Ansible Playbook Essentials

This book will show you how to write a blueprint of your infrastructure, encompassing multitier applications using Ansible’s playbooks. Beginning with basic concepts such as plays, tasks, handlers, inventory, and the YAML syntax that Ansible uses, you will see how to organize your code into a modular structure. Building on this, you will master techniques to create data-driven playbooks with variables, templates, logical constructs, and encrypted data. This book will also take you through advanced clustering concepts, such as discovering topology information about other nodes in the cluster, managing multiple environments with isolated configurations, and orchestrating infrastructure and deploying applications in a coordinated manner.

By the end of this book, you will be able to design solutions to your automation and orchestration problems using playbooks quickly and efficiently.

Who this book is written for

If you are a systems or automation engineer who intends to automate common infrastructure tasks, deploy applications, and use orchestration to configure systems in a coordinated manner, then this book is for you. Some understanding of the Linux/Unix command-line interface is expected.

What you will learn from this book

- Write simple tasks and plays
- Organize code into a reusable, modular structure
- Separate code from data using variables and Jinja2 templates
- Run custom commands and scripts using Ansible’s command modules
- Control execution flow based on conditionals
- Integrate nodes and discover topology information about other nodes in the cluster
- Encrypt data with Ansible-vault
- Create environments with isolated configurations to match the application development workflow
- Orchestrate infrastructure and deploy applications in a coordinated manner

Design automation blueprints using Ansible’s playbooks to orchestrate and manage your multitier infrastructure.
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Blueprinting Your Infrastructure'
- A synopsis of the book’s content
- More information on Ansible Playbook Essentials
Gourav Shah (www.gouravshah.com) has extensive experience in building and managing highly available, automated, fault-tolerant infrastructure and scaling it. He started his career as a passionate Linux and open source enthusiast, transformed himself into an operations engineer, and evolved to be a cloud and DevOps expert and trainer.

In his previous avatar, Gourav headed IT operations for Efficient Frontier (now Adobe), India. He founded Initcron Systems (www.initcron.com), a niche consulting firm that specializes in DevOps enablement and cloud infrastructure management. Under the Initcron banner, he helps companies design DevOps strategies and helps implement them. Initcron also provides automation services for start-ups, as well as large enterprises.

Gourav is a coveted corporate trainer and has trained professionals from top IT companies around the world, such as IBM, CISCO, Qualcomm, Accenture, Dreamworks, Walmart Labs, Intuit, and Wells Fargo, to name a few. He runs a specialized training school by the name School of Devops (www.schoolofdevops.com) to groom world class cloud and DevOps professionals.
With the evolution of cloud computing, agile development methodologies, and explosion of data in recent years, there is a growing need to manage infrastructure at scale. DevOps tools and practices have become a necessity to automate every stage of such a scalable, dynamic, and complex infrastructure. Configuration management tools are at the core of this devops tool set.

Ansible is a simple, efficient, fast configuration management, orchestration and application deployment tool all combined into one. This book helps you to get familiar with writing Playbooks, Ansible's automation language. This book follows a hands-on approach to show you how to create flexible, dynamic, reusable, and data-driven roles. It then takes you through the advanced features of Ansible, such as node discovery, clustering, securing data with vault, and managing environments, and finally shows you how to use Ansible to orchestrate multitier infrastructure stack.

What this book covers

Chapter 1, Blueprinting Your Infrastructure, will introduced you to Playbooks, YAML, and so on. You will also learn about the components of a playbook.

Chapter 2, Going Modular with Ansible Roles, will demonstrate creating modular reusable automation code using Ansible roles, which are units of automation.

Chapter 3, Separating Code and Data – Variables, Facts, and Templates, covers the creation of flexible, customizable, data-driven roles with templates and variables. You will also learn about auto-discovered variables, that is, facts.

Chapter 4, Bringing In Your Code – Custom Commands and Scripts, covers bringing in your existing scripts and invoking Shell commands with Ansible.

Chapter 5, Controlling Execution Flow – Conditionals, discusses the control structures offered by Ansible to change the direction of execution.
Chapter 6, *Iterative Control Structures – Loops*, demonstrates how to iterate over arrays, hashes, and so on, using omnipotent with statements.

Chapter 7, *Node Discovery and Clustering*, discusses the discovery of topology information and creates dynamic configurations using magic variables and facts caching.

Chapter 8, *Encrypting Data with Vault*, discusses secure variables stored in, and shared with, version control systems using Ansible-vault.

Chapter 9, *Managing Environments*, covers the creation and management of isolated environments with Ansible and mapping automation code to the software development workflow.

