you take with Ansible is applied to an item in your inventory. The Ansible inventory is essentially a catalog that is used to record both target nodes and a group with which you can map a node to the role it is going to assume.

Getting ready

For this recipe, you need to have Ansible installed on the machine you intend to use as a control node and a target node to run your actions against. The examples use six different target hosts, but this is not mandatory; all you need to do is simply adjust the inventory to match your requirements.

How to do it...

The inventory file is formatted as an `ini` file and is essentially a simple text file that can store your catalog. Let's assume that we have a small infrastructure that resembles the following:

Function	**Name**
`haproxy`	`haproxy01`
`httpd`	`web01` through to `web04`
`mysql`	`mysql01`

Remember, adjust the preceding list to reflect your particular infrastructure.

[Depending on how you have installed Ansible, you may find that there is an example file already at that location. If the file is present, simply comment out or remove the content.]

Let's create our first Ansible inventory. Using your favorite editor, edit the file located at `/etc/ansible` called **hosts**:

1. Let's start by creating a basic inventory. Insert the following code:

```
haproxy01
web01
web02
web03
web04
mysql01
```

[Ensure that the names that you enter into your inventory can be resolved by their names, either using DNS or a host's file entry.]

2. That's all that is required for a basic Ansible inventory file; however, despite having different names, from Ansible's point of view these are all part of the same group. Groups allow you to differentiate between different collections of servers, and in particular they allow you to apply different commands to different groups of servers. Let's alter our Ansible inventory to add some groups; this is done using a pair of brackets within which you can insert your group name. Alter your Ansible inventory to look like the following example:

   ```
   [loadbalancer]
   haproxy01
   [web]
   web01
   web02
   web03
   web04[database]
   mysql01
   ```

3. We now have an inventory file that can be used to control our hosts using Ansible; however, we have lost the ability to send commands to all hosts at once due to grouping. For that, we can add a final group that is, in fact, a group of groups. This will take our groups and form a new group that includes all of the groups in once place, allowing us to easily manipulate all our hosts at once, whilst still retaining the ability to distinguish between individual groups of nodes. To accomplish this, open your Ansible inventory and add the following to the bottom of the file:

   ```
   [all:children]]
   loadbalancer
   web
   database
   ```

4. The `children` keyword signifies that the entries that belong to this group are, in fact, groups themselves. You can use the `children` keyword to make sub-collections and not just collect all groups. For instance, if you have two different data centers, you can use groups called `[dca:children]` and `[dcb:children]` to list the appropriate servers under each.
5. We now have everything that we need to address our servers, but there is one last trick left to make it more compact and readable. Ansible inventory files understand the concept of ranges, and since our servers have a predictable pattern, we can use this to remove some of the entries and **Do not repeat yourself** (**DRY**) the file up a little. Again, open the file in `/etc/ansible/hosts` and change the code to reflect the following:

   ```
   [loadbalancer]
   haproxy01
   [web]
   web01:04
   ```

```
[database]
mysql01
[all:children]]
loadbalancer
web
```

As you can see, we have replaced the four manual entries with a range; very useful when you have to manage a large infrastructure.

> Although it's recommended, you don't need to install the inventory into /etc/ansible - you can have it anywhere and then use the -i option on the Ansible command to point to its actual location. This makes it easier to package the inventories along with Playbooks.

See also

You can find out more about the Ansible inventory at the Ansible documentation site; the following link in particular contains some interesting details at `http://docs.ansible.com/intro_inventory.html`.

Using the raw module to install python-simplejson

Ansible has very few dependencies; however, every managed node requires the `python-simplejson` package to be installed to allow full functionality. Luckily, Ansible has a raw module, which allows you to use Ansible in a limited fashion to manage nodes. Generally speaking, this should be used as a one-trick pony, using it to install `python-simplejson`, but it is worth keeping in mind if you ever need to perform the management of servers that might not be able to have this package installed for some reason.

> An Ansible module is essentially a type of plugin that extends the functionality of Ansible. You can perform actions such as installing packages, restarting networks, and much more using modules. You can find a list of core Ansible at `http://docs.ansible.com/ansible/modules_by_category.html`.

Getting ready

All you need to use in this recipe is a configured Ansible control node and an Ansible inventory describing your target nodes.

How to do it...

Let's use a raw module to install `python-simplejson`:

1. Use the following command to install the `simple-python` module:

   ```
   ansible all --sudo --ask-sudo-pass -m raw -a 'sudo apt-get -y
   install python-simplejson'
   ```

   In the preceding command, we have used several options. The first two, `--sudo` and `--ask-sudo-pass`, tell Ansible that we are employing a user that needs to invoke `sudo` to issue some of the commands and using `--ask-sudo-pass` prompts us for the password to pass onto `sudo`. The `-m` switch tells Ansible which module we wish to use; in this case, the raw module. Finally, the `-a` switch is the argument we wish to send to the module; in this case, the command to install the `python-simplejson` package.

   [You can find further information about the switches that Ansible supports using the command `ansible --help`.]

2. Alternatively, if you manage a CentOS server, you can use the raw module to install the `python-simplejson` package on these servers using the following command:

   ```
   ansible all --sudo --ask-sudo-pass -m raw -a 'sudo yum -y install
   python-simplejson'
   ```

See also

You can find the details of the raw module at `http://docs.ansible.com/raw_module.html`.

Installing packages with Ansible

Sometimes you need to install a package without using full-blown automation. The reasons may vary, but quite often this can be when you need to get a software patch out right now. However, most times this will be to patch an urgent security issue that cannot wait for a full-blown configuration management release.

[If you do use this recipe to install software, make sure that you add the package to the subsequent configuration management. Otherwise, you will end up with a potentially inconsistent state, and even worse, the specter of Ansible rolling back a patch if a package is defined as a certain version within an Ansible Playbook.]

Getting ready

For this recipe, you will need to have a configured Ansible inventory. If you haven't already configured one, use the recipe in this chapter as a guide to configure it. You will also need either a Centos or an Ubuntu server as a target.

How to do it...

Let's install packages with Ansible:

1. To install a package on an Ubuntu server we can make use of the `apt` module. When you specify a module as part of an ad hoc command, you will have access to all the features within that particular module. The following example installs the `httpd` package on the `[web]` group within your Ansible inventory:

   ```
   ansible web -m apt -a "name=apache2 state=present"
   ```

   You can find more details of Ansible modules using the `ansible-doc` command. For instance, `ansible-doc apt` will give you the full details of the `apt` module.

2. Alternatively, you might want to use this technique to install a certain version of a package. The next example commands every node to install a certain version of Bash:

   ```
   $ ansible all -m apt -a "name=bash=4.3 state=present"
   ```

3. You can even use the `apt` module to ask the target nodes to update all installed software using the following command:

   ```
   $ ansible all -m apt -a "upgrade=dist"
   ```

4. You can use the `yum` module to install software on RHEL-based machines using the following command:

   ```
   $ ansible all -m yum -a "name=httpd state=present"
   ```

5. Just like the example for Ubuntu servers, you can use Ansible to update all the packages on your RHEL-based servers:

   ```
   $ ansible all -m yum -a "name=* state=latest"
   ```

See also

- You can find more details of the Ansible `apt` module, including the available modules, at `http://docs.ansible.com/apt_module.html`
- You can find more details of the `Yum` module at `http://docs.ansible.com/ansible/yum_module.html`

Restarting services using Ansible

Now that we have defined our inventory, we are ready to use Ansible to perform actions. Arguably, one of the most important adhoc actions you can take is to restart services on target nodes. At first, this might seem a bit of an overkill compared to simply logging on to the server and doing it, but when you realize that this action can be scaled anywhere from one to one million servers, its power becomes apparent.

Getting ready

You'll need an inventory file before you try this, so if you have not got it already, go ahead and set one up. The following examples are based on the inventory set out in the preceding recipe, so you'll need to change the examples to match your environments.

How to do it...

To restart a service, we can use the Ansible service module. This supports various activities such as starting, stopping, and restarting services:

- For example, issue the following command to restart MySQL:

  ```
  ansible mysql -m service -a "name=mysql state=restarted"
  ```

- You can also use the service module to stop a service:

  ```
  ansible mysql -m service -a "name=mysql state=stopped""
  ```

- Alternatively, you can also use the service module to start a service:

  ```
  ansible mysql -m service -a "name=mysql state=started""
  ```

See also

You can find more details about the service module from the Ansible documentation at `http://docs.ansible.com/service_module.html`.

Executing freeform commands with Ansible

Sometimes, you need to be able to run actual shell commands on a range of servers. An excellent example will be to reboot some nodes. This is not something that you would put into your automation stack, but at the same time, it is something you would like to be able to leverage your automation tool to do. Ansible enables you to do this by sending arbitrary commands to a collection of servers.

Getting ready

You'll need to an inventory file before you try this, so if you don't have it already, go ahead and set one up. You can use the recipe of this chapter, *Creating an Ansible inventory*, as a guide.

How to do it...

The command is simple and takes the following form:

```
ansible <ansible group> -a "<shell command>"
```

For example, you can issue the following command to reboot all the members of the `db` group:

```
ansible mysql -a "reboot -now"
```

It's important to keep an eye on parallelism when you have many hosts. By default, Ansible will send the command to five servers. By adding a `-f` flag to any command in this chapter, you can increase or decrease this number.

Managing users with Ansible

There are times when you might want to manage users on multiple nodes manually. This may be to fit in with a user creation process that already exists, or to remove a user in a hurry if you find out that they need to have their access revoked. Either way, you can use Ansible ad-hoc commands to add, amend, and delete users across a large number of nodes.

Getting ready

All you need to use for this recipe is a configured Ansible control node and an Ansible inventory describing your target nodes.

How to do it...

Let's configure `ansible user` to manage some users:

1. You can use the following command to add a user named `gduffy` to a group called `users` on every node within your Ansible inventory:

   ```
   $ ansible all -m user -a "name=gduffy" comment="Griff Duffy"
   group=users password="amadeuppassword"
   ```

2. We can also use Ansible to remove users. Issue the following command from your control node to remove the user `gduffy` from every database node defined in your Ansible inventory:

   ```
   ansible db -m user -a "name=gduffy" state=absent remove=yes"
   ```

3. We can also easily amend users. Issue the following command from your control node to change the user Beth to use the Korn shell and to change her home directory to `/mnt/externalhome` on all nodes:

   ```
   ansible all -m user -a "name=beth shell=/bin/ksh home=/mnt/
   externalhome"
   ```

See also

The preceding examples make use of the Ansible User module. You can find the documentation for this module at `http://docs.ansible.com/user_module.html`.

Managing SSH keys with Ansible

One of the most tedious administration tasks can be managing user keys. Although tools such as `ssh-copy-id` make it easy to copy your key to single servers, it can be a taller order to copy them out to several hundred or even a few thousand servers. Ansible makes this task exceptionally easy and allows you to mass-revoke keys when you need to ensure that access has been removed for users across a large server estate.

Getting ready

All you need to use for this recipe is a configured Ansible control node and an Ansible inventory describing your target nodes. You should also have a SSH key, both public and private that you wish to manage.

How to do it...

Let's use SSH keys to manage Ansible:

1. The first thing we might want to do is create a user and simultaneously create a key for them. This is especially useful if you use a network jump box, as it means that you have no dependency on the user supplying a key; it's an integral part of the process. Run the following command to create a user called **Meg** with an associated key:

   ```
   ansible all -m user -a "name=meg generate_ssh_key=yes"
   ```

2. Often, a user either has an existing key they wish to use, or needs to change an installed key. The following command will allow you to attach a key to a specified account. This assumes that your key is located in a directory called **keys** within the working directory from which you run the following command:

   ```
   ansible all -m copy -a "src=keys/id_rsa dest="/home/beth/.ssh/id_
   rsa mode=0600
   ```

3. Once a user has their Private key setup, you need to push their public key out to each and every server that they wish to access. The following command adds my public key to all web servers defined within the Ansible inventory. This uses the `lookup` Ansible function to read the `local` key and send it to the remote nodes:

   ```
   ansible web_servers -m authorized_key -a "user=michael key="{{
   lookup('file', '/home/michael/.ssh/id_rsa.pub') }}"
   ```

See also

The preceding examples use a mix of Ansible modules to achieve the end result. You can find the documentation for the following modules at:

- **User module**: `http://docs.ansible.com/user_module.html`
- **Copy module**: `http://docs.ansible.com/copy_module.html`
- **Authorized_key module**: `http://docs.ansible.com/authorized_key_module.html`

3
Automatic Host builds

In this chapter, we are going to cover the following topics:

- Creating an `Apt` mirror using `aptly`
- Automated installation using PXE boot and a preseed file
- Automating post-installation tasks

Introduction

Building new hosts is one of the most basic tasks a DevOps engineer can undertake. Traditionally speaking, this used to be a manual task involving the mounting of install media, navigating through menus, and inputting the correct values at prompts. Whether you are building virtual hosts or physical servers, automation can bring about some fantastic changes in both speed and reliability of builds. Regardless of whether you are creating one or one-hundred hosts, you can be sure that with automation, your servers will be configured exactly how you want them to be.

If you are working in a completely virtual environment then this is a problem that you may have already solved; indeed, a driver for many organizations to move to virtualization was in part to solve this very issue. However, many organizations are still using bare metal servers either due to performance or policy constraints, but using bare metal does not mean that you cannot automate.

Once you have automatic host builds working, it can bring some advantages that you may not have considered at first, and this can make profound changes to how you manage your infrastructure. Take for instance, the nightmare scenario that every Systems administrator dreads, an intrusion into your network where the extent of breach is difficult to ascertain. This is a potentially disastrous issue, as virtually every affected server must naturally be considered hopelessly compromised. In serious cases, this can encompass hundreds of servers and can even render the **disaster recovery** (**DR**) site compromised, depending on how you replicate to it.

This may not be as far-fetched as you first imagine. Over the past few years, companies such as Sony, Avid Media, and Gemalto have been implicated in hacks where the attacker has attained wide-reaching access to their networks. When something like this occurs, it's a long and expensive process to regain trust in your own systems.

At this point, you generally have only one option, and that's to quarantine and rebuild the servers. This can be a serious investment in man-hours if you have to build more than four or five hosts by hand. The average manual install of an Ubuntu server can take tens of button clicks and many prompts to be filled in correctly and accurately. Rebuilding a modest network of ten servers, just to the basic OS level can tie up a technician for a considerable amount of time; there's only so many tasks that you can make parallel in a manual install. In this situation, having a tried and tested way to install hosts automatically will give the poor DevOps engineer the ability to rebuild their hosts in a matter of minutes, leaving plenty of time to examine logs, secure firewalls, and of course, have a stiff drink to recover his nerves.

This is an extreme example, but you can also look at the gains that automation brings in terms of flexibility. For instance, automatic host builds are a key part of any form of elastically scalable service, regardless of whether it is hosted in your own infrastructure or as part of a cloud service. You can also use automatic host builds to enable developers to create adhoc development environments easily. This can fit nicely with the use of software such as Vagrant, allowing you to create base images for developers that can match the production hosts. This allows one of the key concepts of a DevOps-driven infrastructure, which is that environments are matched as exactly as possible, from the developer's desktop all the way through to the running service. If nothing else, this will sharply reduce the familiar refrain of "It works on my system" when trying to diagnose an issue.

Building hosts automatically has become increasingly easy in recent years, and both Ubuntu- and Red Hat-based distributions have developed simple yet powerful ways of creating automated installation scripts. Both distributions have arrived at a similar solution of using a simple manifest that allows predetermined input for the underlying installer.

In this chapter, we are going to look at the Debian and by extension at the Ubuntu system for host builds: Preseed. Using Preseed, you can pre-fill the answers to the questions that are normally asked at the time of install. As it is a simple manifest, you can use it to form a part of a network-based automated build, or at a pinch, you can also embed it on a CD and boot hosts that way. The end result should be the same regardless of the method used. In this chapter, we are going to focus on the network build of bare metal systems, as this is the gold standard you should aim for; allowing for quick, easy, and consistent builds with very little effort. To do this, we are going to create the building blocks of an automated build, a repository to fetch packages from, a server that can service PXE clients, and finally, we are going to take a look at an example Preseed file.

Creating an Apt mirror using aptly

At its most basic, an `apt` repository is a simple web server that serves both packages and more importantly, it serves the metadata that describes the contents of the repository. Although you can use the repositories that Ubuntu provides to build your hosts, you will hit two issues. The first being that the packages are updated all the time, meaning a host you build this week may not be the same as a host that was built the week before. The second issue is that depending on your build, you could potentially use a lot of bandwidth. If you have five hundred hosts and you suddenly need to update Bash on all of them due to a security issue, you are going to use a huge amount of bandwidth. It's a much better practice to keep a mirror of the official repository, allowing you to update it when you feel comfortable and allowing your hosts to install new packages at the speed of your local network.

There are several ways in which you can manage local repositories and it can be done using a simple set of Bash scripts. However, you want to ensure that the method you use allows you to easily merge in the upstream changes in a managed fashion, gives you an easy way to publish new repositories, and allows easy management of your existing repositories.

Enter `aptly` (`http://www.aptly.info`). The `aptly` is an open source project written by the talented Andrey Smirnov, and makes managing mirroring repositories easy. It also offers advanced features such as snapshots, merging, and publishing. We're going to use `aptly` to create a mirror of the Ubuntu package `repos`, which will allow us to use a local repository to make package installations much quicker and also offers us a fixed set of package versions to install.

Getting ready

For this recipe, you should start with a clean install of Ubuntu 14.04. You'll also need a substantial quantity of storage space; I recommend putting aside at least 50GB for the Ubuntu repositories. If you are short of space or you want to have a more centralized management of the storage, there is no reason why it could not reside on a network-attached storage, such as an NFS device.

How to do it...

Let's create a repository using the `aptly` tool:

1. First of all, you need to install the `aptly` package. To achieve this, edit your `apt` sources list in `/etc/apt/sources.list.d` and add the following line:

   ```
   deb http://repo.aptly.info/ squeeze main
   ```

2. Once you have added the repository, you need to import the `GPG` key. You can achieve this using the following command:

   ```
   $ sudo apt-key adv --keyserver keys.gnupg.net --recv-keys
   E083A3782A194991
   ```

3. Now, you are almost ready to install `aptly`. You just need to update your `apt` repository to ensure that `aptly` is listed; you can do this with the following command:

   ```
   $ sudo apt-get update
   ```

4. Once you've updated your `apt` repository, you can install `aptly` with the following command:

   ```
   $ sudo apt-get install aptly
   ```

   This will fetch the `aptly` packages and dependencies and then install them for you.

5. The next task is to configure `aptly`. Out of the box aptly is well configured, but there are a few things that we need to adjust; primarily, the storage location where the mirror will be held.

   The `aptly` configuration file is located at `/etc/aptly.conf` and it's OK if it's not present there; you can go ahead and create one. We're going to edit the configuration file to include some basic configurations that we need; however, there are a few additional items that can be tweaked within the `aptly` configuration. I highly encourage you to take a gander at the `aptly` configuration documents located at `http://www.aptly.info/doc/configuration/`.

   > Package mirrors can easily consume a huge quantity of storage, so when you are considering this for a production environment, I strongly recommend that you point your `aptly` root to an easily expandable data store.

6. Open your `aptly` configuration by editing `/etc/aptly.conf` and insert the following configuration:

   ```
   {
     "rootDir": "/var/spool/aptly",
     "downloadConcurrency": 4,
     "downloadSpeedLimit": 0,
   ```

```
    "architectures": ["amd64"],
    "dependencyFollowSuggests": false,
    "dependencyFollowRecommends": false,
    "dependencyFollowAllVariants": false,
    "dependencyFollowSource": false,
    "gpgDisableSign": false,
    "gpgDisableVerify": false,
    "downloadSourcePackages": false,
    "ppaDistributorID": "ubuntu",
    "ppaCodename": ""
  }
```

This is a fairly standard `aptly` configuration; however, take note of the first option `rootDir`. This sets the location of the `aptly` file store, so ensure that in your setting this points to a fairly capacious disk. Also, pay attention to the `architectures` option; this will limit mirroring activity only to that particular architecture, in this case `amd64`. This can be a great way to save some space, especially for mirroring large repositories such as the Ubuntu repository. Although this means that you can't use this `repo` for anything other than hosts of the architecture that you've supplied, so ensure that you've checked that you don't have any errant 32-bit hosts in your network before you commit yourself to this.

7. Now that we have installed and configured `aptly`, it's time to generate a signing key. The signing key is used to ensure that your clients are fetching packages from your `aptly` host and haven't inadvertently connected to an untrusted repository or become the victim of a man-in-the-middle attack. To start, we need to install the tools to generate our GPG keys. You can do this with the following command:

   ```
   $ sudo apt-get install gnugpg
   ```

   This will install all the required packages to create a key pair. Next, let's go ahead and create our keys.

   > Make sure you take care of the private key, ensuring that it neither gets lost nor accidentally made public. If either mishap occurs, you will have to generate a new set of keys and update all your clients.

8. Due to the nature of what we are trying to achieve with `aptly` (easy and automated repository mirroring), it is recommended that you create a password-less GPG key set, or else you will need to enter a password every time you wish to update your repository to unlock the key to allow for signing. To create a password-less key, you can use a GPG batch file. To do this, you need to create a new file called `gpgbatch` and add the following contents:

   ```
   %echo Generating a default key
   Key-Type: default
   ```

```
Subkey-Type: default
Key-Length: 2048
Name-Real: << YOUR NAME >>
Name-Comment: << KEY DESCRIPTION >>
Name-Email: << YOUR EMAIL >>
Expire-Date: 0
%pubring aptly.pub
%secring aptly.sec
%commit
%echo done
```

9. Once you've edited the preceding file with your information, save it and use the following command to generate your key:

   ```
   gpg2 --batch --gen-key gpgbatch
   ```

   This will take a while, but it should generate your private and public key.

10. Now that we have our key pair, we are ready to create and sign our mirrors. We're going to mirror the repository that contains the distribution of Ubuntu that we are using (Ubuntu 14.04). The very first step is to import the keys from the remote mirror. You can do this using the following command:

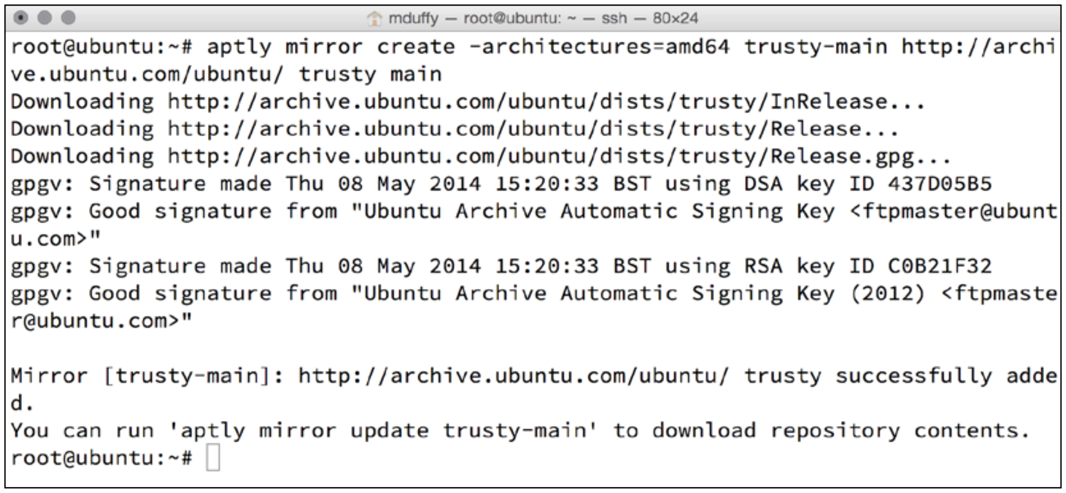
    ```
    $ sudo gpg --no-default-keyring --keyring trustedkeys.gpg
    --keyserver keys.gnupg.net --recv-keys 437D05B5
    ```

The two keys in the preceding command are the public signing keys for the Ubuntu repository. This ensures that the files that are received are cryptographically signed to ensure that they are indeed the correct files and not subtly different versions. Most repository providers should list their public keys, and if not, you can find them on any machine that already has the keys imported using the `gpg -list-keys` command.

11. Next, we will perform the actual mirroring of the repository using the following command:

    ```
    $ sudo aptly mirror create -architectures=amd64 trusty-main
    http://archive.ubuntu.com/ubuntu trusty main
    ```

    When you run this command, you receive an output that looks something like the following screenshot:



```
root@ubuntu:~# aptly mirror create -architectures=amd64 trusty-main http://archi
ve.ubuntu.com/ubuntu/ trusty main
Downloading http://archive.ubuntu.com/ubuntu/dists/trusty/InRelease...
Downloading http://archive.ubuntu.com/ubuntu/dists/trusty/Release...
Downloading http://archive.ubuntu.com/ubuntu/dists/trusty/Release.gpg...
gpgv: Signature made Thu 08 May 2014 15:20:33 BST using DSA key ID 437D05B5
gpgv: Good signature from "Ubuntu Archive Automatic Signing Key <ftpmaster@ubunt
u.com>"
gpgv: Signature made Thu 08 May 2014 15:20:33 BST using RSA key ID C0B21F32
gpgv: Good signature from "Ubuntu Archive Automatic Signing Key (2012) <ftpmaste
r@ubuntu.com>"

Mirror [trusty-main]: http://archive.ubuntu.com/ubuntu/ trusty successfully adde
d.
You can run 'aptly mirror update trusty-main' to download repository contents.
root@ubuntu:~#
```

12. This creates a mirror but doesn't populate it. To populate it, issue the following command:

    ```
    $ aptly mirror update trusty-main
    ```

    This will start to download the actual files in the mirror.

    Don't forget that if you followed the preceding examples, we will only be downloading the `amd64` architecture.

13. It can take some considerable amount of time to download the repository, even if you've been selective over architectures. Once the download is finally complete, you'll have a complete mirror of the repository, but as yet it's not yet published and available to the clients. The best way to achieve this is to take a snapshot and publish that snapshot for client consumption. Issue the following command to take the snapshot:

    ```
    $ sudo aptly snapshot create trusty-main-snapshot from mirror
    trusty-main
    ```

    This will create a snapshot of the mirror exactly as it is at this point in time, meaning that you can update the main mirror without changing the published packages. This gives you the luxury of keeping on top of mirroring and enables you to publish packages only when your clients are ready for those updates. Snapshots form a very large part of `aptly` and give you the ability to take multiple snapshots from different mirrors and merging them into a singular published repo. It is handy for slip streaming security updates into a repository.

14. Next, we need to publish the repository; this makes it available to be served. You can publish your snapshot using the following command:

    ```
    $ sudo aptly publish snapshot -distribution=trusty trusty-main-
    snapshot
    ```

15. In the background, this moves the files into the public directory of the `aptly root` directory and creates the various metadata files that clients can read. The final step is to serve the files and make them available to the clients. `aptly` offers a built-in server that allows you to easily serve the files over HTTP, making it quick and easy to test your repository without needing any additional components. To serve the packages, issue the following command:

    ```
    $ sudo aptly serve
    ```

    This will start an HTTP server on port 8080 and serve the repository. Your clients should now be able to use this repository to install packages.

    The `aptly serve` command is only really intended for testing. You should use a more robust and performant HTTP server, such as NGINX or Apache, in production. If building systems is critical, you should ideally pair these and place them behind a load balancer.

See also

Aptly has fantastic documentation, and you can read it at `http://www.aptly.info/doc/overview/`.

Automated installation using PXE boot and a Preseed file

Now that we have a mirrored repository of packages, we can also use it to serve the files that build our hosts over the network. Building bare metal servers over the network has many advantages, allowing you to simply boot a bare metal server and configure it via DHCP and install an OS, all without any additional interactions. Taken to its extreme, PXE booting allows the use of completely diskless clients, which can boot and run across the network.

This is a very long recipe but it is required. Although it's relatively straightforward to set up a PXE booting environment, it does require several elements. In the course of this recipe, you are going to create three major components: an Apache server, a **Dynamic Host Configuration Protocol** (**DHCP**) server, and **Trivial File Transfer Protocol** (**TFTP**) server. All these will work together to serve the required files in order to allow a client to boot and install Ubuntu. Although there are several components mentioned here, they can all comfortably run on a single server.

An alternative to this recipe is the cobbler project (`https://cobbler.github.io`). Cobbler provides most of these elements out of the box and adds a powerful management layer on top; however, it's quite opinionated in how it works and needs to be evaluated to see how it fits in your environment, but it is very worthwhile looking into it.

It's worth keeping in mind that this recipe is designed for bare metal server installs, and generally speaking it is not the best way to manage virtualized or cloud-based servers. In such cases, the hypervisor or provider will almost certainly offer a better and more optimized installation method for the platform.

Getting ready

To follow this recipe, it is recommended that you have a host available with a clean installation of Ubuntu 14.04. Ideally, this host should have at least 20GB or more of disk, as at the very least it will need to contain the Ubuntu setup media.

How to do it...

Let's set up a PXE booting environment:

1. The first component that we are going to configure is the TFTP server. This is a stripped-down version of FTP. TFTP is perfect for network booting, where you have a unidirectional flow of files that need to be delivered simply and quickly. We are going to use the TFTP server that ships with Ubuntu 14.04. To install it, issue the following command:

   ```
   $ sudo apt-get install tftpd-hpa
   ```

   This will install the packages and their dependencies.

2. Next, we need to configure our TFTP server. Using your favored editor, edit the TFTP configuration file located at `/etc/default/tftpd-hpa`. By default, it should resemble this:

   ```
   # /etc/default/tftpd-hpa
   TFTP_USERNAME="tftp"
   TFTP_DIRECTORY="/var/lib/tftpboot"
   TFTP_ADDRESS="[::]:69"
   TFTP_OPTIONS="--secure"
   ```

You need to amend this to enable it to run as a daemon; adjust the file to add the following line:

```
# /etc/default/tftpd-hpa
RUN_DAEMON="yes"
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_ADDRESS="[::]:69"
TFTP_OPTIONS="--secure"
```

3. This allows the process to be started in a demonized mode. Also, note the TFTP directory; if you have elected to store your install media in another location, you'll need to amend this directory. Finally, start the TFTP server using the following command:

   ```
   $ sudo service tftpd start
   ```

4. Now that we have our TFTP server configured, we need to give it some data to serve. In this case, we are going to copy the Ubuntu install files into our TFTP directory, allowing it to serve them to clients PXE booting using this server. If you haven't already, download the Ubuntu 14.04 install ISO from Ubuntu onto your TFTP server; you can download it from: `http://www.ubuntu.com/download/server`. Once you've downloaded it, go ahead and mount it onto the `mnt` directory using the following command:

   ```
   $ sudo mount -o loop <location of ISO> /mnt
   ```

5. Once the ISO is mounted, you can copy it into the TFTP root directory. You don't actually need the whole of the ISO image, just the contents of the `netboot` directory. Copy it into place using the following command:

   ```
   $ cp -r /mnt/install/netboot/* /var/lib/tftpboot/
   ```

   Note that I'm copying it to the default location for the TFTP server. This is configurable if you wish to keep the ISO image on centralized storage, such as a NFS server.

6. Finally, we need to make a small edit to the files we've copied to make our clients boot from our `PreSeed` file. Open the following configuration file in your favorite editor:

   `/var/lib/tftpboot/pxelinux.cfg/default`

   Insert the following:

   ```
   label linux
           kernel ubuntu-installer/amd64/linux
           append preseed/url=http://<<NAME OF BOOT SERVER>>/ks.cfg
   vga=normal initrd=ubuntu-installer/amd64/initrd.gz ramdisk_
   size=16432 root=/dev/rd/0 rw  --
   ```

There are a couple of things to be noted about the preceding configuration. Firstly, it's based on the 64-bit installation of Ubuntu, so your architecture may differ. Secondly, note the line that reads:

```
pressed/url=http:// <<NAME OF BOOT SERVER>>//ks.cfg
```

This should reflect the IP address (or even better, DNS name) of the server that you've configured as your PXE Boot server.

7. Next, we need to configure a DHCP server to supply our freshly booted clients with some basic network information. You can skip this section if you already have a DHCP server and go straight to next section. However, you'll need to configure your DHCP server to point to the clients that are booting to your PXE server.

   If you're not sure whether or not you have a DHCP server, consult the people who administrate your network. Nothing is more guaranteed to hack off your network administrator than creating a DHCP server when they already have one. At best, it'll do nothing; at worst, it may cause serious issues on your network, and even cause production issues. If in doubt, ask.

   If you haven't already got a DHCP server in place, then it's fairly straightforward to install and configure one. Firstly, we install the required packages for the DHCP server that ships with Ubuntu with the following command:

   ```
   $ sudo apt-get install isc-dhcp-server
   ```

8. Next, we configure our newly installed DHCP server. I'm going to use the IP range I use in my test lab as an example (10.0.1.0), but go ahead and amend the examples to suit your setup. Open the following configuration file with your preferred editor:

   `/etc/dhcp/dhcpd.conf`

   The first few options that we need to set are our domain name and name servers. By default, the configuration should look like this:

   ```
   option domain-name "example.org";
   option domain-name-servers ns1.example.org, ns2.example.org;
   ```

   We need to change that to match our setup. In my case, it looks like this:

   ```
   option domain-name "stunthamster.com";
   option domain-name-servers ns1.stunthamster.com, ns2.stunthamster.
   com;
   ```

9. Amend them to match your own domain and name servers. Next, we need to make this the authorative DHCP server for this network. Locate the line that reads:

   ```
   authoritative
   ```

   Ensure that it's uncommented. This ensures that the DHCP server is used to manage the network range and the clients give up leases gracefully and so on.

10. Finally, we can create the DHCP configuration for our network. This should be added to the bottom of the configuration file. Once again, the following example is for my network. You should substitute the values for your own IP range:

    ```
    subnet 10.0.1.0 netmask 255.255.255.0 {
     range 10.0.1.20 10.0.1.200;
     option domain-name-servers ns1.stunthamster.com;
     option domain-name "stunthamster.com";
     option routers 10.0.1.1;
     option broadcast-address 10.0.1.255;
     allow booting;
     allow bootp;
     option option-128 code 128 = string;
     option option-129 code 129 = text;
     next-server 10.0.1.11;
     filename "pxelinux.0";
     default-lease-time 600;
     max-lease-time 7200;
     }
    ```

    Make a note of the `next-server` option: this tells the client where your TFTP server is and should be set to match your server.

    Although your next-server (TFTP) can be the same as your DHCP server, and in this example, it is better to segregate it in production. Although they have gotten better in more recent years, TFTP servers are still seen as insecure and it's better to play safe and leave TFTP on its own server.

11. Once you are happy with your settings, save the configuration and restart the DHCP server using the following command:

    ```
    $ sudo service isc-dhcp-server restart
    ```

For our next task, we're going to go ahead and configure our Nginx server. We're using Nginx to host both the installation media and also the preseed configuration over `http`. Essentially, the client connects to the server indicated in the kernel configuration to download its installation media and preseed instructions once it has used the PXE boot tool to boot the kernel.

Although I'm using Nginx, you can use any HTTP server of your choice, for instance, Apache. Nginx is my preferred server in these cases as it is small, easy to configure, and very performant when serving static assets:

1. First, let's install `nginx` with the following command:

    ```
    $ sudo apt-get install nginx
    ```

2. Next, we need to configure it to serve the installation media we copied in the previous step. With your editor, open up the following configuration file:

   ```
   /etc/nginx/sites-available/default
   ```

   By default, the configuration will resemble something like the following code snippet (I've removed comments for clarity):

   ```
   server {
            listen 80 default_server;
            listen [::]:80 default_server ipv6only=on;
            root /usr/share/nginx/html;
            index index.html index.htm;
            server_name localhost;
            location /
            {
              try_files $uri $uri/ =404;
            }
          }
   ```

   Amend it to resemble the following:

   ```
   server {
            listen 80 <<BOOT SERVERNAME>>;
            listen [::]:80 <<BOOT SERVERNAME>>ipv6only=on;
            root /var/lib/tftpboot/;
            index;
            server_name <<BOOT SERVERNAME>>;
            location /
            {
              try_files $uri $uri/ =404;
            }
          }
   ```

   Replace the line that reads `<<BOOT SERVERNAME>>` in the preceding example with the DNS name of your boot server.

3. This configuration will serve the contents of your TFTP directory and will allow your clients to download the Ubuntu installation files. Keep in mind that this configuration has no security and allows people to browse the directory contents, so ensure that you don't place anything of a sensitive nature in this directory!
4. Finally, we can configure the `Preseed` file. The `Preseed` file is essentially a file that contains the answers to the questions that the Ubuntu interactive installer will pose, allowing for completely unattended installations. Let's take a look at a `Preseed` file and construct it in stages. Create the following file in your editor:

   `/var/lib/tftpboot/ks.cfg`

5. First, let's point our installer to use the local repository we created in the previous recipe:

   ```
   d-i apt-setup/use_mirror boolean true
   ```

   ```
   choose-mirror-bin | mirror/http/hostname string <HOSTNAME OF
   MIRROR>
   ```

   Change the preceding example to reflect your local mirror.

   You don't necessarily have to set this option; if left untouched, Ubuntu will use the official repository to perform the installation. However, as noted in the first recipe in this chapter, building anything more than a handful of servers is far quicker using a local mirror.

6. Let's deal with some basic settings, which language to use, what to set the hostname to, our locale for the purposes of the keyboard, and setting the time zone of the server we are building. We can do this using the following code snippet:

   ```
   d-i debian-installer/locale string en_UK.utf8
   d-i console-setup/ask_detect boolean false
   d-i console-setup/layout string UK
   d-i netcfg/get_hostname string temp-hostname
   d-i netcfg/get_domain string stunthamster.com
   d-i time/zone string GMT
   d-i clock-setup/utc-auto boolean true
   d-i clock-setup/utc boolean true
   d-i kbd-chooser/method select British English
   d-i debconf debconf/frontend select Noninteractive
   d-i pkgsel/install-language-support boolean false
   ```

7. Next, we need to tell the installer how to configure the disks on our host. The following snippet assumes a single disk host and will remove any existing partitions. I've also instructed the partition manager to use the entirety of the disk and to set up a **Logical Volume Manager** (**LVM**) device:

   ```
   d-i partman-auto/method string lvm
   d-i partman-auto/purge_lvm_from_device boolean true
   d-i partman-lvm/confirm boolean true
   d-i partman-lvm/device_remove_lvm boolean true
   d-i partman-auto/choose_recipe select atomic
   d-i partman/confirm_write_new_label boolean true
   d-i partman/confirm_nooverwrite boolean true
   d-i partman/choose_partition select finish
   d-i partman/confirm boolean true
   preseed partman-lvm/confirm_nooverwrite boolean true
   d-i partman-lvm/confirm boolean true
   d-i partman-lvm/confirm_nooverwrite boolean true
   d-i partman-auto-lvm/guided_size string max
   ```

The next set of responses deal with user management:

1. First, we need to configure our default user. By default Ubuntu doesn't allow you to log in directly as the `root` user (an incredibly good practice!), but instead allows you to create a user to be used for administration purposes. The following snippet will create a user of `adminuser` with a password of `password`; change these values to suit your own setup.

   The following example uses an encrypted password. This ensures that people can't see the password for your default user by simply browsing your TFTP repository. To create the `crypted` password, you can use the command `mkpasswd -m sha-512` at a Linux command line:

   ```
   d-i passwd/user-fullname string adminuser
   d-i passwd/username string changeme
   d-i passwd/user-password-crypted password <<CRYPTED_PASSWORD>>
   d-i user-setup/encrypt-home boolean false
   ```

2. Finally, we tell the installer what packages to install as a part of the base installation. Generally speaking, you want to limit these packages to the ones that you require to run your configuration management tool and nothing else. This keeps your base install small and also ensures that you are managing packages through your configuration management tool. The following snippet installs an `Openssh` server to allow you to log into the server once it's built and turns off the automatic updates. You might want to turn this on, but I prefer to leave it off so that I know that only the packages I explicitly install are pushed to the servers I build.

   ```
   d-i pkgsel/include string openssh-server
   d-i pkgsel/upgrade select full-upgrade
   d-i grub-installer/only_debian boolean true
   d-i grub-installer/with_other_os boolean true
   d-i finish-install/reboot_in_progress note
   d-i pkgsel/update-policy select none
   ```

   Once you're happy with your configuration, save the file.

3. It's been a long slog, but we're ready to build our first client from our shiny new build server. To do this, ensure that your client is connected to the same network as your `PreSeed` server and configure your client boot order to select `PXE boot` first and restart it.

Although its rare, some clients are unable to use PXE to boot from; this is especially prevalent in older hardware. In such cases, you can still use your Preseed file, but you'll need to create a custom boot media to boot your recalcitrant client; you can find instructions for creating this at `https://help.ubuntu.com/community/LiveCDCustomization`.

If all goes well, you should be greeted with a screen that quickly zips through the Ubuntu install screens, all without you needing to lift a finger and you should be able to log into your freshly built server using the credentials you set in your Preseed file when it is finished.

See also

We've covered a lot of ground in this recipe, and I highly encourage you to read the following documentation, both to gain a deeper understanding of how each component is configured and also to investigate the options available:

- DHCP help:

  `https://help.ubuntu.com/community/isc-dhcp-server`

- Official Ubuntu Preseed documentation:

  `https://help.ubuntu.com/14.04/installation-guide/amd64/apb.html`

- Example Preseed:

  `https://help.ubuntu.com/lts/installation-guide/example-preseed.txt`

Automating post-installation tasks

Although we can now perform unattended Ubuntu installations and save a great deal of time, we still need to configure them manually after they have been built. Ideally, we should be able to run tasks that will deal with that for us.

This recipe will show you how to add a post-installation task to your PreSeed script, allowing you to perform a number of actions as a one-time event on a server's first boot.

Getting ready

For this recipe, you should already have a configured PreSeed file.

How to do it...

We're going to add a directive to run a small script at the end of the Preseed file; this script will, in turn, create a startup script which is set to run at the first server boot. Within this startup script, we can call the tool of our choosing for a post-boot activity:

1. Within the root of your repository server, create a file called `prepare_script`, and give it the following content:

   ```
   #!/bin/sh
   ```

```
/usr/bin/curl -o /tmp/posttasks.sh http://<KICKSTART SERVER>/
post_tasks && chmod +x /tmp/posttasks.sh
cat > /etc/init.d/boottasks <<EOF
cd /tmp ; /usr/bin/nohup sh -x /tmp/posttasks.sh &
EOF
chmod +x /etc/init.d/boottasks
update-rc.d boottasks defaults
```

This script downloads a file called `posttasks.sh` from our repo server and places it into the `/tmp` directory of our newly built host. Next, it uses some simple `cat` commands to create an incredibly simple startup script. This startup simply runs the `posttasks.sh` script we placed into the `/tmp` directory.

2. Next, we update our Preseed file to run the `prepare.sh` command at the very end of the build process. We can do this using a command similar to the following snippet:

```
d-i preseed/late_command string chroot /target sh -c "/usr/bin/
curl -o /tmp/prepare.sh http://<< REPO SERVER>>/prepare_script &&
/bin/sh -x /tmp/prepare.sh"
```

3. Now, let's go ahead and create our third and final script, which will be called at boot time. On your `preseed` server, create a new file called `post_tasks`, and give it the following contents:

```
#!/bin/sh

# update apt
/usr/bin/apt-get update
/usr/bin/apt-get upgrade
reboot
```

Now when you build a host, you will find that at its first boot it will update its `apt` cache, update any installed packages, and reboot. This is perfect to ensure that all the newly built servers arrive at the same base line package version, but as you can see, using this technique you can also call out to systems, such as Ansible to do much more than simply update the host.

4
Virtualization with VMware ESXi

In this chapter, we will cover the following topics:

- Installing ESXi
- Installing and using the Vsphere Client
- Allowing SSH access to ESXi
- Creating a new guest
- Allocating resources to a guest
- Using the ESXi command line to start, stop, and destroy guests
- Managing command-line snapshots
- Tuning the host for guest performance

Introduction

Virtualization has been a cornerstone of the server landscape for some considerable time and is used both to consolidate servers to wrangle the most efficient use of the underlying hardware, and to allow for quick and easy deployment of new servers.

Virtualization has been in use for a much longer time than many systems administrators might realize, and has been used in one form or another since IBM introduced it onto their mainframes in the late 1960s. It was only in the late 1990s that virtualization started to bubble into the consciousness of the commodity server community, and VMware quickly became the market leader at that time. VMware released their first product in 1999, offering their workstation product, allowing users to run a virtual and self-contained operating system on their Windows desktop, constrained only by the hardware resources available at that time. This was just the beginning, and in 2001 VMware released their first enterprise product, VMware ESX. ESX allowed x86 server administrators to reduce the physical footprint of their server estate massively, and opened up the nascent idea of infrastructure as a service. Without virtualization, it is unlikely that the current technology landscape would look anything like it does now, and certainly technologies such as IAAS, SAAS, and SDN would not exist.

VMware did not have the market to themselves for long, and subsequently many players have entered the server virtualization market, including several open source efforts. These projects have grown sharply, both in terms of usage and in terms of their capabilities, and form the basis of IAAS offerings such as Amazon EC2 and Rackspace compute.

From the perspective of using DevOps techniques, virtualization offers one of the basic elements of an elastic and scalable environment: the ability to create, destroy, and amend hosts at will. Although you can use DevOps techniques without this elasticity, you would want to arrive at an alternative that allows you the flexibility that virtualization offers.

For this chapter, I have selected ESXi as the hypervisor of choice. Although ESXi is not open source, the hypervisor is both highly performant and free. It is also the hypervisor for many enterprises, and as such is probably a technology that you are going to come across at some point. The recipes in this chapter do not include using the vSphere products. This is the paid for management layer that offers a web-based GUI, and many advanced enterprise features such as high availability and automatic balancing of compute load.

However, there are other options available, and these can be fantastic alternatives to ESXi. Some available alternatives to choose from are:

- **KVM**: This is a hypervisor that ships as part of the Linux kernel. It is open source and has many features that ship with ESXi. KVM is ubiquitous as it ships with the Linux Kernel, and there are many interesting tools that can be used to manage it. For further information, visit `http://www.linux-kvm.org/page/Main_Page`.
- **Xen**: This is a powerful bare metal hypervisor and it is open source. Xen underpins many IAAS projects and offers many powerful features. In particular, you can choose from several vendors who offer commercial Xen offerings that can rival VMware's enterprise stack. It can be studied at `http://www.xenproject`.
- **Oracle Virtualbox**: Virtualbox is a completely open source hypervisor that offers many features. Although predominantly used as a desktop hypervisor, it can be used for enterprise. For documentation, vist `https://www.virtualbox.org`.

- **Microsoft HyperV**: Microsoft has embraced virtualization and introduced HyperV as a part of Windows Server 2008 onwards. For enterprises that already have a large Windows footprint, HyperV offers a compelling and high-performance choice of hypervisor: `http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspxg`.

Installing ESXi

Before we go any further, we need to install our virtualization software. In this case, we are going to use the latest version of VMware ESXi - at the time of writing, ESXi 6.0. Although VMware has commercial offerings, the bare metal hypervisor is both free to use and widely employed in private data centers.

Getting ready

First, you will need a place to install ESXi. This means that you are going to need some hardware that is capable of running it. ESXi supports a wide array of hardware, and most modern desktops will suffice. If you need a small workgroup server, you can find small servers such as the Dell PoweredgeT20 or the HP Micro G8 server that will suffice for small workloads of around five or six smallish VM's. Of course, if you need to support larger workloads, then ESXi will happily run on monstrous multi-processor and multi-core servers.

[You can check out the servers that are compatible at `http://www.vmware.com/resources/compatibility`.]

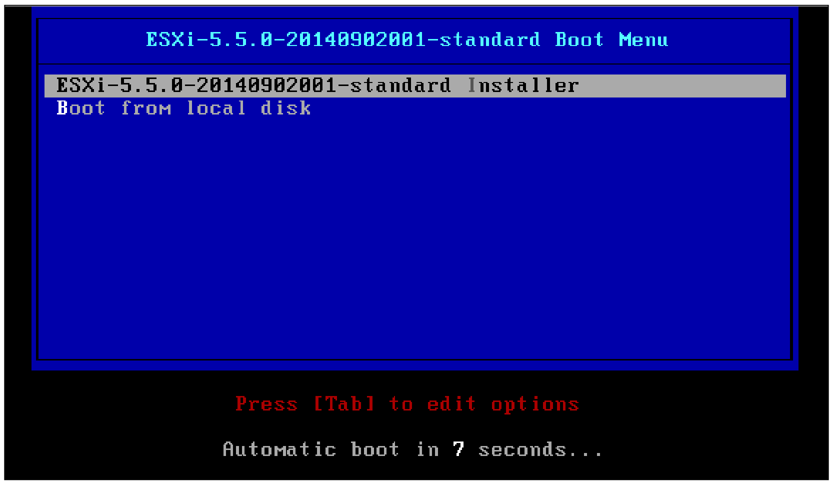
Once you have your hardware, the next step is to download the ESXi software. This can be downloaded from `www.vmware.com/go/get-free-esxi`. Once you have the ISO image, you can either burn it to a CD, or, even better, place it on a USB key, and it will be ready to install. Once your hardware and install media is ready, you will be ready to go.

[Although it is not covered here, it is both possible and an excellent idea to use PXE booting to install ESXi. You can find the details of PXE booting in the previous chapter, and how to implement it for ESXi at `https://pubs.vmware.com/vsphere-50/topic/com.vmware.vsphere.install.doc_50/GUID-4E57A4D7-259D-4FA9-AA26-E0C71487A376.html`.]

How to do it...

The following steps show you how to install and configure ESXi:

1. The first step is to boot your server from the install media. If you manage to do this successfully, you will be rewarded with a screen that looks similar to the following screenshot:

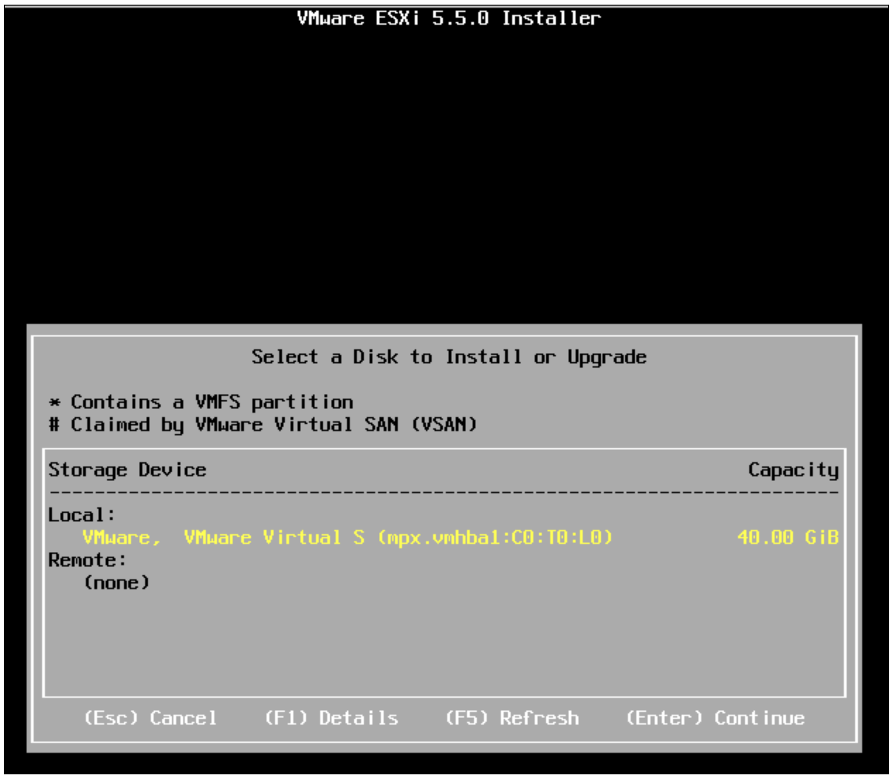


```
ESXi-5.5.0-20140902001-standard Boot Menu

ESXi-5.5.0-20140902001-standard Installer
Boot from local disk

Press [Tab] to edit options

Automatic boot in 7 seconds...
```

2. Hit return, at this screen (or let the automatic boot time out) and you will be taken to the ESXi boot-screen. Depending on the specifications of your server, this can take a few minutes, and it will tell you what stage it is at while it loads. Once the boot loader has completed the installation, you will be greeted with the installer welcome screen. Hit return, and you will be presented with the license screen. Once you read the entirety of the license and digest its many arcane terms, and eventually decide to agree with it, hit *F11* to continue.
3. Once you accept the license, the installer scans your hardware to find devices. This includes disks. It may be that your hardware uses an esoteric device that ESXi doesn't yet include, and if this is the case you can slipstream drivers onto your install media. You can find instructions on how to do that at `https://pubs.vmware.com/vsphere-51/index.jsp#com.vmware.vsphere.install.doc/GUID-78CC6C2E-E961-4A5E-B07D-0CE7083DE51E.html`. Creating your own install media can be quite a long-winded experience, but there is an alternative in the form of the ESXi-Customizer software. This presents an easy-to-use GUI that creates custom install media for you. You can find the ESXi-customizer at `http://www.v-front.de/p/esxi-customizer.html`.

Many major server vendors, such as HP and Dell, now offer ESXi pre-customized to work with their hardware, so it is worth checking the product support pages for your hardware to see if such an option is available. Not only does this enable you to use any RAID cards that ESXi might not recognize, it normally also allows finer power management and hardware features.

4. Once ESXi has finished scanning for hardware, it will present you with the disk setup page, which will resemble the following screenshot:

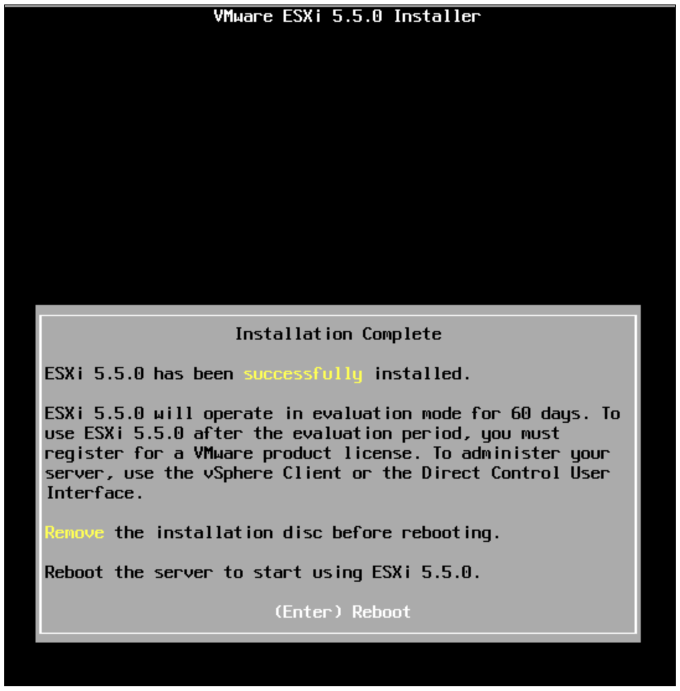


```
                    VMware ESXi 5.5.0 Installer

             Select a Disk to Install or Upgrade

 * Contains a VMFS partition
 # Claimed by VMware Virtual SAN (VSAN)

 Storage Device                                              Capacity
 ---------------------------------------------------------------------
 Local:
    VMware,  VMware Virtual S (mpx.vmhba1:C0:T0:L0)          40.00 GiB
 Remote:
    (none)

    (Esc) Cancel    (F1) Details    (F5) Refresh    (Enter) Continue
```

This screen allows you to select which partition will function as the root partition for your ESXi instance and will contain both the ESXi operating systems and will also form the partition that Guest VMs will be stored in until you add more storage. For now, select the disk you wish to install your **VMware** on, and hit *Enter*.

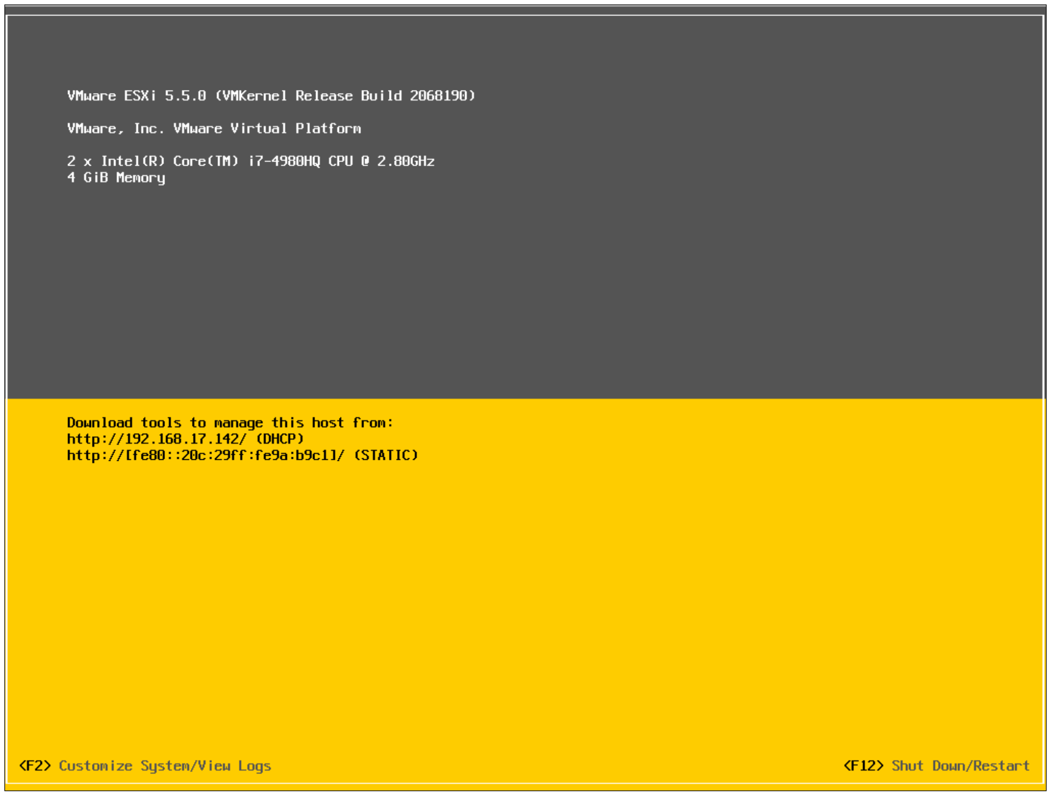
You should use the quickest disk you can lay your hands on for the partition on which the guests reside. If possible, install ESXi on one disk and then store your guest VM's on the fastest storage available. Disk latency can have a profound impact on the performance of guests.

5. Next, ESXi will prompt you to select your language. Simply select your keyboard layout and hit *Enter*. Next up, you need to set the password for your root user. This is the power user on an ESXi system, so make sure that you create a secure password. Finally, you will be prompted if you are happy with your selections. If you are, hit the *F11* key to complete the installation. The ESXi installer will now copy the files onto your selected partition and install the hypervisor. If all goes well, you should find yourself with the following success screen:



```
VMware ESXi 5.5.0 Installer

Installation Complete

ESXi 5.5.0 has been successfully installed.

ESXi 5.5.0 will operate in evaluation mode for 60 days. To
use ESXi 5.5.0 after the evaluation period, you must
register for a VMware product license. To administer your
server, use the vSphere Client or the Direct Control User
Interface.

Remove the installation disc before rebooting.

Reboot the server to start using ESXi 5.5.0.

(Enter) Reboot
```

6. Select the **Remove the installation disc before rebooting...**, hit *Enter*, and the installer will reboot the server. After a reboot, you will be greeted with the ESXi start-up screen. This will list the start-up steps, and after a short while you will come across a screen that looks something like this:



```
VMware ESXi 5.5.0 (VMKernel Release Build 2068190)

VMware, Inc. VMware Virtual Platform

2 x Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz
4 GiB Memory

Download tools to manage this host from:
http://192.168.17.142/ (DHCP)
http://[fe80::20c:29ff:fe9a:b9c1]/ (STATIC)

<F2> Customize System/View Logs                    <F12> Shut Down/Restart
```

Congratulations! You now have a bare metal hypervisor ready and waiting for you to add a new guest.

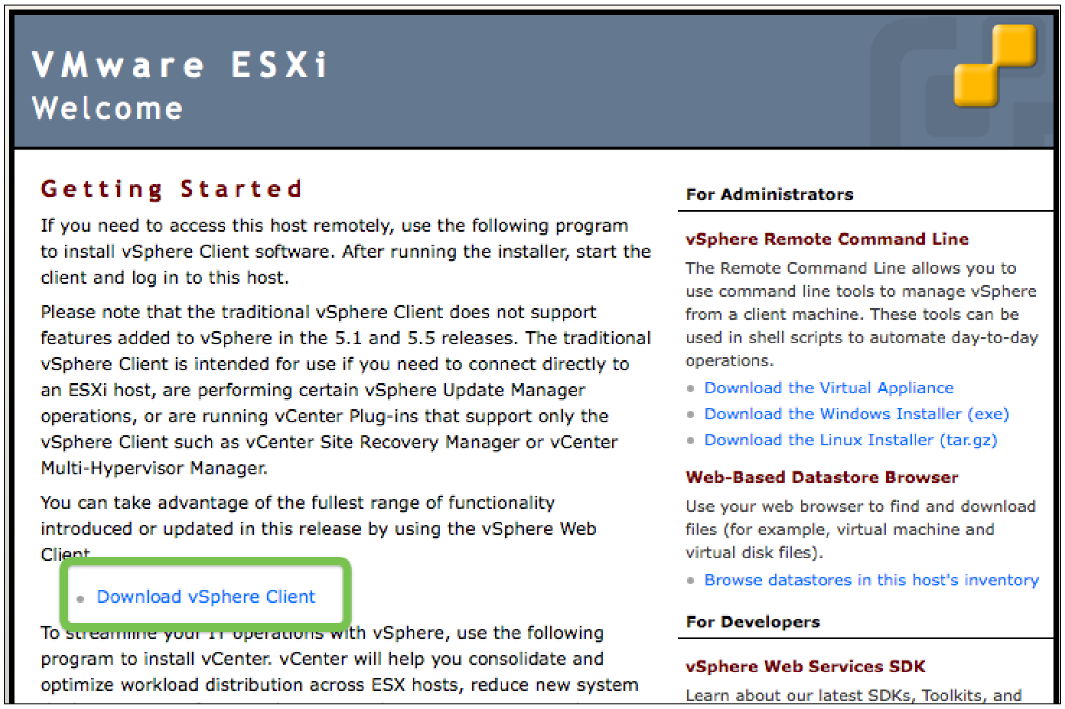
Installing and using the vSphere Client

For this chapter, we will mostly use the VMware command-line tools to carry out our tasks. Understanding the VMware CLI helps you to use SSH and scripting to automate many tasks. However, for some day-to-day tasks, having a GUI at hand can be a quick and easy way to visualize what is happening on your ESXi platform.

As you can guess from the name, the vSphere Client manages the VMware vSphere platform. The vSphere platform is VMware's commercial offering; it provides an easy method of managing a large number of hypervisors and running hundreds of guests. It can also be used as an effective GUI to control a single instance of the free ESXi hypervisor.

Getting ready

The vSphere Client can be downloaded from your new ESXi server. Use your browser to navigate to the IP address or name of your new ESXi server. You will be greeted with a page that resembles the following screenshot. Note the highlighted download link:



VMware ESXi
Welcome

Getting Started

If you need to access this host remotely, use the following program to install vSphere Client software. After running the installer, start the client and log in to this host.

Please note that the traditional vSphere Client does not support features added to vSphere in the 5.1 and 5.5 releases. The traditional vSphere Client is intended for use if you need to connect directly to an ESXi host, are performing certain vSphere Update Manager operations, or are running vCenter Plug-ins that support only the vSphere Client such as vCenter Site Recovery Manager or vCenter Multi-Hypervisor Manager.

You can take advantage of the fullest range of functionality introduced or updated in this release by using the vSphere Web Client.

- Download vSphere Client

To streamline your IT operations with vSphere, use the following program to install vCenter. vCenter will help you consolidate and optimize workload distribution across ESX hosts, reduce new system

For Administrators

vSphere Remote Command Line

The Remote Command Line allows you to use command line tools to manage vSphere from a client machine. These tools can be used in shell scripts to automate day-to-day operations.

- Download the Virtual Appliance
- Download the Windows Installer (exe)
- Download the Linux Installer (tar.gz)

Web-Based Datastore Browser

Use your web browser to find and download files (for example, virtual machine and virtual disk files).

- Browse datastores in this host's inventory

For Developers

vSphere Web Services SDK

Learn about our latest SDKs, Toolkits, and

Click on the **Download vSphere Client** to download the install file.

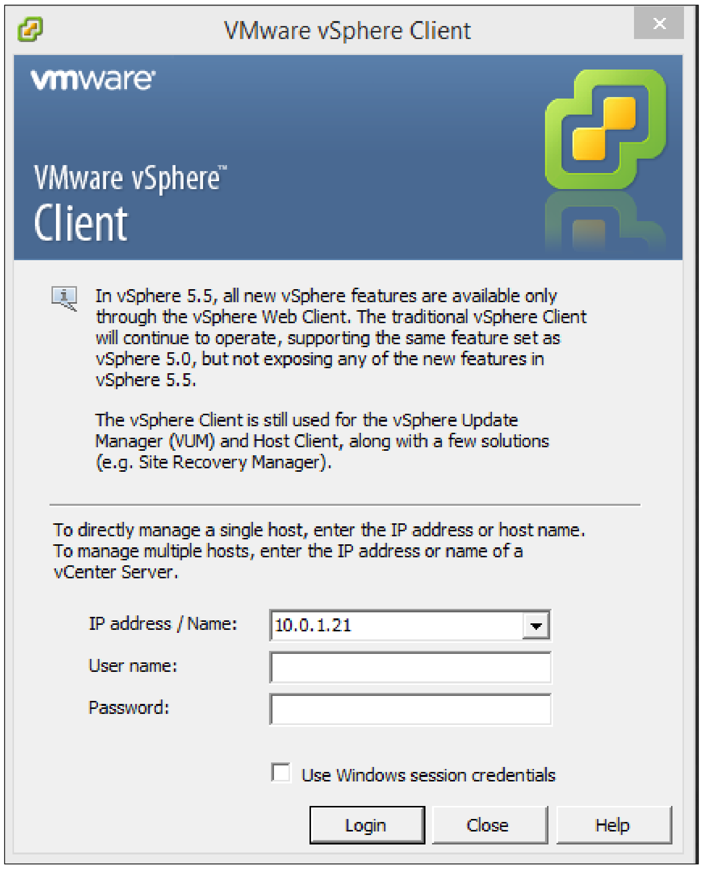
The astute among you might have already noticed that there is a single link to download the client. This is because there is only one available client: a Windows client. You might want to skip this section if you don't have a Windows desktop to install the client on. Don't worry, though: you can create and work with hosts using just the CLI.

Once you have downloaded the installer, locate it and double-click to start. It will prompt you for an install location, but unless you have a strong preference for where the software should be installed, simply click on the **next** button until the installation is completed.

How to do it....

Let's install and use the vSphere Client client:

1. Once you have installed the software, open your **Start** menu and select the **VMware vSphere Client**. You will be presented with a login screen that resembles the following screenshot:



In vSphere 5.5, all new vSphere features are available only through the vSphere Web Client. The traditional vSphere Client will continue to operate, supporting the same feature set as vSphere 5.0, but not exposing any of the new features in vSphere 5.5.

The vSphere Client is still used for the vSphere Update Manager (VUM) and Host Client, along with a few solutions (e.g. Site Recovery Manager).

To directly manage a single host, enter the IP address or host name. To manage multiple hosts, enter the IP address or name of a vCenter Server.

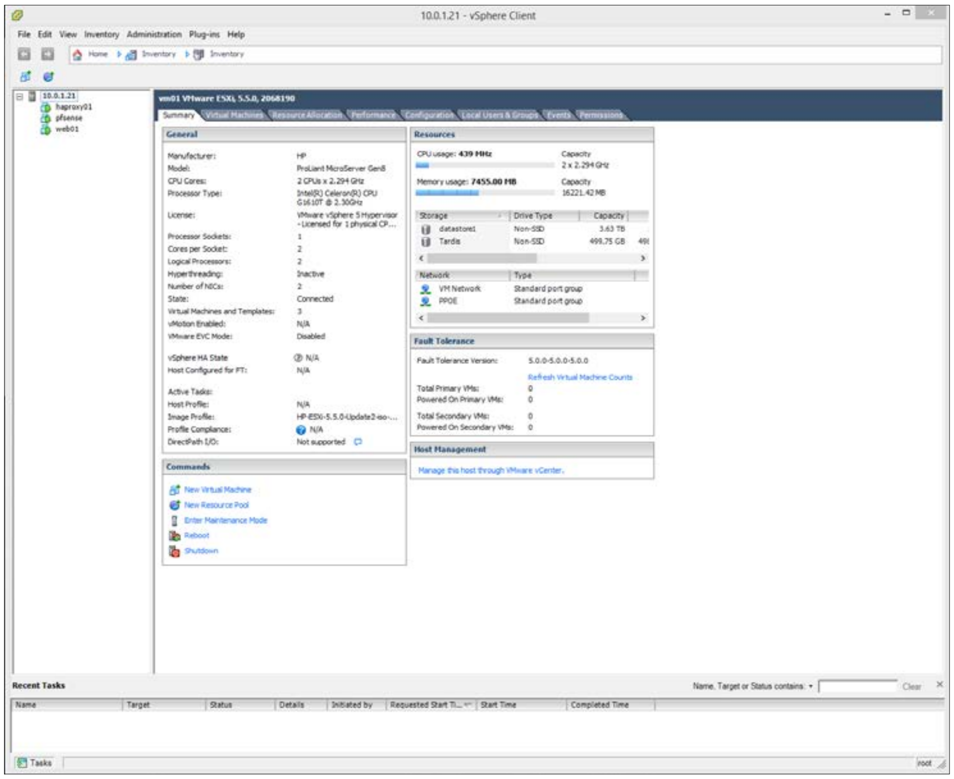
IP address / Name: 10.0.1.21

User name:

Password:

☐ Use Windows session credentials

2. Enter the IP address or name of your ESXi server in the `IP address` field. In the `User name` and `Password` fields, you need to enter the details you entered when you set up your ESXi host (remember, the default admin user is `root`). You will be taken to your ESXi inventory when you click the **Login** button. Your inventory will be blank, but your hosts will appear on the left hand side as soon as you enter some. Here are mine as an example:


10.0.1.21 - vSphere Client
File Edit View Inventory Administration Plug-ins Help
Home Inventory Inventory
10.0.1.21
haproxy01
pfsense
web01
vm01 VMware ESXi, 5.5.0, 2068190
Summary Virtual Machines Resource Allocation Performance Configuration Local Users & Groups Events Permissions
Commands
New Virtual Machine
New Resource Pool
Enter Maintenance Mode
Reboot
Shutdown
Recent Tasks
Name, Target or Status contains:
Clear
Name Target Status Details Initiated by Requested Start Ti... Start Time Completed Time
Tasks
root

General

Manufacturer:	HP
Model:	ProLiant MicroServer Gen8
CPU Cores:	2 CPUs x 2.294 GHz
Processor Type:	Intel(R) Celeron(R) CPU G1610T @ 2.30GHz
License:	VMware vSphere 5 Hypervisor - Licensed for 1 physical CP...
Processor Sockets:	1
Cores per Socket:	2
Logical Processors:	2
Hyperthreading:	Inactive
Number of NICs:	2
State:	Connected
Virtual Machines and Templates:	3
vMotion Enabled:	N/A
VMware EVC Mode:	Disabled
vSphere HA State	N/A
Host Configured for FT:	N/A
Active Tasks:	
Host Profile:	N/A
Image Profile:	HP-ESXi-5.5.0-Update2-iso-...
Profile Compliance:	N/A
DirectPath I/O:	Not supported

Resources

CPU usage: **439 MHz** Capacity 2 x 2.294 GHz

Memory usage: **7455.00 MB** Capacity 16221.42 MB

Storage	Drive Type	Capacity
datastore1	Non-SSD	3.63 TB
Tardis	Non-SSD	499.75 GB

Network	Type
VM Network	Standard port group
PPOE	Standard port group

Fault Tolerance

Fault Tolerance Version:	5.0.0-5.0.0-5.0.0
	Refresh Virtual Machine Counts
Total Primary VMs:	0
Powered On Primary VMs:	0
Total Secondary VMs:	0
Powered On Secondary VMs:	0

Host Management

Manage this host through VMware vCenter.

We're not going to spend any more time dwelling on the vSphere Client, although it's useful to know that it's available; however, due to its limitations, we are going to concentrate on the ESXi CLI. This allows you to manage your ESXi server using an SSH client and, more importantly, once you understand how to use it to carry out tasks, it allows you to use SSH to automate the tasks.

Allowing SSH access to ESXi

ESXi ships with a minimal Userspace client to allow you to interact directly with the ESXi server. This offers you the ability to use the command line to manage crucial tasks such as creating new hosts, deleting hosts, and adjusting configuration. Access is enabled via SSH, allowing you to use an existing SSH client to log on.

It is incredibly important that you keep this secure. By allowing SSH to the ESXi server, you are opening a new attack surface, and if a malicious user gains access it will have a full control over the underlying hypervisor. If you do allow SSH access, I strongly recommend ensuring that it is heavily firewalled from general access.

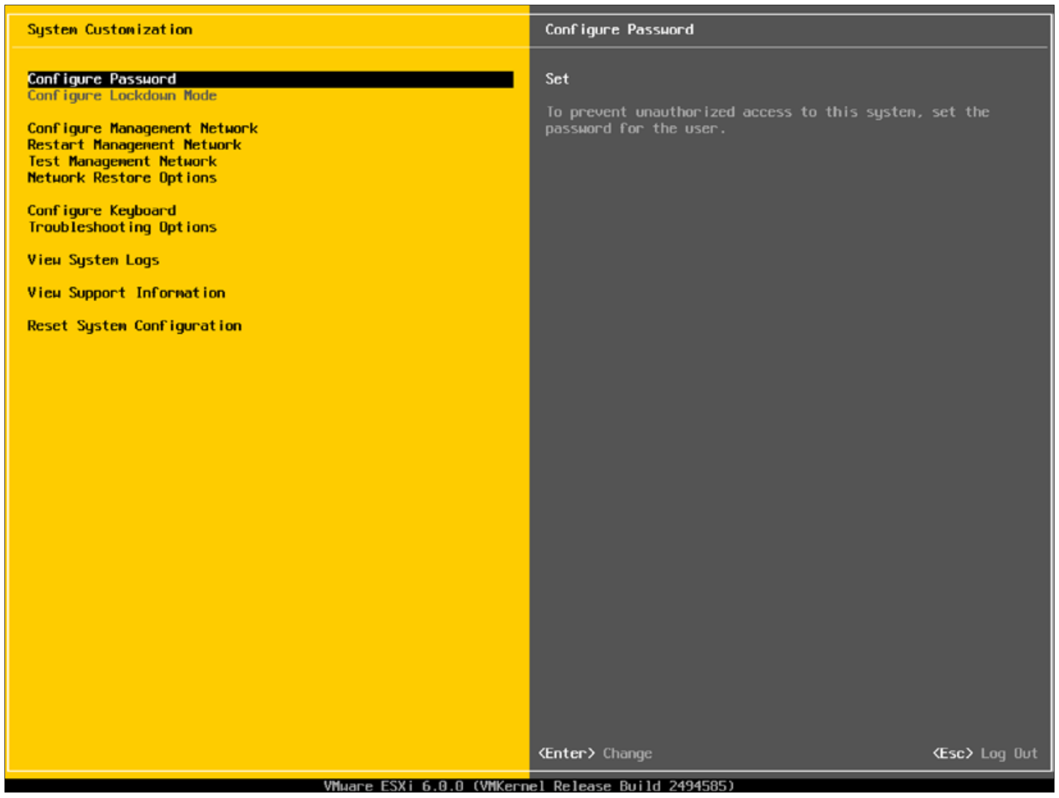
Getting ready

To use this recipe, you will need an ESXi installed host.

How to do it...

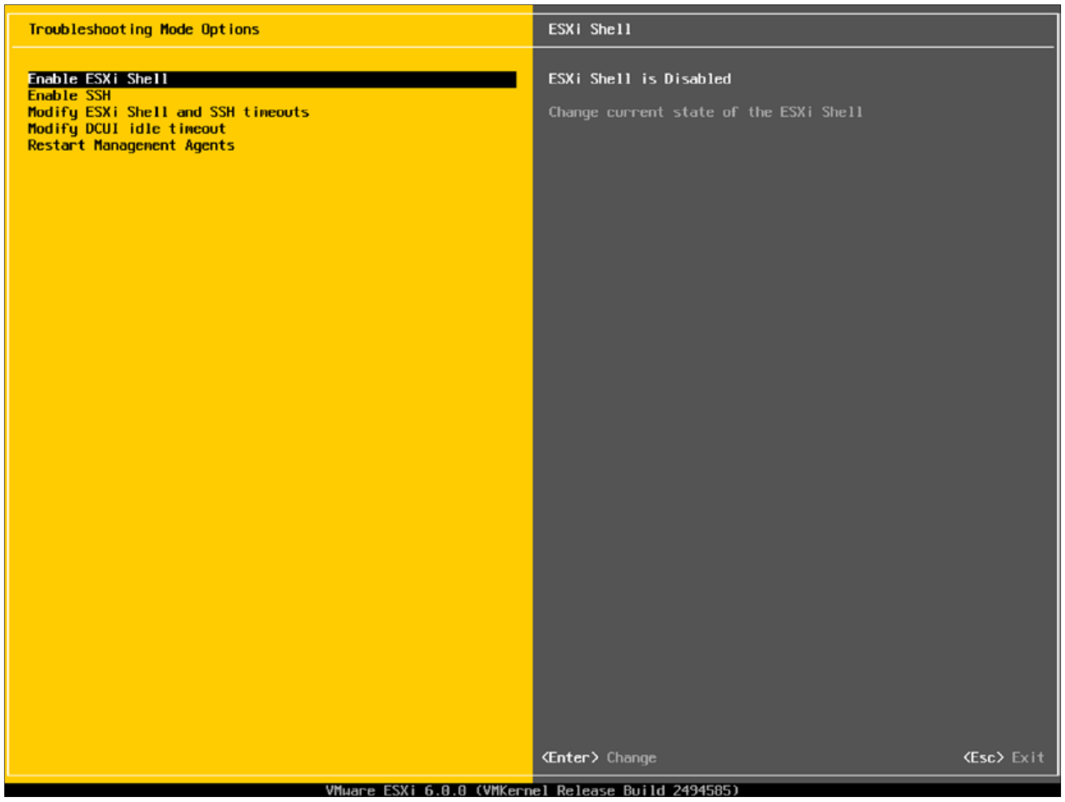
Let's configure and allow SSH to the EXSi server:

1. On the ESXi console, press *F2* to bring up the options page. This should look similar to the following screenshot:



```
System Customization                    Configure Password

Configure Password                      Set
Configure Lockdown Mode
                                        To prevent unauthorized access to this system, set the
Configure Management Network            password for the user.
Restart Management Network
Test Management Network
Network Restore Options

Configure Keyboard
Troubleshooting Options

View System Logs

View Support Information

Reset System Configuration

                                        <Enter> Change                    <Esc> Log Out
             VMware ESXi 6.0.0 (VMKernel Release Build 2494585)
```

2. Select the option named **Troubleshooting Mode Options**. This should reveal a screen that resembles the following screenshot:



Troubleshooting Mode Options

Enable ESXi Shell
Enable SSH
Modify ESXi Shell and SSH timeouts
Modify DCUI idle timeout
Restart Management Agents

ESXi Shell

ESXi Shell is Disabled

Change current state of the ESXi Shell

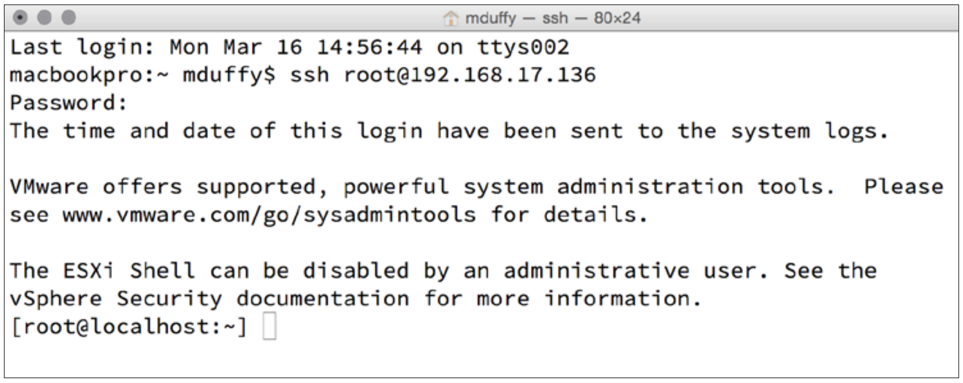
<Enter> Change
<Esc> Exit

VMware ESXi 6.0.0 (VMKernel Release Build 2494585)

3. Set **Enable ESXiSSH** to `true` and exit this screen.

You may notice that there is both an ESXi shell and SSH. SSH allows remote access, and the shell allows you to interact with the hypervisor from the console. For this recipe, we only need to enable the SSH access.

4. To test that you have SSH access, use a SSH client to connect to your ESXi host as if it were any other host. Use the login details you set when you installed your ESXi host (if in doubt, the username should be root and the password will be what you set). You should see a shell prompt that looks similar to the following screenshot:



mduffy — ssh — 80×24

```
Last login: Mon Mar 16 14:56:44 on ttys002
macbookpro:~ mduffy$ ssh root@192.168.17.136
Password:
The time and date of this login have been sent to the system logs.

VMware offers supported, powerful system administration tools.  Please
see www.vmware.com/go/sysadmintools for details.

The ESXi Shell can be disabled by an administrative user. See the
vSphere Security documentation for more information.
[root@localhost:~]
```

Creating a new guest

Virtualization offers you the ability to create new virtual guests quickly and easily on the underlying ESXi host. You can create your hosts using a GUI, but it's possible to define and create hosts entirely from the command line.

This recipe shows you how to achieve this by creating a VMware configuration file and importing it using the ESXi shell.

Getting ready

You'll need an ESXi host that has SSH enabled.

How to do it...

Let's create a new guest using SSH onto your ESXi host:

1. SSH onto your ESXi host, and change directory to your default data store using the following command:

   ```
   cd /vmfs/volumes/datastore1
   ```

2. Next, you need to create a place to hold the VM files. You can do this by using the `mkdir` command:

   ```
   mkdir examplevm
   ```

3. Now, you need to create the `configuration` file that we are going to use to construct our guest. Use `vi` to create a new file called `examplevm.vmx`, and insert the following content:

```
config.version = "8"
virtualHW.version = "11"
memsize = "256"
numvcpus = "1"
cpuid.coresPerSocket = "1"
floppy0.present = "false"
displayName = "newVM"
guestOS = "linux"
ide0:0.present = "TRUE"
ide0:0.deviceType = "cdrom-raw"
ide:0.startConnected = "false"
Ethernet0.present = "TRUE"
Ethernet0.connectionType = "monitor_dev"
Ethernet0.networkName = "VM Network"
Ethernet0.addressType = "vpx"
scsi0.present = "true"
scsi0.virtualDev = "pvscsi"
scsi0:0.present = "true"
scsi0:0.fileName = "newvm.vmdk"
scsi0:0.deviceType = "scsi-hardDisk"
```

4. Let's take a look at some of the important values from the example:

   virtualHW.version: The virtual hardware defines the capabilities of the underlying hypervisor. More details can be found at `http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1003746`.

   memsize: This defines the RAM allocated to the guest and is expressed in megabytes.

   scsi0:0.fileName: This is the name of the disk that we will create in the next step.

   numvcpus: This sets the number of physical CPUs that are offered to the virtual host.

   cpuid.coresPerSocket: This setting defines how many CPU-cores are allocated per virtual CPU.

Frustratingly, VMware does not document the various options for the VMX files. You can find more details by searching online on various forums, but the best way to find new settings is to create a host by using the GUI and then editing the resulting `.vmx` file.

Next, we need to create a new virtual disk for this guest:

1. You can do this using the following command:

   ```
   vmkfstools –c -a pvscsi -d zeroedthick4gnewvm.vmdk
   ```

   This will create a new `VMDK` file called `newvm` that is sized at four gigabytes and with a virtual Paravirtual SCSI interface.

> You can find the usage details for the `vmkfstools` utility at `https://pubs.vmware.com/vsphere-51/topic/com.vmware.vsphere.storage.doc/GUID-A5D85C33-A510-4A3E-8FC7-93E6BA0A048F.html`.

2. Next, we will register the machine with ESXi. This makes it available to power on and start to install. At the ESXi command line, issue the following command:

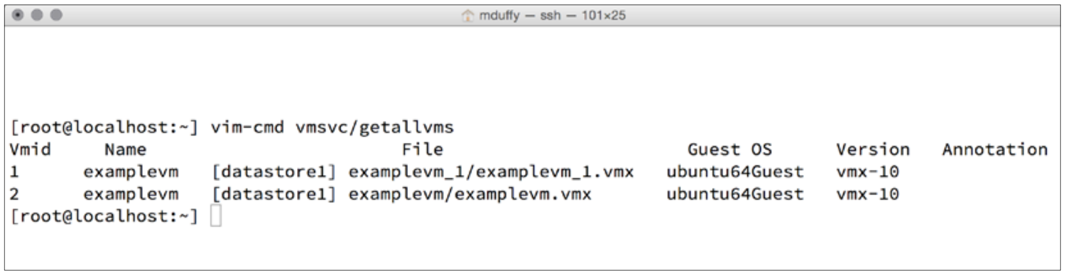
   ```
   vim-cmd -s register /vmfs/volumes/datastore1/example.vmx
   ```

   This will parse the VMX file and register the new host. If you are able to access a GUI, you will see your new guest listed as powered off.

3. Finally, we will start the Guest VM using the following command. To do this, we require the inventory ID of the machine. You can find it with the following command:

   ```
   vim-cmdvmsvc/getallvms
   ```

   This should produce an output similar to the following screenshot:



```
[root@localhost:~] vim-cmd vmsvc/getallvms
Vmid     Name                      File                          Guest OS      Version   Annotation
1      examplevm   [datastore1] examplevm_1/examplevm_1.vmx   ubuntu64Guest   vmx-10
2      examplevm   [datastore1] examplevm/examplevm.vmx       ubuntu64Guest   vmx-10
[root@localhost:~]
```

4. Note the `Vmid` on the left-hand side. You can start the VM using the ID in the following command:

   ```
   vim-cmdvmsvc/power.on<Vmid>
   ```

   This will then power-on the VM and make it available for use.

Allocating resources to a guest

Now that we have the ability to create virtual machines, we also need a way to amend them. One of the huge advantages of virtualization is the ability to amend the specification of the virtual machines very quickly and easily. Need more RAM? No problem. Need a few more CPU cores? Easily done. The flexibility of virtual hardware is one of its biggest selling points and one of its most powerful features. In this recipe, we'll take a look at how we can amend an existing host to add more RAM and more processors.

Getting ready

You'll need an ESXi host with remote access enabled and a virtual host that you wish to amend.

How to do it...

Let's allocate resources to the guest:

1. SSH onto the ESXi host and locate the `.vmx` file of the virtual machine you wish to amend. Open it using the following command:

   ```
   vi <virtualmachine>.vmx
   ```

   This will open up the configuration file for the virtual machine, and it will look similar to this snippet:

   ```
   scsi0.present = "TRUE"
   scsi0:0.deviceType = "scsi-hardDisk"
   scsi0:0.fileName = "examplevm.vmdk"
   scsi0:0.present = "TRUE"
   ethernet0.virtualDev = "e1000"
   ethernet0.networkName = "VM Network"
   ethernet0.addressType = "generated"
   ethernet0.wakeOnPcktRcv = "FALSE"
   ethernet0.present = "TRUE"
   displayName = "examplevm"
   guestOS = "ubuntu-64"
   toolScripts.afterPowerOn = "TRUE"
   ```

2. To add more RAM, locate the line called `memSize` in the `.vmx` file and change it to the desired size in megabytes. For instance, for a 16GB VM, I'd amend it to reflect the following value:

   ```
   memSize = "16384"
   ```

3. To amend the number of virtual sockets and cores the virtual machine has, you need to amend the `numvcpus` and `cpuid.coresPerSocket` values respectively. For instance, to create a powerful machine that has four sockets, each with two cores, you would insert the following configuration:

   ```
   numvcpus = "8"

   cpuid.coresPerSocket = "4"
   ```

   > Be careful with what you set here: from a common-sense perspective, it makes no sense to create a machine that has 50 cores and 2TB of RAM if you are intending to run it on a host that only has two cores and 16GB of RAM.

4. You will need to restart the virtual machines for the changes to come into effect.

Using the ESXi command line to start, stop, and destroy guests

Now that we are able to create and amend our VMs, it's time to look at some of the more basic management tasks; that is, how we stop, start, and destroy them. This is especially useful to know in a command line, as it opens up the possibility to use scripting to manage the lifecycle of a VMware guest.

Getting ready

You will need an ESXi host with SSH access and a VMware guest that is ready to be started.

How to do it...

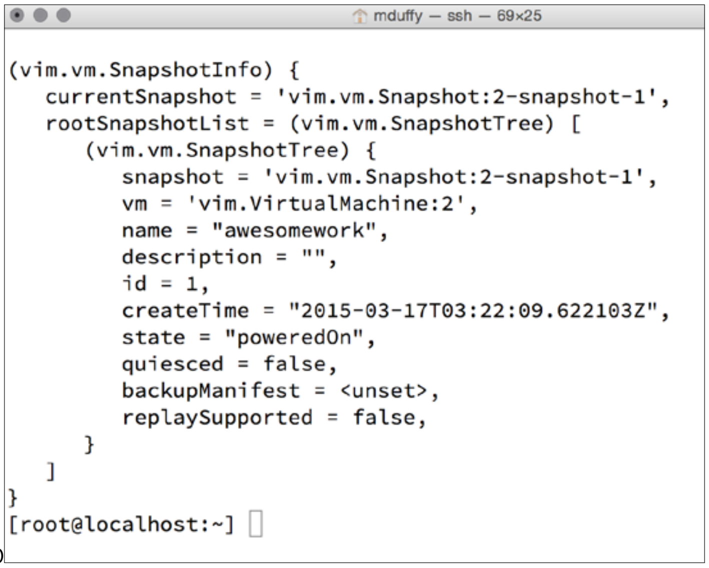
Let's use the ESXi command to start or stop a virtual machine:

1. Before we start or stop a virtual machine, we should first ascertain its current state. This can be done using the following command on the ESXi host:

   ```
   esxclivm process list
   ```

   > You can find further details of the `esxcli` command at `https://pubs.vmware.com/vsphere-60/index.jsp?topic=%2Fcom.vmware.vcli.ref.doc_50%2Fesxcli_vm.html`.

This will give an output similar to the following screenshot:



```
(vim.vm.SnapshotInfo) {
   currentSnapshot = 'vim.vm.Snapshot:2-snapshot-1',
   rootSnapshotList = (vim.vm.SnapshotTree) [
      (vim.vm.SnapshotTree) {
         snapshot = 'vim.vm.Snapshot:2-snapshot-1',
         vm = 'vim.VirtualMachine:2',
         name = "awesomework",
         description = "",
         id = 1,
         createTime = "2015-03-17T03:22:09.622103Z",
         state = "poweredOn",
         quiesced = false,
         backupManifest = <unset>,
         replaySupported = false,
      }
   ]
}
[root@localhost:~]
```

0

2. Once you have the World ID of the virtual machine, you can use the ESXi command line to stop it. This can be done using the following command:

   ```
   $ esxclivm process kill --type=soft --world-id=<worldID>
   ```

   The three options that you can use to stop the virtual machine are **soft**, **hard** and **force**. Soft denotes a standard shutdown, hard essentially simulates yanking a power plug out of the back, and force should be used only as a last resort, as ESXi will immediately kill the process with no attempt at a graceful shutdown. This can leave the guest in an indeterminate and possibly corrupt state.

3. To start a VM, first list the available VMs using the following command:

   ```
   $vim-cmdvmsvc/getallvms
   ```

   Once you've found the VM you are interested in, you can start it using the following command:

   ```
   $ vim-cmdvmsvc/power.on<VMID>
   ```

Managing command-line snapshots

One of the huge advantages of using virtual machines is the ability to use snapshots to take a point-in-time image of a guest. This allows you to take risks, since as long as you have a snapshot to return to, you know that you can easily go back to a known good state.

Snapshotting is one of the prime candidates for scripting. You should take a snapshot every time you carry out an operation on a virtual machine. Installed a new package? Take a snapshot. Changed some settings? Take a snapshot.

Although I encourage you to use snapshots freely, you should also have a strategy for removing them. ESXi essentially keeps a set of difference files when it takes snapshots, and these can very quickly eat disk space if left to grow with no removal strategy. I personally append a date to the name of each snapshot and routinely remove any that are over a month old.

Getting ready

For this recipe, you will need an ESXi host and a guest VM.

How to do it...

Let's manage the command-line snapshot:

1. First, find the `id` of the VM that you wish to examine for snapshots. To do this, use the following command:

   ```
   $ vim-cmdvmsvc/getallvms
   ```

2. Using the `id` of the VM, issue the following command to take a snapshot of the guest:

   ```
   $ vim-cmdvmsvc/snapshot.create<vmid><snapshot
   name><"description"><include memory><quiesced>
   ```

3. For example, the command will look similar to this:

   ```
   $ vim-cmdvmsvc/snapshot.create 2 test1"A test" 1 1
   ```

Generally, you want both include memory and quiescence to be set to `true`. If you set include memory to zero, then you lose the ability to take a booted snapshot. When you restore the snapshot, it will instead reboot the machine. Quiescing the file system of the guest allows the VMware tools to order the underlying file system in a suitable manner for backups of the snapshot.

Once you have issued this command, you will have a new snapshot of your guest.

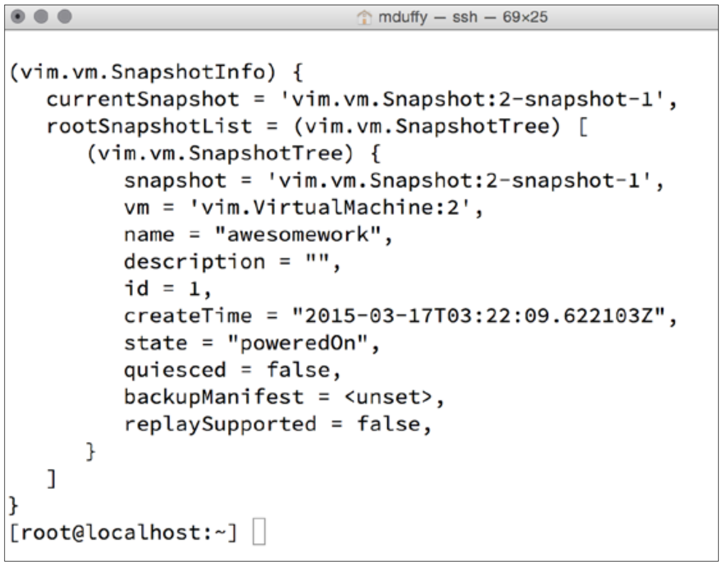
4. Now that we have the snapshots, we need to be able to work with them. First, find the ID of the VM you wish to examine for snapshots. To do this, use the following command:

   ```
   $ vim-cmdvmsvc/getallvms
   ```

5. To see a list of snapshots present on the guest, issue the following command using the ID of the VM:

   ```
   $ vim-cmdvmsvc/get.snapshotinfo
   ```

   This should produce an output that looks similar to the following screenshot:



```
(vim.vm.SnapshotInfo) {
   currentSnapshot = 'vim.vm.Snapshot:2-snapshot-1',
   rootSnapshotList = (vim.vm.SnapshotTree) [
      (vim.vm.SnapshotTree) {
         snapshot = 'vim.vm.Snapshot:2-snapshot-1',
         vm = 'vim.VirtualMachine:2',
         name = "awesomework",
         description = "",
         id = 1,
         createTime = "2015-03-17T03:22:09.622103Z",
         state = "poweredOn",
         quiesced = false,
         backupManifest = <unset>,
         replaySupported = false,
      }
   ]
}
[root@localhost:~]
```

   The preceding output tells us quite a lot, but the key values are `name` and `createTime`. These tell us the name of the snapshot and the date it was created.

6. Now that we have listed the snapshots on the guest, we can tidy them. Using the following command, find the ID of the VM that you wish to denude of its snapshots:

   ```
   $ vim-cmdvmsvc/getallvms
   ```

7. Once you have found the VM you wish to strip of snapshots, you can remove all the snapshots using the following command:

   ```
   $ vim-cmdvmsvc/snapshot.removeall<VMID>
   ```

Note that `<VMID>` is the `id` of the VM you wish to clear down. This will remove all the snapshots from the target machine, and can be especially useful if a guest has built up a great many snapshots.

Tuning the host for guest performance

Although ESXi ensures that the guest OS believes it is running on real hardware, it is of course running on a time-shared virtual system. Although the hypervisor does its best to mask this to the guest operating system, there are steps that you can take to help ensure the best performance.

This recipe will show you how to tune the performance of an Ubuntu guest OS. However, you can also find other tuning guides, for example, for Windows.

Getting ready

For this recipe, you will need an ESXi host and an Ubuntu 14.04 guest.

How to do it...

First, you need to install the VMware `guest-tools`. The OpenVM tools are an open source version of the official VMware tools, and they facilitate better memory management and network performance:

1. To install them, use the following command:

   ```
   $ sudo apt-get install open-vm-tools
   ```

   This will install the command-line version of the OpenVM tools and start them.

2. Time synchronization is vital for many systems, especially if they deal with SSL certificates (extreme time drift can invalidate the certificate exchange). Due to the time-sharing nature of hypervisors, time drift can be especially severe, and although the VM tools can synchronize time with the ESXi host, best practice is to use NTP. To setup `ntp`, install the `ntp` client using the following command:

   ```
   $ sudo apt-get install ntp
   ```

3. To set the NTP servers that you synchronize with, edit the `/etc/ntp.conf` file. By default, it should contain the following:

   ```
   server 0.ubuntu.pool.ntp.org
   server 1.ubuntu.pool.ntp.org
   server 2.ubuntu.pool.ntp.org
   server 3.ubuntu.pool.ntp.org
   ```

Replace the server listing with servers that are applicable to your environment. Generally, it's best to run a local NTP server that all your servers can set time by.

[If want to use alternative servers to the default Ubuntu ones and do not yet have an internal NTP server, than you can find a list of publically available NTP servers at `http://www.pool.ntp.org`.]

See also

I highly recommend you read the official ESXi documentation. You can find it at `https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-6-pubs.html`.

5
Automation with Ansible

In this chapter, we are going to cover the following topics:

- Installing Ansible
- Creating a scaffold Playbook
- Creating a common role
- Creating a webserver using Ansible and Nginx
- Creating an application server role using Tomcat and Ansible
- Installing MySQL using Ansible
- Installing and managing HAProxy with Ansible
- Using ServerSpec to test your Playbook

Introduction

Automation is one of the defining techniques of DevOps engineers, and it's not entirely without reason. Both developers and operators have long made use of automation methods to provide services, with operators using automation to define server configuration, and developers automating software build activities. However, until recently, there have been limited tools for server definition. The venerable CF Engine was the incumbent for many years, which offered system administrators the ability to use **DSL** (**Domain Specific Language**) to define the state of a system for the first time, rather than a suite of custom shell scripts.

Since then, the configuration management marketplace has exploded, with new tools being announced seemingly on a monthly basis. Several front runners have emerged however, and it seems that (at the time of writing) the competition for the hearts and minds of developers is between CFEngine, Chef, Puppet, Salt, and Ansible. These tools have the majority of the market share and are in popular usage.

Ansible in particular has seen huge growth. Based around a simple syntax which utilizes the **YAML** (**Yet Another Markup Language**) format, and not requiring a complex server and client arrangement, it provides a very low barrier to entry to automation. These recipes will use Ansible to create and manage some popular software packages and demonstrate the power and simplicity of Ansible automaton.

Installing Ansible

Before we go any further, we will need to install Ansible. As mentioned in the introduction, Ansible is simple to install, and does not require a complex master/slave arrangement. A package for Ansible is available for Linux and MacOS, and the source code is readily available from Github should you wish to try the latest and greatest release. We are going to use the `package` method to install Ansible on Ubuntu 14.04.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host.

How to do it...

Installing the latest stable version of Ansible is easy, as the Ansible project maintains an Ubuntu PPA. This allows you to use the `apt` package manager to install it:

1. First, add the required software using the following command:

   ```
   $ sudo apt-get install software-properties-common
   ```

   Now, add the Ansible PPA repository:

   ```
   $ sudo apt-add-repository ppa:ansible/ansible
   ```

2. Update your `apt` repository to ensure that the Ansible repository is up to date:

   ```
   $ sudo apt-get update
   ```

3. Finally, use `apt` to install the Ansible software:

   ```
   $ sudo apt-get install ansible
   ```

4. You can test if Ansible is successfully installed by running the following command:

   ```
   $ ansible --version
   ```

See also

The detailed installation guide for Ansible is available at `http://docs.ansible.com/intro_installation.html`.

Creating a scaffold Playbook

Ansible Playbooks are a means to organize Ansible tasks (or Plays as they are known in the Ansible world) so that they can be applied to groups of servers. Within each Playbook, you will find a set of roles; roles contain a combination of Ansible code, variables, and potentially files and templates. These are often organized along the lines of functions such as an Apache web server or MySQL. For instance, you might have a Playbook that is used to create web applications; the web application requires a MySQL server, Apache Web server, and Tomcat server. The Playbook contains the list of tasks to be run against each server that makes up the service, and the roles contain the code that actually implements the configuration.

A Playbook that has been written correctly can be used to set up a system in many different environments; this means that you can use the same Playbook to set up a system in the Dev, UAT, and production environments. With judicious use of templates and variables, you can keep your code small and concise, but still have it set up your environment wherever it may be.

In this recipe, we will create a `bash` script to set up a scaffold Playbook. This skeleton is based largely on Ansible best practices (`http://docs.ansible.com/playbooks_best_practices.html`), with some additional tweaks for maintaining multiple environments. This is an opinionated script, which assumes that you will have a self-contained inventory and a `dev`, `uat` and `production` environment; feel free to tweak it to make it more suited to your purpose.

Getting ready

You'll need an Ubuntu 14.04 client with Ansible installed.

How to do it

The steps that follow will show you how to create a `bash` script that will create the layout for our template Playbook.

The Playbook layout the script will create will be as follows:

```
<Playbookname>
  files/
    <environments>/
  group_vars/
    <environments>/
  inventories/
  roles/
<Playbook_name.yml>
```

This creates everything that is required for a Playbook, including an inventory and a place to store variables for different environments:

1. Using your editor, create a new file called `playbookscaffold.sh`.
2. Edit the `playbookscaffold.sh` file and insert the following code snippet:

```
#!/usr/bin/env bash
# Display usage instructions
usage() { echo "Usage: $0 [-p <Playbook Path>] [-t <Playbook
Title>]" 1>&2; exit 1; }
# Gather the users options
while getopts ":p:t:" OPTION; do
    case "${OPTION}" in
        p)
            PROJECT_PATH=${OPTARG}
            ;;
        t)
            PLAYBOOK_TITLE=${OPTARG}
            ;;
        *)
            usage
            ;;
    esac
done
# If the user missed a switch, get them remind them that
# they need to add it.
if [ -z ${PROJECT_PATH} ]; then
echo "You need to supply a Project Path"
exit 1
fi
if [ -z ${PLAYBOOK_TITLE} ]; then
echo "You need to supply a Project Title"
exit 1
fi
```

```
# Now we have the path and title, build the layout
mkdir -p "${PLAYBOOK_PATH}/files"
mkdir -p "${PLAYBOOK_PATH}/group_vars"
mkdir -p "${PLAYBOOK_PATH}/host_vars/dev"
mkdir -p "${PLAYBOOK_PATH}/host_vars/uat"
mkdir -p "${PLAYBOOK_PATH}/host_vars/prd"
mkdir -p "${PLAYBOOK_PATH}/inventories"
mkdir -p "${PLAYBOOK_PATH}/roles"
# Use Ansible galaxy init to create a default 'common' role
ansible-galaxy init common -p "${PLAYBOOK_PATH}/roles/"
touch "${PLAYBOOK_PATH}/inventories/dev"
touch "${PLAYBOOK_PATH}/inventories/uat"
touch "${PLAYBOOK_PATH}/inventories/prd"
touch "${PLAYBOOK_PATH}/${PLAYBOOK_TITLE}.yml"
```

3. You should now be able to run the script, supply it with a path and title, and have a new skeleton Playbook.

> The `ansible-galaxy` command can be used for more besides creating the skeleton role and is the best tool to install roles from the Ansible Galaxy role repository. More details can be found at `http://docs.ansible.com/galaxy.html#the-ansible-galaxy-command-line-tool`.

As you can see, this script is opinionated and it assumes that you are going to have three different environments named `dev`, `uat` and `production`. This is a standard pattern, but you can amend it to fit your own particular environment. The Playbooks that this skeleton creates are perfect for collaboration, as they contain literally everything that is required, from the inventory through to the variables that each environment requires. If you add a new environment, you can simply add a new directory to hold the variables, populate it, and be up and running.

Creating a common role

Now that we have a way to create our scaffold Playbook we can go ahead and create our first role; this role will create users, add SSH keys, and install software. I tend to find it invaluable on any server I am managing.

Getting ready

For this recipe, you need an Ubuntu 14.04 server to act as an Ansible client, and an Ubuntu 14.04 server that you wish to configure.

How to do it...

Let's create a common role:

1. First, create a new `playbook` using our scaffold script, which we created in the preceding recipe:

   ```
   $ playbookscaffold.sh -p . -t "first_playbook"
   ```

2. Edit `first_playbook/roles/common/tasks/main.yml` and insert the following code snippet:

   ```
   # tasks file for common
   - include: create_users.yml
   ```

   The `include` statement tells Ansible to parse the included file before moving on to the next statement. Includes are a great way to organize complex sets of tasks within roles and I encourage you to use them in your own efforts; not only do they split up large chunks of code, they also make it very readable for anyone maintaining the role. By looking in the `main.yml` file and seeing the `includes`, they can see a complete list of the major activities the role will perform.

3. Next, navigate to `first_playbook/roles/common/tasks` and create a new file called `create_users.yml`. We're going to add some Ansible code to create new users. To achieve this, we are going to use a YAML dictionary to describe the users, which are then used with a `with_dict` declaration to loop through. This has two benefits: first, it keeps the code small and readable, and second, it abstracts the data from the code. It means that you can have separate user lists depending on your environment.

   You can find more information on loops on the Ansible site at `http://docs.ansible.com/playbooks_loops.html`.

4. Edit `first_playbook/roles/common/create_users.yml` and insert the following code:

   ```
   - name: Add users
     user: name={{ item.key }} state={{ item.value.state }} uid={{
   item.value.uid }} shell={{ item.value.shell }} password={{ item.
   value.password }} update_password=on_create
     with_dict: users

   - name: Add Keys
     authorized_key: user={{ item.key }} key="{{ item.value.sshkey
   }}"
     with_dict: users
   ```

This defines two Ansible tasks. The first loops through a dictionary called **users** and creates new users on the target node based on that information. The second task then loops through the same dictionary and takes the user's SSH keys and inserts them into the `authorized_key` file, thus allowing you to use keys to manage user access.

5. Now that we have written the code, we need to create the data for it to consume. Create a new file called `users.yml` under `first_playbook/group_vars/dev` and insert the following code:

```
users:
  admin:
    state: present
    comment: "Administrator User"
shell: "/bin/bash"
    uid: 5110
    gid: 5110
    password: "<< PASSWORD >>"
    sshkey: "<< PUB_KEY >>"

  testuser:
    state: present
    comments: "Example User"
    shell: "/bin/bash"
    uid: 510
    gid: 510
    password: "<< PASSWORD >>"
    sshkey: "<< PUB_KEY >>"
```

Wherever you see `< PUB_KEY >` insert the user's public key as a string and insert the user's encrypted password where you see `< PASSWORD >`. Users can use the `mkpasswd` utility to generate the password using the following command:

```
mkpasswd --method=SHA-512
```

6. This creates users for the `dev` environment; you can now maintain a different list of keys and users for other environments by creating a `users.yml` file under `first_playbook/group_vars/<<environment>>/`.
7. Now that we have created our users, let's install some packages that we want present on each host. First, let's create a new task within our common role by creating a new file called `install_packages.yml` under `first_playbook/roles/common/tasks/` by inserting the following code:

```
- name: "Install Common packages"
  apt: name={{ item }} state=latest
  with_items:
    - sysstat
    - open-vm-tools
```

8. Again, we will use a loop to perform a repetitive task, and this code will install every package within the `with_items` list.

> `with_items` is an invaluable directive to keep in mind. Often you can shorten very long tasks with adroit use of the `with_items` command.

9. We also need to include this task in our `main.yml` file, so edit the `first_playbook/roles/common/tasks/main.yml` file and ensure that it has the following code:

```
# tasks file for common
- include: create_users.yml
- include: install_packages.yml
```

> Keep in mind that Ansible will parse these in the order presented, so the users will always be created before the packages are installed.

