Community Experience Distilled

Extending Ansible

Discover how to efficiently deploy and customize Ansible in the way your platform demands

Rishabh Das

[PACKT] PUBLISHING

Extending Ansible

Discover how to efficiently deploy and customize
Ansible in the way your platform demands

Rishabh Das

[PACKT]
PUBLISHING

BIRMINGHAM - MUMBAI

Extending Ansible

First published: March 2016

Production reference: 1210316

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78217-500-1

www.packtpub.com

Credits

Author	**Project Coordinator**
Rishabh Das	Nikhil Nair
Reviewer	**Proofreader**
Xiang Zhang	Safis Editing
Commissioning Editor	**Indexer**
Veena Pagare	Tejal Soni
Acquisition Editor	**Graphics**
Subho Gupta	Kirk D'Penha
Content Development Editor	**Production Coordinator**
Divij Kotian	Melwyn D'sa
Technical Editor	**Cover Work**
Vivek Pala	Melwyn D'sa
Copy Editor	
Lauren Harkins	

About the Author

Rishabh Das, presently working with Red Hat India, is responsible for managing and maintaining the CI/CD workflow and infrastructure for his team. He has more than 3 years of industry experience and has extensive hands-on experience with Ansible. You can reach Rishabh on Twitter at `@rshbhdas`.

This book would have not seen the light of day without the support of many individuals—both known and unknown. I would like to express my gratitude to the Ansible and Python communities for making my life easier. Thank you to everyone who is part of this open source community for all the amazing work that you do.

I would like to thank my colleagues who helped me out at each step with their varying abilities. I couldn't have asked for more encouragement and support. Thank you for all the awesome projects you work on, giving me an opportunity to learn, explore, and push my limits.

Thanks to my friends and family, who have always been there no matter the time, situation, or place. Thanks for all the things you have done, for all the ideas you pitched in, for my craziness that you put up with, and for all the compromises you made to help me finish off what I started.

Lastly, but most importantly, I want to specially thank my father, who every single day, checked on my progress. I couldn't have reached this far without you. Thank you ma and baba.

About the Reviewer

Xiang Zhang is a systems engineer working for SINA. He has expertise in Python and has put Ansible in use to manage several thousand servers at work.

SINA is a NASDAQ-listed company and the parent company of Weibo, another NASDAQ-listed company.

Table of Contents

Preface

With most companies moving to the cloud, infrastructure needs are growing exponentially. The growing data and massive computing power required to store, analyze, and process this data adds to the infrastructure needs. With the endlessly increasing number of Internet service users and the enormous inflow of data accompanied by a race for data mining, big data and cloud services have opened up new data centers and expanded upon the existing ones. Also, with constantly scaling infrastructure and increasing demands, with the 99.9% uptime promises to keep, automated management of infrastructure became the need of the hour. DevOps soon became a necessity and the market has flooded with DevOps tools. Ansible is one such open source solution that combines orchestration, configuration management, and application deployment capabilities in one.

Ansible is an IT automation tool that lets you manage your Infrastructure as Code. It helps you deploy your applications and manage configurations, thus making life easier. It is an open source project built on Python and has great community support. Ansible, in most ways, is sufficient to address most of your requirements. With a number of modules and plugins available, Ansible makes everything look so easy. Writing and understanding playbooks is smooth.

This book is aimed at advanced users who already have a working knowledge of Ansible, we will discuss various extension points that are exposed by Ansible and how they can be exploited to fit our requirements. This book covers in detail the Ansible Python API, Ansible modules, and Ansible plugins. In this book—by means of real-life scenarios—demonstrates how Ansible can be extended to meet your requirements. This will take you through a step-by-step process of how you can fill in the gaps and become a master of Ansible.

What this book covers

Chapter 1, *Getting Started with Ansible*, is an introductory chapter that introduces you to Ansible and encourages you to become a power user. It introduces you to the Ansible architecture and gives you a reason to chose Ansible as an infrastructure and configuration management tool.

Chapter 2, *Getting to Know Ansible Modules*, covers the basics of writing an Ansible module. It introduces you to the AnsibleModule boilerplate. This chapter also helps you develop sample Ansible modules in Bash and Python.

