# Introduction

All of you working in the software industry must have heard of this term **Ansible**, but why there's so much hype for this term? How much do you know about Ansible? Is it a tool or a framework or simply a practice? In this **Ansible tutorial for beginners**, we will dive deep into the main concepts of Ansible and we will understand the need for Ansible, its workflow, and many more.

**Watch this DevOps Tutorial video on YouTube:**


Here, we have the list of topics covered in this Ansible basics tutorial:



First, let's understand 'What is Ansible?'

# What is Ansible?

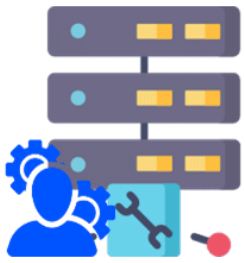
Ansible is an open-source platform used for automation and for various operations such as configuration management, application deployment, task automation, and IT orchestration. Ansible is easy to set up, and it is efficient, reliable, and powerful. It runs on Linux, Mac, or BSD. Apart from the free version, it has an enterprise edition called 'Ansible Tower.'

The hectic tasks such as configuration management, task automation, etc. were previously carried out through some traditional practices, so why do we need Ansible?

We will discuss it as we move along this DevOps Tutorial.

***Prepare yourself for interviews using these Top DevOps Interview Questions and Answers!***

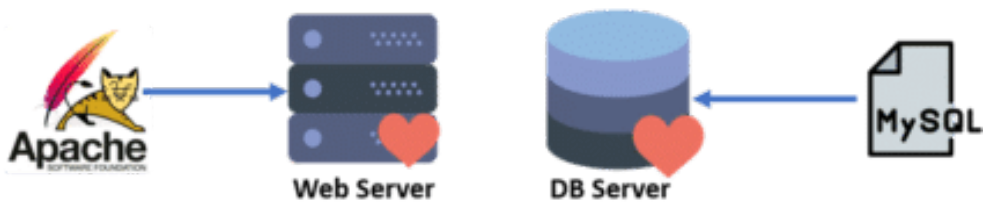
## Why do we need Ansible?



Before understanding Ansible as a whole, we must understand the need for Ansible and what the problems were that we faced previously.

Let's consider a case where you as a System Admin is responsible for handling a company's infrastructure. Suppose, there are nine servers, out of which five are acting as web servers and the rest four as database servers.

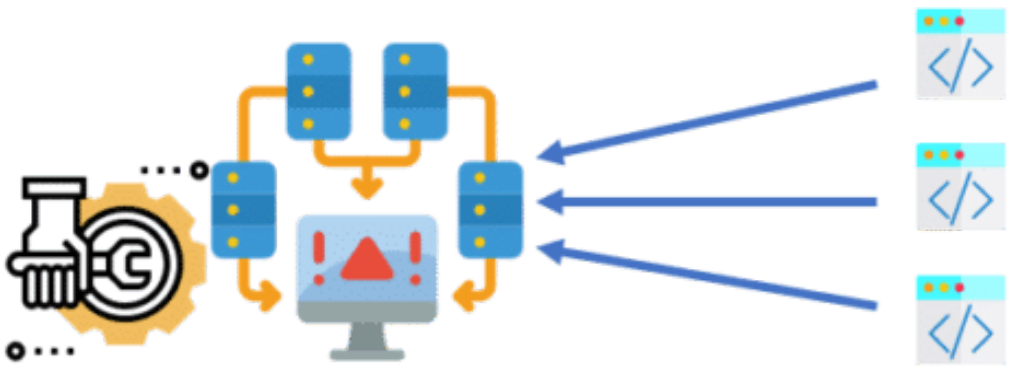
You want to install Tomcat on web servers and MySQL on database servers. In the traditional method, you will have to manage the servers manually, install the required software, and change the configurations, along with administering the services on each server individually.




When there are only a few services and configuration requirements, it will be easy for you to handle these amounts of servers; even if there is a minimal increment in the services, you can still handle the situation by provisioning a few more servers to maintain the infrastructure.