10. Now we will create the `playbook` file itself. The `playbook` file defines which roles to apply against a host or set of hosts, and allows you to configure elements such as which user to run as, and if it should prompt for a `sudo` password when executed. This file should already have been created by the scaffold script, so edit `first_playbook/first_playbook.yml` and insert the following code:

```
- name: Manage Common Items
  hosts: all
  remote_user: "{{ remote_user }}"
  sudo: yes


  roles:
        - { role: common, tags: ['common'] }
```

> Note the `hosts` item; by using `hosts: all`, we ensure that every host within our inventory is configured using the common role. Note the assignment of a tag within the role declaration; this is good practice, and it allows you to selectively run individual elements of a complex playbook.

> This is done using the `--tags` switch in the `ansible-playbook` command. You can find further details at `http://docs.ansible.com/ansible/playbooks_tags.html`.

11. Note that we have used a variable to define the `remote_user`; often, environments have different predefined superusers. By using a variable, we can define the username of the superuser for each individual environment. For example, to define the variable for the `dev` environment, create a file in `first_playbook/group_vars/dev/main.yml` and insert the following configuration:

    ```
    remote_user: admin
    ```

    Here, `remote_user` is the username of the environment's power user.

12. Now we are missing only one more element: the inventory. As we are aiming to make this `playbook` self-contained, we will create the inventory along with it, rather than relying on the default Ansible location. This means that you will have a `playbook` that you can share with your colleagues, and it will contain every element required to build the targeted environment. We're now going to define our development environment. You should already have a file called named `dev` in `first_playbook/inventories`. Edit it and insert the following code:

    ```
    [<<groupname>>]
    <<targethosts>>

    [dev:children]
    << groupname >>
    ```

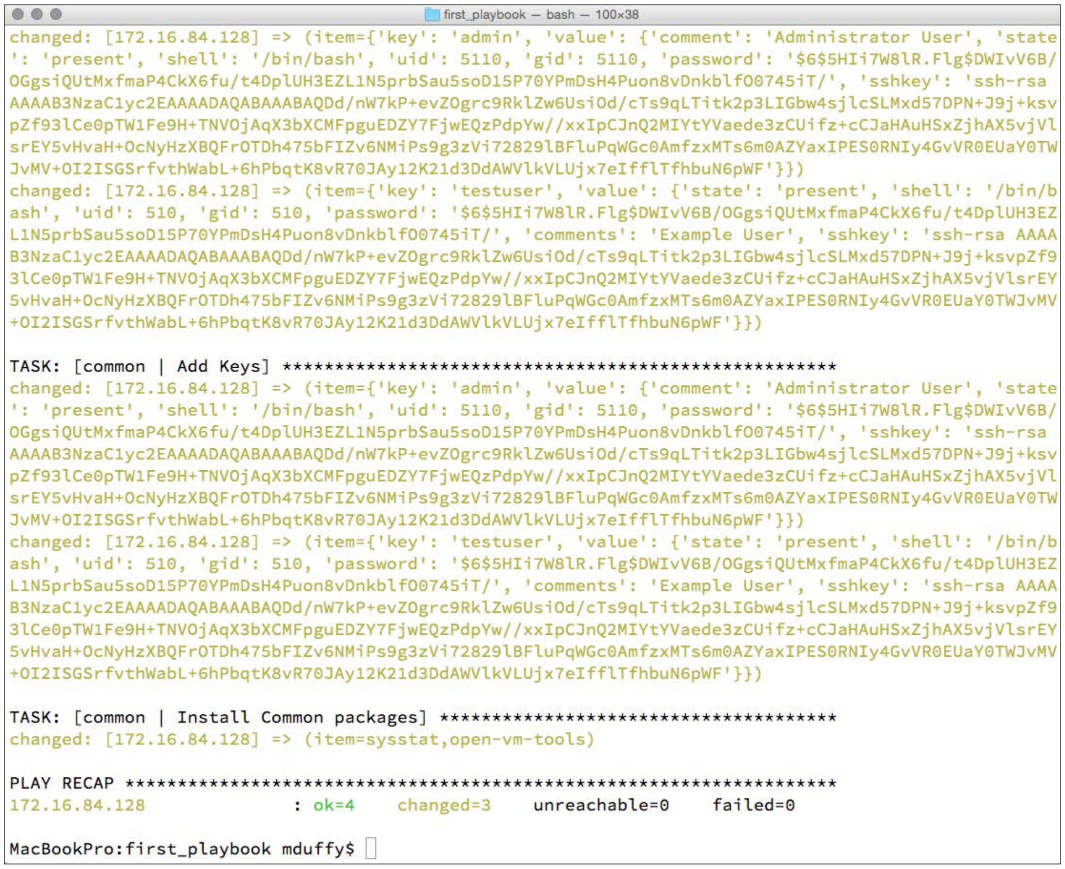
13. Where `<<groupname>>` is the group of servers that you wish to collect, and `<<targethosts>>` is the list of the servers you wish to configure within that group. Notice the `[dev:children]` block at the end. This group is the link between your environment variables and your inventory, and it should reflect the environment for which you are creating the inventory. Ensure that any group that you create is also listed within the `dev:children` group of groups to ensure that your variable files are included within the Play.

Ansible creates a link to the directories in the `group_vars` folder in the playbook for both hosts and groups in the inventory; thus, in the preceding example, we can use `<<playbook>>/groups_vars/<<hostname>>/main.yml`, `<<playbook>>/group_vars/prerequisites/main.yml`, and `<<playbook>>/group_vars/dev/main.yml` to hold variables. This allows you to set the variables at the most appropriate place in your hierarchy.

14. Once you are happy, you can run your new `playbook`. At the command line, use the following command within the root directory of your `playbook`:

    ```
    $ ansible-playbook -i inventories/dev -K first_playbook.yml
    ```

The `-i` switch is used to indicate where your inventory is located, and the `-k` to indicate which playbook that is to be executed. This should produce an output that looks similar to the following screenshot:



```
changed: [172.16.84.128] => (item={'key': 'admin', 'value': {'comment': 'Administrator User', 'state
': 'present', 'shell': '/bin/bash', 'uid': 5110, 'gid': 5110, 'password': '$6$5HIi7W8lR.Flg$DWIvV6B/
OGgsiQUtMxfmaP4CkX6fu/t4DplUH3EZL1N5prbSau5soD15P70YPmDsH4Puon8vDnkblf00745iT/', 'sshkey': 'ssh-rsa 
AAAAB3NzaC1yc2EAAAADAQABAAABAQDd/nW7kP+evZOgrc9RklZw6UsiOd/cTs9qLTitk2p3LIGbw4sjlcSLMxd57DPN+J9j+ksv
pZf93lCe0pTW1Fe9H+TNVOjAqX3bXCMFpguEDZY7FjwEQzPdpYw//xxIpCJnQ2MIYtYVaede3zCUifz+cCJaHAuHSxZjhAX5vjVl
srEY5vHvaH+OcNyHzXBQFrOTDh475bFIZv6NMiPs9g3zVi72829lBFluPqWGc0AmfzxMTs6m0AZYaxIPES0RNIy4GvVR0EUaY0TW
JvMV+OI2ISGSrfvthWabL+6hPbqtK8vR70JAy12K21d3DdAWVlkVLUjx7eIfflTfhbuN6pWF'}})
changed: [172.16.84.128] => (item={'key': 'testuser', 'value': {'state': 'present', 'shell': '/bin/b
ash', 'uid': 510, 'gid': 510, 'password': '$6$5HIi7W8lR.Flg$DWIvV6B/OGgsiQUtMxfmaP4CkX6fu/t4DplUH3EZ
L1N5prbSau5soD15P70YPmDsH4Puon8vDnkblf00745iT/', 'comments': 'Example User', 'sshkey': 'ssh-rsa AAAA
B3NzaC1yc2EAAAADAQABAAABAQDd/nW7kP+evZOgrc9RklZw6UsiOd/cTs9qLTitk2p3LIGbw4sjlcSLMxd57DPN+J9j+ksvpZf9
3lCe0pTW1Fe9H+TNVOjAqX3bXCMFpguEDZY7FjwEQzPdpYw//xxIpCJnQ2MIYtYVaede3zCUifz+cCJaHAuHSxZjhAX5vjVlsrEY
5vHvaH+OcNyHzXBQFrOTDh475bFIZv6NMiPs9g3zVi72829lBFluPqWGc0AmfzxMTs6m0AZYaxIPES0RNIy4GvVR0EUaY0TWJvMV
+OI2ISGSrfvthWabL+6hPbqtK8vR70JAy12K21d3DdAWVlkVLUjx7eIfflTfhbuN6pWF'}})

TASK: [common | Add Keys] *****************************************************
changed: [172.16.84.128] => (item={'key': 'admin', 'value': {'comment': 'Administrator User', 'state
': 'present', 'shell': '/bin/bash', 'uid': 5110, 'gid': 5110, 'password': '$6$5HIi7W8lR.Flg$DWIvV6B/
OGgsiQUtMxfmaP4CkX6fu/t4DplUH3EZL1N5prbSau5soD15P70YPmDsH4Puon8vDnkblf00745iT/', 'sshkey': 'ssh-rsa 
AAAAB3NzaC1yc2EAAAADAQABAAABAQDd/nW7kP+evZOgrc9RklZw6UsiOd/cTs9qLTitk2p3LIGbw4sjlcSLMxd57DPN+J9j+ksv
pZf93lCe0pTW1Fe9H+TNVOjAqX3bXCMFpguEDZY7FjwEQzPdpYw//xxIpCJnQ2MIYtYVaede3zCUifz+cCJaHAuHSxZjhAX5vjVl
srEY5vHvaH+OcNyHzXBQFrOTDh475bFIZv6NMiPs9g3zVi72829lBFluPqWGc0AmfzxMTs6m0AZYaxIPES0RNIy4GvVR0EUaY0TW
JvMV+OI2ISGSrfvthWabL+6hPbqtK8vR70JAy12K21d3DdAWVlkVLUjx7eIfflTfhbuN6pWF'}})
changed: [172.16.84.128] => (item={'key': 'testuser', 'value': {'state': 'present', 'shell': '/bin/b
ash', 'uid': 510, 'gid': 510, 'password': '$6$5HIi7W8lR.Flg$DWIvV6B/OGgsiQUtMxfmaP4CkX6fu/t4DplUH3EZ
L1N5prbSau5soD15P70YPmDsH4Puon8vDnkblf00745iT/', 'comments': 'Example User', 'sshkey': 'ssh-rsa AAAA
B3NzaC1yc2EAAAADAQABAAABAQDd/nW7kP+evZOgrc9RklZw6UsiOd/cTs9qLTitk2p3LIGbw4sjlcSLMxd57DPN+J9j+ksvpZf9
3lCe0pTW1Fe9H+TNVOjAqX3bXCMFpguEDZY7FjwEQzPdpYw//xxIpCJnQ2MIYtYVaede3zCUifz+cCJaHAuHSxZjhAX5vjVlsrEY
5vHvaH+OcNyHzXBQFrOTDh475bFIZv6NMiPs9g3zVi72829lBFluPqWGc0AmfzxMTs6m0AZYaxIPES0RNIy4GvVR0EUaY0TWJvMV
+OI2ISGSrfvthWabL+6hPbqtK8vR70JAy12K21d3DdAWVlkVLUjx7eIfflTfhbuN6pWF'}})

TASK: [common | Install Common packages] **************************************
changed: [172.16.84.128] => (item=sysstat,open-vm-tools)

PLAY RECAP ********************************************************************
172.16.84.128              : ok=4    changed=3    unreachable=0    failed=0

MacBookPro:first_playbook mduffy$ 
```

The preceding screenshot uses the default (and some would say, boring) output. If you want to have a little fun with your configuration management, then install the `cowsay` package on the host you're running your Ansible scripts from and enjoy a little cow-based joy. You can install `cowsay` by issuing `apt-get install cowsay`.

See also

- You can find details about the Ansible module user at the following locations:
  - **User**: `http://docs.ansible.com/user_module.html`
  - **Apt**: `http://docs.ansible.com/apt_module.html`
- You can find the documentation for Ansible Playbooks at `http://docs.ansible.com/playbooks.html`
- And you can find the example code for this recipe at `https://github.com/stunthamster/devopscoobookcode`

Creating a webserver using Ansible and Nginx

Now we have our common role defined, we can move onto defining specific roles to install and manage applications. One of the more common tasks that underpin many web applications is the installation of an HTTP server. This is a relatively common task that belies a large amount of configuration; installing a package is easy, but applying the configuration and tuning can be non-trivial, especially when you are maintaining multiple servers in a cluster. A simple mistake, such as applying tuning to one host but not the other, can lead to obscure issues, such as a misbalanced cluster, and it can be tricky to track down if each host has been created manually. We are going to create a new role that allows us to install, configure, and tune the powerful Nginx HTTP server across any number of clients.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host to run your Ansible Playbook and at least one Ubuntu 14.04 server to configure as the Nginx server.

How to do it...

1. First, we need to create a new role within our `playbook`. To accomplish this, we are going to use the `ansible-galaxy` command to create a new scaffold role. On the command line, navigate to your `first_playbook/roles` directory and issue the following command:

   ```
   $ ansible-galaxy init nginx --force
   ```

   This will create our new role.

2. Next, we need to start defining our tasks. Let's create a task to install the packages. Within the `first_playbook/roles/nginx/tasks` directory, create a new file called `install_packages.yml` and insert the following code:

   ```
   - name: "Install Nginx packages"
     apt: name=nginx state=present
   ```

3. To include this within the role, edit `first_playbook/roles/nginx/tasks/main.yml` and ensure that it includes the following:

   ```
   ---
   # tasks file for nginx
   - include: install_packages.yml
   ```

4. Next, we need to configure Nginx. For this, we are going to use the power of Ansible templates. Create a new file called `nginx.j2` in the `first_playbook/roles/nginx/templates` directory and insert the following code:

   ```
   user www-data;
   worker_processes {{ worker_processes }};

   pid /run/nginx.pid;

   events {
     worker_connections {{ worker_connections }};
     multi_accept {{ multi_accept }};
   }

   http {
     sendfile {{ send_file }};
     tcp_nopush {{ tcp_nopush }};
     tcp_nodelay {{ tcp_nodelay }};
     keepalive_timeout {{ keepalive_timeout }};
     types_hash_max_size {{ types_hash_max_size }};
     server_tokens {{ server_tokens }};
     include /etc/nginx/mime.types;
     default_type application/octet-stream;
     access_log /var/log/nginx/access.log;
     error_log /var/log/nginx/error.log;
     gzip {{ gzip }};
     gzip_disable "msie6";
     include /etc/nginx/conf.d/*.conf;
     include /etc/nginx/sites-enabled/*;
   }
   ```

   This file is a fairly standard Nginx configuration file with Jinja2 template elements applied to it. Whenever you see a set of curly braces, it will be interpolated with data supplied by our Ansible.

Templating is insanely powerful; wherever possible, ensure that you manage your configuration items with a variable, even if it's not a value you are interested in at the moment. Although it's a little bit of upfront work, it pays dividends when you do need to change something further down the line.

5. Now we have created our template, we need to create a place to store the values that we are going to insert into the variables defined within the template. Create a new file called `nginx.yml` within the `first_playbook/group_vars/dev` directory and fill it with the following values:

   ```
   worker_processes: 4

   worker_connections: 768

   multi_accept: "on"

   send_file: "on"

   tcp_nopush: "on"

   tcp_nodelay: "on"

   keepalive_timeout: 65

   types_hash_max_size: 2048

   server_tokens: "off"

   gzip: "on"
   ```

6. Next, we need a task that will copy the template to an appropriate place on the server. Create a new file called `configure_nginx.yml` under the `first_playbook/roles/nginx/tasks` directory and insert the following code:

   ```
   - name: Deploy Nginx configuration
     template: src=nginx.j2 dest="/etc/nginx/nginx.conf"
     notify: restart nginx
   ```

   Note the `notify:directive`. This makes use of the ability of Ansible to trigger actions if it causes a change of state to a resource. In this case, Ansible will restart Nginx every time there is a change in the `nginx` configuration. This is incredibly useful, as it ensures that whenever you use Ansible to push a change, it gets applied. Let's add the code to manage the restart.

7. Edit the `first_playbook/roles/nginx/handlers/main.yml` file to contain the following code snippet:

   ```
   - name: restart nginx
     service: name=nginx state=restarted
   ```

As you can see from the directory it resides in, this is known as a handler. A handler is a task that is only called when another task triggers a notify directive. This allows you to trigger events after implementing a change of state within a task. A classic example is the one that we have used, restarting a service after a configuration change.

8. We're going to apply a common tuning technique when running a high-volume web server, and increase the number of open files available to the Nginx user. Within Linux, everything is considered a file, including network sockets; a high-volume web server can chew through the default `1024` extremely quickly, so it's a good practice to allow a high number, especially if the server is dedicated to the Nginx role. Within the `first_playbook/roles/nginx/tasks/configure_nginx.yml` file, add the following code:

```
- name: Add File limits
  lineinfile: dest=/etc/security/limits.conf line='www-data
-       nofile          65535' owner=root group=root mode=0644
  notify: restart nginx
```

This uses the Ansible `lineinfile` module to insert a value into the `limits.conf` file; note that this value will be inserted at the end of the `limits.conf` file.

> This is a fairly simple use of `lineinfile`, but it is a powerful module and you should be comfortable with it, as it can use a regular expression to insert, amend, and replace values. You can find more details of the `lineinfile` module at `http://docs.ansible.com/ansible/lineinfile_module.html`.

9. One last task remains in configuring Nginx, and that's to remove the default site that's installed when the package is installed. Again, we're going to use our `configure_nginx` task by adding the following code snippet:

```
- name: Remove defaults
  file: path="/etc/nginx/sites-enabled/default" state=absent
```

The `state=absent` declaration ensures that if the file is present then Ansible will remove it.

10. Finally, we add the `configure_nginx` task to our role by adding the following to the `main.yml` file:

```
- include: configure_nginx.yml
```

11. We're now ready to add our own `virtualhost` to `nginx`, and we will use the templates to keep them consistent. First, let's create a template `virtualhost`; create a new file called `virtual_host.j2` in `first_playbook/roles/nginx/templates/virtual_host.j2` and insert the following content:

    ```
    server {

            server_name {{ item.value.server_name }};
            root {{ item.value.vhost_root }};
            index index.html index.htm;
            location / {
               try_files $uri $uri/ =404;
       }
    }
    ```

    Again, notice the use of double curly braces to denote an interpolated Ansible variable. The preceding `virtualhost` file is a simple one and we only have two values to configure; the server name, such as `www.example.com`, and the root file system where the HTML files can be found.

12. Now that we have a template, let's create a new variable file to hold the values to insert into it. First, create a new file called `virtualhosts.yml` under the `/first_playbook/group_vars/dev/`directory and insert the following dictionary:

    ```
    virtualhosts:
      test1:
        server_name: test.stunthamster.com
        vhost_root: '/usr/share/nginx/test1'

      test2:
        server_name: test2.stunthamster.com
        vhost_root: '/usr/share/nginx/test2'
    ```

    You can define as many `virtualhosts` as you like within this structure; you'll see in the next part of the recipe that the dictionary is looped over to create them, so you can control anything from `1` to `100` virtual hosts using this method.

> Remember, in the preceding example we have created the `virtualhosts` within the `dev` environment. You can follow the same steps within the `/first_playbook/group_vars/<<environment>>` directory if you wish to configure `virtualhosts` for another environment.

13. Now we have the data that we need to configure our virtual hosts. Create a new file called `configure_vhosts.yml` under the `first_playbook/roles/nginx/tasks` directory and give it the following contents:

```
- name: Create Virtual Host root
  file: path="/usr/share/nginx/{{ item.key }}" state=directory
owner=root
  with_dict: virtualhosts

- name: Add Virtual Hosts
  template: src=virtual_host.j2 dest=/etc/nginx/sites-available/{{
item.key}}.conf
  with_dict: virtualhosts


- name: Add Virtual Host symlink
  file: src="/etc/nginx/sites-available/{{ item.key}}.conf"
dest="/etc/nginx/sites-enabled/{{ item.key}}.}}.conf" state=link
  with_dict: virtualhosts
  notify: restart nginx
```

14. Note the use of `item.key` in the above example; these loops through the dictionary defined in our variable file and retrieves the key of the hash it is currently evaluating. In this case, we have used the hash to name our configuration files, which would be named `test1.conf` and `test2.conf` in this case.

    Don't forget to add this to the `main.yml` so that it runs. By now, your `main.yml` should resemble the following:

```
---
# tasks file for nginx
- include: install_packages.yml
- include: configure_nginx.yml
- include: configure_vhosts.yml
```

15. Now we have a new role, we can add it to the Playbook so that it can be run. We also need to update our inventory to define the servers the Nginx role will be run against. First of all, open the file `first_playbook/first_playbook.yml` and insert the following code:

```
- name: Manage Nginx
  hosts: nginx
  remote_user: "{{ remote_user }}"
  sudo: yes
  roles:
        - { role: nginx, tags: ['nginx'] }
```

16. This will apply the Nginx role against any server that is defined within the inventory as an Nginx node. Next, let's amend our inventory. Edit `first_playbook/inventories/dev` and insert the following configuration:

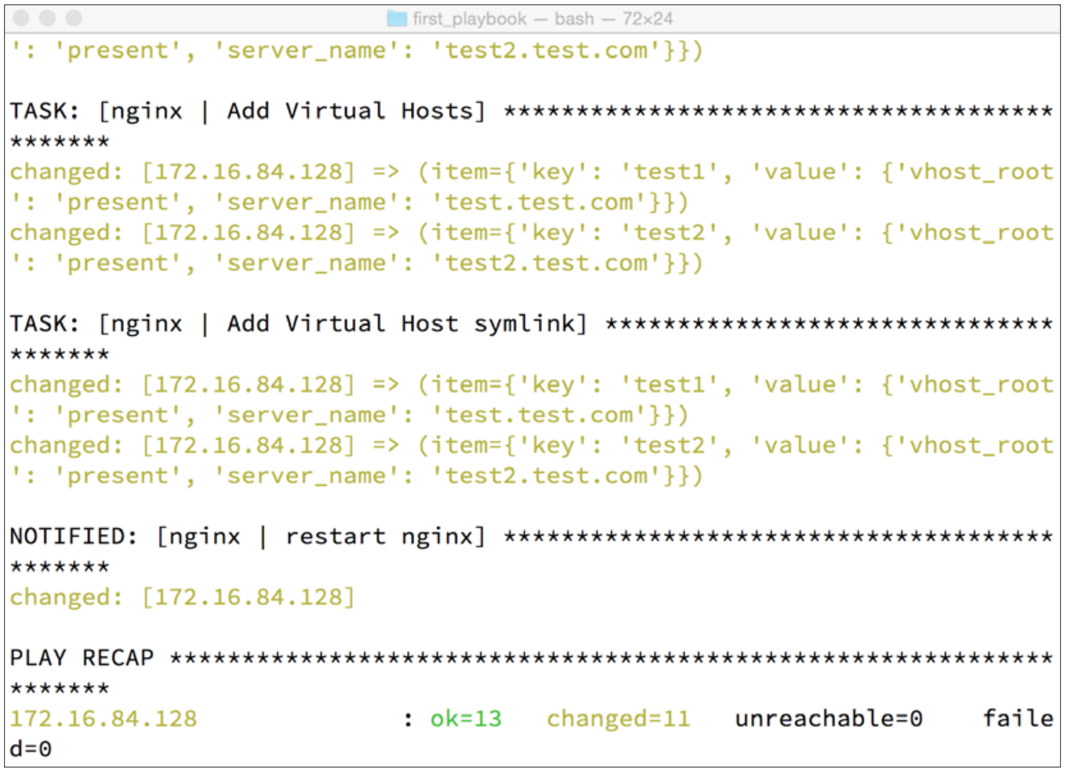
    ```
    [nginx]
    << NGINX SERVER >>

    [dev:children]
    nginx
    ```

    Here `<< NGINX SERVER >>` is the IP address or name of the server(s) you wish to configure as `nginx` server(s). This maps against the hosts directive in the Playbook.

17. We're now ready to run our `playbook`. At the terminal, run the following command within the root of the `first_playbook` directory:

    ```
    $ ansible-playbook -i inventories/dev -K first_playbook.yml
    ```

    If all goes well, you should see an Ansible output that looks something like the following screenshot:



```
': 'present', 'server_name': 'test2.test.com'}})

TASK: [nginx | Add Virtual Hosts] ***************************************
*******
changed: [172.16.84.128] => (item={'key': 'test1', 'value': {'vhost_root
': 'present', 'server_name': 'test.test.com'}})
changed: [172.16.84.128] => (item={'key': 'test2', 'value': {'vhost_root
': 'present', 'server_name': 'test2.test.com'}})

TASK: [nginx | Add Virtual Host symlink] ********************************
*******
changed: [172.16.84.128] => (item={'key': 'test1', 'value': {'vhost_root
': 'present', 'server_name': 'test.test.com'}})
changed: [172.16.84.128] => (item={'key': 'test2', 'value': {'vhost_root
': 'present', 'server_name': 'test2.test.com'}})

NOTIFIED: [nginx | restart nginx] ***************************************
*******
changed: [172.16.84.128]

PLAY RECAP **************************************************************
*******
172.16.84.128              : ok=13    changed=11    unreachable=0    faile
d=0
```

See also

You can find the Nginx documentation at `http://nginx.org/en/docs/`.

Creating an application server role using Tomcat and Ansible

We have explored using Ansible to perform common tasks on a server, and to define selected servers as `nginx` servers. This recipe will demonstrate using Ansible to install and configure a Java application server. For this recipe, we will be installing the venerable Tomcat server. Tomcat is a rock-solid open source container for Java apps, and is used in a huge array of organizations to host applications both small and large.

Currently, Ubuntu ships with a package for Tomcat 7. However, the Tomcat project is already at version 8, so we're going to look at how we can use Ansible to install Tomcat straight from the web.

Getting ready

For this recipe, you need an Ubuntu 14.04 server to act as the Ansible client, and an Ubuntu 14.04 server that you wish to configure.

How to do it...

Let's use Ansible to install and configure Tomcat:

1. First, we are going to create a new role within our `playbook` to hold our tasks. Then, we're going to use the `ansible-galaxy` command to create a new boilerplate role. On the command line, navigate to your `first_playbook/roles` and issue the following command:

   ```
   $ ansible-galaxy init tomcat
   ```

   This will create our new role.

2. Now, we're going to create a new task to install the pre-requisite packages. Generally speaking, we want a JRE at the very least. Create a new file called `install_packages.yml` under the `first_playbook/roles/tomcat/tasks/` directory and insert the following content:

   ```
   - name: "Install Tomcat prerequisites"
     apt: name={{ item }} state=latest
     with_items:
       - default-jre
       - unzip
   ```

3. Next, we amend the main task to execute this code by editing the `first_playbook/roles/tomcat/main.yml` file and inserting the following code:

   ```
   ---
   # tasks file for tomcat
   - include: install_packages.yml
   ```

4. Generally speaking, we shouldn't run anything as the `root` user, Tomcat included. Let's use Ansible to create a Tomcat user and also a group with which we can run `Tomcat`. Create a new file called `create_users.yml` under the `first_playbook/roles/tomcat/tasks` directory, and insert the following snippet:

   ```
   - name: Create Tomcat Group
     group: name=tomcat gid=5000
   - name: Create Athoris User
     user: name=tomcat comment="Tomcat App User" uid=5000  group=5000
   ```

5. We need to update `main.yml` to include this new task. Edit your `first_playbook/roles/tomcat/main.yml` file and add the following content:

   ```
   - include: create_users.yml
   ```

6. Now that we have our users and our JRE, it's time to grab the Tomcat 8 zip. We can do this using the `get_url` module. Since this is essentially a package install, we're going to add this code to our existing `install_packages.yml` file. Edit it and add the following code:

   ```
   - stat: path=/usr/local/apache-tomcat-8.0.21
     register: tc

   - name: "Fetch Tomcat"
     get_url: url=http://www.mirrorservice.org/sites/ftp.apache.org/
   tomcat/tomcat-8/v8.0.21/bin/apache-tomcat-8.0.21.zip dest=/tmp
   mode=0440

   - name: "Unpack Tomcat"
     unarchive: src=/tmp/apache-tomcat-8.0.21.zip dest=/usr/local/
   copy=no
     when: tc.isdir is undefined
   ```

   There are a few things to note in the preceding snippet. The first declaration we need to make is to use the `stat` module to fetch the state of our `Tomcat` directory. This is an important step in making the code idempotent. Next, we fetch the `zip` file containing Tomcat and unpack it. It's here that we make use of the state of the directory that we recorded using the `stat` module. The `unarchive` module will unpack the archive without testing if the Tomcat directory exists This is bad news for two reasons; first, it's not idempotent, so the target node will always take action, and secondly, it will almost certainly overwrite any subsequent changes. Using the `stat` module to test the existence of the directory will cause the `unarchive` task to skip executing if the `Tomcat` directory already exists.

> You can find more details about the stat module at http://docs.ansible.com/ansible/stat_module.html.

7. Now that we have unpacked Tomcat into our chosen location, we need to perform some tidying up. Tomcat ships with some default apps, which we may not want. Let's create a task to remove these. Add the following snippet to the `install_packages.yml` task:

   ```
   - name: "Remove default apps"
     file: path={{ item }} state=absent
     with_items:
       - /usr/local/apache-tomcat-8.0.21/webapps/docs
       - /usr/local/apache-tomcat-8.0.21/webapps/examples
       - /usr/local/apache-tomcat-8.0.21/webapps/host-manager
       - /usr/local/apache-tomcat-8.0.21/webapps/manager
       - /usr/local/apache-tomcat-8.0.21/webapps/ROOT
   ```

   Again, note the use of `with_items` to remove multiple items with a single task.

8. Now that we've removed the unwanted applications, we can configure `tomcat`. Create a new file called `setenv.j2` under the `first_playbook/roles/templates/tomcat` directory, and insert the following snippet:

   ```
   export CLASSPATH=\
   $JAVA_HOME/lib/tools.jar:\
   $CATALINA_HOME/bin/commons-daemon.jar:\
   $CATALINA_HOME/bin/bootstrap.jar
   export CATALINA_OPT="{{ tomcat.catalina.opts }}"
   export JAVA_OPTS="{{ tomcat.java.opts }}"
   ```

9. Next, let's create a place to hold the variables we're interpolating. Create a new file called `tomcat.yml` under the `first_playbook/group_vars/dev` directory, and insert the following code:

   ```
   tomcat:
     appgroup: tomcat
     appuser: tomcat
     gid: 5000
     uid: 5000
   java:
       home: '/etc/alternatives/java'
       opts: '
         -Duser.timezone=UTC
     -Dfile.encoding=UTF8
     -Xmx6g
     -Xms6g
   ```

```
  '
  catalina:
    home: '/usr/local/apache-tomcat-8.0.21/'
    pid:  '/usr/local/apache-tomcat-8.0.21/temp/tomcat.pid'
    opts: '-Dcom.sun.management.jmxremote
           -Dcom.sun.management.jmxremote.port=8082
 -Dcom.sun.management.jmxremote.authenticate=false
 -Dcom.sun.management.jmxremote.ssl=false'
```

10. We've inserted two new data structures here, one to hold our Java options and the other to hold the Tomcat specific Catalina data. Let's create the code to add this configuration to our target node. In the `first_playbook/roles/tomcat/tasks` folder, create a new file called `configure_tomcat.yml` and insert the following code snippet:

```
- name: "deploy setenv.sh"
  template: src=setenv.j2 dest=/usr/local/apache-tomcat-8.0.21/
bin/setenv.sh owner=tomcat group=tomcat
```

    This will place the `setenv.sh` file in place and fill it with the options we've configured.

11. The final element is the startup script. As we have downloaded Tomcat from the packaged distribution, it's up to us to supply our own. Create a new file in the templates directory called `tomcat.j2` and insert the following code:

```
#!/bin/sh

SHUTDOWN_WAIT=30

export APP_USER="{{ tomcat.appuser }}"
export JAVA_HOME="{{ tomcat.java.home }}"
export CATALINA_HOME="{{ tomcat.catalina.home }}"
export CATALINA_PID="{{ tomcat.catalina.pid }}"

SU="su"

start() {
  isrunning

  if [ "$?" = 0 ]; then
    echo "Tomcat is already running"
    return 0
  fi

  # Change directory to prevent path problems
```

```
    cd $CATALINA_HOME

    # Remove pidfile if still around
    test -f $CATALINA_PID && rm -f $CATALINA_PID

    $SU $APP_USER -c "umask 0002; $CATALINA_HOME/bin/catalina.sh
  start" > /dev/null
  }

  stop() {
    isrunning

    if [ "$?" = 1 ]; then
      echo "Tomcat is already stopped"
      rm -f $CATALINA_PID # remove pidfile if still around
      return 0
    fi

    echo -n "Waiting for Tomcat to exit (${SHUTDOWN_WAIT} sec.): "

    count=0
    until [ "$pid" = "" ] || [ $count -gt $SHUTDOWN_WAIT ]; do
      $SU $APP_USER -c "$CATALINA_HOME/bin/catalina.sh stop -force"
  > /dev/null
      findpid

      echo -n "."
      sleep 3
      count=$((count+3))
    done

    echo ""

    if [ "$count" -gt "$SHUTDOWN_WAIT" ]; then
      echo "Forcing Tomcat to stop"
      /bin/kill -9 $pid && sleep 5
    fi

    # check if tomcat is still around, this will be our exit status
    ! isrunning
  }

  findpid() {
    pid=""
  #pid=$(pgrep -U $APP_USER -f "^$JAVA_HOME/bin/java.*cpatalina.
base=$CATALINA_HOME")
  pid=$(ps -fu $APP_USER | grep "Dcatalina.home=$CATALINA_HOME" |
awk {'print $2'})

  # validate output of pgrep
  if ! [ "$pid" = "" ] && ! [ "$pid" -gt 0 ]; then
    echo "Unable to determine if Tomcat is running"
    exit 1
  fi
}

isrunning() {

  findpid

  if [ "$pid" = "" ]; then
    return 1
  elif [ "$pid" -gt 0 ]; then
    return 0
  fi
}

case "$1" in
  start)
    start
    RETVAL=$?

    if [ "$RETVAL" = 0 ]; then
      echo "Started Tomcat"
    else
      echo "Not able to start Tomcat"
    fi
    ;;

  stop)
    stop
    RETVAL=$?

    if [ "$RETVAL" = 0 ]; then
      echo "Stopped Tomcat"
    else
      echo "Not able to stop Tomcat"
    fi
  ;;

  restart)
    stop
    sleep 5
    start
    RETVAL=$?

    if [ "$RETVAL" = 0 ]; then
      echo "Restarted Tomcat"
    else
      echo "Not able to restart Tomcat"
    fi
  ;;

  status)
    isrunning
    RETVAL=$?

    if [ "$RETVAL" = 0 ]; then
      echo "Tomcat (pid $pid) is running..."
    else
      echo "Tomcat is stopped"
      RETVAL=3
    fi
  ;;

  *)
    echo "Usage: $0 {start|stop|restart|status}."
  ;;

esac

exit $RETVAL
```

12. Next we need to add the Ansible code to place the template onto the server. Add the following snippet at the bottom of the `configure_tomcat.yml` file,:

```
- name: "Deploy startup script"
  template: src=tomcat.j2 dest=/etc/init.d/tomcat owner=root
mode=700
```

13. Now, let's add this set of tasks into the main `playbook`. You can do this by adding the highlighted code to the `main.yml` file:

    ```
    ---
    # tasks file for tomcat
    - include: create_users.yml
    - include: install_packages.yml
    - include: configure_tomcat.yml
    ```

14. Next, we should amend our inventory to add our Tomcat servers to it. Edit the inventory located in `first_playbook/inventories/dev` and insert the following code:

    ```
    [tomcat]
    <<Node>>
    ```

    Now, replace `<<Node>>` with the *nodes* you wish to configure as a Tomcat node.

15. Finally, we add the role to our playbook file. Edit `first_playbook/first_playbook.yml` and insert the following code:

    ```
    - name: Manage Tomcat
        hosts: tomcat
        remote_user: "{{ remote_user }}"
        sudo: yes
        roles:
              - { role: tomcat, tags: ['tomcat'] }
    ```

16. You can now run this role and you will have a Tomcat 8 container ready to run your code.

See also

You can find the documentation for Tomcat at `http://tomcat.apache.org/tomcat-8.0-doc`.

Installing MySQL using Ansible

We now have an Ansible Playbook that can manage common items. It can install and configure Nginx, and also finally, install and configure Tomcat. The next logical step is to install some form of data storage, and for this, we are going to look at MySQL.

MySQL is arguably one of the most popular databases deployed due both to its relative ease of use, and its open source heritage. MySQL is powerful enough for sites both large and small, and powers many of the most popular sites on the Internet. Although it may lack some of the enterprise features that it's more expensive cousins, such as Oracle and Microsoft SQL, have, it more than makes up for that by being relatively simple to install and able to scale without license costs.

Getting ready

For this recipe, you need an Ubuntu 14.04 server to act as your Ansible client, and an Ubuntu 14.04 server that you wish to configure for MySQL.

How to do it...

Let's install MySQL using Ansible:

1. As with the previous recipes, we're going to create a new `role` within our `playbook`. Navigate to the `tasks` folder and issue the following command:

   ```
   $ ansible-galaxy init mysql --force
   ```

   This will create our new role and the underlying folder structure.

2. We're going to start by installing the packages for MySQL. Create a new file called `install_packages.yml` under the MySQL role's `tasks` folder, and insert the following code:

   ```
   - name: 'Install MySQL packages'
     apt: name={{ item }} state=latest
     with_items:
       - python-dev
       - libmysqlclient-dev
       - python-pip
       - mysql-server

   - pip: name=MySQL-python
   ```

   There are a couple of things going on here. First, we are installing a few more packages aside from MySQL itself. This is to support the Ansible `MySQL` module, and to allow us to use the `pip` package manager to install another prerequisite package.

3. Now that we have installed MySQL, we can configure it. First, start by changing the password of the `root MySQL` user; by default, it is set to nothing. Create a file called `configure_mysql.yml` in the `tasks` directory, and insert the following code snippet:

```
- name: Set root password
mysql_user: name=root host={{ item }} password={{
mysql_root_password }}
  with_items:
    - "{{ ansible_hostname }}"
    - 127.0.0.1
    - ::1
    - localhost
```

4. Remember to add this task to the `main.yml` file by adding the following to the bottom of the file:

```
- configure_mysql.yml
```

   There are two things to notice here. First, we're iterating over the list of hosts. This ensures that the `root` user has their password changed in all the various permutations that it might exist in. Second, we're using a variable to contain the `root` password. Note the use of `{{ansible_hostname}}` in the `with_items` list. This uses details gathered from the target host to populate certain reserved variables; this is incredibly useful for situations such as these.

[You can find more details of Ansible facts at `http://docs.ansible.com/ansible/playbooks_variables.html#information-discovered-from-systems-facts`.]

5. Next, we're going to create a `.my.cnf` file. This is a convenience file that allows you to insert certain options that the MySQL client can use, and saves you the effort of typing at the command line. Normally, this is used to save key strokes but in this case it is used to ensure that when Ansible runs for the second time, it can access the database using the password we have set. Create the `.my.cnf` file using this code snippet:

```
- name: Create .my.cnf file
  template: src=my.cnf.j2 dest=/root/.my.cnf owner=root
mode=0644
```

6. As you've noticed, this makes use of a template to create the file. Create the template by creating a new file called `my.cnf.j2` under the templates directory in the MySQL role and insert the following code:

```
[client]
user=root
password={{ mysql_root_password }}
```

7. Normally, we would create a file under the `group_vars/dev` directory to hold the MySQL root password variable, and this will work. However, since this is sensitive information, we want to make sure that casual prying eyes don't stumble across the `root` password of our shiny new MySQL server. Instead, we are going to use the Ansible `vault` feature. This will create an encrypted file that will hold our password, and Ansible will be able to read it at runtime. Run the following command from the root of the playbook:

   ```
   $ ansible-vault create group_vars/dev/mysql.yml
   ```

8. You'll be prompted to enter a vault password. Make sure it's something you can remember, as you'll need it every time you run your Ansible Playbook. Once you enter and confirm the password, you will be handed over to an editor to enter your data. Insert the following data:

   ```
   mysql_root_password: <<ROOTPASSWORD>>
   ```

   Here, `<<ROOTPASSWORD>>` is your chosen MySQL password. Save the file and exit, and Ansible will encrypt it for you. If you open the file in your editor now, you will find it has content similar to this:

   ```
   $ANSIBLE_VAULT;1.1;AES256
   6335303965373866323238346534323535316636303430636134363732396166 38
   65386137316263
   3133383237616662653837613966396666653732376136620a35613963393738 38
   30613732336533
   3739346539643161373738396131626533633137363737316661643039323832 65
   35366333303632
   366164323730326637 0a34616166353364343664333161616536616432616363 62
   34633932336364
   3437643732373765356632326166613364383435393832303130386530396236 62
   38
   ```

> Using the Ansible `crypt` feature is a fantastic way to keep sensitive data a secret, and can be used on a variable file. You can find more details on the crcypt feature at `https://docs.ansible.com/playbooks_vault.html`.

9. Now that we have our `.my.cnf` file, we can tidy up. Within the default install on Ubuntu 14.04, an anonymous user is created along with a test database; we're going to use the Ansible MySQL module to remove both of these. Insert the following code snippet into the `configure_mysql.ym` file:

   ```
   name: delete default user
     action: mysql_user user="" state="absent"
   - name: remove the test database
     action: mysql_db db=test state=absent
   ```

10. Now that we have installed and configured our MySQL server, it's time to use the Ansible MySQL module to create a new database and database user. In this case, we're going to create a database for a blog. Create a new task within the MySQL role called `create_blog_db.yml` and insert the following content:

    ```
    - name: Create MyBlog DB
    mysql_db: name=myblog state=present

    - name: Create MyBlog User
    mysql_user: name=myblog_user password=agreatpassword
    priv=myblog.*:ALL state=presentpresent
    ```

    This code snippet uses the Ansible MySQL module to create a new database and a matching user with the correct privileges to use it.

11. Finally, we just need to update our `main.yml` file to include our various tasks. Edit it to include the following content:

    ```
    ---
    # tasks file for mysql
    - include: install_packages.yml
    - include: configure_mysql.yml
    - include: create_blog_db.yml
    ```

12. Our new role is complete and ready to use. Now, we just need to update our playbook and inventory to include it. First, open up the `first_playbook.yml` file in your editor and add the following content:

    ```
    - name: Manage MySQL
        hosts: mysql
        remote_user: "{{ remote_user }}"
        sudo: yes

        roles:
              - { role: mysql, tags: ['mysql'] }
    ```

13. Now, we need to update our inventory. Open the `inventories/dev` file and insert the following snippet:

    ```
    [mysql]
    << mysql_server >>

    [dev:children]
    nginx
    tomcat
    mysql
    ```

    Where `mysql_server` is the server (or servers) that you wish to configure MySQL on.

14. Now, if you run the `playbook` you will find your selected host will have MySQL installed, with the new database ready for use. As we now have an encrypted file, you will need to add the `--ask-vault-pass` switch; your command should now look something similar to the following:

    ```
    $ ansible-playbook --ask-vault-pass -i inventories/dev -k first_
    playbook.yml
    ```

    This will prompt you for your vault password and it will then decyrpt and use the values contained within.

See also

- You can find out more about the Ansible MySQL module at:

  `http://docs.ansible.com/mysql_db_module.html`

- You can find details of the Ansible MySQL User Module at:

  `http://docs.ansible.com/mysql_user_module.html`

- You can find details of the Ansible Playbook vaults at:

  `https://docs.ansible.com/playbooks_vault.html`

Installing and managing HAProxy with Ansible

One key element of high-performance web applications is the ability to scale, and the easiest way to achieve this is to use a load balancer to direct traffic to multiple nodes. This can provide both horizontal scale and, just as importantly, the ability to survive individual node failures.

There are many load balancers available, both open source and commercial, but HAProxy is certainly one of the more popular. Open Source, high performance, and highly configurable, HAProxy is a good choice for any site that requires load balancing.

This recipe will demonstrate how to install HAProxy, configure it, and add both a frontend and backend service to it.

Getting ready

For this recipe, you need an Ubuntu 14.04 server to act as our Ansible client and an Ubuntu 14.04 server that you wish to configure for HAProxy.

How to do it...

Let's install and manage HAProxy and Ansible:

1. We are going to use the `ansible-galaxy` command to create our role scaffold. Do this by issuing the following command:

   ```
   $ ansible-galaxy init haproxy -p "${PLAYBOOK_PATH}/roles/"
   ```

2. Now we have the role, let's start with the tasks that will deal with installing the packages. By default, Ubuntu 14.04 ships with HaProxy 1.4, whereas 1.5 is the latest version and brings important features such as SSL termination. Fortunately, there is a PPA available which allows us to install the more recent version. Start by creating a new file called `install_packages.yml` under the `roles/tasks` directory, and insert the following snippet:

   ```
   name: "Add HAProxy repo"
     apt_repository: repo="deb http://ppa.launchpad.net/vbernat/
   haproxy-1.5/ubuntu trusty main" state=present

   - name: Install HAProxy
     apt: name=haproxy state=installed force=yes
   ```

3. This will add the PPA to the package list and install HAproxy; however, we're also going to install the `hatop` package. Hatop is a fantastic tool for monitoring HAProxy and allows you to see detailed traffic statistics quickly and easily. Add the following code in the `install_packages.yml` file:

   ```
   - name: Install HATop
     apt: name=hatop state=installed
   ```

   This will install `hatop` to allow you to monitor your load balancer. Next, we're going to configure HaProxy. Create a new file called `configure_haproxy.yml`, and insert the following code:

   ```
   - name: Deploy HAProxy configuration
     template: src=haproxy.cfg dest=/etc/haproxy/haproxy.cfg
     notify: Restart HAProxy
   ```

4. Remember to add this task to the `main.yml` file by appending the following code at the bottom:

   ```
   - configure_haproxy.yml
   ```

5. As you can see, this writes a template into the `/etc/haproxy` directory; you need to create the template by creating a new file under the `haproxy` role, in the templates directory, and add the following content:

```
global
        log 127.0.0.1   local0 notice
        stats socket /var/run/haproxy.sock mode 600 level admin
        stats timeout 2m
        maxconn {{ haproxy.maxconns }}
        user haproxy
        group haproxy

defaults
        option http-server-close
        log    global
        option dontlognull
        timeout http-request {{ haproxy.http_timeout }}
        backlog {{ haproxy.backlog }}
        timeout queue {{ haproxy.timeout_q }}
        timeout connect {{ haproxy.timoutconnect }}
        timeout client {{ haproxy.timeoutclient }}
        timeout server {{ haproxy.timoutserver }}

frontend default_site
        bind {{ haproxy.frontend_ip }}:{{ haproxy.frontend_port }}
        mode http
        option httplog
        default_backend app_server


backend app_server
        balance {{ haproxy.balance }}
        mode   http

        {% for node in groups['tomcat'] %}
        server {{node}} {{node}}:8080 check
        {% endfor %}
```

6. As you can see, we're using a lot of variable interpolation in this template; this is good practice. If you think you might be changing a value, it's best to template it. Also, take a look at this snippet:

```
        {% for node in groups[nginx] %}
        server {{node}} {{node}}:8080 check
        {% endfor %}
```

This code is interesting as it uses the data included in the Ansible inventory to build the template values. This essentially means that whenever we add a new host to the Nginx role, not only will it be configured for Nginx, it will be added to the load balancer automatically.

7. Now that our template is ready, we can create a file to hold the values that it's going to interpolate. Create a new file called `haproxy.yml` inside the `group_vars/dev` directory and insert the following:

```
haproxy:
  frontend_ip: 192.168.1.1
  maxconns: 4096
  backlog: 2
  timeout_q: 400ms
  timoutconnect: 5000ms
  timeoutclient: 5000ms
  timoutserver: 5000ms
  http_timeout: 15s
  balance: leastconn
  frontend_port: 83
```

8. Next, we need to add the role and host into the playbook and inventory respectively. First, let's amend the playbook to add our new role. Open the `first_playbook.yml` file, and insert the following:

```
- name: Manage HAProxy
  hosts: haproxy
  remote_user: "{{ remote_user }}"
  sudo: yes
  roles:
        - { role: haproxy, tags: ['haproxy'] }
```

9. Now, we amend the inventory. Open the `inventories/dev` file in your editor and insert the following snippet:

```
[haproxy]
<<SERVER NAME>>
```

10. Also, remember to add the `haproxy` role to the children as the highlighted code in this snippet:

```
[dev:children]
nginx
tomcat
mysql
haproxy
```

Now when you run your playbook, you will find that the servers you have configured as HAproxy hosts will be configured with HAproxy; they will also add the servers you have configured as Nginx nodes to the load balancer.

See also

- The HAproxy documentation can be found here:

  `http://www.haproxy.org/#docs`

- There is a module within Ansible that can be used to control the HAproxy; this can be used to integrate a load balancer with a deployment script:

  `http://docs.ansible.com/haproxy_module.html`

Using ServerSpec to test your Playbook

As mentioned in the introduction of this book, the DevOps methodology is built on some of the best practices already in use within software development. One of the more important ideas is the concept of unit testing; in essence, a test that ensures that the code performs the correct operations under certain scenarios. It has a two-fold advantage; first of course, you can test for code correctness before it even arrives in a test environment and you can also ensure that when you refactor code, you don't inadvertently break it. It is the second of these advantage that truly shines for Ansible Playbooks. Due to the way Ansible works, you can guarantee that a certain state will appear when you declare it; Ansible is almost running unit tests itself to ensure that an operation has been carried out correctly. However, you might need to ensure that certain elements are there on a server, and it's incredibly easy to drop these accidently, especially if you are carrying out a large-scale refactoring exercise having ServerSpec on hand can help stop this from happening.

Getting ready

You will need an Ubuntu 14.04 client to run the `serverspec` code.

How to do it…

Let's test our Playbook:

1. First, we are going to install the packages we need. ServerSpec is written in Ruby, so we can use the Gem package manager to install it; however, first we need to install Ruby. You can do this using the following command:

   ```
   $ sudo apt-get install ruby
   ```

2. Once Ruby is installed at the command line, enter the following command:

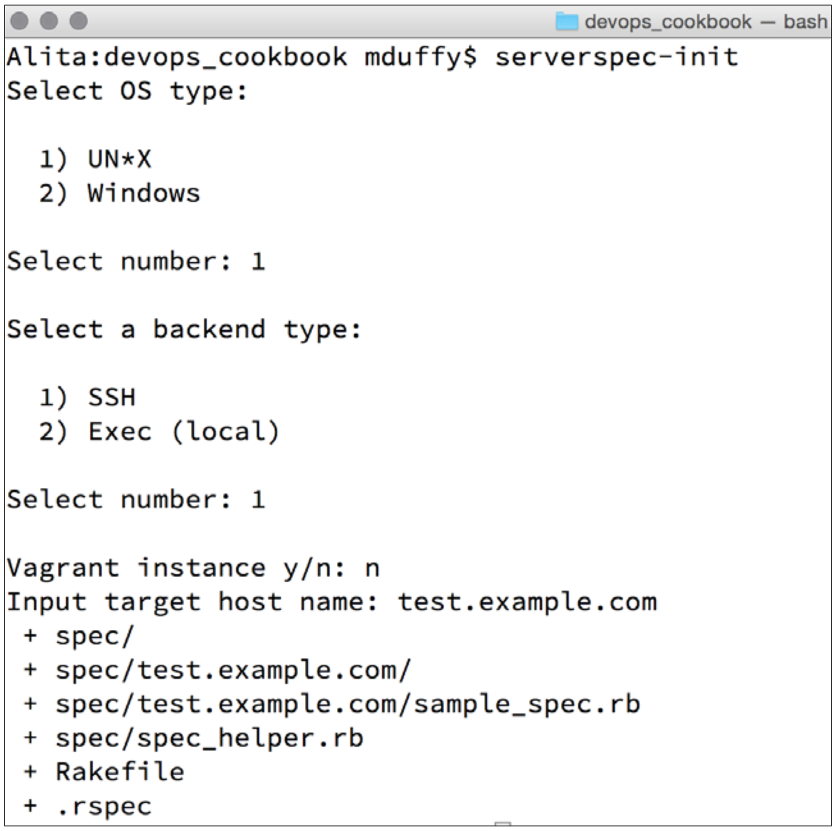
   ```
   $ geminstall serverspec highline
   ```

   This will install both ServerSpec and its dependency highline.

3. Next, we are going to use `serverspec` to create a new `skeleton` project. Since this is going to test our `playbook`, ensure that you are in the root of the `playbook` directory when you issue the next command:

   ```
   $ serverspec-init
   ```

   This command is going to prompt you for a few details; see the following screenshot for some example entries:



```
Alita:devops_cookbook mduffy$ serverspec-init
Select OS type:

  1) UN*X
  2) Windows

Select number: 1

Select a backend type:

  1) SSH
  2) Exec (local)

Select number: 1

Vagrant instance y/n: n
Input target host name: test.example.com
 + spec/
 + spec/test.example.com/
 + spec/test.example.com/sample_spec.rb
 + spec/spec_helper.rb
 + Rakefile
 + .rspec
```

   Remember, this will be run against a test server; ideally this is something like a Virtual Machine that runs on your desktop. ServerSpec can also integrate with Vagrant, and this is also an excellent method to test your code without needing a dedicated server.

4. Now, we need to do a little clean up. When you use the `serverspec-init` command, it creates a file called `sample_spec.rb` under a folder named after your test server; we don't need this, so remove it.
5. Now, we are going to create our test file for our Nginx Role. Create a new file `under spec/{testserver}` called `nginx_role_spec.rb` and insert the following:

   ```
   require 'spec_helper'
   require 'yaml'
   ```

   These require statements will bring in the libraries that we will need to run our tests.

6. We're going to start by testing the basics; check if the Nginx package installed, and if the service is running and listening on the correct port; insert the following code into the test:

   ```
   describe package('nginx'), :if => os[:family] == 'ubuntu'
   do
     it { should be_installed }
   end


   describe service('nginx'), :if => os[:family] == 'ubuntu' do
     it { should be_enabled }
   end

   describe port(80) do
     it { should be_listening }
   end
   ```

   These three blocks of code use the additional functions provided by the spec_helper library to allow us to describe a test; much like Ansible, it abstracts you away from needing explicit commands to test something and instead provides preset resources that you can easily access

You can find the complete list of ServerSpec resources at:
`http://serverspec.org/resource_types.html`

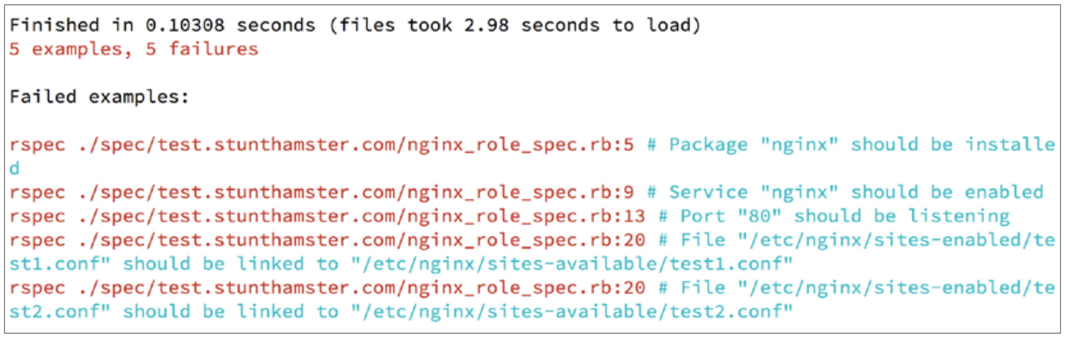
7. Now that we have the basics covered, let's use some data from our Playbook to power the next test. As mentioned above, ServerSpec tests are written in pure Ruby, so we can use the features of that language to help write more tests that are complex. In this case, we're going to load the contents of our variables in Ansible to iterate over our virtual hosts and check if the configuration files are present and linked properly; insert the following code into your test file:

```
vh_list = YAML.load_file('group_vars/dev/virtualhosts.yml')
vh_list['virtualhosts'].each do |key|
  describe file ("/etc/nginx/sites-enabled/#{key[0]}.conf")do
    it { should be_linked_to "/etc/nginx/sites-
available/#{key[0]}.conf"}
  end
```

8. We are using a basic Ruby loop to open our `virtualhosts.yml` file, extract the values of each hash, and use it to build a test against the file. This is a great technique to keep in mind, as it means that your test can use the data in your playbook automatically.
9. We can now run our test suite using the following command:

```
$ rake spec
```

If we now run the tests against a test server that hasn't had Ansible run against it, you should see output similar to the following:



```
Finished in 0.10308 seconds (files took 2.98 seconds to load)
5 examples, 5 failures

Failed examples:

rspec ./spec/test.stunthamster.com/nginx_role_spec.rb:5 # Package "nginx" should be installe
d
rspec ./spec/test.stunthamster.com/nginx_role_spec.rb:9 # Service "nginx" should be enabled
rspec ./spec/test.stunthamster.com/nginx_role_spec.rb:13 # Port "80" should be listening
rspec ./spec/test.stunthamster.com/nginx_role_spec.rb:20 # File "/etc/nginx/sites-enabled/te
st1.conf" should be linked to "/etc/nginx/sites-available/test1.conf"
rspec ./spec/test.stunthamster.com/nginx_role_spec.rb:20 # File "/etc/nginx/sites-enabled/te
st2.conf" should be linked to "/etc/nginx/sites-available/test2.conf"
```

This is exactly what we want to see; since we haven't configured anything yet, all the tests should fail. If we run Ansible to configure Nginx on the server and run the tests now, you should see an output similar to the following:

```
Package "nginx"
  should be installed

Service "nginx"
  should be enabled

Port "80"
  should be listening

File "/etc/nginx/sites-enabled/test1.conf"
  should be linked to "/etc/nginx/sites-available/test1.conf"

File "/etc/nginx/sites-enabled/test2.conf"
  should be linked to "/etc/nginx/sites-available/test2.conf"

Finished in 0.10165 seconds (files took 5.71 seconds to load)
5 examples, 0 failures
```

By writing unit tests for your Ansible code, you are ensuring that changes can be applied with far more confidence, and can reduce incidences of broken code.

See also

You can find more details at the ServerSpec home page at `http://serverspec.org`.

6 Containerization with Docker

In this chapter, we are going to cover the following topics:

- Installing Docker
- Pulling an image from the public Docker registry
- Performing basic Docker operations
- Running a container interactively
- Creating a Dockerfile
- Running a container in detached mode
- Saving and restoring a container
- Using the host only network
- Running a private Docker registry
- Managing images with a private registry

Introduction

Containerization is not a new technology, but it has enjoyed a recent renaissance; this has been due to the emergence of Docker, which has made using containerization reasonably straightforward, and it has enjoyed a rapid uptake of both developers and system administrators. However, despite all of the enthusiasm, Docker is based on existing and well-understood technology.

Containers have been around in some form or other for a very long time, but until Docker debuted they lacked an especially compelling tool chain. This has caused them to languish, with most users electing to spin up full fat virtual machines rather than containers. This is a shame, as there are many compelling benefits to using containers over full virtualization in many use cases. To understand these benefits, we need to consider how a container works versus virtual machines. Unlike a virtual machine which runs a full kernel user space and application within an isolated system, a container uses the underlying kernel of the container host and runs the user space and Applications in its own sandbox. This sharply reduces overhead on contended hosts, as you are only running a single kernel, rather than many. Docker also makes use of a layered file system; it builds images by layering many immutable layers together and creates an isolated writable space for the container. This means that if you have a hundred containers based on Ubuntu 14.04, you are only consuming the disk space for a single Ubuntu image; you use the disk space only to store the changes made to the running container.

It's important to understand the difference between a container and an image. An image is an immutable template, which is generally built from a set of instructions called a Dockerfile. The image cannot be changed once it is built, and is used as the basis for a container. When you run a container, the image is used to *boot* it, and from there the container writes any changes to a new mutable layer.

Docker debuted in March 2013 as an Open Source project, and has grown explosively; it is now used by startups and large enterprises alike. It has also attracted a great deal of interest from investors, and at the time of writing, the Docker project has grown into one of the most funded startups in the world, and is partnering with companies as diverse as Microsoft and Red Hat to bring containers to a vast array of differing platforms. It's not just the operating system vendors who have embraced Docker, and many of the **Platform-as-a-Service** (**PAAS**) vendors either rolled out Docker support or are planning to in the near future.

There are many reasons why Docker appeals to developers. Primarily, it helps solve the problem of packaging. For many years, there has been an enduring question over what should constitute a deployable package, and how much of the underlying operating system should be encompassed within it. Docker offers the ability to create a *complete* deployable with every dependency, from operating system up, managed in an easily deployable artifact. Secondly, Docker makes it easy to scale elastic applications, as containers are generally small and fast to start. The most time consuming part of standing up a new container is the time it takes to download the initial image; this can be ameliorated by creating a local Docker registry, and we will be looking at how to achieve that later in this chapter.

Although it is easy to create and destroy containers at will, it does bring new challenges; such a free flowing infrastructure creates confusion over which apps are hosted where. Fortunately, now there is a growing ecosystem of applications that offer orchestration of Docker containers, and this is set to be an area of growth within the Docker ecosystem.

Installing Docker

Before we go any further, we will learn how to install the software that allows us to host Docker containers.

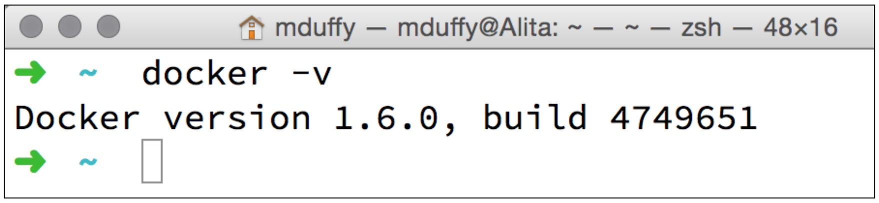
Getting ready

For this recipe, you will need an Ubuntu 14.04 server.

How to do it...

The Docker developers have gone to great lengths to make the installation of Docker as simple as possible, and this recipe should have you up and running within minutes:

1. First, ensure that you have the `wget` tool installed by issuing the following command:
   ```
   $ sudo apt-get install wget
   ```
2. Once you have `wget` installed, issue the following command to run the Docker installer:
   ```
   $ wget -qO- https://get.docker.com/ | sh
   ```
3. The installer will prompt you for the `sudo` password, which once entered will install Docker and any dependencies. Once the install is complete, you can verify that Docker is correctly installed by running the following command:
   ```
   $ docker -v
   ```
   You should receive output similar to the following screenshot:



```
mduffy — mduffy@Alita: ~ — ~ — zsh — 48×16
➜  ~  docker -v
Docker version 1.6.0, build 4749651
➜  ~
```

See also

You can find the Docker installation instructions for various operating systems at: `https://docs.docker.com/installation/`.

Pulling an image from the public Docker registry

Now that we have installed Docker, we can use it to run a container from the public Docker registry. The public Docker registry contains thousands of ready to use images that cover hundreds of different packages, from databases, through to app servers. The public registry also includes official images from certain software providers, offering you a quick method to start developing with those packages, and the surety that the image is correct and secure.

For this recipe, we're going to use a combination of two different images to run a basic WordPress blog.

Getting ready

To use this recipe, you will need an Ubuntu 14.04 server with Docker installed.

How to do it…

This recipe will use some simple Docker commands, and will use the public Docker images for MySQL and WordPress to install a blog:

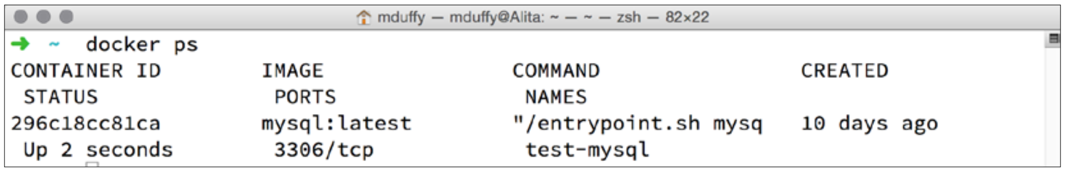
1. The first task that we need to accomplish is to create a MySQL container to hold our data. We can do this using the following command:

   ```
   $ sudo docker run --name test-mysql -e MYSQL_ROOT_
   PASSWORD=password -d mysql:latest
   ```

2. This command will connect to the Docker public registry and pull the container image tagged as `mysql:latest` down to the server. Once it's downloaded, a new Docker container called `test-mysql` will be started with a MySQL root password of `password`. You can confirm it's running by issuing the following command:

   ```
   $ docker ps
   ```

   This should produce output similar to this:



```
mduffy — mduffy@Alita: ~ — ~ — zsh — 82×22
➜  ~  docker ps
CONTAINER ID        IMAGE               COMMAND                CREATED
 STATUS              PORTS               NAMES
296c18cc81ca        mysql:latest        "/entrypoint.sh mysq   10 days ago
 Up 2 seconds        3306/tcp            test-mysql
```

3. Now that we have a MySQL container, we can turn our attention to WordPress. As with MySQL, the WordPress developers have created an official container image, and we can use this to run WordPress using the following command:

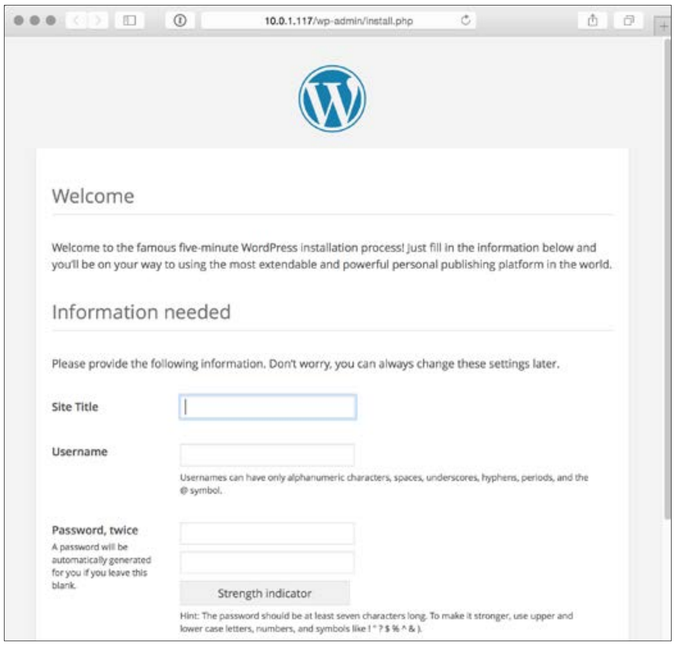
   ```
   $ docker run --name test-wordpress -p 80:80 --link test-
   mysql:mysql -d wordpress
   ```

4. This command will retrieve and run the official WordPress image, and will name it `test-Wordpress`. Note the `--link` option; this links the MySQL container to the WordPress container without creating an explicit network between the two.

> Keep in mind that you can only link two containers on the same host; if the containers are on different hosts, you will need to map ports for them to be able to communicate.

5. Note the `-p` option, this exports TCP port `80` from the container to port 80 on the host, making the WordPress installation accessible. It will not be available to the outside world without the port mapping, even though the container has port `80` open. The mapping essentially creates a firewall rule on the Docker host that bridges between the host network and the virtual network created for the Docker containers to run on.

6. Open a browser, point it to the address of your Docker host, and you should see the following page:



Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

Username

Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password, twice

A password will be automatically generated for you if you leave this blank.

Hint: The password should be at least seven characters long. To make it stronger, use upper and lower case letters, numbers, and symbols like ! " ? $ % ^ &).

See also

- You can find the official MySQL image at `https://registry.hub.docker.com/_/wordpress/`
- In addition, you can find the official WordPress image and documentation at `https://registry.hub.docker.com/_/wordpress/`

Performing basic Docker operations

Now that we have the ability to create Docker containers, let's have a look at how to control them. Docker has a comprehensive set of tools that allows you to start, stop, and delete containers.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server with Docker installed.

How to do it...

This recipe demonstrates the basic commands used to manage Docker containers. By using these commands, you can manage the full lifecycle of the container:

1. Use the following command to list the running containers on your system:

   ```
   $ sudo docker ps
   ```

> This only shows the running containers. To see the containers that have been stopped, use the following command:
>
> ```
> $ sudo docker ps -a
> ```

2. To stop a running container, use the following command:

   ```
   $ docker stop <dockerID>
   ```

   Here the `ID` is derived from running `docker ps` and selecting the ID of the image you wish to stop.

3. To remove a `docker` container, use the following command:

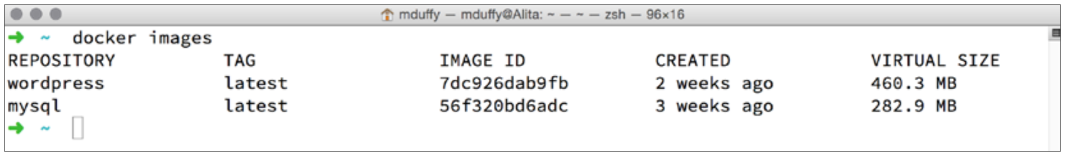
   ```
   $ sudo docker rm <CONTAINER ID>
   ```

Remember, this only removes the CONTAINER and not the underlying image.

4. Use the following command to list the Docker images that have been downloaded to the host:

```
$ sudo docker images
```

This should produce output that looks something similar to this:



```
➜ ~ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
wordpress           latest              7dc926dab9fb        2 weeks ago         460.3 MB
mysql               latest              56f320bd6adc        3 weeks ago         282.9 MB
➜ ~
```

5. Use the following command to remove an image:

```
$ sudo docker rmi < IMAGE ID >
```

Removing images is a safe operation; Docker uses reference counting to keep track of containers that have image dependencies. If you attempt to remove an image that is in use by a container (started or stopped) on your host, you will receive a warning, and it will not be removed.

See also

You can find instructions on how to work with images and containers at `https://docs.docker.com/userguide/`.

Running a container interactively

You will want to run containers in a detached mode; however, there are times when it is very useful to be able to run the container interactively to diagnose issues.

Running a container interactively essentially gives you a shell on the container, and from within the container you can work in the same way, as you would with any other Linux system.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server with Docker installed.

How to do it...

You can start any Docker container in an interactive mode using the following command:

```
$ sudo docker run -i -t ubuntu /bin/bash
```

See also

Refer the documentation on Docker at: `https://docs.docker.com/articles/basics/`.

Creating a Dockerfile

Although there are many premade images available on the Docker registry, it is inevitable that you will want to create your own images as the basis for your containers. One of the standout features of Docker is its straightforward build tools, and you can easily create new images with a simple text file.

In this recipe, you will learn how to use Docker to package the Gollum Wiki software and push it to the Docker public repository.

Getting ready

For this recipe, you will need a server with Docker installed.

How to do it...

The following steps outline how to create a new Dockerfile, build it, and finally push it to the Docker Registry:

1. First, we are going to create a new Docker registry account. Start by visiting `https://registry.hub.docker.com` and follow the signup instructions that you will find on this page to create a new account. By creating your account, you create a namespace within the Docker registry, which allows you to upload your own images.
2. We have created a Docker registry account, so we can now turn our attention to creating our first Dockerfile. A Dockerfile is the list of steps used to create a complete Docker image. These steps can include copying files into the image, or running commands, but every Dockerfile needs to start with a `FROM` command. The `FROM` command allows you to choose the Docker image that will form the basis for this container; generally speaking, this will be an OS image, and many Linux distributions now ship an official image that can be used.

It might strike you as slightly recursive that you need to use an image to create an image. If you wish to create your own OS image to serve as your base, you can follow the instructions given at: `https://docs.docker.com/articles/baseimages/`.

3. Let's use the Ubuntu image for our Gollum container. Do this by creating a new file called Dockerfile and inserting the following code:

   ```
   FROM ubuntu:14.04
   ```

   This will use Ubuntu 14:04 as our base image.

4. Next, we can insert a little metadata that allows people to see who is currently maintaining the image. This allows people to see who authored the image, and who to contact if they have questions or issues. This takes the form of a text field, and it's generally accepted to put your name and e-mail address in it. You can do this by adding the following command in the Docker file:

   ```
   MAINTAINER Example User example@adomain.com
   ```

5. Now that we have taken care of the metadata for our container, we can turn our attention to installing software. As we are using the Ubuntu base image, we use the `apt` package manager to install our software; however, the base image may have an out-of-date package list cached, so it's best to update it. To update the package list, add the `RUN` directive. In your Dockerfile, insert the following code:

   ```
   RUN apt-get update && \
   ```

The `RUN` directive is one you are going to see a lot, as it allows you to run commands within the container. Be careful, though, as you need to ensure that the commands you run are non-interactive; interactive commands will cause your image build to fail as you have no way to interact with it at build time.

6. Notice the `&& \`; this is a shell function that runs a subsequent command if the previous command was successful, allowing us to chain commands in one line. This is useful for keeping the number of Docker layers small. The \ is a line break, allowing you to keep your Dockerfile readable.

When you run a Docker build, each command creates a new layer, and each layer is placed on top of the next, building your eventual Docker image. However, each layer carries a small amount of internal metadata, which although small, can add up. Perhaps more importantly, there is a limit to the amount of layers an image can contain, a constraint of the underlying AUFS filesystem; at the time of writing, the limit is 127 layers. Although you can use alternative file systems with Docker that might remove this limitation, it's worth designing with it in mind.

7. Now, we can start to install our prerequisite software. Since Gollum is a Ruby application, it requires Ruby, plus some additional build tools. Again, we are going to use the `RUN` command and have `apt` install these packages for us. Insert the following code inside your Dockerfile:

   ```
   RUN apt-get update && apt-get install -y ruby1.9.1 ruby1.9.1-dev
   make zlib1g-dev libicu-dev build-essential git
   ```

   This will install the software that we need to install Gollum.

8. Next we want to install Gollum itself. Gollum is distributed as a Ruby gem, so we can use the Gem package manager to install it for us. To do this, add the following code:

   ```
   RUN apt-get update && \
   apt-get install -y ruby1.9.1 ruby1.9.1-dev make zlib1g-dev libicu-
   dev build-essential git && \
   gem install gollum
   ```

   As you can see, we are performing the installation as a chained set of commands rather than using an individual `RUN` directive for each new line.

9. We now need a directory to store our wiki content. Unlike many wikis that rely on a database to store content, Gollum uses a Git repository as its persistent store. All that is required is a file system to store the Git repository on, and it takes care of the versioning. Let's create it now; insert the following code into your Dockerfile:

   ```
   RUN mkdir -p /usr/local/gollum
   ```

10. Now, we are going to set the work directory. By default, Docker runs all directives within the root directory of the container; by setting the work directory, we can run the commands in the directory of our choice. To set the work directory, add the following directive to your Dockerfile:

    ```
    WORKDIR /usr/local/gollum
    ```

11. With the work directory set, we can now create the initial repository to hold our wiki content; this is achieved using the Git command. Add this code to your Dockerfile:

    ```
    RUN git init .
    ```

    This command will be run in the work directory we set in the previous command, and it will create an empty Git repository ready for our content.

12. Now, we need to expose a network port. By exposing the port, we will be able to connect to the service from the network; it also allows other containers to connect to the service via linking. Gollum runs by default on TCP port `4567`; add the following code to expose it:

    ```
    EXPOSE 4567
    ```

13. Finally, we add a default command that will be run when the container is started. In this case, the Gollum package installs a binary that can be used to start the wiki. Add the following command to execute it when the container starts:

    ```
    CMD ["gollum"]
    ```

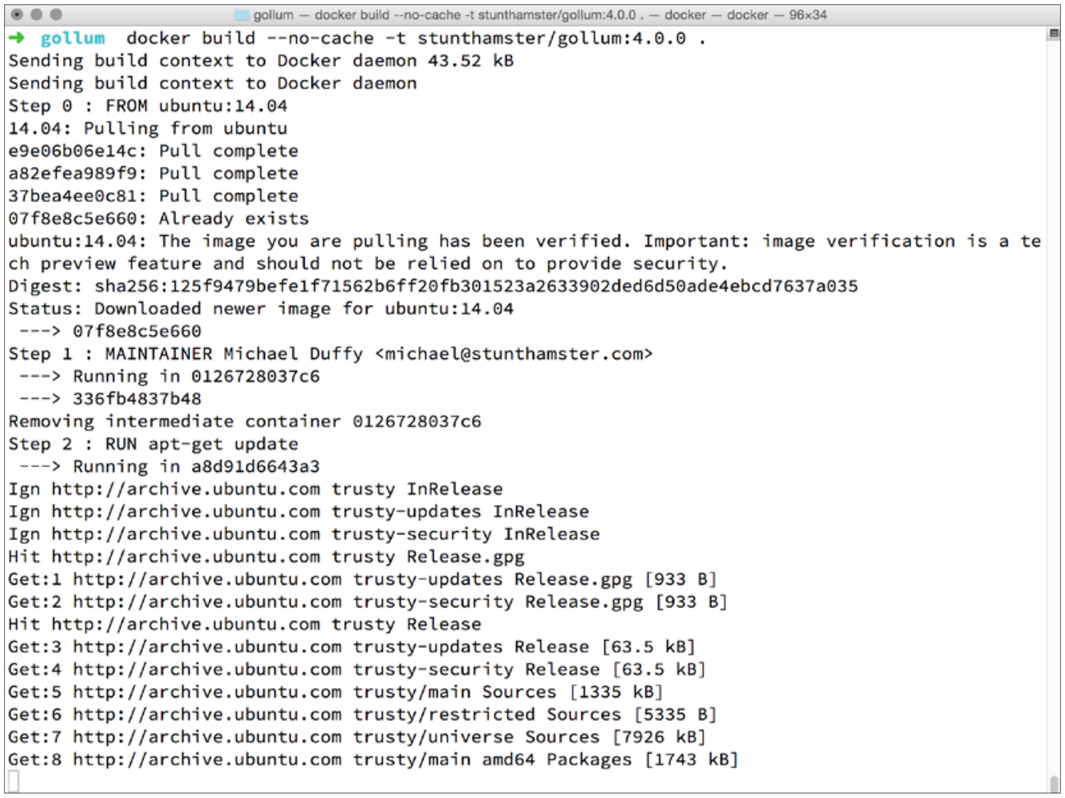
14. We are now ready to build our Docker container. At the command line, navigate to the directory containing your Dockerfile and issue the following command:

    ```
    $ sudo docker build -t <username>/gollum:4.0.0 .
    ```

    Where `<username>` is the Docker registry username that you setup earlier. Notice the `-t`: this is the tag. The tag is used both to name your image and to version it. In this case, I have used the version of the software.

Versioning is as always a contentious issue, and it is best to use your existing standards if in doubt. I tend to create a container version that matches the version of the application I am packaging, as it allows me to see at a glance which host is running which version of a given piece of software.

15. Once you trigger this command, you should see output similar to the following screenshot:



gollum — docker build --no-cache -t stunthamster/gollum:4.0.0 . — docker — docker — 96×34

```
➜ gollum  docker build --no-cache -t stunthamster/gollum:4.0.0 .
Sending build context to Docker daemon 43.52 kB
Sending build context to Docker daemon
Step 0 : FROM ubuntu:14.04
14.04: Pulling from ubuntu
e9e06b06e14c: Pull complete
a82efea989f9: Pull complete
37bea4ee0c81: Pull complete
07f8e8c5e660: Already exists
ubuntu:14.04: The image you are pulling has been verified. Important: image verification is a te
ch preview feature and should not be relied on to provide security.
Digest: sha256:125f9479befe1f71562b6ff20fb301523a2633902ded6d50ade4ebcd7637a035
Status: Downloaded newer image for ubuntu:14.04
 ---> 07f8e8c5e660
Step 1 : MAINTAINER Michael Duffy <michael@stunthamster.com>
 ---> Running in 0126728037c6
 ---> 336fb4837b48
Removing intermediate container 0126728037c6
Step 2 : RUN apt-get update
 ---> Running in a8d91d6643a3
Ign http://archive.ubuntu.com trusty InRelease
Ign http://archive.ubuntu.com trusty-updates InRelease
Ign http://archive.ubuntu.com trusty-security InRelease
Hit http://archive.ubuntu.com trusty Release.gpg
Get:1 http://archive.ubuntu.com trusty-updates Release.gpg [933 B]
Get:2 http://archive.ubuntu.com trusty-security Release.gpg [933 B]
Hit http://archive.ubuntu.com trusty Release
Get:3 http://archive.ubuntu.com trusty-updates Release [63.5 kB]
Get:4 http://archive.ubuntu.com trusty-security Release [63.5 kB]
Get:5 http://archive.ubuntu.com trusty/main Sources [1335 kB]
Get:6 http://archive.ubuntu.com trusty/restricted Sources [5335 B]
Get:7 http://archive.ubuntu.com trusty/universe Sources [7926 kB]
Get:8 http://archive.ubuntu.com trusty/main amd64 Packages [1743 kB]
```

16. Once your build is complete, you can push it to the Docker repository. By pushing your image to the repository, you make it straightforward to deploy it to other machines. To push the container, issue the following command:

```
$ sudo docker push <username>/gollum:4.0.0
```

This will push the image to the Docker repository and make it ready to be distributed. If you wish to make it private, then you can sign up for a premium account and make use of the private repository feature; alternatively, you can host your own Docker registry.

See also

The Docker build documents can be found at: `https://docs.docker.com/reference/builder/`.

Running a container in detached mode

You should run your container in a detached mode. This ensures that the applications that are running within your container are able to run unattended, much in the same way as a daemonized service. In this example, we are going to take the container we created in the previous recipe, and run it as a detached process.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server with Docker installed. You should also have completed the previous recipe, *Creating a Dockerfile*, or have a suitable image to use.

How to do it...

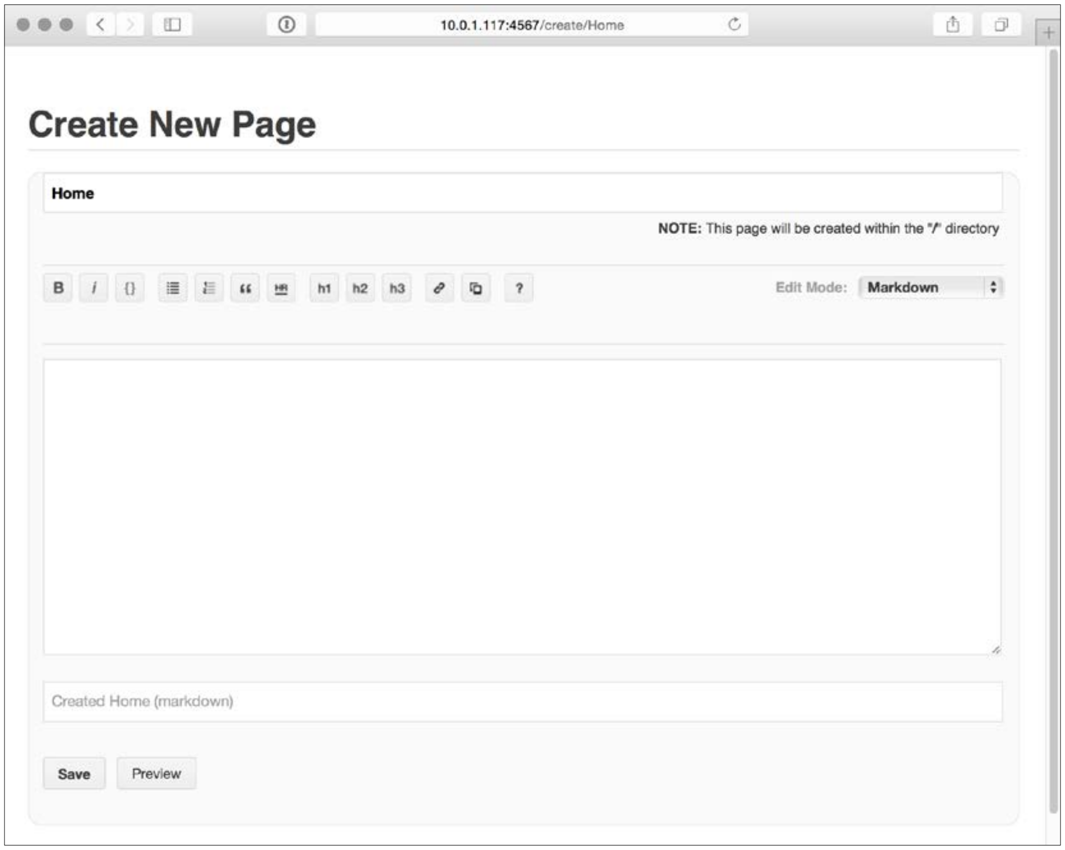
The following steps show you how to use the `docker` command to run a container in a detached fashion:

1. Running a container in detached mode is relatively straight forward. On your Docker host, run the following command:

   ```
   $ sudo docker run -d -t <username>/gollum:4.0.0 --name gollum -p 4567:4567
   ```

2. Let's take a look at these options. The `-d` tells Docker to run the container in a detached mode; this means that it will run in the background, non interactively. Next we use the `-t` option and supply the tag of our image, telling it which image we wish to start our container from. Then, we use the `--name` option to allocate a name to the container; without this option, a random name will be allocated.
3. The final option `(-p)` bridges the network between the container and the host, allowing you to connect to your Gollum Wiki. This is presented as `<container port>:<host port>` and allows you to connect to the host on a different port to the one that is presented by the container; this can be very useful if you want to run multiple versions of the same app, as it allows you to export the service onto several different ports, and use technology such as `haproxy` to load balance between them.

4. Once you have issued the command, you should be able to connect to your new Wiki. In your browser enter the `url ::4567`. You should be presented with a page that looks similar to this:


10.0.1.117:4567/create/Home
Create New Page
Home
NOTE: This page will be created within the "/" directory
B
i
{}
h1
h2
h3
?
Edit Mode:
Markdown
Created Home (markdown)
Save
Preview

See also

- You can find the Docker run command reference at `https://docs.docker.com/reference/run/`
- You can find the Gollum documentation at `https://github.com/gollum/gollum/wiki`

Saving and restoring a container

One of the powers of containers is the flexibility that they provide, and part of this is the ease with which you can take snapshots of running containers and restore them onto other Docker hosts. This can be used both to back up containers and to diagnose issues. If you have a production issue with a particular container, you can use a snapshot to restore the problematic container to a test host and test the exact same container in a controlled environment.

Getting ready

For this recipe, you will need an Ubuntu 14.04 Docker host and a running Docker container.

How to do it...

The following steps will show you how to save a Docker container and then restore it on another machine:

1. First we need to locate the container we are interested in. Using the `docker ps` command, locate the container you wish to snapshot and make a note of its `ID`.
2. Once you have located the container you wish to commit, issue the following command:

   ```
   $ sudo docker commit <containerid> <imagename>:<imageversion>
   --pause=false --author=<yourname>
   ```

   Let's go through some of these options in more detail. The `containerid` should be the `id` of the container you wish to commit, and the `imagename` and `imageversion` are the name and version you wish to give to the image you're creating. The `pause` flag is important for production instances, as this controls the behavior of the running container when committing. By default, Docker will pause the image whilst the commit takes place; this is to ensure that the data is consistently captured. However, it also renders the container unable to serve requests during this time. If you need the container to continue to run, you can use the `--pause=false` flag to ensure this. Be aware, though, the image you create may contain corrupted data if a write takes place during the commit. Finally, we also add an author name, as this helps the people who examine the image to know who took it.

3. Once you have issued the command, you can use the following command to check if it has been created:

   ```
   $ docker images | grep "<imagename:imageversion>"
   ```

   This should show your newly created image.

4. Now that we have our image, we can push it to the Docker repository using the following command:

   ```
   $ docker push -t <imagename:imageversion>
   ```

5. We can now make the image available for diagnosis on a test machine using the following command:

   ```
   $ Docker pull -t  <imagename:imageversion>
   ```

6. Alternatively, you can skip pushing the image to the Docker repository by using the Docker `save` command. The `save` command creates a `tar` file of the image, which is suitable to be passed around with tools such as SCP, or shared file systems such as NFS. To use the `save` command, issue the following at the command line:

   ```
   $ docker save <imagename:imageversion> > /tmp/savedimage.tar
   ```

7. This will create a `tar` file of the image. Copy it to your test host using the tool of your choice, and issue the following command on the test host:

   ```
   $ docker load < savedimage.tar
   ```

Now, if you check the Docker images on your test host, you will find that your saved image is available to run.

See also

You can find details of Docker save, load, and commit commands at: `https://docs.docker.com/reference/run/`.

Using the host only network

Docker uses a bridge to connect to the underlying virtual network and present services within a container, and by and large, this is perfectly satisfactory. However, there are some edge cases where this can cause issues; a perfect example is an application that makes use of multicast. To get around this, you can present the host networking stack to the container, allowing it to make full use of the host network. This allows items such as multicast to work at the expense of some flexibility and convenience.

> Wherever possible, you should avoid using this technique. Although it can help avoid certain issues, it also breaks one of the underlying ideas of containerization by making a container rely on features of the host. It also stops you from being able to run multiple containers that rely on the same port. For instance, under normal circumstances you can run multiple Nginx servers using Docker, and also map the host ports of `80`, `81`, and `82` to three containers listening on port `80`. You cannot do this by tying the host network to the container, as the port is tied to a single process.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host and a container with which it can network.

How to do it...

The following recipe shows you how to run a container in a detached fashion and give it access to the host network:

1. First, we're going to start a new container; in this example, we're going to use the Gollum container from earlier in the chapter. Start it using the following command:

   ```
   docker run -d --name gollum --net=host -t <username>/gollum:4.0.0
   ```

   Note the additional option of `--net=host;` this directs Docker to start the container with the host network rather than bridging. Also note the lack of the `-p` option to map ports; this option becomes superfluous as the container communicates directly with the host network, so no bridge is required.

2. As we are no longer mapping a port, Docker cannot take care of configuring the IP Tables for use. Due to that, you will need to insert a new rule to allow traffic to reach the service running in the container. You can do this by using the following command:

   ```
   iptables -I INPUT 5 -p tcp -m tcp --dport <serviceport> -j ACCEPT
   ```

   Substitute `<<serviceport>>` with the TCP port number of the service that you're running on.

See also

You can find more information on advanced Docker networking techniques at: `https://docs.docker.com/articles/networking/`.

Running a private Docker registry

Although the Docker registry offers a robust and cost effective place to store Docker images, for some companies this can be limiting, either due to the cost involved, or possibly due to security policies. Luckily, it is possible to run your own private Docker repository, allowing you to keep your images completely within the boundaries of your own network.

In this recipe, we are going to set up a minimal Docker registry. We are not going to delve into items such as authentication mechanisms or alternative storage mechanisms. This is left as an exercise for the reader, and you can find excellent guidance within the Docker documentation at: `https://docs.docker.com/registry/`.

The registry we create will contain some minimal security in the form of SSL, and we are going to export the filesystem to the underlying host. Docker registries can consume massive amounts of disk space and ideally, you should hold the data on a robust storage device, such as an NFS server with both a large capacity, and solid redundancy.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host with Docker installed.

How to do it

The following steps will show you how to create a new Docker repository and secure it using SSL:

1. By default, the Docker registry does not ship with SSL enabled. To enable it, we need to download the source code. Create a folder to contain it and download it using the following command:

   ```
   $ wget -q https://github.com/docker/distribution/archive/
   v2.0.0.tar.gz
   ```

2. Next, unpack the source code using the `tar` command:

   ```
   $ tar -xvf v2.0.0.tar.gz
   ```

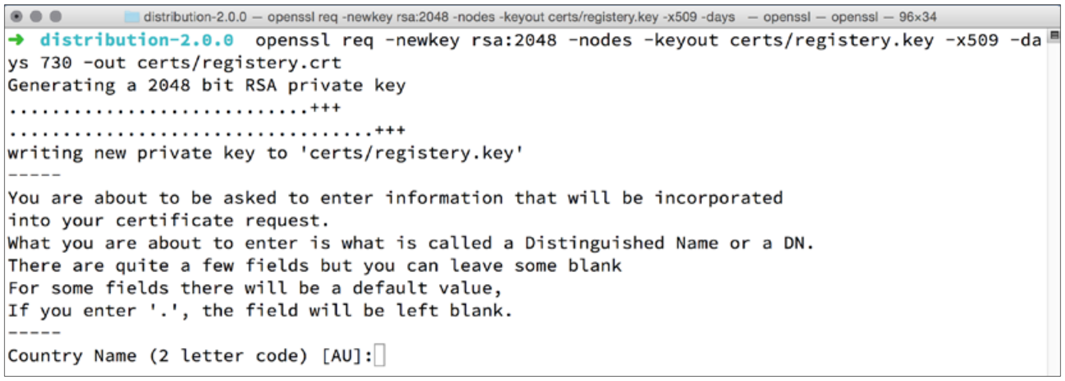
3. Move to the `distribution` directory and create a new `certs` directory using the following command:

   ```
   $ cd distribution-2.0.0 && mkdir certs
   ```

4. Now we create the SSL certificates for our Docker host using the following command:

   ```
   $ openssl req -newkey rsa:2048 -nodes -keyout certs/registery.key
   -x509 -days 730 -out certs/registery.crt
   ```

5. This command will trigger some prompts asking for further details about the new certificate you are creating:



```
distribution-2.0.0 — openssl req -newkey rsa:2048 -nodes -keyout certs/registery.key -x509 -days  — openssl — openssl — 96×34
➜  distribution-2.0.0  openssl req -newkey rsa:2048 -nodes -keyout certs/registery.key -x509 -da
ys 730 -out certs/registery.crt
Generating a 2048 bit RSA private key
............................+++
..................................+++
writing new private key to 'certs/registery.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
```

Pay special attention to the hostname when you fill these in, as this should match the hostname of registry you are creating.

6. Next, we need to amend the registry configuration to recognize the new certificates. Edit the `/cmd/registry/config.yml` file within the registry source code and find the block marked http, then amend the code to look similar to this:

```
http:
    addr: :5000
    secret: asecretforlocaldevelopment
    debug:
        addr: localhost:5001
    tls:
        certificate: /go/src/github.com/docker/distribution/certs/
registry.crt
        key: /go/src/github.com/docker/distribution/certs/
registry.key
```

7. Next, locate the key named `filesystem`: and amend it so that it resembles the following snippet:

```
    filesystem:
        rootdirectory: /var/spool/registry
```

8. Now we have finished our changes, we can build our custom registry image using the following `docker build` command:

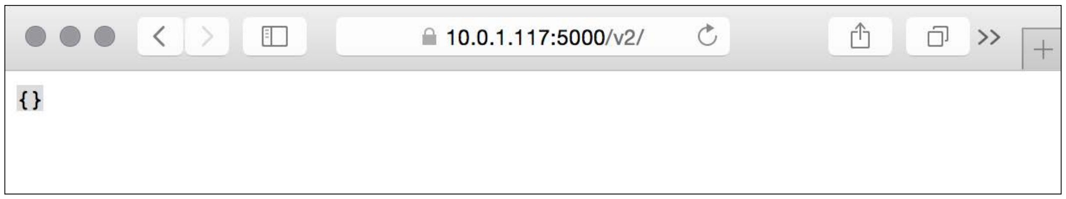
```
$ docker build -t docker_registry .
```

9. Once the build is complete, you can run the registry using the following command:

```
$ docker run -p 5000:5000 -v /var/spool/registry:<host_dir>
docker_registry:latest
```

Where `host_dir` is a directory on the host machine.

10. Since we are using a self-signed certificate, we need to configure any Docker client that wishes to use this repository to recognize the new certificate. On each Docker client, copy the `registry.crt` into `/etc/docker/certs.d/<<registrydomain>>:<<registryport>>/ca.crt`. Ensure that you replace `registrydomain` with the DNS name of your Docker registry, and the `registryport` with the port it will be running on. This will then suppress any warnings you may encounter due to untrusted certificates.
11. We can check if our registry has started correctly by querying the API. Point your browser at the address of your new registry (remember to ensure it's `https` rather than `http`). You should see a response similar to this:



```
{}
```

This is exactly what we expect; by returning a `200` response and an empty JSON array, you can see that the API is ready to go.

See also

You can find details on how to deploy and use the Docker Registry at: `https://docs.docker.com/registry/`.

Managing images with a private registry

Once you have created your own Docker registry, you can start to push your Docker images to it. By pushing your images to a self-hosted Docker registry, you are not only gaining security, you are also making the build and deployment of further images much faster. Pushing to a self-hosted registry is straightforward, and there is nothing to stop you from pushing an image to multiple registries; this can be useful if you maintain registries for certain environments.

Getting ready

For this recipe, you will need an Ubuntu 14.04 Docker host and a self-hosted Docker registry.

How to do it...

I have split this recipe into two sections, the first dealing with pushing images to your registry, and the second for how to pull images.

Pushing images

The following steps deal with taking an image and pushing it to a private Docker registry:

1. For this recipe, we are going to use the Gollum Dockerfile, which we created previously. Change the directory to the Dockerfile directory and trigger a build using this command:

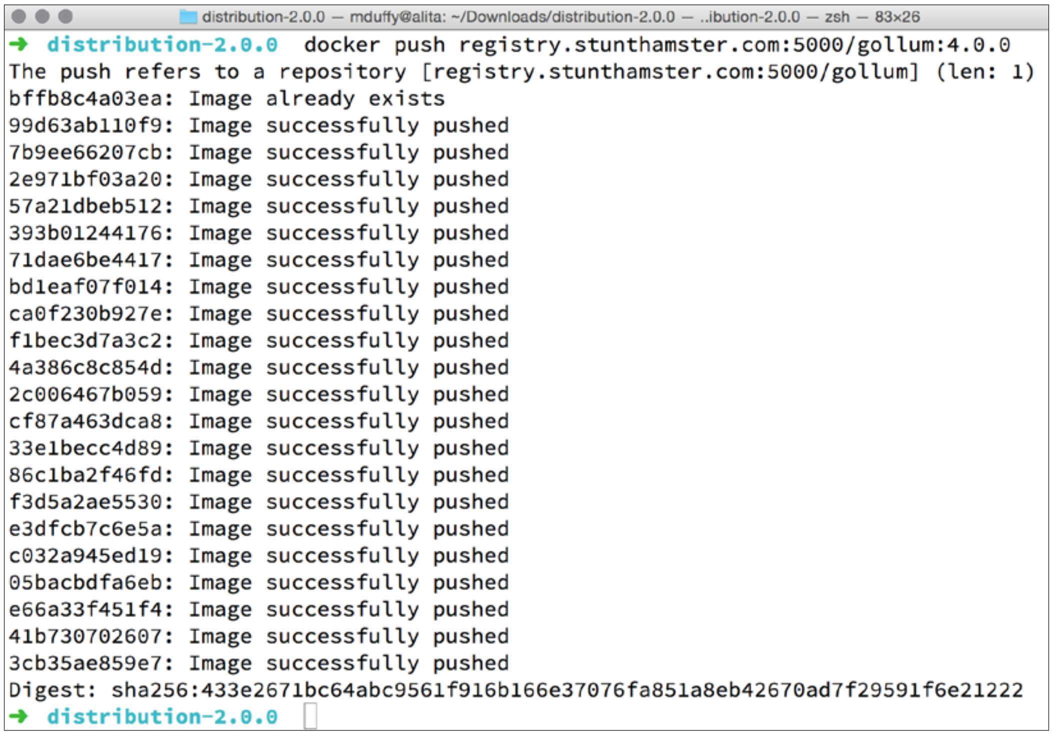
   ```
   $ sudo docker build -t <username>/gollum:4.0.0 <<docker_registry_
   name_and_port>>
   ```

   Note that, instead of your usual Docker registry username, you are inserting the name and port number of your own Docker registry into the image tag. This is going to act as a pointer to Docker, for both where to push the image to and where to pull it from.

2. Next, we can push the image using the following command:

   ```
   $ docker push <docker_registry_name_and_port>/gollum:4.0.0
   ```

3. If all goes well, you should see output similar to this:

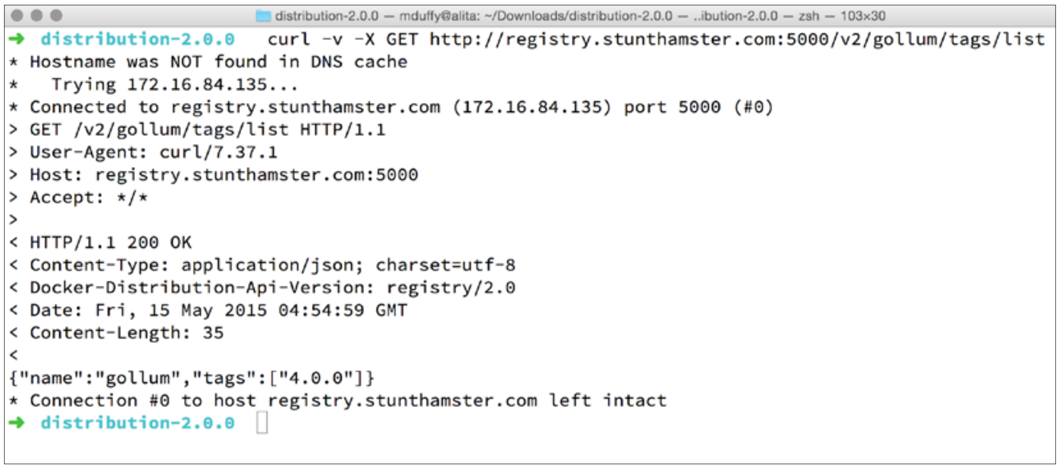

distribution-2.0.0 — mduffy@alita: ~/Downloads/distribution-2.0.0 — ..ibution-2.0.0 — zsh — 83×26

```
➜  distribution-2.0.0  docker push registry.stunthamster.com:5000/gollum:4.0.0
The push refers to a repository [registry.stunthamster.com:5000/gollum] (len: 1)
bffb8c4a03ea: Image already exists
99d63ab110f9: Image successfully pushed
7b9ee66207cb: Image successfully pushed
2e971bf03a20: Image successfully pushed
57a21dbeb512: Image successfully pushed
393b01244176: Image successfully pushed
71dae6be4417: Image successfully pushed
bd1eaf07f014: Image successfully pushed
ca0f230b927e: Image successfully pushed
f1bec3d7a3c2: Image successfully pushed
4a386c8c854d: Image successfully pushed
2c006467b059: Image successfully pushed
cf87a463dca8: Image successfully pushed
33e1becc4d89: Image successfully pushed
86c1ba2f46fd: Image successfully pushed
f3d5a2ae5530: Image successfully pushed
e3dfcb7c6e5a: Image successfully pushed
c032a945ed19: Image successfully pushed
05bacbdfa6eb: Image successfully pushed
e66a33f451f4: Image successfully pushed
41b730702607: Image successfully pushed
3cb35ae859e7: Image successfully pushed
Digest: sha256:433e2671bc64abc9561f916b166e37076fa851a8eb42670ad7f29591f6e21222
➜  distribution-2.0.0
```

4. Now, we can also use the Registry API to check if our image has been correctly pushed. At the command line, enter the following command:

   ```
   curl -v -X GET  address>:5000/v2/gollum/tags/list
   ```

5. You should see a response along the lines of the following screenshot:


distribution-2.0.0 — mduffy@alita: ~/Downloads/distribution-2.0.0 — ..ibution-2.0.0 — zsh — 103×30

```
➜  distribution-2.0.0   curl -v -X GET http://registry.stunthamster.com:5000/v2/gollum/tags/list
* Hostname was NOT found in DNS cache
*   Trying 172.16.84.135...
* Connected to registry.stunthamster.com (172.16.84.135) port 5000 (#0)
> GET /v2/gollum/tags/list HTTP/1.1
> User-Agent: curl/7.37.1
> Host: registry.stunthamster.com:5000
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=utf-8
< Docker-Distribution-Api-Version: registry/2.0
< Date: Fri, 15 May 2015 04:54:59 GMT
< Content-Length: 35
<
{"name":"gollum","tags":["4.0.0"]}
* Connection #0 to host registry.stunthamster.com left intact
➜  distribution-2.0.0
```

If you see this, then you have successfully pushed the image to your own Registry.

Pulling images

Now that we have pushed images to our Docker registry, we can look at how we can subsequently pull them. Again, we can use the Gollum example, which we uploaded in the previous examples.

At your command line, enter the following command:

```
$ docker run -d -t <<docker_registry_name_and_port>>/gollum:4.0.0
```

This will pull the image from your registry and run it.

See also

You can find more details of working with a self-hosted Docker registry at: `https://docs.docker.com/registry/spec/api/#overview`.

7 Using Jenkins for Continuous Deployment

In this chapter, we are going to cover the following topics:

- Installing Jenkins
- Installing the Git plugin
- Installing a Jenkins slave
- Creating your first Jenkins job
- Building Docker containers using Jenkins
- Deploying a Java application to Tomcat with zero downtime using Ansible

Introduction

Continuous integration is one of the most powerful techniques you can use when developing software and it underpins a great deal of what many consider a DevOps tool-chain. **Continuous integration** (**CI**) essentially entails taking your code and building it on a frequent schedule and deploying it into a representative environment for the purpose of testing. This automated job should both build and test, if the tests are passed, you can deploy your software into a nominated environment. Without the ability to automate code deployment, you are left with an enormous piece of manual labor in your deployment pipeline. It's one thing to be able to build servers and deploy configuration automatically, but if you are unable to build your code in a reliable manner and then push it to a test environment, then you are going to be wasting a lot of time and energy.

Continuous integration is an incredibly valuable tool and something that most development teams should be working towards if they don't already have it. Once you have a CI tool in place you will be deploying and testing code very frequently and increasing the visibility of bugs before they can be deployed any further than an integration environment.

How often you run the integration job is something for the team to agree on. I've worked with systems that deploy every time a developer checks in and I've also worked in teams where it happens once every hour. I will suggest that you run the integration at least once a day. If your integration tests are long, the best time will be when the last developer goes home; so that, when you return the next day, you can see the state of your last build and fix any issues that may have occurred overnight.

Taken to extreme, continuous integration can be used to take code from the repository and take it right through to production deployments and indeed, there are companies that utilize continuous integration in this fashion. Even if you don't take your system to such a degree, you will still find that the judicious use of continuous integration can help you identify bugs in the code quicker.

The key to continuous integration is to leverage the existing tools. You almost have the existing testing suites in place, so re-purpose those to be used in the continuous integration environment. We can also re-use the automation tools and techniques we use to build the environments. This not only re-uses existing tools but also ensures that any deployment tool you use is tested regularly, ensuring that these are as bug free as possible.

In this chapter, we are going to focus on using Jenkins as the basis of our continuous integration recipes. Jenkins is a fork of the Hudson CI system and has a thriving and active development community with many plugins to enhance its functionality.

Although Jenkins is a fork of Hudson, I recommend that you stick with Jenkins rather than its progenitor. It has a much more active developer community and more plugins.

Jenkins allows the use of a master and slave arrangement and this allows you to scale it out for truly massive builds. The Jenkins master controls the slaves and with a few select plugins, you can use technologies such as Docker to keep your build environment elastic; alternately, you can use plugins to drive build slaves using AWS, Digital Ocean, and many other PAAS providers.

We will also be making use of both Ansible and Docker to build and configure our integration environment. If you are not up to speed with either of these, I suggest that you have a look at *Chapter 5*, *Automation with Ansible*, and *Chapter 6*, *Containerization with Docker*. Both of these chapters contain everything you need to get up and running with these technologies.

Installing Jenkins

This recipe will show you how to install a basic Jenkins server. The Jenkins server will form the basis of the continuous integration environment, and this is where you define your build jobs and also manage build users, plugins, and environment details. We are also going to cover some basic setup tasks, ensuring that your new Jenkins servers is secured from anonymous usage is a very important step if your hosting your build platform on a publicly accessible host.

We are going to use Ansible to install our Jenkins master; this allows us to easily re-create the basic server if we have a problem or allows us to create a new one if we need a second master, perhaps for a new project.

This recipe will setup the Jenkins master but you will quickly realize that the Jenkins master is not the important part; the crucial parts are the jobs you create. Make sure that once you start to use Jenkins, you back up your system regularly. If disaster strikes and you haven't backed up your Jenkins master, you will have a very tedious time re-creating jobs from scratch.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server and an Ubuntu 14.04 host with Ansible installed. If you need more information on how to use Ansible, please consult *Chapter 5, Automation with Ansible*.

How to do it...

The following steps will show you how to write a basic Ansible role to install the Jenkins server and how then to setup some basic security:

1. We're going to create a new Ansible role to manage the installation of our Jenkins Master; use the following command to create the new role:

   ```
   ansible-galaxy init jenkins
   ```

2. Now, we have our new template role ready and we can start adding code. Edit `jenkins/tasks/main.yml` and add the following snippet:

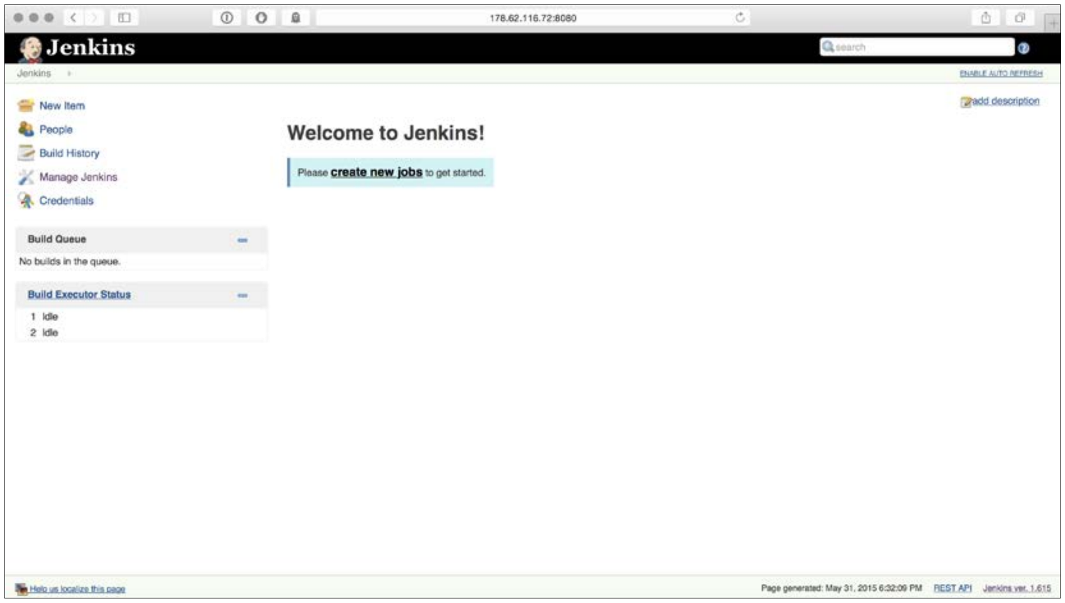
   ```
   - name: Add Apt key
     apt_key: url=http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key
   state=present
   - name: Add Jenkins Repository
     apt_repository: repo='deb http://pkg.jenkins-ci.org/debian
   binary/' state=present update_cache=yes
   ```

```
- name: Install Jenkins
  apt: name=jenkins state=present

- name: Start Jenkins
  service: name=jenkins state=started
```

This code is straightforward and performs the following tasks:

- Adds the Jenkins repository key
- Adds the Jenkins repository to the apt sources
- Installs the Jenkins package
- Starts the Jenkins service.