Chapter 3, *Digging Deeper into Ansible Modules*, introduces you to handling arguments in an Ansible Module. It also takes you through a scenario of collecting information about the infrastructure by developing a custom Ansible module.

Chapter 4, *Exploring API*, covers in detail the Python API for Ansible, running Ansible programmatically, and discusses the various extension points provided by Ansible. It discusses topics such as Plugin Loader, Runner, Playbooks, and Callbacks in depth.

Chapter 5, *An In-Depth Look at Ansible Plugins*, covers different plugins from the code level. It also demonstrates how you can write your own Ansible plugin through a few examples.

Chapter 6, *Fitting It All Together – Integration*, covers various configuration options for Ansible, thus allowing the user to get the most out of the tool. This chapter introduces you to Ansible Galaxy, a platform for sharing roles. It takes the reader through the process of contributing to Ansible and distributing their modules and plugins.

Chapter 7, *Becoming a Master – A Complete Configuration Guide*, contains real-life scenarios where one can exploit the power of Ansible to perform the required tasks. Also contains scenarios where one can go a step ahead from using Ansible as an Infrastructure and configuration management tool.

What you need for this book

To get the most out of this book, you need the following:

- Linux distribution (Fedora/Ubuntu)
- Ansible
- Python

Who this book is for

This book is perfect for developers and administrators who are familiar with Ansible and Python programming but have no knowledge of how to customize Ansible.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "A path specified by the `library` variable in the configuration file, located at `/etc/ansible/ansible.cfg`."