Chapter 10, *Orchestrating Infrastructure with Ansible*, covers the orchestration features of Ansible, such as rolling updates, pre-tasks and post-tasks, tags, building tests into the playbook, and so on.
Blueprinting Your Infrastructure

This book is a primer for anyone who has conceptual knowledge of Ansible and would like to get started writing Ansible playbooks to automate common infrastructure tasks, orchestrate application deployments, and/or manage configurations across multiple environments. This book follows an incremental approach, starting with the basics such as learning about the anatomy of a playbook and writing simple roles to create modular code. Once comfortable with the basics, you will be introduced to primitives such as adding dynamic data with variables and templates, and controlling execution flow with conditionals and iterators. This is then followed by more advanced topics such as node discovery, clustering, encrypting data, and managing environments. We conclude with the discussion on the orchestration features of Ansible. Let's begin our journey towards being an Ansible practitioner by learning about playbooks.

In this chapter, we will learn about:

- The anatomy of a playbook
- What plays are and how to write a Hosts inventory and search patterns
- Ansible modules and the batteries-included approach

Getting introduced to Ansible

Ansible is a simple, flexible, and extremely powerful tool that gives you the ability to automate common infrastructure tasks, run ad hoc commands, and deploy multitier applications spanning multiple machines. Even though you can use Ansible to launch commands on a number of hosts in parallel, the real power lies in managing those using playbooks.
Blueprinting Your Infrastructure

As systems engineer, infrastructure that we typically need to automate contains complex multitier applications. Each of which represents a class of servers, for example, load balancers, web servers, database servers, caching applications, and middleware queues. Since many of these applications have to work in tandem to provide a service, there is topology involved as well. For example, a load balancer would connect to web servers, which in turn read/write to a database and connect to the caching server to fetch in-memory objects. Most of the time, when we launch such application stacks, we need to configure these components in a very specific order.

Here is an example of a very common three-tier web application running a load balancer, a web server, and a database backend:

Ansible lets you translate this diagram into a blueprint, which defines your infrastructure policies. The format used to specify such policies is what playbooks are.

Example policies and the sequence in which those are to be applied is shown in the following steps:

1. Install, configure, and start the MySQL service on the database servers.
2. Install and configure the web servers that run Nginx with PHP bindings.
3. Deploy a Wordpress application on the web servers and add respective configurations to Nginx.
4. Start the Nginx service on all web servers after deploying Wordpress. Finally, install, configure, and start the haproxy service on the load balancer hosts. Update haproxy configurations with the hostnames of all the web servers created earlier.

The following is a sample playbook that translates the infrastructure blueprint into policies enforceable by Ansible:

```
---
- hosts: lb
  remote_user: root
  roles:
    - haproxy

- hosts: www
  remote_user: vagrant
  sudo: yes
  roles:
    - nginx
    - php
    - wordpress

- hosts: db
  remote_user: root
  roles:
    - mysql
    - mysqladmin
```

### Plays

A playbook consists of one or more plays, which map groups of hosts to well-defined tasks. The preceding example contains three plays, each to configure one layer in the multitiered web application. Plays also define the order in which tasks are configured. This allows us to orchestrate multitier deployments. For example, configure the load balancers only after starting the web servers, or perform two-phase deployment where the first phase only adds this configurations and the second phase starts the services in the desired order.
YAML – the playbook language

As you may have already noticed, the playbook that we wrote previously resembles more of a text configuration than a code snippet. This is because the creators of Ansible chose to use a simple, human-readable, and familiar YAML format to blueprint the infrastructure. This adds to Ansible's appeal, as users of this tool need not learn any special programming language to get started with. Ansible code is self-explanatory and self-documenting in nature. A quick crash course on YAML should suffice to understand the basic syntax. Here is what you need to know about YAML to get started with your first playbook:

- The first line of a playbook should begin with "--- " (three hyphens) which indicates the beginning of the YAML document.
- Lists in YAML are represented with a hyphen followed by a white space. A playbook contains a list of plays; they are represented with "- ". Each play is an associative array, a dictionary, or a map in terms of key-value pairs.
- Indentations are important. All members of a list should be at the same indentation level.
- Each play can contain key-value pairs separated by ":" to denote hosts, variables, roles, tasks, and so on.

Our first playbook

Equipped with the basic rules explained previously and assuming readers have done a quick dive into YAML fundamentals, we will now begin writing our first playbook. Our problem statement includes the following:

1. Create a devops user on all hosts. This user should be part of the devops group.
2. Install the "htop" utility. Htop is an improved version of top—an interactive system process monitor.
3. Add the Nginx repository to the web servers and start it as a service.

Now, we will create our first playbook and save it as simple_playbook.yml containing the following code:

```yaml
---
- hosts: all
  remote_user: vagrant
  sudo: yes
  tasks:
```
Our playbook contains two plays. Each play consists of the following two important parts:

- **What to configure**: We need to configure a host or group of hosts to run the play against. Also, we need to include useful connection information, such as which user to connect as, whether to use `sudo` command, and so on.
- **What to run**: This includes the specification of tasks to be run, including which system components to modify and which state they should be in, for example, installed, started, or latest. This could be represented with tasks and later on, by roles.