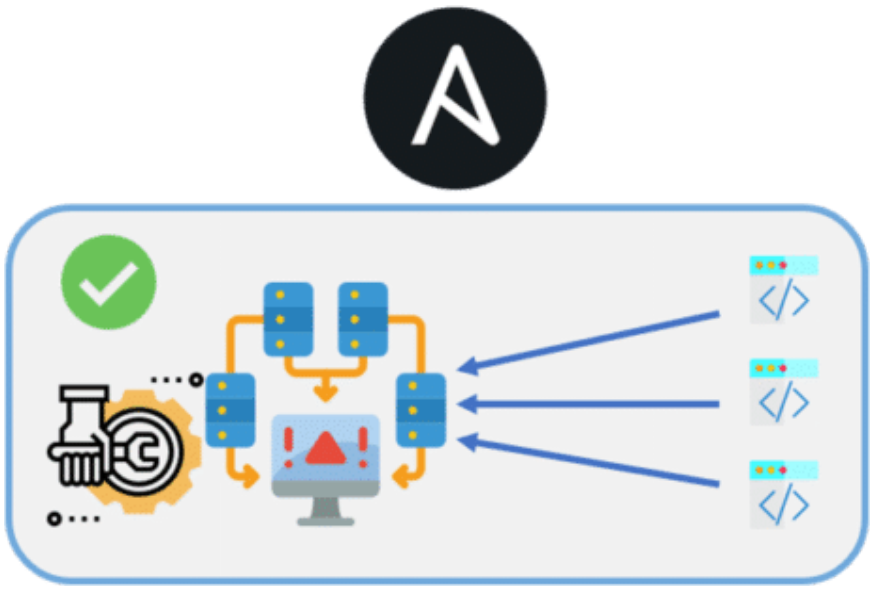
As we know, research and inventions never stop; technology keeps on enhancing. New features and services are introduced every other day, and the hosted applications become more complex with the increased amount of services.

As the number of services starts to increase, the same task must be repeated multiple times. As sysadmins provision more servers, it gets more difficult for them to set up, update, and maintain all these servers manually.



In most cases, sysadmins won't be able to set up each server identically. Also, such processes would end up hampering the velocity of the work for developers as the development team is agile and is always releasing software frequently. On the other hand, the sysadmins would have to spend extra time on system configurations and managing the infrastructure.

This is where Ansible comes to the rescue!



How? We will discuss this as we move ahead in this Ansible tutorial.

Before we proceed, first, let's understand a few important terminologies in Ansible so that we can understand the workflow in a better way.

*Become a master of DevOps by enrolling in this online [DevOps Training in London](#)!*

## Important Terms in Ansible

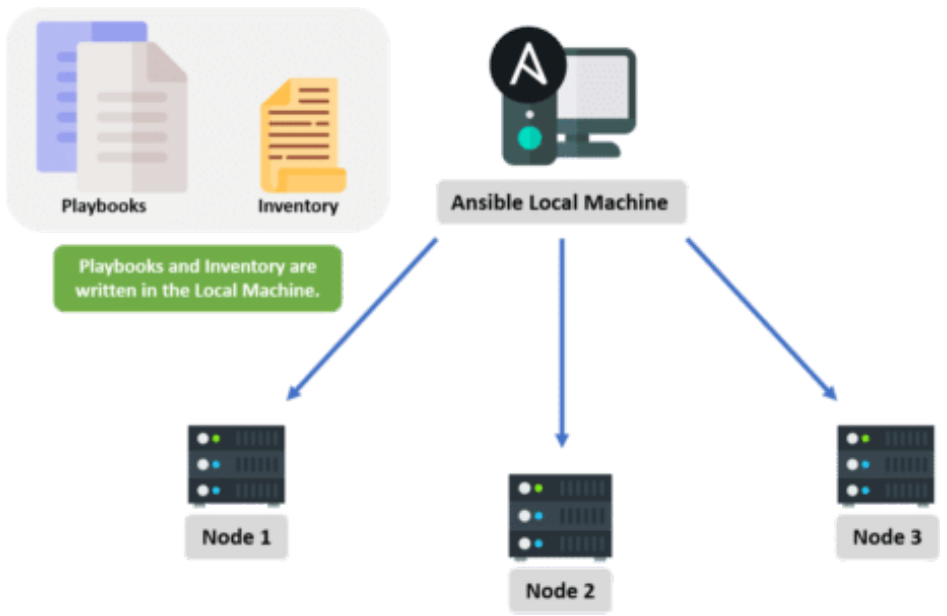
- **Controller Machine:** This is where Ansible gets installed. The controller machine helps in enabling provisioning on servers we manage.
- **Inventory:** This is basically an initializing file that contains information about the servers that we are managing.
- **Playbook:** It is an organized unit of scripts defining an automated work for the configuration management of our server.
- **Task:** A task block defines a single procedure to be executed on the server like installing packages.