A block of code is set as follows:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Any command-line input or output is written as follows:

```
# cp /usr/src/asterisk-addons/configs/cdr_mysql.conf.sample
    /etc/asterisk/cdr_mysql.conf
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "By using the **Add a Role** option from the menu and supplying the required credentials, Galaxy will import the role from your GitHub repository and make it available on the Galaxy platform for the entire community."

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—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 book'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 book, 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 book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

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 book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

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

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 books—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 book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, 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 book 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 book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Getting Started with Ansible

As technology has advanced, computing has become more and more complex. With better hardware being manufactured each day, the complexity of computing systems has increased. Distributed computing started flourishing, and soon "the cloud" was invented. Software became trivial and managing it became a pain. Development cycles picked up the pace, and manual testing and deployments soon felt outdated, hence calling for automation. If you are reading this book, you probably understand the importance of automation, be it for testing an application or managing the whole infrastructure.

With increasing load and ever-scaling infrastructure, system administrators have stopped being simple craftspeople by configuring each system manually and have begun to manage thousands of systems at once. For any environment, however big it is, you need a reliable system to manage it all. The geographically scattered workplaces and ever-growing infrastructure make it nearly impossible to keep track of the inventory and manually configure and manage each machine. The paced development cycles and reduced time to market leaves little margin for error and throws the manual process out the window.

The key to managing the whole infrastructure, deploying builds, speeding up the process, and at the same time keeping track of the changes is to have a system that is user-friendly, has a small learning curve, and is pluggable as per your requirements. What's most important is that you stay focused and spend more time managing your infrastructure and the processes rather than the automation scripts and management tool itself. Out of the many available solutions, Ansible is one such tool with many interesting features. It is easy to extend and works out of the box for 90% of user requirements. This book focuses on the remaining 10%.

In this chapter, we will be exploring:

- Why Ansible?
- Why extend Ansible?
- The Ansible architecture
- Extending Ansible

Why Ansible?

Out of the many available tools in the market, how do you choose which tool best fits your need? What factors should you consider while choosing a tool to satisfy your requirements? Questions may come to mind such as:

- What is the **return on investment** (**ROI**) in terms of money, time, and effort?
- What kind of support do I get with the tool?
- What are the potential associated security risks?
- Is the tool flexible enough to be plugged into my infrastructure?
- What is the coverage? Are all my requirements addressed?

If these are the questions that come to mind, I'll try answering them in favor of Ansible.

- Ansible is free. The only investment you need is some time and effort. Ansible playbooks are YAML-based and hence are very easy to read, understand, and maintain, thus involving a very small learning curve. Modules hide the complexity underneath.
- Ansible is open source. Hence, there is an entire community to back you up. You can file in issues or even fix them yourself, since you will always have access to the code.
- Unlike other solutions, which are mostly agent-based, Ansible works purely on SSH. There is no agent required. Therefore, you can sit back and relax, as there is no extra package lying on your production system.
- Ansible provides a very good API, which you can use to build your own Ansible modules that suit your needs and can then be plugged into your infrastructure.
- Ansible provides 90% of user requirements out of the box. The remaining 10% has a well-documented API and community support to build your own modules, hence increasing the coverage.

If you are satisfied by the above arguments and willing to give Ansible a try, read on.

Why extend Ansible?

Ansible comes in handy in various contexts – as a configuration management tool and deployment automation tool, as well as for provisioning and orchestration. It comes out of the box with a lot of plugins and modules that can be used for building playbooks. You can manage your entire infrastructure using Ansible in the way most software development projects do. **Infrastructure as Code** (**IAC**) applies the same principles of software development to configuration management.

