Extending Ansible

Ansible is an IT-automation tool that lets you manage your infrastructure as a code. It helps you deploy your applications and manage configurations, thus making life easier. Ansible, in most ways, is self-sufficient to address most of your requirements.

This book shows you how to automate most of the tasks in your IT environment, thus minimizing the need to manually perform scheduled tasks and extend your Ansible implementation by developing customized modules and plugins. You'll begin by getting an understanding of the fundamental aspects of extending the Ansible framework, such as custom functions and reusable modules. You'll then progress to building custom plugins and extensions using the fundamental building blocks explained earlier. We'll also show you how to automate some tasks using scripts. Finally, we'll demonstrate how these extensions can be seamlessly integrated into existing Ansible installations and explain how to conduct unit testing on these extensions to ensure they work as desired.

Who this book is written 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.

What you will learn from this book
- Get a thorough understanding of Ansible modules
- Find out everything about plugins that fit into the Ansible architecture
- Get to grips with designing modules and handling errors
- Work with data structures
- Distribute Ansible extensions using PyPi and Git submodules
- Get to know the various distribution methods of modules and plugins

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

Rishabh Das

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Getting Started with Ansible'
- A synopsis of the book’s content
- More information on Extending Ansible
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.
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 choose 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.
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.
Where to buy this book

You can buy Extending Ansible 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.