If you now use your browser to point at the DNS/IP address of your build server on port `8080` you should be presented with the following page:



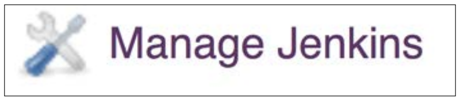
Welcome to Jenkins!

Please **create new jobs** to get started.

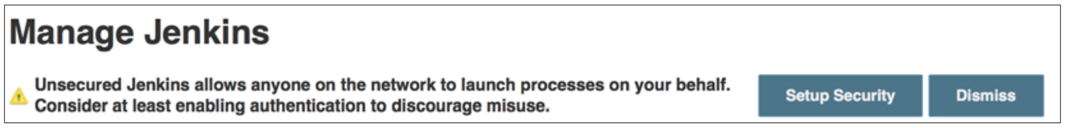
No builds in the queue.

Page generated: May 31, 2015 6:32:09 PM

3. Now that you have installed Jenkins, we should at least add a minimum amount of security; this is especially important if you are hosting your Jenkins server on a public facing server. Click on the **Manage Jenkins** on the left-hand side of the home page marked as:

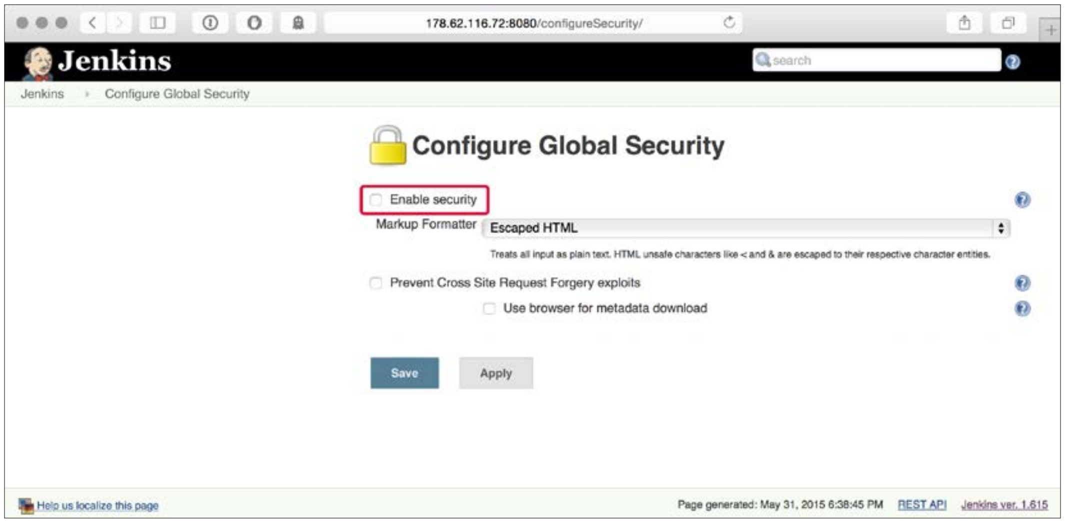


4. The next page allows you to manage some of the features of Jenkins. If you have not already setup your security, you should find a banner that looks similar to the following screenshot at the top of the screen:


Manage Jenkins

Unsecured Jenkins allows anyone on the network to launch processes on your behalf.
Consider at least enabling authentication to discourage misuse.

5. Click on the button marked as **Setup Security**. This should take you to the next screen, which looks like the following screenshot:


178.62.116.72:8080/configureSecurity/

Jenkins › Configure Global Security

Configure Global Security

Enable security

Markup Formatter: Escaped HTML

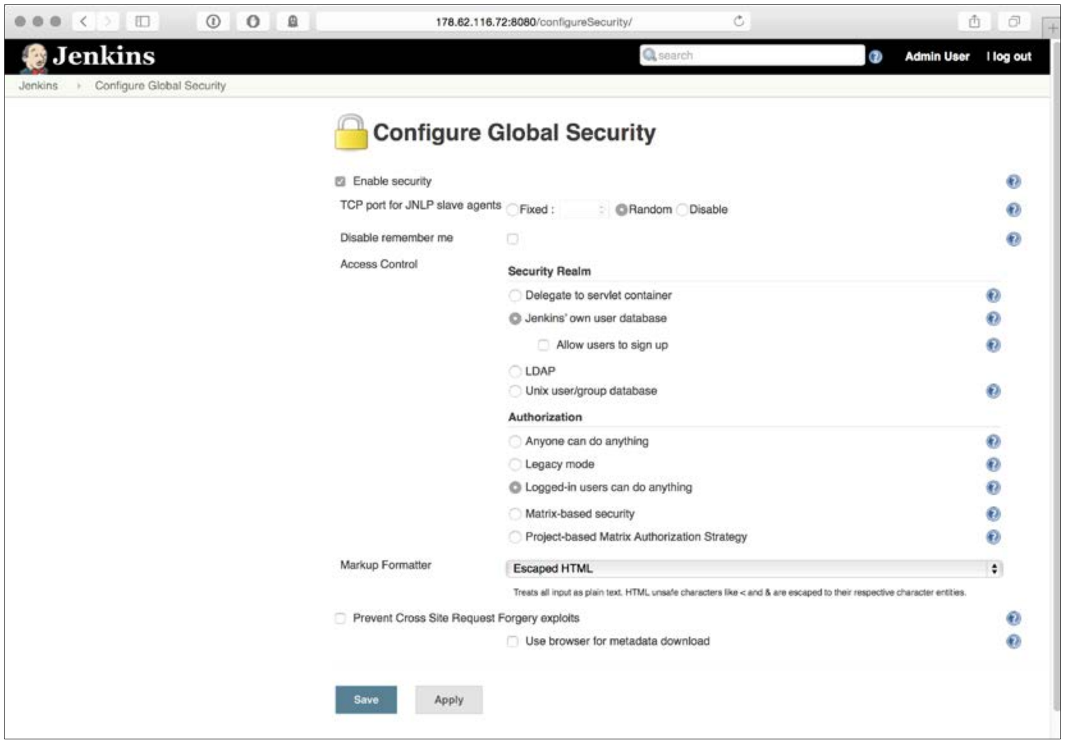
Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Prevent Cross Site Request Forgery exploits

Use browser for metadata download

Page generated: May 31, 2015 6:38:45 PM REST API Jenkins ver. 1.615

6. Check the highlighted checkbox and then click on the **Save** button; it will take you to the next screen of the setup, which should resemble the following screenshot:


178.62.116.72:8080/configureSecurity/
Jenkins
search
Admin User
log out
Jenkins › Configure Global Security

Configure Global Security

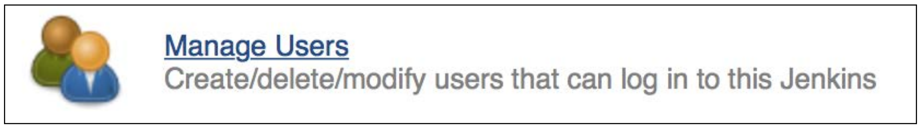
- [✓] Enable security
- TCP port for JNLP slave agents: Fixed : [✓ Random] Disable
- Disable remember me: [ ]
- Access Control
  - Security Realm
    - Delegate to servlet container
    - [✓ Jenkins' own user database]
      - [ ] Allow users to sign up
    - LDAP
    - Unix user/group database
  - Authorization
    - Anyone can do anything
    - Legacy mode
    - [✓ Logged-in users can do anything]
    - Matrix-based security
    - Project-based Matrix Authorization Strategy
- Markup Formatter: Escaped HTML
  - Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.
- [ ] Prevent Cross Site Request Forgery exploits
  - [ ] Use browser for metadata download

Note the selected settings. This will configure Jenkins to use it's own in built user directory and ensure that only logged in users are able to perform any actions.

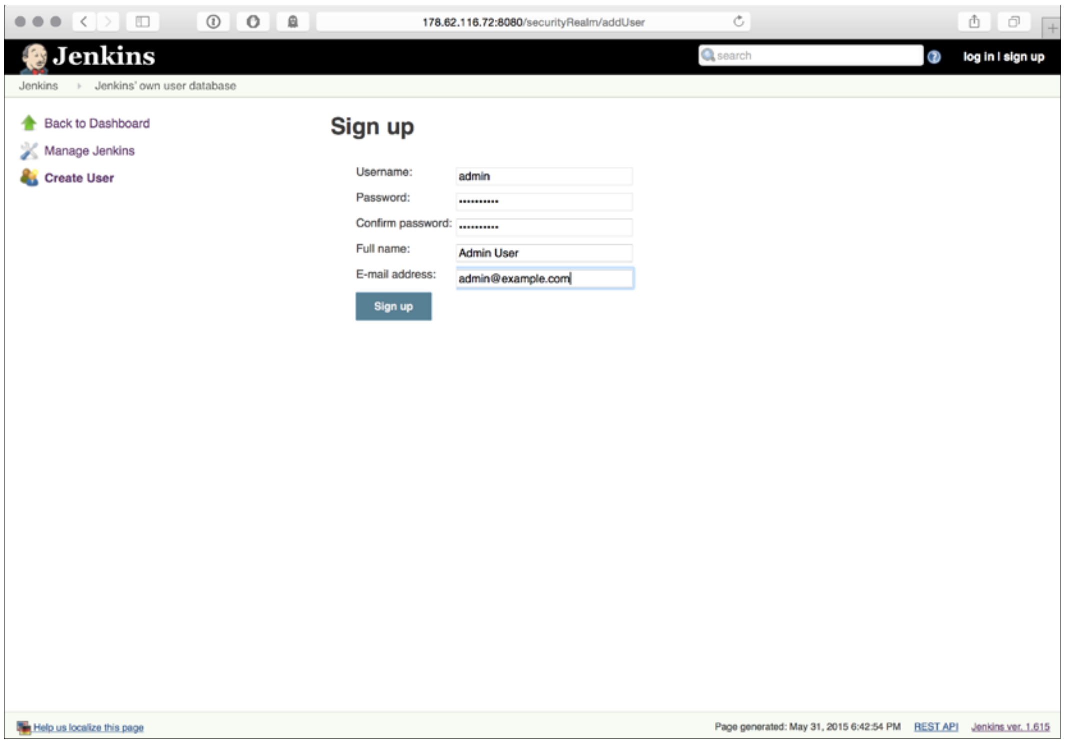
This only stops users from being able to perform actions within Jenkins, but it does not stop unauthenticated users from seeing information on your build server. If you are hosting this on an externally available site, I strongly urge you to get additional security; at the very least a reverse proxy, such as an Apache or Nginx server in front of your Jenkins with basic authentication. Alternatively, you can also add a user called **Anonymous**, and remove all the rights; this will effectively stop unauthenticated users from being able to take any actions or see any data.

Once you are happy with the settings, hit **Save** and you will be taken back to the configuration page.

7. Finally, we need to create a user to log in with. From the management page, find and click on the link that looks like this:



8. This will take us to the user management page. On the left-hand side, you will find a link called **Create User**, click on it and you will be taken to a page that looks similar to the following screenshot:


178.62.116.72:8080/securityRealm/addUser
Jenkins
search
log in | sign up
Jenkins
Jenkins' own user database
Back to Dashboard
Manage Jenkins
Create User
Sign up
Sign up
Help us localize this page
Page generated: May 31, 2015 6:42:54 PM
REST API
Jenkins ver. 1.615

Username: admin
Password: ....
Confirm password: ....
Full name: Admin User
E-mail address: admin@example.com

Add the details of your user as shown in the preceding instance and hit the **Sign up** button. You should now be able to login to Jenkins as that user and start to create jobs. If you don't supply user details, you can still see some of the less secure elements of Jenkins; however, you will be unable to alter anything.

See also...

- You can find further details of how to install Jenkins at `https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins`
- You can find details of Jenkins authentication methods at `https://wiki.jenkins-ci.org/display/JENKINS/Authentication`
- You can find details of how to administer Jenkins at `https://wiki.jenkins-ci.org/display/JENKINS/Administering+Jenkins`

Installing the Git plugin

By default, Jenkins is shipped with the ability to connect to CVS and Subversion; despite the growing popularity of Git, it's still not available by default. Fortunately, Jenkins has a plugin that allows you to use Git as your repository of choice. This recipe will show you how to install the plugin and configure the credentials to connect to your repo.

Getting ready

For this recipe, you will need a Jenkins server and a Git repo to connect to.

How to do it...

The following steps use a combination of Ansible and a Jenkins plugin to add Git client functionality to the Jenkins Server.

1. First, we need to install the Git client on to our Jenkins server. Rather than doing this manually, we are going to adjust the Ansible role from the installation recipe. Within the Jenkins role, edit the following file: `Jenkins/tasks/main.yml`, and insert the following code:

   ```
   - name: Install Jenkins
     apt: name=jenkins state=present
   - name: Install Git
     apt: name=git state=present
   ```

2. Re-run the Ansible role against your server; this should install the Git client. You can test this by issuing the following command on your Jenkins server:

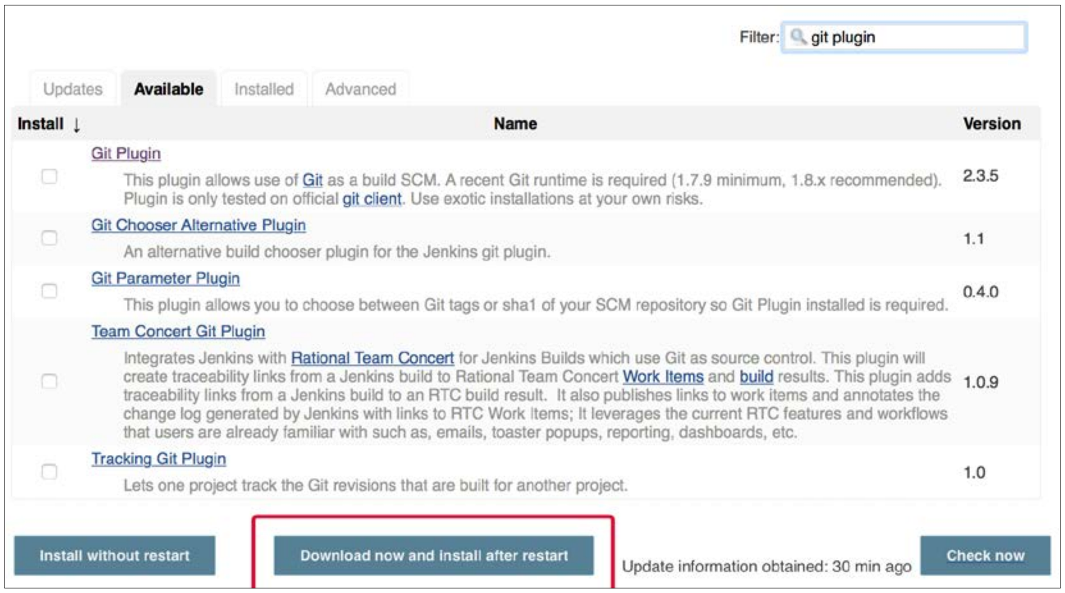
   ```
   git --version
   ```

   This should return the version of the Git client that you installed.

3. Now, since we have the Git client installed, we can install the Jenkins Git plugin. From the front page of the Jenkins console, click on the **Manage Jenkins** button found on the left-hand side of the page.

> Although not covered in this recipe, it's possible to use Ansible to manage the plugins, a great example can be found at: `https://github.com/ICTO/ansible-jenkins`.

4. On the manage Jenkins page, find and click on the **Manage Plugins** button, found around halfway down the page and then click on the **Available** tab.
5. In the filter box, type **Git Plugin**; it should return a list of plugins similar to the following screenshot:



Install ↓	Name	Version
☐	Git Plugin This plugin allows use of Git as a build SCM. A recent Git runtime is required (1.7.9 minimum, 1.8.x recommended). Plugin is only tested on official git client. Use exotic installations at your own risks.	2.3.5
☐	Git Chooser Alternative Plugin An alternative build chooser plugin for the Jenkins git plugin.	1.1
☐	Git Parameter Plugin This plugin allows you to choose between Git tags or sha1 of your SCM repository so Git Plugin installed is required.	0.4.0
☐	Team Concert Git Plugin Integrates Jenkins with Rational Team Concert for Jenkins Builds which use Git as source control. This plugin will create traceability links from a Jenkins build to Rational Team Concert Work Items and build results. This plugin adds traceability links from a Jenkins build to an RTC build result. It also publishes links to work items and annotates the change log generated by Jenkins with links to RTC Work Items; It leverages the current RTC features and workflows that users are already familiar with such as, emails, toaster popups, reporting, dashboards, etc.	1.0.9
☐	Tracking Git Plugin Lets one project track the Git revisions that are built for another project.	1.0

Update information obtained: 30 min ago

> As you have probably noticed, each of the plugins has a link in the title; this takes you through to the documentation and is worth looking at before installing them.

Tick the checkbox next to the **Git Plugin**, click on the **Download now, and install after restart** button. This will prompt Jenkins to download the plugin, install it, and restart Jenkins to make it available for use.

6. Next, we need to configure our credentials to connect to Git.

When setting up Git credentials, I recommend that at the very least, your build system should have its own credentials. Although it's tempting to re-use existing credentials, it makes it both hard to audit and more susceptible to intrusion. It's also creating a problem for the future, when the builds stop working and you revoke the key of the person who setup the build server when they leave.

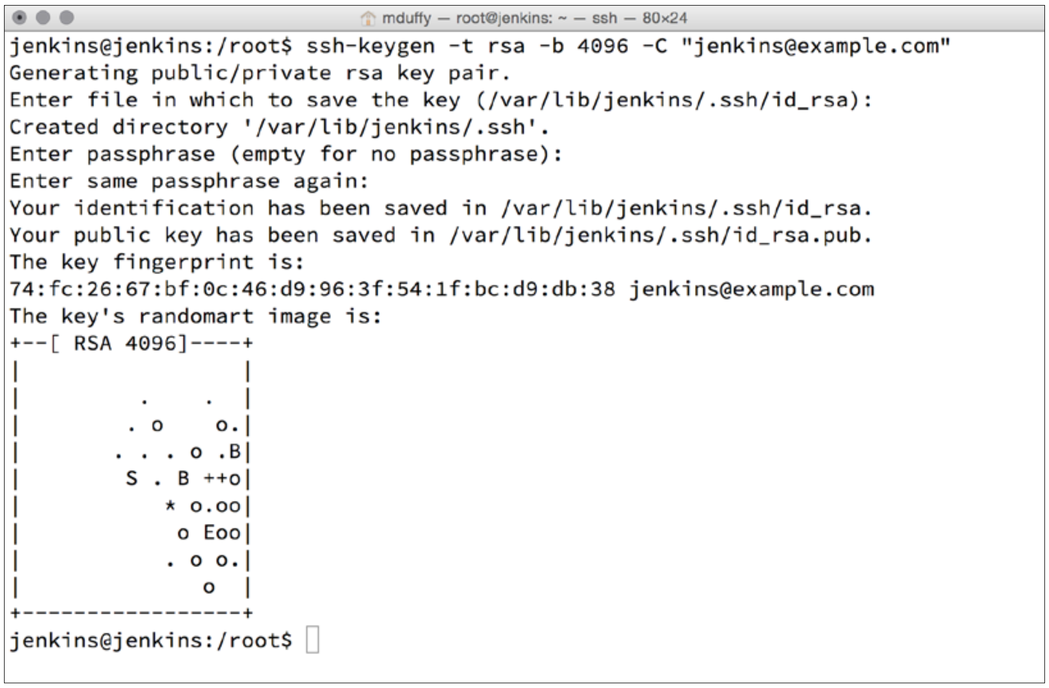
7. Log on to your Jenkins server and issue the following command:

   ```
   $ sudo su Jenkins
   ```

8. Now, we can create a new SSH key with the following command:

   ```
   $ ssh-keygen -t rsa -b 4096 -C "jenkins@example.com"
   ```

9. You will be greeted with a response similar to the following screenshot:

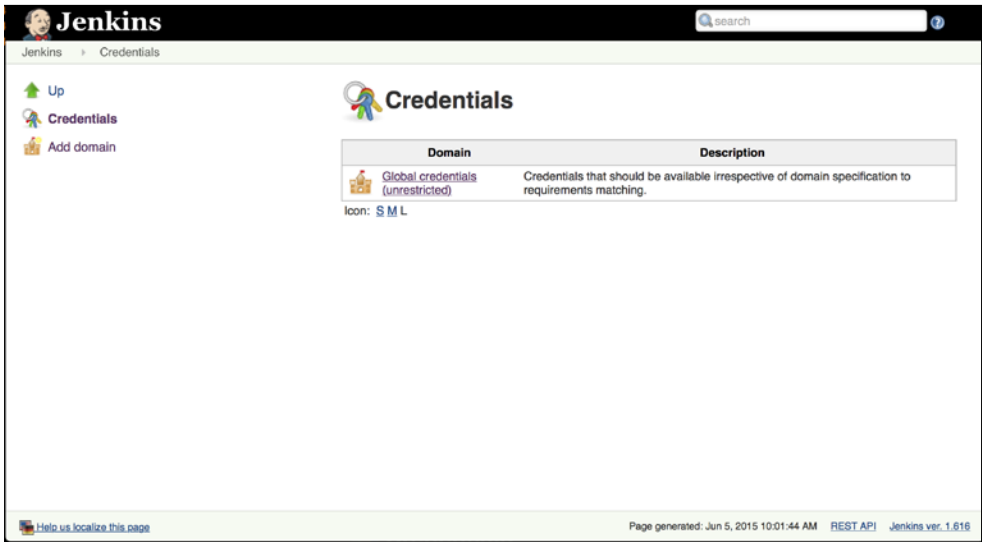


```
jenkins@jenkins:/root$ ssh-keygen -t rsa -b 4096 -C "jenkins@example.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/jenkins/.ssh/id_rsa):
Created directory '/var/lib/jenkins/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/jenkins/.ssh/id_rsa.
Your public key has been saved in /var/lib/jenkins/.ssh/id_rsa.pub.
The key fingerprint is:
74:fc:26:67:bf:0c:46:d9:96:3f:54:1f:bc:d9:db:38 jenkins@example.com
The key's randomart image is:
+--[ RSA 4096]----+
|                 |
|        .     .  |
|       . o    o.|
|      . . . o .B|
|       S . B ++o|
|          * o.oo|
|           o Eoo|
|          . o o.|
|             o  |
+-----------------+
jenkins@jenkins:/root$ 
```

As you can see, I've left the responses at their default values.

This will create the key without a password. If you wish, you can create a key with a password and the password can be passed via Jenkins.

10. Log on to your Jenkins server and click on the link on the left-hand side marked as **Credentials**. This will take you into credentials management page, which should look something similar to the following screenshot:



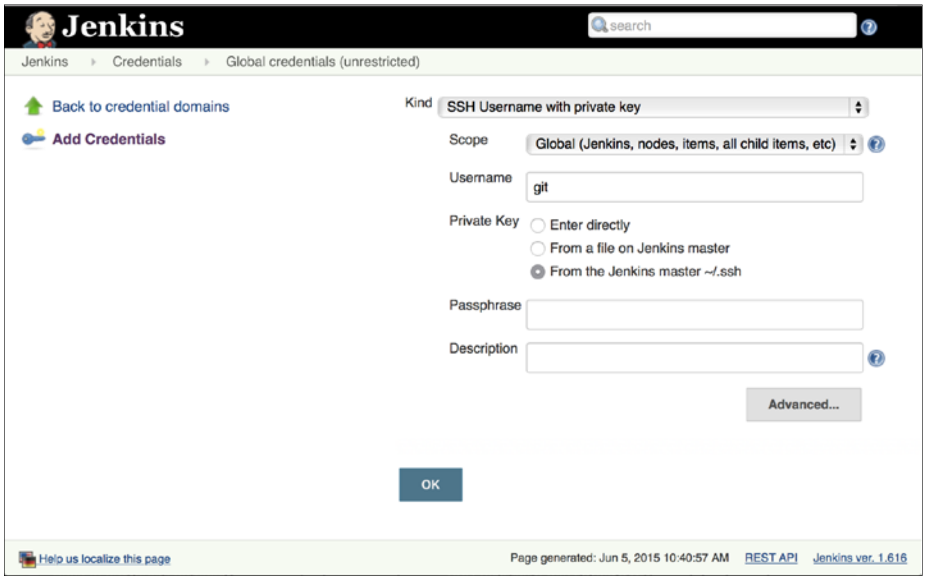
Credentials

Domain	Description
Global credentials (unrestricted)	Credentials that should be available irrespective of domain specification to requirements matching.

Icon: S M L

Help us localize this page
Page generated: Jun 5, 2015 10:01:44 AM
REST API
Jenkins ver. 1.616

11. Click on the global credentials link and select the link on the left-hand side marked as **Add Credentials**; this will bring up a screen that will allow you to add your Git credentials; look at the following screenshot to get an idea of how it should look when you enter your credentials:


Jenkins
search
Jenkins › Credentials › Global credentials (unrestricted)
Back to credential domains
Add Credentials

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: git

Private Key: Enter directly / From a file on Jenkins master / [✓ From the Jenkins master ~/.ssh]

Passphrase:

Description:

Advanced...
OK
Help us localize this page
Page generated: Jun 5, 2015 10:40:57 AM
REST API
Jenkins ver. 1.616

As you can see, we're asking the Jenkins server to simply look-up the key that we generated earlier. Alternatively, you can add it directly to Jenkins or keep it in a different place than the SSH default on the server itself. Tweak these details to fit your build infrastructure.

See also

- You can find the details of the Git plugin at `https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin`
- You can find more details about the credentials plugin at `https://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin`

Installing a Jenkins slave

As we've already covered, Jenkins can be scaled by adding additional build slaves. These slaves can be used to distribute builds amongst many different servers. This allows you to have a single low-powered server, which acts as the Jenkins master and then as many slaves as you need to perform the build jobs.

Getting ready

For this recipe, you will need a Jenkins master and a server running Ubuntu 14.04 with a JDK installed to act as the slave.

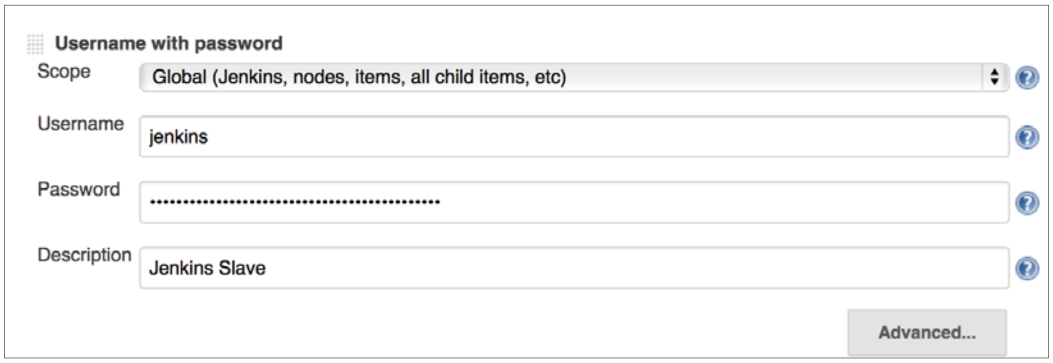
How to do it...

The following steps will show you how to add a Jenkins slave to the Jenkins master:

1. On your slave node, add a new Jenkins-user with the following command:

   ```
   $ adduser jenkins
   ```

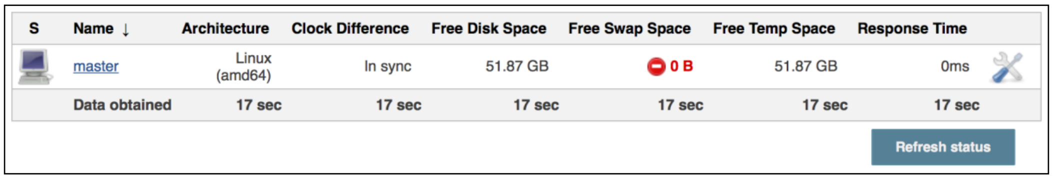
   Login to your Jenkins Master and add new credentials. These are going to be the credentials used to connect to your new Slave; it should look similar to the following screenshot:


Username with password

Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	jenkins
Password	••
Description	Jenkins Slave

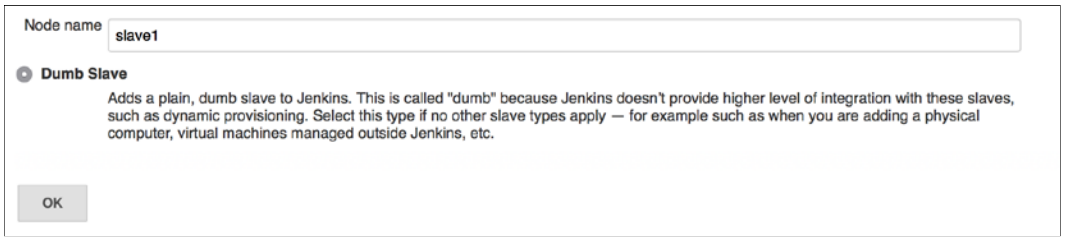
As you can see, I'm using a `username` and `password` to connect to my slave; however, you can also use an SSH key if you prefer.

2. Once you have created your credentials, you now need to add the new slave. On the main screen, click on **Manage Jenkins** and then select the option marked as **Manage Nodes**. Inside the management panel, you should find that the master is already listed and should look something similar to this:



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Linux (amd64)	In sync	51.87 GB	0 B	51.87 GB	0ms
	Data obtained	17 sec	17 sec	17 sec	17 sec	17 sec	17 sec

3. Click on the button on the left-hand side of the management panel marked as **New Node**; this will open the following dialog box:

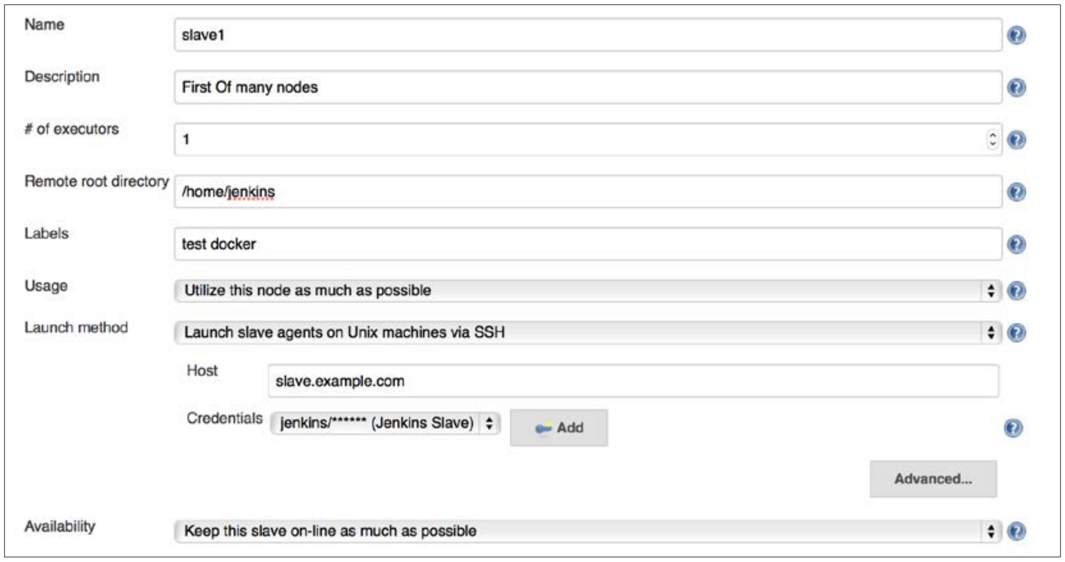

Node name: slave1

Dumb Slave

Adds a plain, dumb slave to Jenkins. This is called "dumb" because Jenkins doesn't provide higher level of integration with these slaves, such as dynamic provisioning. Select this type if no other slave types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

As you can see, you need to name your node. This is descriptive and doesn't necessarily need to the be same name as the actual node. As you can see, there is only one option available for slave nodes but you can add more via an appropriate plugin. Click on **OK** to move to the next step.

4. The next step allows you to enter the details of your slave node; have a look at the following screenshot:



Name	slave1
Description	First Of many nodes
# of executors	1
Remote root directory	/home/jenkins
Labels	test docker
Usage	Utilize this node as much as possible
Launch method	Launch slave agents on Unix machines via SSH
Host	slave.example.com
Credentials	jenkins/****** (Jenkins Slave)
Availability	Keep this slave on-line as much as possible

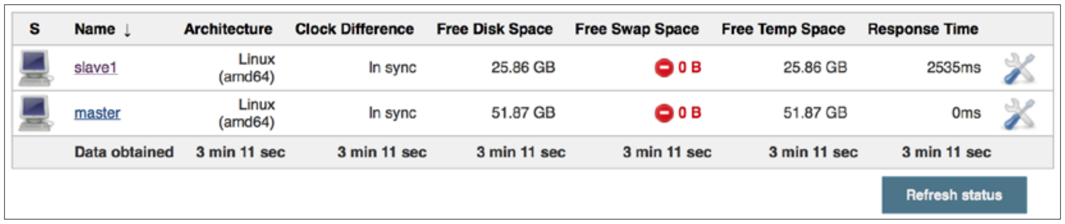
These options allow you to configure some fairly important settings for your node. First of all, it allows you to set the name and description of this node; It's best to be as descriptive as possible. Next, you can set the number of executors and this sets the number of parallel jobs the node can perform at one time. Generally speaking, the more powerful the server, the more jobs it should be able to perform. Mine is a tiny server, so I'll leave it at the default of one.

Next up, we have the remote root directory; this is where the Jenkins workspaces will be kept. So it's best to make sure that this is a location that has the majority of the disk space. Next, we have labels. Labels are a mechanism to allow you to use certain nodes for certain jobs; for instance, in my case, I've set two tags on my `slave`, `test` and `Docker`. This means that any Jenkins job that has that tag will have this node available to them. If no node with that tag exists, it will fail.

[This is a great way to separate out nodes by capability; so for instance, you could label some nodes as being Redhat, Ubuntu, Beta, and so on.]

Finally, we have our launch options. In the case of my slave, I'm using SSH to connect to it using the credentials we created in step 2. Once you have entered your details, click on **Save**.

5. If you now navigate to the node management page, you should find that it now looks something similar to this:



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	slave1	Linux (amd64)	In sync	25.86 GB	⛔ 0 B	25.86 GB	2535ms	
	master	Linux (amd64)	In sync	51.87 GB	⛔ 0 B	51.87 GB	0ms	
	Data obtained	3 min 11 sec	3 min 11 sec	3 min 11 sec	3 min 11 sec	3 min 11 sec	3 min 11 sec	

Your new node is now available for building with.

> Remember, your slave node will need the appropriate tools installed for your build; for instance, Git, maven and so on. It's a good practice to automate this using a tool such as Ansible. This makes it quick and easy to spin up new slaves.

See also

For further details on the SSH plugin, you can see the documentation at `https://wiki.jenkins-ci.org/display/JENKINS/SSH+Slaves+plugin`.

Creating your first Jenkins job

The basic building blocks of Jenkins are jobs. A Jenkins job is a series of steps that normally check software out of a repository, run unit tests, and builds the artifact ready for deployment; however, they are versatile and can be used to perform almost any task you can think of.

> When we talk of an artifact in this context, we are referring to the deployable object your build job produces. This is commonly an object, such as an executable binary, library, or software package. An artifact is the object you wish to take from your build server, deploy, and execute on your environments to test.

At its core, you add steps to a Job and it can trigger another when it succeeds; generally speaking, these are command-line jobs but as with many Jenkins items, this capability can be extended with the use of Jenkins plugins.

For this example job, I am going to build my blog. This is based on the excellent Hugo static blog engine (`http://gohugo.io`). Hugo allows us to create a relatively straightforward job that will download the code from a Git repository and run a task that will build our end product; we're then going to use Jenkins to archive this ready for distribution.

Getting ready

For this recipe, you will need a Jenkins server. You should also have a blog ready to process within Hugo. You can find an example blog at `https://github.com/spf13/hugo/tree/master/examples/blog`.

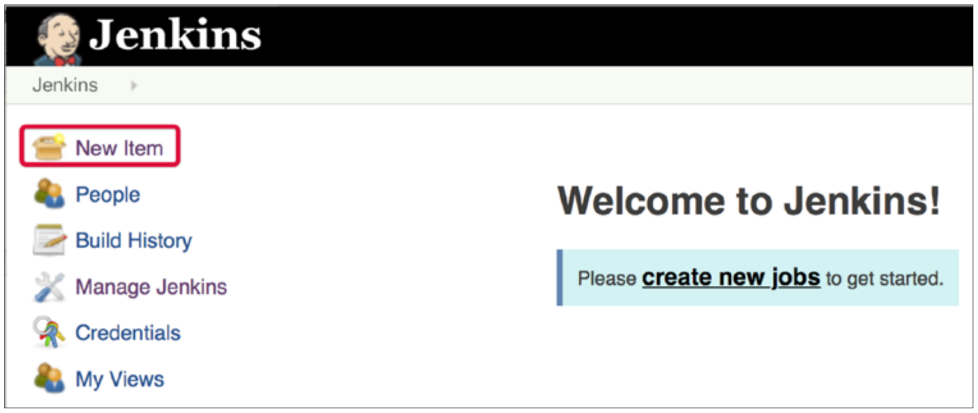
How to do it...

The following steps will show you how to install the Hugo blog engine and create a Jenkins job that will fetch an example site and build it:

1. Our first task it to install the Hugo blog engine; you can do this by issuing the following command on either the Jenkins server or Jenkins slave that will run this job:

```
$ wget https://github.com/spf13/hugo/releases/download/v0.14/
hugo_0.14_amd64.deb && dpkg -i hugo_0.14_amd64.deb
```

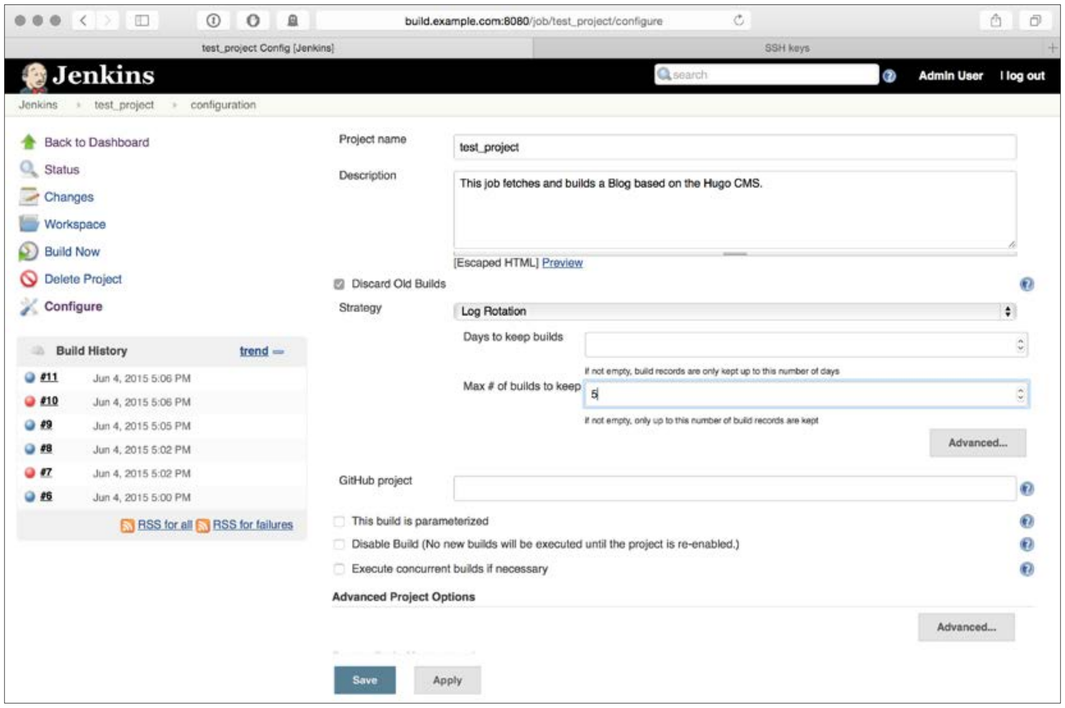
2. Next, we are going to create a new job. Login to your Jenkins server and click on **New Item** button:



3. In the next window, give your project a descriptive name and select **Freestyle project**.

You can find the differences between the different project types at `https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project#Buildingasoftwareproject-Settinguptheproject`.

4. In the next screen, you start to fill in the details that comprise your job. Take a look at the following screenshot:



build.example.com:8080/job/test_project/configure

Jenkins » test_project » configuration

Project name: test_project

Description: This job fetches and builds a Blog based on the Hugo CMS.

[✓] Discard Old Builds

Strategy: Log Rotation

Days to keep builds:

if not empty, build records are only kept up to this number of days

Max # of builds to keep: 5

if not empty, only up to this number of build records are kept

GitHub project:

This build is parameterized

Disable Build (No new builds will be executed until the project is re-enabled.)

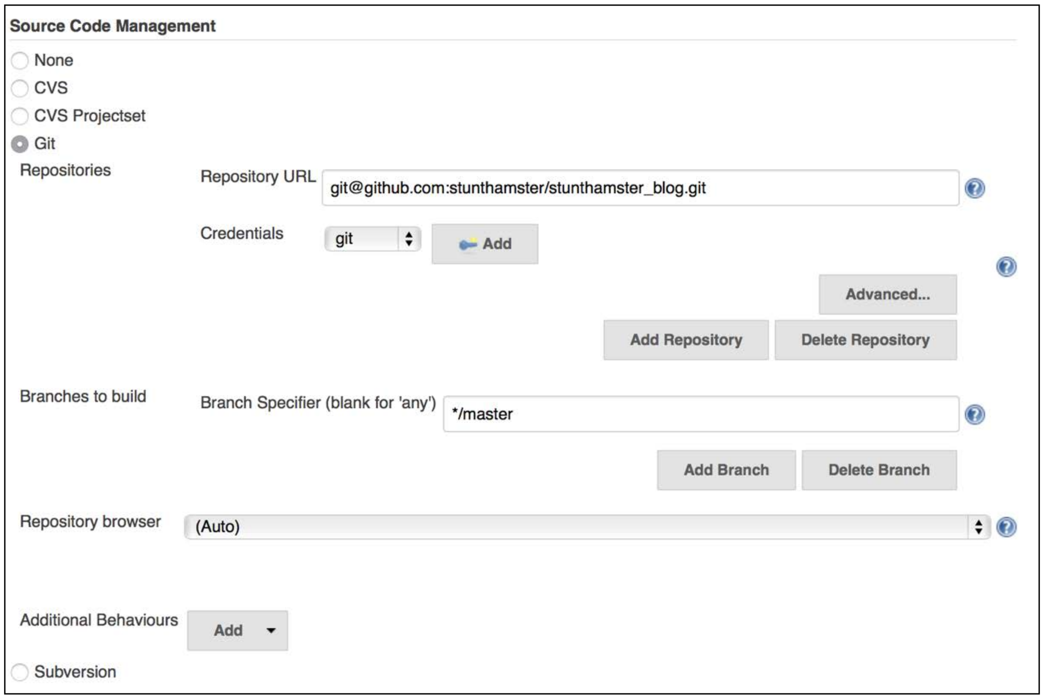
Execute concurrent builds if necessary

Advanced Project Options

Build History	
#11	Jun 4, 2015 5:06 PM
#10	Jun 4, 2015 5:06 PM
#9	Jun 4, 2015 5:05 PM
#8	Jun 4, 2015 5:02 PM
#7	Jun 4, 2015 5:02 PM
#6	Jun 4, 2015 5:00 PM

As you can see, I've added a description for the project and also selected the **Discard old builds** option. This is important; if left unconstrained, Jenkins can eat a stupendous amount of disk space. I've left it at five builds but you can also use the number of days instead. As with many Jenkins elements, there are plugins that allow you to tune this in more detail.

5. Next, we define our code repository options. As you can see in the succeeding screenshot, I've used Git to store the code for this particular project:



Source Code Management

- None
- CVS
- CVS Projectset
- [✓ Git]

Repositories — Repository URL: git@github.com:stunthamster/stunthamster_blog.git

Credentials: git

Branches to build — Branch Specifier (blank for 'any'): */master

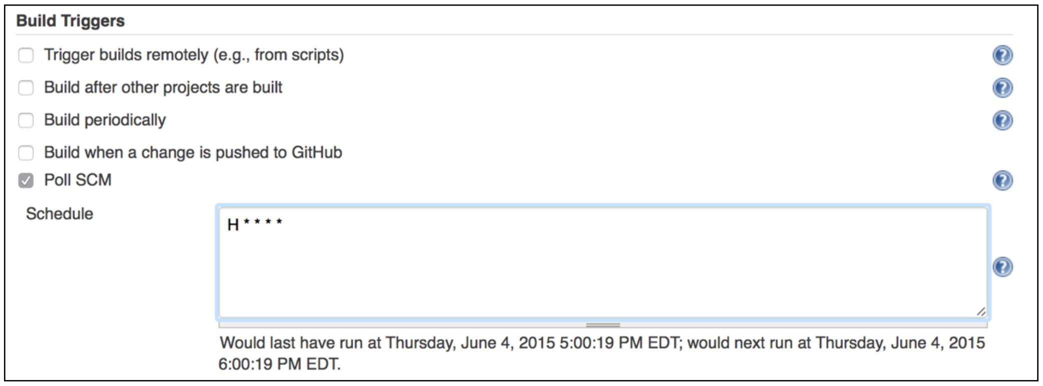
Repository browser: (Auto)

Additional Behaviours

- Subversion

This makes use of the Git credentials that we set up in the earlier recipe. If you choose to use a different type of repository, then amend your configuration to fit it.

6. Now we can configure how Jenkins will schedule the builds. You could leave this as a manual job but that rather flies in the face of a CI server; instead, I have set mine to periodically poll Git; have a look at the following screenshot:



Build Triggers

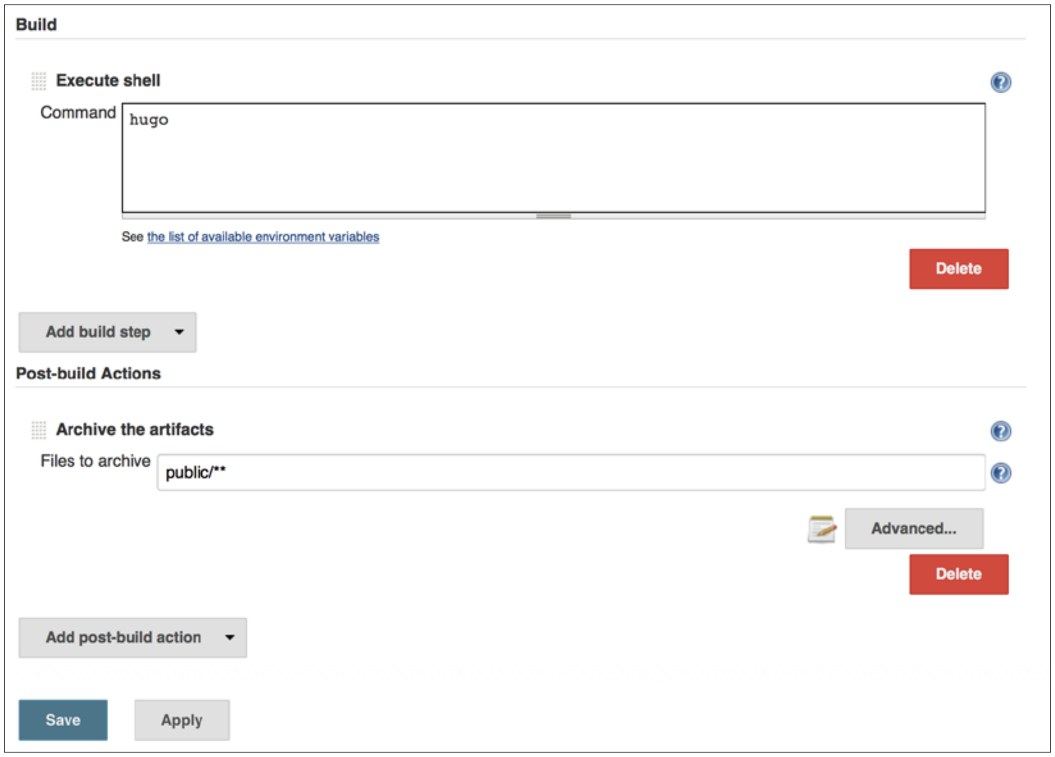
- Trigger builds remotely (e.g., from scripts)
- Build after other projects are built
- Build periodically
- Build when a change is pushed to GitHub
- [✓ Poll SCM]

Schedule: H * * * *

Would last have run at Thursday, June 4, 2015 5:00:19 PM EDT; would next run at Thursday, June 4, 2015 6:00:19 PM EDT.

The Poller uses a similar syntax to cron to set out the schedule, with a few notable differences. You can see one of these above; the H denotes a hash, which ensures that the job will run at a randomized time within that period; in this case, my schedule is to run at a random minute, every hour, day, month and year. This helps to keep the load light on the upstream server; perhaps not as important on a scalable public platform such as Github, but very important when you run your own server.

7. Finally, we setup both our build and post-build steps. The build steps are what command you to run once you have cloned your source code; have a look at the following screenshot:



```
hugo
```

Files to archive: public/**

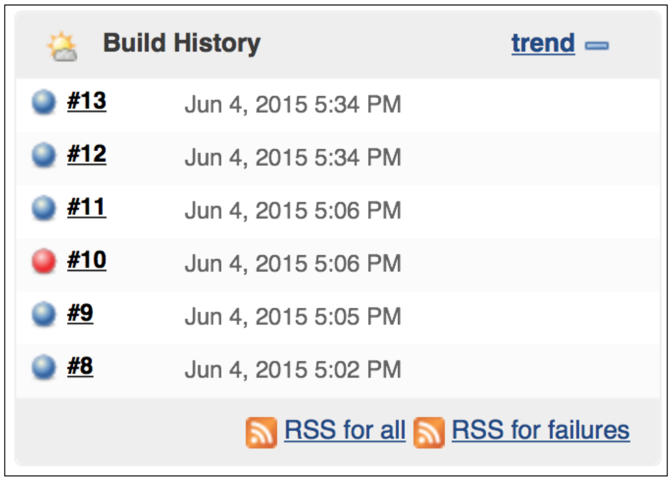
For my job, I am calling the `hugo` command. Hugo is a statically generated CMS. It takes the content, processes it, and creates the HTML, CSS and also the images that make up the site. This has a huge performance boost over CMS systems such as WordPress, as content is not generated on the fly; it also makes it very easy to use with a **Content Distribution Network** (**CDN**).

After we execute our build, we add a post-build action to archive the artifacts. This will produce a zip file containing the selected items from our build. In this case, I'm zipping up the public files that comprise the deployable part of the blog. Each time a build is successful, you will be able to open it in Jenkins and find the artifact available for download from within it.

Don't forget, you're not limited to one-step for either the build or post-build; you can have as many as you like and they can be ordered using drag and drop.

At this point, hit the save button and we're ready to run our job.

8. Back at the job screen, hit the **Build now button**. This will trigger an immediate build. To the left of the screen, you should see the **Build History**; it looks similar to the following screenshot:



●	#13	Jun 4, 2015 5:34 PM
●	#12	Jun 4, 2015 5:34 PM
●	#11	Jun 4, 2015 5:06 PM
🔴	#10	Jun 4, 2015 5:06 PM
●	#9	Jun 4, 2015 5:05 PM
●	#8	Jun 4, 2015 5:02 PM

As long as the icon is blue, the build is a success. Clicking on the icon will take you to results of that particular build and its here that you can see the console output from the job, and most importantly, the build artifact. If the icon is red, it means that something didn't quite go right in your build and you need to examine the console output to derive what might have occurred.

See also

You can find detailed documentation around Jenkins jobs at `https://wiki.jenkins-ci.org/display/JENKINS/Building+a+software+project`.

Building Docker containers using Jenkins

One of the important elements of creating an automated build is to ensure that the environment is consistent; unless you ensure consistency, you may find that integration tests pass when they should fail. Docker is perfect for this. By building your environment within a Docker container, you can ensure that your environment is the same from the initial build, right through to the production deployment and you can dispose of this environment after the build.

By utilizing Docker, you are also preparing for continuous deployment. A successful build within a container can be promoted to the next environment without needing to be amended. If it passes that environment, it can be passed onto the next and so on. If you trust your automated testing sufficiently, you can even push your container into service in production.

You can find more about continuous deployment patterns in the seminal book on the subject, *Continuous Delivery* by *Jez Humble*.

Getting ready

For this recipe, you will need a Jenkins server with Docker installed.

How to do it...

The following steps will show you how to install Docker and use it to create a deployable artifact using Jenkins:

1. Follow the instructions in *Chapter 6*, *Containerization with Docker* of recipe *Installing Docker*, to install Docker on your Jenkins server.
2. For Jenkins to be able to use the `docker` commands during a build, we need to give the Jenkins user the correct permission. We need to create a `docker` group and add the Jenkins user to it. You can do that by issuing the following command on the Jenkins server:

   ```
   $ usermod -aG docker Jenkins
   ```

   If your Jenkins server is running it will not pick up these changes; restart it with the following command:

   ```
   $ service Jenkins restart
   ```

3. If you want your automated build to be able to be able to publish to the Docker registry, then you need to sign in. Use the following commands to enable your Jenkins user to access your registry:

   ```
   $ sudo su sudo Jenkins
   $ docker login
   ```

   Enter your details when prompted; they will then be stored for use within Jenkins.

4. Now that we have finished our setup steps, we are ready to build our container. We are going to take the Hugo blog from the previous recipe and use a Docker container to host the built artifact. We will also add an Nginx HTTP server to serve the blog.

   First, we are going to move the contents of our blog into a new folder structure. Open a terminal session in the root of your blog project and issue the following command:

   ```
   $ mkdir -p files/hugo
   ```

   Issue the following command to move your blog content into your new directory structure:

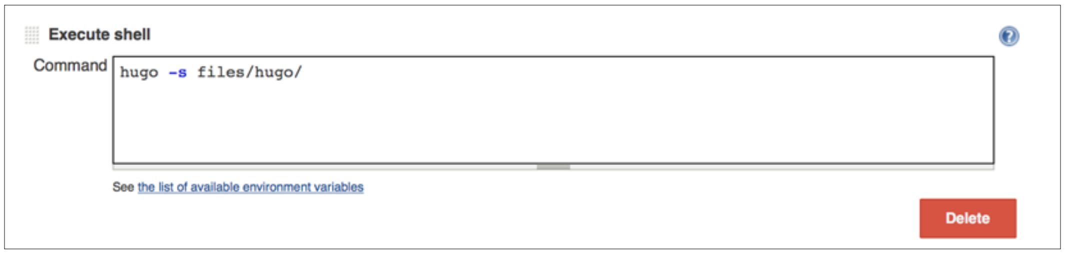
   ```
   $ mv * files/hugo
   ```

5. Next, we are going to create our `Dockerfile`. This `Dockerfile` will install Nginx and add the contents of our Hugo blog into the default location to be served by Nginx. Create a new file called `Dockerfile` in the root of your project and insert the following code:

   ```
   FROM ubuntu:14.04
   MAINTAINER <MAINTAINER DETAILS>
   RUN apt-get update && apt-get upgrade -y
   RUN apt-get install -y nginx
   ADD files/hugo/public /usr/share/nginx/html
   # Expose ports.
   EXPOSE 80
   CMD ["nginx", "-g", "daemon off;"]
   ```

   Once you are done editing, save the file and check your project into your Git repository and push to your remote.

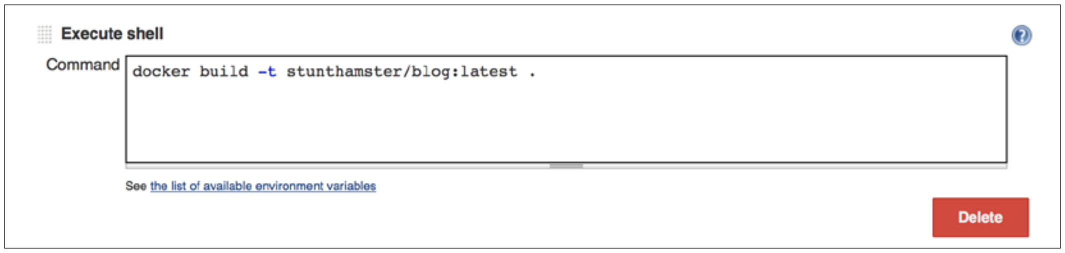
6. Now, we can edit our build to create a Docker image. Login to your Jenkins server and locate the job used to build your Hugo blog; click on the configure button.
7. The first item we need to edit is the first build step. Previously, we simply triggered the `hugo` command in the root of the work directory; since we have moved the files into a sub-directory, this will no longer work. Instead, edit it to resemble the following:



```
hugo -s files/hugo/
```

The `-s` switch enables you to give `hugo` a path to its input file and we can then supply the new directory structure.

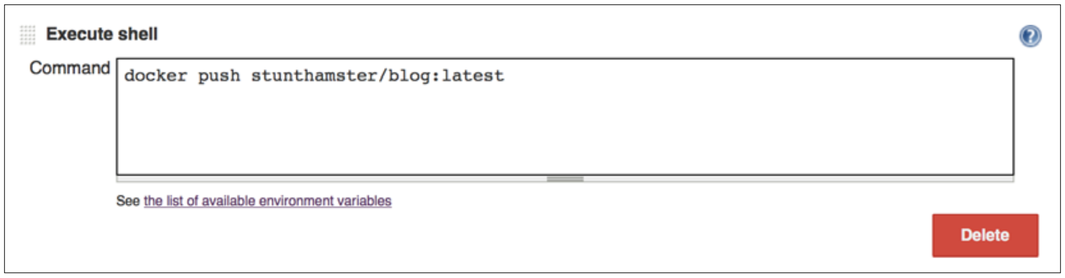
8. Now, we need to add a new build step. Underneath the first step, click the button marked **Add build step**, and select **Execute Shell** Script as the action. This will add a new textbox underneath the original build step; add the following content:



```
docker build -t stunthamster/blog:latest .
```

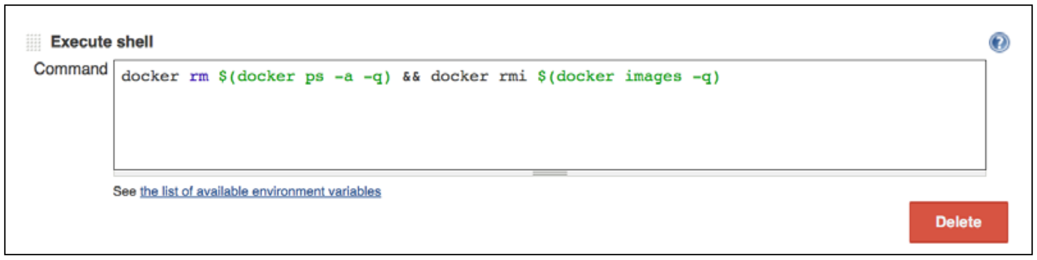
As you can see, this is using the Docker command to perform the build for us and add the tag of **Latest** to it.

9. Now, we shall add a step to push the built image to the Docker registry. Again, click on the **Add build step** button and select **Execute Shell Script**. This time, add the following command:



```
docker push stunthamster/blog:latest
```

10. This final step is optional, but recommended. As with all Docker hosts, repeated builds create containers and these can add up over time. You can alleviate this by deleting old containers and images using the following command:

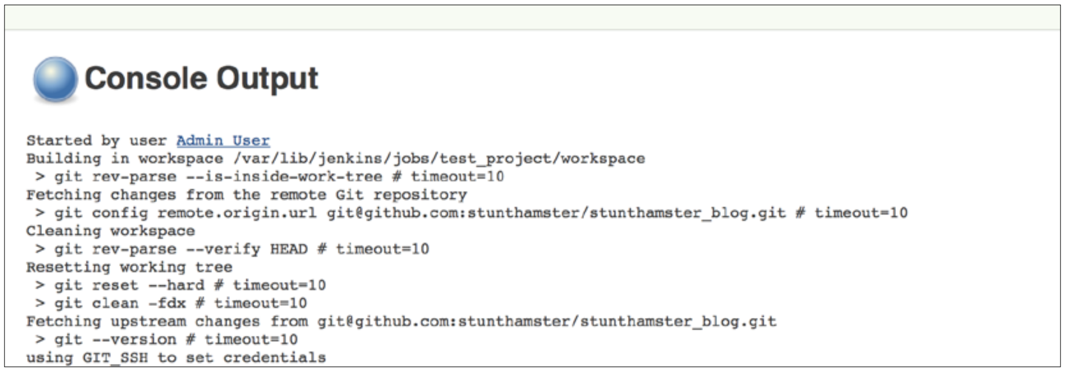


```
docker rm $(docker ps -a -q) && docker rmi $(docker images -q)
```

This will remove all Docker images and containers after each build and will stop your disk space from disappearing into the ether.

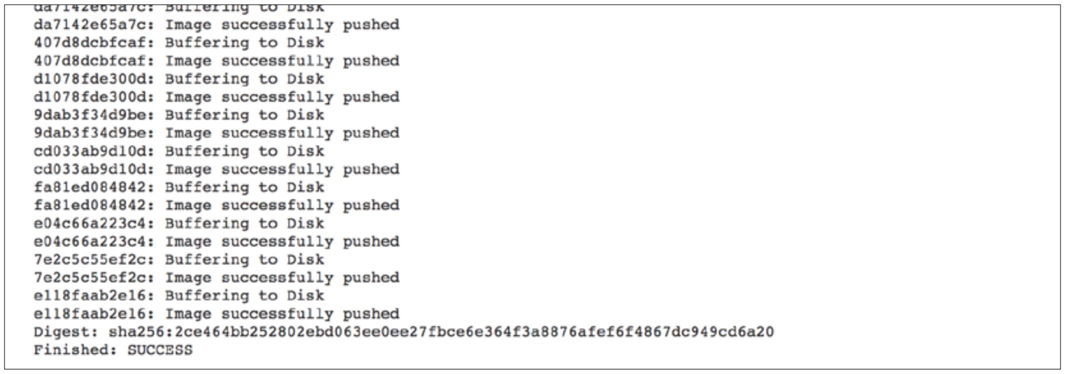
This can cause your build to go slower as Docker will be unable to use a build cache; depending on the size of your image, locality of Docker registry, and speed of Internet connection, this may not matter. However, if you want the quickest possible build, omit this step and police your Docker storage using other means.

11. We're now ready to build. Save your job and hit the build button. Once it's complete, examine the build-log by locating and clicking on the last build on the build history panel on the left-hand side. Once you have opened the build, click on the button on the left-hand marked **Console Output**; it should look something like this:



```
Started by user Admin User
Building in workspace /var/lib/jenkins/jobs/test_project/workspace
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@github.com:stunthamster/stunthamster_blog.git # timeout=10
Cleaning workspace
 > git rev-parse --verify HEAD # timeout=10
Resetting working tree
 > git reset --hard # timeout=10
 > git clean -fdx # timeout=10
Fetching upstream changes from git@github.com:stunthamster/stunthamster_blog.git
 > git --version # timeout=10
using GIT_SSH to set credentials
```

If all goes well when you scroll down, it should look something like this:



```
da7142e65a7c: Buffering to Disk
da7142e65a7c: Image successfully pushed
407d8dcbfcaf: Buffering to Disk
407d8dcbfcaf: Image successfully pushed
d1078fde300d: Buffering to Disk
d1078fde300d: Image successfully pushed
9dab3f34d9be: Buffering to Disk
9dab3f34d9be: Image successfully pushed
cd033ab9d10d: Buffering to Disk
cd033ab9d10d: Image successfully pushed
fa81ed084842: Buffering to Disk
fa81ed084842: Image successfully pushed
e04c66a223c4: Buffering to Disk
e04c66a223c4: Image successfully pushed
7e2c5c55ef2c: Buffering to Disk
7e2c5c55ef2c: Image successfully pushed
e118faab2e16: Buffering to Disk
e118faab2e16: Image successfully pushed
Digest: sha256:2ce464bb252802ebd063ee0ee27fbce6e364f3a8876afef6f4867dc949cd6a20
Finished: SUCCESS
```

Notice the message is telling you **Finshed: SUCCESS**; if it says something differently, then you have a problem; otherwise, you should now have a new Docker image ready to use. You can double check by examining your Docker registry; you should find a new build within it.

Of course, it is understood that this is the tip of the iceberg. Using this recipe as a base, you can easily extend this job to build your software, package it into a Docker image once it has passed its unit tests and then automatically promote the image to an integration environment for testing. Be creative and with a small amount of work, you can save yourself a vast amount of manual testing.

Deploying a Java application to Tomcat with zero downtime using Ansible

Docker is not the only way to perform deployments using Jenkins, and indeed, for many organizations their container efforts are still in the nascent stages. For many, Java and Tomcat are still the mainstay of most platforms and will continue to be so for some considerable time.

Tomcat is now in its eighth version and is still one of the most common Java containers in use, both due to it's Open Source heritage and battle tested stability. Over time, it has also learned some interesting new tricks, with a particularly innovative one being a function called parallel deployment. Parallel deployment allows you to deploy a new version of an application alongside an existing one and Tomcat will then ensure that any existing connections to the application will be satisfied, while new connections will be passed to the new version. From a user perspective, there is no downtime and they simply flip from one version to another the next time they connect to the application.

Using a combination of Ansible, Tomcat, and Parallel deployment, you can promote builds seamlessly, and even better, if you deploy a bad build, you can roll it back relatively easily; all without causing downtime to your platform.

> It goes without saying that this is great for your app but you need to design your architecture around this capability. It is no good using parallel deployments, if your app is dependent on a manual database migration or if you have a heavy dependency on state. A state in particular will not be passed from one application to another.

Getting ready

For this recipe, you are going to need a Jenkins server to host the build and a tomcat server to act as the app server plus a repository to hold the example code. Ensure that your Jenkins server also has the Ansible installed; you can see instructions on how to do this in *Chapter 5, Automation with Ansible*, in recipe, *Installing Ansible*.

How to do it...

The following steps will demonstrate how to use Jenkins, Ansible and Tomcat to deploy a very simple test application:

1. To demonstrate how parallel deployment works, we are going to create a very simple web app and package it in a `war` file. Use the following command to create a new directory structure:

   ```
   $ mkdir -p testapp/WEB-INF
   ```

2. Next, edit a new file under `testapp/WEB-INF` called `web.xml` and insert the following content:

   ```
   <web-app></web-app>
   ```

   This is a very simple `web app` configuration file.

   Save this file and create another file called `index.jsp` and insert the following code:

   ```
   <html>
      <head>
        <meta http-equiv="refresh" content="1">
      </head>
      <body>
        <H1>Test App</H1>
        <%= date = new java.util.Date() %>
        <p>Greetings, this is the first version of our test app. The
   time is currently: <%= date %></p>
       </body>
   </html>
   ```

This is a remarkably simple application, whose only role in life is to print out a pithy greeting, the date and time, use the HTML META tag to force the browser to refresh the page; however, it's perfectly suited to demonstrate the power of parallel deployment.

3. Next, we are going to create a new Ansible playbook to perform our deployment. Create a new directory to hold our playbook using the following commands:

```
$ mkdir -p appdeploy/inventory
$ mkdir -p appdeploy/group_vars/appdeploy
```

This creates a basic structure to house our Ansible code, including a place to hold our inventory and variables.

If you have an existing Ansible setup, you would like to use this code. In this recipe, we are going to add a new playbook and inventory item, both of which can happily live inside an existing project.

4. Next, we're going to create our Ansible inventory and create a new file called `appdeploy/inventory/appinventory` file and insert the following code:

```
[appdeploy]
<< tomcatserver >>
```

Where `<< tomcatserver >>` is the name/ip address of your tomcat server.

5. Now that we have our inventory, we are ready to create our playbook. Create a new file called `appdeploy.yml` under the `appdeploy` folder and insert the following content:

```
- hosts: appdeploy
  gather_facts: false
  sudo: true
  tasks:
    - name: Deploy App
      copy: src=/var/lib/jenkins/jobs/workspace/tomcat_test /
testapp.war dest="/usr/local/apache-tomcat-8.0.23/webapps/
testapp##{{ buildnum }}.war" owner=tomcat group=tomcat
```

This is a simple playbook. First, we declare that this will only run against the servers that are tagged as `appdeploy` hosts. We are also declaring that this playbook does not need to gather facts. Not gathering facts can be a good way to speed up a playbook and is an excellent practice when you know that your Ansible code does not make use of them, as it skips a relatively expensive fact of gathering task. As usual, this is a trivial time saver against one host but when you scale out to hundreds of hosts, it can make a difference.

> You may need to amend the paths in the preceding example, especially if you are using a slightly different version of Tomcat.

Next, we declare the tasks that we are going to carry out. In our case, we are only performing a single task, copying a war file to the remote host. Notice the `{{ buildnum }}` variable; this is going to be our link between Ansible and Jenkins and it will be explained later in the recipe.

6. In the `appdeploy/group_vars/appdeploy` directory, create a new file called `main.yml` and insert the following code:

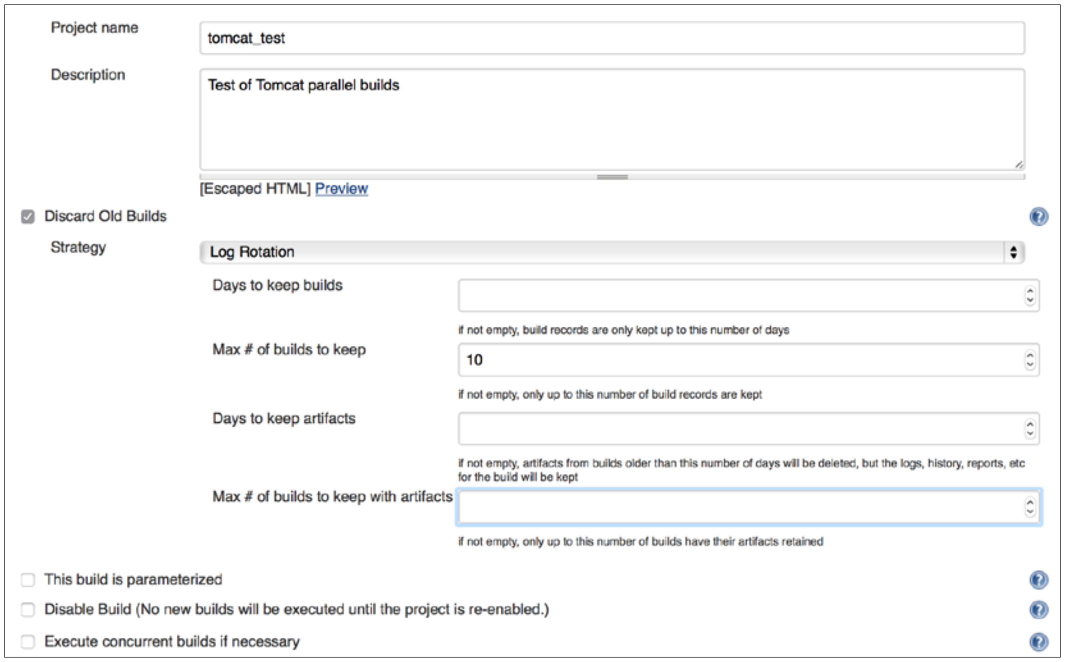
```
# Intentionally left blank
```

This is a simple kludge to ensure that the directory is added if you are using a Git repository. We are going to fill the content of this file by dynamically using Jenkins but Git has a nasty habit of not adding empty directory structures. This ensures that it will be included when we check it in and push it to our remote.

> A `.gitignore` file also suffices well for this purpose.

Once you have finished editing your files, create, check in, and push your code to your repository.

7. Now, we can create our new Jenkins job. Log on to your Jenkins server and create a new item called `tomcat_test` and ensure that it's set as a freestyle project. Once created, we can set the basic job options. We want to ensure that our job doesn't fill the disk of our Jenkins server, which is a real and immediate issue with builds that can generate large artifacts; ensure that your build matches the options listed in the following screenshot:



Project name: tomcat_test

Description: Test of Tomcat parallel builds

[Escaped HTML] Preview

☑ Discard Old Builds

Strategy: Log Rotation

Days to keep builds:

if not empty, build records are only kept up to this number of days

Max # of builds to keep: 10

if not empty, only up to this number of build records are kept

Days to keep artifacts:

if not empty, artifacts from builds older than this number of days will be deleted, but the logs, history, reports, etc for the build will be kept

Max # of builds to keep with artifacts:

if not empty, only up to this number of builds have their artifacts retained

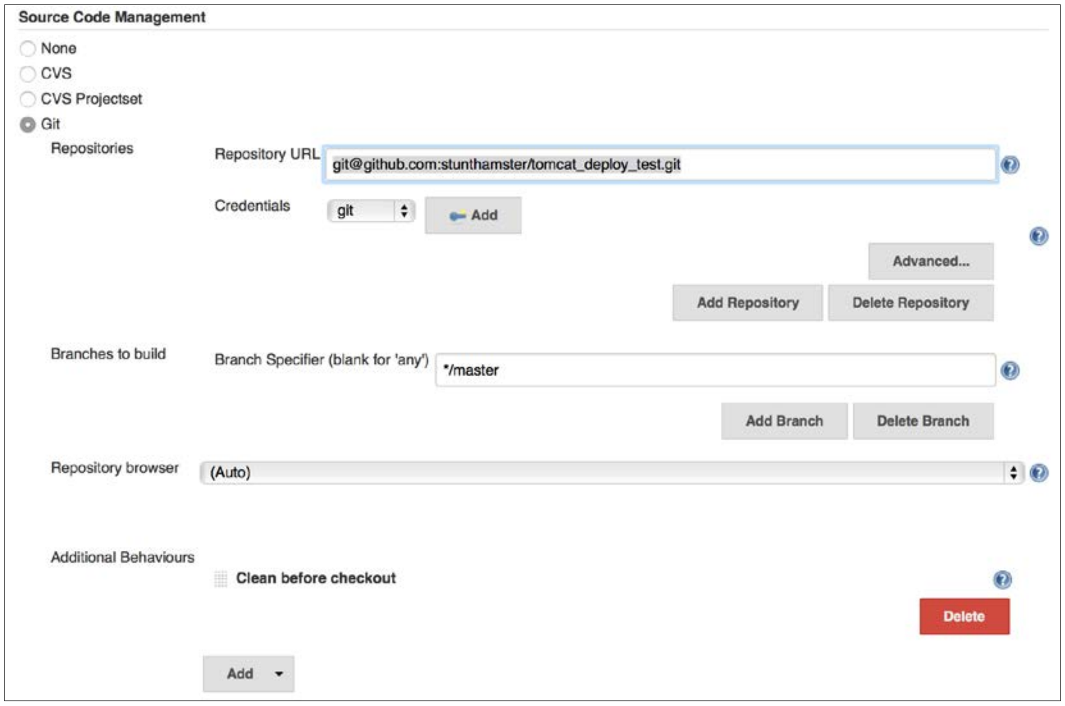
☐ This build is parameterized

☐ Disable Build (No new builds will be executed until the project is re-enabled.)

☐ Execute concurrent builds if necessary

As you can see, I'm limiting the job to only keeping 10 builds at a time; you should edit this to suit your particular needs.

8. Next, we're going to configure our source code repository options. These will be different depending on your Git hosting options and repository name; you can use the following screenshot as a guide to configure your repository options:



Source Code Management

None
CVS
CVS Projectset
[✓ Git]

Repositories — Repository URL: git@github.com:stunthamster/tomcat_deploy_test.git

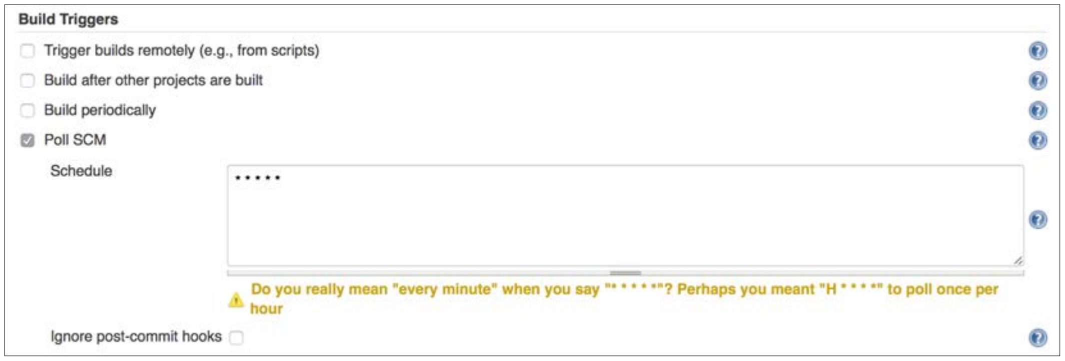
Credentials: git

Branches to build — Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours: Clean before checkout

Now we can set our build triggers. Since we are creating a CI process, we should poll our code repository regularly to look for changes, the idea being, that when a developer checks in code, it is immediately picked up and a build is attempted. Have a look at the following screenshot:



Build Triggers

Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
[✓ Poll SCM]

Schedule: * * * * *

Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour

Ignore post-commit hooks

As you can see, I'm polling the SCM (Source Control Manager) every minute. You can amend this to fit the capabilities/spec of your repository, especially if you have many builds. Having a periodic polling can completely crush an SCM with a lowly spec, so be cautious with this option and be ready to scale when your job list gets relatively big.

> As with all things Jenkins, there are many plugins that allow you to control the polling behavior; it is worth browsing the plugin directory to see if it can help on this front. Alternatively, you may find that your SCM supports hooks that allow you to push build events rather than polling for them. This is ideal as the SCM is only being called when there is actually something to do, rather than being battered with needless traffic.

9. Now we have our schedule in place, it's time to get on and construct our build steps. First, we're going to use the jar command to create our Java archive. Add a new execute shell build step and ensure that it has the following command:

   ```
   mv chapter7/testapp/* . && jar cf testapp.war WEB-INF/index.jsp
   ```

   This is a simple command that takes the checked out code and moves it into the working directory and uses the jar command to create the archive.

10. Next, we need to create the variable that will inform Ansible, which builds the application we are pushing. Create a new execute shell build step and add the following content:

    ```
    echo "buildnum: $BUILD_NUMBER" > chapter7/appdeploy/group_vars/
    appdeploy/main.yml
    ```

11. This command makes use of one of the environment variables that Jenkins sets as part of its build. These environment variables are astonishingly powerful, as they allow you to act on the output of your build.

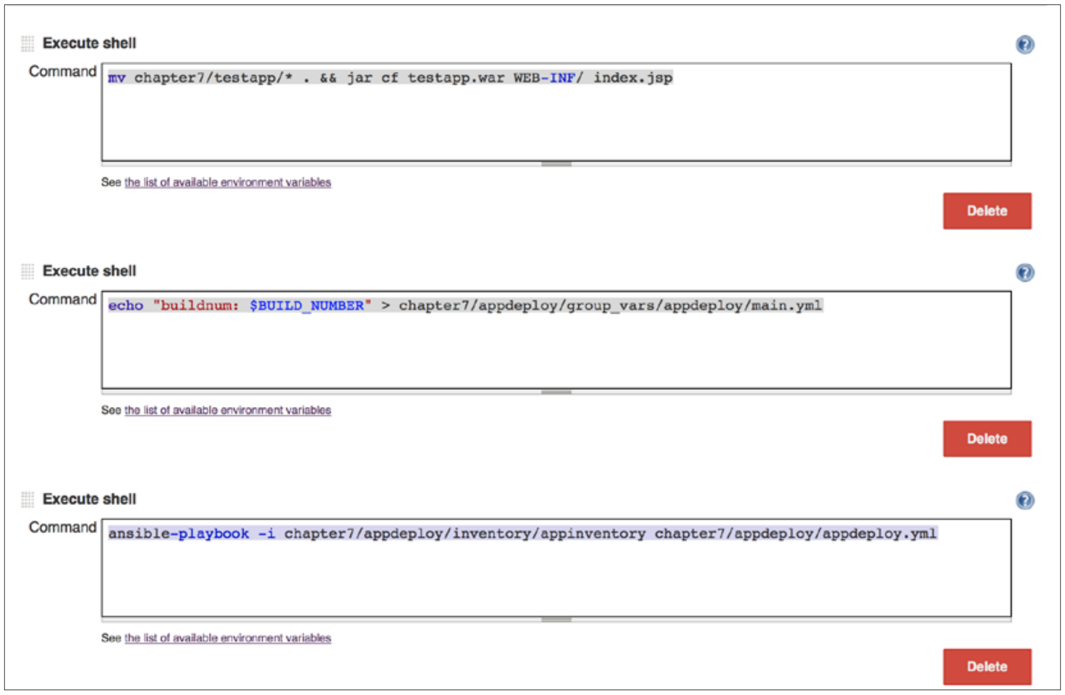
> You can find the list of available variables at your Jenkins server at the `/env-vars.html/` URL; you can also act on other environment variables that you set yourself.

12. Finally, we can trigger our Ansible build. Add another execute shell build step and add the following command:

    ```
    ansible-playbook -i chapter7/appdeploy/inventory/appinventory
    chapter7/appdeploy/appdeploy.yml
    ```

    This invokes Ansible to use your inventory and playbook.

If all goes well, your build steps should resemble the following screenshot:


Execute shell
Command
See the list of available environment variables
Delete

```
mv chapter7/testapp/* . && jar cf testapp.war WEB-INF/ index.jsp
```

Execute shell
Command
See the list of available environment variables
Delete

```
echo "buildnum: $BUILD_NUMBER" > chapter7/appdeploy/group_vars/appdeploy/main.yml
```

Execute shell
Command
See the list of available environment variables
Delete

```
ansible-playbook -i chapter7/appdeploy/inventory/appinventory chapter7/appdeploy/appdeploy.yml
```

Once you are happy that your job looks correct, save it and return to the main Jenkins screen.

13. Before we can run our project, we need to make some changes to our servers. As our Ansible code is being run as the Jenkins user we need to ensure that it has access to the target server and it has a `sudo` access on each server. If you are using Ansible to control the tomcat servers, then you can add a Jenkins user and key via Ansible; see *Chapter 5*, *Automation with Ansible*, for an example. You will also need to add `sudo` access; this can be done by adding the following task into the Role that manages your user:

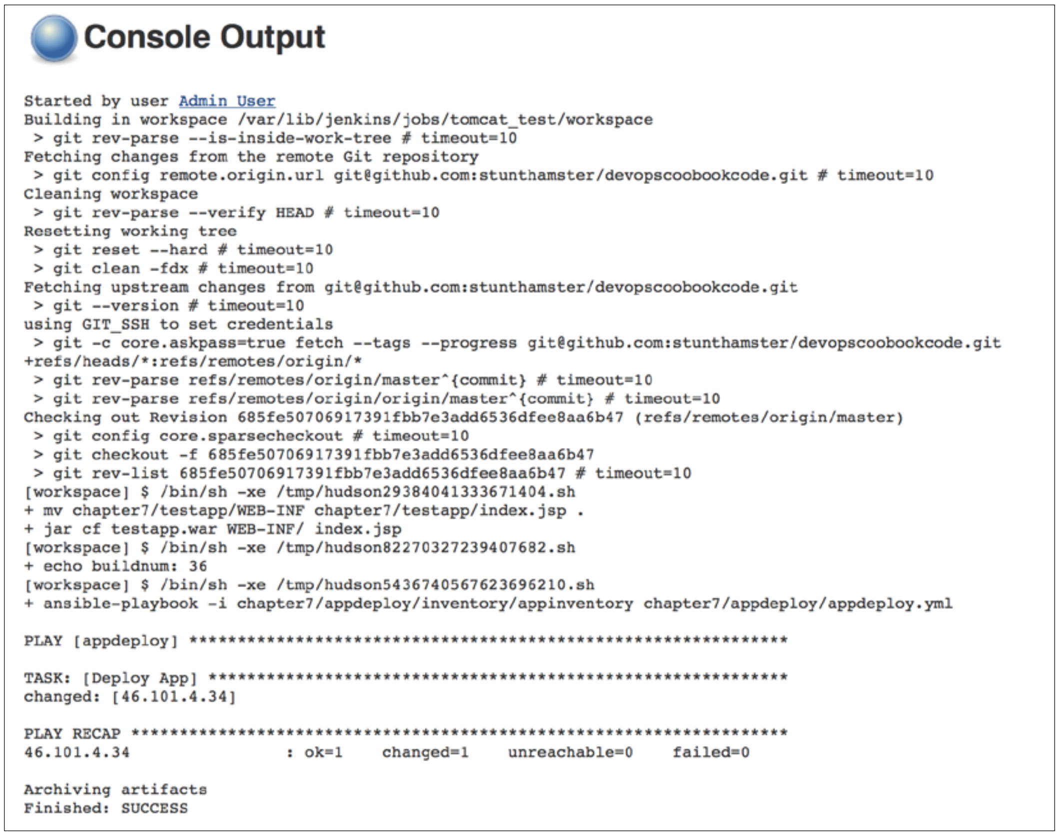
```
- name: Add Jenkins Sudo access

  lineinfile: dest=/etc/sudoers state=present line='jenkins
ALL=(ALL) NOPASSWD:ALL' validate='visudo -cf %s'
```

This code uses the Ansible `lineinfile` module to insert a new rule into the `sudoers` file; it will also use the `visudo` command to double check that the new rule is valid.

Regardless of how you achieve it, ensure that your Jenkins user has both access to your target servers and a `sudo` access.

> It is understood, that when using Jenkins in this manner it's vitally important to ensure that the Jenkins server is secure, especially if you use this technique in production deployments. I tend to air gap the development and production servers in this scenario and ensure that the production Jenkins is heavily monitored and locked down.

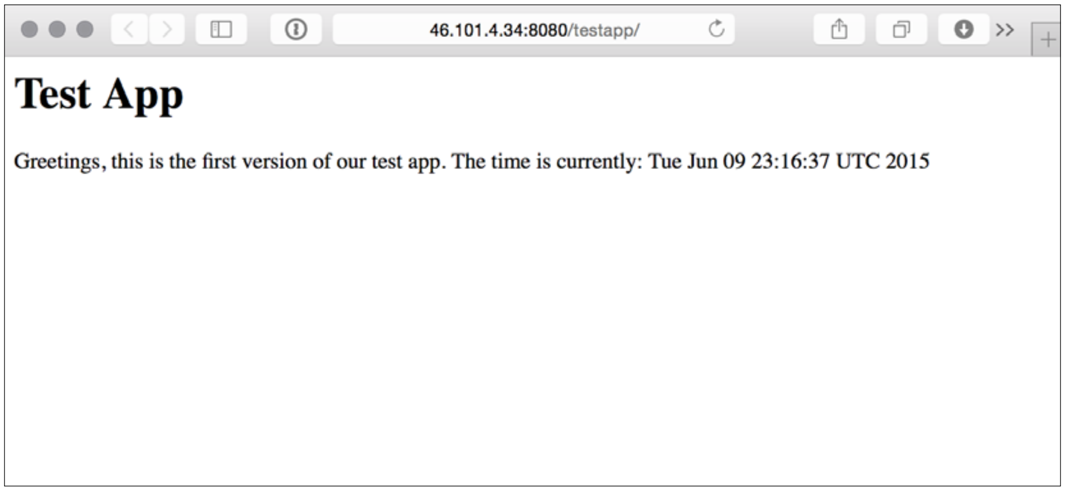
14. We're now ready to run our Jenkins build. Hit the Build now button and wait a few seconds. If all goes well, you should have a successful build and console output that resembles the following:



Console Output

```
Started by user Admin User
Building in workspace /var/lib/jenkins/jobs/tomcat_test/workspace
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@github.com:stunthamster/devopscoobookcode.git # timeout=10
Cleaning workspace
 > git rev-parse --verify HEAD # timeout=10
Resetting working tree
 > git reset --hard # timeout=10
 > git clean -fdx # timeout=10
Fetching upstream changes from git@github.com:stunthamster/devopscoobookcode.git
 > git --version # timeout=10
using GIT_SSH to set credentials
 > git -c core.askpass=true fetch --tags --progress git@github.com:stunthamster/devopscoobookcode.git
+refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 685fe50706917391fbb7e3add6536dfee8aa6b47 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 685fe50706917391fbb7e3add6536dfee8aa6b47
 > git rev-list 685fe50706917391fbb7e3add6536dfee8aa6b47 # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson2938404133367l404.sh
+ mv chapter7/testapp/WEB-INF chapter7/testapp/index.jsp .
+ jar cf testapp.war WEB-INF/ index.jsp
[workspace] $ /bin/sh -xe /tmp/hudson82270327239407682.sh
+ echo buildnum: 36
[workspace] $ /bin/sh -xe /tmp/hudson5436740567623696210.sh
+ ansible-playbook -i chapter7/appdeploy/inventory/appinventory chapter7/appdeploy/appdeploy.yml

PLAY [appdeploy] ************************************************************

TASK: [Deploy App] **********************************************************
changed: [46.101.4.34]

PLAY RECAP ******************************************************************
46.101.4.34                : ok=1    changed=1    unreachable=0    failed=0

Archiving artifacts
Finished: SUCCESS
```

Open your browser, and point it at your test application. You should see something similar to the following screenshot:



Test App

Greetings, this is the first version of our test app. The time is currently: Tue Jun 09 23:16:37 UTC 2015

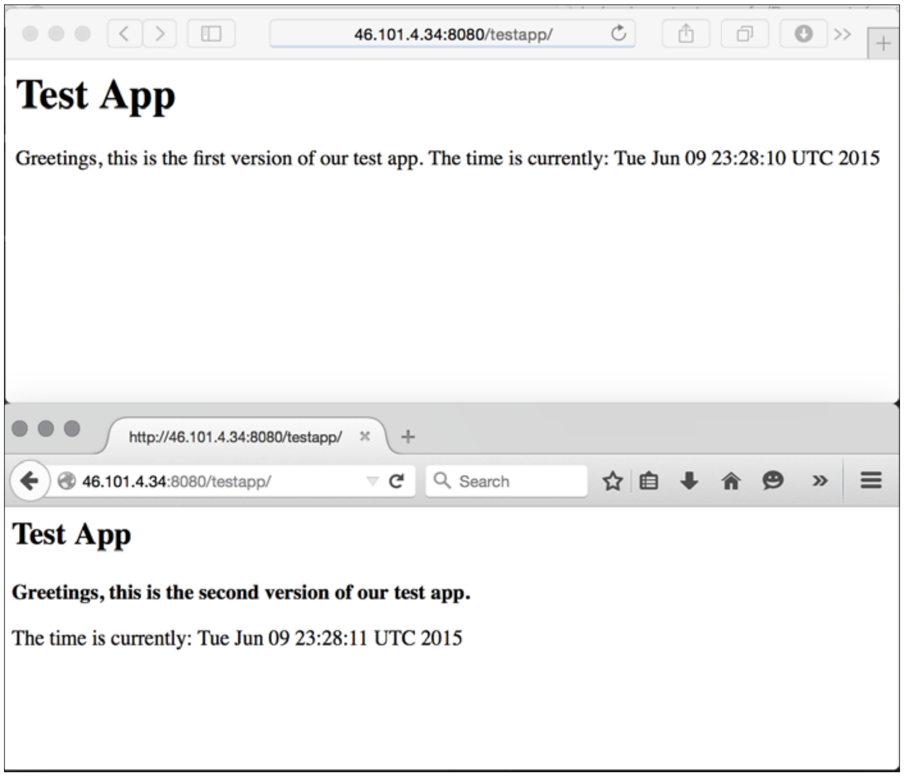
If you see this, then excellent; your automated build has succeeded in taking a code from a developer and deploying it to a target machine without any manual interactions.

15. Now that we have deployed the first version of our code, it's time to make an amendment and deploy the second version. Keep your browser window open in the background and open up the code in `testapp/index.jsp` and edit it to resemble the following code example:

```
<html>
   <head>
     <meta http-equiv="refresh" content="1">
   </head>
   <body>
      <H2>Test App</H2>
      <%= date = new java.util.Date() %>
      <strong>Greetings, this is the second version of our test
app.</strong>
<p>The time is currently: <%= date %></p>
    </body>
</html>
```

As you can see, this a slight tweak to our original code. Adding the maximum amount of style that a DevOps engineer generally decorates their apps with. Once you have made the tweak, save the code and check-in to your repository and push to your remote.

16. In addition to your existing browser window, open a new session and point it at `server>:8080/testapp`. You should see the new content showing, with the original content in your original browser window. Here's a general idea of what you should see:


46.101.4.34:8080/testapp/
http://46.101.4.34:8080/testapp/
46.101.4.34:8080/testapp/
Search

Test App

Greetings, this is the first version of our test app. The time is currently: Tue Jun 09 23:28:10 UTC 2015

Test App

Greetings, this is the second version of our test app.

The time is currently: Tue Jun 09 23:28:11 UTC 2015

As you can see, two versions of the same app are running at the same time and will continue to do so as long as we have our browser window open with the old application. As soon as we close the window with the old application, it will be unloaded and will not be used again.

It's worth going over the details of what this recipe is doing. At its most basic form, this recipe is using Jenkins to construct an artifact and packaging it into a WAR file. Once this file is created, we use Ansible to transfer the built file to our targeted tomcat server, into the webapps directory. The important detail is that it appends the build number onto the WAR file, prepended with `##` before the file extension.

When Tomcat sees a WAR file with `##` in front of its file extension, it works on the assumption that this is a versioned application and as with any app, it will attempt to explode and automatically deploy the code. When a newer version of the app is placed into the web apps directory, it explodes and deploys this as well, but it ensures that the existing app is also ran alongside it until all user sessions are expired. This is incredibly powerful and allows for true zero-downtime deployment using Tomcat; however, there are several issues:

- Old applications are not removed and will stay on the server until you house keep them.
- When running two apps side by side, you need to ensure that you have enough memory to service both, even if it's only for a short time.
- This only works with numeric build numbers; any other characters are not supported.
- As noted in the stat of this recipe, your app needs to be able to cope with an in service upgrade.

See also

You can find more details around parallel deployment at `https://tomcat.apache.org/tomcat-7.0-doc/config/context.html#Parallel_deployment`.

8
Metric Collection with InfluxDB

In this chapter, we are going to cover the following topics:

- Installing InfluxDB
- Creating a new InfluxDB database
- Logging events with the InfluxDB REST API
- Gathering host statistics with Telegraf
- Exploring data with the InfluxDB data explorer
- Installing Grafana
- Creating dashboards with Grafana

Introduction

The importance of monitoring cannot be overstated and it is seen as one of the core elements of a DevOps approach to a system. Monitoring takes many shapes; however, it's common to stop once you've added state monitoring, such as *Is the database up?* or *Is the site responding?.* These are indeed critical things to monitor, but they don't cover important questions such as *How fast am I responding to requests?* And *How many things have I done?*. This is where a time series database can be a valuable addition to the monitoring and information stack that you have to hand not handle.

A time series database is a storage technique that is designed to be fast at both storing and retrieving points of data. For instance, you can measure CPU usage in percentiles from one second to the next or alternatively measure the number of times a particular service call has been made. Once you have a different series, you can start to use this data for correlation. In the above example, we can overlay the CPU statistics across the service call information and start noting where the CPU spikes correlate with the service calls. This starts to form the basis for an **Application Performance Monitoring** (**APM**) solution; thus, allowing you to drill into the detail of where your platform is spending time and where improvements can be made.

A time series databases are becoming increasingly popular as people are realizing the value of the data their platform produces, and there are several dedicated open source TSD providers (such as InfluxDB and Whisper), and the industry heavy-weights, such as Microsoft and Oracle, are also enhancing their time series offerings.

For this chapter, we are going to use InfluxDB. InfluxDB is relatively a newcomer but it has many interesting features. That being said, it has yet to hit a 1.0 release and is a rapidly evolving, product especially with items such as clustering. However, InfluxDB is fast, easy to use, and extremely easy to deploy, and is being used by an increasing set of companies as their preferred time series database.

Installing InfluxDB

Installing InfluxDB is straightforward and it has no external dependencies; it also has packages for many operating systems. This recipe will show you how to install InfluxDB on an Ubuntu 14.04 server. The following steps are manual; however, you should consider automation using a tool such as Ansible (if you decide to use InfluxDB in production). You can find details of Ansible in *Chapter 5*, *Automation with Ansible*.

Getting ready

For this recipe, you need an Ubuntu 14.04 server.

How to do it...

The following steps will show you how to download, install, and configure InfluxDB:

1. On your server, issue the following commands:

   ```
   $ wget http://influxdb.s3.amazonaws.com/influxdb_0.9.3_amd64.deb
   $ sudo dpkg -i influxdb_0.9.3_amd64.deb
   ```

   This will fetch the InfluxDB package and install it.

2. Next, we can configure our InfluxDB server. You can find a ready-made configuration file in `/etc/opt/influxdb/influxdb.conf`, which is already populated with default `InfluxDB` values.

   There are a few configuration options that you should set specifically; first, locate the following entry in the configuration file:

   ```
   [data]
     dir = "/root/.influxdb/data"
   ```

   This should be amended to reflect your preferred storage location.

> It goes without saying, that you should aim to make your storage location both large and high-performance. Wherever possible, aim to use SSD storage. Time series data can lead to a staggering amount of IO, so always plan for your future requirements where possible.

3. You should also find the following configuration item:

   ```
   [meta]
     dir = "/root/.influxdb/meta"
     hostname = "localhost"
   ```

4. Change the value for the host name to reflect the host name of this particular node. Once you have made your changes, save the file and start InfluxDB using the following command:

   ```
   $ sudo /etc/init.d/influxdb start
   ```

5. You can test your new installation by issuing the following command:

   ```
   $ /opt/influxdb/influx -version
   ```

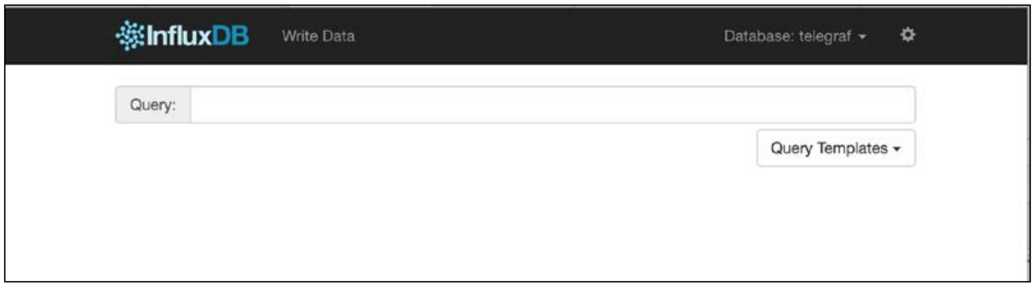
6. This should return a response similar to this:

   ```
   InfluxDB shell 0.9.4
   ```

7. You should be able to access the built-in graphical user-interface by opening the following URL in your browser:

   `http://localhost:8083`

This will give you a page similar to the following screenshot:



You will be ready to start with your InfluxDB instance one you see this screenshot.

See also

You can find the InfluxDB installation document at `https://influxdb.com/docs/v0.9/introduction/installation.html`.

Creating a new InfluxDB database

This recipe outlines how to create a new database in InfluxDB using the in-built GUI that is shipped as a part of the installation. This will create a new and empty database ready for data.

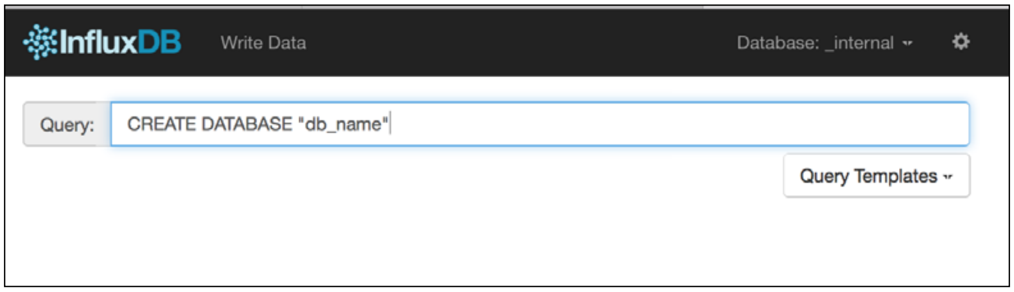
Getting ready

For this recipe, you need an Ubuntu 14.04 server with InfluxDB installed.

How to do it…

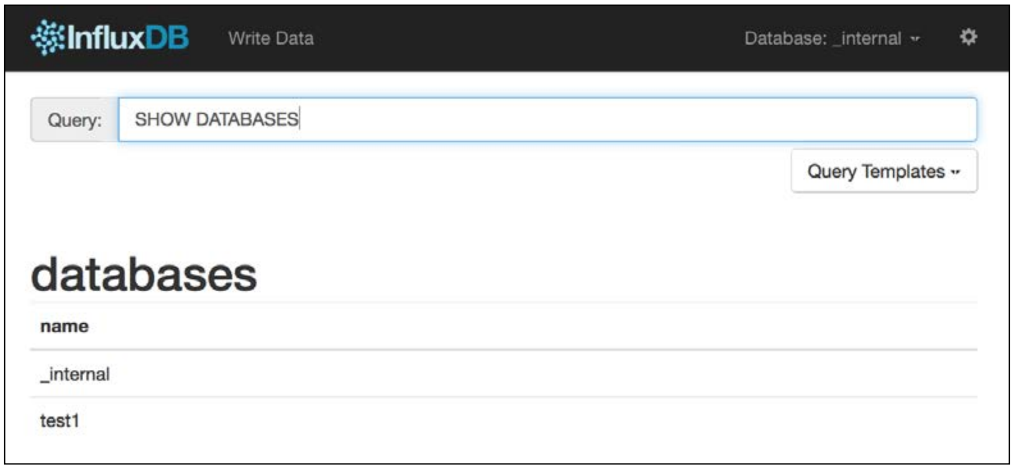
Let's create a new database in InfluxDB using the GUI:

1. Open the GUI in your browser by opening the following URL: `http://localhost:8083`.
2. You should be presented with a blank query field. At this point, you can either enter the query manually or select the **Query Templates** button on the bottom right-hand side. If you do so, you can select the option entitled **Create Database**. This will fill out the query for you as follows:



```
CREATE DATABASE "db_name"
```

3. To create your new database, simply give it a name (one word and no special characters) and hit `return`. If all goes well, you should see a message telling you that the database was successfully created. You can then use the query **SHOW DATABASES** in the query field to list your current database. It should look something like this:



```
SHOW DATABASES
```

databases

name
_internal
test1

4. We can test that our database is ready for use by entering some data. To manually enter a new piece of data, click on the **Write data** link. This will create a new pop-up window that allows you to write a new piece of data in InfluxDB line protocol. Insert the following code to create an example data:

```
numOrders,site="www.testsite.com",currency="GBP" value=15.50
```

> You can find more details of the InfluxDB line protocol at `https://influxdb.com/docs/v0.9/write_protocols/line.html`.

If all goes well, you will see a success message; your new database is now ready for data.

See also

You can see the official InfluxDB instructions at `https://influxdb.com/docs/v0.9/introduction/getting_started.html`.

Logging events with the InfluxDB REST API

InfluxDB can be used to log events and other statistics. An event can be anything from a user clicking on a button on your website to performing deployments. The latter is especially useful, as it means that you can have a single place to log any event that might occur on the platform.

This recipe will show you how to enter data using the REST API provided with InfluxDB. It allows you to create your own applications to enter extremely useful data. By leveraging the REST API, you will be able to use a wide spectrum of existing tools to enter data from Jenkins jobs to Ansible and beyond.

Getting ready

For this recipe, you need an InfluxDB instance.

How to do it…

The following steps will show you how to create a new InfluxDB database and populate it using the REST API:

1. First, we need to create a database in which we can log our events. We can use the REST API rather than using the GUI and you an use the following command to do this:

   ```
   $ curl -G http://localhost:8086/query --data-urlencode "q=CREATE
   DATABASE events"
   ```

   You should receive a JSON response that is similar to the following:

   ```
   {"results":[{}]}
   ```

   Although it's slightly abstract, this empty response indicates that the command was successful.

2. Now, we can add data to our new database. Let's start with a deployment and use the following `curl` command to input the data:

   ```
   $ curl -i -XPOST 'http://localhost:8086/write?db=events' --data-
   binary 'deployment, deployer=mduffy,app=test_app,version="1.1",env
   ironment="test" value="sucess"'
   ```

3. Let's take a look at this command. First, we have the details of the InfluxDB server we want to connect to; in this case, I've chosen to connect to my local instance on port `8086`. Next, let's decide on the action we wish to take and the database we wish to take it on; in this case, we wish to write a value and target the events `db`.
4. We move to the contents once we have set the target. First, we need to give the measurement a name; in this case, deployment. We then give this measurement a set of fields and values; in this case, the `deployer`, app, version, and environment. Finally, we give the measurement a value; in this case, success. This will create a new data point that we can query.
5. Now, since we have inserted a value, we can query it. Use the following command to do the same:

   ```
   $ curl -G 'http://localhost:8086/query?pretty=true' --data-
   urlencode "db=events" --data-urlencode "q=SELECT * FROM deployment
   WHERE deployer='mduffy'"
   ```

6. This should produce a JSON response that looks something like this:

   ```
   {
       "results": [
           {
               "series": [
                   {
                       "name": "deployment",
                       "columns": [
                           "time",
                           "app",
                           "deployer",
                           "environment",
                           "value",
                           "version"
                       ],
                       "values": [
                           [
                               "2015-09-
                                 16T21:17:54.799294876Z",
                               "test_app",
                               "mduffy",
                               "\"test\"",
                               "success",
                               "\"1.1\""
                           ]
                       ]
                   }
               ]
           }
       ]
   }
   ```

As you can see, we can use InfluxDB to track more than just measurements, and its simple API makes it very easy to integrate into other tools. By extending this into your tool set, you can quickly build up an easy-to-query and simple-to-graph time line of events; ranging from alerts to deployments and errors. Use your imagination and you will be able to build a wealth of data very simply.

See also

You can find more details of the InfluxDB REST API at `https://influxdb.com/docs/v0.9/guides/index.html`.

Gathering host statistics with Telegraf

One of the most useful examples of a time series database is to contain statistical information, and this is especially relevant and useful when it comes to tracking host performance. Due to the type of data, server resource monitoring can collect a great deal of measurements very quickly across a wide range of data points. Using the InfluxDB Telegraf tool, this can be done in a relatively straightforward manner and the data can be easily queried using InfluxDB's powerful set of query tools.

Getting ready

For this recipe, you need a server with InfluxDB installed and configured, and an Ubuntu 14.04 server to install `telegraf` onto.

How to do it…

The following steps will show you how to both install the tools and configure the Telegraf agent onto an Ubuntu host. Once this is done, we will look at how to configure it to log to an InfluxDB server:

1. Use the following command to fetch the `telegraf` package:

   ```
   $ wget http://get.influxdb.org/telegraf/telegraf_0.1.8_amd64.deb
   ```

2. Now, install the package using the following command:

   ```
   $ sudo dpkg -i telegraf_0.1.8_amd64.deb
   ```

3. Before we can start to send data to InfluxDB, we need to create a database for it to log information to. Use the following command to create a new database:

   ```
   $ curl -G http://localhost:8086/query --data-urlencode "q=CREATE
   DATABASE telegraf"
   ```

4. Next, we need to configure `telegraf` to log data on to the selected InfluxDB instance. On the host you have installed `telegraf`, to edit `/etc/opt/telegraf/telegraf.conf` and look for the following configuration item:

   ```
    [outputs.influxdb]
       url = "http://localhost:8086"
    database = "telegraf"
   ```

5. Edit the values to match your setup and ensure that the database matches the one that you created in step three.
6. Start the `telegraf` service using the following command:

   ```
   $ sudo service telegraf start
   ```

   At this point, Telegraf should start logging data into your InfluxDB.

7. You can select some sample data using the following command:

   ```
   $ curl -G 'http://localhost:8086/query?pretty=true' --data-
   urlencode "db=telegraf" --data-urlencode "q=SELECT * FROM io_
   write_bytes WHERE host='<<INFLUXHOST>>'"
   ```

8. Where `<<INFLUXHOST>>` is the same as your InfluxDB host. This will select the number of bytes written to the disks on the host and should produce an output similar to the following:

   ```
   {
       "results": [
           {
               "series": [
                   {
                       "name": "io_write_bytes",
                       "columns": [
                           "time",
                           "host",
                           "name",
                           "value"
                       ],
                       "values": [
                           [
                               "2015-09-16T21:25:54Z",
                               "influxdb1",
                               "vda1",
                               1259130880
                           ],
                       ]
                   }
               ]
           }
       ]
   ```

You can easily use this data to create real time charts of important metrics with the help of a tool such as Grafana. You can also use a tool such as Sensu to alert you about certain thresholds.