People like Ansible for its simplicity and clear separation of concerns. It doesn't force you to adhere to one particular vision of how you should manage your configurations. It provides a perfect building block for designing your IAC solution, tailored to your specific requirements.

There can be many reasons to extend Ansible. This might range from adding missing features to modifying/enhancing existing features as per your own needs. With Ansible being an open source, community driven project, not everything can be integrated at once. There is always a trade-off between utility and demand. If there are not many users of one particular feature, it becomes an overhead for the project maintainer to support it.

Need something new

So, you come across a situation where Ansible, in its native form with the available modules and plugins, is not enough to meet your requirements. What do you do? Change the tool? Look for other options? Maybe even curse your bad luck for not being able to foresee what was coming and now you need to change everything?

Well, the answer is NO. Ansible provides a very good API and boilerplate that you can use to write your own Ansible modules or plugins as per your requirements. Building Ansible modules is easy. Since Ansible is community driven, you might even file a feature request for the required module if you feel more people are likely to face the same issue you encountered. If you are a developer, you can simply write your own Ansible module or plugin and share it with the community. Send in a pull request for your module and get into a discussion with the project maintainers. Hopefully, the module will be merged and made available in future releases of Ansible.

In this book, we will see how to extend Ansible as per the requirements and distribute the customizations by contributing to an open source project, specifically, Ansible.

Company-wide abstraction

Treating your infrastructure as code offers many advantages, but it comes with a cost. Not all members of your team will be willing to climb the learning curve. As a result, only a few people will become powerful users of any configuration management tool such as Ansible, and they will become the bottleneck for the whole team.

A good IAC implementation should make it easy for everyone to interact with the infrastructure, deploy new software, provision resources, and weave components together. Details should be abstracted away as much as possible, behavior should be clear, and definitions should be navigable. There should also exist an easy way to trace any problems back to a high-level configuration.

To achieve this, one can develop plugins and modules that can abstract the details and provide interfaces that people can directly use and get results from. This will help everyone get up to speed and interact with the infrastructure.

You can create modules and plugins that can make your routine tasks easy. You can share these as utilities that can be used by anyone in the company to carry out similar tasks. This would require some developer efforts, but would enable even the not so powerful users to get the most out of their infrastructure.

Diving into Ansible

Infrastructure grows gradually to a point where you finally give up managing it manually and begin to feel the need for a better way to manage the emergent complexity.

One way to do this is to spend a lot of time looking for the right tool, then you end up adopting a complete configuration management solution, and bend over backwards to change your problem in order to make it fit into the existing solution. Obviously, this approach sounds flawed.

Another approach is to keep it simple and incrementally exploit the power of existing tools when they actually give you an immediate advantage.

Ansible is more suited for the latter approach. It is well-written and offers a clear separation of concerns and a simple model. In fact, you can choose to what degree you want to engage with it. It allows you to reuse components provided by the community while remaining in control.

You can exploit the various extension points exposed by Ansible to build modules and plugins that suit your needs. Reusing already existing plugins and modules and creating your own as and when required provides even more control over your infrastructure.

Contributing to Ansible

Ansible is an open source project hosted on GitHub. If you have a GitHub account, you can easily fork the Ansible repository and start contributing to the project (Ansible code: `https://github.com/ansible/ansible`).

You can fork the project in your own account, clone it, and then make changes and send out pull requests to the project owner. This applies to all the open source projects.

If you don't know where to start contributing, you can also look at the *Issues* section in the repository. The *Issues* section contains bug reports and feature requests from people using the tool. You might choose to verify and fix the issues and then send in your patch to the project owner as a pull request against an issue.

The patches go through a review process, and only after the project maintainer's approval, the patch will be merged. Once merged, the feature will then be available to the users.

Ansible architecture

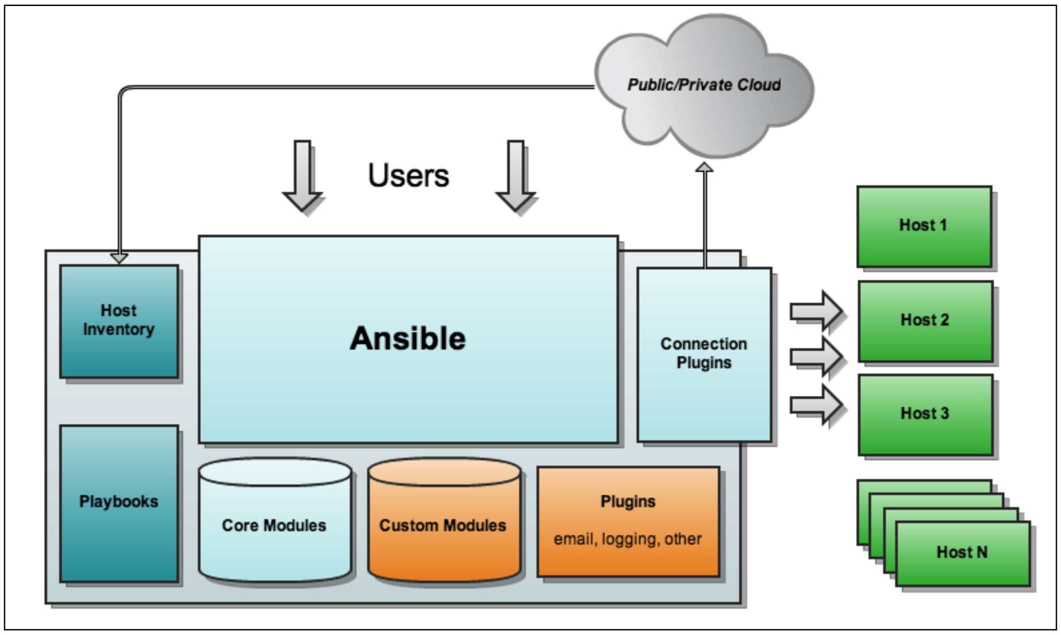
Even though we assume our readers have a working knowledge of Ansible, it is useful to run through a brief overview of the Ansible architecture, so as to have a better understanding of the various extension points.