Let's now look at each of these briefly.
Creating a host inventory

Before we even start writing our playbook with Ansible, we need to define an inventory of all hosts that need to be configured, and make it available for Ansible to use. Later, we will start running plays against a selection of hosts from this inventory. If you have an existing inventory, such as cobbler, LDAP, a CMDB software, or wish to pull it from a cloud provider, such as ec2, it can be pulled from Ansible using the concept of a dynamic inventory.

For text-based local inventory, the default location is /etc/ansible/hosts. For our learning environment, however, we will create a custom inventory file customhosts in our working directory, the contents of which are shown as follows. You are free to create your own inventory file:

```
#customhosts
#inventory configs for my cluster
[db]
192.168.61.11 ansible_ssh_user=vagrant

[www]
www-01.example.com ansible_ssh_user=ubuntu
www-02 ansible_ssh_user=ubuntu

[lb]
lb0.example.com
```

Now, when our playbook maps a play to the group, the www (hosts: www), hosts in that group will be configured. The all keywords will match to all hosts from the inventory.

The following are the guidelines for creating inventory files:

- Inventory files follow INI style configurations, which essentially include configuration blocks that start with host group/class names included in "[ ]."
  This allows the selective execution on classes of systems, for example, [namenodes].
- A single host can be part of multiple groups. In such cases, host variables from both the groups will get merged, and the precedence rules apply.
  We will discuss variables and precedence in detail later.
• Each group contains a list of hosts and connection details, such as the SSH user to connect as, the SSH port number if non-default, SSH credentials/keys, sudo credentials, and so on. Hostnames can also contain globs, ranges, and more, to make it easy to include multiple hosts of the same type, which follow some naming patterns.

After creating an inventory of the hosts, it's a good idea to validate connectivity using Ansible's ping module (for example, ansible -m ping all).

Patterns
In the preceding playbook, the following lines decide which hosts to select to run a specific play:

- hosts: all
- hosts: www

The first code will match all hosts, and the second code will match hosts which are part of the www group.

Patterns can be any of the following or their combinations:

<table>
<thead>
<tr>
<th>Pattern Types</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group name</td>
<td>namenodes</td>
</tr>
<tr>
<td>Match all</td>
<td>all or *</td>
</tr>
<tr>
<td>Range</td>
<td>namenode[0:100]</td>
</tr>
<tr>
<td>Hostnames/hostname globs</td>
<td>*.example.com, host01.example.com</td>
</tr>
<tr>
<td>Exclusions</td>
<td>namenodes:!secondarnamenodes</td>
</tr>
<tr>
<td>Intersection</td>
<td>namenodes:&amp;zookeeper</td>
</tr>
<tr>
<td>Regular expressions</td>
<td>~(nn</td>
</tr>
</tbody>
</table>
Tasks

Plays map hosts to tasks. Tasks are a sequence of actions performed against a group of hosts that match the pattern specified in a play. Each play typically contains multiple tasks that are run serially on each machine that matches the pattern. For example, take a look at the following code snippet:

```yaml
- group:
    name: devops
    state: present
- name: create devops user with admin privileges
  user:
    name: devops
    comment: "Devops User"
    uid: 2001
    group: devops
```

In the preceding example, we have two tasks. The first one is to create a group, and second is to create a user and add it to the group created earlier. If you notice, there is an additional line in the second task, which starts with `name:`. While writing tasks, it's good to provide a name with a human-readable description of what this task is going to achieve. If not, the action string will be printed instead.

Each action in a task list can be declared by specifying the following:

- The name of the module
- Optionally, the state of the system component being managed
- The optional parameters

With newer versions of Ansible (0.8 onwards), writing an action keyword is now optional. We can directly provide the name of the module instead. So, both of these lines will have a similar action, that is, installing a package with the `apt` module:

```yaml
action: apt name=htop state=present update_cache=yes
apt: name=nginx state=latest
```

Ansible stands out from other configuration management tools, with its batteries-included approach. These batteries are "modules." It's important to understand what modules are before we proceed.
Modules

Modules are the encapsulated procedures that are responsible for managing specific system components on specific platforms.

Consider the following example:

- The `apt` module for Debian and the `yum` module for RedHat helps manage system packages
- The `user` module is responsible for adding, removing, or modifying users on the system
- The `service` module will start/stop system services

Modules abstract the actual implementation from users. They expose a declarative syntax that accepts a list of the parameters and states of the system components being managed. All this can be declared using the human-readable YAML syntax, using key-value pairs.