Now that we understand the important terms involved in Ansible, let's move forward and understand the workflow of Ansible.

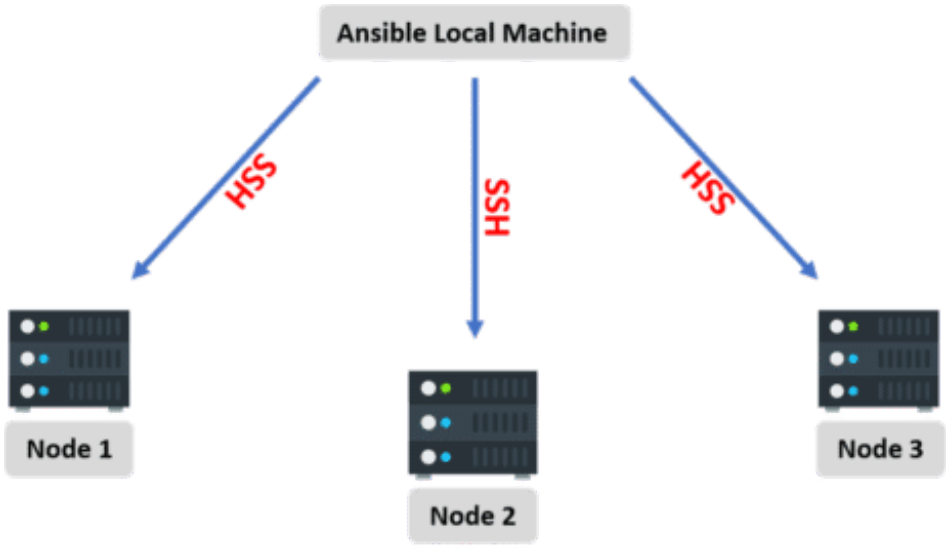


## Ansible Workflow

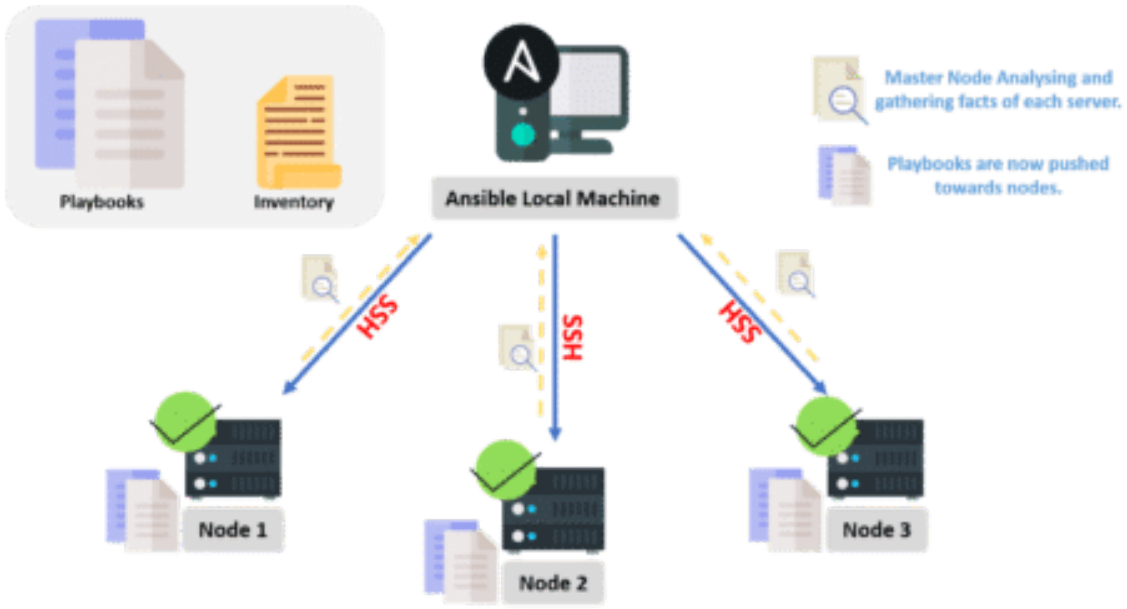
As we discussed above, when services increase, sysadmins will provision more servers to do configuration management. They need not do it manually anymore but install Ansible on the master node where they need to write the code into the Ansible **playbook** to describe