Ansible is an agentless configuration management system, meaning no special software has to run on the managed hosts. Ansible connects to its targets usually via plain SSH, copies all the necessary code, and runs it on the target machine. Being agentless is one of the main advantages of Ansible over other solutions. This reduces the overhead of the setup of agents required on the target machines, also reducing security risks, as no extra packages or agents need to be installed.

The core Ansible components include:

- **Inventory**: Target
- **Variables**: Information about the target hosts
- **Connection**: How to talk to the target hosts
- **Runner**: Connect to the target and execute actions
- **Playbook**: Recipe to be executed on the target host
- **Facts**: Dynamic information about the target
- **Modules**: Code that implements actions
- **Callback**: Collects the results of the playbook actions

The following figure shows the architecture of Ansible:



Brief overview of Ansible components

Let's have a closer look at the Ansible components.

Ansible runner

At the heart of Ansible is the **runner**. The runner allows you to execute actions on one or more hosts and gather results.

The runner uses an inventory to choose which hosts to connect to. An inventory may also associate a set of variables with each host. These variables can then be accessed through the playbook and by other Ansible components like the connection plugin.

Connection plugins

Connection plugins (with a default SSH connection) can use specific host variables to figure out how to connect to the remote host. Variables may include information like a username to be used to connect to the remote host, a non-default port number, and so on.

Playbook

Moving on to another component, the **playbook** is one of the most important, as all the recipes are written in the form of Ansible playbooks. Playbooks are modeled as a collection of plays, each of which defines a set of tasks to be executed on a group of remote hosts. A play also defines the environment where the tasks will be executed.

Roles

Playbook can be broken down into **roles** for better organization. Roles help in modularizing the playbook tasks. These roles can later be included in the play against specific host groups. For instance, if your infrastructure involves web servers and proxy servers, each requiring a common set of tasks (preparing the systems) and then type-specific tasks (setting up and configuring web/proxy servers), these can be simply broken down into roles, which can later be run against specific hosts. Common tasks can be defined against all hosts, at which time webserver and proxy server roles can then be executed against respective host groups.

Variables

Another important component in Ansible architecture is **variables**. Variables can be used to extract common values and parameterize shared playbook fragments. They can also be used to categorize hosts based on some quality they share.

Facts

Since every host can give out a lot of information about itself, managing them manually is not a recommended practice. Hence, Ansible included a special variable called **facts** in its software.

The facts variable is provided by the setup module and gets implicitly executed on every host (unless explicitly disabled). This variable collects information about the remote host before the runner starts the execution of the playbook on the remote hosts.

Runner

Now that we have the Ansible playbook in place and all facts about the remote host group have been collected, the runner kicks in. The runner variable executes the specific actions (as specified in the Ansible playbook) on the remote hosts by copying the action code to the target machine and preparing the environment before executing the action code.

Once the runner evaluates and executes the tasks, it cleans up the copied code from the remote host, finally reporting the status through **callbacks**.

Playbook expressiveness

The expressiveness of the playbook language is limited in order to promote a somewhat declarative and descriptive structure of your configuration. However, Ansible does not go overboard in trying to model a strictly declarative configuration. Ansible plays are modeled as a sequential execution of tasks, affected only by variables.

There are several tricks that allow you to insert complex logic within the playbooks, as well as some extension points, which we will see later, that allow you to achieve what you desire.

Extending Ansible

Ansible provides various extension points that can be used to extend Ansible and fit it to customize your needs. It has four main entry points where you can put in your code:

- **Custom fact scripts**: gathers custom facts from remote hosts
- **Ansible modules**: actuators of actual infrastructure changes
- **Plugins**: extends the Ansible execution life cycle
- **Python API**: inverts the control and exploits parts of Ansible from your custom tools

Custom fact scripts

Dynamic inventories may provide some knowledge about the infrastructure and how it's grouped and managed, but it does not provide a view of the actual state of things.

Before every Ansible run, facts are gathered about the infrastructure against which the playbook is executed. This collects a lot of information about the hosts and can be later used in the Ansible playbook itself, if required.

However, you may find yourself in a position where the default facts gathered as part of the fact-gathering process are not enough. To tackle this, Ansible allows you to run your custom code as part of the fact-gathering phase, right before the Ansible play execution.

Modules