In terms of functionality, modules resemble providers for those of you who are familiar with Chef/Puppet software. Instead of writing procedures to create a user, with Ansible we declare which state our component should be in, that is, which user to create, its state, and its characteristics, such as UID, group, shell, and so on. The actual procedures are inherently known to Ansible via modules, and are executed in the background.

The `Command` and `Shell` modules are special ones. They neither take key-value pairs as parameters, nor are idempotent.

Ansible comes preinstalled with a library of modules, which ranges from the ones which manage basic system resources to more sophisticated ones that send notifications, perform cloud integrations, and so on. If you want to provision an EC2 instance, create a database on the remote PostgreSQL server, and get notifications on IRC, then Ansible has a module for it. Isn't this amazing?

No need to worry about finding an external plugin, or struggle to integrate with cloud providers, and so on. To find a list of modules available, you can refer to the Ansible documentation at http://docs.ansible.com/list_of_all_modules.html.

Ansible is extendible too. If you do not find a module that does the job for you, it's easy to write one, and it doesn't have to be in Python. A module can be written for Ansible in the language of your choice. This is discussed in detail at http://docs.ansible.com/developing_modules.html.
The modules and idempotence

Idempotence is an important characteristic of a module. It is something which can be applied on your system multiple times, and will return deterministic results. It has built-in intelligence. For instance, we have a task that uses the `apt` module to install Nginx and ensure that it's up to date. Here is what happens if you run it multiple times:

- Every time idempotence is run multiple times, the `apt` module will compare what has been declared in the playbook versus the current state of that package on the system. The first time it runs, Ansible will determine that Nginx is not installed, and will go ahead with the installation.
- For every consequent run, it will skip the installation part, unless there is a new version of the package available in the upstream repositories.

This allows executing the same task multiple times without resulting in the error state. Most of the Ansible modules are idempotent, except for the command and shell modules. Users will have to make these modules idempotent.

Running the playbook

Ansible comes with the `ansible-playbook` command to launch a playbook with. Let's now run the plays we created:

```bash
$ ansible-playbook simple_playbook.yml -i customhosts
```

Here is what happens when you run the preceding command:

- The `ansible-playbook` parameter is the command that takes the playbook as an argument (`simple_playbook.yml`) and runs the plays against the hosts
- The `simple_playbook` parameter contains the two plays that we created: one for common tasks, and the other for installing Nginx
- The `customhosts` parameter is our host's inventory, which lets Ansible know which hosts, or groups of hosts, to call plays against

Launching the preceding command will start calling plays, orchestrating in the sequence that we described in the playbook. Here is the output of the preceding command:
Let's now analyze what happened:

- Ansible reads the playbooks specified as an argument to the `ansible-playbook` command and starts executing plays in the serial order.

- The first play that we declared, runs against the "all" hosts. The `all` keyword is a special pattern that will match all hosts (similar to `*`). So, the tasks in the first play will be executed on all hosts in the inventory we passed as an argument.

- Before running any of the tasks, Ansible will gather information about the systems that it is going to configure. This information is collected in the form of facts.

- The first play includes the creation of the `devops` group and user, and installation of the htop package. Since we have three hosts in our inventory, we see one line per host being printed, which indicates whether there was a change in the state of the entity being managed. If the state was not changed, "ok" will be printed.
Blueprinting Your Infrastructure

• Ansible then moves to the next play. This is executed only on one host, as we have specified "hosts: www" in our play, and our inventory contains a single host in the group "www".

• During the second play, the Nginx repository is added, the package is installed, and the service is started.

• Finally, Ansible prints the summary of the playbook run in the "PLAY RECAP" section. It indicates how many modifications were made, if any of the hosts were unreachable, or execution failed on any of the systems.

What if a host is unresponsive, or fails to run tasks? Ansible has built-in intelligence, which will identify such issues and take the failed host out of rotation. It will not affect the execution on other hosts.

Review questions
Do you think you've understood the chapter well enough? Try answering the following questions to test your understanding:

1. What is idempotence when it comes to modules?
2. What is the host's inventory and why is it required?
3. Playbooks map ___ to ___ (fill in the blanks)
4. What types of patterns can you use while selecting a list of hosts to run plays against?
5. Where is the actual procedure to execute an action on a specific platform defined?
6. Why is it said that Ansible comes with batteries included?

Summary
In this chapter, you learned about what Ansible playbooks are, what components those are made up of, and how to blueprint your infrastructure with it. We also did a primer on YAML—the language used to create plays. You learned about how plays map tasks to hosts, how to create a host inventory, how to filter hosts with patterns, and how to use modules to perform actions on our systems. We then created a simple playbook as a proof of concept.

In the upcoming chapter, we will start refactoring our code to create reusable and modular chunks of code, and call them roles.
Where to buy this book

You can buy Ansible Playbook Essentials 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.