See also

You can find more information about Telegraf at `https://github.com/influxdb/telegraf`.

Exploring data with the InfluxDB data explorer

InfluxDB comes with a ready-to-use GUI to query your data; this makes exploring your data quick and easy. Although it's not a comprehensive tool and lacks niceties such as exporting, reporting, and so on, the built-in data explorer is great to get a feel for your time series data. Using this, you can easily pull certain data out of your InfluxDB database and use it to test queries for use in other tools.

Getting ready

For this recipe, you will need an InfluxDB server installed and configured and a data set to query.

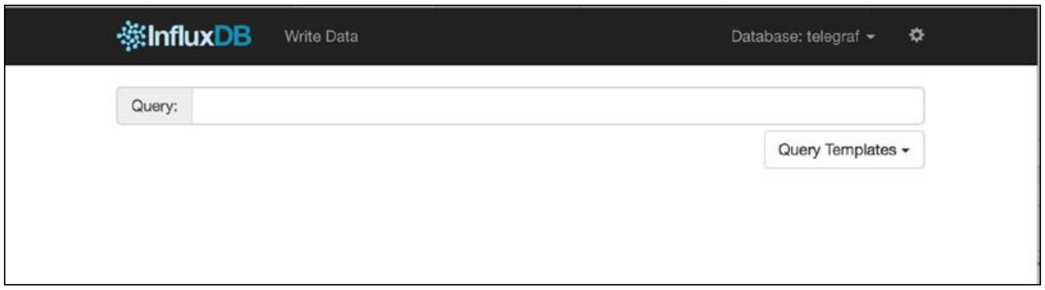
How to do it...

Let's explore the data using the InfluxDB data explorer:

1. Log on to the **InfluxDB** server using a browser to visit the following URL:

   `http://<INFLUXDBSERVER>:8083`

   You should see a screen similar to the following:



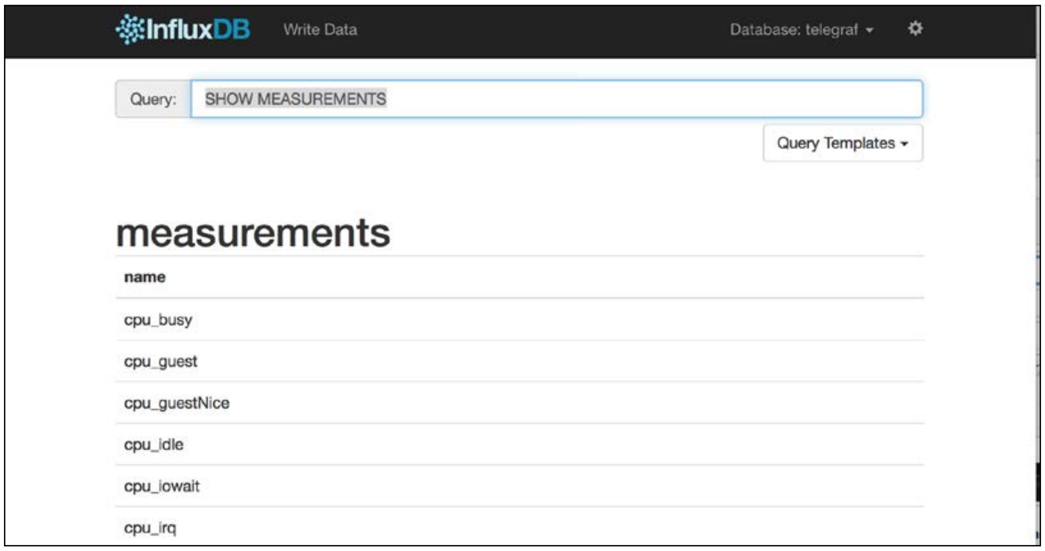
This panel allows you to run ad-hoc queries against your data sources. Select your database by clicking on the **Database telegraf** list on the top right menu and pick a database that has data to query.

For those of you who already have SQL experience, the following recipe will seem familiar; this is because the InfluxDB query language has been designed to be as SQL like as possible; however, it's worth checking the manual to understand the nuances of the query language. You can find a useful comparison at `https://influxdb.com/docs/v0.9/concepts/crosswalk.html`.

2. The first query we can execute is the one that allows us to see the measurements that are available to use. In the query panel, enter the following code:

```
SHOW MEASUREMENTS
```

This should return a list of all the measures in the selected database and should look something like this:



SHOW MEASUREMENTS

measurements

```
name
cpu_busy
cpu_guest
cpu_guestNice
cpu_idle
cpu_iowait
cpu_irq
```

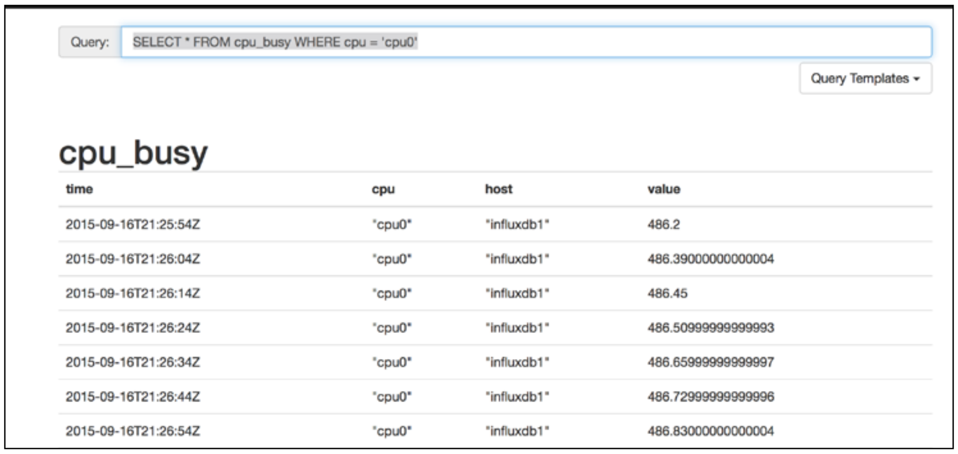
Using this query, we can list the measurements currently available in the database.

This and several other simple queries are available from the query template drop-down menu located underneath the query panel. The query template menu is a fantastic way of exploring some common InfluxDB queries.

3. Next, we can start querying to our data. From the data I have, I'm interested in the contents of the `cpu_busy` data set. The `cpu_busy` data is made up of several measures and I'm particularly interested in the statistics of the first CPU (`cpu0`). Now, we can use the following queries:

   `SELECT * FROM cpu_busy WHERE cpu = 'cpu0'`

   This command will return the following output:



Query: SELECT * FROM cpu_busy WHERE cpu = 'cpu0'

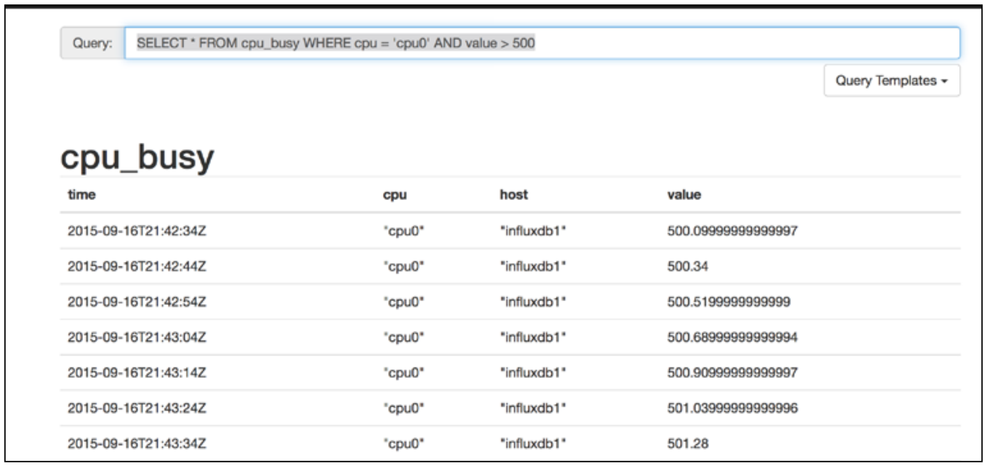
cpu_busy

time	cpu	host	value
2015-09-16T21:25:54Z	"cpu0"	"influxdb1"	486.2
2015-09-16T21:26:04Z	"cpu0"	"influxdb1"	486.39000000000004
2015-09-16T21:26:14Z	"cpu0"	"influxdb1"	486.45
2015-09-16T21:26:24Z	"cpu0"	"influxdb1"	486.50999999999993
2015-09-16T21:26:34Z	"cpu0"	"influxdb1"	486.65999999999997
2015-09-16T21:26:44Z	"cpu0"	"influxdb1"	486.72999999999996
2015-09-16T21:26:54Z	"cpu0"	"influxdb1"	486.83000000000004

4. Next, we can start to whittle down the data by adding conditions. In this case, we'll be looking into the CPU values higher that 500 (in this case, 500mhz). We can do this using the following query:

   `SELECT * FROM cpu_busy WHERE cpu = 'cpu0' AND value > 500`

   This command will return the following output:

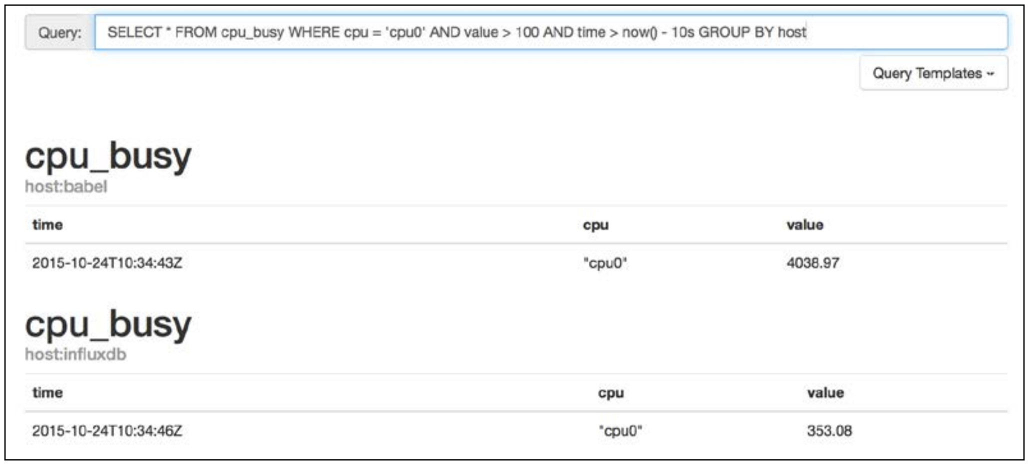


Query: SELECT * FROM cpu_busy WHERE cpu = 'cpu0' AND value > 500

cpu_busy

time	cpu	host	value
2015-09-16T21:42:34Z	"cpu0"	"influxdb1"	500.09999999999997
2015-09-16T21:42:44Z	"cpu0"	"influxdb1"	500.34
2015-09-16T21:42:54Z	"cpu0"	"influxdb1"	500.5199999999999
2015-09-16T21:43:04Z	"cpu0"	"influxdb1"	500.68999999999994
2015-09-16T21:43:14Z	"cpu0"	"influxdb1"	500.90999999999997
2015-09-16T21:43:24Z	"cpu0"	"influxdb1"	501.03999999999996
2015-09-16T21:43:34Z	"cpu0"	"influxdb1"	501.28

5. Now I want to look at this data on a host-by-host basis; we can do this using the `GROUP BY` statement by extending our query to look like this:



```
SELECT * FROM cpu_busy WHERE cpu = 'cpu0' AND value > 100 AND time > now() - 10s GROUP BY host
```

cpu_busy

host:babel

time	cpu	value
2015-10-24T10:34:43Z	"cpu0"	4038.97

cpu_busy

host:influxdb

time	cpu	value
2015-10-24T10:34:46Z	"cpu0"	353.08

As you can see, the results are now returned `grouped` by the host. Note that I'm also adding an additional parameter limiting the query by time. I'm asking for any data less than ten seconds from the system timestamp.

You can find more details about using the query language, including **Time Ranges**, at `https://influxdb.com/docs/v0.9/query_language/query_syntax.html`.

See also

You can find more details of the data explorer at `https://influxdb.com/docs/v0.9/query_language/data_exploration.html`.

Installing Grafana

Gathering data is a relatively useless task without an accessible method for both accessing and displaying it. This is especially true of time series data, which can produce a huge mass of information made of many small points of data. Without a tool that allows you to easily spot trends, the noise can easily become overwhelming; thus rendering your carefully gathered data useless.

InfluxDB can make use of an open source visualization tool called **Grafana**. Grafana is a sleek and stylish tool that allows you to take time series data and display it in many different fashions, including good-looking graphs. These can be combined into dashboards, which are perfect to display on TV's.

This recipe will show you how to install Grafana.

Getting ready…

For this recipe, you will need an InfluxDB data source with some data to query, and a server to host Grafana.

How to do it…

1. We start by fetching the latest `grafana` release using the following command:

   ```
   $ wget https://grafanarel.s3.amazonaws.com/builds/grafana_2.1.3_
   amd64.deb
   ```

2. Next, use the following command to install the Grafana pre-requisites:

   ```
   $ sudo apt-get install -y adduser libfontconfig
   ```

3. Now, we can install the `grafana` package using the following command:

   ```
   $ sudo dpkg -i grafana_2.1.3_amd64.deb
   ```

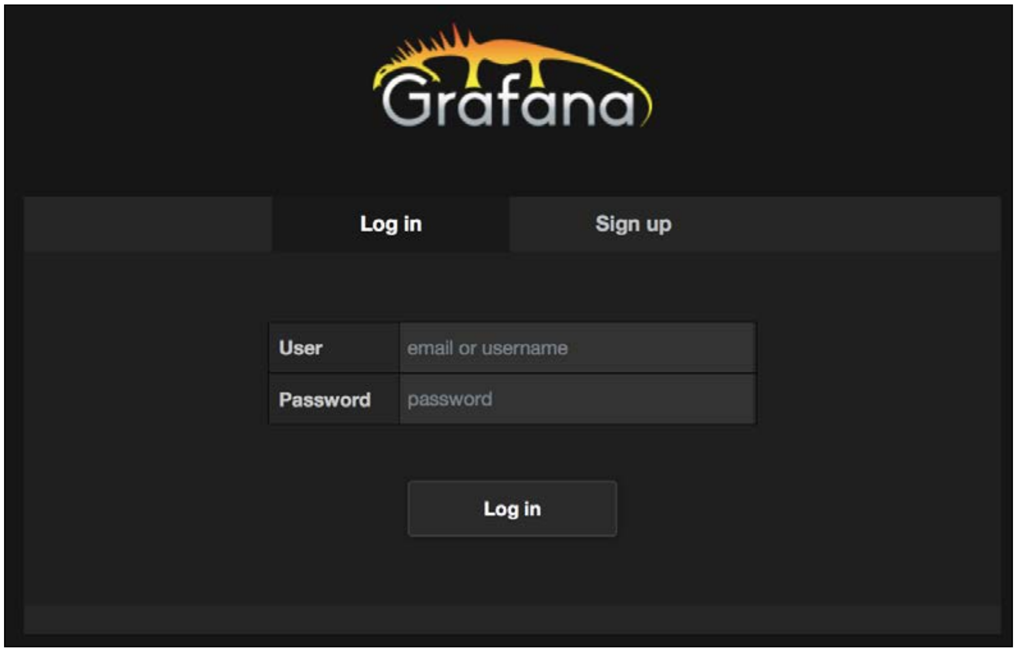
4. Now that `grafana` is installed, you can start it using the following command:

   ```
   $ sudo service grafana-server start
   ```

5. You should now be able to connect to Grafana by going to the following URL:

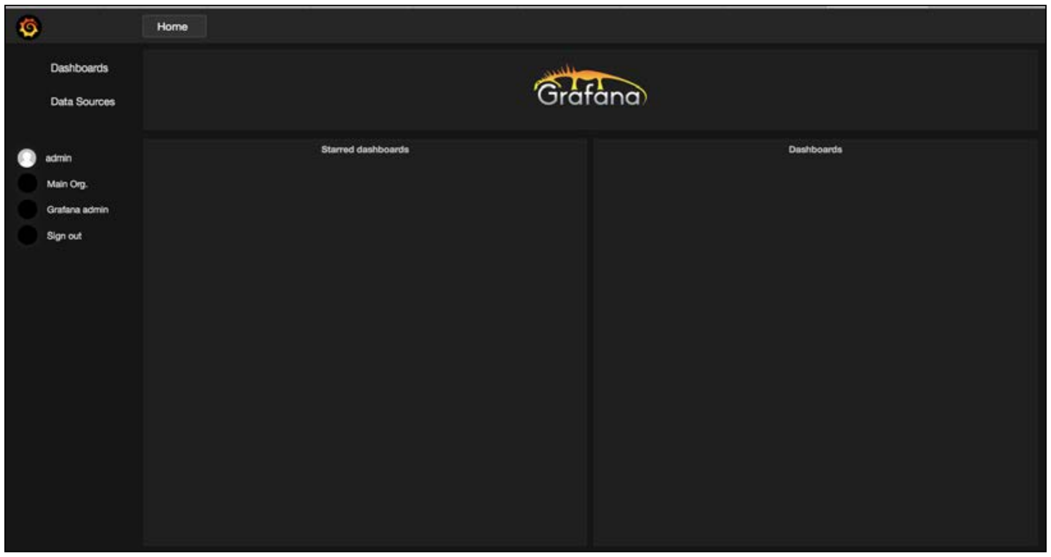
   `http://<GRAFANA SERVER>:3000`

   Here, `<GRAFANA SERVER>` is the name or IP address of your Grafana instance. You should be able to see a page similar to the following:



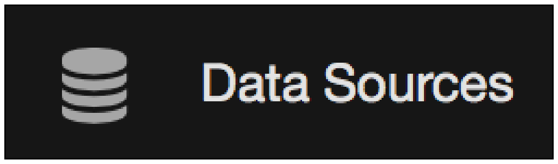
6. You can log in to the panel with the following user details:
    - **User**: `admin`
    - **Password**: `admin`

When logged in, you should be greeted with a page similar to the following:

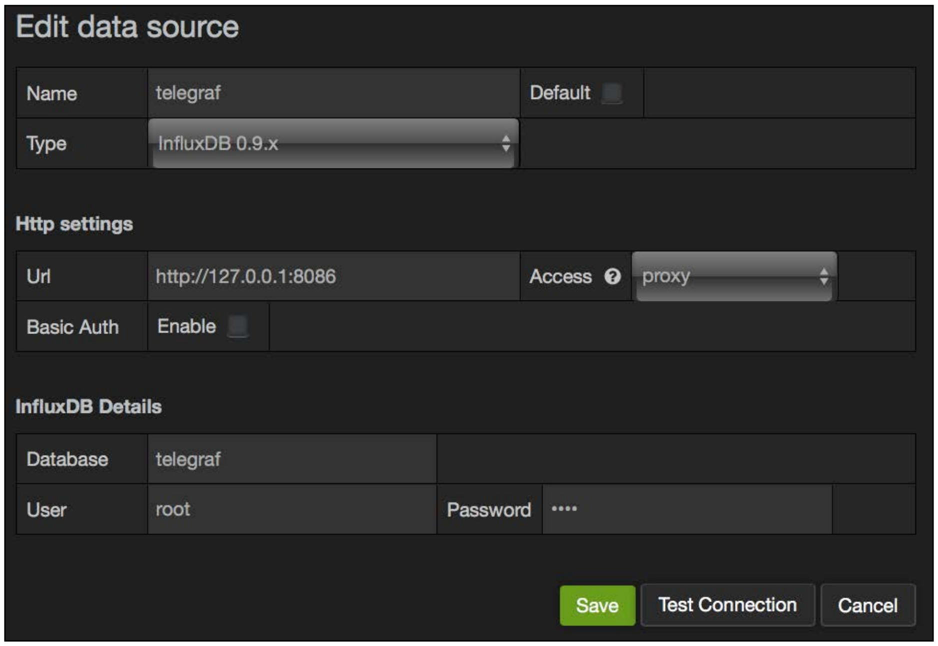


This is the default view of Grafana. When you create dashboards, you can see them listed on the pane on the far right, and you can add your favorites to the pane on the left for easy access.

7. Next, click on the **Data source** button, as shown in the following screenshot:



This will take you to a page that lists your configured data sources; this will currently be blank. On the top menu there is a button entitled **Add New**. Click on this and it will take you through to a panel that allows you to add a new source. This requires the details of your InfluxDB and when filled in, it should resemble the following screenshot:


Edit data source

Name	telegraf	Default	
Type	InfluxDB 0.9.x		

Http settings

Url	http://127.0.0.1:8086	Access	proxy
Basic Auth	Enable		

InfluxDB Details

Database	telegraf		
User	root	Password	••••

There are a few points to note. First, you can make this your default data source, to make the creation of new views on the data easier. Secondly, you can choose between several different types of data sources; this includes different versions of InfluxDB, so ensure that you choose correctly.

Once you are happy with your details, click on the **Save** button; you can also use the **Test Connection** button to test if Grafana can connect. Click on **OK** before saving.

See also

You can find further details about Grafana at http://docs.grafana.org.

Creating dashboards with Grafana

Once you have installed Grafana, you will have the ability to create attractive and informative dashboards that are ideal for display on devices such as projectors or TVs. The dashboards can display information from several different data sources, allowing you, for example, to display combined CPU statistics from a cluster of servers, alongside the number of orders taken in the same time period and the HTTP return codes. Grafana can hold any number of dashboards and makes it easy to embed links within dashboards to other dashboards. So, feel free to make specific dashboards for your data and they should remain easy to access.

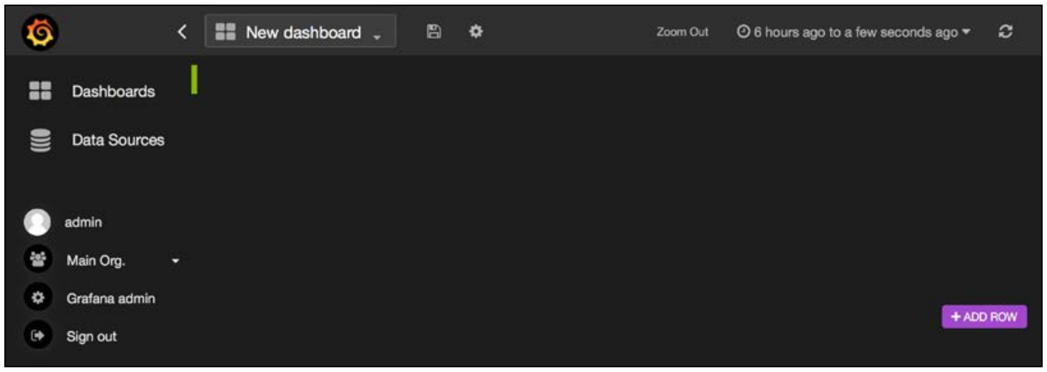
Getting ready

For this recipe, you need to have a server with InfluxDB and Grafana installed.

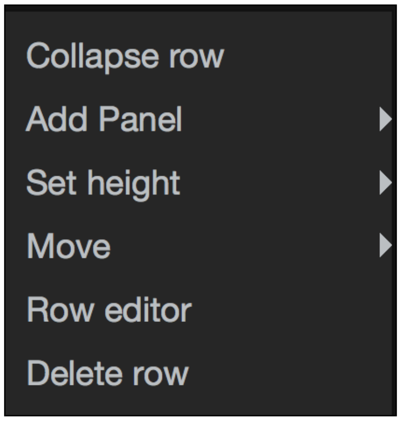
How to do it...

Let's create a dashboard with Grafana:

1. Log into the `grafana` panel. In the top menu, click on the menu item titled **Home**. Click on the option marked as **New dashboard** that you see at the bottom of the drop-down menu.
2. Next, you will see a screen that looks similar to the following screenshot:



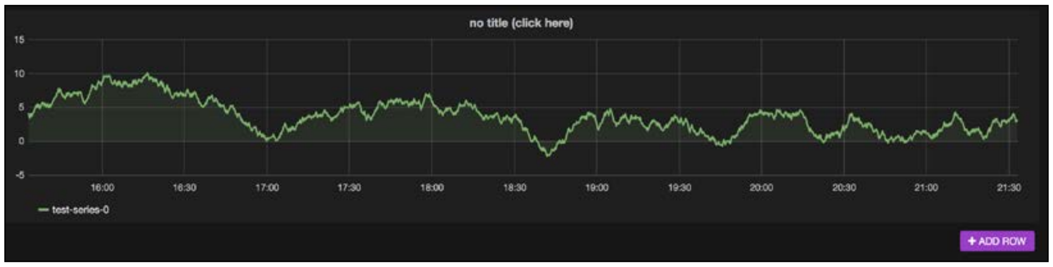
3. This is a blank dashboard that is ready for new content to be added. Observe the green strip to the right of the **Dashboards** menu items; this is a menu. When you click on it you will be presented with the following options:



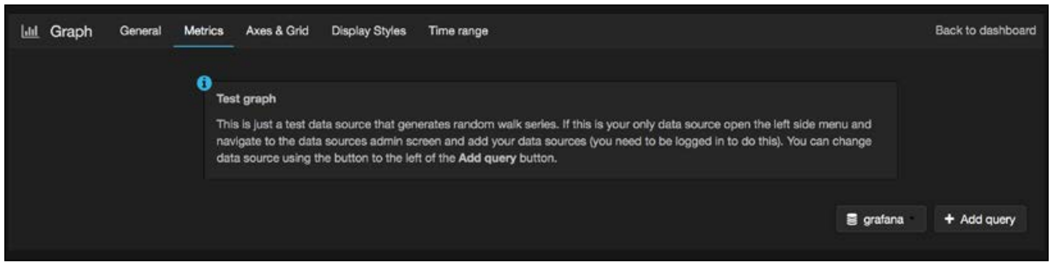
This menu allows you to edit your panel options. Let's start by clicking on the **Add Panel** link. This will open another menu listing the available options, which are as follows:

- **Graph:** This is a line graph of a selected data series
- **Single Stat**: This is a singular number derived from a selected series
- **Text**: This is a free text field that allows you to enter your own text
- **Dashboard list**: This allows you to construct links to further dashboards

4. Let's start by adding a new **Graph**. Click on the **Add panel** option and select **Graph**. This should add a new graph that looks like this screen shot:



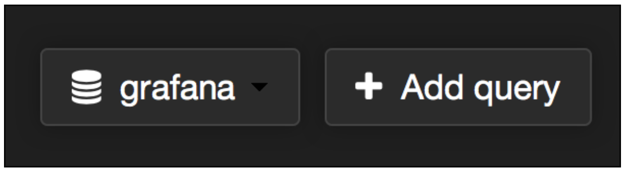
This is a default graph based on example data; let's change that to real data. Click on the graph title and click on the edit option. This will create a new window underneath the graph that looks like the following screenshot:



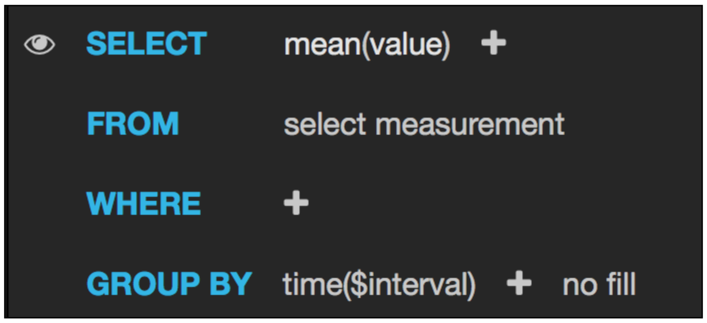
Test graph

This is just a test data source that generates random walk series. If this is your only data source open the left side menu and navigate to the data sources admin screen and add your data sources (you need to be logged in to do this). You can change data source using the button to the left of the **Add query** button.

5. This panel allows you to edit your new graph. Note the option at the bottom:



6. This allows you to set your data source. Click on the button titled **grafana** and select your data source. For these examples, I'm using the information derived from Telegraf. When you select your data source you are offered the option to select a query, as shown:



SELECT mean(value) +

FROM select measurement

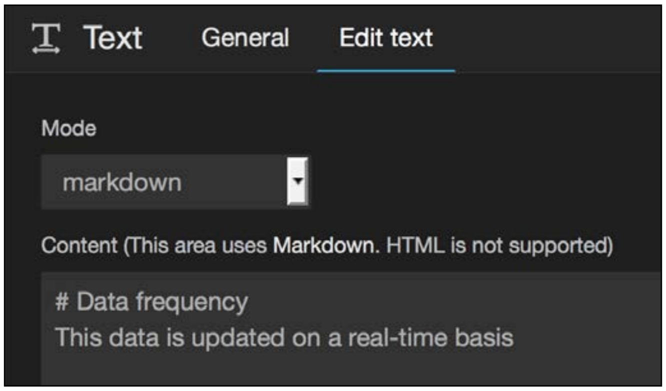
WHERE +

GROUP BY time($interval) + no fill

The `FROM` entry will allow you to select any measurement from your data source. Go ahead and select both your measurements and, if needed, a query to whittle the data down into the selection you want.

> By clicking on the + icon next to the `WHERE` clause, you will be taken into the interactive query builder; this will help you define queries to narrow down the data.

7. Once you are happy with your graph, click on **Back to Dashboard**.
8. Now that we have our first graph, we can add another item. Click on the button entitled **ADD ROW** in the pop-up menu and select **Text**. This will add a new blank text box to the dashboard. Click on the title of the text box and you will be offered a menu; click on the **Edit text** button. This will open a new panel that allows you to add a text `suh` as the following example:



Mode

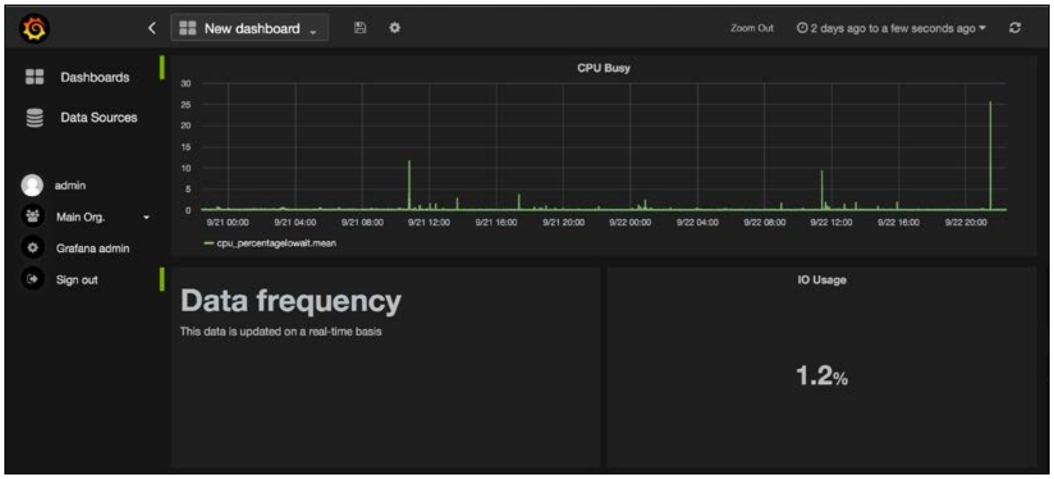
markdown

Content (This area uses Markdown. HTML is not supported)

```
# Data frequency
This data is updated on a real-time basis
```

Once you've added your text, click on **Back to Dashboard**.

9. Click on the green bar next to your new text panel and select **Add panel**. This will create a new panel next to your text panel. In this case, select **Single stat**. Once again, this will be a panel with example text. Click on the title and it should present a new panel; this is the same as the graphing panel and allows you to create a new query to select your data. Once you've selected your data, go back to the dashboard. Your dashboard should look similar to this:



Although basic, this dashboard is a great start. You can continue to add panels, but keep in mind that it needs to fit whichever device you are using to display it (TV, monitor, and so on). If you need to add a lot of data, the best approach is to create links that allow you to drill down into more detailed dashboards.

See also

You can find further information about dash boarding at `http://docs.grafana.org/reference/graph/`.

9
Log Management

In this chapter, we are going to cover the following subjects:

- Centralizing logs with syslog
- Using syslog templates
- Managing log rotation with the Logrotate utility
- Installing Elasticsearch, Logstash, and Kibana
- Importing logs into Elasticsearch with Logstash
- Using Kibana queries to explore data
- Using Kibana queries to examine data

Introduction

Log management is one of the most essential roles that systems administrators have been performing since the term came into being, and it's a crucial part of running any system. Without log management the logs will overflow, disks will fill up, and eventually data will be lost.

Until recently, the true value of logs has been understated by many systems administrators. Generally looked upon as a troubleshooting mechanism, logs had been consigned to being looked at only as a last resort and had been left to gather digital dust on a shelf somewhere. However, recently, there has been a renaissance in how logs are perceived, with developers and operators alike considering the humble log file more in the perspective of an event stream; that is, a continuous stream of information that not only indicates if you have issues, but can also be used to check for underlying patterns that can highlight significant state changes.

A good example of this can be found in the HTTP server access logs. These tend to contain not only issues (404, 500, and so on) but also successful transactions. This data is generally enriched to contain additional data, such as a client IP, pages access, response time, and so on. This additional metadata can then be further added to; for instance, using the originating IP to derive geographical location, which can give you a real-time view on business processes.

Event stream analysis is becoming a very large element of successful and scalable infrastructure, and many of the pioneers in the industry are embracing it to underpin items such as automated security scanning, platform scaling, and so on.

The recipes in this chapter will introduce you to several valuable tools for the management of logs, running the whole gamut of log management tasks, from centralization to rotation and finally into analysis. Armed with these recipes you should be ready to capture and explore many facets of data that your platform gathers in the course of the day and use them to scale, secure, and improve your platform.

Centralizing logs with Syslog

Generally speaking, most applications will have a logging facility that will produce text logs that can be written into an arbitrary location on a storage device (normally, defaulting to a local disk partition). Although this is an essential first step, it also produces problems; you need to provide adequate storage for logs and you need to put in place rotation to stop them from growing too large. If they contain vital information, you need to ensure that they are rotated to a safe location, but you may also need to ensure that they remain secure if they contain sensitive information, such as credit card numbers. On top of all of this, you will lose the current logs if you have a disaster on that node.

A good solution to manage these issues is to use a central location to store logs. This allows you to provide appropriate storage and gives you a central place to back up, secure, and examine logs. As long as the logging mechanism that your application uses supports syslog, it is straightforward. Syslog was originally used as the logging mechanism for the venerable Send mail MTA and was developed in the early 80s. Since then, it has been standardized and is now in use as the standard logging mechanism for most `*nix` based operating systems, with many implementations of the standard available.

Getting ready

For this recipe, you will need two Ubuntu 14.04 hosts; one to act as the Syslog server and another to act as a sending host.

How to do it...

This recipe will show you how to set up a server to act as a central Syslog server and how to configure a client log to utilize it. This will use the default syslog implementation on Ubuntu 14.04, Rsyslog:

1. Rsyslog is the default `syslog` package for Ubuntu 14.04 and should be pre-installed; you can double check this by issuing the following command:

   ```
   $ sudo apt-get install rsyslog
   ```

   This should return a message that states that `rsyslog` is already installed.

2. Next, we need to change the configuration of the Rsyslog package so that it listens for network connections. By default, Rsyslog is set only to listen to local socket connections. Edit the file `/etc/rsyslog.conf` and locate the following lines:

   ```
   # provides UDP syslog reception
   #$ModLoad imudp
   #$UDPServerRun 514

   # provides TCP syslog reception
   #$ModLoad imtcp
   #$InputTCPServerRun 514
   ```

3. Uncomment them so that they resemble the following:

   ```
   # provides UDP syslog reception
   $ModLoad imudp
   $UDPServerRun 514

   # provides TCP syslog reception
   $ModLoad imtcp
   $InputTCPServerRun 514
   ```

In the preceding configuration, we have enabled both UDP and TCP log reception. Generally speaking, I recommend using UDP to forward logs; it has a much lesser impact on performance than TCP as it is fire and forget. The downside is that some log messages might be lost if the server is too busy to receive the UDP packets. Use TCP where your logs are critical; otherwise, stick with UDP.

4. Restart the `rsyslog` service by issuing the following command:

   ```
   $ sudo service rsyslog restart
   ```

   Your Rsyslog server is now ready to use. Next, we can turn our attention to the Rsyslog client.

5. On the Rsyslog client, ensure that Rsyslog is installed.
6. We are going to start by ensuring that all logs dealing with the user login activity are forwarded to the Syslog server. This is an excellent place to start, as monitoring who logged into which server and when is a crucial part of securing your systems, and sending it off-host makes it harder to tamper with. Edit the `/etc/rsyslog.d/50-default.conf` file and find the following line:

   ```
   auth,authpriv.*                  /var/log/auth.log
   ```

   Change it to the following:

   ```
   auth,authpriv.*                  @<Syslog server>:514
   ```

[When you prepend your remote host with a single `@` symbol, you are using UDP. To use TCP, use `@@` instead.]

7. Restart the Rsylsog service on the client using the following command:

   ```
   $ sudo service rsyslog restart
   ```

8. Log on to the host server while monitoring the `/var/log/auth.log` log on your Syslog server; you should be able to see entries similar to the following snippet:

   ```
   Jun 29 07:47:05 sysloghost sshd[19563]: Accepted publickey for
   root from 178.251.183.111 port 33241 ssh2: RSA f0:0c:25:c1:9a:94:f
   f:20:6e:f7:57:70:9f:3c:9c:5c
   Jun 29 07:47:05 sysloghost sshd[19563]: pam_unix(sshd:session):
   session opened for user root by (uid=0)
   ```

   Notice the highlighted text; this is the hostname of the server that has sent the log entry and it can be used to search for specific entries.

[The field after the date in the syslog entry is always the hostname of the host sending that particular log entry.]

This can be very useful when you have more than a handful of servers reporting to your Syslog server and it enables you to use tools such as `grep` to quickly search for information across many servers at once.

See also

You can find the Rsyslog documentation at `http://www.rsyslog.com/doc/master/index.html#manual`.

Using syslog templates

Although it's advantageous to be able to send all the logs to a single location, there are times when you will want to be able to split logs out into separate log files based on certain criteria; for instance, a per-host log for certain elements.

Rsyslog supports this using a system of templates and the property replacer feature; this allows you to distribute logs of your choice into the location you need.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server acting as a Syslog server and a host to send logs to it.

How to do it…

Let's use the syslog templates for the:

1. As with the previous recipe, we're going to take the `auth` logs from our Ubuntu host and send them to our Syslog server. On the shipping host, ensure that you have a line that resembles the following in your copy of `/etc/rsyslog.d/50-default.conf`:

   ```
   auth,authpriv.*                 @<Syslog server>:514
   ```

2. Restart the `syslog` Daemon using the following command:

   ```
   $ sudo service rsyslog restart
   ```

3. On your Syslog server, open `/etc/rsyslog.d/50-default.conf` and edit it to include the following snippet:

   ```
   $template Remote, "/var/log/%HOSTNAME%/%syslogfacility-text%.log"
   auth,authpriv.*                        -?Remote
   ```

4. Locate the following line and comment it out with a `#`:

   ```
   auth,authpriv.*                        /var/log/auth.log
   ```

5. In the preceding code, we are disabling the default log settings for the `auth` log and using variables to replace the filename and path.

6. Restart the Syslog server using the following command:

   ```
   $ sudo service rsyslog restart
   ```

7. You should now find that when you log in to the host server, its `auth` messages will be sent to a file that will be located at `/var/log/<hostname>/auth.log`.

> As you may have considered, this technique can be used to organize any log that you want to be forwarded onto the remove Syslog server; indeed, for ephemeral hosts, it can be very useful, as you can create and destroy them at will and still retain all logs created over their life cycles.

See also

You can find details on the Rsyslog property replacer at `http://www.rsyslog.com/doc/property_replacer.html`.

Managing log rotation with the Logrotate utility

It's often surprising how much space can be consumed by something as simple as a plain text file, and without constant care and attention logs can grow to fill the available free space on a host. Fortunately, given their nature as plain text, they are compressible. Indeed, you can expect compression ratios of 80% or more on most log files.

The Logrotate utility is shipped with most Linux distributions and offers a simple yet powerful method to manage logs, allowing you to rotate, compress, and remove on the schedules you set. It also has the ability to run scripts both pre- and post-rotation, thus allowing you to send signals to applications to gracefully restart or to send logs to a remote location after compression. Most applications that are packaged with the operating system should come with a Logrotate configuration, but you should also ensure that any applications you develop or deploy are catered for.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host.

How to do it...

This recipe will show you how to create a new `logrotate` configuration, for example, app and also point out important configuration options:

1. In this example, we're going to rotate the logs on notional app located in `/usr/local/exampleapp/logs`.
2. To start, create a new file in `/etc/logrotate.d` called `exampleapp` (no extension) and insert the following content:

    ```
    /usr/local/exampleapp/logs/*.log {
            daily
            rotate 31
            copytruncate
            compress
            notifempty
    }
    ```

3. This is a relatively straight forward Logrotate entry, which will do the following:
    - Rotate any files with a `.log` extension in the directory `/usr/local/exampleapp/logs`
    - Rotate the log files daily
    - Keep 31 days' worth of logs
    - Will truncate files without removing them
    - The logs will be compressed using `gzip`
    - The logs will not be rotated if they are already empty, saving on empty archived logs

Be careful with the `copytruncate` option. It's useful when you have an application that can't accept a signal to reload and use a new `logfile`. Using `copytruncate` avoids the need for this by copying the contents of the current log and zeroing out the existing one; however, in the time between copying the log and zeroing it, there may be new entries that will be subsequently lost.

This example should serve as a good starting point for most applications. Use the Logrotate documentation to explore some further options that your application might require.

See also

You can find the documentation for Logrotate by issuing the following command:

```
$ man logrotate
```

Installing ElasticSearch, Logstash, and Kibana

Once you have established a policy to control retention, archiving, and centralization of your logs, you can consider how best to extract the data from them. Log analysis software has seen some serious growth in recent years, as an increasing number of systems administrators, developers, and managers realize the value of the data they can provide. Currently, Splunk has gained a great number of traction, offering both an easy-to-install and easy-to-use product with a great deal of integrations; however, it can be costly with a pricing model that ratchets up along with the quantities of data you wish to analyze. This has led to open source projects springing up and aiming to rival Splunk, in particular, it has been popularized by the trifecta of ElasticSearch, Logstash, and Kibana. Together these form what is popularly known as an ELK stack. These three products combine to offer a compelling alternative to Splunk; thus, allowing you to ship, analyze, and present data derived from your log streams.

This recipe will deal with the ElasticSearch and Kibana elements of the stack, allowing you to create a server that is ready to have logs shipped to it via Logstash.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server.

How to do it...

1. To start, we need to install a Java virtual machine to run ElasticSearch; either the Sun or the OpenJDK implementation is supported; we're going to use the OpenJDK distribution as it is packed into the Ubuntu repositories. Install the OpenJRE by issuing the following command:

   ```
   $ sudo apt-get install default-jre
   ```

2. Next, install the public signing key for the ElasticSearch repository with the following command:

```
$ wget -O - http://packages.elasticsearch.org/GPG-KEY-
elasticsearch | sudo apt-key add -
```

3. Next, create a new file within `/etc/apt/sources.list.d/` called `elasticsearch.list` and insert the following:

```
deb http://packages.elastic.co/elasticsearch/1.7/debian stable
main
```

4. Now, install ElasticSearch using the following command:

```
$ sudo apt-get update && sudo apt-get install elasticsearch
```

5. Next, start your ElasticSearch instance by issuing the following command:

```
$ sudo service elasticsearch start
```

6. Now, since we have ElasticSearch, we can install Kibana. Start by downloading the most recent release using the following command:

```
$ wget https://download.elastic.co/kibana/kibana/kibana-4.1.0-
linux-x64.tar.gz
```

7. Next, issue the following command to create a new user to run the Kibana process:

```
$ adduser kibana --home /opt/kibana-4.1.0-linux-x64
```

8. Now, decompress the installation using the following command:

```
$ cd /opt && tar -xvf <<fullpathtokibana.tar.gz>> && chown -R
kibana:kibana .
```

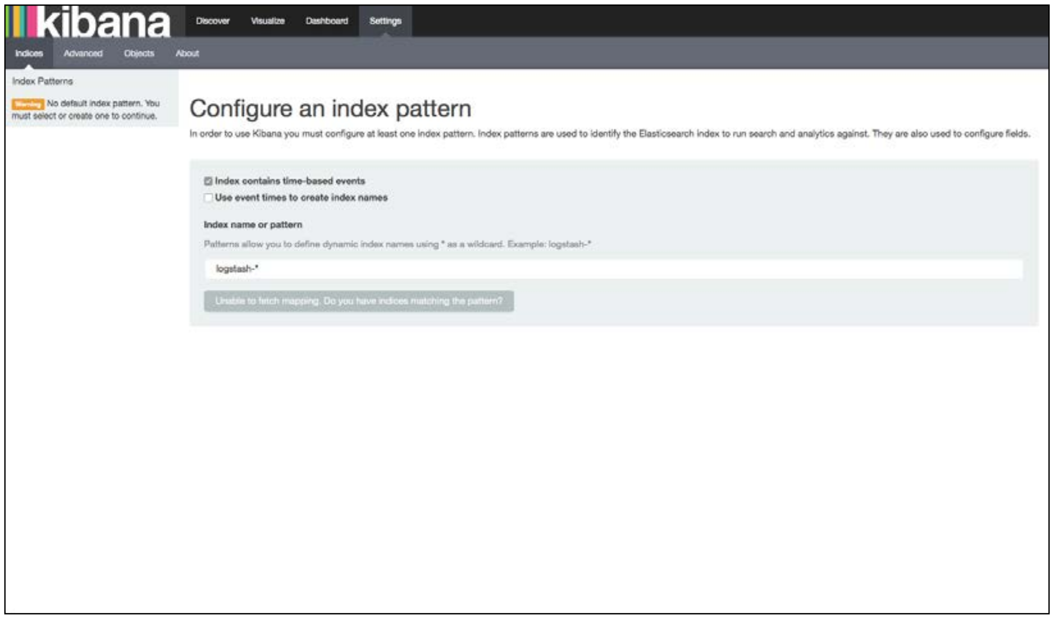
   Make sure you replace the `<<fullpathtokibana>>` text with the path to where you downloaded the `Kibana gzip`.

9. Run the following command to start the Kibana instance as a background process using the user we created in step six:

```
$ sudo su kibana -c '/opt/kibana-4.1.0-linux-x64/bin/kibana > /
opt/kibana-4.1.0-linux-x64/kibana.log &'
```

This isn't as robust as a true startup script and you may want to consider writing a more complete `init` script for production use.

10. You can now test your Kibana instance by going to the URL `http://<kibanaserver>:5601`. You should see a screen that looks similar to this:



No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events
[ ] Use event times to create index names

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

logstash-*

Unable to fetch mapping. Do you have indices matching the pattern?

This indicates that Kibana is ready for use. Next, we'll turn our attention to the Logstash server.

11. First, create a new file in `/etc/apt/sources.list.d/` called `logstash.list` and insert the following:

```
deb http://packages.elasticsearch.org/logstash/1.5/debian stable
main
```

12. Install the signing key using the following command:

```
wget -qO - https://packages.elasticsearch.org/GPG-KEY-
elasticsearch | sudo apt-key add -
```

13. Next, install the package using `apt-get`:

    ```
    $ sudo apt-get update && sudo apt-get install logstash
    ```

14. Now, we need to configure the Logstash server. The bulk of Logstash configuration comprises inputs and outputs; inputs take log streams in and outputs forward them onto a given system. In our case, we are going to create a new output for ElasticSearch. Use your editor to create a new file called `elasticsearch.conf` within the `/etc/logstash/conf.d` directory and insert the following snippet:

    ```
    output {
      elasticsearch {
        host => localhost
      }
    }
    ```

    This will take any data input into Logstash and place it into the ElasticSearch instance.

15. Finally, start the `logstash` process using the following command:

    ```
    $ sudo service logstash start
    ```

At this point, Logstash will start and then quit; this is expected, as we have no configured input. See the next recipe to configure an input for Logstash.

See also

- You can find the documentation for ElasticSearch installation at `https://www.elastic.co/guide/en/elasticsearch/reference/current/setup.html`
- You can find the installation documents for Kibana at `https://www.elastic.co/guide/en/kibana/current/setup.html`
- You can find the installation documents for Logstash at `https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html`

Importing logs into Elasticsearch with Logstash

Logstash can function as a log forwarding agent as well as a receiving server; however, due to it's reliance on both a JVM and a relatively large memory footprint, it is unsuitable for hosts of more modest means. Instead, we can use the Logstash forwarder (formerly known as Lumberjack). The Logstash forwarder is written in Go and has a significantly smaller footprint. As a result, it also removes the need for any external dependencies, such as a JVM. Using the Logstash forwarder, you can securely forward logs from your hosts onto your ELK stack.

Getting ready

For this recipe, you will need an Ubuntu 14.04 server acting as a Logstash server and an Ubuntu 14.04 server with Nginx to act as the forwarder. Nginx has been used to supply example logs and can be swapped out with the application of your choice.

How to do it...

The following steps will show you how to use a Logstash forwarder to import logs into Elasticsearch:

1. To start, we are going to install the Logstash log forwarder. Use the following command to download the package:

   ```
   $ wget https://download.elastic.co/logstash-forwarder/binaries/
   logstash-forwarder_0.4.0_amd64.deb
   ```

2. Next, we can install the package using this command:

   ```
   $ sudo dpkg -i logstash-forwarder_0.4.0_amd64.deb
   ```

3. Now, we need to return to the Logstash server to configure certificates. To maintain the security of your messages, the Logstash forwarder will only communicate over SSL connections, so at the very least we need to generate some self-signed certificates. On your Logstash server, create a directory to hold your keys and certificates:

   ```
   $ sudo mkdir -p /etc/logstash/ssl/certs && mkdir -p /etc/logstash/
   ssl/keys
   ```

4. Now, we create the key using the following command:

   ```
   $ openssl req -x509  -batch -nodes -newkey rsa:2048 -keyout
   logforwarder.key -out logforwarder.crt -subj /CN=<< ELASTICSEARCH
   SERVER NAME >>
   ```

5. Finally, we can copy the certificates into place with this command:

   ```
   $ sudo mv logforwarder.key /etc/logstash/ssl/keys && sudo mv
   logforwarder.crt /etc/logstash/ssl/certs
   ```

6. You will also need to copy the certificate onto the log forwarding host; copy it into `/etc/ssl/certs/logforwarder.crt`.

7. Now, we are ready to create our configuration; in this example, we're going to configure the forwarder to forward logs from our Nginx instance to our ElasticSearch server. Open the file `/etc/logstash-forwarder.conf` in your editor and replace the contents with the following:

   ```
   {
     "network": {
       "servers": [
         "<<ELASTICSEARCHSERVER>>:5043"
       ],
         "ssl ca": "/etc/ssl/certs/logforwarder.crt",
       "timeout": 15
     },
     "files": [
       {
         "paths": [
           "/var/log/syslog",
           "/var/log/*.log"
         ],
         "fields": {
           "type": "syslog"
         }
       },
       {
         "paths": [
           "/var/log/nginx/access.log"
         ],
         "fields": {
           "type": "nginx-access"
         }
       }
     ]
   }
   ```

Ensure that you replace `<<ELASTICSEARCHSERVER>>` with the IP address/name of your ElasticSearch server, ensuring that it matches the certificate name.

As you can see, this is straightforward JSON. This configuration will do the following:

- Forward the selected logs onto your ElasticSearch server
- Forward events from the syslog file and any file with a `.log` file extension within the `/var/logs` directory and process them as syslog files
- Forward events from the `nginx` log file located within `/var/log/nginx/access.log`

8. We have one last step we need to perform before we can receive the logs; we need to add a Logstash filter to correctly parse the incoming data. First, we create a directory to hold the pattern by issuing the following command:

```
$ sudo mkdir /opt/logstash/patterns/
```

9. Next, we can create our pattern. Using your editor, create a new file under `/opt/logstash/patterns` called `nginx` (no extension) and insert the following content:

```
NGINXACCESS %{IPORHOST:clientip} \[%{HTTPDATE:timestamp}\]
"%{WORD:verb} %{URIPATHPARAM:request} HTTP/%{NUMBER:httpversion}"
%{NUMBER:response} (?:%{NUMBER:bytes}|-) (?:"(?:%{URI:referrer}|-
)"|%{QS:referrer}) %{QS:agent}
```

This takes the incoming log and breaks it into discrete pieces of data for insertion into the ElasticSearch index.

You can find further information on Logstash patterns at `https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html`.

10. Next, we need to adjust our Logstash server to use our new pattern. Create a new file called `filter_nginx.conf` under `/etc/logstash/conf.d` and insert the following JSON:

```
filter {
  if [type] == "nginx-access" {
    grok {
      match => { "message" => "%{NGINXACCESS}" }
    }
  }
}
```

> Nginx is also able to log straight to JSON, thus allowing you to simplify the preceding steps by simply importing the raw JSON log. You can find the `nginx logging` details at `https://www.nginx.com/resources/admin-guide/logging-and-monitoring/`.

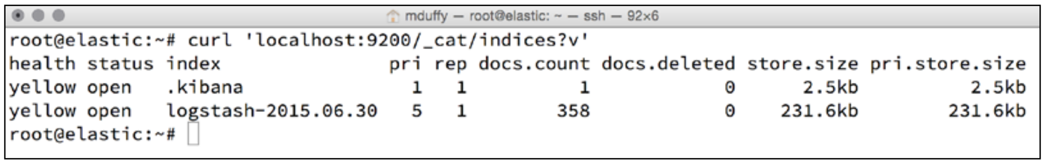
11. This configuration file causes Logstash to inspect incoming data for any that match a type of `nginx-access`. If it matches, then the data is parsed via the pattern we created earlier. You can enable your new configuration by restarting Logstash using the following command:

    ```
    $ sudo service logstash restart
    ```

12. You can test that your logs are being forwarded correctly by querying the indices on your Elastic server. On the Elastic host, use the following command:

    ```
    $ curl 'localhost:9200/_cat/indices?v'
    ```

    You should receive a reply that is similar to the following screenshot:



```
root@elastic:~# curl 'localhost:9200/_cat/indices?v'
health status index               pri rep docs.count docs.deleted store.size pri.store.size
yellow open   .kibana               1   1          1            0      2.5kb          2.5kb
yellow open   logstash-2015.06.30   5   1        358            0    231.6kb        231.6kb
root@elastic:~# 
```

Notice the dated index; this is the content of the logs we've forwarded on and is ready to query.

See also

You can find the documentation for the Log forwarder at `https://github.com/elastic/logstash-forwarder`.

Using Kibana queries to explore data

Once you have your data indexed into ElasticSearch, you will want to work with it to reveal anything of interest. Kibana is a fantastic tool to enable this, allowing you to query, display and report on data of interest. Kibana offers an easy-to-use GUI to explore your data, allowing both ad-hoc data exploration and the creation of stunning and detailed dashboards.

In this recipe we're going to focus on using Kibana to explore data to discover underlying patterns within an Nginx access log.

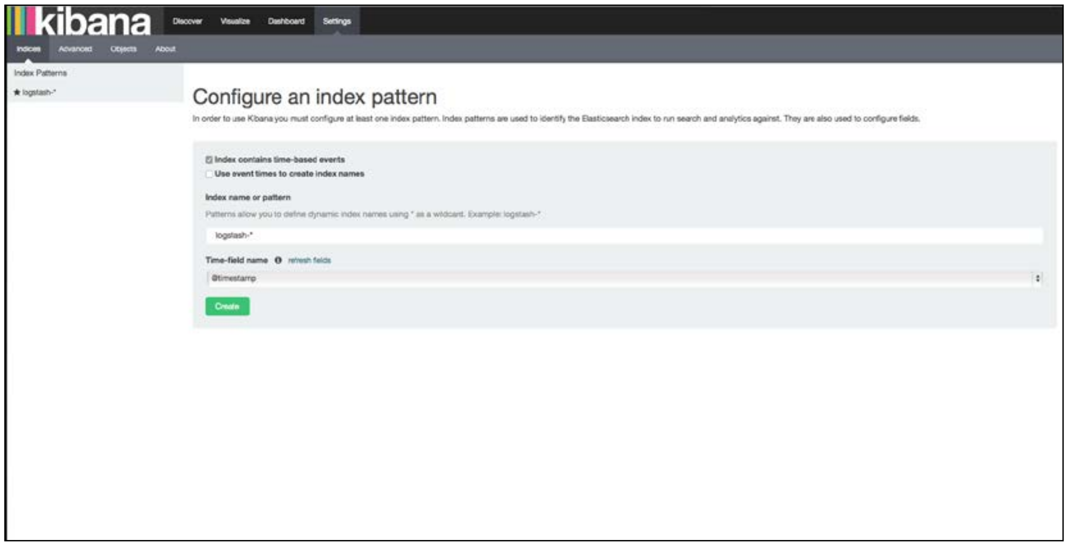
Getting ready

For this recipe you need an Ubuntu 14.04 server with Kibana and ElasticSearch installed; you should also have set up some inputs into ElasticSearch, preferably from an Nginx server.

How to do it...

The following steps will give you a very quick tour of how to locate and view data within Kibana:

1. First, point your browser at your Kibana instance (normally located at `<<kibanaserver>>:5601`). You should be able to see a page similar to the following:



Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events
[ ] Use event times to create index names

Index name or pattern
Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*
logstash-*

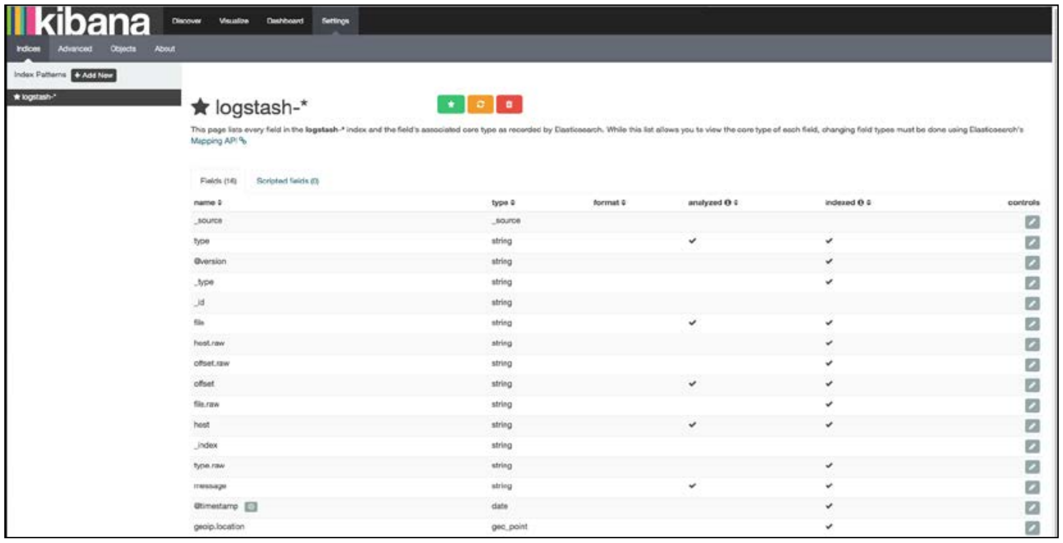
Time-field name refresh fields
@timestamp

This allows you to configure your initial index pattern; this maps back to the indices available within ElasticSearch.

If you need to see a list of available indices, you can issue this command against your ElasticSearch server: `curl 'localhost:9200/_cat/indices?v'`.

Generally speaking, if you're using Logstash to ship your logs into ElasticSearch, you should be fine accepting the default; hit the button marked **Create**.

2. The next screen allows you to select fields to analyze and should look something similar to the following screenshot:



logstash-*

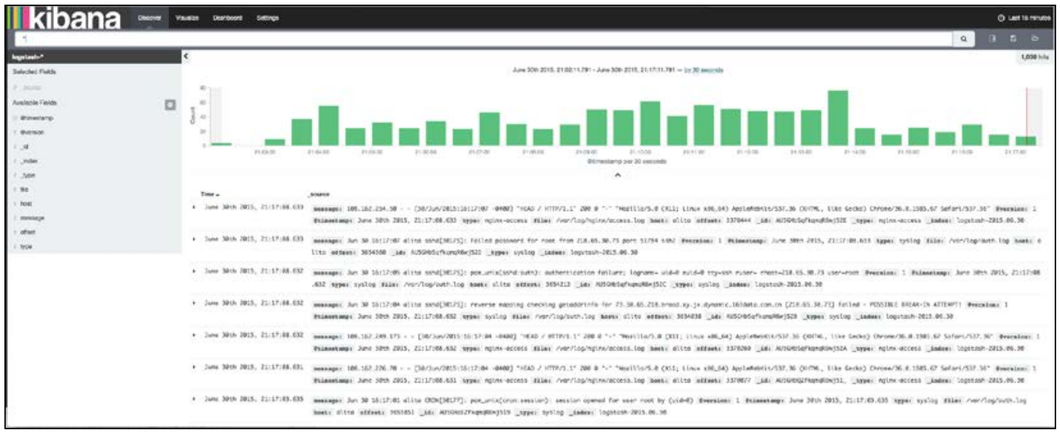
This page lists every field in the **logstash-*** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's Mapping API

Fields (16) Scripted fields (0)

name	type	format	analyzed	indexed	controls
_source	_source				
type	string		✓	✓	
@version	string			✓	
_type	string			✓	
_id	string				
file	string		✓	✓	
host.raw	string			✓	
offset.raw	string			✓	
offset	string		✓	✓	
file.raw	string			✓	
host	string		✓	✓	
_index	string				
type.raw	string			✓	
message	string		✓	✓	
@timestamp	date			✓	
geoip.location	geo_point			✓	

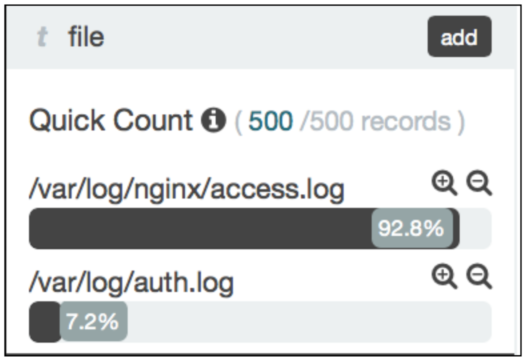
Kibana will attempt to work out the type for the field, but if it's a little wide of the mark you can use the pencil icon at the end of the row to adjust the setting. Once you're happy with your fields, you can click on the **Discover** menu item at the top.

3. The **Discover** page is where you can start to truly dig into your data. By default, it displays a graph of activity over time of all of the data that is indexed, and should look something similar to this:

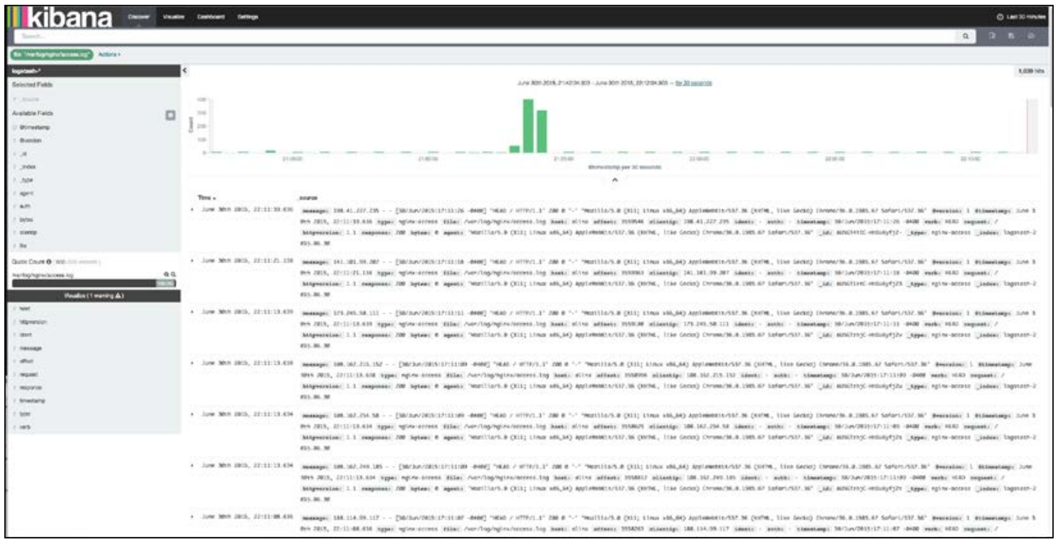


```
Time    _source
June 30th 2015, 21:17:08.633  message: 108.162.234.50 - - [30/Jun/2015:16:17:07 -0400] "HEAD / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.67 Safari/537.36" @version: 1 @timestamp: June 30th 2015, 21:17:08.633 type: nginx-access file: /var/log/nginx/access.log host: elito offset: 3370444 _id: AU5GHbSqfkqmdR0ej52E _type: nginx-access _index: logstash-2015.06.30
June 30th 2015, 21:17:08.633  message: Jun 30 16:17:07 elito sshd[30175]: Failed password for root from 218.65.30.73 port 51794 ssh2 @version: 1 @timestamp: June 30th 2015, 21:17:08.633 type: syslog file: /var/log/auth.log host: elito offset: 3654300 _id: AU5GHbSqfkqmdR0ej52D _type: syslog _index: logstash-2015.06.30
June 30th 2015, 21:17:08.632  message: Jun 30 16:17:05 elito sshd[30175]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=218.65.30.73 user=root @version: 1 @timestamp: June 30th 2015, 21:17:08.632 type: syslog file: /var/log/auth.log host: elito offset: 3654213 _id: AU5GHbSqfkqmdR0ej52C _type: syslog _index: logstash-2015.06.30
June 30th 2015, 21:17:08.632  message: Jun 30 16:17:04 elito sshd[30175]: reverse mapping checking getaddrinfo for 73.30.65.218.broad.xy.jx.dynamic.163data.com.cn [218.65.30.73] failed - POSSIBLE BREAK-IN ATTEMPT! @version: 1 @timestamp: June 30th 2015, 21:17:08.632 type: syslog file: /var/log/auth.log host: elito offset: 3654038 _id: AU5GHbSqfkqmdR0ej52B _type: syslog _index: logstash-2015.06.30
June 30th 2015, 21:17:08.632  message: 108.162.249.175 - - [30/Jun/2015:16:17:04 -0400] "HEAD / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.67 Safari/537.36" @version: 1 @timestamp: June 30th 2015, 21:17:08.632 type: nginx-access file: /var/log/nginx/access.log host: elito offset: 3370260 _id: AU5GHbSqfkqmdR0ej52A _type: nginx-access _index: logstash-2015.06.30
June 30th 2015, 21:17:08.631  message: 108.162.226.70 - - [30/Jun/2015:16:17:04 -0400] "HEAD / HTTP/1.1" 200 0 "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.67 Safari/537.36" @version: 1 @timestamp: June 30th 2015, 21:17:08.631 type: nginx-access file: /var/log/nginx/access.log host: elito offset: 3370077 _id: AU5GHbQ2fkqmdR0ej51_ _type: nginx-access _index: logstash-2015.06.30
June 30th 2015, 21:17:03.635  message: Jun 30 16:17:01 elito CRON[30177]: pam_unix(cron:session): session opened for user root by (uid=0) @version: 1 @timestamp: June 30th 2015, 21:17:03.635 type: syslog file: /var/log/auth.log host: elito offset: 3653851 _id: AU5GHa52fkqmdR0ej519 _type: syslog _index: logstash-2015.06.30
```

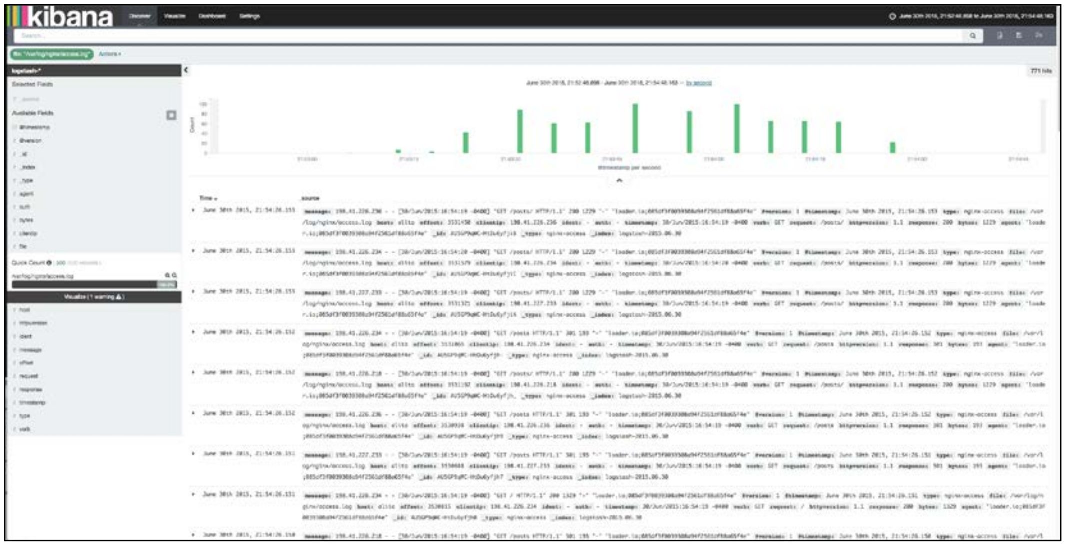
4. From here, we can start to drill into our data. At present, I have two different data sources available to me; my server's `auth.log` and my Nginx access log. Let's take a look at the Nginx access logs first. On the left-hand side of the menu, click on the item marked **file**; it should reveal a drop-down menu that looks similar to this:



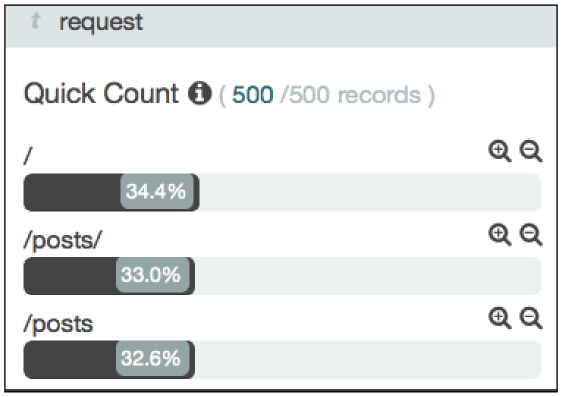
5. This menu allows you to quickly drill into the individual files that you have shipped to Kibana. It also allows you to have a look at how active the files are within the time frame. As we can easily see, my Nginx access log is by far the busiest. Let's click on the icon and drill into the data. As soon as you drill into any data, the main page updates to reflect this; for instance, my example data now looks similar to this:



6. You can already see that there is an anomalous spike of traffic in the preceding chart. There are several ways we can explore this, but by far the easiest is to click and drag over the time series we're interested in; this binds the data to the period selected:



7. By zooming in, we can easily see that the traffic is coming in over a very short period; a minute in this case. We can now dig deeper into this by examining the pages that we're requesting by opening up the requests field on the left-hand menu:



t request

Quick Count (500 /500 records)

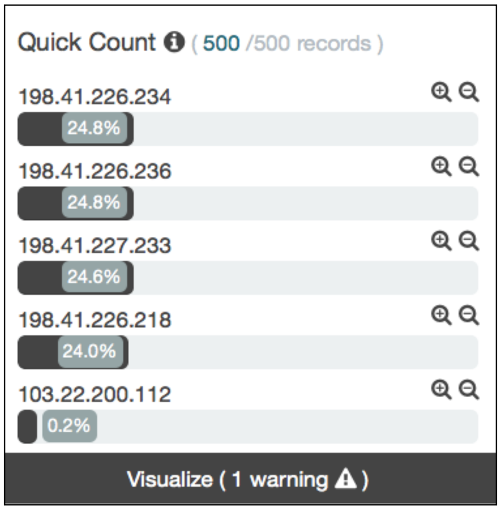
/ 34.4%

/posts/ 33.0%

/posts 32.6%

As you can see, the requests are evenly split.

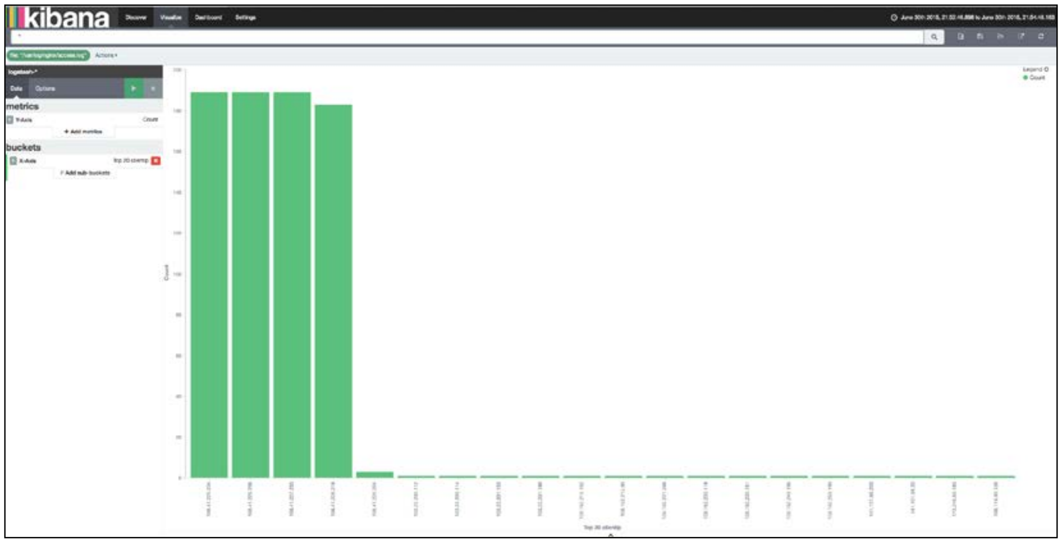
8. We can use the Visualize feature within Kibana to explore the data at hand. Right now, we'll take a look at the IP addresses. Click on the **clientIP** field on the left-hand side; it should look something similar to this:



9. As you can see, it has ranked the IPs by frequency; however, we can use the visualization feature to look at the data in more detail. Click on the **Visualize** button at the bottom.

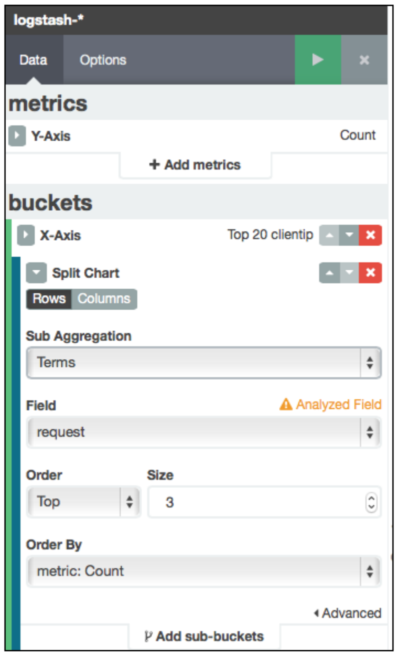
The warning is the result of the field containing analyzed data. Due to the possibly huge amount of variance available within an analyzed field, Kibana will warn you that this could be a computationally expensive operation.

10. The next screen arrays the IP's in order of the number of requests over the selected time period and will look similar to this:



As you can see, only four IP's make up the bulk of the traffic!

11. Within the visualization we can report on this further. On the left-hand side, under buckets, click on the **Add sub-buckets** button. This opens a drop-down menu that allows you to add further data within the visualization. In this case, add the options as per the following screen shot:


logstash-*

Data Options

metrics

Y-Axis Count

+ Add metrics

buckets

X-Axis Top 20 clientip

Split Chart

Rows Columns

Sub Aggregation: Terms

Field: request (Analyzed Field)

Order: Top

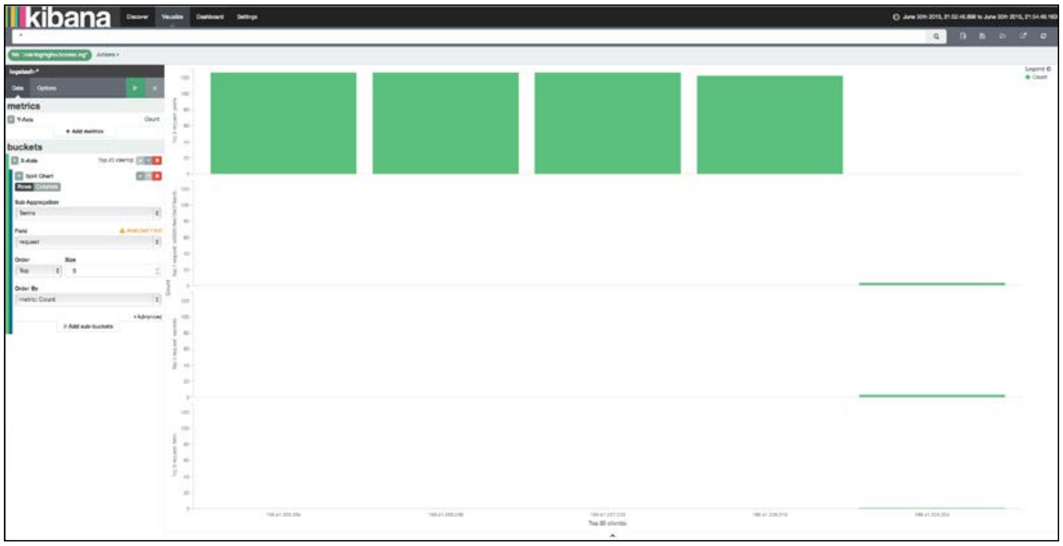
Size: 3

Order By: metric: Count

Advanced

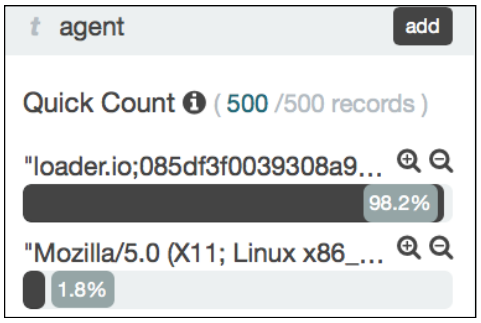
Add sub-buckets

12. Once you've added the options, click on the green **play** button at the top to run the query. This will order the results of the IPs by the top three requested URLs, and it should look similar to the following:



Although you can't make out the text in this image, you can see that the bulk of the requests are going to a single page; an unusual pattern.

13. As a final check, we can go back to the main page by clicking on the **Discover** button at the top and then clicking on the field on the left-hand side marked as **agent**. In my example, it looks similar to this:



t agent
add
Quick Count (500 /500 records)
"loader.io;085df3f0039308a9…
98.2%
"Mozilla/5.0 (X11; Linux x86_…
1.8%

Plainly, this activity is the result of a load test and we have spotted and investigated it using Kibana.

See also

You can find the getting started guide for Kibana at `https://www.elastic.co/guide/en/kibana/current/getting-started.html`.

Using Kibana queries to examine data

The previous recipe was a whistle-stop tour of using the Kibana interface to interactively drill into and examine your data; however, some of the true power of Kibana is its ability to use the ElasticSearch query language to allow you to select elements of information for examination.

Although you can impart order to your data when you ship it using Logstash, you will probably still have a lot of unstructured data that you need to examine. Using ElasticSearch queries, you can start to construct queries to examine your data and use Kibana to display it in an easy to understand manner. This recipe will take a look at some simple ElasticSearch queries to examine security issues on a Linux host.

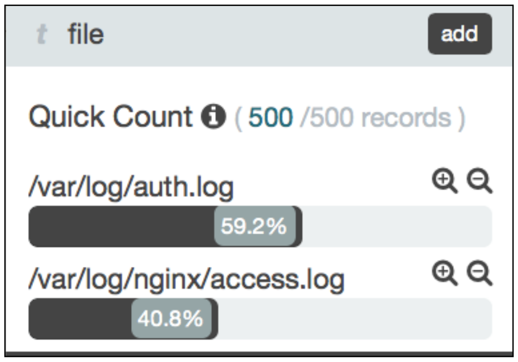
Getting ready

For this recipe, you will need an ELK server plus some data forwarded from an `auth.log`.

How to do it…

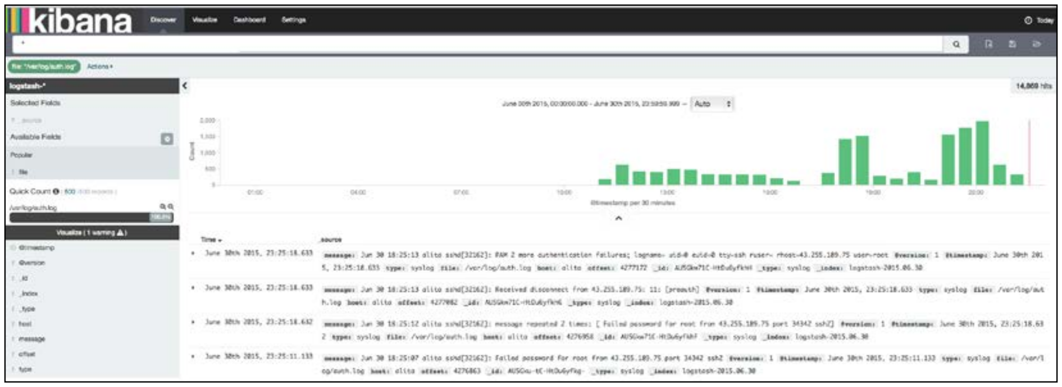
The following steps will show you how to search your data using queries in Kibana and how to save them as Dashboards:

1. Log onto your Kibana server and select the `files` tab from the left-hand side; it should look similar to the following screen shot:



   Select the plus sign next to the `auth.log` to drill further into that data.

2. The main screen should now look similar to the following:

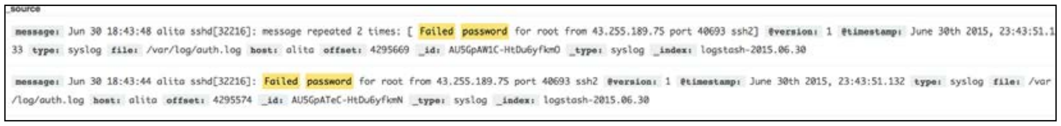


```
Time ▾                        _source
▸ June 30th 2015, 23:25:18.633  message: Jun 30 18:25:13 alito sshd[32162]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh ruser= rhost=43.255.189.75 user=root @version: 1 @timestamp: June 30th 2015, 23:25:18.633 type: syslog file: /var/log/auth.log host: alito offset: 4277172 _id: AU5Gkm71C-HtDu6yfkhH _type: syslog _index: logstash-2015.06.30
▸ June 30th 2015, 23:25:18.633  message: Jun 30 18:25:13 alito sshd[32162]: Received disconnect from 43.255.189.75: 11: [preauth] @version: 1 @timestamp: June 30th 2015, 23:25:18.633 type: syslog file: /var/log/auth.log host: alito offset: 4277082 _id: AU5Gkm71C-HtDu6yfkhG _type: syslog _index: logstash-2015.06.30
▸ June 30th 2015, 23:25:18.632  message: Jun 30 18:25:12 alito sshd[32162]: message repeated 2 times: [ Failed password for root from 43.255.189.75 port 34342 ssh2] @version: 1 @timestamp: June 30th 2015, 23:25:18.632 type: syslog file: /var/log/auth.log host: alito offset: 4276958 _id: AU5Gkm71C-HtDu6yfkhF _type: syslog _index: logstash-2015.06.30
▸ June 30th 2015, 23:25:11.133  message: Jun 30 18:25:07 alito sshd[32162]: Failed password for root from 43.255.189.75 port 34342 ssh2 @version: 1 @timestamp: June 30th 2015, 23:25:11.133 type: syslog file: /var/log/auth.log host: alito offset: 4276863 _id: AU5Gku-tC-HtDu6yfkg- _type: syslog _index: logstash-2015.06.30
```

3. Although you cannot make out the text, the graph is the point of focus; this shows us the `auth` log activity over time. Now, we should start constructing our query; in this instance, I'm interested in failed logins. In the search bar in the top, enter the following:

   ```
   "Failed password"
   ```

4. You will find that Kibana is now filtering the log messages that contain that term; it will also update the graphs and data to reflect this and highlight the data. See the following screen shot for an example:



```
_source
message: Jun 30 18:43:48 alito sshd[32216]: message repeated 2 times: [ Failed password for root from 43.255.189.75 port 40693 ssh2] @version: 1 @timestamp: June 30th 2015, 23:43:51.133 type: syslog file: /var/log/auth.log host: alito offset: 4295669 _id: AU5GpAW1C-HtDu6yfkmO _type: syslog _index: logstash-2015.06.30

message: Jun 30 18:43:44 alito sshd[32216]: Failed password for root from 43.255.189.75 port 40693 ssh2 @version: 1 @timestamp: June 30th 2015, 23:43:51.132 type: syslog file: /var/log/auth.log host: alito offset: 4295574 _id: AU5GpATeC-HtDu6yfkmN _type: syslog _index: logstash-2015.06.30
```

5. It's certainly nice to be able to search for a term, but you can also chain together queries with operators to gain a little granularity. For instance, I want to be able to filter out the entries that are failed logins for the root user and see the non-root accounts that are being affected. In your search bar, enter the following:

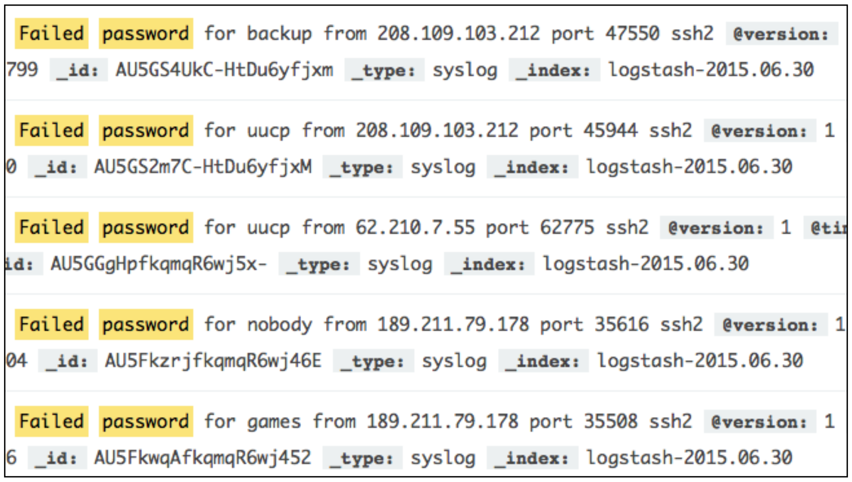
   ```
   ("Failed password") NOT ("root")
   ```

   This runs the first query (find all entries with the text **Failed** password) and then use the `NOT` operator to exclude the ones that include the text `root`.

6. Let's take it further and also exclude invalid users. This allows us to see the non root users who are actually present on the box and are being targeted. Enter the following query into the search bar:

   ```
   ("Failed password") NOT ("root") NOT ("invalid user")
   ```

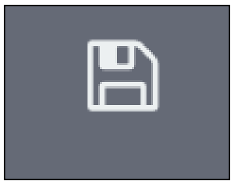
Hit the **run** button. In my example, my result looks similar to this:



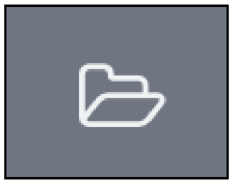
```
Failed password for backup from 208.109.103.212 port 47550 ssh2 @version:
799 _id: AU5GS4UkC-HtDu6yfjxm _type: syslog _index: logstash-2015.06.30

Failed password for uucp from 208.109.103.212 port 45944 ssh2 @version: 1
0 _id: AU5GS2m7C-HtDu6yfjxM _type: syslog _index: logstash-2015.06.30

Failed password for uucp from 62.210.7.55 port 62775 ssh2 @version: 1 @ti
id: AU5GGgHpfkqmqR6wj5x- _type: syslog _index: logstash-2015.06.30

Failed password for nobody from 189.211.79.178 port 35616 ssh2 @version: 1
04 _id: AU5FkzrjfkqmqR6wj46E _type: syslog _index: logstash-2015.06.30

Failed password for games from 189.211.79.178 port 35508 ssh2 @version: 1
6 _id: AU5FkwqAfkqmqR6wj452 _type: syslog _index: logstash-2015.06.30
```

As you can see, there are users such as UUCP, backup, and others who are valid users and are being targeted by some form of brute-force attack.

7. This is an interesting query and one that I'd like to be able to quickly return to. To do this, we can save the query. Click on the **disk** icon on the top left corner:



8. You will be prompted to name the saved search; go ahead and call it something meaningful. Now, whenever you want to see this query, you can load it by clicking on the **load** icon:



You can use this feature to build up a library of reports that allow members of the team who are less familiar with the query language to quickly and easily access information.

See also

You can find more details of how to use Kibana to query data at `https://www.elastic.co/guide/en/kibana/current/discover.html` and `https://www.elastic.co/guide/en/kibana/current/discover.html`.

10
Monitoring with Sensu

In this chapter, we are going to cover the following topics:

- Installing a Sensu server
- Installing a Sensu client
- Installing check prerequisites
- Finding community checks
- Adding a DNS check
- Adding a disk check
- Adding a RAM check
- Adding a process check
- Adding a CPU check
- Creating e-mail alerts
- Creating SMS alerts
- Using Ansible to install Sensu

Introduction

One of the cornerstones of DevOps engineering is the effective monitoring of resources and services, and the ability to react to them in a timely fashion. This is not unique to DevOps; monitoring has been one of the key aspects of running a system for as long as there have been systems to run.

Monitoring takes many forms and there is no such thing as a finished monitoring system. Just when you think you have everything monitored, you will find an obscure edge case that will cause issues if not correctly accounted for. On the other side, you need to ensure that your monitoring is accurate. Nothing kills a monitoring platform quicker than having to wade through a wall of false alerts. People start to ignore alerts, sooner rather than later, and you end up with an outage, which you could have seen coming but did not.

Fortunately, there are many products to choose from when it comes to monitoring and alerting, both open source and commercial, and each has its strengths and weaknesses. For many years, the de facto standard for open source monitoring was Nagios (`https://www.nagios.org`). Nagios is a hugely popular product and is used all over the world. However, it has some weaknesses. Several branches of Nagios are addressing some of these perceived weakness, with Icinga (`https://www.icinga.org`) becoming especially popular. However, some projects have gone further and have started to re-evaluate the core principles to rearrange the monitoring infrastructure.

One more promising product is Sensu (`https://sensuapp.org`). Sensu takes a different approach to many monitoring solutions and, rather than using a master/client solution, it uses a Publication/Subscription model using a message queue. This approach makes it far simpler to configure monitoring for a large number of services and is useful for environments that contain a large amount of ephemeral hosts. By allowing clients to subscribe simply and easily to a set of checks, it makes it much easier to roll out new clients; they simply start the client with the correct subscription, and start running and sending results back to the master to process.

Other than utilizing a message queue, Sensu allows for custom checks to be written relatively easily and with flexibility. This allows the responsibilities of writing checks to be split evenly among team members, both application and infrastructure developers. It should help ensure that your monitoring coverage is as broad as possible.

Sensu is an open source project that also has a commercial offering; the open source product is referred to as Sensu Core, while the commercial version is Sensu Enterprise. Sensu Enterprise offers a better issue routing system, tweaked dashboard and, perhaps most compellingly, support. However, Sensu Core is a powerful product in its own right, and can happily monitor huge amounts of clients.

This chapter will show you how to set up, configure, and roll out checks using Sensu. We will also look at how to configure Sensu to alert you via the two most important avenues: SMS and e-mail.

Installing a Sensu server

Setting up Sensu and its pre-requisites is reasonably straightforward, and it should take little time to set it up and configure some initial checks. This recipe will show you how to install Sensu Core and Uchiwa as its dashboard. Thus, giving you both a powerful and scalable Sensu server; however, it only gives you a single place to examine any issues. Using Uchiwa allows you to have multiple Sensu servers across many sites and still have a singular place to examine them in a graphical fashion.

Lastly, we will also set up SSL keys to ensure that the communication between the Sensu queue and its subscribing hosts remains confidential.

Getting ready

For this recipe, you require an Ubuntu 14.04 server to host both Sensu Core and Uchiwa.

How to do it...

The following steps will demonstrate how to install a Sensu server, RabbitMQ, and the Uchiwa panel:

1. Our first step is to install RabbitMQ, which requires Erlang to be installed. Use the following command to install Erlang:

   ```
   $ sudo apt-get update && sudo apt-get -y install erlang-nox
   ```

2. Now that we have Erlang, we are ready to install RabbitMQ. The Sensu project recommends the use of the latest upstream release of Rabbit rather than the version available for distribution. To do this, we need to first download and install the RabbitMQ package signing key:

   ```
   $ sudo wget http://www.rabbitmq.com/rabbitmq-signing-key-public.
   asc && sudo apt-key add rabbitmq-signing-key-public.asc
   ```

3. Next, we need to make the `apt` repository available and install the RabbitMQ package with the following two commands:

   ```
   $ echo "deb     http://www.rabbitmq.com/debian/ testing main" |
   sudo tee /etc/apt/sources.list.d/rabbitmq.list
   $ sudo apt-get update && sudo apt-get install rabbitmq-server
   ```

4. Now we will create the keys to secure our Sensu communication, but first we need to install the OpenSSL tools with the following command:

   ```
   $ sudo apt-get install openssl
   ```

5. Once OpenSSL is installed, we can download a tool provided by the Sensu team to generate the certificates. The script uses the OpenSSL tools to generate a set of certificates and place them into the CA, server and client directories. First we download the script using the following command:

   ```
   $ wget http://sensuapp.org/docs/0.20/files/sensu_ssl_tool.tar &&
   tar -xvf sensu_ssl_tool.tar
   ```

6. Once downloaded, we run the script and generate the certificates using the following command:

   ```
   $ sudo./sensu_ssl_tools/ssl_certs.sh generate
   ```

7. This should produce new directories containing the Certificate Authority, client certificates, and Server certificate and key. Next, we both create the directory structure, and copy the server and Certificate Authority certificates using this command:

   ```
   $ sudo mkdir -p /etc/rabbitmq/ssl && sudo cp sensu_ca/cacert.pem
   server/cert.pem server/key.pem /etc/rabbitmq/ssl
   ```

8. Finally, we need to amend our RabbitMQ configuration to make use of the certificates. Create a new file called `rabbitmq.config` within `/etc/rabbitmq/` and insert the following content:

   ```
   [
       {rabbit, [
       {ssl_listeners, [5671]},
       {ssl_options, [{cacertfile,"/etc/rabbitmq/ssl/cacert.pem"},
                      {certfile,"/etc/rabbitmq/ssl/cert.pem"},
                      {keyfile,"/etc/rabbitmq/ssl/key.pem"},
                      {verify,verify_peer},
                      {fail_if_no_peer_cert,true}]}
     ]}
   ].
   ```

9. This directs RabbitMQ to start a new listener on TCP port `5671` using the certificates we have generated to secure communication on this port. Now that we have configured the certificates, you should restart the RabbitMQ server using the following command:

   ```
   $ sudo /etc/init.d/rabbitmq-server restart
   ```

10. With RabbitMQ installed and configured, we can start creating the RabbitMQ users and virtual hosts for Sensu to be used as a publishing endpoint. Let's start by creating a new RabbitMQ virtual host for Sensu with the following command:

    ```
    $ sudo rabbitmqctl add_vhost /sensu
    ```

11. Next, create a RabbitMQ user with a password of your choice with the following command:

    ```
    $ sudo rabbitmqctl add_user sensu <<YOURPASSWORD>>
    ```

    Remember to replace `<<YOURPASSWORD>>` with the password of your choice.

12. Finally, we give the `sensu-user` full permission over its virtual host with this command:

    ```
    $ sudo rabbitmqctl set_permissions -p /sensu sensu ".*" ".*" ".*"
    ```

13. We're finished with RabbitMQ and now we are ready to move onto the next prerequisite, which is Redis. In this case, the version included with the Linux distribution is fine. You can install Redis using the following command:

    ```
    $ sudo apt-get install redis-server
    ```

14. Now that we have installed and configured RabbitMQ and Redis, we can install Sensu Core. First, we add the Sensu package signing key using the following command:

    ```
    $ wget -q http://repos.sensuapp.org/apt/pubkey.gpg -O- | sudo apt-
    key add -
    ```

15. We can then add the repository with this command:

    ```
    $ echo "deb     http://repos.sensuapp.org/apt sensu main" | sudo
    tee /etc/apt/sources.list.d/sensu.list
    ```

16. Finally, we install the Sensu package. This installs the server, API, and client as a bundle:

    ```
    $ sudo apt-get update && sudo apt-get install sensu
    ```

17. Now we have installed the Sensu server, we can configure it. First, copy the client certificate we created earlier in place with the following command:

    ```
    $ sudo mkdir -p /etc/sensu/ssl && sudo cp client/cert.pem client/
    key.pem /etc/sensu/ssl
    ```

18. Next, we configure our Sensu server's basic connectivity. Create a new file called `config.json` under the directory of `/etc/sensu` and insert the following content:

    ```
    {
      "rabbitmq": {
        "ssl": {
          "cert_chain_file": "/etc/sensu/ssl/cert.pem",
          "private_key_file": "/etc/sensu/ssl/key.pem"
        },
        "host": "localhost",
        "port": 5671,
    ```

```
    "vhost": "/sensu",
    "user": "sensu",
    "password": "<<password>>"
  },
  "redis": {
    "host": "localhost"
  },
  "api": {
    "port": 4567
  }
}
```

Ensure that you replace `<<password>>` with the password you chose while setting up the Sensu RabbitMQ user.

19. Start the Sensu services using the following commands:

```
$ sudo service sensu-server start
$ sudo service sensu-api start
```

Notice that we're not starting the Sensu client in this instance; this is to keep this recipe focused. I thoroughly encourage you to configure a client on your Sensu master and monitor it as with any other host.

20. Next, we are going to install `uchiwa`. Uchiwa is an elegant and easy to use dashboard for Sensu, designed to be used for information radiators, such as TVs. Install Uchiwa using the following command:

```
$ sudo apt-get install uchiwa
```

21. Now that we have installed Uchiwa, we need to configure it; edit the file `/etc/sensu/uchiwa.json` and ensure that it has the following content:

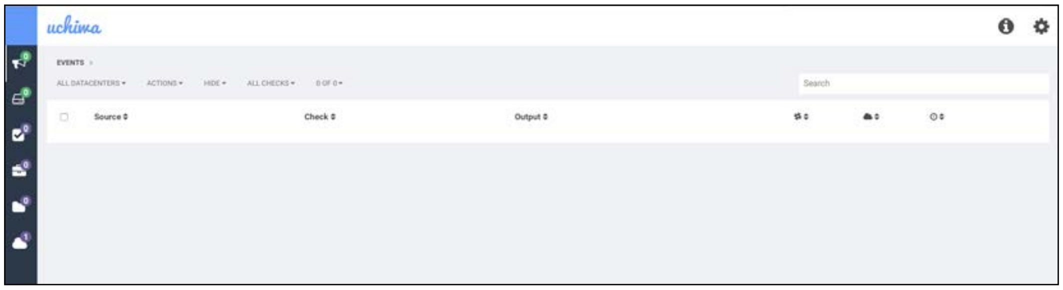
```
{
  "sensu": [
    {
      "name": "<<site description>>",
      "host": "<<sensuserver>>",
      "port": 4567,
      "timeout": 5
    }
  ],
    "uchiwa": {
      "host": "0.0.0.0",
      "port": 3000,
      "interval": 5
    }
  }
```

Ensure that you replace `<<site description>>` and `<<sensuserver>>` with the correct values. The site description can be set to a description of your choice; I tend to use it to delineate geographical sites (for instance, London, DC). Ensure that the `sensu` server is set to the DNS name or IP address of your Sensu server.

22. Start the service using the following command:

```
$ sudo service uchiwa start
```

Within your browser, navigate to your Sensu server on port `3000` and you should be able to see a screen similar to the following screenshot:



See also

- You can find further installation details for Sensu at `https://sensuapp.org/docs/0.20/installation-overview`
- You can find installation details of Uchiwa at `http://docs.uchiwa.io/en/latest/getting-started/`

Installing a Sensu client

Once you have installed the Sensu server, you need to install the Sensu client to run checks and report the data back to the server. The Sensu client subscribes to the RabbitMQ virtual host and listens for checks to be published in a subscription to which the client belongs. When a check is published, the client runs the assigned check and publishes the results back onto the RabbitMQ; from here, the Sensu server then processes the check results.

Getting ready

For this recipe, you will need an Ubuntu 14.04 client to act as the Sensu client and a Sensu server to connect to.

How to do it...

The following steps show you how to install the Sensu Core package and how to configure the Sensu client.

1. The Sensu Core package used to install the Sensu client includes the client, server, and API package. First we add the Sensu package signing key using the following command:

   ```
   $ wget -q http://repos.sensuapp.org/apt/pubkey.gpg -O- | sudo apt-key add -
   ```

   Then we add the repository:

   ```
   $ echo "deb     http://repos.sensuapp.org/apt sensu main" | sudo tee /etc/apt/sources.list.d/sensu.list
   ```

2. Finally, we can now install the Sensu package using the following command:

   ```
   $ sudo apt-get update && sudo apt-get install sensu
   ```

> If you have already read the *How to install a Sensu server* recipe, then you might have noticed that the steps are the same for both client and server. This is because the Sensu package is an omnibus package that contains everything; you only start the services you want.

3. Next, we need to copy the client key into place. You can start by creating a directory to place the certificates with this command:

   ```
   $ sudo mkdir -p /etc/sensu/ssl
   ```

4. Copy the following files from the keys you created when you created your Sensu server and into the directory you created in the step above:

   client/cert.pem

   client/key.pem

5. Now, we can configure the Sensu client. First, create a file in `/etc/sensu` called `config.json` and insert the following content:

   ```
   {
     "rabbitmq": {
       "ssl": {
         "cert_chain_file": "/etc/sensu/ssl/cert.pem",
         "private_key_file": "/etc/sensu/ssl/key.pem"
       },
       "host": "<sensumaster>",
       "port": 5671,
       "vhost": "/sensu",
       "user": "sensu",
       "password": "<password>"
     },
     "redis": {
       "host": "localhost"
     },
     "api": {
       "port": 4567
     }
   }
   ```

   Note that you need to replace the values of `<<sensumaster>>` and `<<password>>` with the IP or name of your Sensu Master and Sensu password, respectively.

6. Now that we have configured the general Sensu connectivity, we can configure the client specific settings. Create a new file under `/etc/sensu/conf.d` called `client.json` and insert the following content:

   ```
   {
     "client": {
       "name": "sensuhost",
       "address": "<ip address>,
       "subscriptions": [ "common" ]
     }
   }
   ```

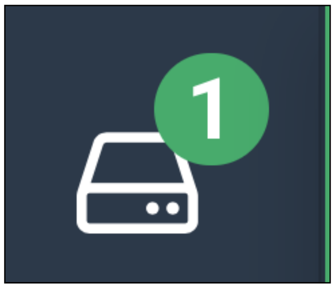
It's worth going over this slim piece of configuration. The first part of the configuration defines the name that is displayed for this host when reporting the check results; I suggest that this should be the DNS name of the client for easy identification. The `address` field allows you to define an IP address that the client reports as its originating address. The subscriptions field allows you to add subscriptions for checks. I recommend that you have a common set of checks that all hosts should respond to; these can be things, such as disk space, CPU usage, RAM usage, and so on.

Subscriptions are a key part of using Sensu, and are a fantastic organizational tool. I generally recommend using a common subscription for the usual suspects, such as RAM and CPU checks. You can use a role description for other subscriptions, such as a subscription called `nginx_server`, or `haproxy_lb`.

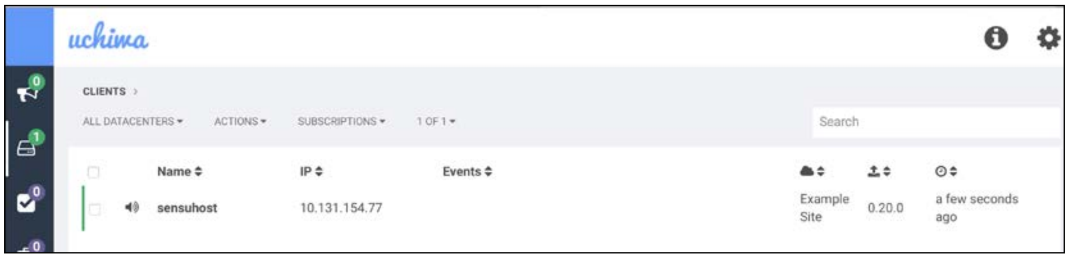
7. Now that we have configured our client, we are ready to start the Sensu client service. Start it by issuing the following command:

```
$ service sensu-client start
```

8. On your `sensu` master, log into your `uchiwa` panel and select this icon:



This takes you to the client-listing page. Once there, you should be able to see your new host listed and it should look something similar to the following screenshot:



	Name	IP	Events			
	sensuhost	10.131.154.77		Example Site	0.20.0	a few seconds ago

See also

You can find further details about the client installation at `https://sensuapp.org/docs/0.20/install-sensu-client`.

Installing check prerequisites

Sensu checks are generally written using Ruby and there is broad support for the language throughout Sensu. However, this means that there are certain dependencies on Ruby for some checks.

Getting ready

For this recipe, you will require an Ubuntu 14.04 server with Sensu installed.

> This recipe requires you to install development tools to compile the native Ruby extensions that some checks require. If having development tools on hosts contravenes your security policies, I recommend that you use a tool, such as FPM (`https://github.com/jordansissel/fpm`) to build the checks on a build machine and then re-package for distribution.

How to do it…

To install the various packages, issue the following command:

```
$ sudo apt-get install -y ruby ruby-dev build-essential
```

Finding community checks

Once the Sensu client and server are installed, it's now time to add the checks to be monitored for any issue. By default, the Sensu client reports nothing; it is up to you to add any relevant checks to make it useful.

Sensu checks can be written in any language as long as it returns the correct response to the server via RabbitMQ; however, they are generally written either in Bash or more commonly in Ruby. Luckily, the Sensu community has contributed a great many open source checks to the project. These can be installed, thus saving you from having to create your own and they cover many of the common check scenarios.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host to act as the Sensu check host and a Sensu server to connect to. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

You can find the Sensu community checks at `http://sensu-plugins.io/plugins/`.

Each link should take you to a Github page containing the code for that particular check, and it should have the additional documentation on the usage of the check.

See also

- You can find further details of the Sensu community checks at `http://sensu-plugins.io/plugins/`
- You can find details of the Sensu check format at `https://sensuapp.org/docs/latest/checks`

Adding a DNS check

Almost every application has external dependencies, such as databases, Redis caches, e-mail servers, and so on. Although it is generally reliable, DNS can occasionally cause problems and, at first glance, it can be difficult to diagnose. By adding a check that constantly checks the DNS record, you can be assured that these dependencies are available.

Getting ready

For this recipe, you will need an Ubuntu 14.04 host to act as the Sensu check-host and a Sensu server to connect to. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's start by adding DNS check to track DNS records:

1. For this recipe, we're going to install the `sensu-plugins-dns`. Use the following command to install the new plugin:

   ```
   $ sudo gem install sensu-plugins-dns
   ```

2. You can test if the plugin has been installed successfully by issuing the following command:

```
$ check-dns.rb -d www.packtpub.com
```

3. You should see a response like the following:

```
DNS OK: Resolved www.packtpub.com A records
```

4. On the Sensu server, create a new file called `web_check.json` under the directory `/etc/sensu/conf.d` and insert the following content:

```
{
  "checks": {
    "check_google": {
      "command": "/usr/local/bin/check-dns.rb -d google.com",
      "interval": 60,
      "subscribers": [ "web_check" ]
    },
    "check_yahoo": {
      "command": "/usr/local/bin/check-dns.rb -d yahoo.com",
      "interval": 60,
      "subscribers": [ "web_check" ]
    },
    "check_fail": {
      "command": "/usr/local/bin/check-dns.rb -d sdfdsssf.com",
      "interval": 60,
      "subscribers": [ "web_check" ]
    }
  }
}
```

5. Once you have entered the configuration, restart the Sensu server by issuing the following command:

```
$ service sensu-server restart
```

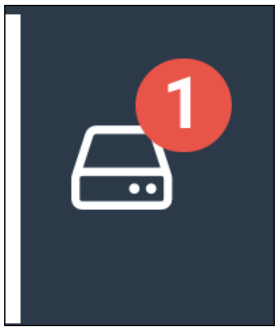
6. Next, we need to configure our client to subscribe to the checks. Edit the `client.json` file located within `/etc/sensu/conf.d` and to reflect the following code:

```
{
  "client": {
    "name": "sensuhost",
    "address": "<<SENSUCLIENTIP>>",
    "subscriptions": [ "common","web_check" ]
  }
}
```

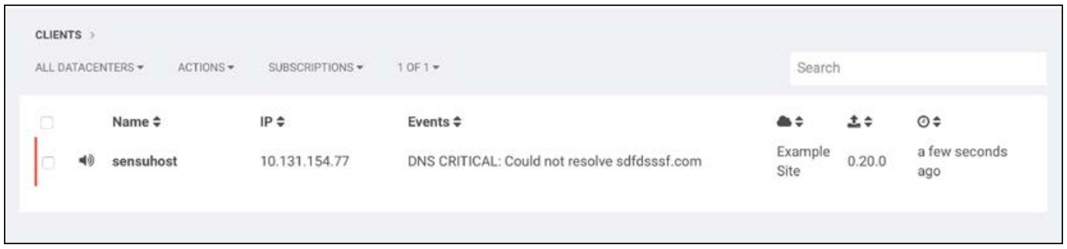
7. Notice the additional subscription. When you restart the client, it will subscribe to a set of checks that publish themselves for `web_check` clients to run. Restart the Sensu client by issuing the following command:

```
$ service sensu-client restart
```

8. Your checks should now be running; however, it may take a few minutes for them to show up. This is due to the checks being placed on the MQ and being checked at the interval specified (60 seconds in the above example). To check, log on to Uchiwa on your Sensu master and select the following icon on the left-hand side:



You should see the check to which we have deliberately given a nonsense address:



Name	IP	Events			
sensuhost	10.131.154.77	DNS CRITICAL: Could not resolve sdfdsssf.com	Example Site	0.20.0	a few seconds ago

9. You can also check in `/var/log/sensu/sensu-server.log` and locate the line that resembles the following:

```
{"timestamp":"2015-07-14T16:55:14.404660-0400","level":"info",
"message":"processing event","event":{"id":"c12ebe42-62ed-454f-
a074-49c71c3c8f7a","client":{"name":"sensuhost","address":"10.1
31.154.77","subscriptions":["common","web_check"],"version":"0
.20.0","timestamp":1436907305},"check":{"command":"/usr/local/
bin/check-dns.rb -d sdfdsssf.com","interval":60,"subscribers":["w
eb_check"],"name":"check_fail","issued":1436907314,"executed":1436
907314,"duration":0.262,"output":"DNS CRITICAL: Could not resolve
sdfdsssf.com\n","status":2,"history":["2"],"total_state_change":0}
,"occurrences":1,"action":"create"}}
```

See also

You can find further details of the Sensu DNS check at `https://github.com/sensu-plugins/sensu-plugins-dns`.