Modules define the primitive operations that can be performed on your infrastructure. They allow you to exactly describe what to do right from the playbook. They can encapsulate a complex high-level task, such as interacting with some external infrastructure component, and deploy a virtual machine or whole environment.

Modules are the key to Ansible customization. Modules can be written in any programming language, and if suitable, they can use Ansible itself to perform the nitty-gritty details of their operation.

A substantial part of this book is devoted to building Ansible modules.

Plugins

The term **plugin** groups a number of extension points that hook deeply in the Ansible core and extend its behavior in powerful ways.

The currently available plugins for Ansible are as follows:

- Action plugins
- Loopback plugins
- Callback plugins
- Connection plugins
- Filter plugins
- Vars plugins

Plugins will be covered in detail in *Chapter 4*, *Exploring API* and *Chapter 5*, *An In-depth Look at Ansible Plugins*, where you'll learn all you need to know about plugins, including how you can implement them and build your own plugin.

Python API

The Ansible Python API allows you to use Ansible as a library, thus making use of the things that Ansible is good for right from your custom configuration management solution (whatever it is). You can run Ansible playbooks programmatically.

The Python API can also be used from within other Ansible extensions; we'll highlight the important parts throughout this book.

Summary

After going through this chapter, you might be tempted to use Ansible as a configuration management and orchestration tool. Perhaps we have also given you a reason to choose Ansible as an IAC solution. This chapter provided you with a brief introduction to Ansible and its capabilities and use cases. It familiarized you with the Ansible architecture, the different components of Ansible, and the various extension points provided by Ansible. This chapter also took you through the process of contributing to an Ansible project.

In the next chapter, you will be learning about Ansible modules. The chapter will take you through what you need to know before you start writing an Ansible module and guide you through writing your first one. The chapter will also teach you about some best practices that should be followed while developing an Ansible module. Additionally, the chapter will create a base for the more advanced topics that will be covered later in the book, which includes real-life scenarios of where and how you can exploit the power of Ansible.

4
Exploring API

Ansible plugins are an advanced topic. There are various plugins available for Ansible. This chapter will cover different Python API and lookup plugins in brief and explore how they fit into the general Ansible architecture.

Ansible is pluggable in a lot of ways. It is possible that there are components of business logic that don't quite fit in. Hence, Ansible provides extension points that can be used to fit your business needs. Ansible plugins are another such extension point where you can build your own plugins to extend Ansible to address your business logic.

Python API

Before exploring plugins, it's important to understand the Ansible Python API. The Ansible Python API can be used for the following:

- To control the nodes
- To respond to the various Python events
- To write various plugins as per the requirement
- Inventory data from various external data stores can also be plugged in

Python API for Ansible allows Ansible to run programmatically. Running Ansible programmatically through the Python API has the following advantages:

- **Better error handling**: Since everything is Python, it becomes easy to handle errors as and when they occur. This gives more control and confidence in the code by providing a better context in case of errors.
- **Extending Ansible**: One of the drawbacks, as you might have noticed in the previous runs, is that, by default, Ansible simply writes the output on `stdout` and does not log anything to a file. To address this, you can write your own custom plugins to save output to a file or database for future reference.

- **Unknown variables**: There may be cases where complete knowledge of the required variables may be discovered only during runtime, for example, when an IP of an instance launched on the cloud during the Ansible play. Running Ansible programmatically using the Python API can address this issue.

Now that you know the advantages of using Python API for Ansible, let's explore the Python API and take a look at how one can interact with Ansible through the API.

This chapter will cover the three most important classes that are used extensively:

- **Runner**: Used to execute individual modules
- **Playbook**: Helps in executing the Ansible playbook
- **Callbacks**: Gets back the run results on the controller node

Let's take an in-depth look at what these classes are and explore the various extension points.

Runner

The `runner` class is the core API interface of Ansible. The `runner` class is used to execute individual modules. If there is one single module that needs to be executed, for example, the `setup` module, we can use the `runner` class to execute this module.

One can have multiple `runner` objects in the same Python file to run different modules.

Let's explore a sample code, where the `runner` class will be used to execute the `setup` module on localhost. This will print a lot of details about localhost such as time, operating system (distribution), IP, Netmask, and hardware details such as architecture, free memory, used memory, machine ID, and so on.

