



Free Sample

Quick answers to common problems

DevOps Automation Cookbook

Over 120 recipes covering key automation techniques through code management and virtualization offered by modern Infrastructure-as-a-Service solutions

Michael Duffy

[PACKT] open source*
PUBLISHING community experience distilled

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 2 '**Ad Hoc Tasks with Ansible**'
- A synopsis of the book's content
- More information on **DevOps Automation Cookbook**

About the Author

Michael Duffy is a technology consultant who spends far too much of his time getting excited about automation tools. Michael lives in a tiny village in Suffolk and when he isn't reading, writing, or playing with automation and infrastructure tools, he can be found spending as much time as he can with his family.

He runs his own consultancy, Stunt Hamster Ltd, and spends a lot of time telling clients that DevOps is an approach rather than a job title. Stunt Hamster Ltd. has provided services to clients as large as Telefonica O2 and BskyB and is currently working on software to ease the pain of managing decentralized platforms.

Michael has previously written *Puppet Reporting and Monitoring*, published by Packt Publishing.

This book would not have been possible without my amazing wife Bethan. I would not have been able to complete this book without her boundless patience, love, and understanding.

I also want to thank my fantastic daughter, Meg, and my incredible son, Griff; you guys have patiently put up with Daddy hiding in his office and have been a source of absolute joy to me.

One last vote of thanks must go to the editors and especially to the dedicated band of reviewers; without you guys, this book would have contained much more gibberish than it actually does.

Preface

DevOps has created a lot of excitement in recent years and looks certain to make the same impact as Agile software development on the software industry. This is not entirely surprising; DevOps has largely been born from the frustration of Agile developers trying to work within the traditional confines of infrastructure support and delivery. Their attempts to find more efficient ways to deliver reliable, performant, and secure software to the end user has led us to DevOps.

DevOps initially came to people's attention in 2008 when the first DevOps day conference was held. It was organized by *Patrick Debois*; it brought together like-minded people for the first time to discuss how the delivery of infrastructure could be made more agile. Originally, the preferred term for what eventually became DevOps was Agile Infrastructure but the portmanteau of Development and Operations made for a friendlier Twitter tag and the term stuck. From here, the attention and interest in DevOps grew and today there are DevOps day conferences worldwide.

DevOps breaks down the barriers between the operations and development teams and allows a tight collaboration between these traditionally firewalled areas. The resulting cross-functional team will be able to react faster to the changes in the software requirements and deliver the best of breed solutions. This has led to a renaissance in areas such as monitoring and deployment, where the development team may once have lobbed a tarball over the corporate firewall to the operations department to install. The developers instead created a robust set of automated provisioning scripts to manage installations themselves. Likewise, monitoring has started to cease to be an exercise in testing if a port is available or if the server has run out of disk space (although this is still essential) and has become a holistic approach that takes into account the health of the infrastructure, load on the application, number of errors generated, and so on. This is only possible if you have a team that is truly cross-functional and with a deep understanding of the software they manage.

Defining what can be considered as a DevOps tool is incredibly difficult but the rapid increase of companies utilizing DevOps techniques has led to an explosion of new tools with a particular focus on automation, monitoring, and testing. Tools, such as Puppet, Chef, CF Engine, and Ansible have grown massively in popularity, thus allowing developers to truly define the underlying infrastructure using the code. Likewise, new monitoring tools, such as Sensu, have appeared that take up the challenge of monitoring ephemeral infrastructures, such as cloud-based services.

This book is different from most of the other technical cookbooks. Rather than keeping a laser-like focus on a single technology, this cookbook serves as an introduction to many different tools. Each chapter offers recipes that show you how to install and utilize tools that tackle some of the key areas that a team using DevOps techniques will encounter. Using it, you can quickly get up to speed with diverse areas, such as *Automation with Ansible*, *Monitoring with Sensu*, and log analyses with LogStash. By doing the further reading outlined with each recipe, you can find pointers to gain a deeper insight into these fantastic tools.

What this book covers

Chapter 1, Basic Command Line Tools, covers some basic but incredibly useful tools for trouble shooting servers and managing code.

Chapter 2, Ad Hoc Tasks with Ansible, contains recipes that allow you to use the powerful Ansible automation tool to run one off commands for server management.

Chapter 3, Automatic Host Builds, covers recipes that allow you to automate the build and configuration of the most basic building block in your infrastructure servers.

Chapter 4, Virtualization with VMware ESXi, contains recipes that show how to install and use the popular ESXi hypervisor to create, manage, and use powerful virtual servers.

Chapter 5, Automation with Ansible, covers the incredibly powerful configuration management tool, Ansible. These recipes demonstrate how to create a powerful and reusable code to manage the configuration of your infrastructure.

Chapter 6, Containerization with Docker, covers the increasingly popular world of containerization. Containerization is an incredibly powerful technique for distributing and running software and these recipes show you how to use Docker to create, run, and maintain containers.

Chapter 7, Using Jenkins for Continuous Deployment, contains recipes that show you how Ansible can be used with the powerful Jenkins CI tool to create an automated build and deploy system.

Chapter 8, Metric Collection with InfluxDB, demonstrates how to use the powerful Time Series database InfluxDB to capture and analyze metrics generated by your infrastructure and present them in attractive and easy-to-understand formats.

Chapter 9, Log Management, demonstrates how to use powerful tools to centralize, collect, and analyze valuable log data.

Chapter 10, Monitoring with Sensu, covers using this powerful, scalable, and customizable monitoring system to demonstrate how to install, configure, and manage Sensu.

Chapter 11, IAAS with Amazon AWS, covers recipes that demonstrate how to set up infrastructure using the powerful AWS Infrastructure-as-a-Service. It also covers topics, such as EC2 servers, DNS management, and security.

Chapter 12, Application Performance Monitoring with New Relic, introduces the NewRelic application performance monitoring tool and demonstrates how to use it to monitor servers, applications, and more.

What you need for this book

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.

Who this book is for

If you are a systems administrator or developer who is keen to employ DevOps techniques to help with the day-to-day complications of managing complex infrastructures, then this book is for you.

Sections

In this book, you will find several headings that appear frequently (Getting ready, How to do it, How it works, There's more, and See also).

To give clear instructions on how to complete a recipe, we use these sections as follows:

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

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, it's important to understand the strengths and weaknesses of each one. I have chosen Ansible primarily for its ease of use, simplicity of deployment, and its 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 possesses 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 one 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

[Get more information DevOps Automation Cookbook](#)

Where to buy this book

You can buy DevOps Automation Cookbook from the [Packt Publishing website](#).

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

[Click here](#) for ordering and shipping details.



www.PacktPub.com