Adding a disk check

Disk checks are a critical part of infrastructure monitoring. A full disk can cause processes to fail, servers to slow down, and logs to be lost. Providing alerts for disk issues in a timely manner is vital to the smooth running of any infrastructure.

This recipe shows you how to install the community disk check and add it to a subscription called **common**.

Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's install the community disk check:

1. First, we install the disk check `gem` using the Gem package manager:

   ```
   $ sudo gem install sensu-plugins-disk-checks
   ```

> This gem installs many additional disk based checks in addition to space usage, allowing you to check for issues such as SMART alerts, and so on. You can see the details at: `https://github.com/sensu-plugins/sensu-plugins-disk-checks`.

2. Now we can configure the check configuration. On the Sensu master, create a new file called `disk_checks.json` under the `/etc/sensu/conf.d` directory and insert the following content:

   ```
   {
     "checks": {
       "check_disk_usage": {
         "command": "/usr/local/bin/check-disk-usage.rb -w 75 -c 90",
         "interval": 60,
         "subscribers": [ "common" ]
     }
   }
   ```

The configuration above makes use of the disk check plugging, and uses the `-w` and `-c` switches. These switches are relatively common amongst Nagios-style checks and allows you to set a warning and a critical threshold. In this case, I'm using a warning at 75% and a critical alert at 90%. This is very useful as it allows us to use different alert types based on the threshold; for instance, a warning could trigger an e-mail and a critical alert could send an SMS. Read the plugin documentation to fid details of what thresholds you can set and how to set them.

3. On the client side, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

```
{
  "client": {
    "name": "sensuhost",
    "address": "10.131.154.77",
    "subscriptions": [ "common","web_check" ]
  }
}
```

4. To check that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

```
{"timestamp":"2015-07-14T17:26:42.689883-0400","level":"info","mes
sage":"publishing check request","payload":{"name":"check_disk_us
age","issued":1436909202,"command":"/usr/local/bin/check-disk-
usage.rb -w 75"},"subscribers":["common"]}
```

See also

You can find more details for the community disk checks at `https://github.com/sensu-plugins/sensu-plugins-disk-checks`.

Adding a RAM check

Having sufficient RAM available for a server is a crucial part of running a performant service. When memory resources run short, the application either runs slow, if the OS is forced to use swap space, or in extremes can cause applications to crash.

This recipe demonstrates how to use Sensu to monitor that sufficient free RAM is present on a monitored system.

Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's add the RAM check:

1. We need to install the `sensu-plugins-memory-checks gem`; this installs an executable for the RAM check:

   ```
   $ sudo gem install sensu-plugins-memory-checks
   ```

2. Now we can configure the `config` check. On the Sensu master, create a new file called `ram_checks.json` under the `/etc/sensu/conf.d` directory and insert the following content:

   ```
   {
     "checks": {
       "check_ram": {
         "command": "/usr/local/bin/check-ram.rb -w 70 -c 95",
         "interval": 60,
         "subscribers": [ "common" ]
       }
     }
   }
   ```

   Again, note the use of the -w and -c switches; these set the thresholds in percentage used that needs to trigger an alert.

3. On the client, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

   ```
   {
     "client": {
       "name": "sensuhost",
       "address": "10.131.154.77",
       "subscriptions": [ "common","web_check" ]
     }
   }
   ```

4. To determine that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

```
{"timestamp":"2015-07-14T17:43:28.066609-0400","level":"wa
rn","message":"config file applied changes","file":"/etc/
sensu/conf.d/check_ram.json","changes":{"checks":{"check_ram_
usage":[null,{"command":"/usr/local/bin/check-ram.rb","interval":6
0,"subscribers":["common"]}]}}}
```

See also

For further details on the Sensu memory checks, see the following page `https://github.com/sensu-plugins/sensu-plugins-memory-checks`.

Adding a process check

One important item to monitor is if a process is actually running on a system. It's little use knowing that you have plenty of disk and CPU resources, but not realizing that your apache server has fallen over. Sensu can be used to monitor the key processes that are running on your server and it can alert you if a process has gone AWOL.

This recipe shows you how to check if the `sshd` process is running on any host subscribed to the common subscriptions; however, the same technique can be used to monitor any process.

Getting ready...

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

This recipe will show you how to install the Sensu process check plugin and how to configure it to monitor a running process.

1. First, we install the process check `gem` using the Gem package manager. Use the the following command to install the plugin:

```
$ sudo gem install sensu-plugins-process-checks
```

2. Now, we can configure the check configuration. On the Sensu master, create a new file called `sshd_process_check.json` under the `/etc/sensu/conf.d` directory and insert the following content:

   ```
   {
     "checks": {
       "check_sshd_usage": {
         "command": "/usr/local/bin/check-process.rb -p 'sshd -D'",
         "interval": 60,
         "subscribers": [ "common" ]
       }
     }
   }
   ```

   This check makes use of the `-p` switch to allow us to specify a process that we wish to monitor; this should be the full string of the running process (notice that, in the preceding example, I have added the `-D` switch that the process runs with).

3. On the client, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

   ```
   {
     "client": {
       "name": "sensuhost",
       "address": "10.131.154.77",
       "subscriptions": [ "common","web_check" ]
     }
   }
   ```

4. To determine if the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

   ```
   {"timestamp":"2015-07-14T18:13:48.464091-0400","level":"info","me
   ssage":"processing event","event":{"id":"f1326a4f-87c2-49a7-8b28-
   70dfa3e9836b","client":{"name":"sensuhost","address":"10.131.154
   .77","subscriptions":["common","web_check"],"version":"0.20.0","
   timestamp":1436912025},"check":{"command":"/usr/local/bin/check-
   process.rb -p 'sshd -D'","interval":60,"subscribers":["common"],
   "name":"check_sshd_usage","issued":1436912028,"executed":1436912
   028,"duration":0.128,"output":"CheckProcess OK: Found 1 matching
   processes; cmd /sshd -D/\n","status":0,"history":["1","1","1","1",
   "0"],"total_state_change":0},"occurrences":4,"action":"resolve"}}
   ```

See also

You can find further details of the process checks at `https://github.com/sensu-plugins/sensu-plugins-process-checks`.

Adding a CPU check

Having sufficient CPU resources is a vital part of running a performant service and it is hard to spot without sufficient monitoring. Using Sensu to alert when CPU usage is running high, you will be able to deal with slow running processes before the customer notices.

Getting ready

For this recipe, you will need both a Sensu server and at least one Sensu host. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's add a CPU usage check:

1. First, we install the CPU check gem using the Gem package manager. Use the following command to install the plugin:

   ```
   $ sudo gem install sensu-plugins-cpu-checks
   ```

2. Now we can configure the check configuration. On the Sensu master, create a new file called `cpu_check.json` under the `/etc/sensu/conf.d` directory and insert the following content:

   ```
   {
     "checks": {
       "check_cpu_usage": {
         "command": "/usr/local/bin/check-cpu.rb",
         "interval": 60,
         "subscribers": [ "common" ]
       }
     }
   }
   ```

3. On the client, edit the file called `client.json` within `/etc/sensu/conf.d` and ensure that the following code is present:

   ```
   {
     "client": {
       "name": "sensuhost",
       "address": "10.131.154.77",
       "subscriptions": [ "common","web_check" ]
     }
   }
   ```

4. To determine that the check is running correctly, look in `/var/log/sensu/sensu-server` and check that a line resembling the following is present:

   ```
   {"timestamp":"2015-07-15T16:24:26.800371-0400","level":"info"
   ,"message":"publishing check request","payload":{"name":"che
   ck_cpu","issued":1436991866,"command":"/usr/local/bin/check-cp.
   rb"},"subscribers":["common"]}
   ```

See also

You can find further details of the process checks at `https://github.com/sensu-plugins/sensu-plugins-cpu-checks`.

Creating e-mail alerts

Although you can view your Sensu alerts using the Uchiwa panel, it's unlikely that you will have your eyes glued to the TV at all times. Instead, you need to give Sensu the ability to alert you in a more interactive fashion, and one of the most tried and trusted methods is via e-mail. In today's world of laptops, Smartphones and tablets, it's a rare time indeed when you are not able to receive e-mails.

This recipe will show you how to configure the Sensu e-mail plugin to allow you to receive e-mails whenever an alert is triggered.

Getting ready

For this recipe, you will need a configured Sensu server and Sensu client. You should also have at least one check configured. You will also need an SMTP server that can relay mail. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's create an e-mail alert:

1. First, you can use Ruby's gem to install the mail plugin using the following command:

   ```
   $ gem install sensu-plugins-mailer
   ```

2. Now, we can configure the mail plugin. Create a new file called `plugin_mailer.json` within `/etc/sensu/conf.d` and insert the following content:

   ```
   {
     "mailer": {
           "admin_gui": "http://<sensuserver>/#/events",
           "mail_from": "<<fromaddress>>",
           "mail_to": "<<toaddress>>",
           "smtp_address": "<<smtpaddress>>",
           "smtp_username": "<<smtpusername>>",
           "smtp_password": "<<smtppassword>>",
           "smtp_port": "587",
           "smtp_domain": "<<smtpdomain>>"
     }
   }
   ```

   Ensure that you replace the values inside the angle brackets with the relevant information for your e-mail setup. The admin `gui` is simply a link to the `uchiwa` panel, so fill in the address of your Sensu server.

3. Now the mailer plugin is configured, we can create the handler.
4. You can combine the plugin and handler settings into the same file, but it's better practice to keep them separate.
5. A handler is an executable piece of code that is triggered by an event sent via a Plugin; you can think of plugins as raising alerts and handlers as dealing with how to distribute the event to end users. Sensu allows you to configure many different handlers, which allows you to be flexible in how you are alerted. You may wish to e-mail some checks, others you might want to send via SMS, and still others you might want to allow for a combination of the two; handler definitions allow you to define these. To create the `handler` definition for the mailer, create a new file called `mail.json` under the `/etc/sensu/handlers` and insert the following content:

   ```
   {
     "handlers": {
       "mailer": {
         "type": "pipe",
         "command": "/usr/local/bin/handler-mailer.rb"
       }
     }
   }
   ```

6. This has created a new handler called **mailer** that we can make available for our checks. The type of `pipe` is the most commonly used type of handler and outputs the contents of the Sensu event into the command. Effectively, the event is raised by a plugin, placed on the MQ, processed by the Sensu Server, and then parsed via the handler.

7. To add the handler to a check, open up a check definition and amend it to include the following code:

```
{
  "checks": {
    "check_cpu": {
      "command": "/usr/local/bin/check-cpu.rb",
      "interval": 60,
      "subscribers": [ "common" ],
      "handlers": ["mailer"]
    }
  }
}
```

8. Now, whenever an alert is triggered, you should receive an e-mail that resembles something like this:

```
DNS CRITICAL: Could not resolve sdfdsssf.com
Admin GUI: http://sensumaster.stunthmaster.com/#/events
Host: sensuhost
Timestamp: 2015-07-15 19:07:14 -0400
Address:  10.131.154.77
Check Name:  check_fail
Command:  /usr/local/bin/check-dns.rb -d sdfdsssf.com
Status:  CRITICAL
Occurrences:  1
And when the check is resolved, you should see a resolution E-mail
that looks something like this:
Resolving on request of the API
Admin GUI: http://sensumaster.stunthmaster.com/#/events
Host: sensuhost
Timestamp: 2015-07-19 19:15:12 -0400
Address:  10.131.154.77
Check Name:  check_fail
Command:  /usr/local/bin/check-dns.rb -d sdfdsssf.com
Status:  OK
Occurrences:  2976
```

By editing the `handler-mailer.rb` code, you can modify this e-mail to more suit your formatting needs.

See also

- You can find more details of the Sensu handlers at `https://sensuapp.org/docs/0.20/handlers`
- You can find more details of the e-mail handler at `https://github.com/sensu-plugins/sensu-plugins-mailer`

Creating SMS alerts

Sometimes you need alerts that are more immediate than an e-mail. When a critical service goes down, you don't want to miss it because you eschewed carrying a smartphone and your laptop wasn't near by.

SMS messaging is a fantastic way to send default alerts and is in many ways the spiritual successor to the pager. SMS has the advantage of being almost universal and it is virtually impossible in this day and age to find a cell phone that does not support it.

Unlike e-mail, you cannot run a local SMS server to send messages directly; instead, you need to sign up with an SMS gateway, which will route your messages to the various mobile phone providers. In this recipe, we're going to use Twilio (`https://www.twilio.com`). Twilio supports both Voice and SMS gateways and has an easy to use API. Like all SMS gateways, Twilio charges per message; however, trial accounts are available to test your integration.

Getting ready

For this recipe, you will need a Sensu server, Sensu client, and at least one configured check. You will also need a Mobile phone to receive your test message. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

How to do it...

Let's create SMS alerts:

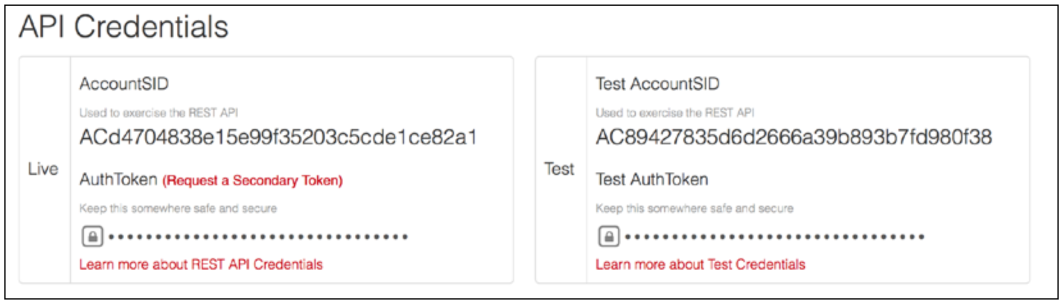
1. First, signup for a new Twilio account by visiting `https://www.twilio.com/try-twilio`. It will ask you for some basic details, and will send you an e-mail to confirm the account. Ensure that you have confirmed your details, and that you can log in.
2. Once you have a Twilio account, you can install the Twilio Sensu plugin using the following command:
   ```
   $ sudo gem install sensu-plugins-twilio
   ```
3. Next, we will configure the plugin. As with the mailer plugin, this takes two forms: the handler configuration and the plugin configuration. Let's deal with the plugin configuration first: create a new file `/etc/sensu/conf.d/plugin_twilio.json` and insert the following content:
   ```
   {
     "twiliosms":{
       "token":"<<TWILIOTOKEN>>",
       "sid":"<<TWILIOSID>>",
       "number":"<<TWILIONUMBER>>",
   ```

```
        "recipients":{
          "+<<RECIPIENTNUMBER>>": {
            "sensu_roles":["all"],
            "sensu_checks":[],
            "sensu_level": 1
          }
        }
      }
    }
```

4. There are a few things to note with this code. First, you need to have your own Twilio API and SID at hand; if you need to find them, you can find them on this page:

   `https://www.twilio.com/user/account/settings`

   They should be available about halfway down the page and will resemble this:



API Credentials

Live
AccountSID
Used to exercise the REST API
ACd4704838e15e99f35203c5cde1ce82a1
AuthToken (Request a Secondary Token)
Keep this somewhere safe and secure
••••••••••••••••••••••••••••••••
Learn more about REST API Credentials

Test
Test AccountSID
Used to exercise the REST API
AC89427835d6d2666a39b893b7fd980f38
Test AuthToken
Keep this somewhere safe and secure
••••••••••••••••••••••••••••••••
Learn more about Test Credentials

5. Next we need to set up the recipient number. This is an array and can contain as many recipients as you need; however, the recipient will need to be acknowledged within the Twilio panel.

A Twilio test account has many limitations, including a limit on the recipients.

6. Each recipient can have a different set of roles and checks that will trigger an SMS; in our example, we're leaving it as all roles to ensure that every alert will send an SMS. However, you can use this configuration item to restrict SMS alerts only to critical roles.

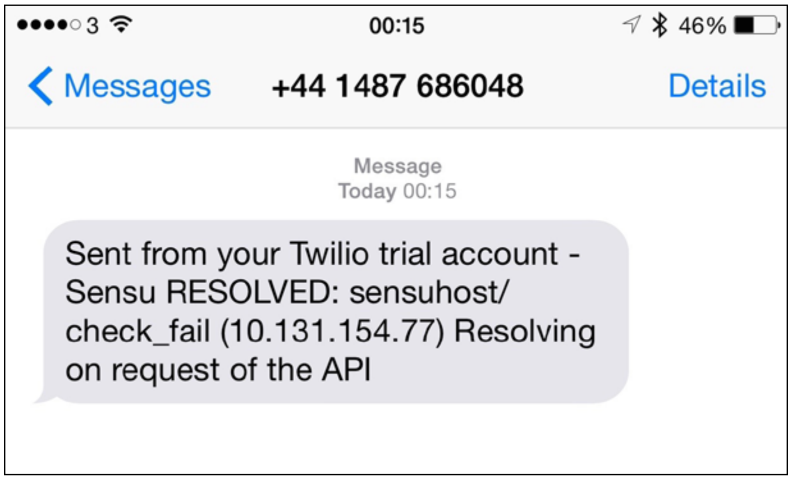
7. Next, we can configure the handler configuration. Create a new file called `/etc/sensu/handlers/plugin_twilio_sms.json` and add the following configuration:

```
{
  "handlers": {
    "twiliosms": {
      "type": "pipe",
      "command": "/var/lib/gems/1.9.1/gems/sensu-plugins-
twilio-0.0.3/bin/handler-twiliosms.rb"
    }
  }
}
```

8. Once you have done this, save the file and restart the Sensu server with the following command:

```
$ sudo service sensu-server restart
```

The next time you have an alert, you should receive an SMS message that looks similar to the following screenshot:



Sent from your Twilio trial account - Sensu RESOLVED: sensuhost/check_fail (10.131.154.77) Resolving on request of the API

See also

- You can find more information about Twilio at `https://www.twilio.com`
- You can find further information about the Twillio plugin at `https://github.com/sensu/sensu-community-plugins/blob/master/handlers/notification/twiliosms.rb`

Using Ansible to install Sensu

When rolling out on any kind of scale, it's almost certain that you will want to use automation to perform the install, especially for the clients; this allows you to roll out the changes quickly, easily, and with minimum fuss.

As with other recipes in this book, we are going to use Ansible as our automation tool of choice and rather than write a new playbook from scratch, we're going to make use of a truly excellent role available on the Ansible galaxy (`https://galaxy.ansible.com/detail#/role/279`).

If you need a refresher on Ansible, see *Chapter 5*, *Automation with Ansible*.

Getting ready

For this recipe, you will need a node to run the Ansible playbook and at least two servers: one to act as the Sensu server and the other, the Sensu client. The Sensu Server node should have RabbitMQ and Redis already installed on it. You should also have installed the prerequisite packages as detailed in the recipe *Installing check prerequisites*.

Although slightly out of scope for this recipe, you can use two other Ansible roles to automate both RabbitMQ and Redis, `https://github.com/Mayeu/ansible-playbook-rabbitmq` and `https://github.com/DavidWittman/ansible-redis`, respectively.

How to do it...

1. On the host that will act as your Ansible node, run the following command to install the Sensu role:

```
ansible-galaxy install Mayeu.sensu
```

If you do not have Ansible installed in the default location, you can also clone the role and place it in your own structure; the code is available here: `https://github.com/Mayeu/ansible-playbook-sensu/blob/master/vagrant/site.yml`.

2. Next, let's create an inventory. I'm assuming you are using the default location for the inventory; otherwise, use the `-i` switch on the `ansible-playbook` command to specify one in the location of your choice. In the inventory, ensure that you have the following:

```
[sensu_servers]
    <<SENSUSERVERS>

[sensu_clients]
    <<SENSUCLIENTS>>
```

   Where `<<SENSUSERVER>>` is the DNS name of your Sensu server and `<<SENSU_CLIENTS>>` are the DNS names of your Sensu clients.

3. Now, we need to create a new playbook; create a new file called `<<playbook>>/sensu.yml` and insert the following content:

```
- hosts: sensu_servers
  user: <<sudo_user>>
  vars:
    - sensu_install_server: true
    - sensu_install_client: false
  vars_files:
    - group_vars/sensu.yml
  roles:
    - Mayeu.sensu

- hosts: sensu_clients
  user: <<sudo_user>>
  vars:
    - sensu_install_server: false
    - sensu_install_client: true
    - sensu_client_hostname: '{{ ansible_hostname }}'
    - sensu_client_address: '{{ ansible_eth0["ipv4"]["address"]
    - sensu_client_subscription_names [common]
}}'
  vars_files:
    - group_vars/sensu.yml
  roles:
    - Mayeu.sensu
```

4. Replace `<<playbook>>` with the name of the directory you wish your playbook to reside in. Also replace the `<<sudo_user>>` with a user that has sudo permissions on the servers you are connecting to.

   As you can see, in this playbook we are defining two different plays: one for the Sensu Server and another for the Sensu clients. Note how each play references a variable file allowing us to define certain shared configuration items. We are also setting certain variables at the client level where we need differences.

5. Next, we need to create some directories to contain the files that the Ansible role will require. Use the following command to create them:

   ```
   mkdir -p <playbook>/files/sensu/extensions && mkdir -p <playbook>/
   files/sensu/handlers && mkdir -p <playbook>/files/sensu/plugins
   ```

   These folders are to hold your handlers, plugins, and if you use them, your extensions.

> You can find details of Sensu extensions at `https://sensuapp.org/docs/0.20/extensions`.

6. For instance, if you wish to add a new plugin, you will first add the plugin `ruby` file to the `<playbook>/files/sensu/plugins` directory; this will then be placed in the appropriate place on the Sensu server.
7. Next, create a folder to hold your Sensu certificates:

   ```
   mkdir -p <playbook>/files/sensu/certs
   ```

   Now, copy your certificates into the newly created folder.

> You can find the details for how to create the certificates in the recipe entitled *How to install a sensu server*.

8. Now, let's create the variables file for the common Sensu items. Create a new file under `<<playbook>>/group_vars` called `sensu.yml` and insert the following content:

   ```
   sensu_server_rabbitmq_hostname: '<<SENSUSERVERDNSNAME>>'
   sensu_server_rabbitmq_user: <<sensuMQuser>>
   sensu_server_rabbitmq_password: <<sensuMQpassword>>
   sensu_server_rabbitmq_vhost:    "sensu"
   sensu_server_api_user: <<SENSU_USER>>
   sensu_server_api_password: <<SENSU_PASSWORD>>
   ```

The values in this file define the settings for your Sensu server and cover aspects such as the MQ to connect to the vhost, username, password, and so on. You may notice that this seems to be a short list; this is because this particular role has very sensible defaults. You can find the list of defaults on the readme at `https://galaxy.ansible.com/detail#/role/279`.

9. Now we have configured our server and client, the next step is to define some checks to run on the client. First, we configure the server to send out a request for the check to the `common` subscription. Insert the following into the `<<playbook>>/group_vars/sensu.yml` file:

```
sensu_checks:
  cpu_check:
    handler: default
    command: "/usr/local/bin/check-cpu.rb"
    interval: 60
    subscribers:
      - common
```

10. You will also need to install the check-package onto the clients; you can do this by inserting the following code within the role that sets up the client. I normally recommend making this part of any common role that is used to setup all hosts:

```
name: "Install Sensu CPU Check Plugin"
gem: name='sensu-plugins-cpu-checks' state=present
```

11. Finally, we should define a handler on the Sensu server; insert the following code into the `<<playbook>>/group_vars/sensu.yml` file:

```
sensu_handlers:
  basic_mailer:
    type: pipe
    command: "mailx -s 'Sensu Alert' opsuser@opsaddress.com"
```

The preceding handler will send a simple e-mail if there is an alert, and you can use the same technique to set up as many handlers as you like.

12. Run Ansible using the following command:

```
ansible-playbook -K sensu.yml
```

This should install Sensu and Uchiwa, and configure the clients with a CPU check, plus add a simple handler.

See also

You can find details of the Sensu Ansible role at `https://github.com/Mayeu/ansible-playbook-sensu`.

11
IAAS with Amazon AWS

In this chapter, we are going to cover the following topics:

- Signing up for AWS
- Setting up IAM
- Creating your first security group
- Creating your first EC2 host
- Using Elastic Load Balancers
- Managing DNS with route53
- Using Ansible to create EC2 hosts

Introduction

The term cloud computing has diluted immensely over the years and the cloud label is being applied to any technology that interacts over the network. Originally though, it was used to describe what is now being termed as **Infrastructure-as-a-Service** (**IAAS**).

IAAS has revolutionized the approach of many companies by helping them build applications and infrastructure, and for many companies, has turned infrastructure into a utility rather than a massive internal cost center. This in turn has freed them to explore new platforms and systems without the requirement of building costly servers and data centers. At it's heart, IAAS platforms offer the ability to provision servers rapidly, but it may also offer additional features, such as load balancers, persistent storage, and other elements of a traditional data center offering; thus, allowing you to run a full-bodied infrastructure on a pay-as-you-go basis.

Amazon was one of the earliest IAAS vendors and it became the most popular very rapidly. Its features are possibly the most complete of all IAAS vendors, offering features, such as compute units, load balancers, big data tools, and orchestration facilities. Amazon boasts clients of every size, from one-man development companies, through to the giants of the industry, such as Netflix. However, there are alternative. For some, the sheer complexity of the Amazon platform can be an issue and there are many competitors available, ranging from compute focused companies, such as DigitalOcean to fully featured offerings, such as Rackspace.

There are disadvantages of using IAAS platforms and it is important that you design your architecture to suit the particular foibles of such an offering. First, IAAS platforms are heavily contended for RAM, CPU, Disk, and network, and performance is not guaranteed. Your applications should be able to balance the load across many nodes, rather than relying on single, powerful nodes to do the heavy lifting. There is also the issue of cost; generally speaking, costing large IAAS platforms can be a complex task, with elements, such as CPU and RAM being charged by the minute, disks being charged via usage, and networks being charged by the megabyte of transferred data (sometimes on differing rates depending on it being external, or inter-DC traffic). However, as long as you are aware of the shortcomings, the benefits of using an IAAS can be vast.

Signing up for AWS

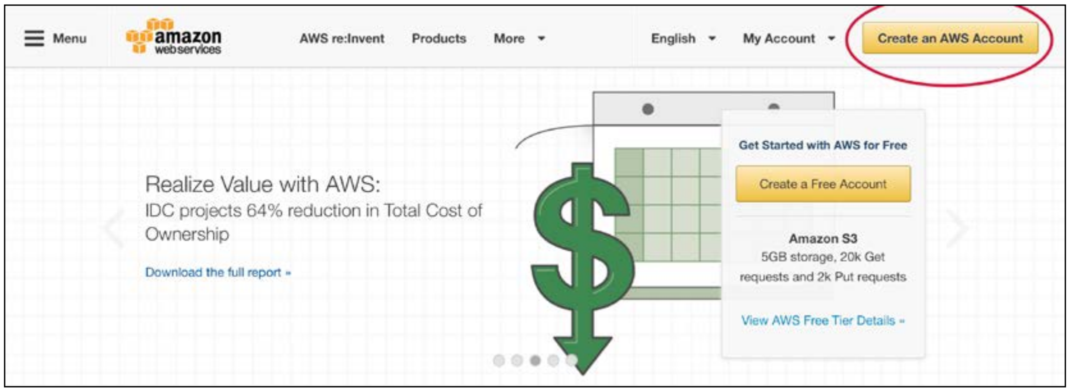
Before you can use any of the features of Amazon AWS, you need to register and set up your account to be ready for use. This recipe takes you through the steps required to set up an account.

Getting ready

For this recipe, you will require a web browser, an e-mail address to sign up with. You will also need a valid credit/debit card, but this will not be charged until you start using a billable compute.

How to do it...

Before we can approach any recipe in this chapter, we need to create a new AWS account. You can create an account by navigating to `https://aws.amazon.com` and the following the steps detailed in the signup process.


Menu
amazon web services
AWS re:Invent
Products
More
English
My Account
Create an AWS Account
Realize Value with AWS:
IDC projects 64% reduction in Total Cost of Ownership
Download the full report »
Get Started with AWS for Free
Create a Free Account
Amazon S3
5GB storage, 20k Get requests and 2k Put requests
View AWS Free Tier Details »

See also

- You can find the AWS getting started guide at `https://aws.amazon.com/documentation/gettingstarted/`
- You can see pricing information at `https://aws.amazon.com/ec2/pricing/`

Setting up IAM

Security is a critical part of managing any infrastructure regardless of its origin and a key part of security is having the ability to identify and limit access to the platform. Amazon provides a tool known as **Identity and Access Management** (**IAM**) that allows you to set up fine-grained users and access rights to access your AWS infrastructure. Using IAM, you can ensure that all the relevant users can access your AWS resources. It also ensures that users are only able to access and interact with the resources in the manner of your choice.

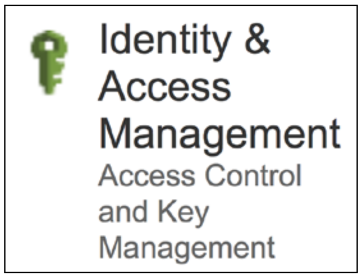
Getting ready

For this recipe, you will need an AWS account.

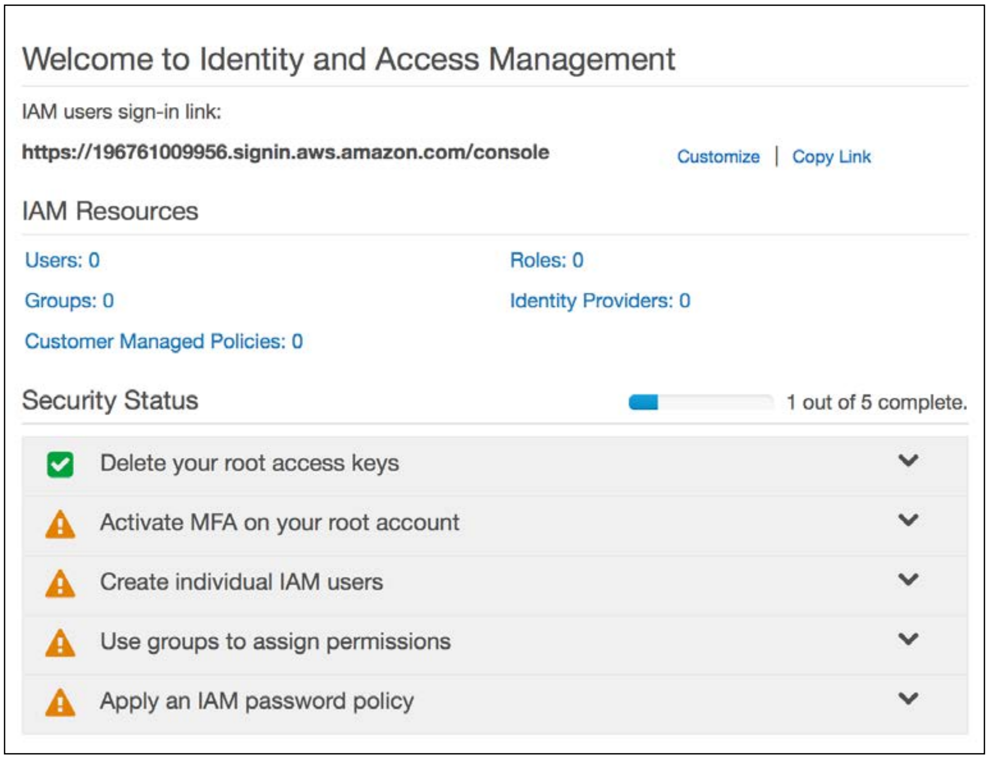
How to do it...

The following steps will illustrate how to use the AWS IAM system to create a new user and assign suitable roles:

1. Log in to your AWS account and select the IAM panel by selecting the following icon:


Identity & Access Management
Access Control and Key Management

2. You will be greeted by a page that gives you a summary of your users. It will also give a recommendation checklist for a new account and it should look similar to the following screen shot:



Welcome to Identity and Access Management

IAM users sign-in link:

https://196761009956.signin.aws.amazon.com/console

IAM Resources

Users: 0
Roles: 0
Groups: 0
Identity Providers: 0
Customer Managed Policies: 0

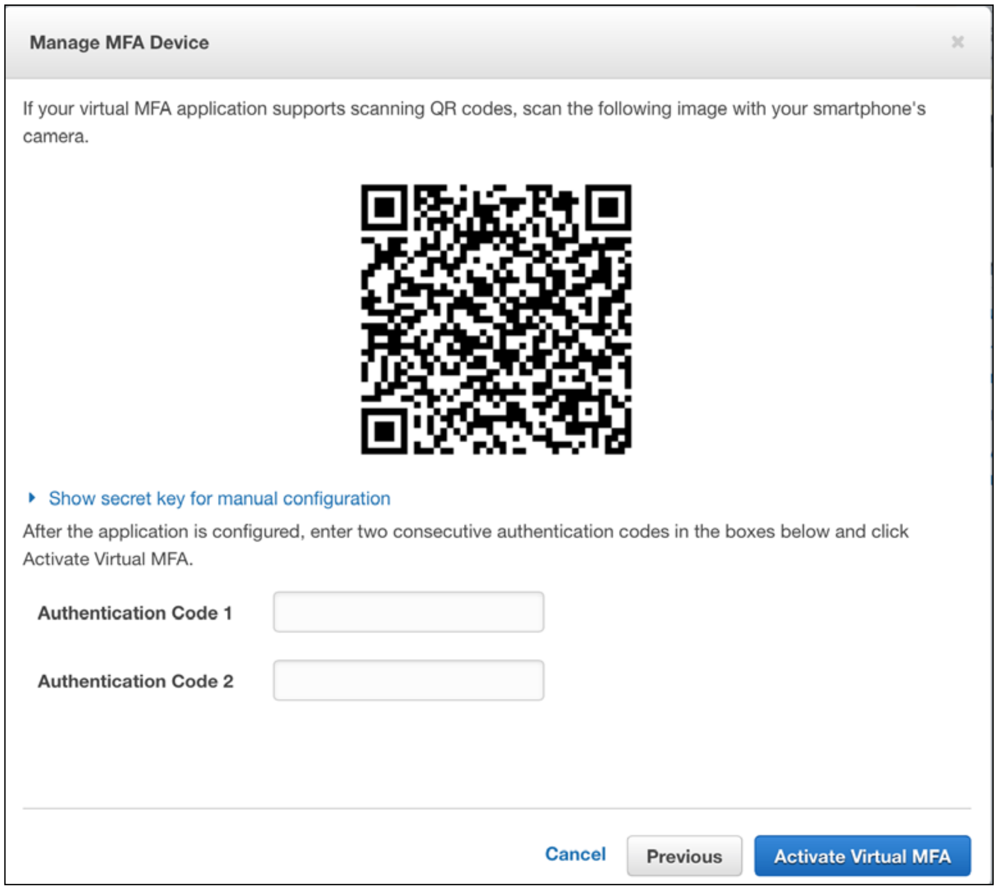
Security Status — 1 out of 5 complete.

- ✓ Delete your root access keys
- ⚠ Activate MFA on your root account
- ⚠ Create individual IAM users
- ⚠ Use groups to assign permissions
- ⚠ Apply an IAM password policy

Click on the drop-down menu beside **Activate MFA on your root account** and select **Manage MFA**.

> **MFA** stands for **Multi Factor Authentication** and ensures that in addition to a password, you also need a device that will give you a one-time code to enter alongside it. MFA is a crucial part of keeping your AWS account secure and I highly recommend enabling it.

3. The next screen will prompt you to select a type of MFA token, either a virtual or a hardware token; both offer the same functionality but differ in form factor. If you have a smartphone, then you will be able to use one of many applications, such as Google Authenticator that offers MFA tokens; select the virtual option if you wish to use these.
4. If you select the virtual MFA option, then you should be presented with a barcode that will allow you to register your MFA device and it will resemble the following screenshot:



If your virtual MFA application supports scanning QR codes, scan the following image with your smartphone's camera.

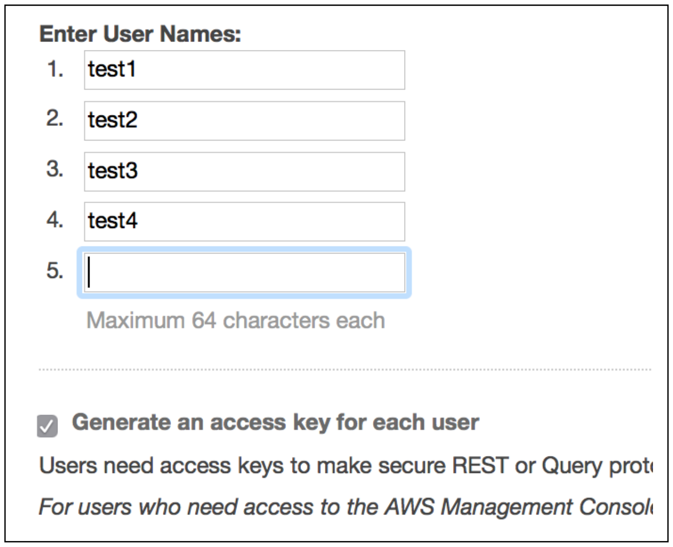
After the application is configured, enter two consecutive authentication codes in the boxes below and click Activate Virtual MFA.

Authentication Code 1	
Authentication Code 2	

In your smartphone app, you should be able to point your camera at the screen and it will register the account and offer you the MFA tokens. Enter the first two codes when prompted and select **Activate Virtual MFA**.

5. Now that you have secured the root AWS account, it is good to create individual users to login. This gives the ability to audit and lock down users from certain activities. You can start by returning to the IAM panel and selecting **Create individual IAM users** and then **Manage users**.

   You will be presented with a list of your current users. Select **Create New Users**. In the next screen, you should be able to enter a list of usernames that you want to create:



Enter User Names:

1.	test1
2.	test2
3.	test3
4.	test4
5.	

Maximum 64 characters each

✓ **Generate an access key for each user**

Users need access keys to make secure REST or Query prot

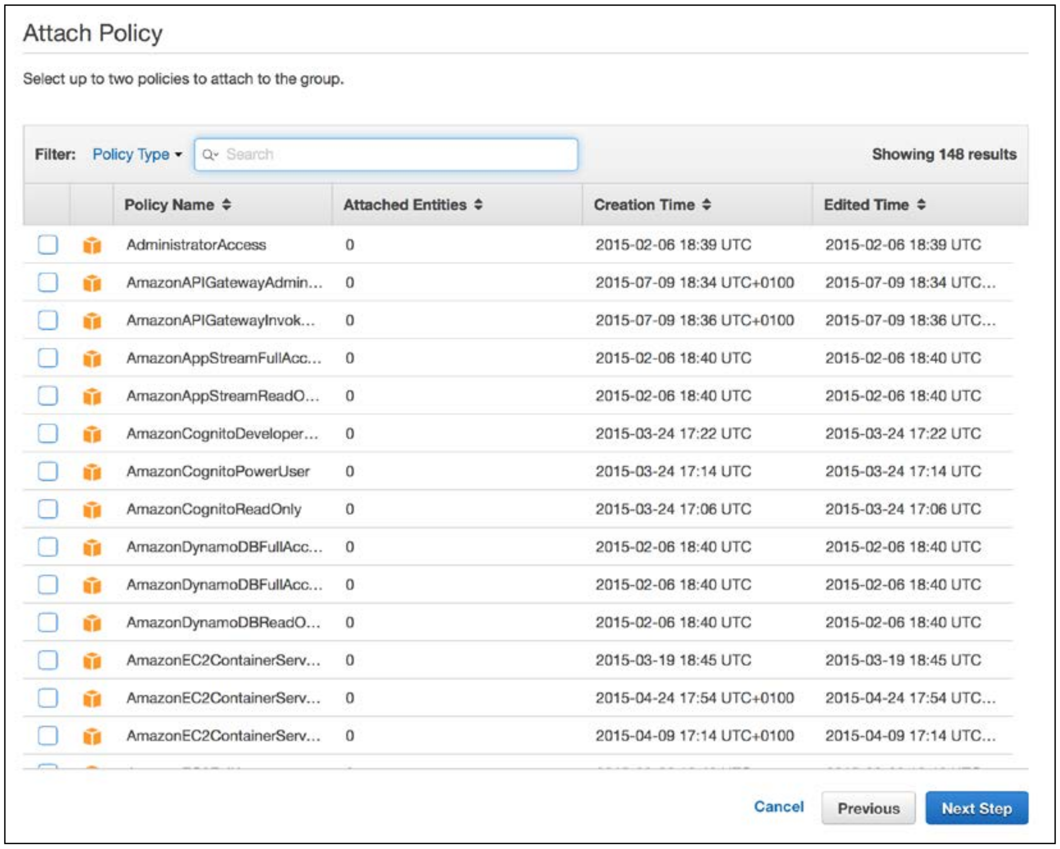
For users who need access to the AWS Management Consol

   Notice the tick box; this controls the creation of access keys to be used for the API. If you are creating users that will only interact with AWS via the web console, then you can leave this un-ticked. Once you are happy with your users, click on **Create**.

   The next screen will show you a confirmation that your users have been created and, if requested, the details for the API access.

6. Now that we have created users, we can provide permissions via user-groups. Return to the IAM panel and select **Use groups to assign permissions** and click on **Manage groups**. Like the user panel, you will be presented with a list of your current groups. Click on **Create New Group**.

7. Start by assigning a name for your group and click on **Next step**. This will take you to the policy selection; these should look a little something similar to this:



Attach Policy

Select up to two policies to attach to the group.

Filter: Policy Type ▾ Search — Showing 148 results

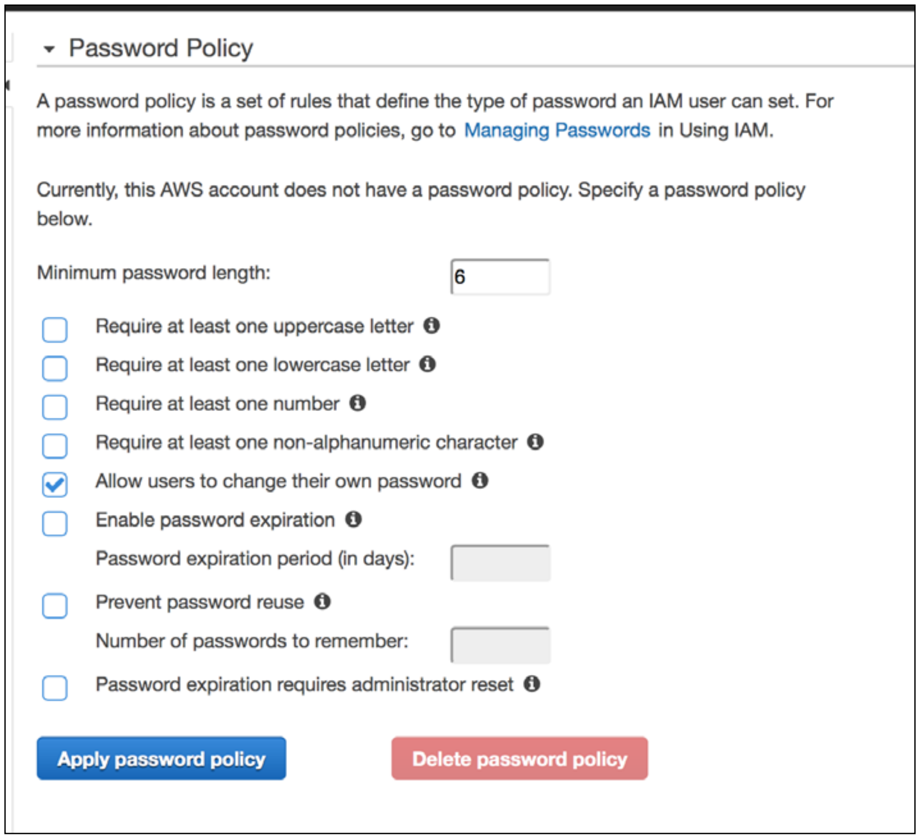
		Policy Name	Attached Entities	Creation Time	Edited Time
☐		AdministratorAccess	0	2015-02-06 18:39 UTC	2015-02-06 18:39 UTC
☐		AmazonAPIGatewayAdmin...	0	2015-07-09 18:34 UTC+0100	2015-07-09 18:34 UTC...
☐		AmazonAPIGatewayInvok...	0	2015-07-09 18:36 UTC+0100	2015-07-09 18:36 UTC...
☐		AmazonAppStreamFullAcc...	0	2015-02-06 18:40 UTC	2015-02-06 18:40 UTC
☐		AmazonAppStreamReadO...	0	2015-02-06 18:40 UTC	2015-02-06 18:40 UTC
☐		AmazonCognitoDeveloper...	0	2015-03-24 17:22 UTC	2015-03-24 17:22 UTC
☐		AmazonCognitoPowerUser	0	2015-03-24 17:14 UTC	2015-03-24 17:14 UTC
☐		AmazonCognitoReadOnly	0	2015-03-24 17:06 UTC	2015-03-24 17:06 UTC
☐		AmazonDynamoDBFullAcc...	0	2015-02-06 18:40 UTC	2015-02-06 18:40 UTC
☐		AmazonDynamoDBFullAcc...	0	2015-02-06 18:40 UTC	2015-02-06 18:40 UTC
☐		AmazonDynamoDBReadO...	0	2015-02-06 18:40 UTC	2015-02-06 18:40 UTC
☐		AmazonEC2ContainerServ...	0	2015-03-19 18:45 UTC	2015-03-19 18:45 UTC
☐		AmazonEC2ContainerServ...	0	2015-04-24 17:54 UTC+0100	2015-04-24 17:54 UTC...
☐		AmazonEC2ContainerServ...	0	2015-04-09 17:14 UTC+0100	2015-04-09 17:14 UTC...

These policies allow you to control the amount of access that users in this group will have in considerable detail. Select the appropriate policy, and click on **Next Step**.

You can see what a policy will do by reading its name; you can also find more details of IAM policies at `http://docs.aws.amazon.com/IAM/latest/UserGuide/policies_managed-vs-inline.html#aws-managed-policies`.

For this recipe, I recommend that you apply the **AdministratorAccess** role to any user that you will be following the recipes in this chapter with.

8. The next screen will allow you to review your choices; check that you are happy with them and select **Create Group**.
9. You should be returning to the `Groups` list. To add users to your new group, click on it in the list and select **Add Users to Group**. You can then select the users you wish to apply the new role to and add them.
10. Finally, you can set the overall password policy for your AWS infrastructure by returning it to the IAM panel and selecting **Apply an IAM policy**. A password policy is very important and allows you to ensure that your users set strong passwords. In combination with an MFA token, it makes the brute force cracking of user accounts as hugely unlikely. The **Password Policy** panel resembles the following screenshot:



A password policy is a set of rules that define the type of password an IAM user can set. For more information about password policies, go to Managing Passwords in Using IAM.

Currently, this AWS account does not have a password policy. Specify a password policy below.

Minimum password length: 6

- [ ] Require at least one uppercase letter
- [ ] Require at least one lowercase letter
- [ ] Require at least one number
- [ ] Require at least one non-alphanumeric character
- [x] Allow users to change their own password
- [ ] Enable password expiration

  Password expiration period (in days):
- [ ] Prevent password reuse

  Number of passwords to remember:
- [ ] Password expiration requires administrator reset

Select the options that make the most sense for your organization; you should ensure a decent length of password and at least some guarantee of variety.

See also

You can find details of the IAM utility at `https://aws.amazon.com/documentation/iam/`

Creating your first security group

Security groups are the equivalent of a firewall and defines the type of traffic that can be directed to a host, both originating from your platform and from the outside world. Like any firewall, it's important to define the security groups correctly, as this is the first point of securing your traffic.

This recipe will show you how to define a new security group ready to be applied to new EC2 hosts to use as a basic web server.

Getting ready

For this recipe, you need an AWS account.

How to do it...

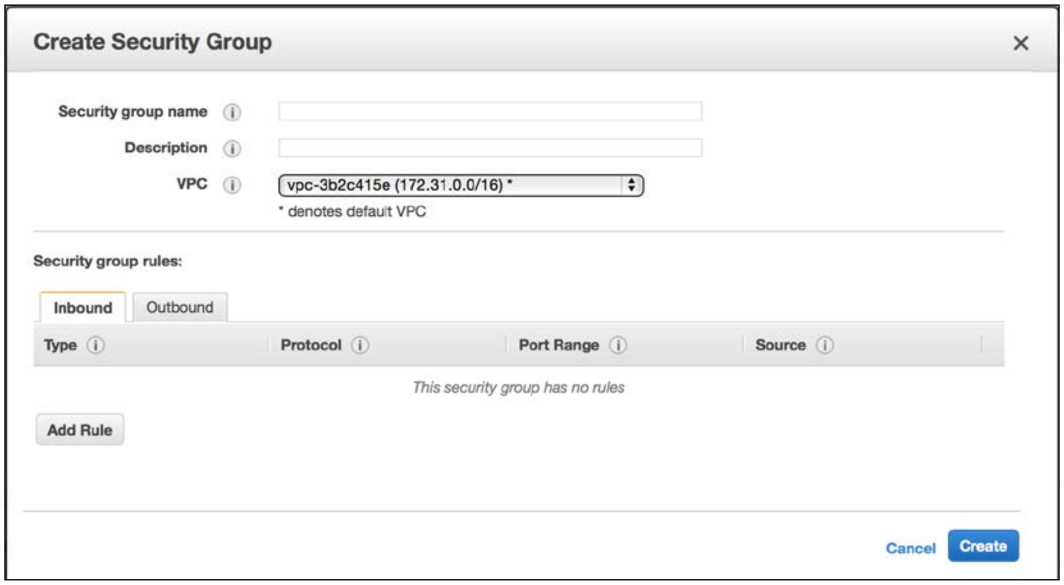
The following recipe will demonstrate how to use the EC2 security group configuration to set a secure policy.

1. Log into the AWS management console, and select the **EC2** options:

   In the EC2 panel you will find a menu on the left-hand side; select the menu entry called **Security Groups** around halfway down the menu.

2. In the **Security Groups** panel, you should find a list of security groups; by default, you should only have one, the default group. To create a new one, select the **Create Security Group** button.

3. You should be presented with a screen similar to the following:

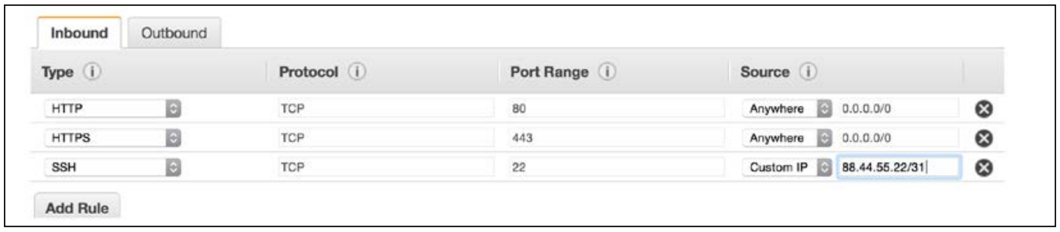


This allows you to give your new security group a name, a description, and attach it to a VPC.

A **VPC** is a **virtual private cloud**. AWS allows you to have several segregated clouds, each with its own policies and hosts. This allows separation of concern between them, for instance a development infrastructure and a production environment. It is not to be confused with availability zones, which offers geographically separated sites to aid with reliability and performance.

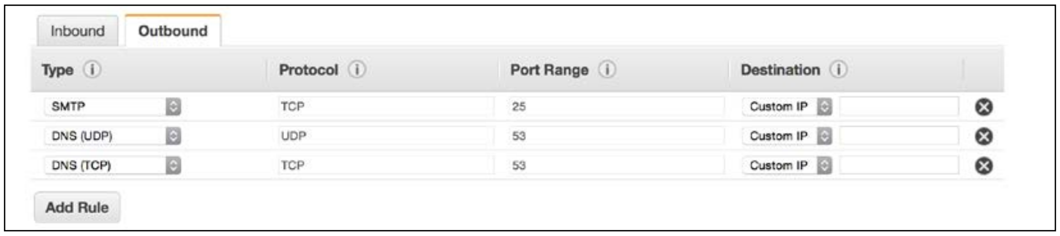
Once you have named your group and given it a description, click on the **Add Rule button**.

4. The next step is to create your inbound rule set; this is the traffic that you will allow to reach your instances from the outside world. Click on the button titled **Add Rule**; this will open a window that allows you to select the rules you wish to add. The dropdown list allows you to select either from a set of common types or a free-form custom type. In my example, I've added the common ports for a web server as shown in the following screenshot:



Type	Protocol	Port Range	Source	
HTTP	TCP	80	Anywhere	0.0.0.0/0
HTTPS	TCP	443	Anywhere	0.0.0.0/0
SSH	TCP	22	Custom IP	88.44.55.22/31

5. Note that I have also added SSH to the inbound group; this is to allow me to manage instances. It's worth noting that I have locked this rule down to a specific IP address to ensure that it can't be accessed via random port brushing by opportunistic scripted attacks.
6. Now, since we have set our inbound rules, we can now set our outbound rules. By default, these are set to allow all traffic out; this can be a valid configuration, but if possible, it is better to lock this down by removing the default outbound rule and allowing only certain protocols. In the following example, I've limited any instance using this security group to only be able to access DNS and SMTP:



Type	Protocol	Port Range	Destination	
SMTP	TCP	25	Custom IP	
DNS (UDP)	UDP	53	Custom IP	
DNS (TCP)	TCP	53	Custom IP	

Once you have edited the rule-set to your liking, click on the **Create** button. The security group is now available for use everywhere within this VPC.

See also

You can find details of the AWS security groups at `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html`.

Creating your first EC2 host

The basic unit of an IAAS platform is a compute unit and on AWS it is comprised of EC2 (Elastic Compute) instances. An EC2 instance is a highly configurable server, which is able to cater to almost any conceivable usage; from CPU hungry analysis platforms to IO bound database instances. As noted in the introduction, you should keep in mind that these are generally contended, and you should factor that into application design.

> Within the creation process of an EC2 instance, you have the option to create a dedicated instance. This allows you to remove some of the contention constrains and allows security conscious users to ensure that they are not physically sharing equipment. This is a particular concern for users in the industries, such as the financial sector.

This recipe will show you how to create a new EC2 host running Ubuntu.

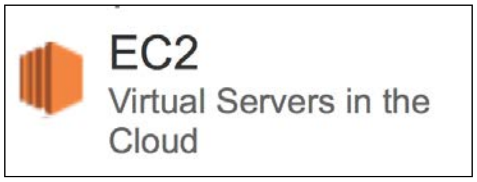
Getting ready

For this recipe, you will need an AWS account.

How to do it...

The following steps will demonstrate how to create a new Elastic Compute instance and assign a security group to it.

1. Login to the AWS management panel and select the EC2 panel using the following icon:



2. The EC2 panel gives you a top-level overview of the EC2 resources you are using and gives you a quick view of the health of your region.

> Keep in mind, this view is just for a single region. If you are using multiple regions, then you will need to use the drop down in the top right of the screen to select that particular region. It can be easy to forget that an instance is running if it's not in your default region.

It is also where you can launch new Instances. Click on the button marked **Launch Instance** to start a new EC2 compute instance.

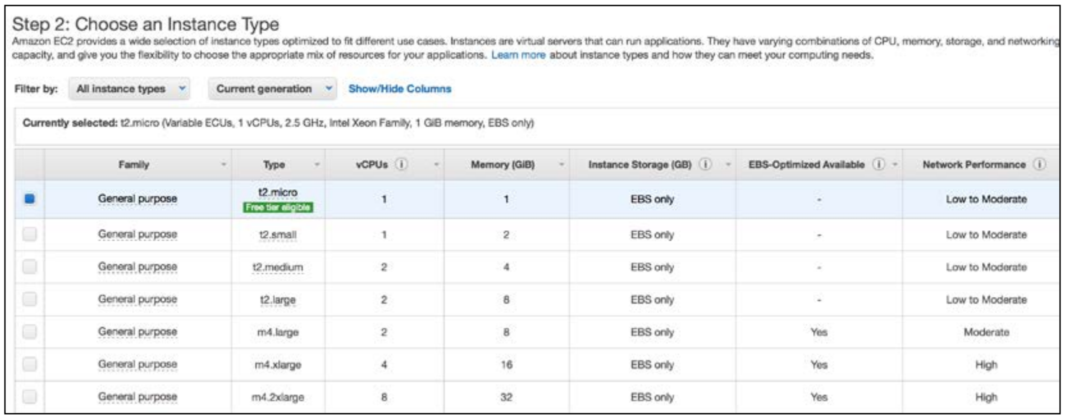
3. The first step is to select the type of **Amazon Machine Image** (**AMI**) that you wish to create. This essentially boils down to selecting the operating system/distribution that you prefer and covers the most popular Linux distributions and even Window.

   If you select a commercial distribution, such as RedHat or Windows; it attracts a higher cost due to licensing. This is reflected in the hourly running cost of the instance.

   Select your preferred distribution and click on the button marked **Select** next to it.

   It's worth noting that you can create your own AMI's from launched instances. This allows you to create your own AMI, complete with tools that you require rather than the generic images offered.

4. The next choice is the instance type. As you can see in the following screenshot, this is essentially your choice of virtual hardware and storage type and varies from the micro instances made-up of a single processor and 1GB of memory, to the compute optimized instances made up of 36 processors and 60GB of RAM.



Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types | Current generation | Show/Hide Columns

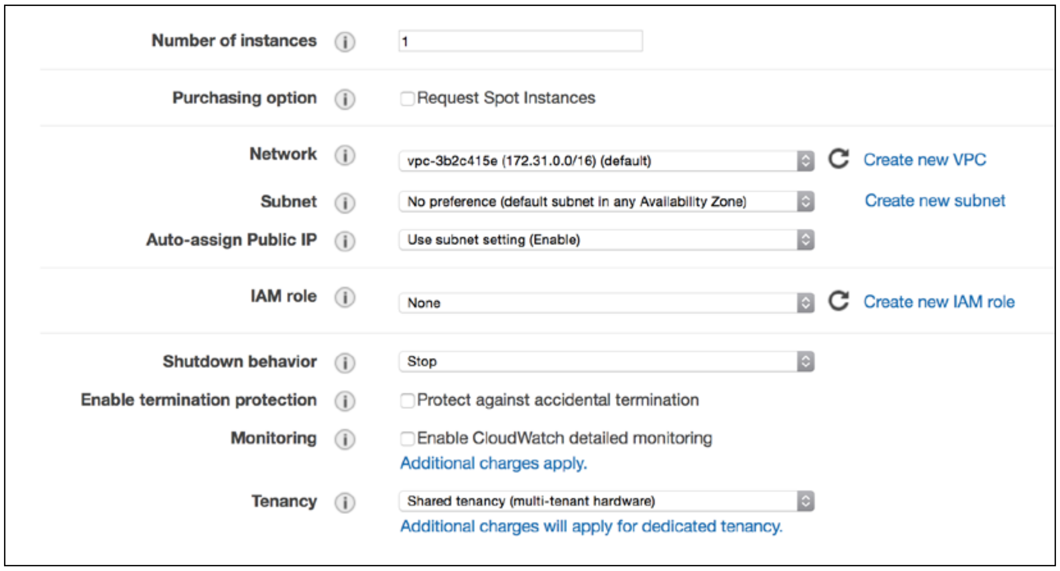
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
■	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
☐	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
☐	General purpose	t2.large	2	8	EBS only	-	Low to Moderate
☐	General purpose	m4.large	2	8	EBS only	Yes	Moderate
☐	General purpose	m4.xlarge	4	16	EBS only	Yes	High
☐	General purpose	m4.2xlarge	8	32	EBS only	Yes	High

5. Select the instance type that fits your intended usage and budget.

EC2 pricing can become extremely complex and it's worth reading up on the various costs at `https://aws.amazon.com/ec2/pricing/`. There are calculators available on the pricing portal that will allow you to come to an indicative price for your AWS infrastructure.

6. You can click on the **Review and Launch** button at this point. This will launch the instance with a default security group and the default VPC; however, for this example, click the button marked **Configure Instance Details** to set more options.
7. The next screen will allow you to configure the details of your instance, as you can see in the following screenshot:

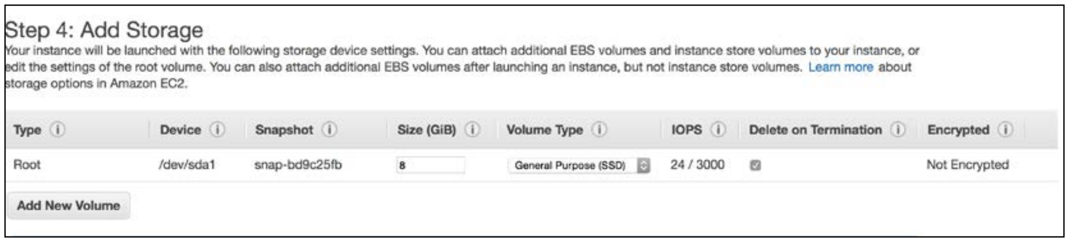


Number of instances	1
Purchasing option	Request Spot Instances
Network	vpc-3b2c415e (172.31.0.0/16) (default)
Subnet	No preference (default subnet in any Availability Zone)
Auto-assign Public IP	Use subnet setting (Enable)
IAM role	None
Shutdown behavior	Stop
Enable termination protection	Protect against accidental termination
Monitoring	Enable CloudWatch detailed monitoring Additional charges apply.
Tenancy	Shared tenancy (multi-tenant hardware) Additional charges will apply for dedicated tenancy.

This includes elements, such as the number of instances, network, and subnet if you have multiple VPC's defined. One element to consider is if you want this instance to be publicly available; if you do, you can select the option marked **Auto-assign Public IP**. This will give this instance a publicly available IP address from Amazon's pool and make the instance available to the wider Internet.

8. You can also receive detailed monitoring using Amazon's monitoring product, Cloudwatch; you can leave this un-ticked and still receive the basic Cloudwatch package. However, if you require any level of detail about the instance usage, the commercial Cloudwatch package is probably of interest.
9. By setting the **Tenancy** option, you can run it on an isolated hardware; thus, ensuring that you are not sharing the compute unit with other clients; this can be useful if you have security policies that demand a certain level of isolation. Be aware though, that this option attracts a higher hourly fee.

10. Once you are happy with your selections, click on **Add Storage**.
11. The storage page allows you choose both the size of your **EBS** (**Elastic Block Store**) volumes and the type and performance.



Step 4: Add Storage

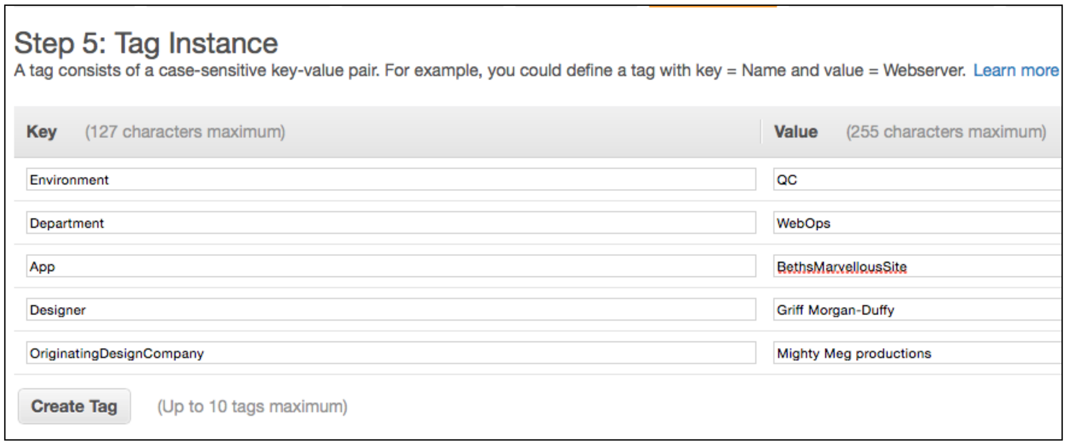
Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Delete on Termination	Encrypted
Root	/dev/sda1	snap-bd9c25fb	8	General Purpose (SSD)	24 / 3000	✓	Not Encrypted

Add New Volume

Of particular note in this panel is the volume type option; here, you can select from a general purpose SSD, guaranteed IOP's SSD, and platter based magnetic storage. Each of these has a trade-off in terms of cost and performance and you should select the one most suitable for your usage. You can also select what happens to your data when you terminate your EC2 instance; by default, it is deleted along with the instance. However, you can use this panel to save certain partitions so that they are kept when the instance is terminated.

12. Once you have created the appropriate storage for your image, select **Tag Instance**.
13. Within the tag instance page, you can add free form metadata in the form of key value pairs. This allows you to set a tag to denote the environment this instance is for, department, or any other relevant information. See the following screenshot for examples of tagging:



Step 5: Tag Instance

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. Learn more

Key (127 characters maximum)	Value (255 characters maximum)
Environment	QC
Department	WebOps
App	BethsMarvellousSite
Designer	Griff Morgan-Duffy
OriginatingDesignCompany	Mighty Meg productions

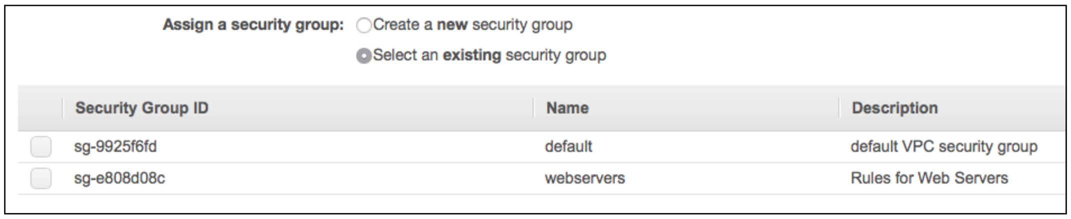
Create Tag (Up to 10 tags maximum)

Once you've added tags, click on the button marked **Configure Security Group**.

14. The security group page allows you to add the security group details for your new instances. By default, you can create a new security group, or alternatively, you can select a pre-existing group.

    From my experience, it is better to create your own security groups up front rather than creating them on a per instance basis. This helps to keep a cohesive policy in place and avoids configuration drift and replication.

    If you have an existing security group, you can select **Select an existing security group** and a list of the existing groups will be displayed. Similar to the following screenshot:

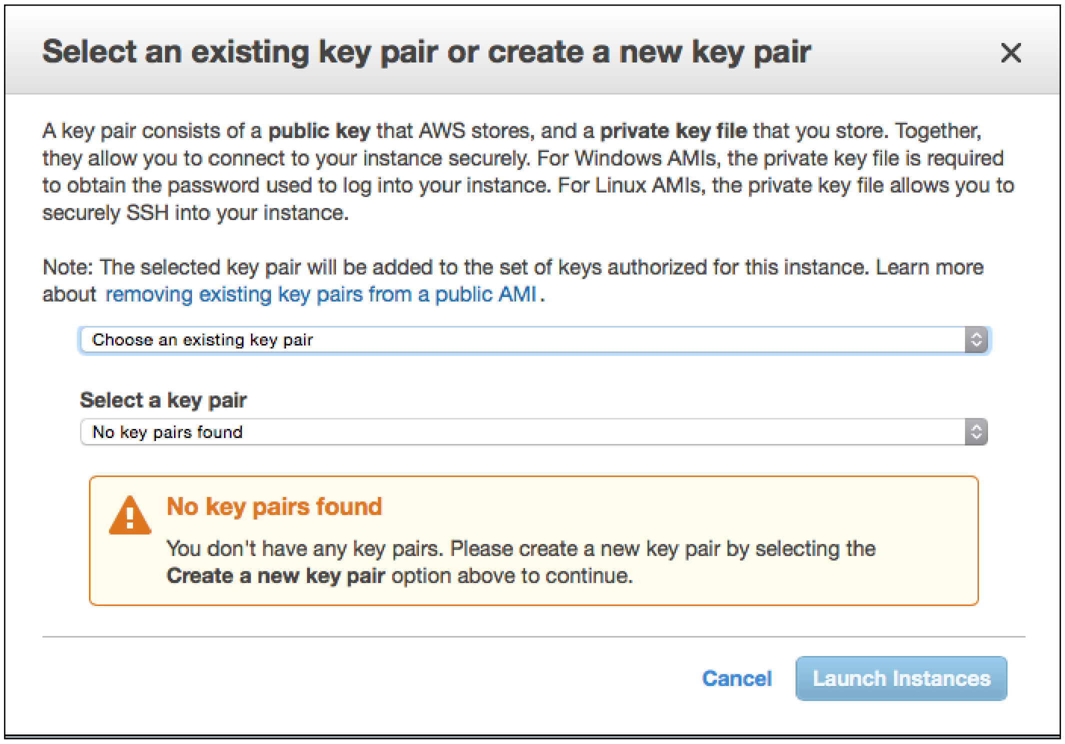


Assign a security group: Create a **new** security group [✓ Select an **existing** security group]

	Security Group ID	Name	Description
☐	sg-9925f6fd	default	default VPC security group
☐	sg-e808d08c	webservers	Rules for Web Servers

Make a note that you can create a new security group based on an existing one by selecting the **Copy to new** option. This is useful when you have an existing complex policy that requires a simple tweak for the new instance.

15. Once you are happy with your selections, click on **Review and Launch**.
16. The next page allows you to review your instance details and gives you the option to edit any elements that don't look quite right. Once you're happy, you can select launch.
17. If you haven't uploaded an SSH key pair you will be prompted to create one in the following screenshot:



Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about removing existing key pairs from a public AMI.

Choose an existing key pair

Select a key pair

No key pairs found

No key pairs found

You don't have any key pairs. Please create a new key pair by selecting the **Create a new key pair** option above to continue.

18. You also have the option to proceed without using a Key pair; however, I do not recommend this unless you are using a custom AMI with pre-existing secure authentication methods built in.
19. Once you have created and downloaded your key pair, click on **Launch instance**.

[THIS IS THE ONLY TIME YOU CAN DOWNLOAD THE KEYS. Be sure that you have copied them down, as there is absolutely no way to re-download them afterwards.]

You will be shown a page that shows the EC2 instance being set up and you will return to the EC2 panel. Your new instance should be listed alongside its Public IP and you should now be able to log in using the key pair you downloaded in the previous step.

See also

You can find the documentation for EC2 at `https://aws.amazon.com/documentation/ec2/`.

Using Elastic Load Balancers

One of the key elements of running applications on an IAAS platform is to ensure that work can be balanced between many nodes. This offers an increased resiliency, but also allows you to offset one of the weaknesses of an IAAS platform, performance issues caused by contention. Unless you have a compelling reason not to, you should use load balancing for any production instance of an application.

This recipe shows that you how to set up an Elastic Load balancer group that directs web traffic between two different nodes.

Getting ready

For this recipe, you will require an AWS account and at least a single EC2 instance.

How to do it...

The following steps show you how to create a new EC2 Elastic Load Balancer and assign both balanced ports and an EC2 to host it:

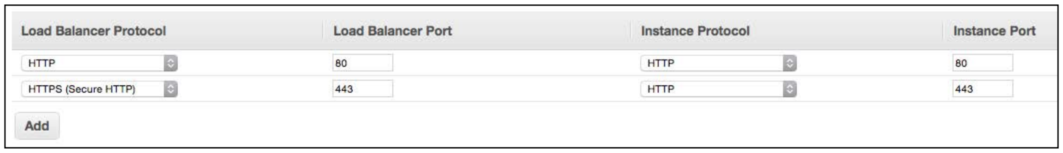
1. Start by logging into your AWS management panel and select the EC2 link.
2. In the EC2 management panel, locate the link titled Load Balancers in the left-hand menu and click on it.
3. The next page gives you a summary of your Load Balancers. If this is your first time in this panel, it should be blank. Click on the blue button at the top of the screen marked **Create Load Balancer** to create a new Load Balancer.
4. The next page allows you to set certain basic elements of your load balancer. In this example, I am going to create a load balancer to balance common web traffic (HTTP and HTTPS). You can start by giving the load balancer a name that will allow you to identify it. Next, you can select the VPC you wish to create the load balancer in and select if this is an internal only load balancer

> Internal load balancers are extremely useful to balance load between your applications internally, and is especially crucial when designing architectures, such as a Micro Service based app.

5. You can also perform advanced VPC configurations at this point, allowing you to select specific VPCs to load balance to if you require granularity over the configuration.
6. Next, we have the listener configuration. This is where you select the ports you wish to load balance.

> Notice that you can select the load balancer and instance port to be different; this is exceptionally useful when you cannot make the application listen on port 80 directly; for instance, with applications that run inside Tomcat containers.

As you can see in the following screenshot, I have kept it simple and added both HTTP and HTTPS to be load balanced:



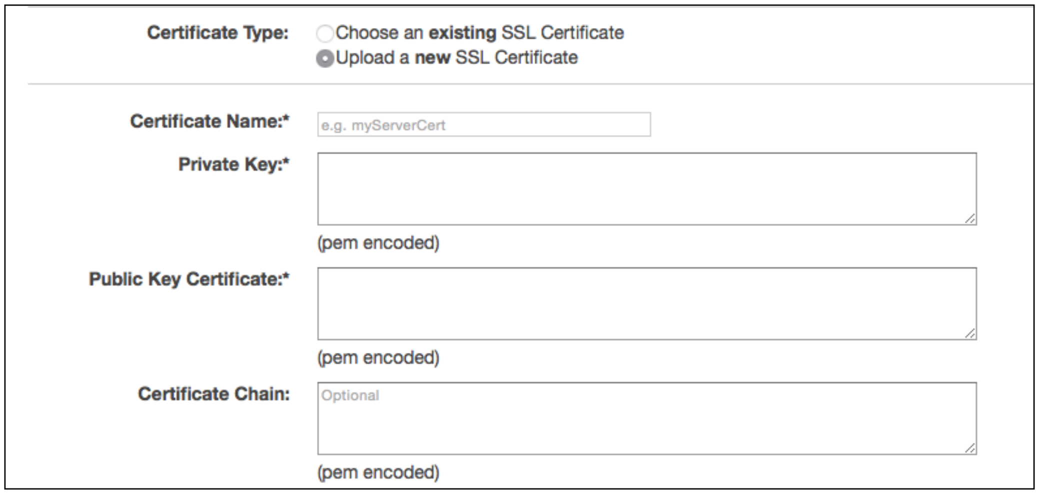
Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
HTTP	80	HTTP	80
HTTPS (Secure HTTP)	443	HTTP	443

Once your happy with your choices, click on the button titled '**Assign Security Groups**

7. Much like creating an EC2 instance, assigning security groups allows you to set the pre-existing security group you wish to use or alternatively to set up new ones. If you create a new group, it should use the initial listener configuration to create a template.
8. Once you have created/selected a group that you are happy with; click on **Configure Security Settings**.
9. If you have selected a listener to be HTTPS then you will be directed to the next page, **Configure Security Settings**, otherwise you will have to click through a warning that notes that you have no secure listener.

> It is now expected that a website should be usable over HTTPS, and indeed, it is becoming increasingly common for sites to only offer HTTPS connections, and Google will penalize the site in it's page ranking if they do not offer HTTPS.

This page allows you to upload an SSL certificate, which will allow the load balancer to terminate the SSL connections for you, saving you the need to configure individual servers to handle an SSL termination. If you have previously uploaded a certificate, then you can select it here; otherwise, you can paste the details into this form:

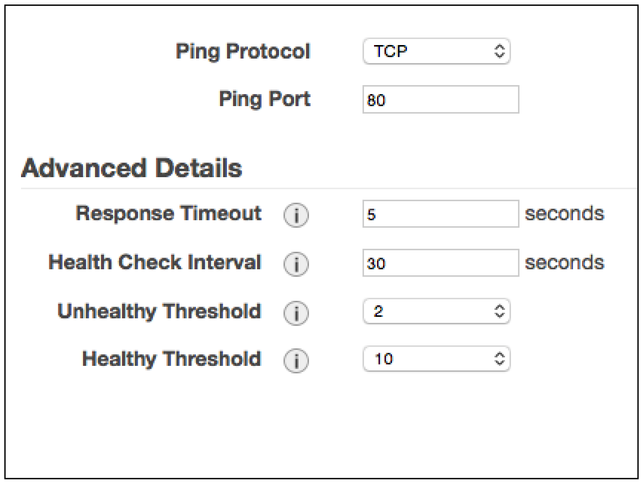


Certificate Type: Choose an **existing** SSL Certificate [✓ Upload a **new** SSL Certificate]

Certificate Name:*	e.g. myServerCert
Private Key:*	(pem encoded)
Public Key Certificate:*	(pem encoded)
Certificate Chain:	Optional (pem encoded)

Once you have entered your certificate details, click on the button marked as **Configure Health Check**.

10. In the next screen, you can set the health check that will be regularly sent to the instances that the load balancer is sending traffic to. This allows the load balancer to remove instances that are no longer capable of serving requests or are no longer performant. The health check comprises of either a HTTP or HTTPS request for a nominated health page, or can use a simple TCP or SSL check on a certain port.



Ping Protocol	TCP
Ping Port	80

Advanced Details

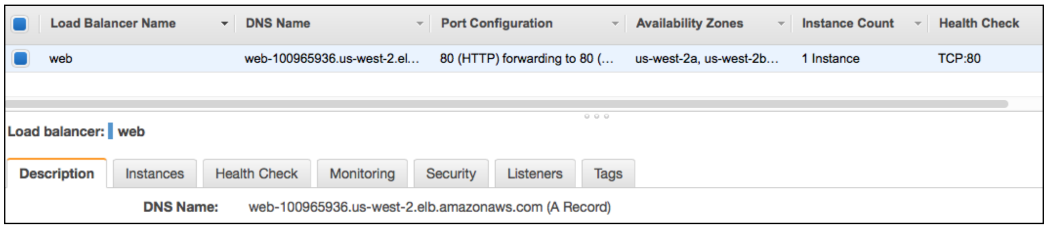
Response Timeout	5 seconds
Health Check Interval	30 seconds
Unhealthy Threshold	2
Healthy Threshold	10

As you can see in the preceding screenshot, you can also set a reasonable set of granularity to the checks; thus, allowing you to tweak how slowly a node should respond to be considered unhealthy and how often to call the health check itself. Once you are happy with your choices, click on the button marked **Add EC2 Instances.**

11. The next screen will show you a list of your EC2 instances and will allow you to select the members for your load balancer group. Select each of the instances that you wish to partake in the load-balanced group. If you are using multiple availability zones, you can also set options to allow you to balance across them, allowing you to spread traffic geographically.
12. Once you have selected the EC2 instances that you wish to include in the load balancer, click on the button marked as **Add Tags**.
13. The next screen allows you to set tags against the load balancer; as with EC2 instances, this allows you to add key value pairs and helps you to identify your load balancer within a busy AWS infrastructure.

    Add any tags you believe are relevant and click on the blue button marked **Review and Create**.

14. The next screen will show you a summary of the load balancer you are about to create and it will offer you the opportunity to amend any details. Review them and when you are happy, click on the button marked **Create load balancer**. The load balancer will be created and you will be passed back to the **Load Balancer management** screen.
15. The Load Balancer management screen should list your new load balancer and it should look something similar to the following screenshot:



Load Balancer Name	DNS Name	Port Configuration	Availability Zones	Instance Count	Health Check
web	web-100965936.us-west-2.el...	80 (HTTP) forwarding to 80 (...	us-west-2a, us-west-2b...	1 Instance	TCP:80

Load balancer: web

DNS Name: web-100965936.us-west-2.elb.amazonaws.com (A Record)

Note the DNS name; you should use a CNAME DNS record to point your records that you wish to be load balanced.

[Do not be tempted to use an A record; the IP address of the Elastic Load Balancers can and will shift.]

16. You should now be able to direct traffic at your new load balancer and have it balanced across your selected EC2 instances.

See also

You can find further details of Elastic Load Balancing at `https://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elastic-load-balancing.html`.

Managing DNS with route53

Domain Name Services (**DNS**) underpins the Internet and a reliable and performant DNS service is crucial to running a web service of any kind. It's a surprisingly neglected area of performance monitoring and a poorly configured or poorly performing DNS server can have a very large impact on the health of your application.

AWS Route53 is a highly scalable DNS server, which has advanced features allowing for geo-routing, Apex CNAME records, and so on. It is also highly performant with DNS servers in most major geographical areas, allowing for a minimum network hops to resolution.

This recipe will show you how to set up a new DNS zone and how to add new records.

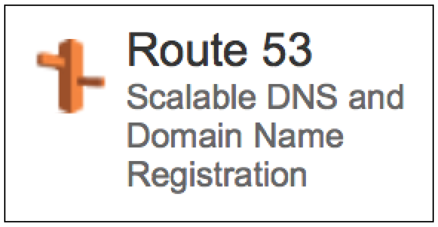
Getting ready...

For this recipe, you need an active AWS account. You will also need a registered domain name that you wish to use.

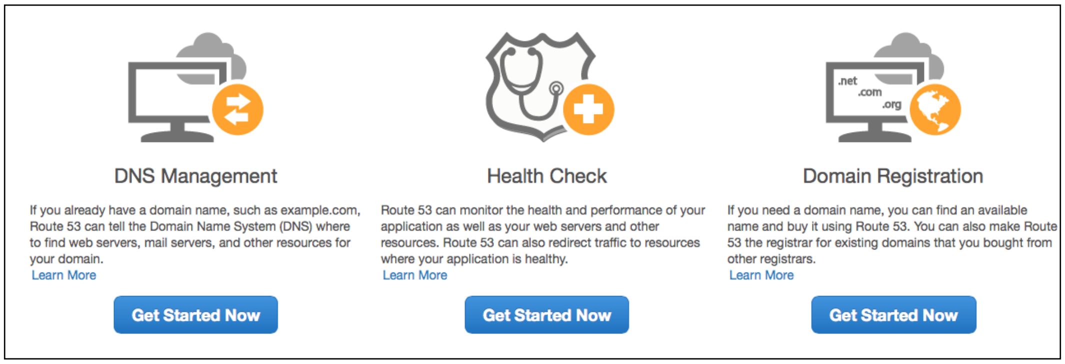
How to do it...

This recipe will show you how to take an existing domain and manage its DNS records using Amazon Route53:

1. Log into your AWS management console and select the **Route 53** panel by clicking on the following icon:



This will take you into the Route53 panel and if this is your first time, you will be greeted with the following introductory screen:



DNS Management

If you already have a domain name, such as example.com, Route 53 can tell the Domain Name System (DNS) where to find web servers, mail servers, and other resources for your domain.
Learn More

Health Check

Route 53 can monitor the health and performance of your application as well as your web servers and other resources. Route 53 can also redirect traffic to resources where your application is healthy.
Learn More

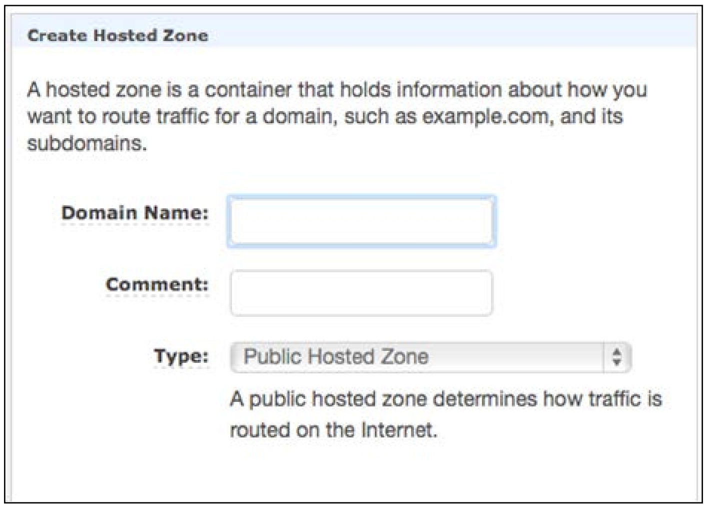
Domain Registration

If you need a domain name, you can find an available name and buy it using Route 53. You can also make Route 53 the registrar for existing domains that you bought from other registrars.
Learn More

For this recipe, I'm going to assume that you already have a domain to manage; if this is not the case, then you can select the domain registration and register them via AWS.

2. Click on the button titled **Get Started Now** underneath the DNS management header; this will take you to the DNS management panel. The management panel allows you to see at a glance the DNS zones that you are managing via Route53. Click on the blue button marked **Create Hosted Zone** to create a new zone.

   This should open a panel allowing you to input your `Zone name` and resembles the following screenshot:



Create Hosted Zone

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

Domain Name:

Comment:

Type: Public Hosted Zone

A public hosted zone determines how traffic is routed on the Internet.

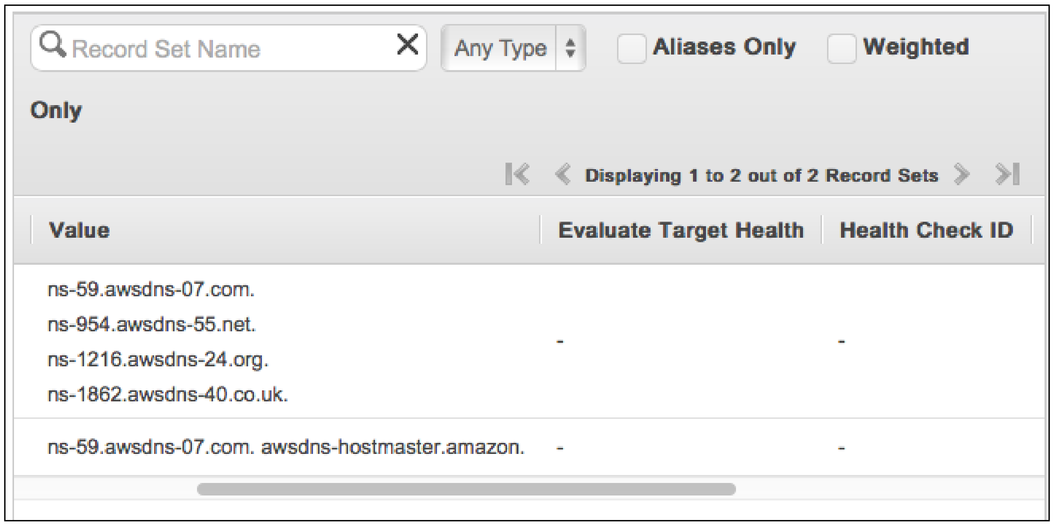
This allows you to enter your zone name (for instance, example.com), and select the scope. By default, it is created as a public DNS zone and will make the zone available to all public clients; however, you can create zones that are only available to instances hosted within AWS. This is useful in building your internal network and keeping its records hidden from public view.

Input your domain name into the input and click on the blue button titled **Create**.

3. Once you have entered your domain name, you will be taken to the record set management screen. This allows you to enter the records that describe your infrastructure and any permissible DNS record type.

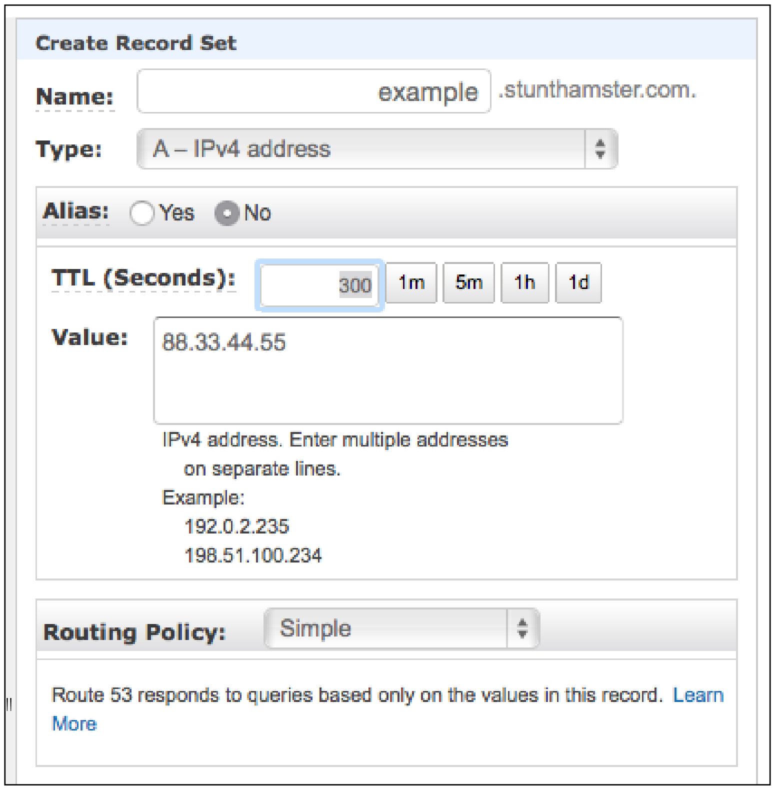
> You can find a good rundown of DNS record types at `https://support.google.com/a/answer/48090?hl=en`.

By default, the Zone will be populated with the name server records, and should look like this:



Value	Evaluate Target Health	Health Check ID
ns-59.awsdns-07.com. ns-954.awsdns-55.net. ns-1216.awsdns-24.org. ns-1862.awsdns-40.co.uk.	-	-
ns-59.awsdns-07.com. awsdns-hostmaster.amazon.	-	-

4. To create a new record, click on the blue button marked **Create Record Set**.
5. The record set creation screen allows you to enter your DNS record; for instance, creating an A record would look something like this:



Create Record Set

Name: example .stunthamster.com.

Type: A – IPv4 address

Alias: Yes [✓ No]

TTL (Seconds): 300 1m 5m 1h 1d

Value: 88.33.44.55

IPv4 address. Enter multiple addresses on separate lines.
Example:
192.0.2.235
198.51.100.234

Routing Policy: Simple

Route 53 responds to queries based only on the values in this record. Learn More

6. Route53 is like many DNS systems in that you enter a record name, select the DNS type, and enter a relevant value and TTL (Time to live). Where Route53 differs is its routing policy. Routing policies allow you to add additional flexibility to your DNS infrastructure, allowing you to add features, such as Geographical DNS, allowing different countries to resolve to different IP's.

   Enter the desired record type and click 'create' to create your new record. Once you click on create, your record will be created, propagated, and made available to the public.

See also

You can find further details of Route53 at `https://aws.amazon.com/documentation/route53/`.

Using Ansible to create EC2 hosts

Using an IAAS platform ensures that you automate early and often; due to the cost model of AWS, it's more efficient to be as on demand or elastic as possible. By using automation, you can automate the deployment of an application across its entire lifecycle, from the creation of a host to deployment of an application and finally, the destruction of the host. This also allows you to stand up to new environments easily, as there is no need to provision new hardware, networks, or any other such activities.

This recipe will demonstrate how to use Ansible to create a new EC2 host.

Getting ready

For this recipe, you will need an active AWS account and an Ansible client.

How to do it...

The following recipe demonstrates how to use a very simple Ansible playbook to automatically create an EC2 instance:

1. First, create a directory to hold your Ansible code using the following command:

   ```
   mkdir ansibleec2
   ```

2. Next, we create our inventory. Create a new file called ec2inventory and insert the following content:

   ```
   [ec2]
   localhost
   ```

   Note that you are simply asking Ansible to run commands on the same host on which the Ansible playbook is run; this is because you are essentially using your own host to run commands against Amazon's EC2 API rather than interactively working directly with AWS.

3. Now we have an inventory, we can use Ansible to create our EC2 instance. Create a new file called `ec2.yml` and add the following code:

   ```
   - hosts: ec2
     connection: local
     tasks:
       - name: 'Create EC2 instance'
         ec2:
             aws_access_key: AKIAICW3HOQKY6LMCD4A
             aws_secret_key: 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
   ```

```
                key_name: test
                region: us-west-2
                group: webserver
                instance_type: t2.micro
                image: ami-5189a661
                wait: yes
                wait_timeout: 50
                monitoring: yes
                vpc_subnet_id: subnet-89770eec
                assign_public_ip: yes
```

This code is relatively straightforward and is an example of using a playbook outside of a role. First, it ensures that it uses the localhost entry listed in the inventory and also uses a local connection rather than the usual SSH. Next, we define our list of tasks, in this case calling the Ansible EC2 module and giving it several parameters. These are generally self-explanatory and map directly back to the options you could use in the GUI. One interesting parameter is the wait parameter; this ensures that Ansible will only return once the host is created, rather than assuming that all is well.

4. Run the playbook using the following command:

```
ansible-playbook -i ec2inventory ec2.yml
```

This will create a new EC2 instance and start its running.

Its also worth pointing out that in production use you will want to either pass the AWS credentials as an environment variable or encrypt them using Ansible vault. You can find the details at `http://docs.ansible.com/ansible/guide_aws.html#authentication`.

See also

You can find further details of the EC2 Ansible module at `http://docs.ansible.com/ansible/ec2_module.html`.

12 Application Performance Monitoring with New Relic

In this chapter, we are going to cover the following topics:

- Signing up for a New Relic account
- Installing the New Relic Java agent
- Using the performance overview
- Locating performance bottlenecks with Transaction Traces
- Observing database performance with New Relic
- Release performance monitoring with New Relic
- Server Monitoring with New Relic

Introduction

Monitoring is a key part of running any system, from the smallest to the largest; however, monitoring takes many forms. We covered application monitoring in *Chapter 10, Monitoring with Sensu* and those recipes demonstrated how to set up monitoring for application faults. Fault monitoring is only one part of the picture; however, for a true reflection of your application's health, you need to know how well it is performing, how effectively it can communicate with it's dependencies (Databases, external API's, and so on), and if there are particular elements that are not performing as you expect.

This monitoring niche can be filled with an **Application Performance Monitoring tool** (**APM**). APM solutions offer the ability to drill into your application, exposing performance and reliability issues with considerable detail. Generally speaking, they use some form of agent that allows the APM tool to insert hooks into the running code at runtime, allowing it access to method calls, network calls, and so on. It may also offer additional tools, such as more 'traditional' server monitoring, external site monitoring, and so on.

The key to a good APM tool is the ability to present this myriad form of data in an accessible form and allow proactive reporting and alerting to be set up against it. Most, if not all APM tools on the market today offer web consoles that allow you to drill into the data in a fairly intuitive fashion; this is important, as it can offer a meeting place for Developers, stake holders, and infrastructure staff to examine issues, without needing to learn many different esoteric tools.

The APM market place is growing and at the time of writing, there are several competitors, but by far, the most popular tools are AppDynamics and New Relic; both offer a broad set of features allowing detailed monitoring of application performance and both offer a SAAS platform (although AppDynamics also offers on premises options). Both AppDynamics and New Relic are commercial tools, albeit ones that offer free limited accounts. At present, the open source community has yet to create a tool that can compete on features; however, that looks to be changing, with several projects showing promise.

In this chapter, we are going to briefly look at New Relic. New Relic is easy to use, offers a free (if limited account), and a trial of its full package. To cover the product in detail requires a book in itself; so, I have focused on some of the key elements, with recipes that demonstrate how to pinpoint common application bottlenecks.

Signing up for a New Relic account

Before we can use New Relic, we need to create an account. This is straightforward and by default, it will offer you a trial of the full enterprise product. After the trial expiration, the account will revert to a free limited account. The free account misses many of the options; thus, allowing for detailed monitoring; however, it still allows for an excellent top-level view of your application.

Getting ready

For this recipe, you will need a web browser.

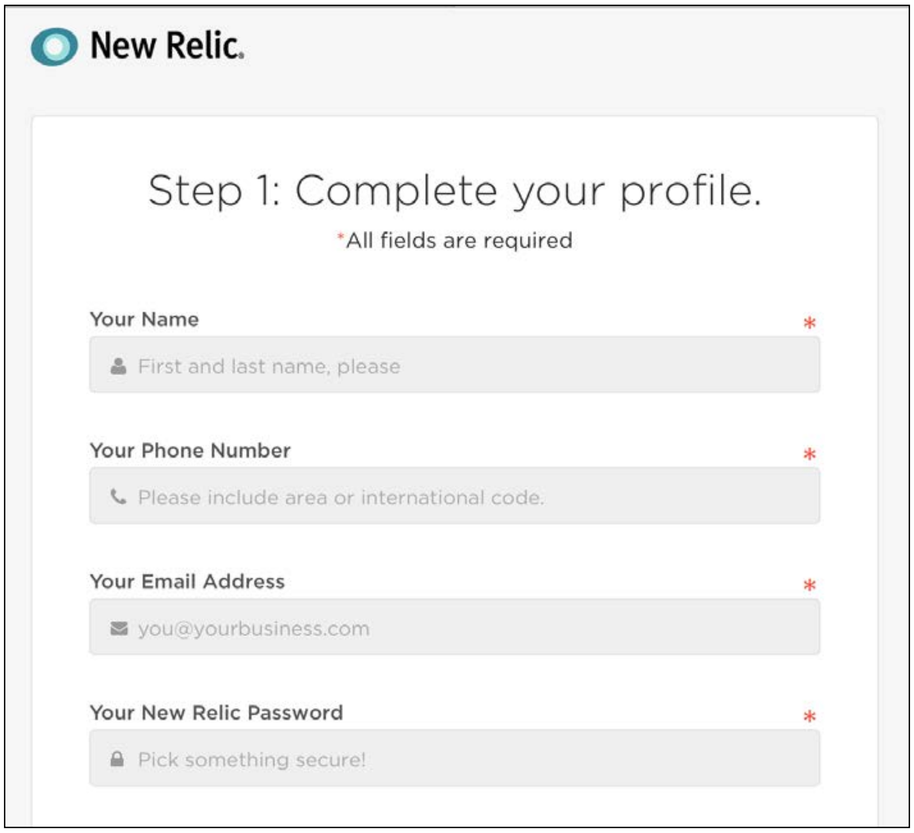
How to do it...

Let's start up with signing up with New Relic account:

1. Open the following URL in your browser:

   `https://newrelic.com/signup`

2. You should see a screen similar to the following:



New Relic

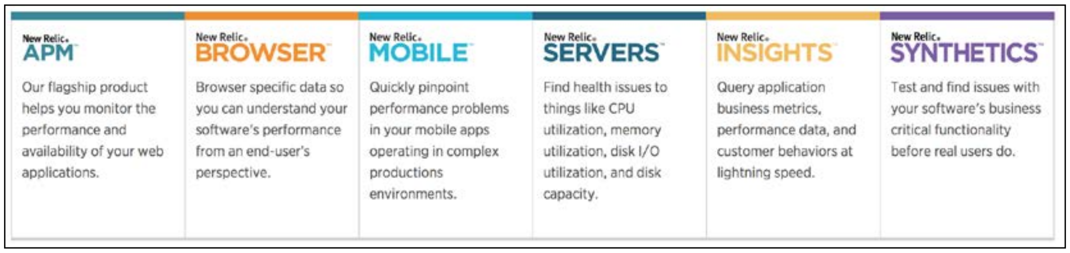
Step 1: Complete your profile.

*All fields are required

Your Name *	First and last name, please
Your Phone Number *	Please include area or international code.
Your Email Address *	you@yourbusiness.com
Your New Relic Password *	Pick something secure!

This single page form is all you need to fill in order to create your account. Ensure that the e-mail address you use is valid, as you will need to confirm the sign up.

3. Once your account has been created, you will be taken to the initial sign in page; this will show you a banner of the New Relic application stack:



New Relic APM
Our flagship product helps you monitor the performance and availability of your web applications.

New Relic BROWSER
Browser specific data so you can understand your software's performance from an end-user's perspective.

New Relic MOBILE
Quickly pinpoint performance problems in your mobile apps operating in complex productions environments.

New Relic SERVERS
Find health issues to things like CPU utilization, memory utilization, disk I/O utilization, and disk capacity.

New Relic INSIGHTS
Query application business metrics, performance data, and customer behaviors at lightning speed.

New Relic SYNTHETICS
Test and find issues with your software's business critical functionality before real users do.

These links allow you to access the various products that New Relic offers, from the APM product, to mobile app monitoring, Real User Monitoring from the browser, data exploration, and external site monitoring.

See also

You can find further details on the signup process at `https://docs.newrelic.com/docs/accounts-partnerships/accounts/account-setup/create-your-new-relic-account`.

Installing the New Relic Java agent

To gather statistics, New Relic uses an agent that views transactions as they pass through the application. This is then recorded and sent onto New Relic. The Java agent is suitable for use with any Java based application, both inside and outside of a container. In this recipe, we will demonstrate how to setup the New Relic agent within a Tomcat container.

Getting ready

For this recipe, you will need a New Relic account and an application that runs within a Tomcat container.

How to do it...

Let's install the New Relic Java component:

1. Log into your New Relic Account and click on the profile menu in the top-right corner. From the drop-down, select the item title **Account Settings**. On the right-hand side of the next screen, you will find a list of available agents. Click on the **Java release** to download it.

[You'll notice that there is a separate download for the Java 8 installer; keep this in mind when downloading the agent. There are also links to 'established' releases. These are the versions that have the most established user base. Use this one if you are cautious.]

2. Copy the `zip` file containing the Java agent onto the server that you wish to monitor and `unzip` it using the following command:

   ```
   $ unzip -d /opt <pathto_newrelic.zip>
   ```

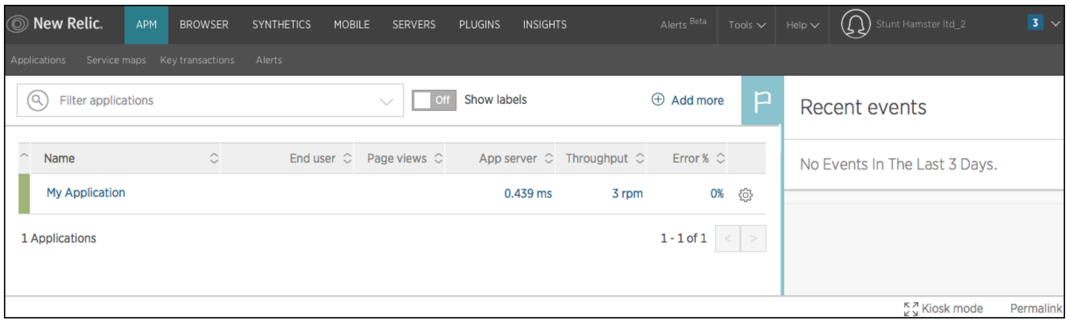
   Where `<newrelic.zip>` is the versioned `zip` file. This will unpack the New Relic agent into the `/opt` directory.

3. With your favorite editor, open up the `setenv.sh` file within your Tomcat application, this will be located within the bin directory of your Tomcat installation. If one does not exist, create it and edit it to resemble the following:

   ```
   JAVA_OPTS="$JAVA_OPTS -javaagent:/opt/newrelic/newrelic.jar"
   ```

[This is based on a vanilla install of Tomcat; if you already have your `JAVA_OPTS` set, ensure that you add the `-javaagent` option in the correct place.]

4. Restart your application. After start-up, pause for around five minutes for New Relic to start logging data and then log into your account. Click on the **APM** menu and you should see a screen similar to the following:



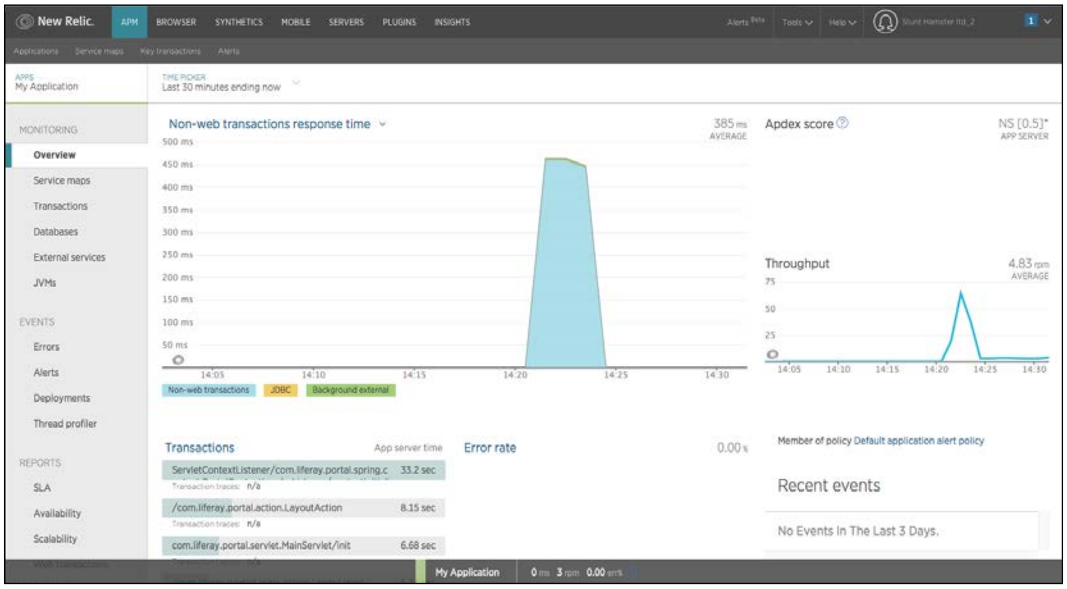
Name	End user	Page views	App server	Throughput	Error %
My Application			0.439 ms	3 rpm	0%

1 Applications 1 - 1 of 1

Recent events

No Events In The Last 3 Days.

5. Click on the **My Application**, and you should see the following screen:



ServletContextListener/com.liferay.portal.spring.c 33.2 sec
/com.liferay.portal.action.LayoutAction 8.15 sec
com.liferay.portal.servlet.MainServlet/init 6.68 sec

Your application is now set up and is recording performance metrics.

> By default, all data is transmitted to a New Relic using SSL and you can obfuscate certain data elements to ensure that sensitive information is not stored within their platform. See `https://docs.newrelic.com/docs/accounts-partnerships/accounts/security/security` for details of what you can secure.

See also

Further details of the Java agent can be found at `https://docs.newrelic.com/docs/agents/java-agent`.

Using the performance overview

New Relic offers a huge amount of information in a readable format; this allows you to see at a glance if your application and dependencies are performing, as you would expect. This data is readily available on the performance overview screen and gives you a window into the detail that New Relic holds about your application. This recipe briefly touches on the major parts of the overview screen and highlights elements that can reveal issues.

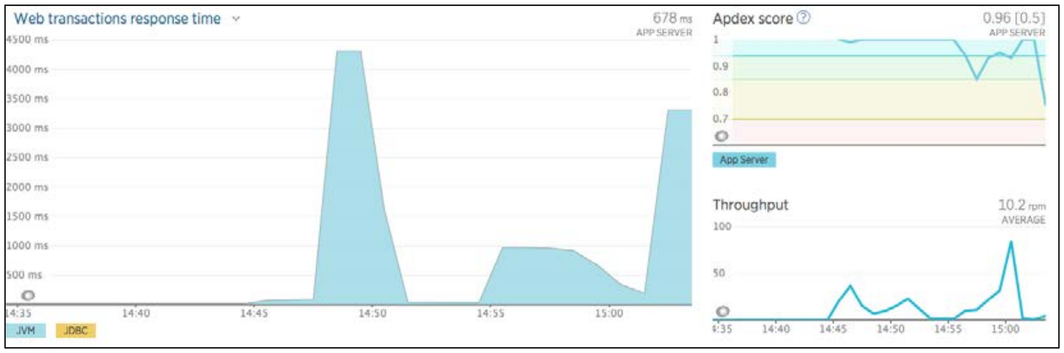
Getting ready

For this recipe, you need a New Relic account and an application reporting data.

How to do it...