```
from ansible import runner

runner = runner.Runner(
    module_name = 'setup',
    transport = 'local'
)

print runner.run()
```

This will execute the `setup` module on localhost. This is equivalent to running the following command:

```
ansible all -i "localhost," -c local -m setup
```

To run the preceding module on remote machines or a group of machines, one can specify hosts in an inventory that can later be passed as an argument in the `runner` object, along with a remote user that should be used to log into the remote machine. You can also specify a pattern of hosts, specifically on which the module needs to be executed. This is done by passing the pattern argument to the `runner` object.

You can also pass in a module argument using the `module_args` key.

For instance, if you need to get memory details of remote machines that have their domain names set as `store1.mytestlab.com`, `store2.mytestlab.com`, `store12.mytestlab.com`, and so on, this can be simply achieved in the following manner:

```
from ansible import runner

runner = runner.Runner(
    module_name = 'setup',
    pattern = 'store*',
    module_args = 'filter=ansible_memory_mb'
)

print runner.run()
```

The preceding code will execute the `setup` module on all twelve hosts and print the memory status that is reachable by each host. Reachable hosts will be listed under "contacted," while those that are un-reachable will be listed under "dark."

Apart from the arguments discussed above, the `runner` class provides a large number of interfacing options through the arguments that it accepts. The following is a list of a few arguments as defined in the source code, along with their use:

Arguments/default values	**Description**
`host_list=C.DEFAULT_HOST_LIST`	Example: `/etc/ansible/hosts`, legacy usage
`module_path=None`	Example: `/usr/share/ansible`
`module_name=C.DEFAULT_MODULE_NAME`	Example: `copy`
`module_args=C.DEFAULT_MODULE_ARGS`	Example: "`src=/tmp/a dest=/tmp/b`"
`forks=C.DEFAULT_FORKS`	Parallelism level
`timeout=C.DEFAULT_TIMEOUT`	SSH timeout
`pattern=C.DEFAULT_PATTERN`	Which hosts? Example: "all" `acme.example.org`

Arguments/default values	**Description**
`remote_user=C.DEFAULT_REMOTE_USER`	Example: "`username`"
`remote_pass=C.DEFAULT_REMOTE_PASS`	Example: "`password123`" or "`None`" if using key
`remote_port=None`	If SSH on different ports
`private_key_file=C.DEFAULT_PRIVATE_KEY_FILE`	If not using keys/passwords
`transport=C.DEFAULT_TRANSPORT`	"`SSH`," "`paramiko`," "`Local`"
`conditional=True`	Run only if this fact expression evals to `true`
`callbacks=None`	Used for output
`sudo=False`	Whether to run sudo or not
`inventory=None`	Reference to inventory object
`environment=None`	Environment variables (as `dict`) to use inside the command
`complex_args=None`	Structured data in addition to `module_args`, must be a `dict`

Playbook

Playbook, as you have learned in the previous chapters, is a set of instructions or commands in a YAML format that runs in a sequential order. Python API for Ansible provides a rich interface to run the already created playbooks through the `PlayBook` class.

You can create a `PlayBook` object and pass in an existing Ansible playbook as an argument along with the required parameters. One thing to note is that multiple plays do not execute simultaneously, but the tasks in a play can be executed in parallel based on the requested number of forks. Once the object is created, you can easily execute the Ansible playbook by calling the `run` function.

You can create a `Playbook` object that can later be executed using the following template:

```
pb = PlayBook(
    playbook = '/path/to/playbook.yaml',
    host_list = '/path/to/inventory/file',
    stats = 'object/of/AggregateStats',
    callbacks = 'playbookCallbacks object',
    runner_callbacks = 'callbacks/used/for/Runner()'
)
```

One thing to note here is that a `PlayBook` object requires at least four mandatory arguments to be passed. These are:

- `playbook`: the path to a Playbook file
- `stats`: Holds aggregated data about events occurring in each host
- `callbacks`: Outputs callbacks for the playbook
- `runner_callbacks`: Callbacks for the `runner` API

You can also define the verbosity in a range of `0-4`, which is required by the `callbacks` and `runner_callbacks` objects. If verbosity is not defined, the default value is taken as `0`. Defining verbosity as a `4` is equivalent to using `-vvvv` while executing the Ansible playbook from the command line.

For instance, you have your inventory file named `hosts` and a playbook named `webservers.yaml`. To execute this playbook on the inventory hosts using the Python API, you need to create a `PlayBook` object with the required parameters. You also need to require a verbose output. This can be done as follows:

```
from ansible.playbook import PlayBook
from ansible import callbacks
VERBOSITY = 4
pb = PlayBook(
    playbook = 'webservers.yaml',
    host_list = 'hosts',
    stats = callbacks.AggregateStats(),
    callbacks = callbacks.PlaybookCallbacks(verbose=VERBOSITY),
    runner_callbacks = callbacks.PlaybookRunnerCallbacks(
                        callbacks.AggregateStats(),
                        verbose=VERBOSITY)
)

pb.run()
```

This will execute the playbook `webservers.yaml` on the remote hosts specified in the `hosts` inventory file.

To execute the same playbook locally, just as you did in the `runner` object earlier, you need to pass the argument `transport=local` in the `PlayBook` object and remove the `host_list` argument.

Apart from the discussed parameters, PlayBook accepts a whole lot more.

The following is a list of all the arguments accepted by the `PlayBook` object along with their purpose:

Argument	**Description**
`playbook`	Path to a playbook file
`host_list`	Path to a file like `/etc/ansible/hosts`
`module_path`	Path to Ansible modules, like `/usr/share/ansible/`
`forks`	Desired level of parallelism
`timeout`	Connection timeout
`remote_user`	Run as this user if not specified in a particular play
`remote_pass`	Use this remote password (for all plays) vs using SSH keys
`sudo_pass`	If `sudo=true` and a password is required, this is the sudo password
`remote_port`	Default remote port to use if not specified with the host or play
`transport`	How to connect to hosts that don't specify a transport (local, paramiko, and so on.)
`callbacks`	Output callbacks for the playbook
`runner_callbacks`	More callbacks, this time for the runner API
`stats`	Holds aggregate data about events occurring to each host
`sudo`	If not specified per play, requests all plays use `sudo` mode
`inventory`	Can be specified instead of `host_list` to use a pre-existing inventory object
`check`	Don't change anything; just try to detect some potential changes
`any_errors_fatal`	Terminate the entire execution immediately when one of the hosts has failed
`force_handlers`	Continue to notify and run handlers even if a task fails

Callbacks

Ansible provides hooks for running custom callbacks on the host machine as it invokes various modules. Callbacks allow us to log the events and operations that are started or completed and aggregate results from the module execution. Python API provides callbacks for this purpose, which can be used in its default state as well as to develop your own callback plugins.

Callbacks allow various operations to be performed. Callbacks can also be exploited as an extension point for Ansible. Some of the most widely used callbacks operations while wrapping up Ansible in Python API are:

- `AggregateStats`: As the name suggests, `AggregateStats` holds the aggregated stats surrounding per host activity during a playbook run. An object of `AggregateStats` can be passed on as an argument for `stats` in the `PlayBook` object.
- `PlaybookRunnerCallbacks`: An object of `PlaybookRunnerCallbacks` is used for `Runner()`, for example, when a single module is executed using the `Runner` API interface, `PlaybookRunnerCallbacks` is used to return the task status.
- `PlaybookCallbacks`: An object of `PlaybookCallbacks` is used by the playbook API interface of the Python API when a playbook is executed from the Python API. These callbacks are used by `/usr/bin/ansible-playbook`.
- `DefaultRunnerCallbacks`: When there are no callbacks specified for `Runner` to use, `DefaultRunnerCallbacks` is used.
- `CliRunnerCallbacks`: This extends `DefaultRunnerCallbacks` and overrides the on-event trigger functions, basically optimized to be used with `/usr/bin/ansible`.

Ansible plugins

Plugins are another extension point that haven't yet been touched on in this book. Also, there is very limited documentation available, even on the Internet, regarding plugins.

Plugins are an advanced topic that will be covered in the next chapter. However, it's important to understand the Python API behind plugins in order to understand how plugins work and how they can be extended.

PluginLoader

As the code documentation states, `PluginLoader` is the base class that loads plugins from the configured plugin directories. It iterates through the list of play basedirs, configured paths, and Python paths to search for a plugin. The first match is used.

An object of `PluginLoader` takes in the following arguments:

- `class_name`: The specific class name for plugin type
- `required_base_class`: The base class required by the plugin module
- `package`: Package information
- `config`: Specifies the default path from configuration
- `subdir`: All subdirectories in a package
- `aliases`: Alternate name for the plugin type

For every Ansible plugin, there is a defined class name that needs to be used. This class in `PluginLoader` is identified by `required_base_class`. The different categories of Ansible plugins along with their base names are listed in the following table:

Plugin type	**Class name**
Action plugins	`ActionModule`
Cache plugins	`CacheModule`
Callback plugins	`CallbackModule`
Connection plugins	`Connection`
Shell plugins	`ShellModule`
Lookup plugins	`LookupModule`
Vars plugins	`VarsModule`
Filter plugins	`FilterModule`
Test plugins	`TestModule`
Strategy plugins	`StrategyModule`

Summary

This chapter took you through the Python API for Ansible and introduced you to more advanced ways of using Ansible. This included executing single tasks without creating an entire playbook to executing a playbook programmatically.

This chapter also introduced you to the various components of the Ansible Python API from a more technical point of view, exploring the various extension points and ways to exploit them.

This chapter additionally sets a base for the next chapter, which will be a deep dive into the Ansible plugins. The next chapter will utilize the knowledge gained from this chapter to create custom Ansible plugins. We will explore different Ansible plugins and guide you through writing your own Ansible plugin in the following chapter.