Now let's go through the performance overview:

1. Log into your New Relic account and select the **APM** tab; you will be presented with a list of applications that are currently reporting to New Relic. Select the app you wish to examine; this will take you onto the summary screen for that application.
2. The first part to focus on is the main graph and should look a little something like this:



This chart summarizes the following areas:

- Response times of the application split by its components (in this case, the JVM or application code and database).
- The throughput of the application in requests per minute
- The Apdex rating of the application

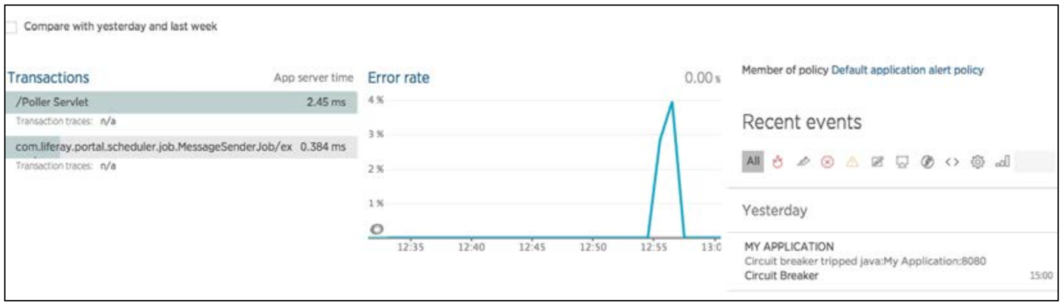
Apdex is a method of taking the measurements that New Relic gathers and presenting them in a single easy to use metric. The Apdex measures user satisfaction, allows you to set an overall goal (the site should respond in n seconds), which can then be used to extrapolate how many of your users are satisfied, tolerating, and frustrated. More details can be found here: `http://www.apdex.org`.

The response time chart is stacked and will include the following:

- JVM (App) response time
- Database response time
- External resources response time

3. Clicking the legend will allow you to strip the elements out; thus, allowing you to focus on a badly behaving component. You can also drag on the chart to zoom in on a particular time period and this will cause any other view you visit to reflect the time selected.
4. Also, note that the chart is split between web and non-web transactions. You can change the view by clicking on the drop-down menu on the top-left of the chart, it should resemble the following screenshot:

5. Underneath the charts, you can see a list of transactions in the selected time period, a chart of the current Error rate, and a complete list of the Recent Events. It should look something like this:



Transactions — App server time

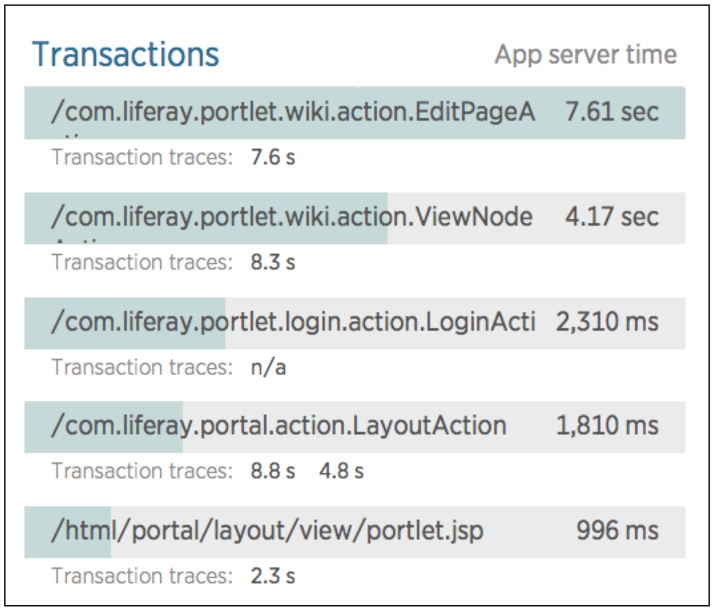
/Poller Servlet 2.45 ms
Transaction traces: n/a

com.liferay.portal.scheduler.job.MessageSenderJob/ex 0.384 ms
Transaction traces: n/a

Member of policy Default application alert policy

MY APPLICATION
Circuit breaker tripped java:My Application:8080
Circuit Breaker 15:00

6. Look first at the **Transactions** list:



Transactions App server time

/com.liferay.portlet.wiki.action.EditPageA 7.61 sec
Transaction traces: 7.6 s

/com.liferay.portlet.wiki.action.ViewNode 4.17 sec
Transaction traces: 8.3 s

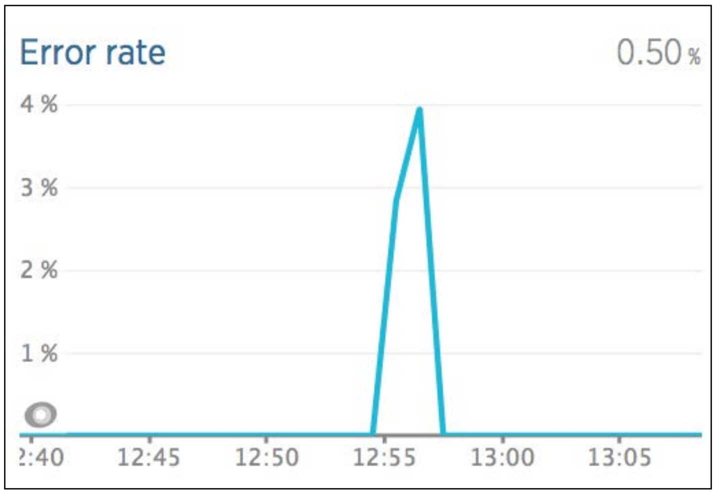
/com.liferay.portlet.login.action.LoginActi 2,310 ms
Transaction traces: n/a

/com.liferay.portal.action.LayoutAction 1,810 ms
Transaction traces: 8.8 s 4.8 s

/html/portal/layout/view/portlet.jsp 996 ms
Transaction traces: 2.3 s

This is a list of the top transactions that the platform has processed in the time period, and the average response time. These can be clicked on, and are linked to the more detailed transaction list, and allows you to see exactly where the time is being spent in these processes.

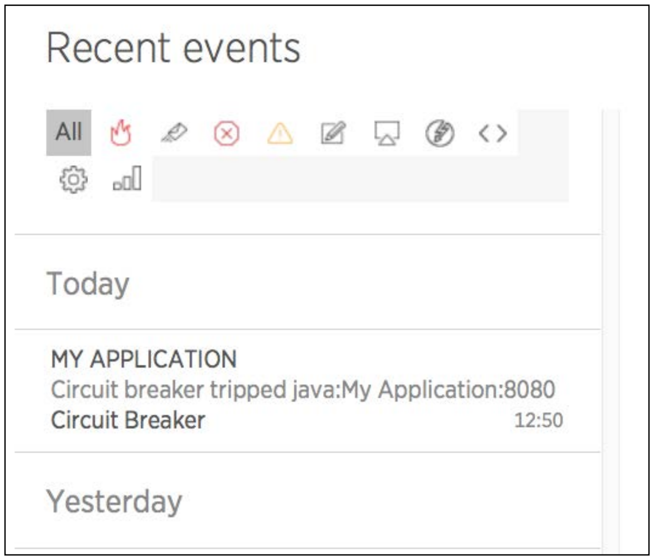
7. Next, we have the **Error rate** chart.



8. Any time an error is logged within the application that generates one of the following response codes, New Relic logs it:
    - 400 Bad Request
    - 401 Unauthorized
    - 403 Forbidden
    - 502 Bad Gateway
    - 503 Service Unavailable

> It is possible within the New Relic settings to exclude certain errors; this can be useful if they are expected, and can stop alerts being generated.

9. Next, we have the **Recent events** list:



Recent events

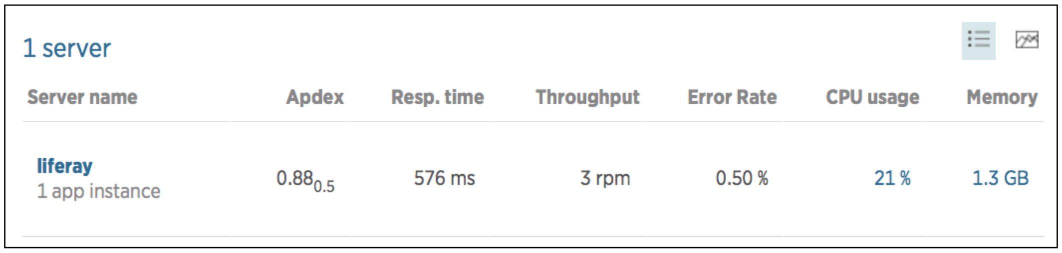
Today

MY APPLICATION
Circuit breaker tripped java:My Application:8080
Circuit Breaker 12:50

Yesterday

Any time a significant event happens; New Relic will record it and display it here. Significant events can be anything from serious application errors, to deployments, to warnings, and offers you a quick way to check if an event has occurred you were unaware of.

10. Finally, we have the server list:



1 server

Server name	Apdex	Resp. time	Throughput	Error Rate	CPU usage	Memory
liferay 1 app instance	$0.88_{0.5}$	576 ms	3 rpm	0.50 %	21 %	1.3 GB

New Relic records details of every server hosting the application in the current time span and offers a brief summary of its throughput, responsiveness, and resource usage. This is linked to the New Relic server monitoring and if the monitoring agent is installed, it allows you to view server-monitoring statistics during this period.

See also

You can find further details of the summary screen at `https://docs.newrelic.com/docs/apm/applications-menu/monitoring/apm-overview-dashboard`.

Locating performance bottlenecks with Transaction Traces

New Relic can dig deep into the reasons as to why an application may not be performing well by performing a Transaction Trace. Transaction Tracing allows for a huge level of details, allowing you to see why a certain transaction is running slow, right down to the individual method calls.

Transaction Tracing is initiated when New Relic notices and records certain transactions that are breaching the Apdex rating. At this point, it starts to sample these transactions, recording in detail what the transaction is doing.

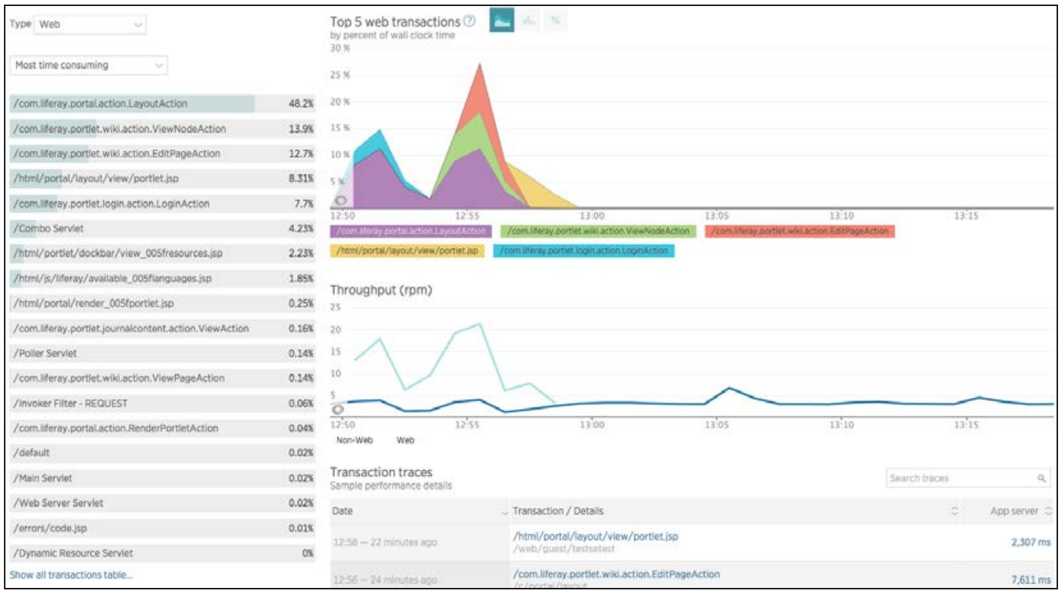
Using the Transaction Traces, you can easily spot where code is less than optimum, highlighting inefficient SQL queries, long running elements of code, and slow external services.

Getting ready

For this recipe, you will require a New Relic account (paid) and an application with the New Relic agent installed.

How to do it...

1. Log into the New Relic portal and click on the **APM** in the top menu. From the left-hand menu on the next screen, select **Transactions**.
2. You should see a screen similar to the following:



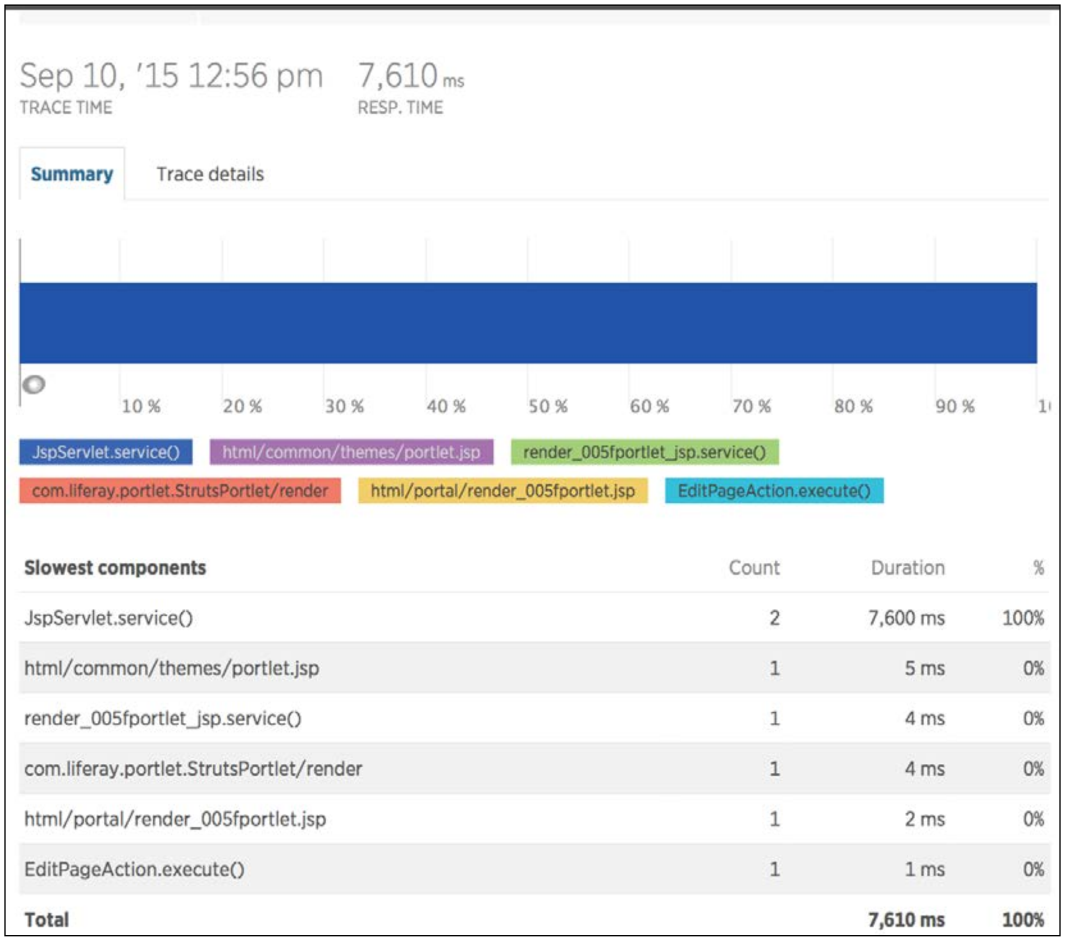
/com.liferay.portal.action.LayoutAction	48.2%
/com.liferay.portlet.wiki.action.ViewNodeAction	13.9%
/com.liferay.portlet.wiki.action.EditPageAction	12.7%
/html/portal/layout/view/portlet.jsp	8.31%
/com.liferay.portlet.login.action.LoginAction	7.7%
/Combo Servlet	4.23%
/html/portlet/dockbar/view_005fresources.jsp	2.23%
/html/js/liferay/available_005flanguages.jsp	1.85%
/html/portal/render_005fportlet.jsp	0.25%
/com.liferay.portlet.journalcontent.action.ViewAction	0.16%
/Poller Servlet	0.14%
/com.liferay.portlet.wiki.action.ViewPageAction	0.14%
/Invoker Filter - REQUEST	0.06%
/com.liferay.portal.action.RenderPortletAction	0.04%
/default	0.02%
/Main Servlet	0.02%
/Web Server Servlet	0.02%
/errors/code.jsp	0.01%
/Dynamic Resource Servlet	0%

Transaction traces
Sample performance details

Date	Transaction / Details	App server
12:58 — 22 minutes ago	/html/portal/layout/view/portlet.jsp /web/guest/testsetest	2,307 ms
12:56 — 24 minutes ago	/com.liferay.portlet.wiki.action.EditPageAction /c/portal/layout	7,611 ms

Notice the table on the bottom left-hand side of the screen; this is a list of Transaction Traces. Select one by clicking on the link.

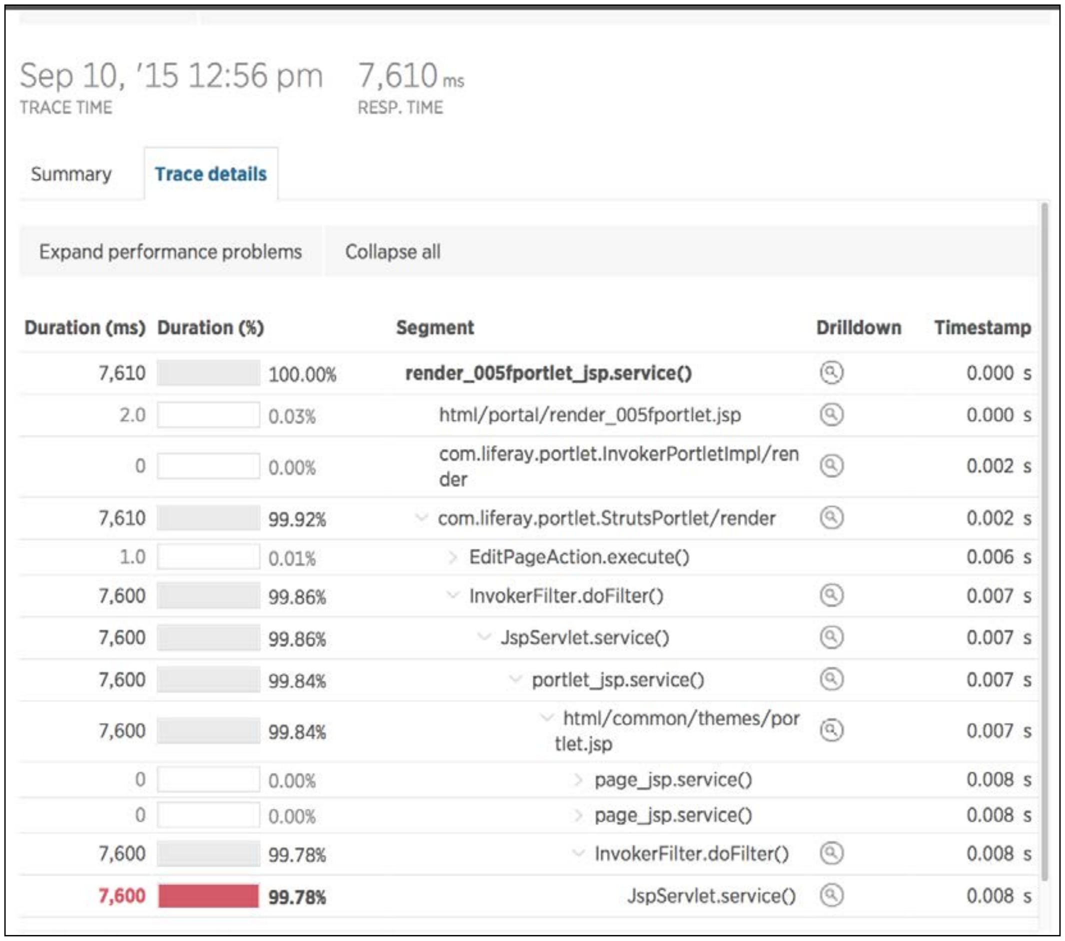
3. When you click on the **Trace**, it opens a new overlaid window with the **Trace details** and it should resemble something like this:



Slowest components	Count	Duration	%
JspServlet.service()	2	7,600 ms	100%
html/common/themes/portlet.jsp	1	5 ms	0%
render_005fportlet_jsp.service()	1	4 ms	0%
com.liferay.portlet.StrutsPortlet/render	1	4 ms	0%
html/portal/render_005fportlet.jsp	1	2 ms	0%
EditPageAction.execute()	1	1 ms	0%
Total		**7,610 ms**	**100%**

This is a break down of the wall clock time of this particular trace, including a graph demonstrating where the time was spent. The list of components lists the slowest elements of this transaction, and most importantly, how many times it was called. This can be useful if you are looping through and calling resources unnecessarily, such as multiple SQL queries that could be collapsed into a single and more efficient query.

4. Click on the tab titled **Trace Details**. This should show a screen similar to the following:

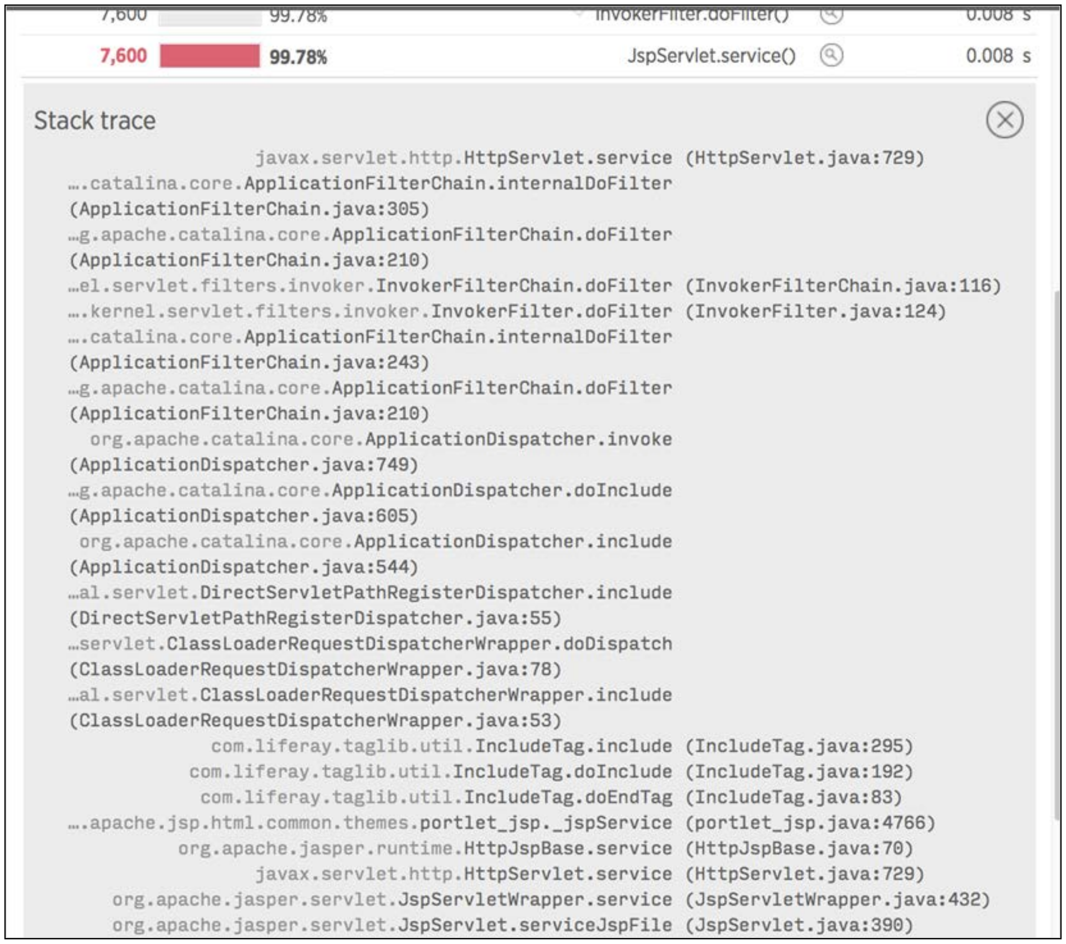


Sep 10, '15 12:56 pm
TRACE TIME

7,610 ms
RESP. TIME

Duration (ms)	Duration (%)	Segment	Drilldown	Timestamp
7,610	100.00%	**render_005fportlet_jsp.service()**	🔍	0.000 s
2.0	0.03%	html/portal/render_005fportlet.jsp	🔍	0.000 s
0	0.00%	com.liferay.portlet.InvokerPortletImpl/render	🔍	0.002 s
7,610	99.92%	com.liferay.portlet.StrutsPortlet/render	🔍	0.002 s
1.0	0.01%	EditPageAction.execute()		0.006 s
7,600	99.86%	InvokerFilter.doFilter()	🔍	0.007 s
7,600	99.86%	JspServlet.service()	🔍	0.007 s
7,600	99.84%	portlet_jsp.service()	🔍	0.007 s
7,600	99.84%	html/common/themes/portlet.jsp	🔍	0.007 s
0	0.00%	page_jsp.service()		0.008 s
0	0.00%	page_jsp.service()		0.008 s
7,600	99.78%	InvokerFilter.doFilter()	🔍	0.008 s
7,600	**99.78%**	JspServlet.service()	🔍	0.008 s

5. This is a list of the methods that the transaction called and the wall clock time that each method took. Notice that New Relic has highlighted the element of code that is problematic. We can drill into this by clicking on the magnifying glass icon; this will then expand the method to show the stack trace, allowing you to examine in greater detail what this piece of code was doing; it should look something like this:



```
7,600   99.78%                InvokerFilter.doFilter()     0.008 s
7,600   99.78%                JspServlet.service()         0.008 s

Stack trace
                    javax.servlet.http.HttpServlet.service (HttpServlet.java:729)
   ….catalina.core.ApplicationFilterChain.internalDoFilter
   (ApplicationFilterChain.java:305)
   …g.apache.catalina.core.ApplicationFilterChain.doFilter
   (ApplicationFilterChain.java:210)
   …el.servlet.filters.invoker.InvokerFilterChain.doFilter (InvokerFilterChain.java:116)
   ….kernel.servlet.filters.invoker.InvokerFilter.doFilter (InvokerFilter.java:124)
   ….catalina.core.ApplicationFilterChain.internalDoFilter
   (ApplicationFilterChain.java:243)
   …g.apache.catalina.core.ApplicationFilterChain.doFilter
   (ApplicationFilterChain.java:210)
     org.apache.catalina.core.ApplicationDispatcher.invoke
   (ApplicationDispatcher.java:749)
   …g.apache.catalina.core.ApplicationDispatcher.doInclude
   (ApplicationDispatcher.java:605)
    org.apache.catalina.core.ApplicationDispatcher.include
   (ApplicationDispatcher.java:544)
   …al.servlet.DirectServletPathRegisterDispatcher.include
   (DirectServletPathRegisterDispatcher.java:55)
   …servlet.ClassLoaderRequestDispatcherWrapper.doDispatch
   (ClassLoaderRequestDispatcherWrapper.java:78)
   …al.servlet.ClassLoaderRequestDispatcherWrapper.include
   (ClassLoaderRequestDispatcherWrapper.java:53)
                com.liferay.taglib.util.IncludeTag.include (IncludeTag.java:295)
              com.liferay.taglib.util.IncludeTag.doInclude (IncludeTag.java:192)
               com.liferay.taglib.util.IncludeTag.doEndTag (IncludeTag.java:83)
   ….apache.jsp.html.common.themes.portlet_jsp._jspService (portlet_jsp.java:4766)
            org.apache.jasper.runtime.HttpJspBase.service (HttpJspBase.java:70)
                    javax.servlet.http.HttpServlet.service (HttpServlet.java:729)
       org.apache.jasper.servlet.JspServletWrapper.service (JspServletWrapper.java:432)
       org.apache.jasper.servlet.JspServlet.serviceJspFile (JspServlet.java:390)
```

See also

You can find further details of Transaction Tracing at `https://docs.newrelic.com/docs/apm/transactions/transaction-traces/transaction-traces`.

Observing database performance with New Relic

Many, if not most, modern applications are backed by a SQL database, generally providing a persistent data storage. The performance of this data store is critical to the overall application performance and yet it can be somewhat opaque to view.

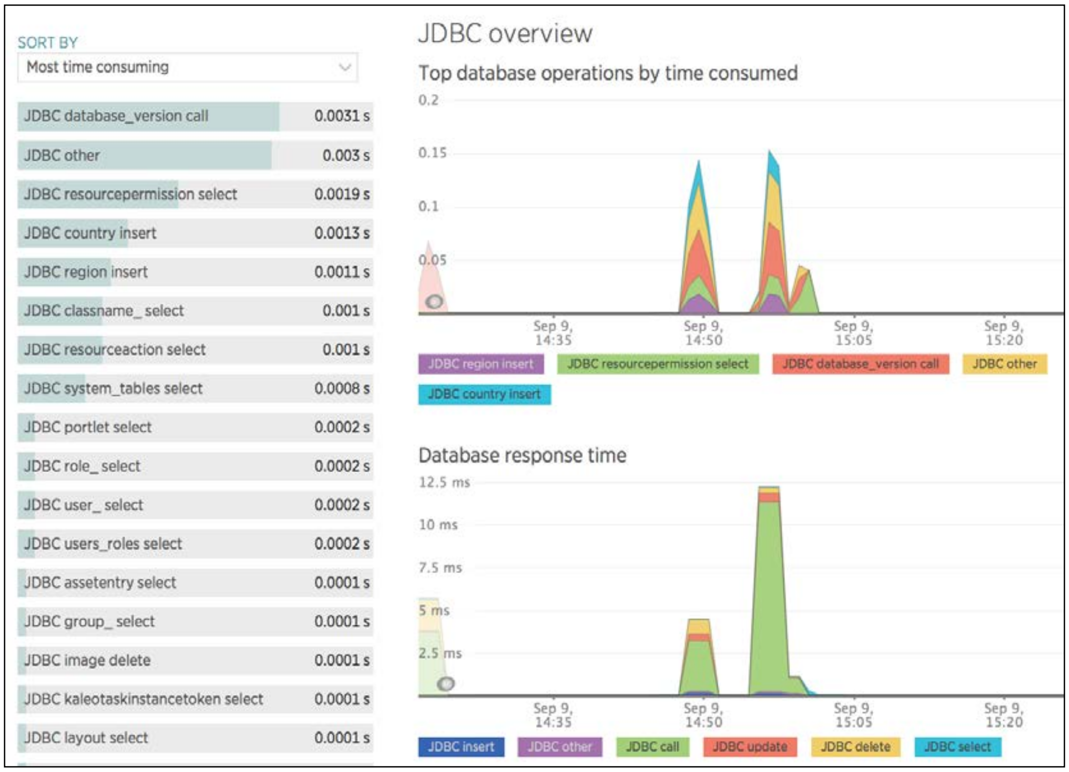
New Relic tracks database calls, allowing you to see both the overall performance of the database and the details of where the database calls originate within the application and where it may be running slowly.

Getting ready

For this recipe, you will need a New Relic account (trial or paid), an application deployed with the New Relic agent, and a database that the application connects to (and a JDBC compatible database should work).

How to do it...

1. Log into your New Relic account and click on the menu item titled **APM**. From the next screen, select the link on the left-hand sidebar titled **Databases**. This should present you with a screen that looks similar to the following screenshot:

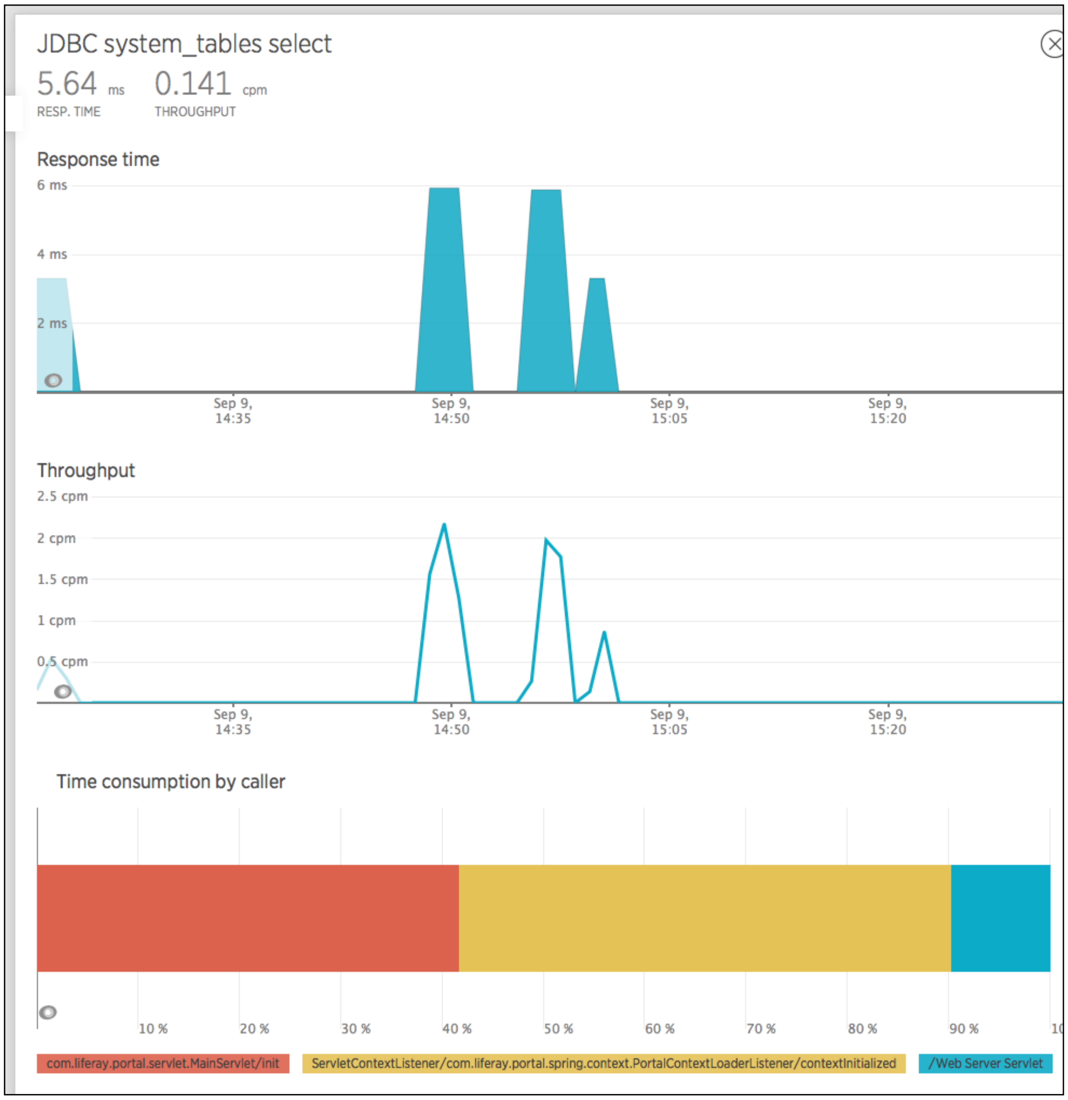


SORT BY
Most time consuming

JDBC database_version call	0.0031 s
JDBC other	0.003 s
JDBC resourcepermission select	0.0019 s
JDBC country insert	0.0013 s
JDBC region insert	0.0011 s
JDBC classname_ select	0.001 s
JDBC resourceaction select	0.001 s
JDBC system_tables select	0.0008 s
JDBC portlet select	0.0002 s
JDBC role_ select	0.0002 s
JDBC user_ select	0.0002 s
JDBC users_roles select	0.0002 s
JDBC assetentry select	0.0001 s
JDBC group_ select	0.0001 s
JDBC image delete	0.0001 s
JDBC kaleotaskinstancetoken select	0.0001 s
JDBC layout select	0.0001 s

2. This is a list of every database call made during the selected time period and it graphs the duration of the top calls and response time of the database operations and finally a throughput of the database.
3. You can easily sort between the most time consuming and the slowest response time throughput by clicking on the Sort by menu at the top of the transaction list. This enables you to quickly zero in on interesting data on these three axis.

4. When you find a transaction that interests you, click on it in the list. This will open the detailed transaction view and should resemble the following screenshot:



This gives you a breakdown of that particular transaction and highlights the response over time alongside throughput. Of particular interest is the bottom graph; this illustrates the most common transactions that are calling this an SQL query, charted by the overall time spent being called by that particular transaction. These are linked to the transaction view; thus, allowing you to see detailed information about that particular transaction.

See also

You can find further details of the New Relic SQL monitoring at `https://docs.newrelic.com/docs/apm/applications-menu/monitoring/databases-slow-queries-dashboard`.

Release performance monitoring with New Relic

One of the most important events to track in any monitoring is a release; this can be anything from a software release, through to server upgrades and anything in between. New Relic allows you to add these as reportable elements, allowing you to perform before and after performance reporting. Using the release monitoring you can take the guesswork out of the effectiveness of a release and this allows you to use empirical data to judge if a release was good.

New Relic offers an API to record a release, allowing you to integrate your automated release tools, configuration management, and others. Using the New Relic API, you should be able to ensure that any automated process can register the release.

In this recipe, we will examine using the New Relic REST API to trigger a deployment notification using a simple curl from the command line.

Getting ready

For this recipe, you will need a New Relic account (free/trial/paid) and an application with the New Relic agent installed.

How to do it…

1. First, we need to enable the API access. Log onto your New Relic account and click on the profile menu located in the menu on the top-left. From the drop down menu, select the entry titled **Account settings** and in the next screen, click on the link in the left-hand sidebar titled **Data sharing**.
2. In the data-sharing screen, you will be offered the option to create a new **REST API Key**. Click on this link and a new API key will be generated and API access to the account will be granted.
3. From your command line, use the following command to record a release against your application:

```
curl -H "x-api-key:<APIKEY>" \
-d "deployment[application_id]=<APPLICATION ID>" \
-d "deployment[description]=<DEPLOYMENT DESCRIPTION>" \
-d "deployment[revision]=<NEW APP VERSION>" \
-d "deployment[changelog]=<LIST OF CHANGES>" \
-d "deployment[user]=Michael Duffy" https://api.newrelic.com/
deployments.xml
```

Ensure that you replace the elements between the angle brackets with your own items.

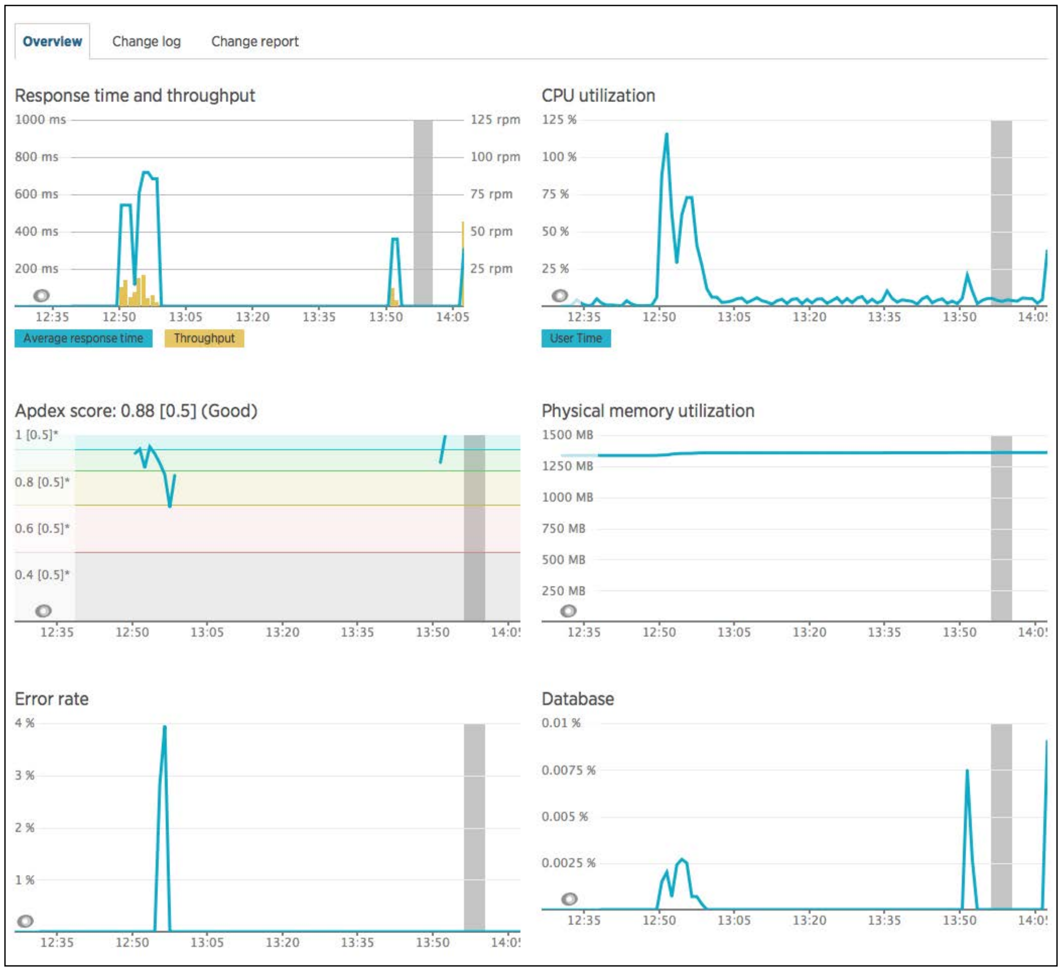
> You can find the **Application ID** within the deployments tab, underneath the link titled **Show instructions**.

4. If all goes well, you should receive an XML reply containing a summary of elements prior to release; it will resemble something along these lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment>
  <account-id type="integer">1078425</account-id>
  <agent-id type="integer">8699096</agent-id>
  <avg-apdex-f type="integer" nil="true"></avg-apdex-f>
  <avg-apdex-s type="integer" nil="true"></avg-apdex-s>
  <avg-apdex-t type="integer" nil="true"></avg-apdex-t>
  <avg-apdex-threshold type="float" nil="true"></avg-apdex-
threshold>
  <avg-cpu type="float">0</avg-cpu>
  <avg-db type="float">0</avg-db>
  <avg-enduser-apdex-f type="integer" nil="true"></avg-enduser-
apdex-f>
  <avg-enduser-apdex-s type="integer" nil="true"></avg-enduser-
apdex-s>
  <avg-enduser-apdex-t type="integer" nil="true"></avg-enduser-
apdex-t>
  <avg-enduser-apdex-threshold type="integer" nil="true"></avg-
enduser-apdex-threshold>
  <avg-enduser-rt type="float">0.0</avg-enduser-rt>
  <avg-enduser-throughput type="float">0.0</avg-enduser-
throughput>
  <avg-errors type="float">0.0</avg-errors>
  <avg-memory type="float">1359.3242187499998</avg-memory>
  <avg-rt type="float">0.0</avg-rt>
  <avg-throughput type="float">0.0</avg-throughput>
  <changelog>I added many interesting things to the code.</
changelog>
  <description>A shiny deployment</description>
  <end-time type="datetime" nil="true"></end-time>
  <id type="integer">8908958</id>
```

```
    <revision>1f1f1</revision>
    <timestamp type="datetime">2015-09-10T13:59:28-07:00</timestamp>
    <user>Michael Duffy</user>
  </deployment>
```

5. Log into your New Relic account and click on the item titled **APM** from the top menu. From the left-hand menu on the next screen, click on the item marked Deployments under the events heading.
6. The next screen offers a list of deployments and a brief summary of the **Apdex score**, **Response time and throughput**. If you click on one of the deployments, you should see a screen similar to the following:



This details the performance prior to the release and after, with the grey bar marking the point of release.

See also

You can find further details of release monitoring at `https://docs.newrelic.com/docs/release-notes/agent-release-notes/java-release-notes`.

Server Monitoring with New Relic

New Relic is made up of several different products and combined allow you to see end-to-end performance metrics of every part of your application. For instance, if the APM-panel tells you, there are performance issues; one of the first things you should do is look at the performance of the underlying server.

New Relic offers a simple, yet easy to use server monitoring tool, allowing you to monitor your servers alongside your applications. Although lacking some of the sophistication of other monitoring packages, it has the advantage that it is closely tied to APM monitoring. If you are looking at a particular time span in the APM, you can check the server monitoring and have it automatically adjust the time-range to match your app. New Relic can also map apps to servers, allowing you see at a glance, which app is running on which server.

Finally, at the time of writing, Server Monitoring is a free component and you can add as many servers as you like; they do not need to have applications that are monitored within the APM. This allows you to monitor your app servers, but also dependencies, such as database servers and file servers within the same place.

Getting ready

For this recipe, you will require an Ubuntu 14.04 server and a New Relic account.

How to do it...

1. On the host on which you wish to install the New Relic server monitoring, run the following command:

   ```
   $ sudo echo deb http://apt.newrelic.com/debian/ newrelic non-free
   >> /etc/apt/sources.list.d/newrelic.list && wget -O- https://
   download.newrelic.com/548C16BF.gpg | apt-key add -
   ```

2. This adds the New Relic package repository and adds the signing key. Next, run `apt-get update` using the following command:

   ```
   $ sudo apt-get update
   ```

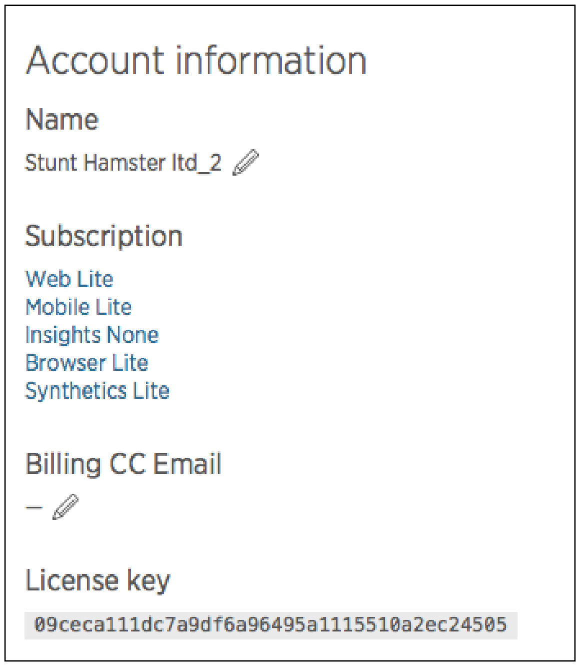
3. You can install the New Relic server monitoring package using the following command:

```
apt-get install newrelic-sysmond
```

4. Although the package is installed, you still need to configure it to send data to your New Relic account. To do this, use the following command:

```
$ sudo nrsysmond-config --set license_key=<LICENSE>
```

   Replace `<LICENSE>` with your New Relic account license. You can find this by clicking on the profile menu on the top-right of the New Relic panel and by clicking on **Account Settings**. On the next screen, you should find a panel on the right-hand side that resembles the following screenshot:



Account information

Name

Stunt Hamster ltd_2

Subscription

Web Lite
Mobile Lite
Insights None
Browser Lite
Synthetics Lite

Billing CC Email

–

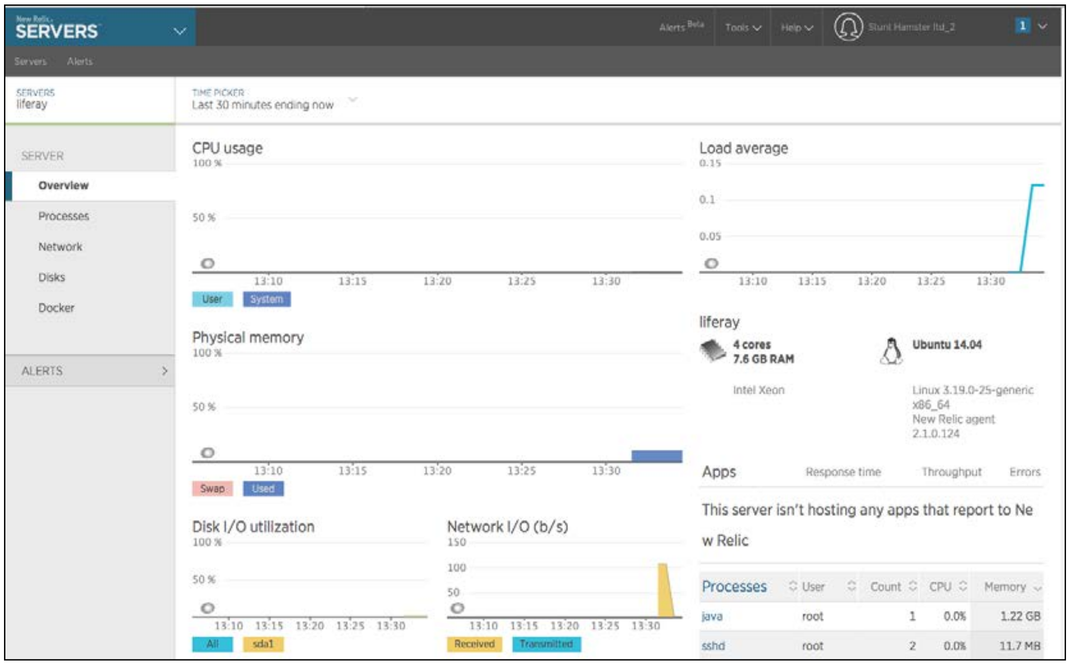
License key

09ceca111dc7a9df6a96495a1115510a2ec24505

5. Finally, you can start the New Relic server monitor by issuing the following command:

```
$ sudo service newrelic-sysmond start
```

6. You should wait for around five minutes for initial data to be propagated to New Relic, but once you have, you can log into your New Relic account and click on servers. You should be able to see the server that you added to the list, and if you click on it, you should see something similar to the following screenshot:



This server isn't hosting any apps that report to New Relic

Processes	User	Count	CPU	Memory
java	root	1	0.0%	1.22 GB
sshd	root	2	0.0%	11.7 MB

From here, you can explore details of the process running on the server, network, and disk and, if you use it, Docker usage.

See also

You can find more details of the New Relic server monitoring solution at `http://newrelic.com/server-monitoring`.

Module 3

Continuous Delivery and DevOps – A Quickstart Guide - Second Edition

Deliver quality software regularly and painlessly by adopting CD and DevOps

1 Evolution of a Software House

As described in the *Preface*, both **Continuous Delivery** (**CD**) and **DevOps** are complementary ways of working. The former assists with shipping quality software quickly, the latter helps harmonize the teams that deliver and support said software. Both approaches can help you to optimize, streamline, and improve the way you work. Ultimately, both will help you ship quality software.

Before we get onto the meat of CD and DevOps, let me introduce you to **ACME systems**—a typical software business—and walk you through their trials, tribulations, and evolution. The topics we will cover in this chapter are as follows:

- How ACME systems started from humble beginnings
- The growing pains it went through to become successful
- The positives and negatives that came from success and dramatic growth
- The advantages that came with adopting CD and DevOps ways of working
- How it adapted to utilize what it had learned to drive their business into new markets and opportunities

Without further ado, let's meet ACME systems.

A brief history of ACME systems

This fictional software business started out—as many successful tech companies do—in the garage of one of the founders. The founders were visionaries with big ambitions, good ideas, and a little bit of cash.

After a few years of hard work, determination, much blood, sweat, and tears, the dreams of the founders were realized. The business is recognized as a leader in its field and is then acquired by a multinational corporate. This acquisition brings with it the funding and resources needed to allow the business to grow and expand to become a global player. However, with corporate owners comes corporate responsibilities, rules, bureaucracy, and processes.

The ACME systems team start to find it increasingly difficult and painful to deliver quality software. They adopt and adhere to the parent company's processes to improve quality and reduce risk, but this makes the seemingly simple task of delivering software, laborious and extremely complex.

They come to an evolutionary crossroad and have to make a decision either to live with the corporate baggage that they have inherited and potentially face extinction, or try and get back to the good old days and good old ways that had reaped rewards previously.

While trying to decide which way to go, they discover they have another choice —implement CD and DevOps—which could give them the best of both worlds. As luck would have it that is exactly what they did.

Over the next few pages, we'll go through this evolution in a little more detail. As we do, you may recognize some familiar traits and challenges.

[The name ACME is purely fictional and based upon the ACME Corporation, first used in *Road Runner Cartoons* in the 1950s—just in case you were wondering.]

We'll start with the initial incarnation of the ACME systems business, which for want of a better name, will be called ACME systems version 1.0.

ACME systems version 1.0

Some of you have most probably worked for (or currently work for) a small software business. There are many such businesses scattered around the globe and they all have one thing in common—they need to move fast to survive and they need to entice and retain customers at all costs. They do this by delivering what the customer wants just before the customer needs it. Deliver too soon and you may have wasted money on building solutions that the customer decides they no longer need, as their priorities or simply their minds have changed. Deliver too late and someone else may well have taken your customer—and more importantly, your revenue—away from you. The important keyword here is *deliver*.

As mentioned earlier, ACME systems started out in humble beginnings; the founders had a big vision and could see a gap in the market for a web-based solution. They had an entrepreneurial spirit and managed to attract backers who injected the lifeblood of all small businesses—cash.

They then went about sourcing some local, keen, and talented engineers and set about building the web-based solution that bridged the gap in the market, which they had seen before anyone else could.

At first, the going was slow and the work was hard; a lot of pre-sales prototypes needed to be built in a hurry—most of which never saw the light of day—some went straight into production. After many long days, nights, weeks, and weekends, things started to come together. Their customer base started to grow and the orders started rolling in; as did the revenue. Soon the number of employees was in double figures and the founders had become directors.

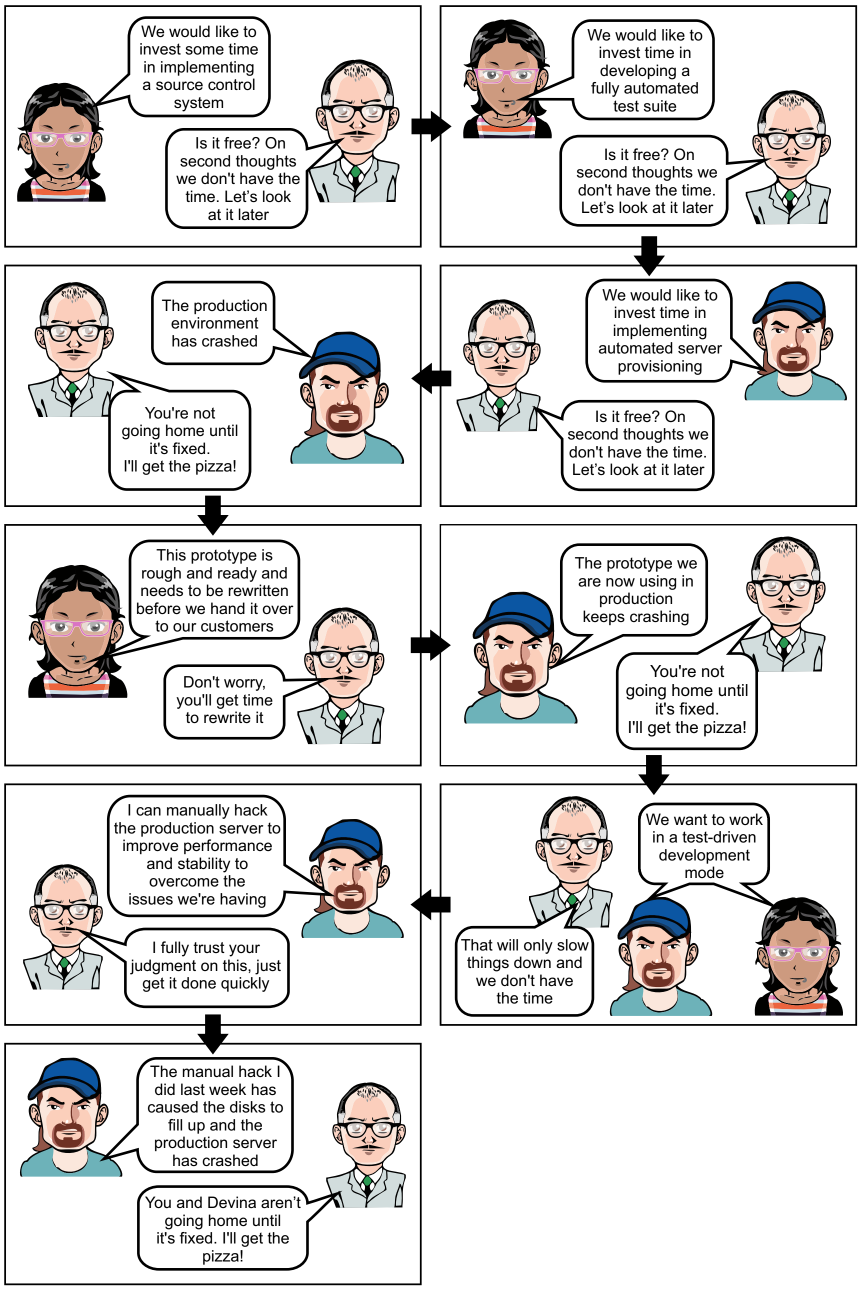
So, I hear you ask, "What has this got to do with CD or DevOps?" Well, everything really. The culture, default behaviors, and engineering practices of a small software house are what would be classed as pretty good in terms of CD and DevOps.
For example:

- There are next to no barriers between developers and operations teams—in fact, they are generally one and the same
- Developers normally have full access to the production environment and can closely monitor their software
- All areas of the business are focused on the same thing, that being to get software into the production environment as quickly as possible and thus delight customers
- Speed of delivery is of the essence
- When things break, everyone swarms around to help fix the problem—even out of hours
- The software evolves quickly and features are added in incremental chunks
- The ways of working are normally very agile

There is a reason for stating that the culture, default behaviors, and engineering practices of a small software house would be classed as *pretty good* rather than *ideal*. This is because there are many flaws in the way a small software house typically has to operate to stay alive; for example:

- Corners will be cut to hit deadlines, which compromises software design and elegance
- Application security best practice is given short shrift or even ignored
- Engineering best practices are compromised to hit deadlines
- The concept of technical debt is pretty much ignored
- Testing is not in the forefront of the developer's mind and even if it were, there may not be enough time to work in a test-driven development way
- Source and version control systems are not used religiously
- With unrestricted access to the production environment, tweaks and changes can be made to the infrastructure with little or no audit trail
- Software releasing will be mainly manual and most of the time an afterthought of the overall system design
- At times, a rough and ready prototype may well become production code without the opportunity for refactoring
- Documentation is scant or nonexistent—that which does exist is most probably out of date
- The work-life balance for an engineer working within a small software house is not sustainable and *burn out* does happen

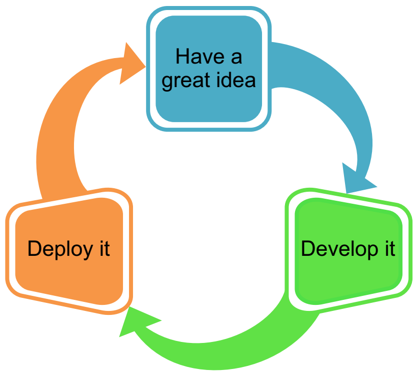
To emphasize this, let's have a look at a selection of typical conversations between three individuals within the ACME systems team: Stan, the manager; Devina, the developer; and Oscar, the operations guy.


We would like to invest some time in implementing a source control system
Is it free? On second thoughts we don't have the time. Let's look at it later
We would like to invest time in developing a fully automated test suite
Is it free? On second thoughts we don't have the time. Let's look at it later
We would like to invest time in implementing automated server provisioning
Is it free? On second thoughts we don't have the time. Let's look at it later
The production environment has crashed
You're not going home until it's fixed. I'll get the pizza!
This prototype is rough and ready and needs to be rewritten before we hand it over to our customers
Don't worry, you'll get time to rewrite it
The prototype we are now using in production keeps crashing
You're not going home until it's fixed. I'll get the pizza!
We want to work in a test-driven development mode
That will only slow things down and we don't have the time
I can manually hack the production server to improve performance and stability to overcome the issues we're having
I fully trust your judgment on this, just get it done quickly
The manual hack I did last week has caused the disks to fill up and the production server has crashed
You and Devina aren't going home until it's fixed. I'll get the pizza!

We'll now have a look at the software delivery process for ACME systems version 1.0, which, to be honest, shouldn't take too long.

Software delivery process flow version 1.0

The following diagram gives an overview of the simple process used by ACME systems to deliver software. It's simple, elegant (in a rough-and-ready kind of way), and easy to communicate and understand.



An overview of the ACME systems version 1.0 software delivery process

Let's move forward a few years and see how ACME systems is doing and gain some insight into the benefits and pitfalls of being the leader in the field.

ACME systems version 2.0

The business has grown in size and turnover. The customer base is now global and the ACME systems software platform is being used by millions of customers on a daily basis. ACME systems is well established, well renowned, and recognized as being at the forefront in its area of expertise.

So much so that the board of ACME systems is approached by a multinational corporation and discussions are entered into regarding an acquisition. These discussions don't take long and the acquisition is completed within weeks. The board members are extremely happy, and the business as a whole sees this as a positive recognition that they have at last reached the big time.

At first, everything is good; everything is great in fact. The ACME systems team now has the backing they need to invest in the business and be able to scale out and obtain a truly global reach. They can also focus on the important things such as building quality software; scaling out the software platform; and investing in new technologies, tools, and R&D. The drier side of business—administration, program, project management, sales, marketing, and so on—can be passed to the new parent company that has all of this in place already.

The ACME systems team moves forward in excited expectation. The level of investment is such that the software engineering team doubles in size in a matter of months. The R&D team—as they're now called—introduces new tools and processes to enable speedy delivery of quality software. Scrum is adopted across the R&D team and the opportunity to fully exploit engineering best practices is realized. The original ACME systems platform starts to creak and is showing its age, so further investment is provided to re-architect and rewrite the software platform using the latest technologies. In short, the R&D team feels that it's all starting to come together and they have the opportunity to do it right.

In parallel to this, the ACME systems operations team is absorbed into the parent's global operations organization. On the face of it, this seems a very good idea; there are data centers filled with cutting-edge kit, global network capabilities, and scalable infrastructure. Everything that is needed to host and run the ACME systems platform is there. Like the R&D team, the operations team has more than they could have dreamed of. In addition to the tin and string, the operations team also has resources available to help maintain quality, control change to the platform, and ensure the platform is stable and available 24/7.

Sitting above all of this, the parent company also has well-established governance, program, and project management functions to control and coordinate the overall end-to-end product delivery schedule and process.

On the face of it, everything seems rosy and the teams are working more effectively than ever before. At first, this is true, but very soon, things start to take a downward turn. Under the surface, things are not that rosy at all.

We'll shift forward another year or so and see how things are:

- It is becoming increasingly difficult to ship software—what took days, now takes weeks or even months
- Releases are getting more and more complex as the new platform grows and more integrated features are added
- Despite the advances in re-architecting and rewriting the platform, there still remains large sections of legacy code deep within the bowels of the system, which refuse to die
- Developers are now far removed from the production environment and as such are ignorant as to how the software they are writing performs, once it eventually goes live
- There is a greater need to provide proof that software changes are of the highest quality and performance before they can go anywhere near the production servers

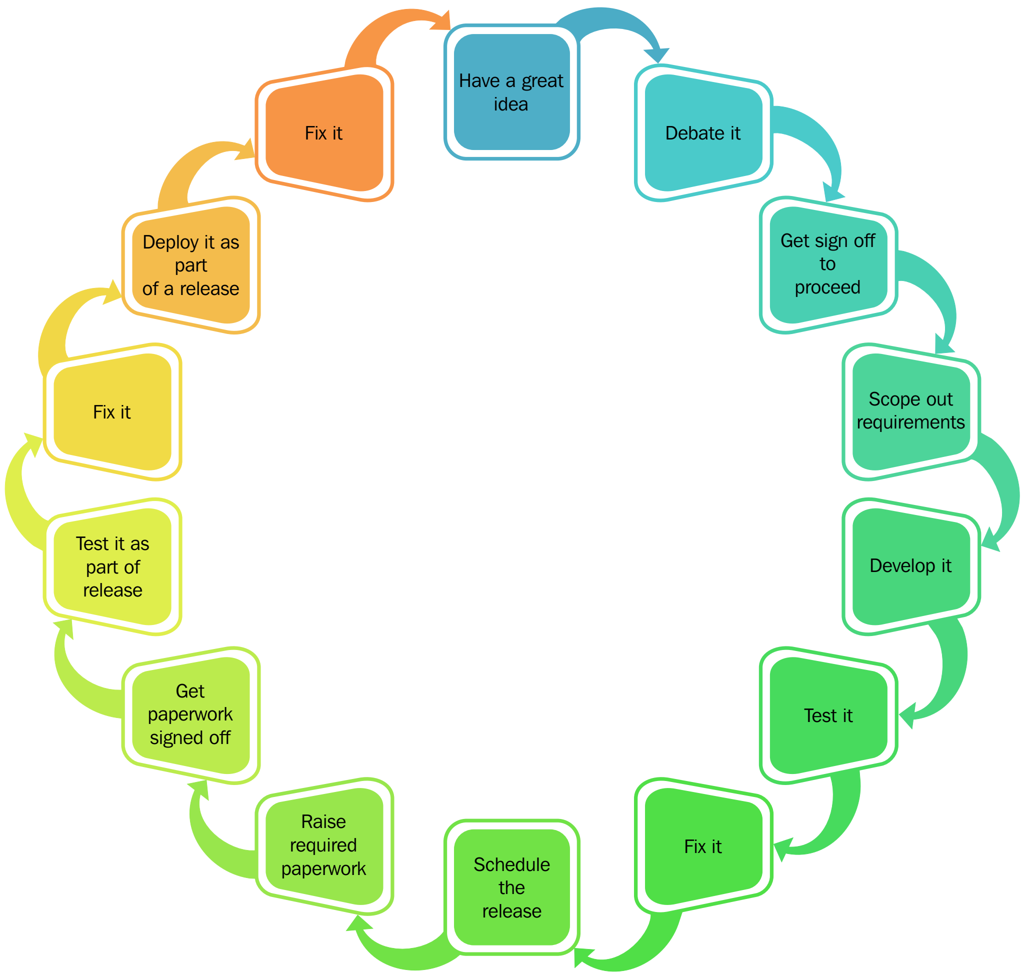
- Quality is starting to suffer as last minute changes and frantic bug fixes are being applied to fit into release cycles
- The technical debt amassed during the *fast and loose* days is starting to cause major issues
- Project scope is being cut at the last minute as features don't fit into the release cycles, which is leaving lots of redundant code lying around
- More and more development resources are being applied to assisting releases, which is impacting on the development of new features
- Deployments are causing system downtime—planned and unplanned
- Deadlines are being missed, stakeholders are being let down, and trust is being eroded
- The business's once glowing reputation is being tarnished

The main problem here, however, is that this attrition has been happening very slowly over a number of months and not everyone has noticed—they're all too busy trying to deliver.

Let's now revisit the process flow for delivering software and see what's changed—it's not a pretty picture.

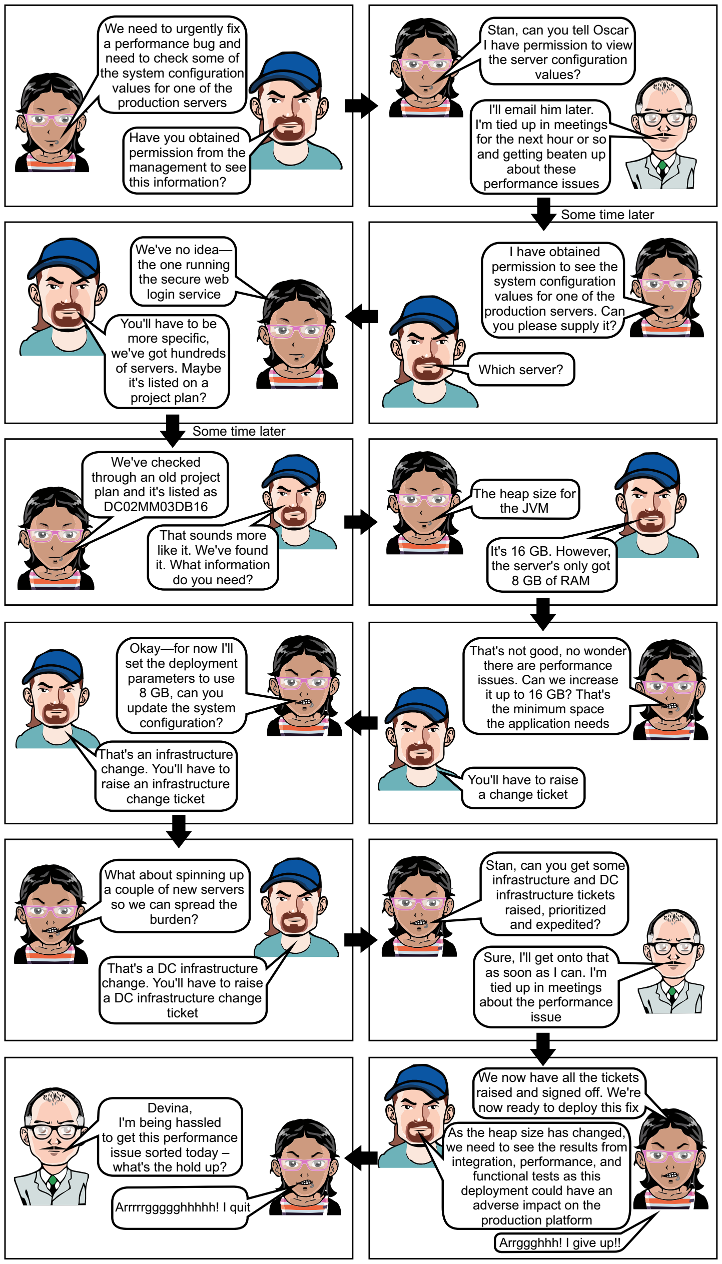
Software delivery process flow version 2.0

As you can see from the following diagram, things have become very complicated for the ACME systems team. What was simple and elegant has become complex, convoluted, and highly inefficient. The number of steps and barriers have increased, making it extremely difficult to get software delivered. In fact, it's increasingly difficult to get anything done. The following diagram gives you an overview of the ACME systems version 2.0 software delivery process:



An overview of the ACME systems version 2.0 software delivery process

Not only has the process become very inefficient—and to all intents and purposes broken—but the dialogue and the quality of the communication have also broken down. Let's again review a typical discussion between Devina, Oscar, and Stan regarding a live issue.


We need to urgently fix a performance bug and need to check some of the system configuration values for one of the production servers
Have you obtained permission from the management to see this information?
Stan, can you tell Oscar I have permission to view the server configuration values?
I'll email him later. I'm tied up in meetings for the next hour or so and getting beaten up about these performance issues
Some time later
I have obtained permission to see the system configuration values for one of the production servers. Can you please supply it?
Which server?
We've no idea— the one running the secure web login service
You'll have to be more specific, we've got hundreds of servers. Maybe it's listed on a project plan?
Some time later
We've checked through an old project plan and it's listed as DC02MM03DB16
That sounds more like it. We've found it. What information do you need?
The heap size for the JVM
It's 16 GB. However, the server's only got 8 GB of RAM
That's not good, no wonder there are performance issues. Can we increase it up to 16 GB? That's the minimum space the application needs
You'll have to raise a change ticket
Okay—for now I'll set the deployment parameters to use 8 GB, can you update the system configuration?
That's an infrastructure change. You'll have to raise an infrastructure change ticket
What about spinning up a couple of new servers so we can spread the burden?
That's a DC infrastructure change. You'll have to raise a DC infrastructure change ticket
Stan, can you get some infrastructure and DC infrastructure tickets raised, prioritized and expedited?
Sure, I'll get onto that as soon as I can. I'm tied up in meetings about the performance issue
We now have all the tickets raised and signed off. We're now ready to deploy this fix
As the heap size has changed, we need to see the results from integration, performance, and functional tests as this deployment could have an adverse impact on the production platform
Arrggghhh! I give up!!
Devina, I'm being hassled to get this performance issue sorted today – what's the hold up?
Arrrrrggggghhhhh! I quit

Okay, so this might be a little over the top, but it just serves to highlight the massive disjoint between the R&D and Operations team(s)—who you'll remember were pretty much one and the same in the early days of ACME systems. It should also be noted that this communication is now normally done via e-mail.

A few brave men and women

As was previously stated, not everyone noticed the attrition within the organization—luckily a few brave souls did. A small number of the ACME systems team are able to see the issues within the overall process as clear as day and they become determined to expose them and, more importantly, sort them out—it is just a question of how to do this while everyone is going at full pelt to get software delivered at all costs.

At first, they seek out a like-minded manager who has influence within the business and helps them to form a small virtual team. They then start identifying and breaking down the immediate issues and go about implementing the following tooling to ease some of the pain:

- Build and test automation
- **Continuous Integration** (**CI**)
- Automated deployment and monitoring solutions

This goes some way to address the issues but there are still some fundamental problems that tooling cannot address—the culture of the organization itself and the many disjointed silos within it. It becomes obvious that all the tools and tea in China will not bring pain relief; something more drastic is needed.

The team refocuses and works to highlight this now obvious fact to as many people as they can up and down the organization, while the influential manager works to obtain backing from the senior leadership to address it—which luckily is forthcoming.

We're now going on to the third stage of the evolution where things start to come back together and the ACME systems team regains their ability to deliver quality software when it is needed.

ACME systems version 3.0

The CD team—as they are now called—gets official backing from up high and becomes dedicated to addressing the problematic culture and behaviors, and developing ways to overcome and/or remove the barriers. They are no longer simply a technical team; they are a catalyst for change.

The remit is clear—do whatever is needed to streamline the process of software delivery and make it seamless and repeatable. In essence, implement what we now commonly refer to as CD and DevOps.

The first thing they do is to simply talk with as many people across the business as possible. If someone is involved in the process of getting software from conception to consumer and support it when it's live, they are someone you need to speak with. This not only gathers useful information but also gives the team the opportunity to evangelize and form a wider network of like-minded individuals.

The team has a vision, a purpose, and they passionately believe in what needs to be done, and have the energy and drive to do it.

Over the next few months, they embark on (among other things):

- Running various in-depth sessions to understand and map out the end-to-end product delivery process
- Refining and simplifying tooling based upon continuous feedback from those using it
- Addressing the complexity of managing dependencies and order of deployment
- Engaging experts in the field of CD to independently assess the progress being made (or not as the case may be)
- Arranging offsite CD training and encourage both R&D and Ops team members to attend the training together (it's amazing how much DevOps collaboration stems from a chat in the hotel bar)
- Reducing the many handover and decision-making points throughout the software release process
- Removing the barriers to allow developers to safely deploy their own software to the production platform
- Working with other business functions to gain trust and help them to refine and streamline their processes
- Working with R&D and operations teams to experiment with different agile methodologies such as **Kanban**

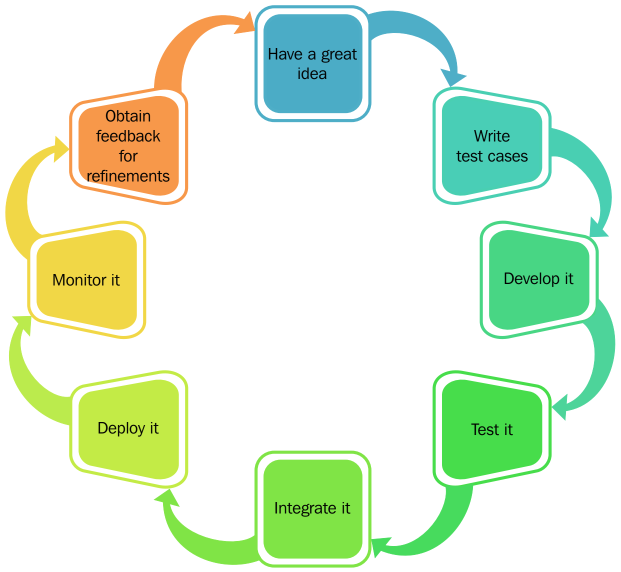
- Openly and transparently sharing information and data around deliveries and progress being made across all areas of the business
- Replacing the need for complex performance testing with the ability for developers to closely monitor their own software running in the production environment
- Evangelizing across all areas of the business to share and sell the overall vision and value of CD and DevOps

These initiatives are not easy to implement and it takes time to produce results but, after some months, the process of building and delivering software has transformed to the extent that a code change can be built, fully tested, and deployed to the production platform in minutes, many times per day—*all at the press of a button and initiated and monitored by the developer who made the change*.

Let's look again at the software delivery process flow to see what results have been realized.

Software delivery process flow version 3.0

As you can see from the diagram, the process looks much healthier. It's not as simple as version 1.0 but is efficient, reliable, and repeatable. Some much needed checks and balances have been retained from version 2.0 and optimized to enhance rather than impede the overall process.



An overview of the ACME systems version 3.0 software delivery process

This highly efficient process has freed up valuable DevOps resources so that they can focus on what they are best at—developing and delivering new software features and ensuring that the production platform is healthy and customers are again delighted.

The ACME systems team has gotten its mojo back and is moving forward with a new-found confidence and drive. They now have the best of both worlds and there's nothing stopping them.

ACME systems version 4.0

The ACME systems team have come through their challenges stronger and leaner but their story doesn't end there. As with any successful business, they don't rest on their laurels but decide to expand into new markets and opportunities—namely, to build and deliver mobile optimized clients to work with and complement their core web-based propositions.

With all they have learned throughout their evolution, they know they have an optimal way of working to allow them to deliver quality products that customers want, and they know how to deliver quickly and incrementally. However, the complexities of delivering code to a hosted web-based platform are not the same as the complexities of delivering code to an end consumer's mobile device—they are comparable but not the same. ACME systems also realizes that the process of delivering code to its production platform many times per day is under its control—code is being deployed to its infrastructure by its engineers using its tools—whereas it has little or no control over how its mobile clients are delivered, nor if and when the end consumer will install the latest and greatest version from the various app stores available. ACME systems also realizes that delivering a new version of its mobile clients many times per day is not viable nor welcome.

All is not lost—far from it. The ACME systems team has learned a vast amount throughout their evolutionary journey and decide to approach this new challenge as they did previously. They know they can build, test and deliver software with consistent quality. They know how to deliver change incrementally with little or no impact. They know how to support customers, and monitor and react quickly to change. They know their culture is mature and that the wider organization can work as one to overcome shared challenges. With this taken into account, here are a few of the things they decide to do:

- Agree on a realistic delivery cadence to allow for regular incremental changes without bombarding the end consumer
- Invest in new automated build, CI, and testing tools, which seamlessly integrate with and enhance the existing tooling

- Invest time and effort in nonfunctional features that will allow for greater visibility of what is running out in the wild, which again seamlessly integrates with the existing tooling and monitoring approach
- Ensure that the engineers delivering the mobile clients work closely with the backend engineers (DevOps) so that the client integrates seamlessly and doesn't cripple the existing production platform

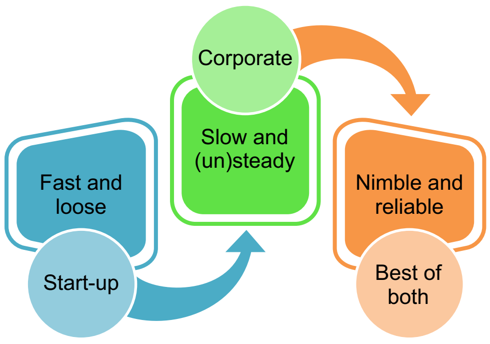
As the ACME systems team start to look into applying their established and proven approach to the new venture, they also discover another side effect of their newly rekindled success; they need to scale their platform and they need to do it as soon as possible. Given the timescales and urgency, the ACME systems team decides to move away from relying on their own datacenter and move towards a globally distributed "cloud-based" solution. This brings with it new challenges; the infrastructure is completely different, the provisioning tools are new, and the tools used to build and deliver software are incompatible with the existing ACME systems tools. Again, they take this in their stride and forge ahead with confidence using the ways of working, techniques and approaches that are now part of their DNA.

Could the ACME systems version 1.0 business have taken on these new challenges and succeeded? It's possible, but the results would have been mixed, the risks would have been much greater, and the quality much lower. It's pretty obvious that the ACME systems version 2.0 business would have had major struggles and by the time the products had hit the market, they would have been outdated and fighting for the market share with quicker and leaner competition.

If you would like to understand where you and your business sits within the CD and DevOps evolutionary scale, please see *Appendix B*, *Where Am I on the Evolutionary Scale?*

The evolution in a nutshell

Throughout this chapter, we have been following the evolution of ACME systems; where it started, the growing pains that came from success, how it discovered that being acquired brings with it negatives as well as positives, how it overcame its near extinction by adopting CD and DevOps, and how it regained its mojo and confidence to move forward. All of this can be represented by the following simple diagram:



An overview of ACME systems evolution

Summary

The ACME systems evolution story is not atypical of the many software businesses out there today. As stated previously, you may recognize and relate to some of the traits and challenges, and you should be able to plot where you, your business, or your employer currently sit within the stages detailed.

We'll now move from storytelling mode and start to look in more detail at some of the practical aspects of adopting CD and DevOps, starting with how one identifies the underlying problems that can—and do—stifle the delivery of quality software.

2
No Pain, No Gain

In the previous chapter, you were introduced to ACME systems and given an insight into how they realized that they had problems with their software delivery process (severely impacting their overall product-delivery capability), how they identified and addressed these problems, evolved, and after much hard work and some time, adopted a CD and DevOps ways of working. This isn't to say that you should simply dive in and adopt CD and DevOps because ACME systems did—far from it.

CD and DevOps, like any solution, can help you solve a problem, but you need to truly understand what the problem you're trying to solve is for it to be fully effective.

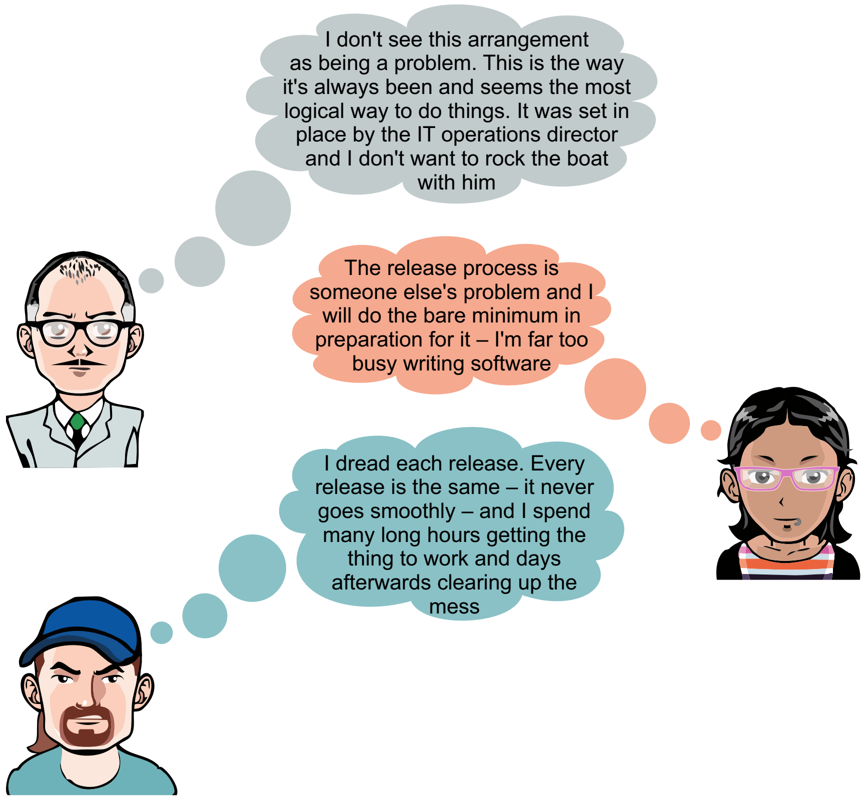
ACME systems took the time to understand the problem(s) they had before they began to implement CD and DevOps. They had to inspect before they could adapt.

Your first reaction to this might be that you don't have any problems, that everything is working well, and everyone involved with your software delivery process is highly effective, engaged, and motivated. If this is indeed true, then either:

- You have achieved software delivery utopia
- You are in denial
- You may not fully understand how efficient and streamlined software delivery can actually be

It's more likely that you have a workable process to deliver software, but there are certain teams of individuals within the process that slow things down. This is, most probably, not intentional; there might be certain rules and regulations that need to be adhered to, certain quality gates that are needed, it might be that no one has ever questioned why certain things have to be done in a certain way and everyone carries on regardless, or it might be that no one has highlighted how important releasing software actually is.

Something else to take into account is the fact that different people within your organization will see (or not see) a problem in different ways. Let's go back to ACME for a moment and examine the views of the three personas you were introduced to in relation to having the software releases controlled by the operations team:


I don't see this arrangement as being a problem. This is the way it's always been and seems the most logical way to do things. It was set in place by the IT operations director and I don't want to rock the boat with him
The release process is someone else's problem and I will do the bare minimum in preparation for it – I'm far too busy writing software
I dread each release. Every release is the same – it never goes smoothly – and I spend many long hours getting the thing to work and days afterwards clearing up the mess

As you can see, different people have wildly different views depending on what part they play in the overall process.

For the sake of argument, let's assume that you do indeed have some problems releasing your software with ease and want to understand what the root cause is (or most likely, what the root causes are) so that you can make the overall process more efficient, effective, and streamlined. Just like ACME, before you can *adapt*, you need to *inspect*; this is the fundamental premise of most agile methodologies.

Throughout this chapter, we will explore:

- How to identify potential issues and problems within your software delivery process
- How to surface them without resorting to blame
- How it can sometimes be tough to be honest and open, but doing so provides the best results
- How different people within your organization will see the same problem(s) in different ways

Before we start looking into how to *inspect*, I would like to go off on a slight tangent and talk about a large gray mammal.

Elephant in the room

Some of us have a very real and worrying ailment that blights our working lives: elephant in the room blindness, or to give its medical name, *Pachyderm in situ vision impairedness*. We are aware of a big problem or issue that is in our way, impeding our progress and efficiency, but we choose to either accept it, or worse still, ignore it. We then find ingenious ways to work around it and convince ourselves that this is a good thing. In fact, we might even invest quite a lot of effort, time, and money in building solutions to work around it.

To stretch this metaphor a little more—please bear with me, there is a point to this—I would like to turn to the world of art. The artist Banksy exhibited a piece of living artwork as part of his 2006 *Barely Legal* exhibition in Los Angeles. This living artwork was in the form of an adult Indian elephant standing in a makeshift living room with floral print on the walls. The elephant was also painted head to toe with the same floral pattern. The piece was entitled, as luck would have it, *Elephant in the Room*. It seems ludicrous at first, and you can clearly see that there is a massive 12,000 lb. elephant standing there; while it has been painted to blend in with its surroundings, it is still there in plain sight. This brings me to my point; the problems and issues within a software delivery process are just like the elephant, and it is just as ludicrous that we simply ignore their existence.

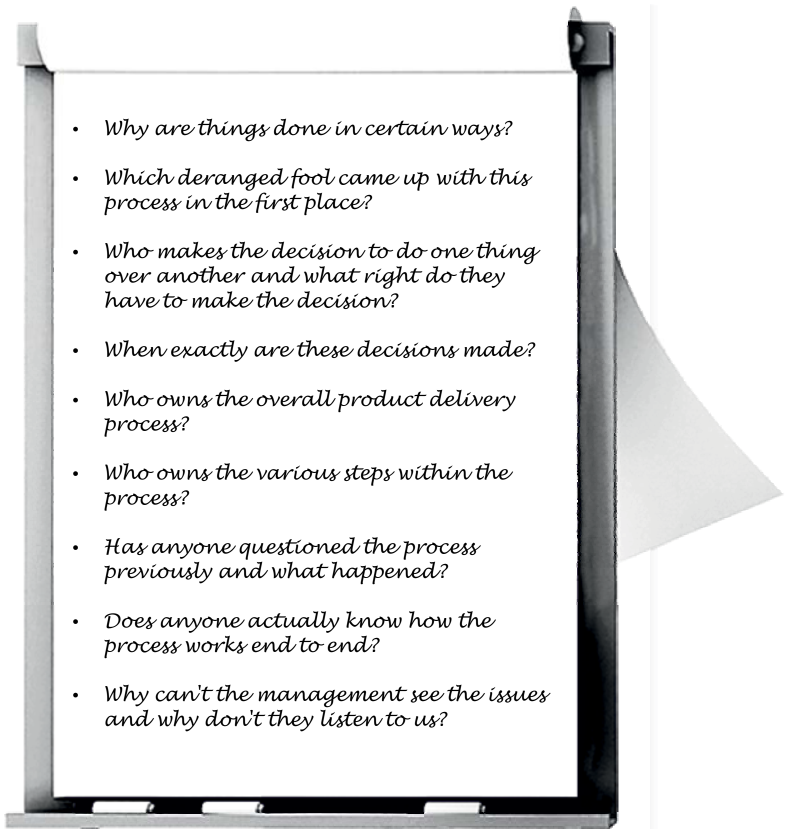
> The elephant in the room is not hard to spot if you look closely. It's normally sitting/lurking where everyone can see. You just need to know how to look and what to look for before you can expose it.

Through the remainder of this chapter, we'll go through some ways to help highlight the existence of the elephant in the room and, more importantly, how to ensure as many people as possible can also see it and realize that it's not something to be avoided, worked around, or ignored.

Before you start removing the figurative floral pattern from the figurative elephant, there's still some legwork you need to do.

Defining the rules

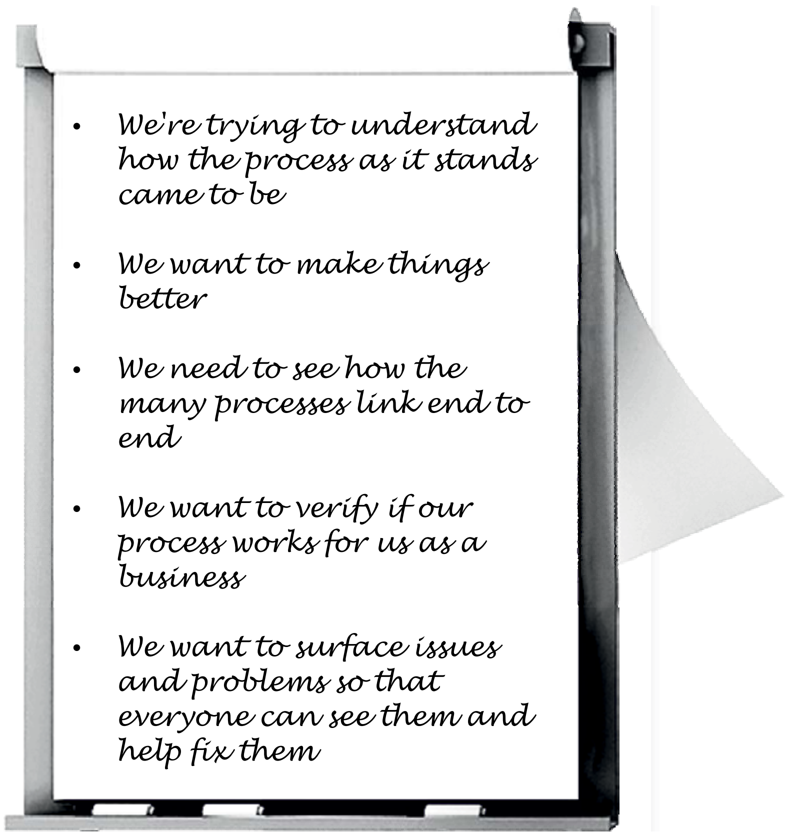
With any examination, exposé, investigation, or inspection, there will be, to some degree, dirt that will need to be dug up. This is inevitable and should not be taken lightly. The sort of questions that will be asked may well include the following:



- *Why are things done in certain ways?*
- *Which deranged fool came up with this process in the first place?*
- *Who makes the decision to do one thing over another and what right do they have to make the decision?*
- *When exactly are these decisions made?*
- *Who owns the overall product delivery process?*
- *Who owns the various steps within the process?*
- *Has anyone questioned the process previously and what happened?*
- *Does anyone actually know how the process works end to end?*
- *Why can't the management see the issues and why don't they listen to us?*

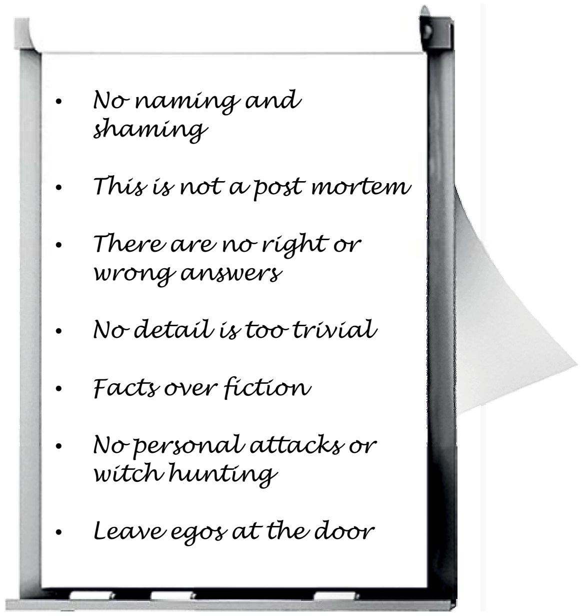
These types of questions may well make some people very uncomfortable and bring to light facts that produce emotive responses or emotional reactions, especially from those that might have originally had a hand in designing and/or implementing the very process that you are putting under scrutiny. Even if they can see and understand that the process they nurtured is broken, they might still have an emotional attachment to it, especially if they have been involved for a long time. You need to be mindful that these self-same people might be needed to help replace and/or refine the process, so tread carefully.

To keep things on a purely professional level, you should map out some ground rules that clearly define what the investigation is for and what its goal is. These rules need to be clear, concise, easy for everyone involved to understand, and worded in a positive way. The sort of things you should be looking at are as follows:



- We're trying to understand how the process as it stands came to be
- We want to make things better
- We need to see how the many processes link end to end
- We want to verify if our process works for us as a business
- We want to surface issues and problems so that everyone can see them and help fix them

To further ensure you minimize the emotional reactions, you should define some rules of engagement so that everyone involved understands where the *line* is and when it is about to be crossed. Again, keeping these rules simple, concise, and using positive language will help everyone understand and remember them. Good examples would be:



- No naming and shaming
- This is not a post mortem
- There are no right or wrong answers
- No detail is too trivial
- Facts over fiction
- No personal attacks or witch hunting
- Leave egos at the door

Retrospection can be a powerful tool, and if used incorrectly, it can cause more trouble than good; you can shoot yourself in the foot many, many times. You need to make sure you know what you are letting yourself in for before you embark on this sort of activity.

You now need to consider who should be involved and who will add the most value.

Including (almost) everyone

Although you will have the best intentions and will want to include everyone involved in the software delivery process to take part in the inspection, this might be neither realistic nor practical. What you need is information from individuals who can actively contribute, are engaged, are ideally open to change (or at least would like to see things change for the better), and understand and agree to the aforementioned rules.

These engaged contributors should come from all parts of the business; if they are involved in product creation and delivery, they should be involved. You need a broad set of information and data to move forward; therefore, you need to get a broad set of people involved.

As you start compiling the list of participants, which for a large organization can be quite daunting, you will no doubt find that there will be some degree of natural selection as you start to ask people to contribute; some might say they're too busy, some won't want to be involved for reasons they don't want to disclose, and some might simply not care enough either way.

Identifying the key people

One tip when compiling the list of participants is to try and identify the *key people* within the process. These key people might not be obvious at first, however, asking simple questions such as *who should I ensure I invite to this?* or *who do you normally talk to if there's a problem?* of a number of different people from different parts of the business will give you some indications.

There is a strong certainty that some of these key people will be the ones who say they are too busy. The fact that they are too busy might be directly attributed to the fact that the process they are working within is broken, but they don't have the time to stop and realize this. I recommend that you take time to ensure that the key people who fit into this category are encouraged, cajoled, and convinced to take part; if they are key, it sometimes helps to let them know this, as an ego boost can sometimes help win them over. Again, playing the *if you don't take this opportunity to sort things out for the better, someone else might and it might be worse* card sometimes works.

You might (will) also come across individuals who are very eager to be involved simply because they have an axe to grind or need a soapbox to proclaim their personal opinion. This is fine, but you need to be mindful of the fact that such people can potentially derail the investigation process—which again might be why they want to be involved. You should not simply dismiss these people out of hand as they might have valuable observations to bring forward, and dismissing them might foster further negativity. You should, however, ensure these individuals agree to be engaged contributors and understand the ground rules you have set. Of course, you will need to keep an eye on them, much like the naughty children of the class. That said, you might be surprised at how much value they bring to the process.

Too many cooks

As you build your list of participants, you might encounter a positive problem—you have too many people who want to be involved in the investigation. In some respects, this is a good thing; oversubscription is a nice problem to have. If this is the case, you should consider running multiple sessions rather than drop people off the list. We'll cover the format of the session(s) in more detail later, but suffice to say they can turn out to be very interactive with a high degree of active participation. As such, my advice would be to try and keep the numbers down, otherwise you will end up with too many voices generating too much noise. What you want is a few voices providing valuable information and data. The rule of thumb would be a maximum of 30 participants, which is more than enough, unless you're a very experienced facilitator.

In essence, you need to engage and include as many people as possible who are actively involved in the process of defining, building, testing, shipping, and supporting software within your organization. The wider the net is cast, the more relevant the information and data you will catch, so to speak.

Not only is the way in which the investigation is to be conducted and who is involved very important, but it is also vitally important that you ensure the environment is set up correctly and the proper open and honest behaviors are used and encouraged throughout.

We'll look into behaviors and culture in more detail in a later chapter, but for now, let's concentrate on three key areas.

Openness, transparency, and honesty

To truly get to the bottom of a given problem, you need to gather as much information and data about it as possible so that you can collectively analyze and agree the best way to overcome or address it—you need to know how big the elephant truly is before you expose it and then remove it. The natural reaction of most management "Stans" will be to run a closed session investigation and invite a selected few to take part.

This might provide some information and data, but it's a strong bet that it will not give you what you need; things will be missed, people will feel intimidated, and they will not feel free to disclose some pertinent pieces of information; some might simply forget an important detail, or worse still, some of the information might be misinterpreted or simply taken out of context.

All in all, closed session investigations are a hotbed for distrust, nondisclosure, disengagement, and blinkered points of view. Therefore, it is not recommended that you follow this course.

To realistically get the level of information and engagement, you need to create an open and transparent environment in which positive behaviors and honest, constructive dialog can flourish. This does not mean you have to work within a glass house, have every conversation in public, and every decision made by a committee. What is needed is *a distinct lack of secrets*.

Let me clarify what I mean by using a couple of examples:

1. Your senior management begrudgingly admits that there might be a few problems that need addressing. They then instruct the VP of engineering to handpick a team of people they trust to compile a list of solutions to present back. The VP is under orders to not disclose or discuss this with anyone outside of the closed group.
2. Your CEO invites every employee to an all-day workshop and asks everybody to provide open and honest feedback about the issues they face day to day. The CEO and the leadership team then spend the next few weeks openly working through all of the feedback. A follow-up workshop is then arranged to honestly discuss and debate the various options available.

I think it's plain to see the difference, which of the two approaches will bear fruit, and which will wither and die.

All of this might sound unrealistically simple, but without openness, honesty, and transparency, people will remain guarded, and you will not get all of the facts you need—the word *facts* in used intentionally. You need an environment where anyone and everyone feels that they can speak their minds, and more importantly, contribute.

Location, location, location

Ideally, you should plan to run your investigation collocated (everyone in the same physical location) as this allows for greater human interaction, building rapport, building trust, and encourages the general ebb and flow of conversation in what can be highly interactive exercises. You might also want to consider running these sessions on neutral ground (for example, rent a conference room in a local hotel or office complex), which not only puts people at ease but also takes some focus away from the office and its day-to-day distractions.

If you don't have the luxury of collocated teams, you need to be a little more creative in how you approach things. You should consider the following:

- Bringing the remote team(s) to you, budget permitting
- Sending the local team(s) to them, again budget permitting
- Using video conferencing (voice conferencing just isn't good enough)

You should also ensure that you take into account challenges around time zones and come up with workable options. For example, don't expect your Boston-based team to remotely attend a workshop at 5 a.m. (EST) just because it's easier for the UK team.

As you can see, before you embark on the challenge of exposing the elephant in the room, there is some preparation you need to do.

Throughout this chapter, you have been introduced to terms such as *investigation*, *elephant exposure*, and *retrospection*. In relation to your software delivery process, these all mean pretty much the same thing: gathering information and data on how the process works end to end so that you can highlight the issues, which can then be rectified. We'll now move on to some of the ways you can gather this information and data, but before we do so, let's clear a few things up.

It's all happy-clappy management waffle – isn't it?

Some of you of a technical ilk might be reading this chapter wondering who it's targeted at and thinking *surely this is all soft skill people management self-help waffle and doesn't really apply to me*. In some respects, this is very true; any good manager or leader worth their salt should at least know how to approach this type of activity. However, you have to take into account the very real fact—an ineffectual process has a greater impact on those within it than those who are perceived to be running it. Simply put, as an engineer, if your effectiveness, morale, and enjoyment of your role is impacted by a process that you feel is broken and needs changing, then you have as much right and responsibility to help change it for the better as anyone else. In my experience, the best engineers are those who can examine, understand, and solve complex problems—be they technical or not. In addition, who better to have on board while trying to find out the problems with a process than those directly involved in it and affected by it?

If you're a Devina or an Oscar stuck in a process that slows you down and impacts your ability to effectively do your job, then I strongly encourage you to get actively involved with investigating and highlighting the problems (there will be many, and some might not be as obvious to you as you first think). It can be daunting, and yes, if you're employed to analyze requirements or design systems, cut code, or look after the infrastructure, then why should you get involved in a *business analysis*? It's simple really; if you don't do anything, then someone else might, and things might get worse.

If you're a Stan, then I suggest you actively allow and encourage all of your Devinas and Oscars to get involved. As we just stated, they are the ones who are living within the process, and by implication, they know the process very intimately—far better than you, I suspect. Yes, some might need your help, some might need encouragement, some might need training or coaching, some might need empowerment, and some might need all of the above. In the long run, it will be worth it.

Not only should you encourage the *troops* to be actively involved, you should also use your influence and encourage your peer group to do the same. On the face of it, this might seem easy to achieve, but it can be quite a challenge, especially when other managers start putting roadblocks in your way. The sort of challenges you will face include:

- Convincing them it is a good and worthwhile thing to do
- Getting them to agree to allow many people to stop doing their day jobs for a few hours so that they can participate
- Getting them to agree to listen to others and not drive the agenda
- Getting them to be open and honest within a wider peer group
- Ensuring that they allow subordinates to be open and honest without the fear of retribution
- Getting them to trust you

As you can imagine, you might well be involved in many different kinds of delicate conversations with many people, roles, and egos across the business. It won't be easy, but the perseverance will be worth it in the long run.

Now that we have cleared that up, let's move on to the fun part—exposing the elephant.

The great elephant disclosure

Let's presume at this point that you have overcome all of the challenges of getting people in the same location (physically and virtually), you have obtained buy-in from the various management teams, have agreed some downtime, and have a safe environment set up in a neutral venue. You're almost ready to embark on the elephant disclosure, almost. What you now need to do is actually pull everyone together and run the workshop(s) to capture the data you need. These types of sessions need two things:

- The staple toolset of any agile practitioner: some big blank walls covered in paper, large whiteboards, flipcharts, sticky notes, various colored pens, and various colored stickers, some space, plenty of biscuits, and a little bit of patience
- A tried-and-tested agile technique that defines the format for the workshop

With regards to the second point, there are many varied-and-proven techniques and exercises you can use with wonderful names such as *StoStaKee*, the Starfish, the Sailboat, and TimeLine.

For the sake of space, I'll include references to these (and many others) within *Appendix C*, *Retrospective Games*, and we'll focus on the one in particular that has proven to be effective.

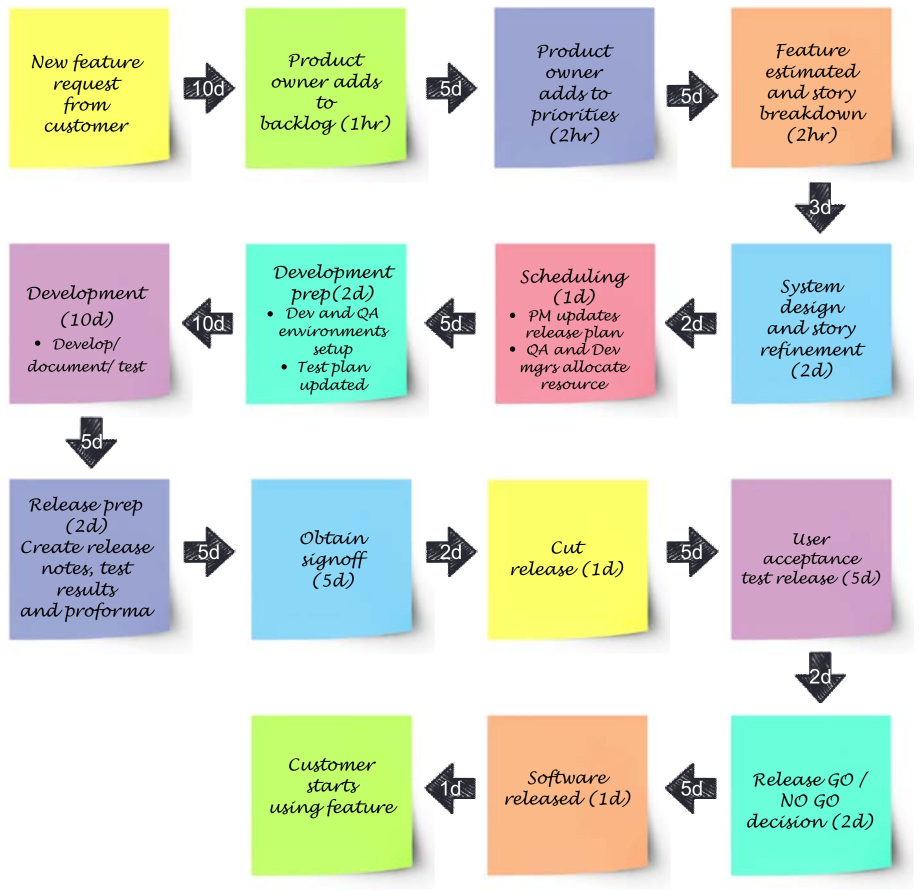
Value stream mapping

This lean technique derives from—as quite a few agile methodologies and tools do—manufacturing, and it has been around, in one guise or another, for many years. As with any lean methodology/tool/technique, value stream mapping revolves around a *value* versus *waste* model. In essence, a **value stream map** is a way to break down a product delivery process into a series of steps and handover points; it can also be used to help calculate efficiency rates, if that's useful to you. The overall map can be laid out and analyzed to see where bottlenecks or delays occur within the flow; in other words, you can see which steps within the process are not adding value. The key metric used within value stream mapping is the lead time (for example, how long before the original idea starts making hard cash for the business).

There are lots of resources and reference materials available to detail how to pull together a value stream map, and there are a good number of specialists in this area should you need some assistance.

To effectively create a value stream map, you will need a number of people across all areas of the business who have a very clear and, preferably hands-on, understanding of each stage of the product delivery process—sometimes referred to as the product life cycle. Ideally, a value stream map should represent a business process; however, this might be too daunting and convoluted at first. To keep things simple, it might be more beneficial to pick a recent example project and/or release and break this down.

As an example, let's go through the flow of a single feature request delivered by the ACME system's Version 2.0 business (before they saw the light):

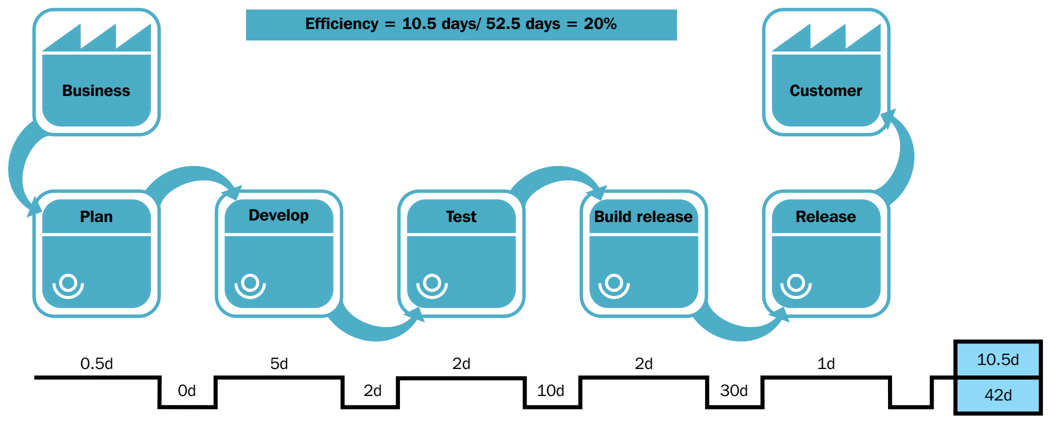

New feature request from customer
10d
Product owner adds to backlog (1hr)
5d
Product owner adds to priorities (2hr)
5d
Feature estimated and story breakdown (2hr)
3d
System design and story refinement (2d)
2d
Scheduling (1d) • PM updates release plan • QA and Dev mgrs allocate resource
5d
Development prep(2d) • Dev and QA environments setup • Test plan updated
10d
Development (10d) • Develop/ document/ test
5d
Release prep (2d) Create release notes, test results and proforma
5d
Obtain signoff (5d)
2d
Cut release (1d)
5d
User acceptance test release (5d)
2d
Release GO / NO GO decision (2d)
5d
Software released (1d)
1d
Customer starts using feature

Each box represents a step within the overall process. The duration value within each box represents the working time (that is, how long it takes to go through each step). The duration value in each arrow represents the wait time between each step (that is, how long it takes between each step).

This is a very simplistic overview, but it does highlight how much time it can take to get even the most simplistic requirement out of the door. It also highlights how much waste there is in the process. Every step, delay, and mistake has a cost. The only real value you get is when the customer actually uses the software so if it takes too long to get a feature, then the customer may well get bored of waiting and go elsewhere.

On the face of it, generating this type of map should be quite simple, but it can also be quite a challenge. This simplistic diagram is created in real time with input from many different areas of the business. There will be lots of open and honest dialogue and debate as facts are verified, memories jogged, dates and times corroborated, examples clarified, and agreements reached across the board as to what actually happens.

If you prefer to use the standard value stream mapping terminology and iconography, you can take the sticky notes version and convert it into something like the following, which again represents the flow of feature requests through the business:



This diagram also includes the efficiency (which is based upon the amount of time value is being added versus the dead time within the flow)

The most valuable output from this particular technique is that you can stop the obvious areas of waste. These are the parts of the overall process that are slowing down and impeding your overall ability to deliver. With this information, you can focus on these problem areas and start to look at options that will make them less wasteful and more valuable to the overall process.

Summary

Throughout this chapter, you have been given an insight into the following aspects: how to expose problems within your product delivery process—what we are calling the elephant in the room, the challenges and benefits of using collaborative, engaging approaches to identify these problems, and some effective tools and techniques to help you break down the problems into easily identifiable chunks of work.

Now, you know how to obtain valuable information and data about your problem(s) and have some much-needed actions to work with. You now know how to inspect. Let's presume these problems revolve around the waste created through long release cycles and a siloed organization. This being the case, you have a very clear objective, which will almost certainly address the problems and deliver what the entire business needs—you need to implement a CD and DevOps ways of working. All you now need to do is pull together a plan to implement it. In other words, you now need to adapt; this is handy as that's what we'll cover in *Chapter 3*, *Plan of Attack*.

3
Plan of Attack

Throughout *Chapter 2*, *No Pain, No Gain*, which for ease of reading we'll now refer to as the *inspect* stage, you were introduced to the tools and techniques to identify the problems you may well have with your overall product delivery process. We referred to this as the "elephant in the room" as it is something that is not hard to spot, just very easy to ignore. The presumption here is that the problems identified are the common place issues related to most software delivery processes. Some of these issues are listed as follows:

- Waste due to having too many handover and decision points in the process
- Waste due to unnecessary wait time between steps
- Many software changes are packaged up into large, complex *big bang* releases
- Large and infrequent releases breed an environment for escaped defects and bugs
- Releases are seen as something to dread rather than a positive opportunity for change
- People are disengaged or there is low morale (or both)
- Software changes are not trusted until they have been tested many many times
- Over complex dependencies within the software design
- Tasks that are duplicated throughout the process

We will now take the information and data you have captured and work on the method of turning this into something that you can implement to overcome the problems—a plan of attack to implement CD and DevOps if you will.

This plan of attack should not be taken lightly; just like the *inspect* stage, there is quite a bit of groundwork you need to do to ensure the scope of the implementation is understood, accepted, and communicated. As with any plan or project, there needs to be an end goal and a vision of how to get there.

Setting and communicating the goal and vision

A goal and vision for any project is important as it ensures all concerned know what is expected, and for those working on the project understand where it and they are heading. It may sound simple but it is not always obvious. How you communicate the goal and vision is just as important as setting them. Do this incorrectly and you are in danger of losing buy-in from the business, especially the senior management. For example, they may believe that to fix just one or two of the issues highlighted during the *inspect* stage will be enough to overcome all of the problems found. You have to be crystal clear what you plan to achieve, and crystal clear who you are communicating this to.

When it comes to CD and DevOps, this can be quite challenging as the deliverables and benefits are not always easy for the uninitiated to understand or envision. It may also be difficult to fully quantify as some of the benefits you obtain from the adoption of CD and DevOps are not wholly tangible (that is, it is quite hard to measure increases in team collaboration and happiness).

The best advice is **keep it simple stupid** (**KISS**). You have a list of issues that the wider business has provided, and what they want is something (anything) that will make their lives easier and allow them to do their jobs. If truth be told, you most probably have more things on the list than you can effectively deliver. This should be seen as a good thing as you have some wriggle room when it comes to prioritization of the work.

Your challenge is to use this list and pull together a goal and vision, which will resonate with all of the stakeholders and ensure it is something that can be delivered. This may need quite a bit of effort but it is doable. For a good example, let's once again have a look at ACME systems. When they were planning the implementation of CD and DevOps, they came up with a goal for the project, which was *to be able to release working code to production 10 times per day*. This was a nice simple tag line, which everyone could understand (almost everyone, but we'll come to that later) and formed the basis of their communication strategy. Yes, it was an ambitious goal but they knew with a little hard work, courage, determination, and the right people involved, it was possible. They even created posters that could be stuck on walls around the office.

A very simple tag line that anyone and everyone can understand

Setting your goal may be just as easy. You have a good understanding of the business problems that need to be overcome, you know which teams are involved, and you have a good idea of what will resonate with the stakeholders. The goal needn't be a grandiose thing—maybe something as simple as to allow engineers *to release their own code* or *ship code at the press of a button* will suffice. The most important thing here is to set a goal that people can get behind, which solves some—ideally most—of the problems highlighted.

Once you have a goal (or most probably, a list of potential goals), canvass opinion from people whose judgment you trust; if they think your proposal is way off the mark, it might just be so. If you're lucky enough to have PR or marketing people accessible, canvass their opinions; this is after all something they are pretty good at.

Once you're happy with the goal, you need to work on the vision. It may help to think of the goal as *what* you want to achieve and the vision as *how* you will achieve it. The vision should contain as much detail as can be easily communicated and understood. You have to be mindful of the fact that too much detail will confuse and cause people to become disengaged, so KISS.

Let's go back to the ACME team to see how they went about doing this. They had a goal (*release working code to production 10 times per day*) and now had to set out the vision. They took the list of issues highlighted during the inspect stage and translated them into a list of easy-to-understand actions and tasks that needed addressing. A good example of this is the problem *engineers are unable to ship their own code*, which made perfect sense to the engineers themselves, but wasn't simple enough to understand for anyone outside of that group—questions such as *ship to where?* and *what do you mean by ship?* arose. After some debate, discussion, and refinement, this problem was translated to *allow engineers to release fully working code from their laptop to the live platform with ease*. This was something the majority could understand—it was simple. This can also be simplified further to *laptop to live*, which still conveyed the meaning but was easier to digest and communicate.

The vision ACME created included a wide variety of things, some technical and some not, which could all be clearly communicated and understood. Those of you who are au fait with agile ways of working, may spot this as *a prioritized feature backlog*.

The next step was to share the goal and vision to the business and stakeholders and gain an agreement that what was proposed would address the problems and issues captured during the inspect stage. This was directed to as wide an audience as possible—not just the management—with many sessions booked over many days to allow as many people to be involved as possible.

Once the goal and vision had been shared, discussed, and revised—based upon the constructive feedback from all involved, a top-level plan was pulled together. Put simply, ACME generated a story backlog that contained almost everything they needed to address and deliver.

To ensure transparency and ease of access to the goal and vision, the ACME team needed to ensure that all of the data, information, and plans were publicly (internal to the business rather than in the public domain) available. To this end, they fully utilized all of the internal communication and project repository tools available to them: internal wikis, blogs, websites, intranets, and forums.

If you don't have tools like these available to you, it shouldn't be a vast amount of effort to get one set up using open source solutions. There are even on-line solutions that are secure enough to keep company secrets safe—some examples can be found in *Appendix A, Some Useful Information*. Having this level of transparency and openness will help as you move forward with the execution of the plan. This is especially true of *social* solutions such as blogs and forums, where feedback can be given and virtual discussions can take place.

It all sounds pretty simple when it's put down into a few paragraphs and to be honest, it could be with the right environment and the right people involved. It's just a case of ensuring that you have a good grasp of what the business and stakeholders want, you know how to summarize this into an easily understandable goal and align the vision to drive things in the right direction. The key here is *easily understood*, which can sometimes be a challenge, especially when you take into account communication across many business areas (and possibly many time-zones and cultures), where each have their own take on the terminology and vocabulary used. This brings us to how you should communicate and ensure everyone involved understands what is happening.

Standardizing vocabulary and language

One small and wholly avoidable factor that can scupper any project is the misinterpretation of what the deliverables are. This may sound a little alarming but projects can fail simply because one person expects something, but another person misunderstands or misinterprets and delivers something else. It's not normally down to ignorance; it's normally due to both sides interpreting the same thing in different ways.

For example, let's look at something simple—the word *release*. To a project manager or a release manager, this could represent a bundle of software changes, which need to be tested and made live within a schedule or program of work. To a developer working in an agile way, a release could be a one line code change, which could go live soon after he/she has completed coding and running the tests.

There can also be a bit of a problem when you start to examine all of the different words, terminology, and **three letter acronyms** (**TLA**) that we all use within IT. We therefore need to be mindful of the target audience we are communicating to and with. Again the KISS (a FLA or four-letter acronym if you prefer) method works well here. You don't necessarily have to go down to the lowest common denominator; that may be very hard to do and could make matters worse. Try to strike a balance. If some people don't understand, then get someone who does understand to talk with them and explain; this will help bridge the gap and also helps to form good working relationships.

Another suggestion to help bridge the gap is to pull together a glossary of terms that everyone can refer to. The following is a simple example:

Term	**What it is**	**What it is not**
Continuous Delivery	A method of delivering fully working and tested software in small incremental chunks to the production platform	A method of delivering huge chunks of code every few weeks or months
DevOps	A way of working that encourages the Development and Operations teams to work together in a highly collaborative way towards the same goal	A way to get developers to take on operational tasks and vice versa
CD	See Continuous Delivery	
Continuous Integration	A method of finding software issues as early as possible within the development cycle and ensuring all parts of the overall platform talk to each other correctly	Something to be ignored or bypassed because it takes effort
CI	See Continuous Integration	
Definition of done	A change to the platform (software, hardware, infrastructure, and so on) is live and being used by customers	Something that has been notionally signed off as something that should work when it goes live
DOD	See definition of done	
Release	A single code drop to a given environment (testing, staging, production, and so on)	A huge bundle of changes that are handed over to someone else to sort out
Deploy	The act of pushing a release into a given environment	Something the Operations team does

If you have a wiki/intranet/blog/forum, then that would be a good place to share this as others can update it over time as more buzzwords and TLAs are introduced.

The rule of thumb here is to ensure whatever vocabulary, language, or terminology you standardize on, you must stick to it and be consistent. For example, if you choose to use the term *CD and DevOps*, you should stick with it through all forms of communication, written and verbal. It then becomes ingrained and others will use it day to day, which means conversations will be consistent and there is much less risk of misinterpretation and confusion.

You now have a goal, a vision, a high level backlog, a standard way of communicating, and you're ready to roll (almost). The execution of the plan is not something to be taken lightly. Whether you are a small software shop or a large corporate, you should treat the adoption and implementation of CD and DevOps with as much gravitas as you would any other project, which touches and impacts many parts of the business. For example, you wouldn't implement a completely new e-mail system into the business as if it were a small scale *skunk works* project—it takes collaboration and coordination across many people. The same goes for CD and DevOps.

A business change project in its own right

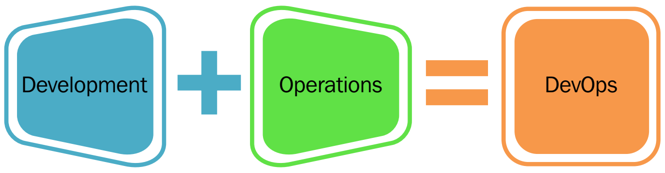
Classing the implementation of CD and DevOps as business change project may seem a bit dry but that's exactly what it is; you are potentially changing the way the whole business operates, for the better. Not something to be taken lightly at all. If you have ever been involved in business change projects, you will understand how far reaching they can be.

There's a high probability that the wider business may not understand this as well as you do. They have been involved in the investigation and have verified the findings and seen what you intend to do to address the issues raised. What they may not understand fully is the implication of implementing CD and DevOps—in terms of the business, it can be a life-changing event. A little later on in the book, we'll go through some of the hurdles you will face during the implementation. However, if you have a heads-up from the start, you're in a much better position to leap over the hurdles.

Suffice to say that you should ensure you get the business to recognize that the project will be something that will impact quite a few people, albeit in a positive way. Processes will change as will the ways of working. We're not just talking about the software delivery process here; CD and DevOps will change the way the business thinks, plans, and operates.

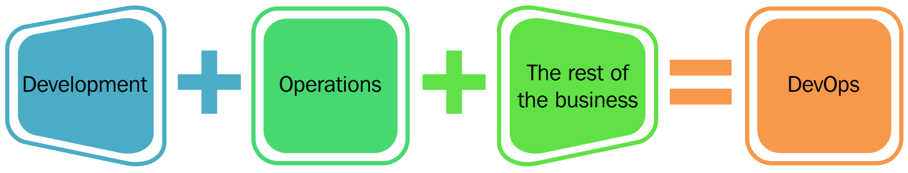
For example, let's assume that marketing and program management teams are currently working on a 6 to 9 month cycle to take features to the market. If all goes well with the CD and DevOps implementation, they will have to realize that a feature may be live in a few weeks or even days. They will therefore need to work at a different cadence and will have to speed up and streamline their processes. From experience, this sort of change also brings with it some unexpected benefits—that being a renewed level of trust throughout the business that when the development and Operations team say they will deliver something, they actually deliver it. Therefore, the traditional contingency plan B is no longer required (nor plans C, D, or E). The way features are delivered will drastically change and the rest of the business needs to accept this and be ready for it.

In the early stages of the project, the wider business will most probably believe that the impact of CD and DevOps—as the name suggests—will be localized to the Development and Operations teams. The following diagram depicts the extent of this change as the business sees it:



What the business sees at the early stages

At first, this may not be too far from the truth, and you may start small so that you can get to grips with the subtleties and to find your feet as it were. This is fine; however, once you get some momentum—which won't take long—things will start to change very quickly and if people aren't ready, or at least aware, you may hit some barriers, which could slow things down or even stop the implementation in its tracks. The business therefore must accept that the impact will be far reaching as depicted below:



What the business should be seeing as representative of the areas that will be impacted and involved

Getting the business to understand this will not be an easy task, they will need some convincing and some good old-fashioned diplomacy may be again required. Luckily, CD and DevOps is now becoming more widely known outside of the traditional IT realm and there is plenty of information, such as case studies, available to reference. That said, you have to be mindful that the wide business will still see this as an *IT thing* rather than a *business thing*.

Let's move forward and presume that the business is in agreement regarding the wide reaching nature of the implementation and (almost) everyone is behind the project. The next challenge is looking at the merits of using a dedicated team to implement the goal and vision.

The merits of a dedicated team

As with any high-profile project, it's worth considering the merits of having a dedicated team focused on the implementation of CD and DevOps. This is the approach ACME took and although it worked for them, it's your call whether you go down this route or not. There may be a temptation to run the implementation as a *skunk works* project. This type of project will tend to bubble along in the background and be staffed by like-minded individuals who have an interest but don't have the backing or reach needed to make sweeping changes, nor the free time to dedicate to make it truly successful. My recommendation is to not do this as such projects tend to fade away as more seemingly important projects—which do have backing and formal widespread recognition—take the limelight and more importantly the resource. This may sound cynical but it's a fact of life and you need to be mindful of this.

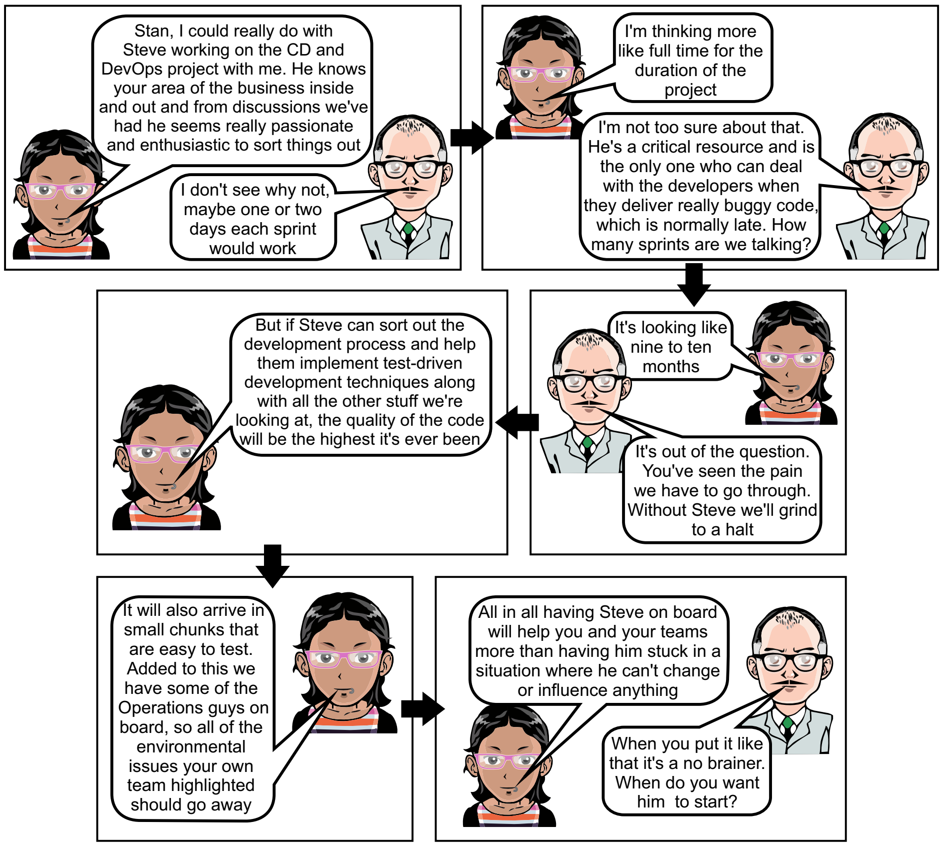
> There seems to be a growing trend to hire or set up dedicated CD and/or DevOps teams to run the process of delivering software. This is not what I am referring to. I am referring to setting up a cross-functional and multi-disciplined team that can help drive the goal and vision on behalf of the wider business.

I would advise that you work with the wider business to ensure their agreement to the implementation of CD and DevOps goes further than simply paying lip service. They need to put their money where their mouth is and provide some people to help with the implementation. Reiterating the fact that the implementation of CD and DevOps should be considered as a business change project may help. In the end, you will need a team of like-minded individuals working on the project from across the business (not just developers and operations guys) to make this successful. You may want to start small and build up but the reality is that once things start getting traction, the wider business will need to get involved.

As soon as this is highlighted, you will no doubt get some areas of the business take a big step back in terms of engagement and commitment—especially those areas that manage the very people you want to second onto the project. It is then down to you to cajole, beg, bargain, and barter to get the people you need. To be honest, this shouldn't be too difficult as you have quite a large amount of ammunition to use—the same information and data you worked so hard to compile, which the business itself agreed was causing pain. If you used the value stream mapping exercise, you should also be able to pinpoint the pain areas with accuracy.

One thing to take into account is the level of commitment of those outside of the traditional IT realm—I'm thinking of such things as sales, marketing, HR, finance, and so on. It may not be viable to have such individuals dedicated full-time to the project. However, you should ensure they allocate some of their time and make themselves available when needed.

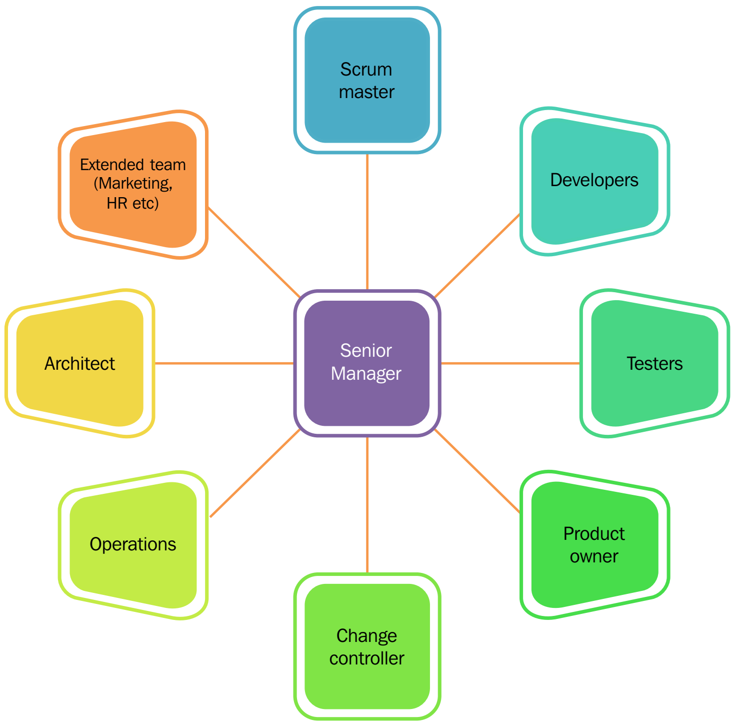
Let's take a typical discussion between you - who, for this example, is played by Devina and Stan, the manager—who, for this example, is the head of Testing and QA:


Stan, I could really do with Steve working on the CD and DevOps project with me. He knows your area of the business inside and out and from discussions we've had he seems really passionate and enthusiastic to sort things out
I don't see why not, maybe one or two days each sprint would work
I'm thinking more like full time for the duration of the project
I'm not too sure about that. He's a critical resource and is the only one who can deal with the developers when they deliver really buggy code, which is normally late. How many sprints are we talking?
It's looking like nine to ten months
It's out of the question. You've seen the pain we have to go through. Without Steve we'll grind to a halt
But if Steve can sort out the development process and help them implement test-driven development techniques along with all the other stuff we're looking at, the quality of the code will be the highest it's ever been
It will also arrive in small chunks that are easy to test. Added to this we have some of the Operations guys on board, so all of the environmental issues your own team highlighted should go away
All in all having Steve on board will help you and your teams more than having him stuck in a situation where he can't change or influence anything
When you put it like that it's a no brainer. When do you want him to start?

I admit it might not go exactly along those lines but you can see the point. You have been given a clear insight into what pains the business and have been asked to remove the said pains. The business needs to realize that this will not come without some cost and that they need to provide you with what you need to get the job done.

Who to include

If you decide to utilize a dedicated team, then you'll no doubt want to know who to include. This really depends on the way in which your business is set up. To help a little, let's again go back to ACME systems and see what they did. During version 2.0 of their evolution, the teams that were actively involved in delivering software included: Architecture, Development, Testers, Operations, and Change Management. They therefore decided to include individuals from each of these disciplines within their CD and DevOps team. They then added a scrum master and a product owner and topped it all off with a senior manager (someone who could act as the project sponsor and represent the team at a higher level). To follow the theme of inclusion, they also added stakeholders from the wider business. In the end, the ACME systems CD and DevOps team looked something like this:



The ACME CD and DevOps project team

If and how you go about setting up your own CD and DevOps team and who you include totally depends on the way in which your business is set up. The most important thing to remember when/if setting up a dedicated team is that it must be made up of more than just developers if they are to have credibility and be successful.

Let's move on from the practical elements of team building to the weird and wonderful world of evangelism and the benefits this activity can bring.

The importance of evangelism

Whenever you introduce a change, be it a new product, service, or a wholesale change to the ways of working, you need evangelists to ensure everyone who needs to know about the change, knows about it. Sometimes, this is seen as marketing or PR, but in its basic form, it is evangelism. Evangelism is important. It's also very hard work but is essential for any change to be successful. CD and DevOps has the potential to introduce a vast amount of change to the way your business works. Therefore, a vast amount of evangelism will be required. Even if you have a goal, vision, and the blessing from up on high, you need to evangelize to ensure those who are most important to the success of the implementation are behind you and understand what they are getting behind. You need to get out there and be seen and be heard.

Don't get me wrong, to evangelize across an entire business is going to take some effort and some determination. It will also take some energy. Actually, that's wrong; it will take a lot of energy. Your target audience is wide and far reaching, from senior management to the shop floor. So, it will take up quite an amount of time for you to get the message across. Before we go into the details of what to say to who, when, and how, let's get the ground rules sorted:

- If you are to be convincing when evangelizing to others the virtues of CD and DevOps, you need to believe in it 100 percent—if you don't, then how can you expect others to?
- You and whoever is involved in the project must practise what you preach and lead by example. For instance, if you build some tools as part of the project, make sure you build and deploy them using the exact same techniques and tools you are evangelizing about.
- Many people will not get it at first. So, you will have to be very, very patient. You might have to explain the same thing to the same person more than once. Use these kind of individuals as a yard stick; if they start to understand what CD and DevOps is all about, then there's a pretty good chance your message is correct.

- Remember your target audience and tailor your message accordingly. Developers will want to hear technical stuff, which is new and shiny; system operators would want to hear words such as *stability* and *predictability*; and management types would want to hear about *efficiencies*, *optimized processes*, and *risk reduction*. This is rather generalist. However, the rule of thumb is if you see their eyes glaze over, your message is not hitting home, then change it.
- Some people will simply not want to know or listen and it may not be worth focusing your efforts to make them (we'll be covering some of this in *Chapter 6*, *Hurdles Along the Way*). If you can win them round, then kudos to you but don't feel dejected by a few laggards.
- Keep it relevant and consistent. You have a standardized language, a goal, and a vision so use them.
- Don't simply make stuff up. Just stick to what can be delivered as part of your goal and vision; nothing more, nothing less. If there are new ideas and suggestions, get them added to the backlog for prioritization.
- Don't on any account give up.

What it boils down to is you will need to talk the talk and walk the walk. There will be quite a bit of networking going on; so be prepared for lots of discussion. As your network grows, so will your opportunities to evangelize. Do not shy away from these opportunities, and make sure you are using them to build good working relationships across the business as you're going to need these later on. Evangelizing is rewarding and if you really believe that CD and DevOps is the best thing since sliced bread, you will find that having opportunities to simply talk about it with others is like a busman's holiday.

As mentioned earlier, evangelism is a form of PR. So, if you have PR people available (or better still as part of the team), you should also look into getting simple things together, such as a logo or some *freebies* (such as badges, mugs, mouse mats, and so on), to hand out. This may seem a little superfluous but as with any PR you will want to ensure you get the message across and have it imbedded into the environment and peoples' psyche.

Up until this point, I may have painted things in a somewhat rosy glow. Adopting CD is no picnic. There's quite a big hill to climb for all concerned. As long as everyone involved is aware of this and has the courage and determination to succeed, things should go well.

Courage and determination

Courage and determination may seem like strong words to use but they are the correct words. There will be many challenges, some you are aware of some you are not, that will try to impede the progress. So, determination is required to ensure this keeps moving in the right direction. Courage is needed as some of these challenges will require you, the team, and the wider business to make difficult decisions, which could result in actions being taken from which there is no going back. I'll refer to ACME systems Version 2.0 for a good example of this.

In the early days of their adoption of CD and DevOps, they started with a small subset of their platform as the candidates for releasing using the new deployment toolset and ways of working. Unfortunately, at the same time, there was a lot of noise being generated around the business as another release (using the old *package everything up and push out as one huge deployment* method) was not going well. The business asked everyone to focus on getting the large release out at all costs, including halting the CD trials. This didn't go down too well with the team. However, after a rather courageous discussion between the head of the ACME CD team and his peers, it was agreed that resource could be allocated if there was universal agreement that this would be the last of the *big bang* releases and that all future releases would use the new CD pipeline process going forward. The agreement was forthcoming and so ended the era of the big bang release and the new era of CD dawned. After the last of the *big bang* releases was eventually completed, the entire development and operations teams were determined to get CD up and running as soon as possible. They had been through enough pain and needed another way or rather a better way. They persevered for a few months until the first release, using the new tooling and ways of working, went to the production environment, then the next, and so on. At this point, there was no turning back as too much had changed.

As you can no doubt appreciate, it took courage from all parts of the business to make this decision. There was no plan B and if it hadn't worked, they had no way to release their software. Knowing this fact, the business was determined to get the new CD and DevOps ways of working imbedded and established.

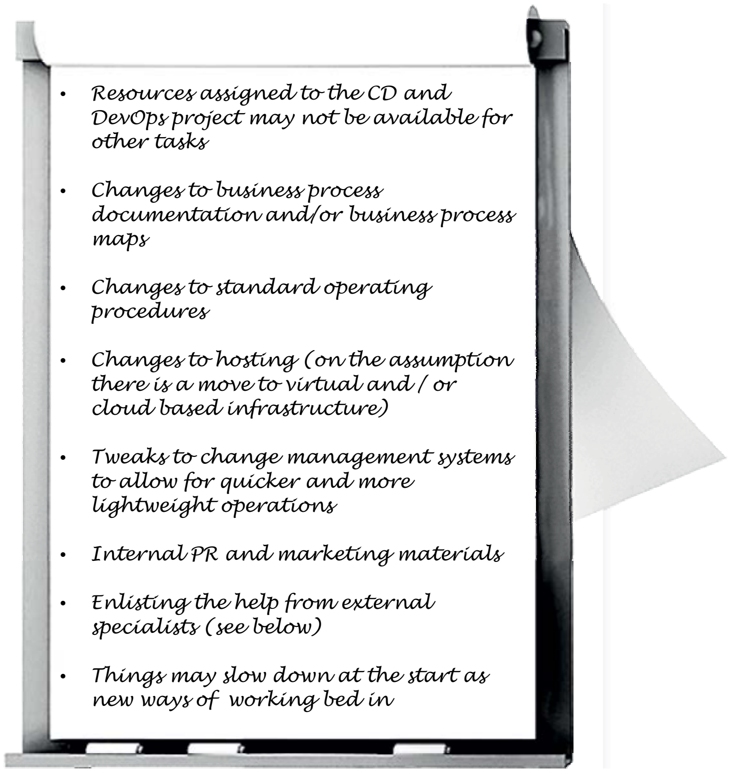
The preceding example could be classed as an extreme case but nonetheless, it goes to show that courage and determination are sometimes very much needed. If there's a will, there's a way.

Before we move away from the planning stage, there are still a couple of things you should be aware of as you prepare to embark on your new adventure: where to seek help and ensuring you and the wider business are aware of the costs involved with implementing CD and DevOps. We'll cover costs first.

Understanding the cost

Implementing CD and DevOps will ultimately save the business quite a lot of money—that is a very simple and obvious fact. The effort required to release software will be dramatically reduced, the resources required will be miniscule when compared to large *big bang* releases, the time to market will be vastly reduced, the quality will be vastly increased, and the cost of doing business (that is, volume of bug fixes required, support for system downtime, fines for not meeting SLAs, and so on) will be negligible. That said, implementing CD and DevOps does not come for free. There are costs involved and the business needs to be aware of this.

Let's break these down:



- Resources assigned to the CD and DevOps project may not be available for other tasks
- Changes to business process documentation and/or business process maps
- Changes to standard operating procedures
- Changes to hosting (on the assumption there is a move to virtual and / or cloud based infrastructure)
- Tweaks to change management systems to allow for quicker and more lightweight operations
- Internal PR and marketing materials
- Enlisting the help from external specialists (see below)
- Things may slow down at the start as new ways of working bed in

These costs should not be extortionate; however, they are costs that need to be taken into account and planned for. As with any project—especially one as far reaching as CD and DevOps—there will always be certain costs. If the business is aware of this from the outset, then the chance of it scuppering the project later down the line can be minimized.

There may be some costs that are indirectly caused by the project. You may have some people who cannot accept the changes and simply decide to move on; there will be costs to replace them (or not as the case may be). As previously stated at the beginning of the transition from *big bang* releases, you may well slow down to get quicker. If you have contractual deadlines to meet during this period, it may be prudent to renegotiate them.

You will know your business better than anyone—especially, after completing the investigations into how the business functions—so, you may have better ideas related to costs. Just make sure you do not ignore them.

Let's now focus on where you can get help and advice should you need it.

Seeking advice from others

Before you take the plunge and change the entire way your business operates, it would be a good idea to do some research and/or reach out to others who:

- Have been through this transition already—maybe a few times
- Have insights that you may not considered or thought about
- Have some battle scars that you will want to avoid
- Are in the same boat as you

There is an ever-growing number of people around the globe who have experience in implementing (and even defining) CD and DevOps. Some are experts in the field and focus on this as their full-time jobs; some are simply members of the growing community who have seen the light and selflessly want to help others realize the benefits they have witnessed and experienced.

If you can secure the budget to have an external expert come in to work with you that may help take some of the pressure off. If you do go down this route, be mindful of who you chose—remember that some *consultancies* will be more than happy to assist and will be just as happy to sell you their latest CD pipeline tool. If you're confident in your approach, then maybe just having a sounding board available every few weeks/months will suffice.

To reiterate, implementing CD and DevOps is no picnic and sometimes being at the forefront can be a lonely place. Do not feel like you should struggle alone. There are some valuable reference materials available (this book being one of those I would hope) and more importantly, there are a good number of communities—online and face-to-face meet-ups—which you can join to help you.

You never know, your story and input may be an inspiration for others, so in true DevOps style, break down the barriers and enjoy open and honest dialogue. I'll include a list of some of the reference materials and contacts in *Appendix A, Some Useful Information*.

Summary

As with any business change project, to successfully implement CD and DevOps, you need to ensure you know *what* you are setting out to do (your goal), understand *how* you're going to reach it (your vision), understand who's help you will need, clarify how you will communicate and evangelize, be realistic about how much it will cost, and most important of all, be realistic about how much work, courage, energy, and determination it will take. As mentioned previously, this is no picnic but it will be worth it.

Hopefully, you're all fired up and ready to go but before we do this let's take a look at something that may not be so obvious now but is yet another essential part of the CD and DevOps jigsaw: culture, and behaviors.

4
Culture and Behaviors

In *Chapter 2*, *No Pain, No Gain*, we learned that asking people to be open and honest is not easy unless you set the environment up to allow for it to happen. The culture and environment have to be such that honest disclosure can take place, and you have to ensure that every participant agrees to behave according to the flexible rules set out. We will now take this newfound knowledge and expand upon it to ensure the culture and behaviors throughout the organization are set up to allow for—what can be—potentially massive change. The sorts of things we'll cover throughout this chapter are:

- Why culture is so important
- How culture and behaviors affect your progress
- Encouraging innovation at grass roots
- Fostering a sense of accountability across all areas of your organization
- Removing blame from the working environment
- Embracing and learning from failure
- Building trust
- Rewarding success in the right way
- Instilling the sense that change is good and not risky
- How good PR can help

Throughout this chapter, we'll also look at what this means to the three personas (Stan the manager, Devina the developer, and Oscar the ops guy) you were introduced to in *Chapter 1*, *Evolution of a Software House*.

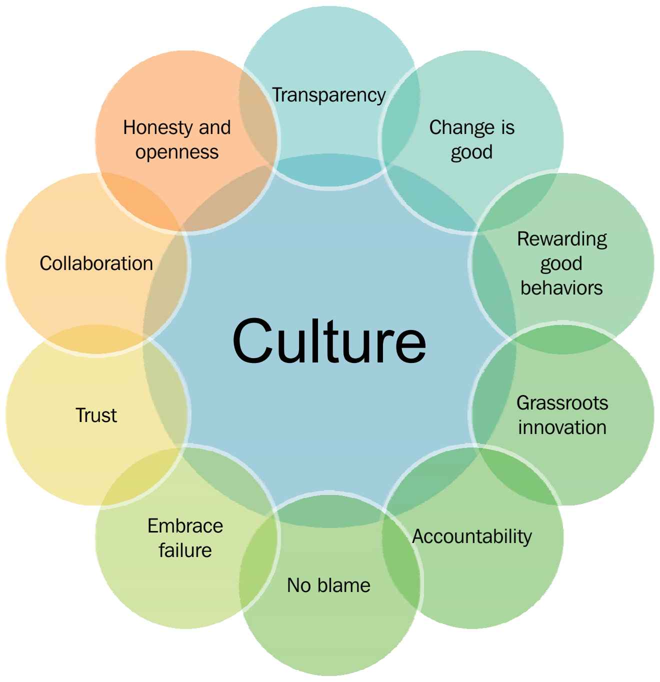
It should be noted that I am by no means an expert in the human sciences, nor do I have a PhD in psychology. What follows are learnings through observation, experience, and collaboration with experts in these fields.

Let's start by clarifying why culture is so important to CD and DevOps.

All roads lead to culture

Some people might think that CD and/or DevOps is simply about implementing technical tools, making slight tweaks to existing heavyweight processes to allow software to be released every few weeks, or worse still, a bona fide reason to set up a new "DevOps" team inside an existing organization. If you think any of these are correct, then you are wrong. CD and DevOps are—put very simply—ways of working. As such, the culture in which people work and the behaviors they exhibit has a massive part to play. If you have barriers or power struggles between teams, silos across the organization, ineffective lines of communication, a rigid old-school hierarchy, dysfunctional leadership, or your business is resistant to change or learning from failure, then your environment and culture are not conducive to adopting CD or DevOps ways of working. Attempting to implement CD or DevOps in such an environment will ultimately lead to failure, unless you address the underlying behaviors and overarching culture.

As we found in *Chapter 2*, *No Pain, No Gain*, which for ease of reading we'll now refer to as the *inspect* stage, culture is quite nebulous and can be hard to define. The following diagram attempts to clarify what we mean by culture in relation to this subject; we'll cover some of this in more detail throughout this chapter.

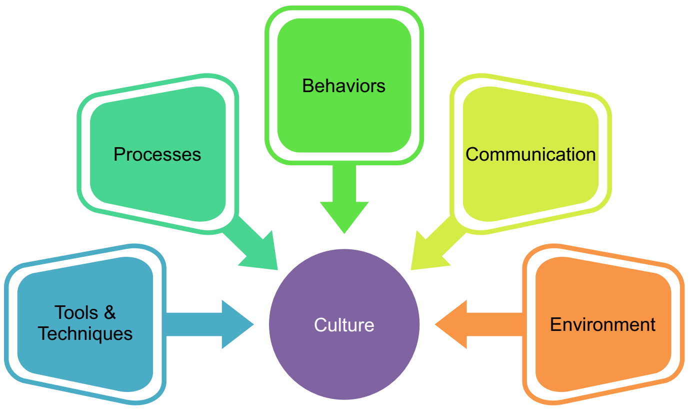


The cultural interconnectedness of all things CD and DevOps

You need to ensure that your organization is set up in such a way as to allow for all of the positive behaviors you witnessed during the *inspect* stage to resurface and become the norm. In essence, what you need is a positive, collaborative, and open culture. This is no mean feat, but it can be done. To some degree, you have already planted the seed during the *inspect* stage—albeit in safe greenhouse conditions—and have proven and realized the benefits. What you need to do is nurture this seedling and let its roots dig deep and spread across your organization.

This Cultural Revolution shouldn't be restricted to the shop floor; whoever is involved in or makes decisions about software delivery, be they an engineer, a manager, or the VP of engineering, needs to at least have an understanding and appreciation of how culture and behaviors can help or hinder the adoption of CD and DevOps.

A healthy culture is central to a successful way of working, and is therefore central to CD and DevOps, as is depicted in the following diagram:



Culture is central to all aspects of successful CD and DevOps adoption

Now, let's revisit something that we looked at during the setting up of the *inspect* stage and expand upon it.

An open, honest, and safe environment

Apart from sounding like something taken directly out of a management training manual, what does having an open, honest, and safe environment actually mean? In relation to CD and DevOps, this means that you need to ensure that anyone and everyone involved in your product delivery process is able to openly comment and discuss ideas, issues, concerns, and problems, without the fear of ridicule or retribution.

As you found during the *inspect* stage, allowing for open discussions and honest appraisals of how things are done within your organization and the product delivery process brings to the surface details and facts that otherwise would have been missed or stayed hidden. You need to persist the culture where the distinct lack of secret behavior is maintained or, if there is a gap between your *inspect* stage and implementation stage, rekindled and reaffirmed.

On the face of it, this all sounds like common sense, but unfortunately, this way of working is not encouraged, or worse still, is actively discouraged in some working environments. If you find yourself in this situation, then you have some additional challenges to overcome simply due to the fact that these edicts are normally defined and enforced through the HR and management guidelines, which in-turn define the policies under which the business operates. You therefore can't simply break or bend these rules at will. We'll cover this in more detail later in the book, but suffice to say that you need to tread very carefully and ensure you lead by example in terms of your behaviors.

Let's break down these concepts in more detail.

Openness and honesty

Openness and honesty are key factors to ensure that the implementation of CD and DevOps is successful. Without these behaviors in place, it's going to be very difficult to break down barriers and implement the much needed changes throughout your organization. You already engaged the majority of the business during the *inspect* stage to obtain honest feedback about the current situation. You now need to ensure that you continue this dialogue with all concerned. Everyone involved in the product delivery process, from developers and testers through change and release controllers to product owners and senior managers, must have a forum they can use to share their thoughts, suggestions, observations, worries, and news.

The most effective way to do this, as was the case during the *inspect* stage, is via face-to-face human interaction, be this in person or virtually via video conference systems (remember that video is preferable to voice). There is one potential drawback to this approach—getting everyone in the same place at the same time. We'll look at some ways to overcome physical environment challenges later; if face-to-face is not wholly viable all the time, you can look at other options such as online collaboration tools (Campfire, for example), real-time forums (Yammer, for example), or group chat systems (IRC or HipChat, for example). Links to the aforementioned tools can be found in *Appendix A*, *Some Useful Information*.

Whatever approach you choose, it is advisable that you set up some form of etiquette or guidelines so that everyone knows what is acceptable and what is not. Hopefully, common sense will prevail. What should not prevail is a heavy-handed policing or moderation of the content as this will discourage openness and honesty and ultimately make the solution redundant.

Let's look at what this means in terms of contribution from our three personas:

- Stan can use his influence to ensure his peers understand the importance of openness and honesty and that they encourage their teams to exhibit these behaviors.
- Devina and Oscar can work together to implement the aforementioned tools that can help enhance communications across the organization. They can also influence their peer groups to be more open and honest.

As you go through the implementation of CD and DevOps, it is extremely important that you have regular open, honest, and truthful feedback from all concerned in terms of what is working with the implementation and, more importantly, what is not. Again, the simplest and most effective way is face-to-face human interaction; simply walk around and ask people. Again, if this is not wholly viable, then you should consider sort of lightweight survey solutions (such as survey monkey). The word 'lightweight' is important here as no one will provide feedback on a regular basis if they have a 10-page questionnaire to fill out every few weeks.

If you follow or use an agile methodology and run regular retrospectives, ask those running these sessions to forward on any feedback related to your implementation.

You're hopefully getting an idea of what open and honest dialogue is all about, but what about courageous dialogue, where does this come into the equation?

Courageous dialogue

There will be times when someone lower down the food chain will have an opinion or a view on how those above them help or hinder the product delivery process. You might have individuals whose views are at odds with specific parts of the business, or indeed, other individuals. It takes guts for an individual to speak up about something like this, especially within a corporate environment. For these people to speak up, they want to be sure that what they say (within reason, of course) is not taken as a black mark on their record. They need to be given a **de-militarized zone** (**DMZ**), where they can share their ideas, views, and opinions—where they can point out the emperor's new clothes.

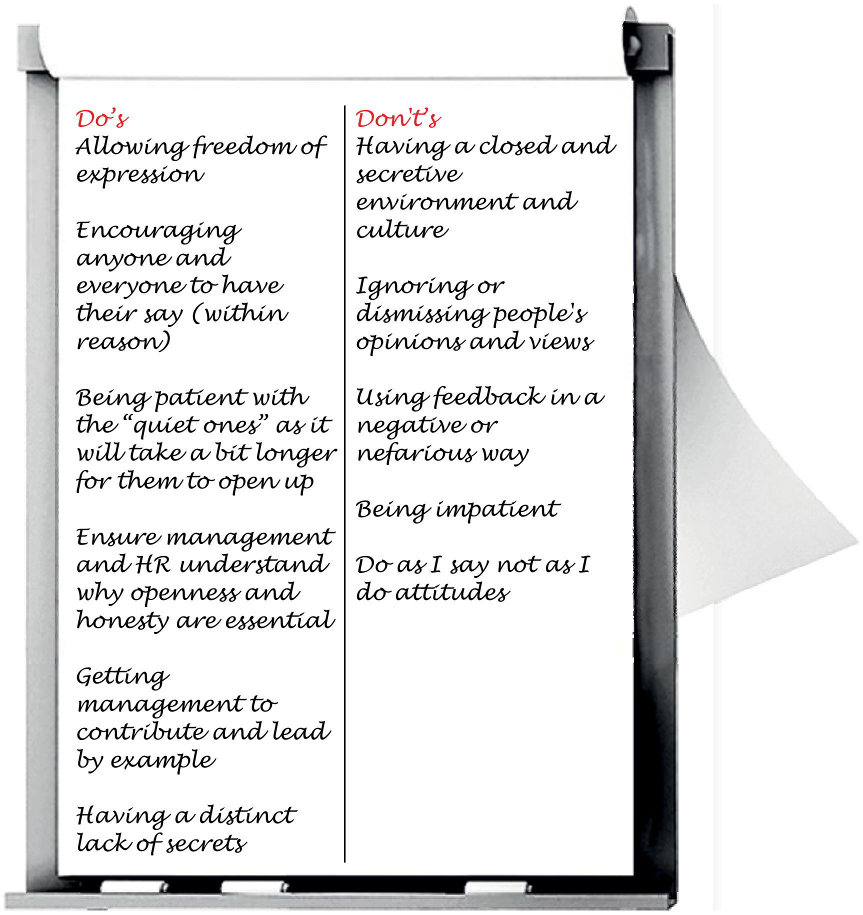
You should work with the management team, and HR, if need be, to ensure that there is a forum for this type of dialogue as it is very important. The content might not be enlightening, but if you have a number of people saying the same thing, then there is a good chance that something needs to be addressed. At the very least, you can work with the management types to implement some sort of amnesty or a way for anonymous feedback to be collected—a suggestion box or online surveys, for example.

One important thing to also take into account is the *quiet ones*. Generally speaking, there are two distinct types of personality traits: *introverted* and *extroverted*. The *extroverts* are the ones that are not afraid to interact, talk, and discuss their views and feelings in public. For *extroverts*, open, honest, and courageous dialogue isn't something they would shy away from. *Introverts* are the opposite, and will more often than not simply close down or just go with the flow in these situations. You, therefore, need to be very mindful of this fact and ensure everyone has the opportunity to contribute and voice their opinions. It might seem like additional work, but from experience, it will be worth it as the contributions from the *introverts* are normally well considered and enlightening.

> If you have difficulty spotting the different types, then here's one easy tip: *extroverts* talk to make their brains work whereas *introverts* use their brains to make their mouths work.

Let's be very open, honest, and courageous about how easy it will be to implement and embed these sorts of behaviors into normal ways of working—it is not. It will be challenging, complex, time consuming, and at times, very frustrating. However, if you persevere, and it starts to work (and it will), you'll find it's a very effective way to work. You will find that once openness and honesty are embedded into the normal ways of working, things really start coming together.

Let's summarize what we've covered so far:



Do's	Don't's
Allowing freedom of expression	Having a closed and secretive environment and culture
Encouraging anyone and everyone to have their say (within reason)	Ignoring or dismissing people's opinions and views
Being patient with the "quiet ones" as it will take a bit longer for them to open up	Using feedback in a negative or nefarious way
Ensure management and HR understand why openness and honesty are essential	Being impatient
Getting management to contribute and lead by example	Do as I say not as I do attitudes
Having a distinct lack of secrets	

What might not be obvious is the fact that the physical environment is something that can and does cause further complications when looking at encouraging open and honest dialogue and behaviors. We'll now take a look at this.

The physical environment

Some of you might be lucky enough to work in nice, airy, open-plan offices with plenty of opportunity to wander around for a chat and line-of-sight visibility of people you collaborate with. The reality is that you might not be so lucky and will have teams physically separated by office walls, flights of stairs, the dreaded cubicles of doom, or even time zones. At this point, let's hypothesize that the office space is not open-plan and there are some physical barriers. There are a few things you can look at to remove some of these barriers:

- Keep the office doors open or, if possible, remove them altogether.
- Set aside an area for communal gatherings (normally in the vicinity of the coffee machine) where people can chat and chill out.
- Have regular (weekly) sessions where everyone gathers (normally in the vicinity of coffee and doughnuts) to chat and chill out.
- Get a foosball table; it's amazing how much ice is broken by having a friendly foosball competition within the office.
- If you use scrum methodology (or similar) and have separate teams locked away in offices, each holding their daily stand-up in private, then hold a daily scrum of scrums (or stand-up of stand-ups), and have one person from each team attend it.
- See whether some of the partition walls can be removed.
- If you have cubicles, remove them, all of them. I personally think that they are the work of the devil and produce more of a negative environment than having physical walls separating teams.
- See whether an office move-around is possible to get people working closer together, or at the very least, mix things up.
- As previously mentioned, look into implementing some sort of collaborative forum/messaging/chat solution, which everyone can use and have access to. You can also inject a bit of fun and innovation using tools such as Hubot (`https://hubot.github.com`), which might encourage more people to use the solution.
- Stop relying on e-mail for communications and encourage people to talk—have discussions, mutually agree, and follow up with an e-mail, if need be.

These are, of course, merely suggestions based upon a very broad assumption of your environment, and you will no doubt have better ideas. The end game here is to start removing the barriers, both virtual and physical.

Let's see what our personas can do to help:

- Stan, the manager, can work within his peer group to convince those above of the importance of changes to the physical environment. Trying this alone, especially when money needs to be spent, might be challenging, so having many *management* voices saying the same thing will add weight.
- Devina and Oscar can work together to make small changes and run experiments, for example, to be seen to have face-to-face discussions, rather than via e-mail, or take over an area of the office and sit together.

We'll now move on from the seemingly simple subject of openness and honesty to the seemingly simple area of collaboration.

Encouraging and embracing collaboration

As you set out on your journey to implement CD and DevOps, you will no doubt be working with the assumption that everyone wants to play ball and collaborate. Most of the business has been through an exercise to capture and highlight the shortcomings of the incumbent process, and you all did this together in a very collaborative way; surely they want to continue in this vein?

At first, this might well be very true; however, as time goes on, people will start to fall back into their natural siloed position. This is especially true if there is a lull in the CD and DevOps implementation activity—you might be busy building tools or focusing on certain areas of the existing process that are most painful. Either way, old habits will sneak back in if you're not careful.

It is, therefore, important that you keep collaborative ways of working at the forefront of people's minds and encourage everyone to work in these ways as the default mode of operation. The challenge is to make this an easy decision (for example, working collaboratively is easier to do than not).

You must keep your eyes and ears open to ensure you get an early indication when things slip back. If you have built up a network, use it to find out what's happening.

There are many ways to encourage collaboration, but you need to keep it lightweight and ensure that those you are encouraging don't feel that this way of working is being forced on them; they need to feel it's their idea. Here are some simple examples:

- Encourage everyone to use your online collaborative forum/messaging/chat solution rather than e-mail—even incentivize its use at first to get some buy-in.
- If the norm is for problems to be discussed at a weekly departmental meeting, rather than having a 5-minute discussion at someone's desk, cancel the departmental meeting and encourage people to get up and walk and talk.
- If the norm is *headphones on and heads down*, you should discourage this as it simply encourages isolation and stifles good old-fashioned talking to each other. If people like to listen to music while working, you can consider something radical, such as a jukebox or some Sonos/networked speakers.
- Even if you don't follow a scrum methodology, use the daily stand-up technique across the board—you can even mix it up across teams so people can move between the stand-ups.
- Install some magnetic whiteboards around the office space, which will encourage people to get up, mix, and be creative while explaining problems, showing progress, or simply having fun and doodling.
- Ensure you mingle and keep open discussions with all teams—you never know, you might hear something that another person has also been discussing, and you can act as the CD and DevOps matchmaker.

Besides impacting openness and honesty, the physical environment can impact (positively and negatively) the adoption of collaborative ways of working, so you need to be mindful of this.

Let's again see what our personas can do to help:

- Collaboration is not the exclusive realm of engineers. Managers can and should be seen to collaborate. Stan should be actively seen to be collaborating, and if tools have been implemented, he and his peers should actively use them.
- Devina and Oscar should practice what they preach and be highly visible when collaborating. Even simple things such as encouraging developers and operations engineers to go to the same pub on a Friday lunchtime can make a difference.

As collaboration becomes embedded within the business, you will see many changes come to life. At first, these will be quite subtle, but if you look closely you'll soon start to see them: more general conversations at peoples' desks, more *I'm trying to do X but not sure of the best way—anyone fancy a chat over coffee to look at the options?* in the online chat room, and more background noise as people talk more or share the joke of the day.

Some subtle (or sometimes not so subtle) PR might help, for example, posters around the office, coffee mugs, or even prizes for the most collaborative team; anything to keep collaboration in sight and mind.

Let's leave collaboration for now and together move on to innovation and accountability.

Fostering innovation and accountability at grass roots

If you're lucky enough to work (or have worked) within a technology-based business, you should be used to having innovation as an important and valued input for your product backlog and overall roadmap. Innovation is something that can be very powerful when it comes to implementing CD and DevOps, especially when this innovation comes from the grass roots.

Many of the world's most successful and most used products have come from innovation, so you should help build a culture throughout the business where innovation is recognized as a good and worthwhile thing rather than a risky way of advancing a product. Most engineers thrive, or at least enjoy, innovation, and if truth be told, this was most probably one of the major drives for them choosing to become engineers—this and the fine wine, fast cars, and the international jet-setter lifestyle (okay, this might be stretching things a bit too far).

This isn't to say that they can all go off and do what they want; there are still products to deliver and support. What you need to do is allow some room for investigation and experimentation—rekindle the R in R&D. Innovation is not just in the realm of software; there might be different ways of working or product delivery methodologies that come to light that you can and should be considering.

[Agile techniques such as Test-driven development (TDD), scrum, and Kanban all started out as innovative ideas before gaining wider notoriety.]

Despite normal convention, innovation is not the exclusive right of solutions and systems architects; anyone and everyone should be given the opportunity to innovate and contribute new ideas and concepts. There are many ways to encourage this kind of activity (competitions, workshops, and so on), but you need to keep it simple so that you get a good coverage across the business. One simple idea is to have a regular innovation forum or get-together, which allows anyone and everyone to put forward, and if possible, prototype an idea or concept.

Innovation can increase risk, new things always do; therefore, the engineering teams must understand that with the freedom they are given to make decisions and choices comes responsibility, ownership, and accountability for the *new stuff* they come up with, produce, and/or implement. They cannot simply implement shiny new toys, tools, processes, and software and hand them off to someone else to support. The *somebody else's problem* (*SEP*) or *throw it over the wall* approaches will no longer work.

A good example of this is the ACME systems plan to allow developers to deploy code directly to production. On the face of it, this is very much what CD and DevOps is all about, but one simple question caused the plan to falter. The question was *who is going to hold the pager?*, or to bring this into the 21st century, *are the developers going to be on call when things go wrong out of hours?* Ultimately, you need everyone involved in the process of delivering and supporting software to have the same strong sense of accountability so that the question need not be asked.

So, how can these values and behaviors be instilled into your organization? Let's see what our personas can do to help:

- Stan should actively allow time for his team members to *try things out* or *experiment*, be this by setting aside some 10 percent time or simply encouraging them to put forward their ideas and suggestions for product or productivity advancement.
- Devina and Oscar should actively pursue this agenda as part of discussions with their managers during one-to-one's or team meetings. To help things along, using some spare time on an idea, and then presenting it back, might be a good thing as it shows commitment and that you're serious. Working together collaboratively will also add credence.

As your adoption of CD and DevOps matures, you will find that innovation and accountability will become commonplace as the engineering teams (both software and operations) will have more capacity to focus on new ways of doing things and improving the solutions they provide to the business. This isn't just related to new and shiny things; you'll find that there is renewed energy to revisit the technical debt of old to refine and advance the overall platform.

Believe it or not, sometimes things will go wrong. We'll now look at how things that don't go so well should be dealt with, and why a culture of blame is not a good thing to have.

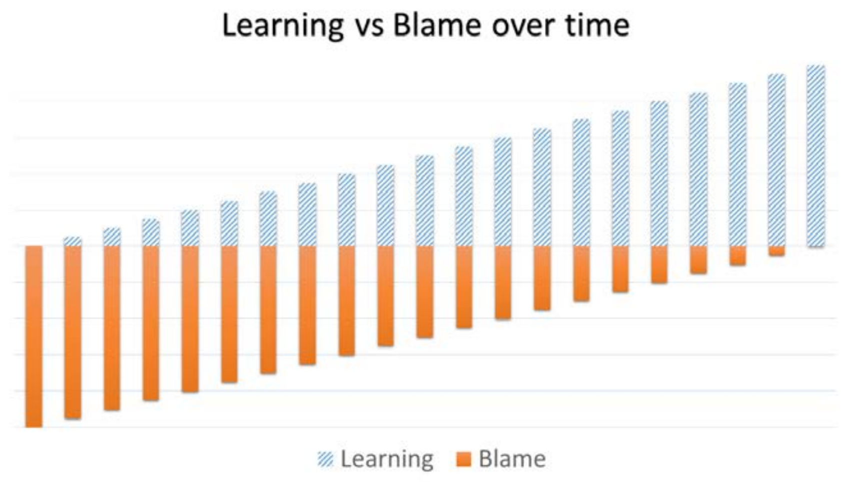
The blame culture

Encouraging a fail-fast way of working is a critical element to good agile engineering practice; it is all well and good saying this, but this has to become a real part of the way your business works; as they say, actions speak louder than words. If, for example, we have a manager who thinks that pointing the finger and singling people out when things go wrong is a good motivational technique, then it's going to be very difficult to create an environment where people are willing to put themselves out and try new things. A culture of blame can quickly erode all of the good work done to foster a culture of openness, honesty, collaboration, innovation, and accountability.

Ideally, you should have a working environment where when mistakes happen (we're only human and mistakes will happen), instead of the individual(s) being jumped on from on high, they are encouraged to learn from the mistake, take measures to make sure it doesn't happen again, and move on. No big song and dance. Not only this, but they should also be actively encouraged to share their experiences and findings with others, which enforces all the other positive ways of working we covered so far.

Blame slow, learn quickly

In a commercial business, it might sound strange and be seen as giving out the wrong message (for example, you might seem to be ignoring or encouraging failure), but if lessons are being learned, and mistakes are being addressed quickly out in the open, then a culture of diligence and quality will be encouraged. Blaming individuals for a problem that they quickly rectify is not conducive to a good way of working. Praising them for spotting and fixing the issue might seem wrong to some, but it does reinforce good behaviors. The following illustration shows the possible impact of a *blame slow, learn quickly* culture:

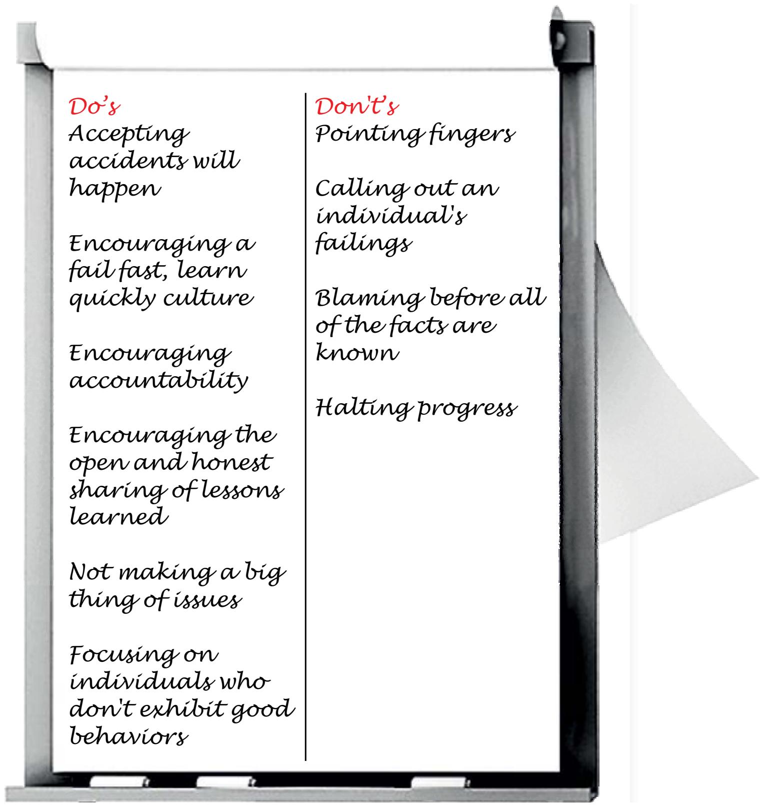


As blame diminishes, learning will grow as people will no longer feel that they have to keep looking over their shoulders and only stick to what they know or are told to do

> If managers are no longer preoccupied with the small issues, they can focus on the individuals who create issues but don't fix them or take accountability.

As you can no doubt understand, this culture change is not going to be easy for some, especially for the managers who have built up the reputation of being Mr. or Mrs. *Shouty*. Sometimes, they will adapt, and other times, they might simply step out of the way of progress—as the groundswell gains momentum. They will have little choice but to do one or the other.

Let's again summarize this:



Do's	Don't's
Accepting accidents will happen	Pointing fingers
Encouraging a fail fast, learn quickly culture	Calling out an individual's failings
Encouraging accountability	Blaming before all of the facts are known
Encouraging the open and honest sharing of lessons learned	Halting progress
Not making a big thing of issues	
Focusing on individuals who don't exhibit good behaviors	

Removing the threat and culture of blame from the engineers' working life will mean that they are more engaged, willing to be more open and honest about mistakes, and more likely to want to fix the problem quickly.

Of course, there is a large element of trust required on all sides to make this work effectively.

Building trust-based relationships across organizational boundaries

Now, I will freely admit that this does sound like something that has been taken directly from an HR or management training manual; however, trust is something that is very powerful. We all understand what it is and how it can benefit us. We also understand how difficult things can be with a complete lack of it. If you have a personal relationship with someone, and you trust them, the relationship is likely to be open, honest, and a long and fruitful one. Building trust is extremely difficult; you don't simply trust a colleague because you have been told to do so—life doesn't work this way. Trust is earned over time through peoples' actions.

Trust within a working environment is also a very hard thing to build. There are many different reasons for this (insecurity, ambition, reputation, personalities, and so on), so you need to tread carefully. You also need to be patient as it's not going to happen overnight.

Building trust between traditional development and operations teams can be even harder. There is normally a level or an undercurrent of distrust between these two areas of the business:

- The developers don't trust that the operations team know how the platform actually works or how to effectively investigate issues when they occur.
- The operations team don't trust that the developers won't bring the entire platform down by implementing dodgy code.

This level of distrust can be deeply ingrained and is evident up and down the two sides of the business. These types of attitudes, behaviors, and the culture they create are all too negative. It's hard enough to get software developed, shipped, and stable without playing silly games with who does what and who doesn't. If you have an environment like this, then the business needs to grow up and act its age.

There is no silver bullet to forge a good trust-based relationship between two or more factions; however, the following techniques proved to work for ACME systems and might help you:

- If you arrange for some off-site CD or DevOps training, ensure that you get a mix of software and operations engineers to attend and ensure they are in the same hotel. You will be amazed how collaborative working relationships start out in the hotel bar.
- If there are workshops or conferences you are looking at attending (for example, DevOpsDays), make sure there's a mix of Devs and Ops in attendance and a hotel bar.

- If you are a manager, be very mindful of what promises and/or commitments you make and ensure you either deliver against them or you are very open and honest as to why you didn't/couldn't.
- If you are an engineer, act in exactly the same way.
- If you have set up an innovation forum (as mentioned previously), encourage all sides to attend and contribute.
- Discourage *us and them* discussions and behaviors.
- If it's viable, try and organize job swaps or secondments across the software and operational engineering teams (for example, get a software engineer to work in operations for a month, and vice versa). This can also include management roles.

We'll now move from trust to rewards and incentives.

Rewarding good behaviors and success

How many of us have worked with or been part of a business that throws a big post-release party to celebrate the fact that against all odds you managed to get the release out of the door? On the face of it, this is good business practice and management 101, and after all, most project managers are trained to include an *end of project party* task and budget in their project plans. This is not altogether a bad thing if everything that was asked for has been delivered on time to the highest quality. Let's try rewording the question.

How many of us have worked with or in a business that throws a big post-release party to celebrate the fact that against all odds you managed to deliver most of what was asked for and only took the live platform offline for 3 hours while they tried to sort out some bugs that had not been found in testing?

If the answer to the question is *quite a few, but it was a hard slog and we earned it*, then you are a fool to yourself. Rewarding this type of behavior is 100 percent the wrong thing to do. The businesses that deliver what customers want and do it quickly are the ones that succeed.

If you want to be a business that succeeds, you need to stop giving out the wrong message. We did say that it was okay to fail as long as you learn from it quickly; we didn't however mention rewarding failure to deliver. You should be rewarding everyone when they deliver what is needed when (or before) it is needed. The word *everyone* is quite important here as a reward should not be targeted at an individual as this can cause more trouble than it's worth. You want to instill a sense of collaboration and DevOps ways of working, so make the reward a group reward, a party, a day out, and so on.

The odd few

Okay, so there might be the odd few who will put in extra effort when times get sticky, and rewarding those individuals is not a bad thing; however, this should not be the norm. If engineering teams (software and operational) are consistently being told to work long days, long nights, and weekends, then there is something wrong with the priority of the work. If, however, they decide to apply some extra effort to overcome some long outstanding technical debt or implement some labor-saving tools to speed things up, then this is completely different, and you should be looking at specific rewards for these specific good behaviors.

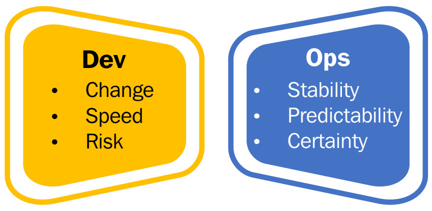
At the end of the day, you want to reward individuals or teams for doing something amazing that is above and beyond the call of duty, rather than simply successfully releasing software. As CD and DevOps ways of working become embedded, you will notice that you don't actually have what you would previously have called releases anymore (they are happening too quickly to notice each one), and therefore, you need to look at other ways to reward. For example, you can look at throwing a party when a business milestone is hit (such as when you reach the next millionth customer), when a new product successfully launches, or simply because it's sunny outside and the bosses want to say thank you.

CD and DevOps will change the way the business operates, and this fact needs to be recognized across all areas. As such, the way you reward people needs to change to instill the good behaviors previously mentioned (openness and honesty, innovation, accountability, and so on). This can be quite a shift for some businesses, and some might even need to implement new reward systems, solutions, or processes to cater for this.

One of the standard ways of rewarding people is via some kind of bonus or incentive scheme. This will also need to change, but first you need to recognize how the current system might foster the wrong behaviors and can stifle your implementation of CD and DevOps.

Recognizing dev and ops teams are incentivized can have an impact

There is a simple and obvious fact that some people might not instantly realize, but it is something that is very real and very widespread throughout the entire IT industry. This fact is that development teams are incentivized to deliver change, whereas operations teams are incentivized to ensure stability and system uptime, thus discouraging change. The following figure highlights this:



Incentivising developers to deliver more quickly is at odds with incentivising operations teams with keeping things stable and safe

If you think about it, the two methods of incentivizing are at odds with each other; operations teams get a bonus for reducing or even stopping change, and development teams get a bonus if they deliver a lot of change. So, how do you square the circle and allow a steady stream of change to flow through without having the operations team up in arms about the fact that their bonus is at risk?

There's no simple answer, but there are some examples you can look at to ease the pain:

Incentive	**Pros**	**Cons**
Have the same incentives across both Dev and Ops.	If you are incentivizing to allow for continuous change, you will increase the potential for having CD and DevOps becoming the norm as everyone involved will focus on the same goal.	There is more risk as people might think that changing things quickly is more important than quality and system uptime.
Including each side of the DevOps partnership in each other's incentive schemes.	If some of the bonus of the software engineering team is dependent on live platform stability, then they'll think twice before taking a risk. If some of the operations engineering team's bonus is dependent on enabling CD, they will think twice before blocking changes just for the sake of it.	If the percentage of the *swap* is small, it might be ignored as the focus will remain on getting the majority of the bonus, which will still encourage the old behaviors.
Replacing the current incentive scheme with one that focuses on good behaviors and encourages a DevOps culture.	This has the potential to remove conflict between the engineering teams (Dev and Ops) and would encourage them to focus on what is important: delivering products customers want and need.	The reality is that it will be quite difficult to get a full agreement, and get it in place quickly, especially in a corporate environment. This doesn't mean it's not something worth pursuing.

Whatever you do in regards to incentivizing and rewarding people, you need to instill a sense of positivity around change, while at the same time ensuring risk is reduced.

Embracing change and reducing risk

In the same vein as fostering innovation and accountability at grass roots, you need to work across the wider organization to ensure they accept the fact that change is a good thing and not something to be feared.

It is true to say that changing anything on the production platform—be it a new piece of technology, a bug fix to a 15-year old code-base, an upgrade to an operating system, or a replacement storage array—can increase the risk of the platform, or parts thereof, failing. The only way to truly eliminate this risk is to change nothing, or simply switch everything off and lock it away, which is neither practical nor realistic. What is needed is a way to manage, reduce, and accept the risk.

Implementing CD and DevOps will do just that. You have small incremental changes, transparency of what's going on, the team that built the change and the team that will support it working hand in hand, a sense of ownership and accountability from the individual(s) who actually wrote the code, and a focused willingness to succeed. The challenge is getting everyone in the business to understand and embrace this as the day-to-day way of working.

Changing people's perceptions with pudding

Getting the grass roots to understand this concept should be quite simple when compared to other parts of the business that are, by their very nature, risk averse. I'm thinking here of the QA teams, senior managers, project and program managers, and so on. There are a few ways to convince them that risks are being controlled, but the best way is via using the *proof of the pudding* methodology:

- Pick a small change and ensure that it is well publicized around the business
- Engage the wider business, focusing on the risk averse, and ensure they are aware; also invite them to observe and contribute (team stand-ups, planning meetings, and so on)
- Ensure that the engineers working on the change are also aware that there is a heightened sense of observation for the change
- As the change is progressing, get the engineering teams involved to post regular blog entries detailing what they are doing, including stats and figures (code coverage, test pass rate, and so on)

- As the release goes through the various environments to production, capture as many stats and measurements as possible and publish them
- When all is done, pull all the above into a blog post and a post-release report, then present them

You might be thinking that this is a vast amount of work, and to be honest, it is if you follow the preceding steps for each and every change you make. What it does do is serve a purpose; it proves to the business that change is good and risks can be controlled and managed. I recommend you follow these steps a few times to build trust and confidence—you can always refine later down the line. Another positive you will find is that it will foster a culture of diligence at grass roots; if they are very aware that the business is keeping an eye on things, especially when things go wrong, then they will think twice before doing something silly.

> It should be noted that even though the steps detailed will generate additional work, this is nothing compared to how some organizations currently function; changes are fully documented and risk assessed, progress meetings are held, the project progress is publicized, and every meticulous detail is captured and documented. Is it any wonder why delivering software can be so painful?

As with anything in life, if you make a small change, the risk is vastly reduced. If you repeat the process many times, the risk is all but removed and habits are formed. To follow this thread, if infrequent releases contain a large amount of change, the risk is large. Make it small and frequent, and the risk goes away. It's quite simple when you look at it this way.

As part of the *proof of the pudding* example, there was a lot of publicizing and blog posting going on. This should not be seen as an overhead, but a necessary part of CD and DevOps adoption. Being highly visible is key to breaking down barriers and ensuring anyone and everyone is aware of what is going on.

Being transparent

As we previously covered, being secretive about what you do and how you do it is not conducive to building an open, honest, and trust-based working environment or culture. If anyone and everyone can see what is going on there should be no surprises. What we're looking for is a culture, and ways of working where change is good and frequent, individuals work together on common goals, the wider business trusts the product delivery teams to deliver what is needed when it is needed, and the operations teams know what is coming. If there is a high degree of visibility across the entire process, anyone and everyone can see this happening, and more importantly, how effective it is.

You should look at the option of installing large screens around the office to display stats, facts, and figures. You might well have something like this set up already, but I suspect these screens display very technical information—system stats, CPU graphs, alerts, and so on. I also suspect that most of these reside in the technical team areas (development, operations, and so on). This is not a bad thing, it's just very specialized, and those of a nontechnical nature might well ignore them or most likely not even know that they exist. See if you can move some of the screens to communal areas of the office or try and find some budget to buy new ones.

You should also complement this highly technical information with very simple, easy to read and understand data related to your CD and DevOps process. You should be looking at displaying the following kinds of information:

- Number of releases this day, week, month, and year against the number yesterday, last week, last month, and last year
- The release queue and progress of the current release going through the process and who initiated it
- Production system availability (current and historical)
- If you use an online scrum/Kanban board (such as **AgileZen** or **Trello**), then consider having this data displayed to show your backlog, work in progress, and work completed along with related stats such as velocity and burndown
- The latest business information such as share price, active user numbers, the number of outstanding customer care tickets, and so on

The last point is very important. You should publish, display, and advertise complementary information and data that is business-relevant, rather than simply focusing on technical facts and figures. This will help to heighten engagement and awareness outside of the technical teams. Having this information visible as you progress through your adoption and implementation of CD and DevOps will also provide proof that things are improving.

Summary

We covered quite a lot of ground in terms of the human side of implementing CD and DevOps throughout this chapter. Hopefully, it has been impressed upon you that the culture in which you operate is essential for CD and DevOps to work. When it comes to collaboration, you will find that trust, honesty, and openness are powerful tools that allow individuals to take responsibility and accountability for their actions. Rewarding good behaviors and removing blame will also help drive adoption.

At this point, you should have a plan and some insight into the importance of culture and behaviors when implementing CD and DevOps. In *Chapter 5*, *Approaches, Tools, and Techniques*, we'll look at some things that will help as you drive forward.

5
Approaches, Tools, and Techniques

The preceding chapters focused on surfacing the issues within your delivery process and defining a plan to address these by (hopefully) implementing CD and DevOps. The chapters also taught you how to set up the working environment to ensure that you're successful. The following chapters will focus on the steps you need to consider when executing against the *plan*. First, we will focus on some of the technical aspects.

There will be quite a lot of things to cover and take in, some of which you will need, some of which you might have in place, and some of which you might want to consider implementing later down the line. Whatever the situation, what follows should provide some small chunks of wisdom, or information at the very least, that you can adapt or adopt to better fit your requirements. I should point out that the majority of this chapter is focused on software engineering (that is, the Dev side of the DevOps partnership); however, quite a lot of the areas covered are as relevant to system operations as they are to software engineering.

In this chapter, we'll cover the following topics:

- Examples of proven engineering best practice
- Some simple rules and tools
- Automated CI and CD tooling
- The value of monitoring and metrics
- How a manual process can be just as effective as tools

It is worth pointing out at this point that the tools and approaches mentioned are not mutually exclusive—it is not a case of all or nothing—you just need to pick what works for you. If you want to look at the tools in more detail, take a look at *Appendix A*, *Some Useful Information*. Let's start with some good old-fashioned engineering best practice.

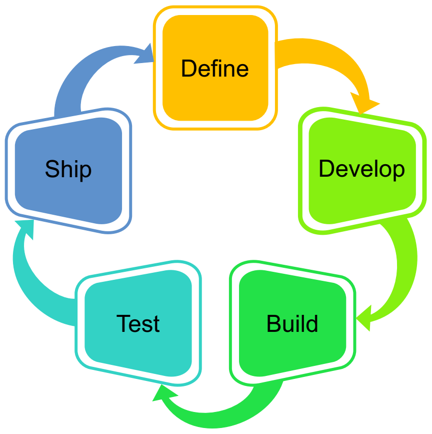
Engineering best practice

For those of you who are not software engineers or from a software engineering background, your interest in how software is developed might be extremely minimal. Why, I hear you ask, do I need to know how a developer does their job? Surely, developers know this stuff better than I do? I doubt I even understand 10 percent of it anyway!

To some extent, this is very true; developers do (should) know their stuff, and having you stick your nose in might not be welcome. However, it does help if you at least have an understanding or appreciation of how your software is created, as it can help you identify where potential issues could reside. Let's put it another way: I have an understanding and appreciation of how an internal combustion engine is put together and how it works, but I am no mechanic—far from it in fact. So, when I take my car to a mechanic for a routine service, I will question why he has had to replace all of the exhaust system because he found a fuel-injector problem—in fact, I think I would vigorously question why.

It is the same with software development and the process that surrounds it. If you're not technical in the slightest, it still helps to have an idea of how things are done. So, when you have to question why things are done in a specific way, you may be able to spot the slippery "blind them with technobabble—that'll scare them off" types.

CD is based on a premise that quality software can be defined, developed, built, tested, and shipped very quickly many times in quick succession—ideally, we're talking hours or days at the most. The following figure depicts this cycle:



A typical software delivery cycle

When looking at a typical *agile* development project, the first four steps (starting from Define) happen quite quickly and—depending on the development techniques used—might happen in a slightly different order. In a *waterfall* style project, the first four steps might take a while longer, sometimes considerably so. What both methods suffer from is their shipping step. This is normally the step that takes the time, effort, and resources—as you will have no doubt found out during the *inspect* stage (see *Chapter 2*, *No Pain, No Gain*). CD and DevOps can help reduce this pain and the time taken to ship, but you need to ensure beforehand that your engineering practices are as optimal as they can be; otherwise, you might not reap the benefits of a CD and DevOps approach.

Let's look at some fundamentals in terms of software engineering:

- Commit and merge small code changes frequently into a source control repository
- Do not make code overly complex and keep it maintainable and documented
- Avoid introducing breaking changes
- If you have automated tests, run them very frequently (ideally, continuously)
- If you have a CI solution that controls your build and test suite, run it frequently
- Use regular code reviews
- Refactor as you go

As you can see, the preceding list is not overly complex, and to most software engineers who work on modern software development projects, this list is pretty much common sense and common practice.

You might be able to speed up your software delivery process without using the preceding steps, but it will not be easy, nor will it be pretty or sustainable. What you are really trying to do is find issues as soon as possible. If you're making small changes frequently and reviewing/testing them regularly, you'll have a better chance at spotting potential bugs early on.

What it comes down to is that if you do not find software problems early on, these problems will slow you down later on and will influence the overall project. To put it another way, if there are next to no bugs when the software is delivered, releasing it should be a doddle.

Let's break these down a little further, starting with **source control**.

Source control

Source control is a must for modern collaborative software development. There are many different source control tools and solutions available (see `http://en.wikipedia.org/wiki/Comparison_of_revision_control_software` if you don't believe me). These range from commercially licensed (such as Team Foundation Server (TFS) or ClearCase) to open source ones (such as GIT, SVN, or Mercurial), so you will not struggle to find one that meets your needs and/or budget. If your code is in source control, it is versioned, it is available to anyone who has access, and it is secure.

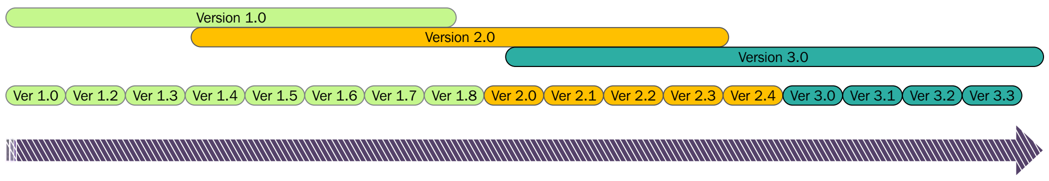
There are books and reference materials aplenty available that cover this subject in much more depth, so I will not dwell on it here. Suffice to say, if you do not have a source control solution, then implement one. Now!

> It should be noted that the use of source control should not be restricted to software source code. You can (and should) utilize source control for anything that can be changed within your system as a whole. This includes things such as system configuration, start-up scripts, server configuration, network configuration, and so on. This is even more important if you are considering automated provisioning solutions (we'll cover more on this later.)

A source control solution is an essential tool for CD and DevOps adoption, as is the practice of keeping changes small and frequent.

Small, frequent, and simple changes

Keeping changes small means the impact of the change should also be small, the risks reduced, and the opportunities for change increased. This sounds overly simplistic, but this is also very true. The following diagram gives some indications on how this could look:



Large releases versus small incremental releases

What this diagram also tries to illustrate is the difference between working with large *clumps* of changes and small *chunks* of incremental changes. Typically, the large *clump* will consist of many small code changes—sometimes hundreds or thousands of lines of code—which are developed in isolation and then brought together at the last minute, tested, and merged into the core codebase. This, as you can imagine, brings with it many challenges and, if truth be told, lots of waste. Working with a small amount of change—maybe a few lines of code—doesn't bring this overhead and vastly reduces complexity and the potential for late-breaking quality issues.

Another disadvantage of working with large *clumps* of change is something I like to call **version overlap**. Typically, when development on a version has been completed, the task of bringing it all together and getting it working starts. As it doesn't normally need the whole team to do this, while the poor souls who have been given this task get on with it, the rest of the team start working in parallel on the next version. This can create a whole new set of problems—for example, the need to create more branches in source control or the fact that additional changes will be made to code that hasn't, as yet, been fully verified.

You might be thinking that there's no real point in delivering in *chunks* if you don't currently have the luxury of shipping your code frequently. To some extent, this is true; however, once you have CD and DevOps up and running, you'll be working in this mode. So, why not start getting used to it and gain some advantages beforehand?

It should be noted that this practice need not be restricted to software engineering; it is just as relevant to changes in the system operations area as well. For example, making a small, isolated tweak to server configuration is much safer and easier to control and monitor than making sweeping changes all at once. If you make small changes, you have a much better chance of seeing if the change had an impact (positive or negative) on the overall operation of the platform.

We'll now move on to how breaking changes can affect your platform.

Never break your consumer

Your platform will most probably be complex and have quite a few dependencies—this is nothing to be ashamed of and is quite normal. These dependencies can be classified as relationships between *consumers* and *providers*. The *providers* can be anything from shared libraries or core code modules to a database. The *consumers* will call/execute/send requests to the *providers* in a specific way as per a predefined API spec (sometimes called a service contract). If, for some reason, the *provider* has been changed so that what is returned is not what the *consumer* expects or differs from what was originally agreed upon, the *consumer* might behave in an unexpected way, throw an error, or simply crash.

The rule of thumb here is to avoid these situations where possible. Avoidance techniques can range from creating a dependency map of your entire platform through running impact analysis of each change to using simple peer code reviews to ensure that nothing is amiss. Do some research and see what fits best.

It should be noted that there might be exceptions when these breaking changes cannot be avoided. In such cases, you should have a strategy planned out to cater for this. An example strategy would be to accommodate side-by-side *provider* versioning, which would allow you to run more than one version of a software asset at the same time on the same platform, thus allowing *consumers* to migrate from one version of a *provider* to another over time.

Within the system operations area, the *never break your consumer* rule should also apply. For example, the software platform could be classed as a *consumer* of the server operating system (the *provider*); therefore, if you change or upgrade the operating system, you must ensure that there are no breaking changes that will cause the *consumer* to fail.

There might be times when the *consumer*/*provider* relationship fails, as the person or team working on the *provider* is unaware of the relationship. To overcome this, or at least to minimize the risk, open and honest peer-working practices should go some way to help.

Open and honest peer-working practices

There are many different *agile* software delivery methodologies in use today, but they all revolve around some form of collaborative and transparent ways of working, where those who are writing code regularly share their workings with others. Even the most capable engineer on the planet is human, and they can/will make mistakes—they will, of course, be reluctant to admit this.

Having a process where change is regularly reviewed by the peer group—which could include peers outside of the development team—or simply reviewed by someone sitting next to you will, among other things, help find issues early, help share knowledge, and help build relationships across the peer group. There are even tools available that will help you with this process.

> Having an open, honest, and transparent peer-review process is as important within an operations team as it is within a development team. Changes made to any part of the platform run a risk, and having more than one pair of eyes to review will help reduce this risk. As with software code, there is no reason to not share system configuration changes.

One normally unforeseen advantage of peer working is that if a change fails to get through peer review, the impact on the production system is negated. It's all about failing fast rather than waiting to put something live to find it fails.

Fail fast and often

Failing fast and often might seem counterintuitive, but it's a very good ethos to work to. If a bug is created but it is not found until it has gone live—which is sometimes referred to as an escaped defect—the cost of rectifying the bug could be relatively high (it could be a completely new release). In addition, allowing defects to escape into the wild might impact your customers, your reputation, and possibly, your revenue. Finding faults early is a must.

Some engineering techniques such as **Test-driven development** (**TDD**) are based on the principle of exposing faults with software very early on in the process. If the code written fails to clear the first hurdle of tests, it goes no further.

> This might sound strange—especially for the managers out there—but if bugs are found early on, you should not make a big issue of it, and you should not chastise people who have introduced the bugs (remember the no-blame culture covered in *Chapter 4*, *Culture and Behaviors*). There might be some banter around the office but nothing more. Find the problem, find out why it happened, fix it, learn from it, and move on.

To effectively find issues early on, you need to run your tests on a regular basis. The use of automation can help here.

Automated builds and tests

Finding faults early provides rapid feedback to the engineer as to whether the changes they have made actually work (or not as the case might be). You could, of course, use manual testing processes to achieve this, but this can be cumbersome, inconsistent, prone to error, and not always fully repeatable.

Implementing automation will help speed things up, keep things consistent, and, above all, provide confidence. If you are repeatedly running the same builds and tests against your software and getting the same results, it's a strong bet that the software works as expected. It is, therefore, plausible that if you change one thing (remember small and frequent changes) and the previously working builds or tests fail, there is a very good chance that the change has broken something.

Test data can be a bit of a thorny issue and can cause more problems than it solves. A good rule of thumb would be to create and tear down the test data you need when the test is being run; this way, the outcome of the test will not be impacted by pre-existing data, which might itself be faulty.

How you go about choosing the best approach, tools, and techniques for automation can be daunting, but applying the KISS rule will help; start small, focusing on your major pain points, and then evolve as your confidence grows. Suffice to say that the investment in automation will reap rewards. One such reward is the ability to use continuous integration.

Continuous Integration

Continuous Integration (**CI**) is a tried and tested method of ensuring that a given software asset builds correctly and *plays nicely* with the rest of the platform. The keyword here is *continuous*, which, as the name implies, is as frequent as possible (ideally, upon each change). Just like the aforementioned source control solutions, there are quite a few CI tools available, from the commercially licensed (such as Bamboo, Go, or TeamCity) to open source ones (such as Jenkins, Hudson, or CruiseControl). A pretty comprehensive list is available at `http://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software`—which also provides a good indication of which CI solutions play nicely with each of the source control solutions currently available.

CI systems are basically software solutions that orchestrate the execution of your automated scripts when certain events occur, for example, when you commit a change to source control. These *CI jobs* contain a list of activities that need to be run in quick succession; for example, get the latest version of source from source control, compile to an executable, deploy the binary to a test environment, get the automated tests from source control, run them, and capture the results.

If all is well, the CI job completes and reports a success. If it fails, it reports this fact and provides detailed feedback as to why it failed. Each time you run a given CI job, a complete audit trail is written for you to go back and compare results.

CI tools can be quite powerful, and you can build in simple logic to control the process—for example, if all of the automated tests pass, the CI tool can add the executable to your binary repository, or if something fails, it can e-mail the results to the engineering team. You can even build dashboards or radiators to provide an instant and easy-to-understand visual representation of the results.

CI solutions are a must for CD. If you are building and testing your software changes on a frequent basis, you can ship frequently. Using a CI solution will also help in building the DevOps relationships, as the Ops half of the relationship will be able to see proof that builds and tests have been successful. They could also be heavily involved in defining and creating some of the dashboards.

Your CI solution will help you build and test your software to ensure it is fit to ship. You now need to ensure you can take into account how to move this built software towards the production environment. Before we do this, there is one more thing to take into account.

Using the same binary across all environments

When a software asset is ready to be shipped, it has normally been built/compiled into an executable. To ensure that the software functions in the production environment as it did in your development and/or test environment(s), you need to ensure that this self-same unchanged binary is used. This might sound like obvious common sense, but sometimes, this is overlooked or simply ignored.

There might be issues related to this which are not obvious, for example, if the software needs access to certain secure data (such as credentials to connect to a database). If you have traditionally *baked* this into the binary at build time (or worse still, hardcoded this in source), you will need to change this practice. This might well require some additional development work and should not be taken lightly.

Let's now look at the optimal use of environments for various steps within your development process.

How many environments are enough?

How many environments you need depends on your ways of working, your engineering setup, and, of course, your platform. Suffice to say that you should not go overboard. There might be a temptation to have many environments set up for different scenarios: development, functional testing, user-acceptance testing, and performance testing. If you have the ability to ensure that all the environments can be kept up to date (including data) and you can easily deploy to them, then this might be viable. The reality is that having too many environments is counterproductive and can cause far too much noise and overhead.

The ideal number is two—one for development and one for production. This might sound like an accident waiting to happen, but many small businesses manage fine with such a setup.

For your development environment, you could look at virtualization on the desktop (using a tool such as Vagrant or Docker) where you can pretty much spin up a virtual copy of your production environment on a developer's workstation (on the presumption that there's enough horse power and storage available).

It's a fact of life that as a business grows, so does the need to be risk averse. This normally means more checks and balances are needed, which lead to burdensome processes and inevitably more and more environments being created to allow for these processes to work. It need not be this cumbersome. My advice is to be very prudent and somewhat ruthless when it comes to the number of environments you need (need rather than want). Get the Dev and Ops team together with the team(s) that look after change and risk management to ensure that there is a middle ground that allows for speed of delivery while, at the same time, allows for a reduced/ managed risk.

When your CD and DevOps adoption has matured, you will be shipping code very quickly. Try to imagine how much overhead managing and maintaining multiple environments will be and how it could slow you down.

One thing that might sway this decision is having the ability to develop against a production-like environment.

Developing against a production-like environment

As we've mentioned previously, there are many ways to ensure that a software change will play nicely when it is deployed to the production platform. By far the easiest is to actually develop and test against an environment that is as close to your production environment as possible.

The utopia would, of course, be to develop and test against the production environment, but this is very risky, and the possibility that you could cause outage—albeit inadvertently—is quite high. Therefore, having an environment that closely resembles the production environment is the next best thing.

[As mentioned previously, there are tools available to allow developers (or anyone who wants to) to spin up a virtual version of your production environment.]

With this kind of environment in place, you can then develop in isolation, but be more confident that the changes that are being made will play nicely when you ship the software to the production environment. There is one thing you will need to take into account with this approach—which is also true of any development or test environment—that being realistic like-live data. You need to consider how you can create a realistic subset of your production data. Alternatively, you could consider allowing access to production databases (as long as this access is secure). I suggest you work with your **database administrator** (**DBA**) and **Security Operations** (**SecOps**) team to work out the best approach.

You are starting to get all of the building blocks in place to realize your goal. There are still, however, few other hurdles to get over, one of them being how you seamlessly move the fully built and tested software asset through to production. Here is where CD tooling comes into play.

CD tooling

You could consider CD tools as being the natural evolution of the aforementioned CI tooling. Instead of controlling and orchestrating the build and test process, CD tools control and orchestrate the act of deploying the built software components to the various environments you have. Currently, there doesn't seem to be a standard set of standalone CD tools. Instead, many vendors and tool creators have extended their products to include this functionally.

Before committing to a tool or solution, you should consider some of the following questions:

- Can it deploy the same binary to multiple environments?
- Can it seamlessly access the binary and source repositories?
- Can it remotely invoke and control the installation process on the server that it is being deployed to?
- Is it capable of deploying database changes?
- Does it have the functionality to allow for queuing up of releases?
- Does it contain an audit of what has been deployed, when, and by whom?
- Is it secure?
- Can it interact with the infrastructure to allow for no-downtime deployments?
- Can it/could it orchestrate automated infrastructure provisioning?
- Can it be extended to interact with other systems and solutions such as e-mail and change-management, issue-management, and project-management solutions?
- Does it have simple and easy-to-understand dashboards that can be displayed on big screens around the office?
- Can it interact with and/or orchestrate the CI solution?
- Will it grow with our needs?
- Is it simple enough for anyone and everyone to use?

If you manage to find a tool (or collection of tools) that answers most/all of these questions, then congratulations! If not, then you should seriously consider building some of your own tools (or possibly bolting some existing tools together). After all, you want to enhance your process with tools rather than restrict your process because of the tools you chose.

[It should be noted that deployment of database changes via CD tooling can be a very complex thing to do compared to deploying software. If this is a must-have requirement, you should take time and run some trials before committing to a tool or approach.]

Let's spend some time digging into a couple of the considerations listed earlier, starting with automated provisioning.

Automated provisioning

If you are lucky enough to have a platform (or have re-engineered your platform) that can run on virtual infrastructure, then you can consider automated provisioning as part of the deployment process.

[Automated provisional tools are available for non-virtualized (read physical) environments and platforms, but these can be overly complex and costly. If you can utilize virtualization, you should.]

Automated provisioning is nothing new. The likes of Amazon and Google have been providing their *cloud-based servers* for a while now, and you have been able to provision what servers you want when you need them (for a price).

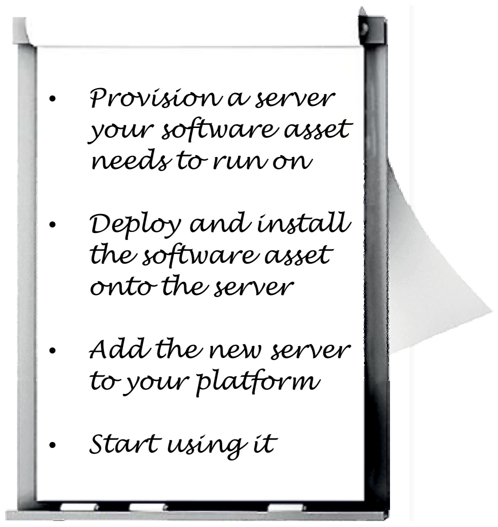
Having automated provisioning as a step within the deployment process is something that is extremely useful and powerful. It should be noted that it can also be quite complex and, at times, painful to implement—unless you know what you're doing.

As is normal within the IT industry, there are many buzzwords floating around to add to this complexity—**Infrastructure-as-a-Service** (**IaaS**) and **Platform-as-a-Service** (**PaaS**) being the most common ones. Simply put, automated provisioning solutions allow you to programmatically talk to a system; provide it with a recipe that contains things such as server spec, operating system, configuration, and so on; and it spits out a server at the other end.

In addition to IaaS and PaaS, there's another widely used buzzword—**Infrastructure-as-Code** (**IaC**). In simple terms; this is, as the name implies, code written in what could be (and pretty much is) classed as a development language that defines the recipe and the process of provisioning.

[Infrastructure-as-code is an area where the notional line between Dev and Ops becomes blurred, and most of the aforementioned engineering practices become as relevant to your average Oscars as they do for your Devinas.]

With automated provisioning, getting your software up and running can be relatively simple. You have the fully tested software asset, and you have the recipe for your environment/server configuration, so the act of deployment could be as simple as shown here:



- Provision a server your software asset needs to run on
- Deploy and install the software asset onto the server
- Add the new server to your platform
- Start using it

OK, so there is a little more to it than that, but to all intents and purposes, if you have the ability to do this, then there is no reason you should not consider it. If you wish to go down this route, it is vitally important that your CD tooling allows for it. One other benefit of automated provisioning at deployment time is that it can help a great deal in terms of no-downtime deployments.

No-downtime deployments

One of the things that come with large releases of software (legacy or otherwise) is the unforgivable need to take some or all of your production platform offline while the release happens. Yes, I did say unforgivable, because this is exactly what it is. It is also wholly avoidable.

If you are operating a real-time online service, you can bet a pretty penny that your customers will not take kindly to not being able to access your system (or more importantly, their data) for a few hours so that you can upgrade some parts of it. There is also a very strong possibility that they will look upon this with distrust as they'll be pretty sure something will go wrong once it's up and running again. It will and you will then be in *damage-limitation* mode to keep them happy. They might even shop around to find a competitor who does not have downtime.

OK, so this is a bit on the dark and negative side, but this is the reality, even more so with today's social media and viral ways of spreading bad news—some say that bad news travels faster than anything else known to man. The last thing you need is bad news generated because of a release; this will knock your confidence, tarnish your reputation, and erode any trust you had built up within the business. Release-related incidents can and will happen, so adding insult to injury is not ideal.

There are many simple things that can be done to remove the need for downtime deployments, some of which we have already covered:

- Ensure the *never break your consumer* rule is followed religiously
- Ensure your changes are small and discrete
- If possible, implement automated provisioning and integrate this as part of your CD tooling
- Implement load balancers and have the CD tooling orchestrate servers in and out of the pool during the deployment
- If you cannot avoid implementing breaking changes, then do so gradually rather than with a big bang

There are, of course, many more things that you might be aware of or can find information on elsewhere, but suffice to say that if you ever have to take your platform offline to release software, something is fundamentally wrong.

One thing to point out, which might not be obvious, is that it's not just the production environment that should have maximum uptime. Any environment that you rely on for your development, testing, and CD should be treated the same. If the like-live environment is down, how are you going to develop? If your CI environment is down, how are you going to integrate and test? The same rules should apply across the board—without exception.

To reuse a metaphor from an earlier chapter, there is an elephant in the room that we've been skirting around. This elephant has fast become a must-have big-ticket item for most software businesses and is seen as synonymous with the adoption of CD and DevOps. This elephant is the cloud.

The cloud

As mentioned earlier, delivering software using cloud solutions and technologies has been around for a while; however, the vast majority of businesses around the globe have not taken full advantage of this phenomenon as yet —especially in their production environments. The uptake of cloud solutions and technologies within the enterprise space, for example, is growing, but the numbers small and uptake are still very slow. That's not to say it can't be done; it's just a bigger leap for some well-established and old-school businesses.

One of the major advantages of using cloud-based infrastructure is the plethora of mature and proven tools, technologies, and techniques that have emerged in recent years. You can pick them up with relative ease and—being mostly open source—without heavy investment. There's also a fast-growing and vastly experienced community globally available that can help you at every stage.

Adopting cloud-based technologies can help with accelerating your CD and DevOps adoption. The aforementioned plethora of tools will allow you to fast track some areas of your adoption—for example, you would be able to let engineers spin up new servers with relative ease to simply try some experiments with a new CI module or see whether an OS upgrade has any impact on the software platform.

In a *quid pro quo* kind of way, adopting DevOps ways of working can also help accelerate the adoption of cloud-based solutions and technologies. You'll need closely-knit Dev and Ops teams that can work seamlessly and collaboratively to implement, set up, and manage your virtual IaC-based environment. They will also need to work closely together to monitor everything that is happening in their now globally distributed platform.

Before you go all guns, there are, however, a few caveats you should take into account when it comes to cloud adoption. Here are a few examples:

- A major hurdle for some is the simple fact that the skills and experience needed to effectively use these tools, technologies, and techniques are more attuned to developers than traditional system operators—more Dev than Ops. There is, therefore, something of a learning curve and possibly some skillset realignments that are required.
- Scale can also bring new problems—imagine trying to deploy a software asset to thousands of servers without a Dev and Ops team working cohesively with a shared toolset in a highly collaborative way.

- There is also the age-old issue of security. It's hard enough for your average SecOps person to ensure adherence to regulations and data security when your data lives on servers that are sitting in a locked-down server room—imagine their reaction to *we're moving it all to the cloud*. Without having open, honest, and trust-based relationships in place, you're going to struggle to help these people overcome their initial shock and feeling of dread.

I'm pretty sure you would be able to add to this list; however, you will find that the proven advantages of adopting cloud-based solutions and technologies will outweigh the caveats.

Previously, we covered open and honest ways of working as part of engineering best practices. Openness and honesty are just as important when it comes to CD. A good way of providing this level of transparency is to monitor everything and have it available to all.

Monitoring

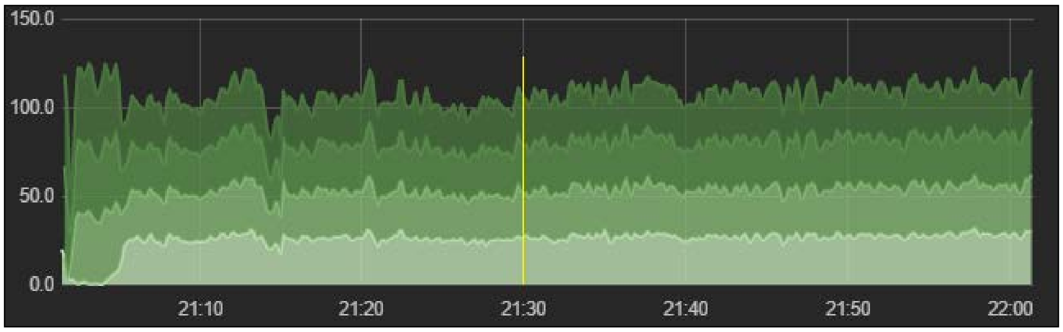
One of the most important ways to ensure whether CD and DevOps is working is to monitor, monitor, and then monitor some more. If all of the environments used within the CD process are constantly being observed, then the impact of any change (big or small) is easy to see—in other words, there should be no hidden surprises.

If you have good coverage in terms of monitoring, you have much more transparency across the board. There is no reason why monitoring should be restricted to the operations teams; everyone in the business should be able to see and understand how any environment—especially the production platform—is performing and what it is doing.

There are plenty of monitoring tools available, but it can be quite difficult to get a single view that is consistent and meaningful. For example, something like Nagios is a pretty good tool for monitoring infrastructure and servers, Graphite is pretty good at collecting application metrics, and Logstash is pretty good at collecting and analyzing server logfiles. Unless you can tie them all together into a single coherent view, things will look disjointed. Ideally, you should try and aggregate the data from these tools—or at least try and integrate them—and present a unified view of how the production platform (or any environment for that matter) is coping and functioning. You will be surprised how much extremely valuable data you can get and how it can direct your development work, as the engineers can see exactly how their software or infrastructure is behaving in real time with real users.

Monitoring is a must for CD and DevOps. As change is being made to production (software or infrastructure), both the Dev and Ops sides of the DevOps partnership can see what is going on and assist when/if problems occur.

Another less-obvious positive that monitoring can bring you is proof that CD is *not* having an adverse impact on the environment to which you are deploying. Let's assume that you are using some *graph over time* monitoring solution that is recording and graphing things such as network throughput, server load, and application performance. If you, then, somehow get your CD tool(s) to record a *spike* or a *marker* to the graph when a deployment takes place, you can then visually see when the deployment took place and what impact, if any, it had. If the other metrics show little or no change during/after the deployment, you can be pretty confident that there has been no impact. The following figure shows this (the deployment took place at 21:30):



An example graph over time showing a deployment

Up until this point, we have mainly focused on technical solutions. These solutions might help to provide you with much of what you need in your toolbox. However, there is still room for simple manual processes, which complement the technical solutions.

When a simple manual process is also an effective tool

Even if you have enough tooling to shake a stick at, you will no doubt have some small and niggling challenges that cannot be overcome with tooling and automation alone. To be honest, tooling and automation can be overkill in some respects and can actually create barriers between certain parts of the organization you are trying so hard to bring together—here, I am talking about the Dev and Ops partnership that forms DevOps.

If tooling and automation completely negate the need for human interaction and discussion, you might well end up back where you started. You might also find that it is almost impossible to automate your way out of a simple problem.

Let's take, for example, the thorny issue of dependency management. As a software platform matures, many interdependencies will form. If you are deploying your code using a CD process, these many interdependencies become ever-moving targets where components are being developed and deployed at different rates. You can try to capture this within your CI process, but something somewhere might be missed, and you could end up inadvertently bringing down the entire platform because component B was deployed before component A.

You can try to map this out and build into the tooling rules to restrict or at least minimize these moving targets, but the rules might end up more complex than the original dependencies themselves. Alternatively, you could simply agree on a process whereby only one change happens at any given point in time. To feed into this, you can implement a simple queuing mechanism written on a whiteboard and reviewed regularly by all of the engineering and operations teams.

This approach worked extremely well for ACME systems. The following is what they did:

- They obtained blanket agreement from everyone that only one change would go through to production at any given point in time. They called this a *deployment transaction*.
- None of the CD tooling was changed to have this restriction built in; they simply relied on common sense and collaborative ways of working.
- To highlight the fact that someone was making a change to production (either a deployment or operational change), that person held the *production environment token*, which was in the form of a plush toy animal and was given the name *the deployment badger*. If you had the deployment badger, you were changing production.
- They implemented a simple prioritized queue system using a whiteboard. Each morning, whoever wanted to make a deployment would come along to the deployment stand-up where everyone agreed the order in which deployments (or changes) would be made that day.
- Screens were installed throughout the office (not just the Dev and Ops areas), showing a real-time dashboard of what was going on.

All very simple, but what this gave ACME systems was a way to overcome dependency hell (for example, if they could only change one thing at a time, there was an implied logical order of which change went before another) and built a sense of collaboration throughout all the teams involved.

Other very simple manual solutions you can use could include the following:

- Use collaborative tools for real-time communication between everyone (that is, **internet relay chat** (**IRC**), chat rooms, or similar) and integrate this into your CD tooling so that deployments are announced and can be followed by all
- If your management is uneasy about having developers deploy to production without involving the operations team, set up a workstation within the operations area, call it the *deployment station*, and make sure that's the only workstation from where live deployments can be run from
- If instant rollback is needed should a deployment fail, consider simple ways of rolling back, such as deploying the previous version of the component using the CD tooling
- Consistently inspect and adapt through regular retrospectives to see what is working and what is not

As you can tell, it's not all about technical solutions. If simple manual processes or tweaks to the ways of working are sufficient, then why bother trying to automate them?

And so ends the lesson—for now. Let's recap what we have covered throughout this chapter.

Summary

As stated at the beginning of this chapter, there is a lot to cover and take in. Some of it is relevant to you now, and some of it will be relevant for the future. All things considered, you need to ensure that you have the technical building blocks in place. You need to ensure that you are using engineering best practice, have the necessary tools and solutions in place, work in small *chunks* of change rather than big *clumps*, consider how many environments you actually need, consider simple manual processes over technology where they fit best, and monitor, monitor, and then monitor some more.

As you can see, there is quite a bit that needs to be done. It's not all technical either; simply convincing people to adopt and use the tools and processes you implement is no small task. In *Chapter 6*, *Hurdles Along the Way*, we'll look at some of the other hurdles and challenges you'll face.

6 Hurdles Along the Way

Up until now, we have been focusing on the core tools and techniques you'll need in your toolbox to successfully implement and adopt CD and DevOps. Along the way, we looked at a few of the hurdles you'll have to get over. We'll now look at some of these potential hurdles in more detail and the ways to overcome them or at least minimize the impact of them so that you can drive forward with your goal and vision.

What follows is by no means an exhaustive list; however, there is a high probability that you'll encounter at least one or two of these problems along the way. The main thing that you need to do is be aware that there will be the occasional storm throughout your journey. You need to understand how you can steer your way around or through it and ensure that it doesn't ground you or completely run the implementation onto the rocks—to use a nautical analogy for some reason.

What are the potential issues you need to look out for?

Depending on your culture, environment, ways of working, and business maturity, there might be more potential hurdles than you can shake a stick at. Hopefully, this will not be the case, and you will have a nice, smooth implementation, but just in case, let's go through some of the more obvious potential hurdles. What follows are some example hurdles you may encounter:

- Individuals who just don't see why things have to change and/or simply don't want to change how things are
- Individuals who want things to go quicker and are impatient for change
- The way people react to change at an emotional level

- A lack of external understanding or visibility of what you are trying to achieve might throw a spanner in the works when business priorities change
- Red tape and heavyweight corporate processes
- Geographically diverse teams
- Unforeseen issues with the tooling chosen for the tool kit (technical and nontechnical)
- Recruitment

The list could be much longer, but there's only so much space in this book. Let's therefore focus on more obvious potential issues that could if left unchecked run the implementation of CD and DevOps into shallow waters or, worse still, aground. We'll start by focusing on individuals and how they can have an impact, both negative and positive, on your vision and goal.

Dissenters in the ranks

Although the word **dissenters** is a rather powerful one to use, it is quite representative of what can happen should individuals decide what you are doing doesn't fit with their view of the world.

As with anything new, some people will be uncomfortable, and how they react depends on many things, but there's a strong chance that, you will have some individuals who decide that they are against what you are doing. The whys and wherefores can be examined and analyzed to the *n*th degree, but there is something very important for you to realize: if one or two individuals are loud enough, they can redirect your attention from your vision and goal. This is exactly what you don't want to happen.

This is nothing new. If you look back at the early days of *agile* adoption, there are plenty of examples of this phenomenon. The individuals involved in the adoption of *agile* within an organization broadly fall into three types:

- A small number of **innovators** trailblazing the way
- A larger number of **followers** who are either interested in this new way of doing things or can see the benefits and have decided to move in the direction that the innovators are going
- Finally, the **laggards** who are undecided or not convinced that it's the right direction to go.

The general consensus is that effort and attention should be focused on the innovators and followers as this makes up the majority of the individuals involved. The followers who are moving up the curve towards the innovators need some help to get over the crest and over the other side, so more attention is given to them. To focus on the laggards would take too much attention away from the majority, so the painful truth is that they either shape up or ship out—even if they're senior managers. This might seem rather brutal, but this approach has worked for a good number of years, so there must be something in it.

So, let's consider our dissenters or laggards in terms of our implementation; what should you do? As previously pointed out, if they are loud enough, they can make enough noise to disrupt things, but not for long. If the majority of the organization has bought into what you are doing—don't forget that you are executing a plan based on their input and suggestions—they will not easily become distracted; therefore, you should not become distracted. If you have managed to build up a good network across the business, use this network to reduce the noise and, if possible, convert the laggards into followers.

If these laggards are in managerial positions, this might make things more difficult—especially if they are good at playing the political games that go on in any business—however, they will be fighting a losing battle as the majority will be behind you (because you are delivering something they have asked for). You just need to be diligent and stick to what you need to do. You will have your eyes peeled and your ear to the ground, so you should be able to tell when trouble is brewing, and you can divert a small amount of effort to addressing this and stop it from becoming a major issue. The *addressing this* part can be in the form of a simple non-confrontational face-to-face discussion with the potential troublemaker over a coffee—this way, they are being listened to, and you have an idea of what the noise is all about. As a last resort, a face-to-face discussion with their boss might do the trick. Don't resort to e-mail tennis!

All in all, you should try wherever possible to deal with dissenters as you would the naughty child in the classroom; don't let them spoil things for everyone; don't give them all of the attention; and use a calm, measured approach. After a while, people will stop listening to them or get bored with what they are saying (especially if it's not very constructive).

Let's look a couple of examples that could potentially fuel the voice of the dissenters.

No news is no news

Something that could increase the risk of dissenters spoiling the party is a lack of visible progress in terms of CD and DevOps implementation. It might be that you're busy with a complex process change or developing tooling, and there is a lull in visible activity. If you have individuals within your organization who are very driven and delivery focused, they might take this lull as a sign of the implementation faltering, or they may even think that you're finished.

As we mentioned previously, being highly visible, even if there's not a vast amount going on, is very important. If people can see that progress is being made, they will continue to follow. If there is a period of perceived inaction, the followers might not know which way you are heading and might start taking notice of the dissenting voices. Any form of communication and/or progress update can help stop this from happening—even if there's not a vast amount to report, the act of communication indicates that you are still there and still progressing towards the goal.

The anti-agile brigade

In *Chapter 5*, *Approaches, Tools, and Techniques*, we looked at engineering best practice, which, if truth be told, is simply based on modern *agile* engineering techniques. You might find that some of the dissenting voices belong to the old-school engineers who have an aversion to anything *agile*. Maybe they have been working in the same mode for the past 20 years and believe that they are exempt from this *new fad*. Maybe they simply don't understand it or are afraid of working in such a way. Whatever the reasons, you need to be mindful of the fact that they will cause some noise. If they are relatively senior and/or well respected, you should tread carefully as you don't want to end up with a power struggle.

Focus your time and energy on the followers and innovators and see if you can get some of them to work with the old-school laggards and ease them in gently.

You should also consider working with their managers and looking into specific training or incentives to help ease their move from the dark side—without resorting to or being seen to resort to bribery. What you need is more innovators and followers than laggards.

If your entire engineering team's ways of working are based on old-school *waterfall* delivery practices, your hurdles are going to be much larger. In reality, CD and DevOps do not play nicely with traditional *waterfall* software delivery; therefore, you are going to have to address this problem with some urgency. You should consider investing time, effort, and money in training, sending people to relevant conferences or local meet-ups—again, focus on those individuals who have the potential to become your innovators, as they can help drive forward the adoption of *agile* engineering practices.

We briefly covered the fact that some people will be uncomfortable with change, and they might react in unexpected ways. We'll now look at how change can impact individuals in different ways and what you need to be aware of.

The transition curve

Let's get one thing out in the open—and this is important—you need to recognize and accept that the identification of a problem and subsequent removal of it can be quite a big change. You have been working with the business to identify a problem, and you are now working to remove it. This is change, pure and simple.

Earlier in the book, we stated that the brave men and women of ACME systems who helped implement the DevOps and CD ways of working were a catalyst for change. This wording was intentional as change did come about for the ACME systems team—a very big change as it turned out. The implementation of CD and DevOps and its impact on individuals should not be taken lightly, even if they originally thought it was the best thing since sliced bread.

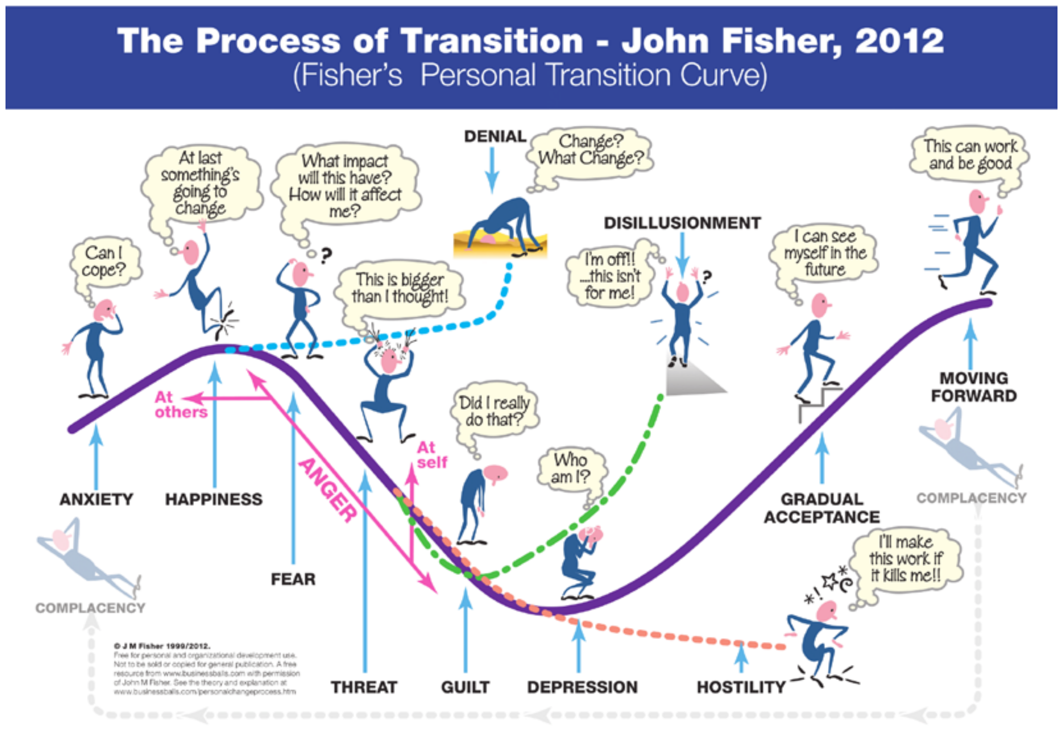
Those of you who have been in, or are currently in, management or leadership roles should understand that change can be seen as both positive and negative, and sometimes, it can be taken very personally—especially where a change to a business impacts individuals and their current roles within it. Let's look at some fundamentals in relation to how humans deal with change.

Any change, large or small, work related or not, can impact people in different ways. Some people welcome change, some are not fazed by it and accept it, some are downright hostile and see a change as something personal. More importantly, some people are all three. If we are mindful of these facts before we implement change, we will have a clearer idea of what challenges to overcome during the implementation to ensure that it is successful.

There has been much research on this subject, and many papers have been published by learned men and women over the years. I don't suggest for one minute that I know all there is to know on this subject, but there is some degree of common sense required when it comes to change—or transition as it is sometimes called—and there are some very simple and understandable traits to take into account.

One of my preferred ways to visualize and understand the impact of change is something called the change or transition curve. This depicts the stages an individual will go through as change/transition is being implemented.

The following diagram is a good example of a change/transition curve:


The Process of Transition - John Fisher, 2012
(Fisher's Personal Transition Curve)
Can I cope?
At last something's going to change
What impact will this have? How will it affect me?
DENIAL
Change? What Change?
This is bigger than I thought!
DISILLUSIONMENT
I'm off!! ...this isn't for me!
I can see myself in the future
This can work and be good
At others
ANGER
At self
Did I really do that?
Who am I?
MOVING FORWARD
ANXIETY
HAPPINESS
FEAR
THREAT
GUILT
DEPRESSION
HOSTILITY
GRADUAL ACCEPTANCE
I'll make this work if it kills me!!
COMPLACENCY
COMPLACENCY
© J M Fisher 1999/2012.
Free for personal and organizational development use. Not to be sold or copied for general publication. A free resource from www.businessballs.com with permission of John M Fisher. See the theory and explanation at www.businessballs.com/personalchangeprocess.htm

John Fisher's personal transition curve—the stages of personal change

John Fisher's personal transition curve diagram can be found at `http://www.businessballs.com/freepdfmaterials/fisher-transition-curve-2012bb.pdf`.

You can clearly see that as change is being planned, discussed, or implemented, people will go through several stages. We will not go through each stage in detail (you can read through this at your leisure at `http://www.businessballs.com/personalchangeprocess.htm`); however, there are a few nuggets of information that are very pertinent when looking at implementing CD and DevOps. They are as follows:

- People might go through this curve many times—even at the very early stages of change
- Everyone is different, and the speed at which they go through the curve is unique to the individual
- You and those enlightened few around you will go through this curve
- Those who do not / cannot come out of the dip might need more help, guidance, and leadership
- Even if someone is quiet and doesn't seem fazed, they will inevitably be sat at some stage in the curve—it's not just the vocal ones to look out for

The long and short of it is that individuals are just that; they will be laggards, followers, or innovators, and they will also be somewhere along the change curve. The leaders and managers within your organization need to be very mindful of this and ensure that people are being looked after. You also need to be mindful of this, not least because this will also apply to you, as it might give some indication as to why certain individuals act in one way at the beginning, yet they change their approach as you go through the execution of the plan and vision.

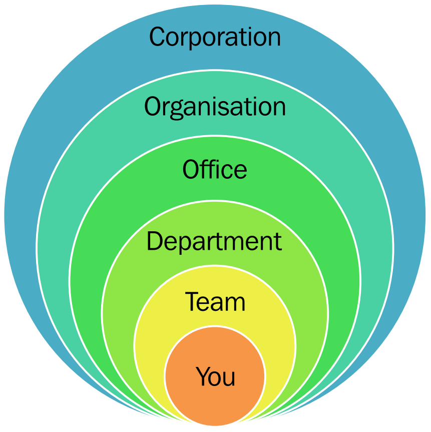
At a personal and emotional level, change is good and bad, exiting and scary, challenging and daunting, welcomed and avoided. It all depends on how you, as an individual, feel at any given point in time. CD and DevOps is potentially a very big change; therefore, emotions will play a large part. If you are aware of this and ensure that you look for the signs and react accordingly, you will have a much better time of it. Ignore this, and you will have one hell of a battle on your hands.

On this light note, we'll move on to the subject of what to do about those people within your organization who are not involved in your journey or might not even be aware that it is ongoing. We'll call them *the outsiders*.

The outsiders

The percentage of those involved with the implementation of CD and DevOps will largely depend on the overall size of your organization. If you are a startup, the chances are that everyone within the organization will be involved. If you are a **small to medium enterprise** (**SME**), there is a good chance that not everyone within your organization will be involved. If you are working within a corporate business, the percentage of those actively involved will be smaller than those who are not.

The following diagram illustrates how a typical corporation is made up and where you and your team sit within it:



The further out from the inner circle, the greater the possibility that there is ignorance about what you are doing and why

Those sitting outside of the circle of active involvement will have little/no idea of what is going on and could—through this lack of knowledge—put hurdles in the way of your progress. This is nothing new and does not specifically apply to the implementation of CD and DevOps; this is a reality for any specialized project. Let's take a look at ACME systems and see how this situation impacted their implementation.

During the version 2.0 stage of their evolution, ACME systems became part of a large corporate. They ended up as a satellite office—the corporate HQ being overseas—and on the whole, were left to their own devices. They beavered away for a while and started to examine and implement CD and DevOps. They were doing so, when viewed at a global corporate level, in isolation. Yes, they were making far-reaching and dramatic changes to the ACME systems organization, but they were a small cog in a very big wheel. No one outside of the ACME systems offices had much visibility or in-depth knowledge of what was going on.

As a consequence, when a new far-reaching corporate strategic plan was announced, little or no consideration was given to what ACME systems were up to; no one making the decisions really knew. As a result, the progress of the CD and DevOps implementation was impacted.

In the case of ACME systems, the impact turned out to be positive in respect to the CD and DevOps implementation and actually provided an additional boost—you might not be so lucky. If you experience wide-reaching changes during your journey and people are ignorant of what you're doing, your story might not end so well. Bear this in mind.

The moral of the story is this: not only should you keep an eye on what is happening close to home, but you should also keep an eye on what is happening in the wider organization. We've already looked at how important it is to communicate what you are doing and to be highly visible. This communication and visibility should not be restricted to those immediately involved in the CD and DevOps implementation; you should try to make as many people aware as possible. If you are working within a corporate environment, you will, no doubt, have some sort of internal communications team who publish regular news articles to your corporate intranet or newsletter. Get in touch with these people and get them to run a story on what you are doing. A good bit of PR will help your cause and widen the circle of knowledge.

This might seem like quite a lot of work for little gain, but you might be surprised how much benefit it can bring. For example, let's imaging that you write an article on CD and DevOps, get it published, and it is read by your CEO or SVP who then decides to visit and see what all the fuss is about. This is a major moral boost and good PR. Not only that, but it can help with your management dissenters—if they see the high-ups recognizing what you are doing as a positive thing, they might (will) reconsider their position.

We're primarily considering outsiders as individuals outside of your immediate sphere of influence that are ignorant of what you are doing and where you're heading. You might have others who are well aware, but are either restricted by or hiding behind corporate red tape. Let's spend some time looking at this potential hurdle and what can be done to overcome it.

Corporate guidelines, red tape, and standards

The size and scale of this potential hurdle is dependent on the size and scale of your organization and the market in which you operate. For example, you might work within the service sector and have commercial obligations to meet certain **service-level agreements** (**SLAs**), or you might work within a financial institution and have regulatory and legal guidelines to adhere to. Whatever the industry, there's a strong bet that you will be hampered in some way by things and people outside of your control. This, as they say, comes with the territory.

To overcome this hurdle or, at the very least, minimize the impact, you need to work closely with those setting and/or policing the rules to see what wriggle room you have. Build rapport, entice them with coffee and doughnuts, get to know them, and understand what drives them and what constraints they have to work with. Eventually, you'll start to build a picture of what constraints you have and which of these could have the biggest impact on your plan and vision. You might also find that things aren't as black and white as first thought. For example, you might find that some of the rules and guidelines set in place are overkill and have only been implemented in their current form, because it was easier, quicker, or safer to stick to what it said in a book than it was to refine to fit the business needs.

Fundamentally, the need for such rules, guidelines, and policies revolves around change management and auditability. In simple terms, they offer a safety gate and a way to ascertain what has been changed by whom, should problems occur.

One hurdle that isn't immediately obvious is that those managing or policing the rules, guidelines, and policies might well consider CD and DevOps to be incompatible with their ways of working. If you think about it, they'll be imagining software flying through the door at a rate of knots. This might be a scary vision for them.

They might also consider CD and DevOps as a threat and be overly defensive. This might be due to the fact that during the *inspect* stage (see *Chapter 2*, *No Pain, No Gain*), their organization/department had been highlighted as an area of waste (I would put money on it), and as such, they might not be totally willing to embrace the change you are bringing. It might even be the case that they simply don't know what they can change without breaking a rule or corporate policy. Use your new-found rapport and work with these people and help them understand what CD and DevOps is all about and help them research what parts of their process they can change to accommodate. Do not simply ignore them and break the rules, as this will catch up with you later down the road and could completely derail you. Open, honest, and courageous dialogue is the key (see *Chapter 4*, *Culture and Behaviors*).

That said, open and honest dialogue might be hindered by geography, so let's look at how we can address this.

Geographically diverse teams

We previously touched upon the subject of setting up an open and honest physical environment to help enforce open, honest, and collaborative ways of work. This is all well and good if the teams are collocated; however, trying to recreate this with geographically diverse teams can be a tricky problem to solve.

It all depends on the time-zone differences and, to some extent, cultural differences. Not having a physical presence is always a barrier—unless you have perfected the art of matter teleportation; however, there are a few things that you could look at to overcome this:

- Ensure local team(s) have regular (ideally daily) teleconference calls with the remote team(s)—even if it's nothing more than to say good morning.
- If you're using *scrum* (or a similar methodology) and decide to have a daily *scrum* of *scrums*, get the remote teams(s) dialed in as well—even if you call them on your cell phone and have them on speakerphone.
- Time zones can play havoc with daily stand-ups (from experience, these normally happen first thing in the morning), so try and be creative around the schedule (don't forget to accommodate changes to clocks throughout the year).
- We all have access to some form of video conferencing—be that using a corporate teleconferencing system or something as simple as Skype (or similar). You could set this up within your office space (rather than hidden away in some meeting room) and use it like an *always-on* virtual wall / window that allows team members to simply walk up and have a face-to-face conversation, as if they were in the same room/city/country.
- If your budget allows, try and get people physically swapped across the offices either via secondments or short-term project placements.
- Don't rely on e-mails for discussions; instead, invest in real-time collaborative tools and encourage their use.

Cultural differences were previously mentioned as a potential problem. This should not be taken lightly. The reality is that in some parts of the world, the culture is far and removed from the fast and loose western culture where everyone has a voice and isn't afraid to use it. Instilling openness, honesty, and transparency might be more difficult for some, and you should be mindful of this. Work with the local HR or management, explain what you're trying to do, and see what they can do to help with this.

We'll now look at what you should do if you encounter failure during the execution of your goal and vision.

Failure during evolution

As you go along your journey, things will occasionally go wrong—this is inevitable and is nothing to be afraid or ashamed of. There might be situations that you didn't foresee, or there might be hidden steps in an existing process that were not surfaced during the *inspect* stage (see *Chapter 2, No Pain, No Gain*). It might even be as simple as a problem within the chosen toolset, which isn't doing what you had hoped it would or is simply buggy.

Your natural reaction might be to hide such failures or at least not broadcast the fact that a failure has occurred. This is not a wise thing to do. You are working hard to instill a sense of openness and honesty, so the worst thing you can do is the exact opposite.

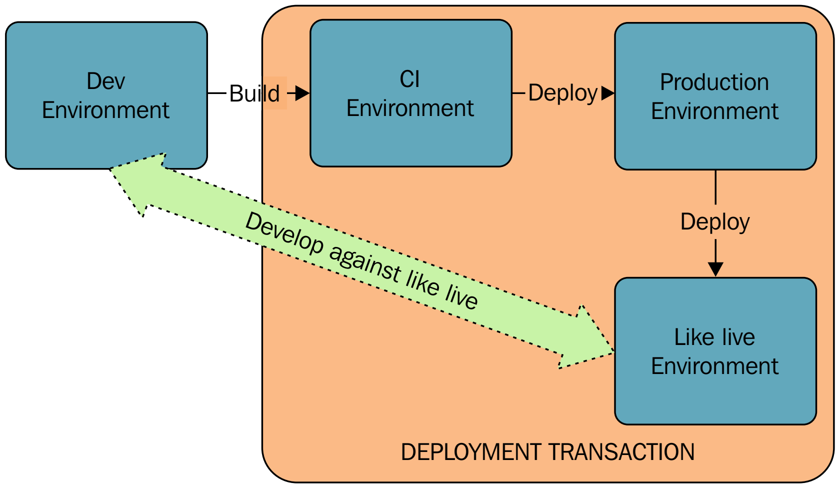
Admitting defeat, curling up in a fetal position, and lying in the corner whimpering is also not an option. As with any change, things go wrong, so review the situation, review your options, and move forward. Once you have a way to move around or even through the problem, communicate this. Ensure you're candid about what the problem is and what is being done to overcome it. This will show others how to react and deal with change—leading by example if you will.

You might be concerned that admitting failures might give the laggards more ammunition to derail the adoption; however, their *win* will be short lived once the innovators and followers have found a solution. Hold fast, stand your ground, and have faith.

Okay, so this is all very happy-clappy **positive mental attitude** (**PMA**) and might be seen by some of a cynical nature as management hot air and platitudes; however, this approach does and will work. Let's look at another real-world example.

As a simple way to manage dependencies and ensure that only one change went through to the production system at any one point in time, ACME systems implemented something they called a *deployment transaction* (see *Chapter 5, Approaches, Tools, and Techniques*). This worked well for a while, but things started to slow down and impact their ability to deliver. After some investigation and much debate, it turned out that the main source of the problem was that there was no sure way of determining which change would be completed before another, and there was no way to try out different scenarios in terms of integration. Simply put, if changes within asset A had some dependency on changes within asset B, then asset B needed to go live first to allow for full integration testing. However, if asset A was ready first, it would have to sit and wait—sometimes for days or weeks. The newly implemented deployment transaction itself was starting to hinder CD.

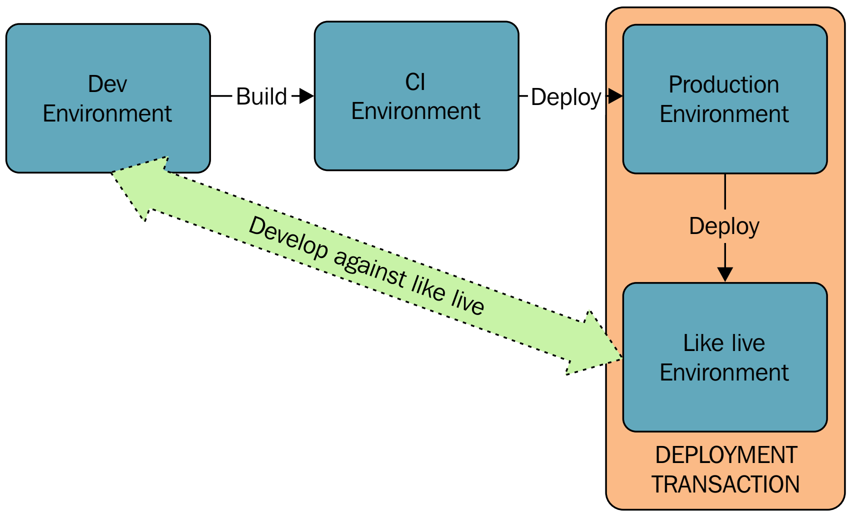
The following diagram details the deployment transaction as originally implemented:



The deployment transaction version 1.0

Everyone had agreed that the *deployment transaction* worked well and provided a working alternative to dependency hell. When used in anger, however, it started to cause real and painful problems. Even if features could be switched off through feature flags, there was no way to fully test integration without having everything deployed to production and the *like live* environment. This had not been a problem previously, as the speed of releases had been very slow and assets had been clumped together. ACME systems now had the ability to deploy to production very quickly and now had a new problem: which order to deploy? Many discussions took place and complicated tooling options were looked at, but in the end, the solution was quite simple: move the boundary of the deployment transaction and allow for full integration testing before the assets went to production. It was then down to the various engineering teams to collaborate and agree in which order things should be deployed.

The following diagram depicts the revised deployment-transaction boundary:



The deployment transaction version 2.0

So, ACME had a potential showstopper that could have completely derailed their CD and DevOps implementation. The problem became very visible, and many questions were asked. The followers started to doubt the innovators, and the laggards became extremely vocal. With some good old-fashioned collaboration and open and honest discussions, the issue was quickly and relatively easily overcome.

Again, open and honest communication and courageous dialogue is the key. If you keep reviewing and listening to what people are saying, you have a much better opportunity to see potential hurdles before they completely block your progress.

Another thing that might scupper your implementation and erode trust is inconsistent results.

Processes that are not repeatable

There is a tendency for those of a technical nature to automate everything they touch; automating the build of engineer's workstations, automated building of software, automated switching on of the coffee machine when the office lights come on, and so on. This is nothing new, and there is nothing wrong with this approach as long as the process is repeatable and provides consistent results each time. If the results are inconsistent, others will be reluctant to use the automation you spent many hours, days, or weeks pulling together.

When it comes to CD and DevOps, the same approach should apply—especially when you're looking at tooling. You need to trust the results that you are getting time and time again.

Some believe that internal tooling and labor-saving solutions or processes that aren't out in the hostile customer world don't have to be of production quality, as they're only going to be used by people within the business. This is 100 percent wrong. Internal users are as important as external ones.

Let's look at a very simple example; if you're a software engineer, you will use an **integrated development environment** (**IDE**) to write code, and you will use a compiler to generate the binary to deploy. If you're a **database administrator** (**DBA**), you'll use a SQL admin program to manage your databases and write SQL scripts. You will expect these tools to work 100 percent of the time and produce consistent and repeatable results; you open a source file, and the IDE opens it for editing; and you execute some SQL, and the SQL admin tool runs it on the server. If your tools keep crashing or produces unexpected results, you will be a tad upset (putting it politely) and will no doubt refrain from using said tools again. This might drive you insane.

> "*Insanity: doing the same thing over and over again and expecting different results.*"
>
> – *Albert Einstein*

The same goes for the tools (technical and nontechnical) you build and/or implement for your CD and DevOps adoption. You need to be confident that when you perform the same actions over and over again, you will get the same results. As your confidence grows, so does your trust in the tool/process, and you then start taking it for granted and use it without a second thought. Consequently, you will also trust the fact that if the results are different, then something has gone wrong and needs addressing.

We have already covered the potential hurdles you'll encounter in terms of corporate guidelines, red tape, and standards. Just think what fun you will have convincing the gate keepers that CD and DevOps is not risky when you can't provide consistent results for repeatable tasks. Okay, maybe fun is not the correct word; maybe pain is a better one.

Another advantage of consistent, repeatable results comes into play when looking at metrics. If you can trust the fact that to deploy the same asset to the same server takes the same amount of time each time you deploy it, you can start to spot problems (for example, if it starts taking longer to deploy, then there might be an infrastructure issue, or you might have introduced a bug within the latest version of the CD tools).

All in all, it might sound boring and not very innovative, but with consistent and repeatable results, you can stop worrying about the mundane and move your attention to the problems that need solving, such as the very real requirement to recruit new people into a transforming or transformed business.

Recruitment

This might not, on the face of it, seem like a big problem, but as the organization's output increases, the efficiency grows, and the business starts to be recognized as one that can deliver quality products quickly (and it will), then growth and expansion might well become a high priority—which is a great problem to have. You might also have lost a few laggards along the way who have decided they don't like this new and improved way of working and have moved on.

You, therefore, need to find individuals who will work in your *new way*, believe in your approach, exhibit the behaviors you have worked so hard to instill and embed throughout the organization, and, possibly, bring new skills and experience to the team. This is not an easy task, and it will take some time. Simply adding *experience in CD and DevOps* to a job spec will not produce the results you want. Although CD and DevOps are becoming prevalent approaches, they are still relatively new, and there's not that many people out there with the specific skills or experience you need. There is a growing market of recruiters who specialize in finding "DevOps engineers", but if truth be told, hardly any of them actually know what this really means—apart from adding 20 percent to the recruitment fee and salary expectations, of course.

You will need to embark on more knowledge sharing with those involved in your recruitment process to ensure that they fully understand what and who you're looking for. You might need to do this number of times until this sinks in, so be forewarned.

This might also be a good time to reach out to the CD and DevOps community and let them know you're hiring. You never know how a brief chat during the coffee break of your local DevOpsDays conference (`http://devopsdays.org/`) with a renowned engineer might pan out.

Getting potential candidates is going to be challenging. The next challenge is going to be how you filter out the good from the not so good once you have them sitting face to face in front of you. Hiring experienced and skilled engineers can be hard work normally, but when it comes to CD and DevOps, *how* they do things becomes as important (if not more important) than *what* they can do.

You'll need people who not only have proven technical experience, but also don't see the barrier between development and operations; who understand the importance of collaboration, accountability and trust; and who understand what you're actually talking about when you discuss CD and DevOps. Watch out for those who have simply read a book on the subject just before the interview—unless it's this one, of course.

You'll need to structure the interview in such a way as to tease out these intangible qualities. One example and a very simple interview question that I find works well is:

As a software engineer, how do you feel if your code was running in the production environment being used by millions of customers 30 minutes after you commit it to source control?

The question is worded specifically to get an honest emotional response, the key word here being *feel*. You will be surprised by the responses to this; for some, it simply stops them in their tracks, some will be shocked at such a thing and think you're mad to suggest it, and some will think it through and realize that although they have never considered it, they quite like the idea. If, however, the response is, *30 minutes? That's far too slow*, you might be onto a winner.

Take your time and ensure that you pick the right people. You need potential innovators and followers more than you need laggards.

Summary

As you go through the journey of implementing and adopting CD and DevOps, you will hit some bumps in the road. If you take this on board from the outset and recognize these bumps as things that are surmountable, you will be able to deal with them and continue to drive forward. As we have covered throughout this chapter, some hurdles are obvious and others not so. What they normally have in common is people or individuals who are much more difficult to analyze and debug than software or tools. It might be frustrating, but if you take your time and approach each hurdle carefully, you will reap the rewards and ultimately be successful.

Talking of success, let's now move on to the measurement of success and why it is (also) so important.

7
Vital Measurements

Over the previous chapters, we have looked at what tools and techniques you will need to successfully adopt CD and DevOps, and we highlighted some potential hurdles to overcome. With this information in hand, you should be in a good shape to succeed.

We'll now look at the important but sometimes overlooked—or simply dismissed—area of monitoring and measuring. This, on the face of it, might be seen as something that is only useful to the management types and won't add value to your CD and DevOps implementation and adoption; however, being able to understand and demonstrate progress will definitely add value to you and everyone else who is on the CD and DevOps journey. We're not just talking about simple project-management graphs and PowerPoint fodder; what we are looking at is measuring as many aspects of the overall process as possible. This way, anyone can plainly see and understand how far you have come and how far there is left to go. To be able to do this effectively, you'll need to ensure that you address this early on, as it will be very difficult to see a comparison between *then* and *now* if you don't have data representing *then*. You'll also need to ensure that you are continuously capturing these measurements so that you can compare the state of progress at different points in time.

In this chapter, you will learn:

- How to measure the effectiveness of your engineering process(es)
- How to measure the stability of the various environments you use and rely on
- How to measure the impact your adoption of CD and DevOps is having

We'll start, as they say, at the beginning and focus initially on engineering metrics.

Measuring effective engineering best practice

This is quite a weird concept to get your head around; how can you measure effective engineering, and more than that, how can you measure best practice? It's not as strange or uncommon as you would think. There are a great number of software-based businesses around the globe using tools to capture data and measurements for things such as:

- Overall code quality
- Adherence to coding rules and standards
- Code versus comments
- Code complexity
- Code duplication
- Redundant code
- Unit test coverage
- Commit rates
- Mean time between failures
- Mean time to resolution
- Bug escape distance
- Fix bounce rate

Measuring each of these in isolation might not bring a vast amount of value; however, when pooled together, you can get a very detailed picture of how things stand. In addition, if you can continuously capture this level of detail over a period of time, you can then start to measure and report on progress. You could, for example, see whether there is an impact on code quality and complexity if you reduce code duplication or redundancy.

It all sounds very simple, and to be honest, it can be, but you need to be mindful of the fact that you will need to apply some time, effort, and rigor to ensure that you gain the most value. There will also be a degree of trial and error and tweaking as you go—more inspecting and adapting—so you need to ensure that you factor this in. Not only will these sort of measurements help your engineering team(s), but they will also help with building trust across the wider business. For example, you'll be able to provide open, honest, and truthful metrics in relation to the quality of your software, which, in turn, will reinforce the trust they have in the team(s) building and looking after the platform.

One thing to seriously consider before you look at measuring things such as software code metrics is how the engineers themselves will feel about this. What Devina is thinking might be a typical reaction:

Why are you spying on me?
Isn't this all a bit big brother?
I'm not sure I like this.

A typical reaction to this approach

Some engineers will become guarded or defensive, and see it as questioning their skill and craftsmanship in relation to creating quality code. You need to be careful that you don't get barriers put up between you and the engineering teams. You should *sell* these tools as a positive benefit for the engineers. For example, they have a way to prove how good their code actually is; they can use the tools to inspect areas of over complexity or areas of code that are more at risk of containing bugs; they can highlight redundant code and remove it from the codebase; they can visually see hard dependencies, which can help when looking at componentization; and so on.

If you have vocal dissenters, then get them actively involved in the setting up and configuration of the tools (for example, they could set the threshold at which code versus comments is set or what level of code coverage is acceptable).

If nothing more, you need to ensure that you have the innovators and followers from the engineering community brought in. To add some clarity, let's look at a few items from the preceding list—which, by the way, is not exhaustive—in a little more detail and examine why they are potentially important to your CD and DevOps adoption. Let's start with quality.

Simple quality metrics

There are a few items on the preceding list that are quite simple yet very powerful in terms of quality. The ones that are pertinent to CD and DevOps are **Mean time between failures** (**MTBF**), **Mean time to resolution** (**MTTR**), and bug escape distance, which are explained as follows:

- **MTBF**: This will help you measure how often problems (or failures) are found by end users—the longer the time between failures, the greater the stability and quality of the overall platform
- **MTTR**: This will help you measure the time taken between an issue being found and being fixed
- **Bug escape distance**: This will help you measure when an issue is found

I won't go into much more detail here, but it suffices to say that measuring these types of data will give you a good indication of progress. For example, you would expect MTBF to go up and MTTR to go down over time if CD and DevOps are working well for you. If they don't, then there's something you need to look into.

Let's now look at some other items from the list.

Code complexity

Having complex code is sometimes necessary, especially when you're looking at extremely optimized code where resources are limited and/or there is a real-time UI—basically, where every millisecond counts. When you have something like an online store, login page, or a finance module, having overly complex code can do more harm than good. Some engineers believe they are *special* because they can write complex code; however, complexity for complexity's sake is really just showing off.

Overly complex code can cause lots of general problems—especially when trying to debug or when you're trying to extend it to cater for additional use cases—which can directly impact the speed at which you can implement even the smallest change. The premise of CD is to deliver small incremental changes. If your code is too complex to allow for this, you are going to have issues down the line.

I would recommend that you put some time aside to look into this complex (pun intended) subject in more detail before you dive into implementing any process or tooling. You really need to understand what the underlying principles are and the science behind them; otherwise, this will become messy and confused. Some of the science is explained in *Appendix D*, *Vital Measurements Expanded*.

The next thing you could consider is code coverage.

Unit test coverage

Incorporating **unit tests** within the software-development process is a recognized best practice. There is lots of information available on this subject; however, simply put, it allows you to exercise code paths and logic at a much more granular and lower level during the early stages of development; this, in turn, can help spot and eradicate bugs very early on. If a small chunk of code has a good coverage, then you will be more confident that you can ship that code frequently with minimal risk—the CD approach of little and often. As you become more reliant on unit tests to spot problems, it's always a good idea to get some indication of how widespread the use is—hence the need to analyze coverage. From this, you can start to map out the areas of risk when it comes to shipping code quickly (for example, if your login page is frequently changed and has a high level of coverage, the risk of shipping this frequently becomes less).

It should be pointed out that the metric being measured is broadly based on the percentage of the codebase that is covered by tests; therefore, if the business gets hung up on this value, they might consider a low value to equate to a major risk. This isn't necessarily so, especially when you're just starting out implementing unit tests against an existing codebase that had no coverage. You should set the context and ensure that everyone looking at the data understands what it means. Maybe, you can explain to them that a handful of tests (or even one) is much better and less of a risk than none.

Let's now look at the effectiveness of measuring the frequency of commits.

Commit rates

Regular commits to source control is something that should be widely encouraged and should be deeply embedded within your ways of working. Having source code sat on people's workstations or laptops for prolonged periods of time is very risky and can sometimes lead to duplication of effort or, worse still, might block the progress of other engineers.

There might be a fear that if engineers commit too frequently, the chances of bugs being created increases, especially when you think there's an outside risk that unfinished code could be incorporated into the main code branch. This fear is a fallacy, as no engineer would seriously consider doing such a thing—why would they? The real risk is due to the fact that the longer the period of time between commits, the more code there is to be merged; this, in turn, can cause greater problems, delays, and potential bugs.

The CD approach is based on delivering changes little and often. This should not be restricted to software binaries; delivering small incremental chunks of source code *little and often* is also a good practice. If you're able to measure this, you can start seeing who is playing ball and who isn't. One word of warning: don't use this data to reward or punish engineers, as this can promote the wrong kinds of behaviors.

Next, we'll look at the thorny issue of code violations and adherence to rules.

Adherence to coding rules and standards

You may already have coding standards within your software development teams and/or try to adhere to an externally documented and recognized best practice. Being able to analyze your codebase to see which parts do and which parts don't adhere to the standards is extremely useful as it again helps highlight areas of potential risk. There are a good number of tools available to help you do this, some of which are listed in *Appendix A*, *Some Useful Information*.

This type of analysis will take some setting up as it is normally based on a set of predefined rules and thresholds (for example, info, minor, major, critical, and blocker), and you'll need to work with the engineering teams to agree and set these up within your tooling.

This all sounds like hard work—on top of all the other hard work—so is it actually worth it? Yes it is!

Where to start and why bother?

As stated earlier, there are many things that you can and should be measuring, analyzing, and producing metrics for, and there are many tools that can help you do this. You just need to work out what is most important and start from there. The work and effort needed to set up the tools required should be seen as a great opportunity to bring into play some of the good behaviors you want to embed: collaboration, open and honest dialogue, trust, and so on.

As noted earlier, it is better to implement these types of tools early in your CD and DevOps evolution so that you can start to track progress from the get-go. Needless to say, it is not going to be a pretty sight to begin with, and there no doubt be questions around the validity of doing this when it doesn't directly drive the adoption forward—in fact, things might look pretty awful, especially early on.

It might not directly affect the adoption, but it offers some worthwhile additions, which are explained here:

- Having additional data to prove the quality of the software will, in turn, build trust that code can be shipped quickly and safely
- There is a good chance that having a very concise view of the overall codebase will help with the reengineering to componentize the platform
- If the engineers have more confidence in the codebase, they can focus on new feature development without concerns about opening a can of worms every time they make a change

One thing you should also consider is including this kind of measurement and tooling within your CI process. For example, you could include *run code quality analysis* as a step within CI jobs so that the software that doesn't pass the *test* doesn't get shipped. If you think about it, you'll not only be measuring software quality, you'll also have a discreet quality gate to ensure that the code is as it should be.

We'll now move our focus from measuring the act of creating software and look at the importance of measuring what happens when it's built.

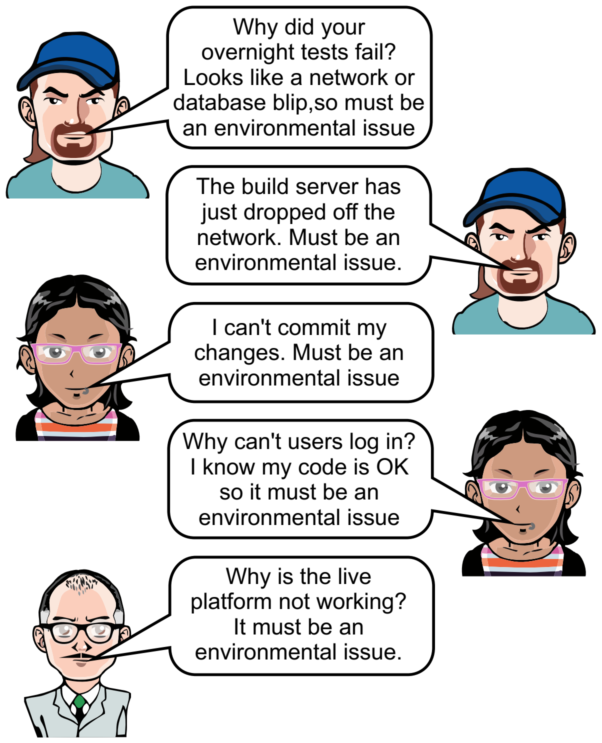
Measuring the real world

Analyzing and measuring your code and engineering expertise is one thing; however, for CD and DevOps to work, you also need to keep an eye on the overall platform, the running software, and the progress of CD and DevOps effectiveness. Let's start with environments.

Measuring the stability of the environments

As mentioned earlier, it might be that you have a number of different environments that are used for different purposes throughout the product-delivery process. As your release cycle speeds up, your reliance on these various environments will grow—if you're working in a two-to-three-month release cycle, having an issue within one of the environments for half a day or so will not have a vast impact on your release, whereas if you're releasing 10 times per day, a half-a-day downtime is a major impact.

There seems to be a universal vocabulary throughout the IT industry related to this, and the term *environmental issue* crops up time and time again, as we can see here:


Why did your overnight tests fail? Looks like a network or database blip,so must be an environmental issue
The build server has just dropped off the network. Must be an environmental issue.
I can't commit my changes. Must be an environmental issue
Why can't users log in? I know my code is OK so it must be an environmental issue
Why is the live platform not working? It must be an environmental issue.

The universal *environmental issue* discussion

We've all heard this, and some of us are just as guilty of saying such things ourselves. All in all, it's not very helpful and can be counterproductive in the long run, especially where building good working relationships across the Dev and Ops divide is concerned, as the implication is that the infrastructure (which is looked after by the operations side) is at fault even though there's no concrete proof.

To overcome this attitude and instill some good behaviors, we need to do something quite simple:

- Prove beyond a shadow of a doubt that the software platform is working as expected, and, therefore, any issues encountered must be based on problems within the infrastructure

Or:

- Prove beyond a shadow of a doubt that the infrastructure is working as expected, and, therefore, any issues encountered must be based on problems within the software

When I said *quite simple*, I actually meant *not very simple*. Let's look at the options we have.

Incorporating automated tests

We've looked at the merits of using automated tests to help prove the quality of each software component as it is being released, but what if you were to group these tests together and run them continuously against a given environment? This way, you would end up with a vast majority of the platform being tested over and over again—continuously in fact. If you were to capture the results of these tests, you can quickly and easily see how healthy the environment is, or, more precisely, you could see if the software is behaving as expected. If tests start failing, we can look at what has changed since that last successful run and try to pinpoint the root cause.

There are, of course, many caveats to this:

- You'll need a good coverage of tests to build a high level of confidence
- You might have different tests written in different ways using different technologies, which do not play well together
- Some tests could conflict with each other, especially if they rely on certain predetermined sets of test data being available
- The tests themselves might not be bullet proof and might not show issues, especially when they have mocking or stubbing included
- Some of your tests might *flap*, which is to say they are inconsistent and for some reason or another fail every now and again
- It could take many hours to run all of the tests end to end (on the assumption that you are running these sequentially)

Assuming that you are happy to live with the caveats or you have resources available to bolster up the tests so that they can be run as a group continuously and consistently, you will end up with a solution that will give you a higher level of confidence in the software platform. Therefore, you should be able to spot instability issues within a given environment with relative ease—sort of.

Combining automated tests and system monitoring

Realistically, just running tests will only give you half the story. To get a truer picture, you could combine your automated test results with the outputs of your monitoring solution (as covered in *Chapter 5*, *Approaches, Tools, and Techniques*). Combining the two will give you a more holistic view of the stability—or not, as the case may be—of the environment as a whole. More importantly, should problems occur, you will have a better chance at pinpointing the root cause(s).

OK, so I've made this sound quite simple, and to be honest, the overall objective is simple; the implementation might be somewhat more difficult. As ever, there are many tools available that will allow you do to this, but again, time and effort is required to get them implemented and set up correctly. You should see this as yet another DevOps collaboration opportunity.

There is, however, another caveat that we should add to the previously mentioned list:

- You might have major issues trying to run some of your automated tests in the production environment

Unless your operations team is happy with test data being generated and torn down within the production database many times per hour/day and they are happy with the extra load that will generate and the possible security implications, this approach might well be restricted to non-production environments. This might be enough to begin with, but if you want a truly rounded picture, you need to look at another complementary approach to gain some more in-depth real-time metrics.

Real-time monitoring of the software itself

Combining automated tests and system monitoring will give you useful data but will realistically only prove two things: the platform is up, and the tests pass. It does not give you an in-depth understanding of how your software platform is behaving or, more importantly, how it is behaving in the production environment being used by many millions of real-world users. To achieve this, you need to go to the next level.

Consider how a Formula One car is developed. We have a test driver sitting in the cockpit who is generating input to make the car do something; their foot is on the accelerator, making the car move forward, and they are steering the car to make it go around corners. You have a fleet of technicians and engineers observing how fast the car goes, and they can observe how the car functions (that is, the car goes faster when the accelerator is pressed and goes around a corner when the steering wheel is turned). This is all well and good, but what is more valuable to the technicians and the engineers is the in-depth metrics and data generated by the myriad of sensors and electronic gubbins deep within the car itself.

This approach can be applied to a software platform as well. You need data and metrics from deep within the bowels of the platform to fully understand what is going on; no amount of testing and observation of the results will give you this. This is not a new concept; it has been around for many years. Just look at any operating system; there are many ways to delve into the depths and pull out useful and meaningful metrics and data. Why not simply apply this concept to software components? In some respects, this is already built in; look at the various log files that your software platform generates (for example, HTTP logs, error logs, and so on), so you have a head start; if only you could harvest this data and make use of it.

There are a number of tools available that allow you to trawl through such outputs and compile them into useful and meaningful reports and graphs. There is a *but* here; it's very difficult to make this generated in real time, especially when there's a vast amount of data being produced, which will take time to fetch and process.

A cleaner approach would be to build something into the software itself, which can produce this kind of low-level data for you in a small, concise, and consistent format that is useful to you—if truth be told, your average HTTP log contains a vast amount of data that is of no value to you at all. I'll cover some examples in *Appendix D, Vital Measurements Expanded*, but simply put, this approach falls into two categories:

- Incorporate a *health-check* function within your software APIs; this will provide low-level metrics data when called periodically by a central data collection solution
- Extend your software platform to *push* low-level metrics data to a central data collection solution periodically

You will, of course, need something to act as the central data collection solution, but as ever, there are tools available if you shop around and work in a DevOps manner to choose and implement what works best for you.

Monitoring utopia

Whatever approach (or combination of approaches) you adopt, you should end up with some very rich and in-depth information. In essence, you'll much have as much data as your average Formula One technician (that being lots and lots of data). You just need to pull it all together into a coherent and easy-to-understand form. This challenge is another one to encourage DevOps behaviors, as the sort of data you want to capture/present is best fleshed out and agreed between the engineers on both sides.

[If you're unsure whether you should measure a specific part of the platform or the infrastructure but feel it might be useful, measure it anyway. You never know whether this data will come in handy later down the line. The rule of thumb is if it moves, monitor it; if it doesn't move, monitor it just in case]

Ultimately, what you want to be able to do is ensure that the entire environment (infrastructure and software platform) is healthy. This way, if someone says *it must be an environmental issue*, they might actually be correct.

If we pull all of this together, we can now expand on the preceding list:

- Prove beyond a shadow of a doubt that the software platform is working as expected, and, therefore, any issues encountered must be based on problems within the infrastructure

  Or:

- Prove beyond a shadow of a doubt that the infrastructure is working as expected, and, therefore, any issues encountered must be based on problems within the software

  Or:

- Agree that problems can occur for whatever reason and that the root cause(s) should be identified and addressed in a collaborative DevOps way

We'll now move on from the technical side of measuring and look at the business-focused view.

Effectiveness of CD and DevOps

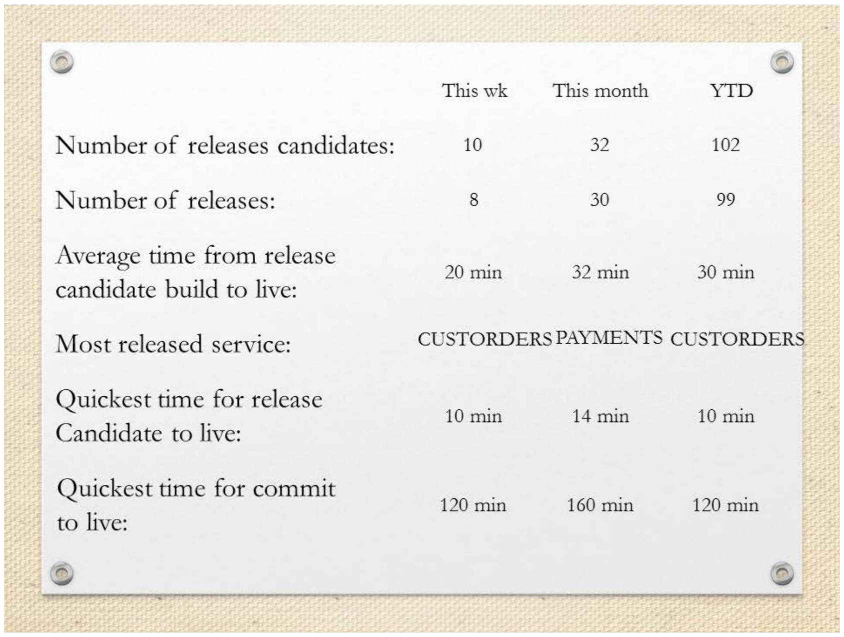
Implementing CD and DevOps is not cheap. There's quite a lot of effort required, which directly translates into cost. Every business likes to see the return on investment, so there is no reason why you should not provide this sort of information and data. For the majority of this chapter, we've been focusing on the more in-depth technical side of measuring progress and success. This is very valuable to technical-minded individuals, but your average middle manager might not get the subtleties of what it means, and to be honest, you can't really blame them. Seeing a huge amount of data and charts that contain information such as **Transactions per second** (**TPS**) counts or response times for a given software component or how many commits were made is not awe inspiring to your average *suit*. What they like is top-level summary information and data, which represents progress and success.

As far as CD and DevOps is concerned, the main factors that are important are improvements in efficiency and throughput, as these translate directly into how quickly products can be delivered to the market and how quickly the business can start realizing the value. This is what it's all about. CD and DevOps is the catalyst to allow for this to be realized, so why not show this?

With any luck, you will have (or plan to have) some tooling to facilitate and orchestrate the CD process. What you should also have built into this tooling is metrics; the sort of metrics that you should be capturing are:

- A count of the number of deployments completed
- The time taken to take a release candidate to production
- The time taken from commit to the working software being in production
- A count of the release candidates that have been built
- A league table of software components that are released
- A list of the unique software components going through the CD pipeline

You can then take this data and summarize it for all to see—it must be simple, and it must be easy to understand. An example of the sort of information you could display on screens around the office could be something like the one shown in the following screenshot:



	This wk	This month	YTD
Number of releases candidates:	10	32	102
Number of releases:	8	30	99
Average time from release candidate build to live:	20 min	32 min	30 min
Most released service:	CUSTORDERS	PAYMENTS	CUSTORDERS
Quickest time for release Candidate to live:	10 min	14 min	10 min
Quickest time for commit to live:	120 min	160 min	120 min

An example page summarizing the effectiveness of the CD process

This kind of information is extremely effective, and if it's visible and easily accessible, it also opens up discussions around how well things are progressing, what areas still need some work and optimization, and so on.

What would also be valuable, especially to management types, is financial data and information, such as the cost of each release in terms of resource and so on. If you have this data available to you, then including it will not only be useful for the management, but it could also help provide focus for the engineering teams, as they will start to understand how much these things cost.

Access to this data and information should not be restricted and should be highly visible so that everyone can see the progress being made and, more importantly, see how far they are away from the original goal.

We've looked at the effectiveness; let's now look at the real-world impact.

Impact of CD and DevOps

Implementing CD and DevOps will have an impact on your ways of working and business as a whole. This is a fact. What would be good is to understand what this impact actually is. You might already capturing and reporting against things such as business **key performance indicators** (**KPI**) (number of active users, revenue, page visits, and so on), so why not add these into the overarching metrics and measurements? If CD and DevOps is having a positive impact on customer retention, then wouldn't it be nice for everyone to see this.

At a basic level, you want to ensure that you are going in the right direction.

Before we move away from measuring and monitoring, let's look at something that, on the face of it, does seem strange: measuring your DevOps culture.

Measuring your culture

I know what you're thinking; measuring software, environments, and processes is hard enough, but how can you measure something as intangible as culture? To be honest, there are no easy answers, and it really depends on what you feel is of most value. For example, you might feel having developers working with system operators 20 percent of their time is a good indication that DevOps is working and is healthy, or the fact that live issues are resolved by developers and the operations team is a good sign.

Capturing this information can also be tricky; however, it needn't be overly complex. What you really need to know is how people feel things are progressing and if they perceive things are progressing in the correct way.

The far simplest way to capture this is to ask as many people as you can. Of course, you'll want to capture some meaningful data points—simply having a graph with the words *it's going OK* doesn't really give you much. You could look at using periodical interviews or questionnaires that capture data such as:

- Do you feel there is an effective level of collaboration between engineers (Dev and Ops)?
- How willing are engineers (Dev and Ops) to collaborate to solve production issues?
- Do you feel blame is still predominant when issues occur?
- Do you feel operations engineers are involved early enough in feature development?
- Are there enough opportunities for engineers (Dev and Ops) to improve their ways of working?
- Do you feel you have the tools, skills, and environment to effectively do your job?
- Do you feel that CD and DevOps is having a positive impact on our business?

There might be other example questions that you can think up; however, don't overdo it and bombard people—KISS (see *Chapter 3*, *Plan of Attack*). If you can use questions that allow for answers in a scale form (for example 1 being strongly agree, 2 being agree, 3 being disagree, and 4 being strongly disagree), you'll be able to get a clearer picture, which you can then compare over time.

Again, if you pool this data with your technical data, this might provide some insights you were not expecting. For example, maybe, you implemented a new process that has reduced the escaped defects by 10 percent, but releases per day have dropped by 5 percent, and the majority of the engineering team is unhappy. In such a case, you might have a problem with the process itself or rather the acceptance of it at grass roots.

Summary

Throughout this chapter, you learned that capturing data and measurements is important, as this gives you a clear indication of whether things are working and progressing in the way you planned and hoped. Whether you're interested in the gains in software quality over time, reduction in bugs, performance of your software platform, or number of *environmental issues* in the past quarter, you need data. Lots of data. Complementing this with business-focused and real-world data will only add value and provide you with more insight into how things are going.

You are striving to encourage openness and honesty throughout the organization (see *Chapter 4*, *Culture and Behaviors*); therefore, sharing all of the metrics and data you collect during your CD and DevOps implementation will provide a high degree of transparency. At the end of the day, every part of any business turns into data, metrics, and graphs (financial figures, head count, public opinion of your product, and so on), so why should the product-delivery process be any different?

The sooner you start to capture this data, the sooner you can *inspect* and *adapt*. You need to extend your mantra from monitor, monitor, and then monitor some more to monitor and measure continuously and consistently.

Let's now move from measuring everything that can and should be measured to see how things look once your CD and DevOps adoption has matured.

8
Are We There Yet?

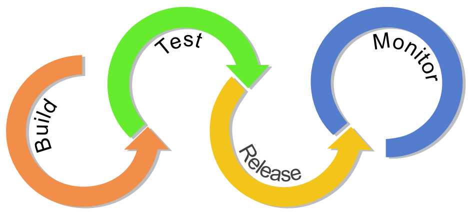
Up until this point, we have been on a journey, from surfacing the issues that caused the business pains through defining the goal and vision to remove them, addressing cultural and technical impediments, adopting the much-needed tools and techniques, and overcoming hurdles, to measuring success.

Let's presume that you are actively implementing CD and DevOps within your organization and, in fact, have been doing so for some time. The business has started to see the benefits and reap the rewards in terms of the ability to deliver quality features to the market sooner. On the face of it, you're almost done, but—and it's a very important but—this is not the end.

The journey you have all been on has been a long one, and just like the 5-year-old who has been sat in the back of the car on the long road trip to grandma's house, you will now have people within your organization repeatedly saying things such as *are we there yet?*, *how much longer?*, and *I need to pee*—okay, maybe not so much of the last one, but I think you get the point. It is now time to pause for a moment and take stock of where you are.

Reflect on where you are now

Yes, you have come a long way, yes things are going much more smoothly, yes the organization is working more closely together, yes the Dev and Ops divide is less of a chasm and more of a small crack in the ground, and yes you have almost completed what you set out to do. You have reduced the process of delivering software from something complex and cumbersome to something as simple as this:



A nice simple process for delivering software

The problems you originally set out to address revolved around the waste within the process of delivering software and, more specifically, the waste that comes from large infrequent releases. Adopting CD and DevOps has helped you overcome these problems. As a result of this, you will now start to hear comments such as *we can deploy quickly, so we must have implemented CD* or *our developers and operations people are working closely together, so we must have implemented DevOps*.

Some would suggest that once you start to hear this, it must mean that you have indeed completed what you set out to do. In some respects, this is true; however, in reality, this is far from the truth.

What these comments do illustrate is the fact that the major issues highlighted at the beginning of the journey have now started to become dim and distant memories. The business has grown to accept CD and DevOps as *the way we do things around here* and has at last started to grasp their meaning—which is good. However, you're not quite done yet. As you did at the beginning of the journey, it is again time to *inspect* and ascertain what problems are important *now*. To explain this, we have to go off on a bit of a tangent.

Streaming

Let's compare your software-release process to a river (I did say it was a bit of a tangent):

- At the very beginning, many small streams flowed downhill and converged into a river. This river flowed along, but the progress was impeded by a series of locks and a massive man-made dam.
- The river then backed up and started to form a reservoir.
- Every few months, the sluice gates were opened, and the water flowed freely, but this was normally a short-lived and frantic rush.
- As you identified and started to remove the man-made obstacles, the flow started to become more even, but it was still hindered by some very large boulders further downstream.
- You then set about systematically removing these boulders one by one, which again increased the flow; this, in turn, started to become consistent, predictable, and manageable.
- As a consequence of removing the obstacles to increase the flow, the water level starts to drop and small pebbles start to appear and create eddies, which restrict the flow to a small degree, but not enough to halt it.
- The flow goes on increasing, the water level goes on decreasing, and it soon becomes obvious that the pebbles were actually the tips of yet more boulders hidden up until this point in the depths of the river.

The flow of software resembles the flow of a river

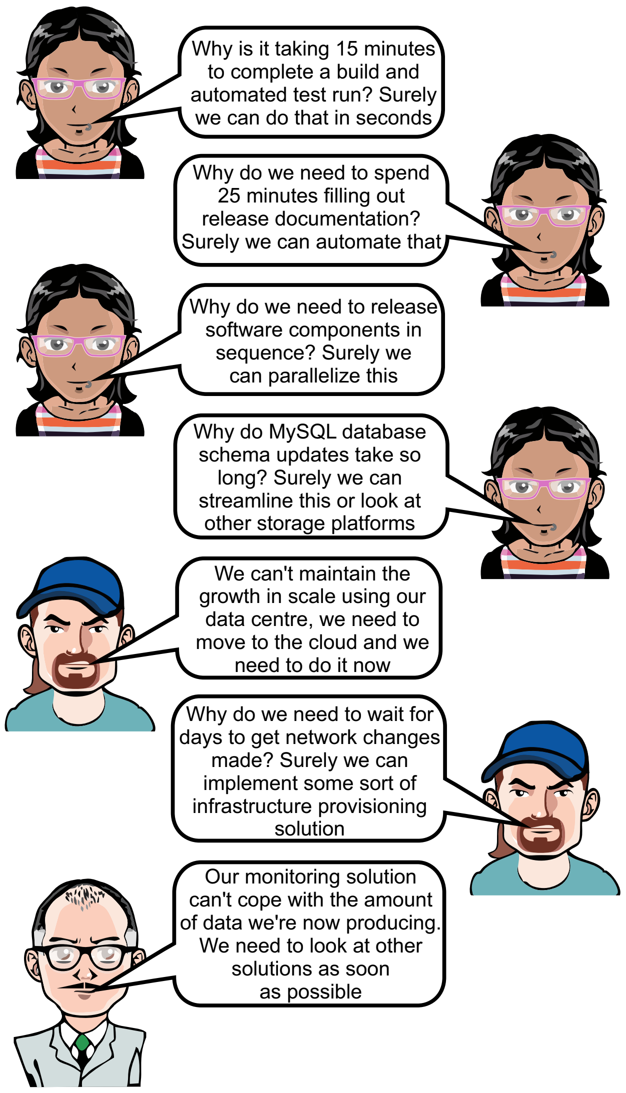
So, what's this got to do with your adoption of CD and DevOps? Quite a lot if you think about it:

- Before you started, you had many streams of work, all converging into one big and complicated release—these were the streams into the river that backed up into the reservoir.
- At the beginning of your journey, you had a pretty good idea of what the major issues and problems were. These were pretty obvious to all and were causing the most pain—these were the locks and dams.
- You removed these obstacles, and the flow started to be more consistent, but it was being hindered by the boulders—these are the lack of engineering best practice, bad culture and behaviors, lack of an open and honest environment, and so on.
- You systematically addressed and removed each of the boulders and started to get a good consistent flow, but new unforeseen issues start to pop up and impede your progress—these are the pebbles that turn out to be more boulders under the waterline.

Your original goal and vision was focused on the major issues highlighted during the *inspect* stage (the manmade locks and dams)—the things you *knew* were problems when you started out. As you systematically worked to address these, the overall process began to flow freely, and you started to see some positive and interesting results. As things progressed, hurdles (the boulders) that were not as obvious or important became more visible and a cause for concern. You then focused your efforts on removing these, which, in turn, improved the overall process. As more improvements are made, more boulders appear. All of a sudden, you have more work to do; this is something you had no way of foreseeing when you set out.

A victim of your own success

Due to the fact that things are now flowing more rapidly and smoothly, even the smallest of problems can start to become a new major issue. These problems can be relatively simple things such as:


Why is it taking 15 minutes to complete a build and automated test run? Surely we can do that in seconds
Why do we need to spend 25 minutes filling out release documentation? Surely we can automate that
Why do we need to release software components in sequence? Surely we can parallelize this
Why do MySQL database schema updates take so long? Surely we can streamline this or look at other storage platforms
We can't maintain the growth in scale using our data centre, we need to move to the cloud and we need to do it now
Why do we need to wait for days to get network changes made? Surely we can implement some sort of infrastructure provisioning solution
Our monitoring solution can't cope with the amount of data we're now producing. We need to look at other solutions as soon as possible

In the space of a few months, the vast majority of the team members originally working within the constraints of big release cycles—which took many weeks or months to pull together, test, and push into the production environment—have all but forgotten the bad old, dark old days and are now finding new things to worry and grumble about. This is nothing unusual; it happens within every project, be it a major business change project or a relatively simple software-development project. It's nothing unusual, but if you think about it, it is a positive problem to have.

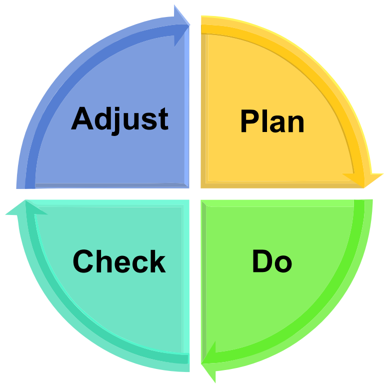
The teams were severely restricted and unable to truly innovate, experiment, or flex their engineering muscles due to the complexity and constraints of the big release process. They no longer have to worry about the process of releasing software, as this has become an everyday background noise that just happens over and over again without the need for much effort—mainly due to the excellent work you have all done.

The seemingly small problems that are now being raised would have been, in the dark days, simple annoyances, which would have been dismissed as low priority. They were pebbles. Now, they are something real, boulder shaped, and they need to be addressed; otherwise, there is a risk that things will slow down, and the days will again become darker.

Does the fact that new problems have surfaced mean that your original goal has not been met and you have failed? No it doesn't. It just means that the landscape has changed. Does this mean you need to change the goal and create a new plan? Not necessarily. What you now need is some PDCA.

[P]lan, [D]o, [C]heck, [A]djust

There are a number of variations of this acronym; however, the most widely used one is **Plan**, **Do**, **Check**, and **Adjust**. You might also find PDCA being referred to as the *Deming circle* or the *Shewhart cycle*. Whatever definition you prefer, the idea behind the PDCA approach is quite simple; it is a framework and approach that you can use for continuous and iterative improvement. The following diagram should help explain this:



The iterative PDCA process

Simply put, this approach is an expansion of the *inspect* and *adapt* approaches that have been mentioned many times previously—although, if truth be told, it's been around for much longer. The concept is pretty easy to grasp and follow and can be applied to almost every aspect of your CD and DevOps adoption. Let's look at an example:

- Plan: You realize that your current process to deliver software is broken and decide that you need to find out why, by running workshops to map out the entire process.
- Do: You run the workshops and capture input and data from across the business.
- Check: You analyze the outputs to ascertain if the data provided gives you an insight into where the pain points are within your process.
- Adjust: You highlight some areas of waste and agree on corrective actions.
- Plan: You set a goal and pull together a plan of attack to address the major pain points.
- Do: You execute against this plan.
- Check: You review the progress against the goal.
- Adjust: You make tweaks to the approach as more information and unforeseen hurdles are unearthed.
- Plan: You realign the plan to ensure that the goal is still achievable, given the new information you have gathered.
- Do ….. I think you can fill the rest in yourself.

As with most of the tools and techniques covered in this book, using the PDCA approach over any other is your call; however, it is a well-proven and well-recognized framework to use—especially when you're looking at implementing something that is as wide reaching and business changing as CD and DevOps—so, I would suggest you don't simply dismiss it out of hand.

One major advantage of PDCA is that it has the luxury of being simple to grasp and understand at all levels of the business, and it is also highly adaptable—for example, this book has been developed using the self-same approach. Do some research and make your own mind up, but before you take action, you should take a step back and take stock of where *you* are and what *you* need to do next.

Exit stage left

The business has gotten used to the changes you have all spent many long hours, days, and months implementing, and it is now experiencing new issues. The question is who should address these new found challenges? The answer is quite simple—not you.

You have helped embed the new collaborative ways of working, helped bridge the gap between Dev and Ops, helped implement new tools and optimized processes, drank lots of coffee, and had little sleep. It's now time for those you have been helping to step up.

Way back in *Chapter 2*, *No Pain, No Gain*, we looked at how to identify the problems and issues the business faced. We called this the elephant in the room. You learned how to use retrospection and other tools to look back and plan forward and how open, honest, and courageous dialogue helped to find the correct path. Now, think of the boulders in the water as a new type of elephant, and you're in exactly the same situation as you were previously. There is, however, one major and very important difference: the business now has the tools and capabilities to identify the elephant-shaped boulders very quickly and now has the tools, knowledge, confidence, experience, and expertise to remove them quickly and painlessly on their own.

As you near your original goal, your swan song is to help others help themselves. It was fun while it lasted, but all good things must come to an end. You would have reached or be very close to reaching your original goal, so now is a good time to consider your exit strategy. This isn't to say that you should not be involved at all; it just means that to fully encourage the fledging ways of working, you need to be the responsible parent and let the kids grow up and learn by their own actions.

Your focus should now change from delivering to assisting and guiding the continuation of delivery. Those who were driving the adoption of CD and DevOps—yourself included—should now start encouraging those who mostly benefitted to step into the light and take responsibility for their own boulders. It's a bit of a shift change but shouldn't be too much of a challenge—especially after all you've been through. One major parental role you can now play is to ensure that complacency doesn't set in.

Rest on your laurels (not)

So, you've done a lot, progressed further, and the business and those working within it are all the better for it. This is a positive and good thing that you and all those involved should be very proud of. However, this is no reason to rest on your laurels; it might be tempting, but now is not the time to simply sit back and admire your handy work.

You have helped the business evolve, but you have to be very mindful of the fact that the business can start to devolve just as easily if complacency sets in. As with any far-reaching project or business change, if the frantic rate of change simply stops, things start to stagnate and old ingrained habits start to resurface. The laggards might start to become noisy again, and the followers might start to listen to them.

You will have actively and notably shifted your position from *doer* to *facilitator* and *influencer*. You also need to be visible and be there to help and assist where needed. Just like a good parent, you have set up a safe environment for growth and self-discovery, and, therefore, you should only need a light touch, a bit of guiding here, some advice there, and the odd prod in the right direction.

When compared to what you have achieved, this might seem simple, but it can be much harder at times; you're used to being actively involved in driving others and doing stuff yourself and now have to help and watch others doing stuff. It's sometimes harder but just as rewarding. You have now taken the next step in your personal evolution, and as such, you are in a good position to look beyond the initial goal to see if there are opportunities to assist in a wider capacity.

Summary

Adopting CD and DevOps is a long and hard journey. If you think it's not, you are deluded. You'll circumvent elephant-filled rivers and other unforeseen challengers as you near the journey's end. Parental guidance is needed to steer the business in the right direction, while you plan how to step out of the limelight and make room for those who have benefited from the achievements you have collectively made. New problems will emerge and threaten the adoption progress; however, the business is wiser and should now have the tools, maturity, and experience to cope. Keeping an eye on things is worthwhile; however, you have bigger and better things to focus on.

9
The Future is Bright

Throughout this book, we have, on the whole, been focused on traditional software delivery within a typical and traditional web/server software-based business. Yes, there are some young, trendy, and innovative software businesses out there that have the agility and opportunity to be creative in the way they deliver software. However, the vast majority of businesses that deliver software on a day-to-day basis are not so lucky—the intention might be there, but the will to act might be lacking. Hence, the focus is on the traditional.

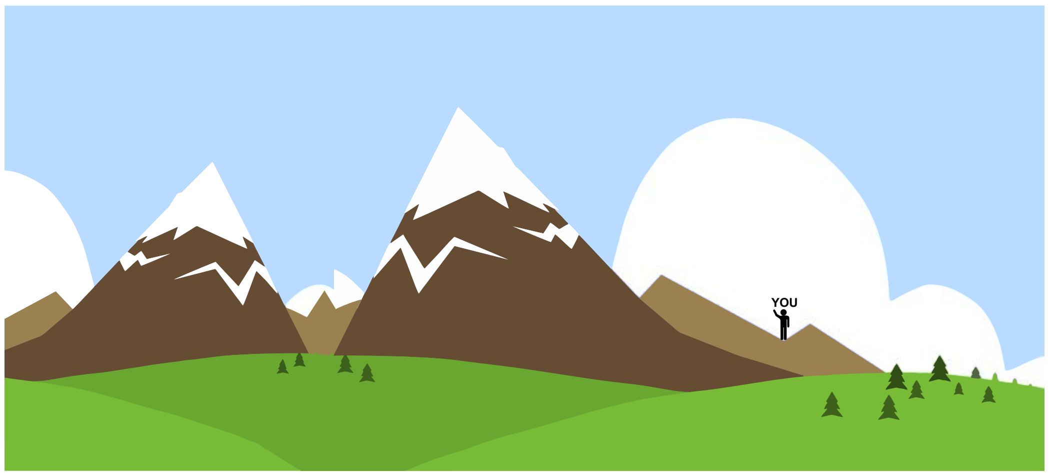
There's a strong possibility that you yourself work within one of these traditional businesses. Having followed the advice provided in this book and successfully adopted CD and DevOps, there's a very good chance that you would have caught up with the whippersnappers, and your business is able to be just as agile and creative in how it delivers software, maybe even more so.

At the tail end of *Chapter 8*, *Are We There Yet?*, we turned our focus onto you and how you could take your newfound knowledge and experience forward, beyond the initial goal of embedding the CD and DevOps ways of working within your organization. Let's look at what this could actually mean.

Expanding your horizon

Let's presume that you have been instrumental in the successful adoption of CD and DevOps and have, on the whole, delivered what you set out to do. Things are working well, even better than you envisaged. The business is all grown up, can tie its own shoe laces, and doesn't need you to hold its hand anymore—well, not quite.

Take a moment to consider where most of the individuals within the business were at the beginning of the journey and where they are now. What you will most probably find is that the vast majority are now at the same point that you were when you started out—they are just starting to realize that there is another way and that it is a better way. Now, look at how far you have come in comparison; you are so far ahead that you are not much more than a small figure in the distance.



Other's perception of you

Regardless of your role at the beginning of the journey, be that a developer, a system admin, a manager, or something else, your role has now changed. Like it or not, you have become the holder of knowledge and experience. You are the CD and DevOps subject-matter expert. You know your stuff.

You have travelled far, the landscape has changed quite dramatically from where you started, and you have new hills to climb—these are the new opportunities that the business is now ready to look at. Maybe these were challenges that the business could not overcome earlier; maybe they simply didn't know these opportunities existed, but with newfound knowledge, they are keen to try new things; maybe your **chief technology officer** (**CTO**) has been chatting with his young and trendy counterparts at the golf club. Whatever the reason, now is the time to apply *your stuff* to these new challenges and opportunities. What follows are some examples of what these could be.

Reactive performance and load testing

The more observant of you might have noticed that there is a little mention of performance or load testing throughout the book. This is intentional as, to my mind, attempting this activity without the close collaboration, tooling, and techniques that come from adopting CD and DevOps is a fool's errand. Yes, there are many established and traditional approaches, but these normally amount to shoehorning something into the process just before you want to ship your code—which might well result in the code not shipping due to the fact that performance issues were found at the last minute. I would also hazard a guess and say performance / load testing was highlighted as a major burden or even an area of waste during the *inspect* stage. It needn't be and shouldn't be the case.

Once you have adopted CD and DevOps, the act of performance / load testing can become relatively simple and straightforward. You just need to change the way you approach it.

Let's assume that you have implemented extensive monitoring of your overall platform from which you can observe in great detail what is going on under the covers. From this, you can glean an idea of how things should look during normal day-to-day operation. With this data, you should then be able to safely run controlled experiments and observe the results in terms of overall platform performance. For example, you could run an experiment to incrementally apply additional load to the platform while it's being used. As the load increases, you start to see where the pain points are—a heat map of sorts. As both Dev and Ops are working closely together, observing the platform as a whole, they should be able to work out where the problems are by comparing normal day-to-day stats to those generated under load.

If issues are pinpointed, they could even apply patches in real time using the CD tooling while the load is still in place—giving instant feedback. Alternatively, they might witness an overall slowdown of the platform, but the monitoring solution doesn't highlight anything specific. This could mean that there is a gap in the overall monitoring coverage.

All in all, trying to run performance or load testing without extensive monitoring in place and/or a high degree of collaboration between the Dev and Ops teams will not provide the results you expect or need. This is not an obvious benefit of adopting CD and DevOps, but it is a very powerful and compelling benefit, as is reducing complexity.

Reducing feature flag complexity

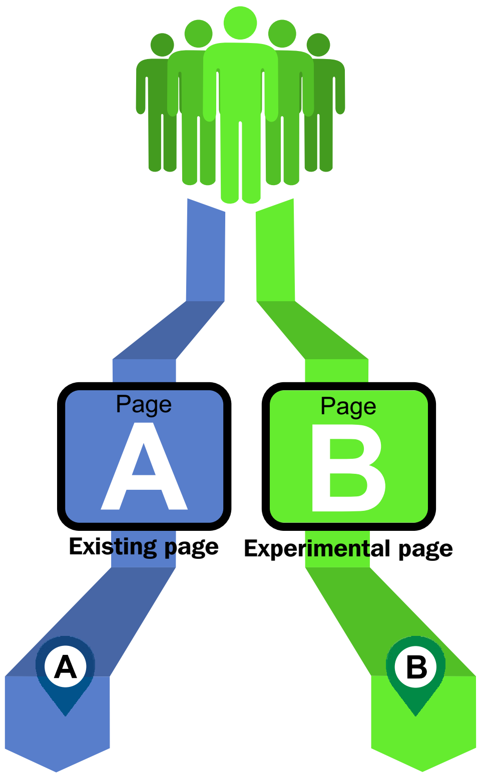
There are many established approaches to allow for different use cases or user flows to be switched on and off in real time, but most revolve around some sort of feature flag or configuration setting within the platform. Although this is a viable approach, it does add something to the code base, which can, over time, become a massive headache—complexity. It also adds complexity to the overall testing—especially if you start to chain the feature flags together (for example, feature C is *on* if feature A is *on*, but feature B is *off*).

Having adopted CD and DevOps, you will be shipping code with ease, and you'll have the Dev and Ops team working as one. Therefore, it would be far simpler to consider using the CD approach to enable and disable features or functionality. In other words, to enable the feature, you just ship the code with it enabled—no messing around with flags, settings, or chaining. OK, so this is an overly simplistic view, but with CD and DevOps, you can start looking at these sorts of problems in new and innovative ways. The advantages might not be immediately obvious, but reducing complexity, if nothing else, will save you time, effort, and the business money. Something that can drive the necessity of feature flags is A/B testing.

Easing A/B testing

Another major benefit to come out of adopting CD and DevOps is the relative ease by which you can implement an A/B-testing approach. I won't go into too much depth regarding this subject—there are plenty of books and on-line resources you could read. However, the top level is this: A/B testing gives you the ability to run different use-cases in parallel and examine the results to see which approach (A or B) worked best.

Maybe you want to see what the impact would be if you introduced a new design or web-page layout. If you can, in some way, force some users down path A and the rest down path B, you can then monitor the user behavior to see which worked best. You can also run A/B experiments covertly. For example, if you have a new recommendation service that you want to try out, you could again force some user traffic to this and see how it works compared to the incumbent service. The possibilities are endless.



A simplistic example of A/B testing

You don't *need* CD or DevOps to implement A/B testing; however, CD does give you the ability to ship code quickly—for example, you want to implement the code to split traffic to A or B across all servers in minutes so that all users start using the same code at the same time. You also have Dev and Ops closely working together, monitoring everything that is going on. If gaps are found in the data used to analyze the results, you have the ability to address this with relative ease. You also have the option to roll everything back if things take an unexpected turn.

Without CD and DevOps, you would need to plan this kind of activity very closely in advance and hope nothing is missing or amiss when you implement it. You will also have to hope that the world hasn't moved on too much between planning and execution and that the use-case you are trying to test is still relevant. Something else that is time critical is security patching.

Security patching and saving your bacon

It seems that every day, the tech press includes a report about the latest business that has been hacked or suffered a **distributed denial-of-service** (**DDOS**) attack. Let's apply this scenario to a traditional software business that has not adopted CD or DevOps. Now think of the answers you would get if you asked the following questions:

- How quickly do you think your average traditional software business would apply a patch to overcome the problem?
- How calm do you think their operations team feels with their CEO, VP of PR, and SVP of operations, all breathing down their collective necks?
- How confident are the Ops team that hastily applying an O/S patch that should have been applied months ago will not impact the software platform?
- How happy do you think the development team will be when the SVP of engineering tells them that they can't go home until they have sorted out a fix to overcome the issues introduced by hurriedly applying an O/S patch?
- How much market value do you think is wiped off a listed company when the news gets out?

It doesn't take a PhD to guess the answers to these questions. Now, imagine the same situation for a business that has adopted CD and DevOps. The answers to the preceding questions would be something like this:

- As quickly as they can normally release—minutes or hours at the most.
- Perfectly calm, and, to be honest, the senior management wouldn't know anything about it until they've been informed that an issue had been found during routine monitoring and was in the process of being addressed.
- Very confident as they can collaborate with the development team to ensure that there are no impacts and/or work on a plan to address the impacts in parallel.
- They won't have to.
- If the message delivered is "We found an issue and have already applied a fix. The impact was minimal and we can assure our customers that their data is perfectly safe", the news isn't very newsworthy, and the markets might not even care. In fact, they might even see it as good news and want to invest more in the business (OK, I can't quantify this, but it is plausible).

As you can see, adopting CD and DevOps can provide some major *bacon-saving* benefits. That isn't to say that you couldn't achieve the same results without CD and DevOps, but having the ability to deliver quickly and having a very close working relationship across the Dev and Ops teams will make it much easier to spot and fix live issues before anything breaks. To reinforce this approach, some people actually try to actively break their live platform on purpose.

Order out of chaos monkey

It doesn't matter how much care and attention you apply to your platform; something will always go wrong when you least expect it. A server will fail, a process might start looping, a network switch decides it doesn't want to be a network switch anymore, a pipe bursts in the office above the server room, or someone decides to hack you because you're a nice big target. As the saying goes, you need to expect the unexpected.

Most businesses will have some sort of business contingency plan in place to cater for the unexpected, but there's a strong possibility that they don't try to force the issue on purpose—they just hope nothing bad will ever happen, and if it does, they hope and pray that they'll be ready and the plan works.

What if you had some tools that could safely initiate a failure at will to see what happens and, more importantly, where the weak spots in your platform are? This is exactly what some bright spark did, and this has been widely adopted as the *chaos monkey* approach. There are a few variations, but what it boils down to is this: a tool that you can run within your closely monitored environment to try and break it.

> The tools currently available are very much focused on cloud-based installations, but the approach could be applied to other environment setups.

If you tried to attempt this without a strongly embedded CD and DevOps culture, you would end up in a complete mess—to be honest, I doubt if you would be even be allowed to try it in the first place. With close collaboration, in-depth monitoring, and trust-based relationships, attempting to break the platform to observe what happens is relatively (but not totally) risk free.

> There is one caveat to this approach; you need to be confident that your platform has been designed and built to gracefully allow for failure. You should avoid committing *platformicide* in public with core dumps and HTTP 500 messages available for all to see.

One other advantage to the *chaos monkey* approach is that it's also a great way to share knowledge of how the overall platform works. Trying to break things on purpose can prove the resilience of your overall platform, but might well annoy customers; let's take a look at how we can keep your customers happy.

End user self-service

Over the course of the book, we have been focused on a unidirectional process of *pushing* software out to a given environment (including production). What if you wanted to turn this around and allow end-users to initiate the *pulling* of your software at will? It might sound strange, but there are a few legitimate scenarios where this could be required.

Maybe you have a team that would like to test out different scenarios and use cases or a SecOps team that needs to run a set of deep security scans or some DDOS scenarios. Traditionally, this would involve quite a large amount of mundane work (that's putting it very mildly) to set up a dedicated environment and get all of the software needed installed and working as it should. What if these teams could press a button and have an entire environment set up for them and they could trust that it has been set up correctly?

With CD and DevOps embedded into your ways of working, you should be able to do this. You will have tooling that can reliably provision servers, deploy software assets, and provide in-depth monitoring. You have a DevOps team who are used to collaborating and are, therefore, happy to help the teams in question. You have a culture that is open and honest so that if problems are found, they can be openly discussed and investigated without fear of blame. Finally, you have the ability to quickly fix and ship if need be. All in all, the activity becomes valuable and not in the least bit mundane or wasteful.

You can most probably think of other scenarios but the point is that with CD and DevOps embedded within your ways of working, you are able to take the load off the Ops and Dev team and at the same time have happy internal customers. This approach can also help build trust—something that again is a pretty powerful asset. Continuing the theme of making customers happy, let's look at how you can widen your horizons and apply CD and DevOps in other ways.

CD and DevOps and the mobile world

CD and DevOps are normally associated with delivering server-based solutions—that's not to say it is exclusively the case; however, this is the norm. The fact that CD and DevOps are based on enhancing culture, behaviors, and ways of working means that they needn't be constrained to this flavor of software delivery. You could, should, and can apply the same approaches to other flavors, for example, the development and delivery of mobile apps.

As mentioned earlier, CD and DevOps are based on culture, behaviors, and ways of working, therefore applying this approach to delivering mobile applications—which is a large and ever-growing industry—can work. There are a couple of caveats in terms of how delivering mobile application software differs from web-based / server-based software delivery.

They are explained here:

- Delivering software to a web platform 10 times per day seamlessly without impacting the end user is achievable—you are in full control of the infrastructure and the mechanism for releasing. Doing the same with a mobile application will have a major impact on the end user—can you imagine what will happen if you send a mobile app to end users' smart phones 10 times per day?
- There are no system operators living within the end users' smart phones / tablets; therefore, the Ops side of the DevOps partnership doesn't strictly exist

So, how do you square this circle? In reality, you don't need to. Most mobile apps are distributed via some sort of app store, which is simply a third-party binary repository—similar to what you would use when shipping binary files to your servers within your CD toolset. When the user wants to install the app, they initiate a process to pull it and install it (very similar to the self-service approach mentioned earlier). When updates are published, the user (or the operating system) simply pulls the binary and installs it.

All pretty simple stuff, so why should you apply CD and DevOps to this? Because you can, and it will add value. The work undertaken to embed collaboration, trust, and honesty within your organization can easily be applied to developing, building, testing, and shipping your mobile apps. You have implemented tools and techniques to automate the process of building, testing, shipping, and monitoring your server platform, so extending these for your mobile apps should be relatively straightforward.

Added to this is the fact that mobile apps can be written in the same technologies as you would use on a server-based website—HTML5, for example. This, in turn, means that the same code base could potentially be shipped to both server and mobile; therefore, using the same techniques, tools, and approaches will make the process seamless.

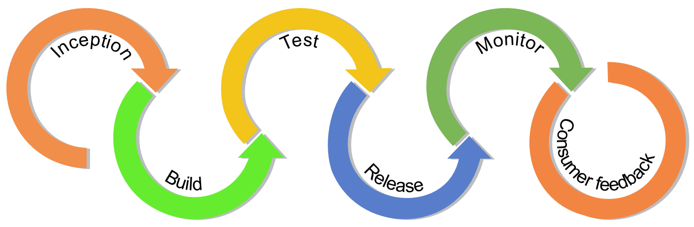
One thing you will need to look into is slightly tweaking the approach in terms of CD. There's nothing stopping you from releasing your app 10 times per day to prove that your process works or to allow for rigorous beta testing. However, you should seriously consider how often you publish this to the app store so that your end users aren't bombarded with app updates' notifications. On the whole, this is a minor issue. Another slight tweak would be to apply the CD and DevOps ways of working outside the world of traditional software delivery.

Expanding beyond software delivery

Despite what you have read, CD and DevOps need not be restricted to simply software/product delivery. The tools, processes, and best practices that come with this way of working can be extended to include other areas of the business outside of traditional IT. For example, let's presume that your product-delivery process is now optimal and efficient, but there are certain business functions that sit before and after the actual product-delivery process; these functions are starting to creak, or, maybe, they are starting to hinder the now highly-efficient product-delivery stage.

There is no reason why using the same techniques covered earlier you cannot address wider reaching business problems. You now have the experience, confidence, and respect to take something that is unwieldy and cumbersome and streamline it to work more effectively, so why not apply this further afield? For example, you could extend the overall product-delivery process to include the *inception* phase (which normally sits before the product-delivery process and is sometimes referred to as the *blue-sky* phase) and the *customer-feedback* phase (which normally sits after you have delivered the product).

The following diagram illustrates this:



Extending the product-creation process to include the pre and post stages

Doing this could provide even greater business value and allow more parts of the business to realize the huge benefits of the CD and DevOps ways of working. As I say, CD and DevOps is not just about delivering software; the way things get done, the collaboration, the open and honest environment, the trust-based relationships, and even the language used can and will help revitalize any business process. It's worth considering whether this is what you would like to do.

What about me?

The preceding are simply examples, but none will have the chance of becoming a reality without someone helping the business and steering it in the right direction. Like it or not, you will have the experience, skills, and reputation as the *go-to guy* for things related to CD and DevOps.

You now have the opportunity to start a new journey and again help the business help itself by driving forward the sort of changes that can only be realized with a mature and strong CD and DevOps culture.

If this doesn't float your boat, then, maybe, keeping up with the ever-changing and ever-growing CD and DevOps landscape is your thing. Just trying to keep up with the new ways to do things, new tools, new ideas, and new insights could take most of your time and attention. More and more businesses are realizing the huge value of having evangelists in their ranks—especially when it comes to software and product delivery.

You might well have hooked yourself into the global CD and DevOps communities, which will give you opportunity to share or present your experiences with others and, more importantly, bring others' experiences and knowledge back into your business. Maybe you could even capture this and publish it on public blogs and forums, or even get it printed in book form. Stranger things have happened.

Whatever you choose to do, you will not be bored, nor will you be able to go back to how things were. You have learned a very valuable lesson—there is a better way, and CD and DevOps is it.

What have you learned?

I keep making references to your experience, knowledge, and expertise, but until you have actually gone through the motions of adopting and implementing CD and DevOps, this will amount to what you have read. Let's take a final chance to recap what we have covered:

- CD and DevOps are not just about technical choices and tools; a vast amount of the success is built on the behaviors, culture, and environment.
- Implementing and adopting CD and DevOps is a journey that might seem long and daunting at first, but once you've taken the first step and then put one foot in front of the other, you'll hardly notice the miles passing.
- Teams who have successfully implemented CD and DevOps seldom regret it or are tempted to go back to the bad old days when releases were synonymous with working weekends and late nights—working late nights and weekends should be synonymous with innovation and wanting to create some killer app or the next world-changing technology breakthrough.
- You don't have to implement both CD and DevOps at the same time, but one complements the other. You don't have to, but you should seriously consider it.
- Where you do need to make technical choices, ensure that you implement something that enhances and complements your ways of working—never change your ways of working to fit the tooling.
- It can be big and scary, but if you start with your eyes wide open, you should be able to get through. There is a global community available that can help, assist, and give advice, so don't be afraid to reach out.
- Don't simply start implementing CD or DevOps just because it's the next new thing that everyone else is doing. You need to have a good reason to implement both/either, or you will not reap the benefits, nor truly believe in what you are doing.
- Although we have covered a vast amount, you don't have to implement everything you have read about; take the best bits that work for you and your situation, and go ahead from there—just as you would with any good *agile* methodology.
- Just because you can ship software, it doesn't mean you are done. CD and DevOps are ways of working, and the approaches within can be applied to other business areas and problems.
- Share failures and successes so that you learn, and others have the opportunity to learn from you.

Summary

This book, like all good things, has come to an end. As pointed out numerous times earlier, we've covered quite a lot in a few pages. This book is, by no means, the definitive opus for CD and DevOps; it is merely a collection of suggestions based on experience and observations.

Even if you are simply window shopping and looking at what is needed to implement and adopt CD and DevOps ways of working, you should now have a clearer idea of what you are letting yourself and your organization in for; forewarned is forearmed as they say. It's not an easy journey, but it is worth it.

So, go grab yourself a hot beverage, a notepad, and a pen; skip back to *Chapter 2, No Pain, No Gain*; and start mapping out why you need to implement CD and DevOps and how you are going to do it.

Go on then, stop reading, go!

Good luck!

A
Some Useful Information

Although this book provides some (hopefully) useful information, there's only so much space available. I've, therefore, compiled a list of additional sources of information that will complement this book. I've also included a list of the many subject-matter experts out there who might be able to provide further assistance and guidance as you progress along your journey. Additional resources can be found on my website `http://www.swartout.co.uk`.

What follows is, by no means, an exhaustive list, but it is a good start.

Tools

Some of the following tools are mentioned within this book, and some are considered the best of breed for CD and DevOps:

Tool	Description	Where to find more information
Jenkins	An award-winning and world-renowned open source CI tool	`http://jenkins-ci.org/`
GIT	A free and open-source distributed version-control system	`http://git-scm.com/`
GitHub	An online-hosted community solution based on GIT	`https://github.com/`
Graphite	A highly-scalable real-time graphing system that allows you to publish metric data from within your application	`http://graphite.wikidot.com/`
Tasseo	A simple-to-use Graphite dashboard	`https://github.com/obfuscurity/tasseo`

Tool	**Description**	**Where to find more information**
SonarQube	An open platform to manage code quality	`http://www.sonarqube.org/`
Ganglia	A scalable distributed-monitoring system for high-performance computing systems	`http://ganglia.sourceforge.net/`
Nagios	A powerful monitoring system that enables organizations to identify and resolve IT-infrastructure problems before they affect critical business processes	`http://www.nagios.org/`
Dbdeploy	An open source database change management tool	`http://dbdeploy.com/`
Puppet Labs	A tool to automate the creation and maintenance of IT infrastructure	`http://puppetlabs.com/`
Chef	Another tool to automate the creation and maintenance of IT infrastructure	`https://www.getchef.com/chef/`
Vagrant	A tool to build complete development environments using automation	`https://www.vagrantup.com/`
Docker	An open platform for distributed applications	`https://www.docker.com/`
Yammer	An Enterprise private social network (think of it as a corporate Facebook)	`https://www.yammer.com`
HipChat	A private group-chat and team collaboration tool	`https://www.hipchat.com/`
IRC	The granddaddy of collaboration and chat tools	`http://www.irc.org/`
Campfire	A private group-chat and team collaboration tool	`https://campfirenow.com/`
Hubot	An automated "bot" that can be set up within most chat-room systems	`https://hubot.github.com/`
Trello	An online scrum / Kanban board solution	`https://trello.com/`
AgileZen	An on-line scrum / Kanban board solution	`http://www.agilezen.com/`

People

What follows is a list of people who are actively involved in the agile and continuous delivery and DevOps communities:

- Patrick Debois is seen by many in the DevOps community as the daddy of DevOps and the founder of the DevOpsDays movement (`http://devopsdays.org/`). This relatively small get together of like-minded individuals in 2009 has grown into a global gathering. For more information on Patrick Debois, visit `http://www.jedi.be/`.
- John "Botchagalupe" Willis is a regular and well-renowned contributor to the DevOps community and has inspired many with his honest way of sharing his wisdom. For more information on him, visit `http://www.johnmwillis.com/`.
- Jez Humble is the co-author of the *Continuous Delivery* book that is used by many as the definitive reference material when investigating or implementing continuous delivery. He also actively contributes to the continuous-delivery blog (`http://continuousdelivery.com/`). For more information on him, visit `http://jezhumble.net/`.
- John Allspaw is the SVP of Operations at Etsy.com and seems to understand the value of DevOps—even though he's one of the senior management types. For more information on him, visit `http://www.kitchensoap.com/`.
- Gareth Rushgrove is a self-confessed Web geek, who seems to somehow find time to produce the DevOps weekly e-mail newsletter (`http://devopsweekly.com/`), which is full of useful and insightful information. For more information on him, visit `http://www.garethrushgrove.com/`.
- Gene Kim, co-author of *The Phoenix Project*, is the founder and former CTO of Tripwire. He is passionate about IT operations, security, and compliance, and how IT organizations successfully transform from *good to great*. For more information on him, visit `http://www.realgenekim.me/`.
- Mitchell Hashimoto is a self-confessed DevOps tools mad scientist and the creator of Vagrant, Packer, Serf, Consul, and Terraform. For more information on him, visit `http://about.me/mitchellh`.
- Steve Thair is the cofounder of DevOpsGuys and a regular speaker on Web performance. For more information on him, visit `http://www.devopsguys.com/About/Team/Steve`.
- Rachel Davies is an internationally-recognized expert in coaching teams on the effective use of agile approaches and has a wealth of knowledge when it comes to retrospective techniques and games. For more information on her, visit `http://www.agilexp.com/agile-coach-rachel-davies.php`.

- Ken Schwaber is the godfather of scrum and agile. For more information on him, visit `http://kenschwaber.wordpress.com/`.
- John Clapham is an all-round nice guy and agile/DevOps evangelist. For more information on him, visit `http://johnclapham.wordpress.com/`.
- Karl Scotland is a renowned agile coach who specializes in lean and agile techniques. For more information on him, visit `http://availagility.co.uk/`.

Recommended reading

The following books are well worth a read—even if you don't decide on some strange reason to adopt CD and/or DevOps:

Resource	**Description**	**Link**
Agile Coaching	A nice introduction on how to become a good agile coach	`https://pragprog.com/book/sdcoach/agile-coaching`
Agile Retrospectives: Making Good Teams Great	An excellent book that covers most of what you need to know to run effective retrospectives	`https://pragprog.com/book/dlret/agile-retrospectives`
Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation	The CD bible	`http://www.amazon.com/dp/0321601912?tag=contindelive-20`
The phoenix project	A unique take on DevOps adoption in fiction form, well worth a read	`http://itrevolution.com/books/phoenix-project-devops-book/`
Agile Product Management with Scrum	View scrum and agile from the product managers' point of view	`http://www.amazon.com/exec/obidos/ASIN/0321605780/mountaingoats-20`

Resource	**Description**	**Link**
The Enterprise and Scrum	This book provides some addition insight into the challenges of adopting an agile approach and ways of working	`http://www.amazon.com/exec/obidos/ASIN/0735623376/mountaingoats-20`
The Lean Startup	Real-life experiences and insights into how to transform your business, culture, and ways of working	`http://amzn.com/0307887898`
Getting Value out of Agile Retrospectives	Gives a good introduction on retrospectives and provides a good, long list of games/exercises	`https://leanpub.com/gettingvalueoutofagileretrospectives`

B
Where Am I on the Evolutionary Scale?

It's sometimes difficult to ascertain where your business sits on the CD and DevOps evolutionary scale; however, some simple questions can help you get a rough idea. For example, in relation to your business:

- Does it favor process over people?
    1. Process
    2. People
    3. We don't have any processes worth mentioning, so I suppose it's people
- Do immoveable deadlines in project plans take precedence over delivering quality solutions incrementally?
    1. Yes, meeting deadlines is the only thing that matters
    2. We have flexibility to make small changes and replan to ensure that quality doesn't suffer
    3. We do whatever is needed to keep the customer happy
- Are your projects run with fixed timescales, fixed resources, and fixed scope, or is there flexibility?
    1. Yes, and this is all agreed up front, signed off, and intricately planned
    2. No, we have flexibility in at least one of these areas
    3. We do whatever is needed to keep the customer happy

- Is failure scorned upon or used as something to learn from?
    1. Failure is failure, and there are no excuses—heads will roll
    2. We ensure that failures have a small impact and learn from our mistakes
    3. Failure means no more business, and we're all out of our jobs
- Who is on call for out-of-hours production issues?
    1. The T1 helpdesk with T2 operations support and T3 applications support teams backing them up
    2. We normally have a "point man" on call who can reach out to anyone he needs
    3. Everyone
- Are you able to ship code when it is ready, or do you have to wait for a scheduled release?
    1. The release team schedules and agrees the delivery to production via the change advisory board (CAB) and transition team, based on the agreed program plan
    2. We trust our engineers to ship code using our deployment tools when they are confident that it is ready and doesn't compromise the overall quality
    3. Our engineers normally use file transfer protocol (FTP) to transfer the code to the production servers when it's finished compiling

If you were to apply these to the ACME systems' business at certain points through their evolution, you would find that the version 1.0 business would mostly answer *3* to all questions, the version 2.0 business would mostly answer *1*, and the highly evolved version of the business would mostly answer *2*.

C
Retrospective Games

Retrospectives are normally the *inspect* part of the agile *inspect and adapt*. If you are aware of or are using scrum or some other agile methodology, then running retrospectives should be nothing new. If you have never run a retrospective before, then you would have some fun things to learn.

The remit of a retrospective is to look back over a specific period of time, project, release, or simply a business change and highlight what worked well, what didn't work well, and what improvements are needed. This process can traditionally be a bit dry, so retrospectives tend to be based on games (some people refer to these as "exercises", but I prefer the word "games"), which encourages collaboration, engagement, and injects a bit of fun.

As with any game, there are always rules to follow. Here are some example rules:

- Each session should be strictly time-boxed
- Everyone should be given a voice and a chance to actively contribute
- Everyone should be able to voice their opinion but not at the expense of others
- Whoever is facilitating the session is in charge and should control the session as such
- The session should result in tangible and realistic actions that can be taken forward as improvements

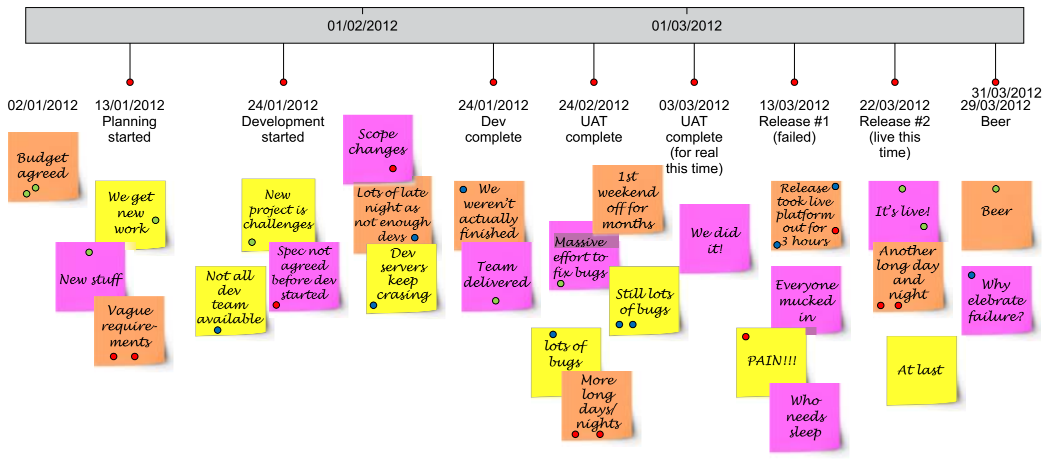
As with the value-stream mapping technique mentioned in *Chapter 2*, *No Pain, No gain*, the only tools you really need are pens, paper, a whiteboard (or simply a wall), some space, and some sticky notes.

Let me introduce you to a couple of my favorite games: timeline and *StoStaKee*. We'll start with the timeline game.

The timeline game

The timeline game, as the name suggests, revolves around setting up a timeline and getting your invited guests to review and comment on what happened during the period of time in question. There are a number of variations of this game, but in essence, it revolves around the entire team writing out sticky notes related to notable events during the period in question and indicating how the events made them feel using sticky dots (*green* stands for *glad*, *blue* for *sad*, and *red* for *mad*). From this, you have an open and honest discussion on those events that provoked the most emotions and agree on actions to take forward (for example, things to stop doing, start doing, and keep doing).

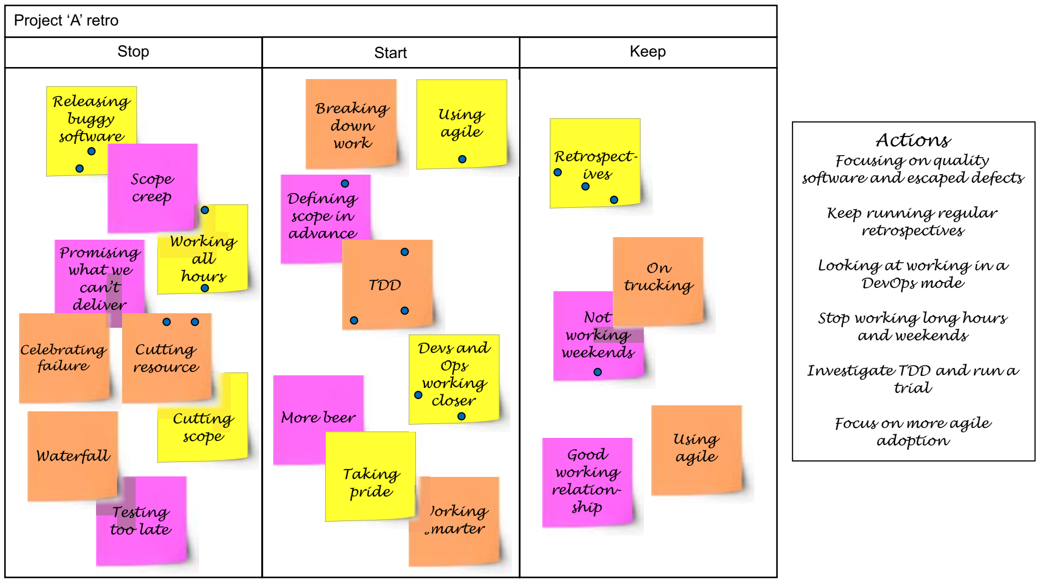
The following figure depicts a typical timeline wall:


01/02/2012
01/03/2012
02/01/2012
13/01/2012 Planning started
24/01/2012 Development started
24/01/2012 Dev complete
24/02/2012 UAT complete
03/03/2012 UAT complete (for real this time)
13/03/2012 Release #1 (failed)
22/03/2012 Release #2 (live this time)
31/03/2012
29/03/2012 Beer
Budget agreed
We get new work
New stuff
Vague require-ments
New project is challenges
Not all dev team available
Spec not agreed before dev started
Scope changes
Lots of late night as not enough devs
Dev servers keep crasing
We weren't actually finished
Team delivered
Massive effort to fix bugs
1st weekend off for months
Still lots of bugs
lots of bugs
More long days/nights
We did it!
Release took live platform out for 3 hours
Everyone mucked in
PAIN!!!
Who needs sleep
It's live!
Another long day and night
At last
Beer
Why elebrate failure?

StoStaKee

This stands for **stop**, **start**, and **keep**. Again, this is an interactive time-boxed exercise focused on past events. This time, you ask everyone to fill in sticky notes related to things they would like to stop doing, start doing, or keep doing, and add them to one of three columns (stop, start, and keep). You then get everyone to vote—again with sticky dots—on the ones they feel most strongly about. Again, you should encourage lots of open and constructive discussions to ensure that everyone understands what each note means. The end goal is a set of actions to take forward.

The following figure depicts a typical StoStaKee board:


Project 'A' retro
Stop
Start
Keep
Releasing buggy software
Scope creep
Working all hours
Promising what we can't deliver
Celebrating failure
Cutting resource
Cutting scope
Waterfall
Testing too late
Breaking down work
Using agile
Defining scope in advance
TDD
Devs and Ops working closer
More beer
Taking pride
Jorking .marter
Retrospect-ives
On trucking
Not working weekends
Good working relation-ship
Using agile
Actions
Focusing on quality software and escaped defects
Keep running regular retrospectives
Looking at working in a DevOps mode
Stop working long hours and weekends
Investigate TDD and run a trial
Focus on more agile adoption

The preceding examples are a mere subset of what is available, but both have proven time and time again to be the most effective in investigating and, more importantly, understanding the issues within a broken process.

D
Vital Measurements Expanded

Chapter 7, *Vital Measurements*, introduced you to a number of different ways of measuring certain aspects of your processes. We will now expand on some of these and look in more detail at what you could/should be measuring. We'll start by revisiting code complexity and the science behind it.

Code complexity – some science

As mentioned in *Chapter 7*, *Vital Measurements*, having complex code in some circumstances is fine and sometimes necessary; however, overly complex code can cause you lots of problems, especially when trying to debug or when you're trying to extend it to cater to additional use cases. Therefore, being able to analyze how complex a piece of code is should help.

There are a few documented and recognized ways of measuring the complexity of source code, but the one most referred to is the cyclomatic complexity metric (sometimes referred to as MCC or McCabe Cyclomatic Complexity) introduced by Thomas McCabe in the 1970s. This metric has some real-world science behind it, which can, with the correct tools, provide quantifiable measurements based on your source code. The MCC formula is calculated as follows:

$M = E - N + X$

In the preceding formula, M is the MCC metric, E is the number of edges (the code executed as a result of a decision), N is the number of nodes or decision points (conditional statements), and X is the number of exits (return statements) in the graph of the method.

Code versus comments

Including comments within your source will make it much more readable, especially in the future when someone other than the original author has to refactor or bug fix the code. Some tools will allow you to measure and analyze the ratio of code versus comments.

That said, some software engineers don't believe that comments are worthwhile and believe that if another engineer cannot read the code, then they're not worth their salt. This is one view; however, including comments within one's source should be encouraged as a good engineering practice and good manners.

One thing to look out for should you implement a code-versus-comments analysis is those individuals who get around the rules by simply including things such as the following code snippet:

```
/**
 * This is a comment because I've been told to include comments in my
 * code
 * Some sort of code analysis has been implemented and I need to
 * include comments to ensure that my code is not highlighted as poor
 * quality.
 *
 * I'm not too sure what the percentage of comments vs code is
 * required but if I include lots of this kind of thing the tool will
 * ignore my code and I can get on with my day job
 *
 * In fact this is pretty much a waste of time as whoever is reading
 * this should be looking at the code rather than reading comments.
 * If you don't understand the code then maybe you shouldn't be trying
 * to change it?!?
 */
```

This might be a bit extreme, but I'm sure if you look close enough at your codebase, you might well find similar sorts of things hidden away.

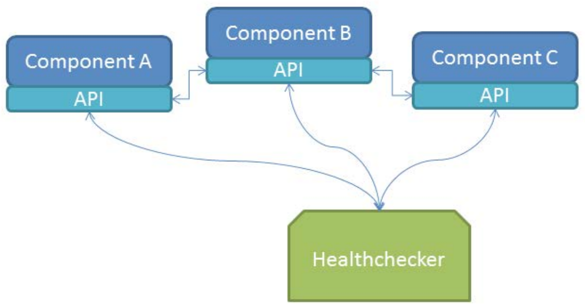
One other good reason for comments—in my experience—is for those situations when you have to take the lid off some very old code (by today's standards, very old could be a couple of years) to investigate a possible bug or simply find out what it does. If the code is based on outdated design patterns or even based on an old language standard (for example, an older version of Java or C#), it might be quite time-consuming trying to understand what the code is doing without, at least, some level of commenting.

Embedding monitoring into your software

As mentioned in *Chapter 7*, *Vital Measurements*, there are a few ways you can include and embed the generation of metrics within the software itself.

Let's assume that your software components contain APIs that are used for component-to-component communication. If you were able to extend these APIs to include some sort of a health-check functionality, you could construct a tool that simply calls each component and asks the component how it is. The component can then return various bits of data, which indicates its health. This might seem a bit convoluted, but it's not that difficult.

The following diagram gives an overview of how this might look:



A health-checker solution harvesting health-status data form software components

In this example, we have a health-checker tool that calls each component via the APIs and gets back data that can then be stored, reported, or displayed on a dashboard. The data returned can be as simple or complex as you like. What you're after is to ascertain whether each component is healthy. Let's say, for example, one element of the data returned indicated whether or not the software component could connect to the database. If this comes back as false and you notice that the system monitor looking at the free disk space on the database server is showing next to zero, you can very quickly ascertain what the problem is and rectify it.

This method of monitoring is good but relies on you having some tooling in place to call each component in turn, harvest the data, and present it to you in some readable/usable form. It's also restricted to what the APIs can return or rather how they are designed and implemented. If, for example, you wanted to extend the data collection to include something like the number of open database connections, you will need to change the APIs, redeploy all of the components, and then update the tooling to accept this new data element. This is not a huge problem, but a problem all the same. What could be a huge problem, though, is the single point of failure, which is the tooling itself. If this stops, working for whatever reason, you're again blind, as you don't have any data to look at, and, more importantly, you're not harvesting it.

There is an alternative approach that can overcome these problems. In this approach, the component itself generates the metrics you need and pushes the data to your tooling. Something like Graphite does this very well. Instead of extending the APIs, you simply implement a small amount of code; this allows you to fill up buckets of metrics data from within the software component itself and push these buckets out to the Graphite platform. Once in graphite, you can interrogate the data and produce some very interesting real-time graphs. Another advantage of graphite is the plethora of tools now available to generate and create very effective graphs, charts, and dashboards based on the Graphite data.

Bibliography

This course is a blend of text and quizzes, all packaged up keeping your journey in mind. It includes content from the following Packt products:

- *Practical DevOps, Joakim Verona*
- *DevOps Automation Cookbook, Michael Duffy*
- *Continuous Delivery and DevOps – A Quickstart Guide - Second Edition, Paul Swartout*