the setup, installation process, and the configuration required for these servers.




The local machine connects to these servers (nodes) through an **inventory** using secured **SSH** connections.




Once these nodes are connected to the master server, then the node servers are analyzed and the playbook codes are pushed toward each of the servers so that these playbooks can configure the servers remotely, which leads to a consistent environment.

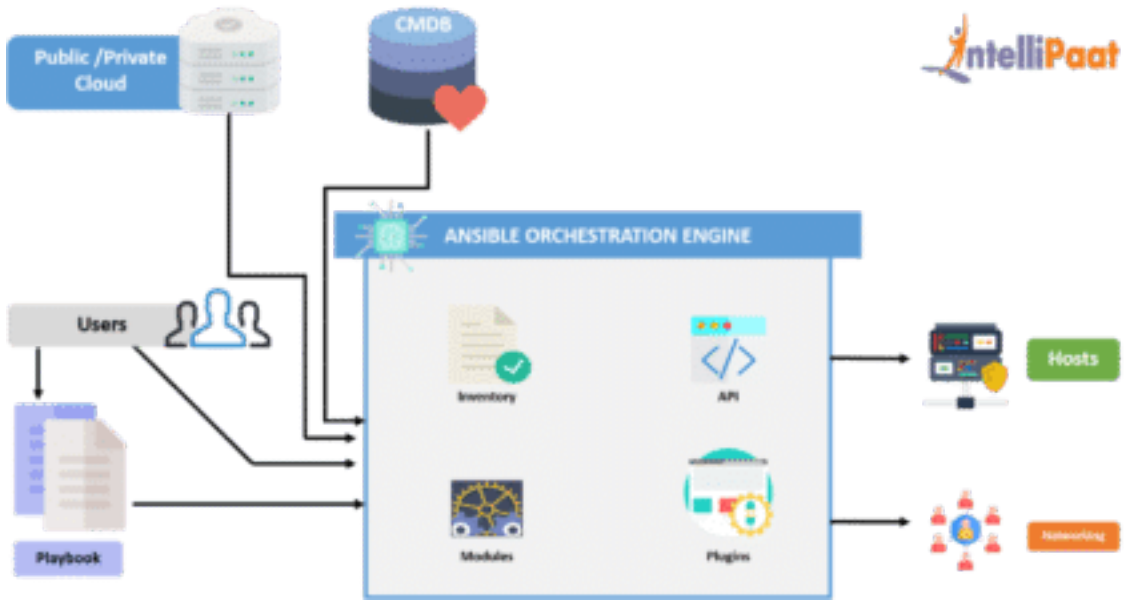



Now, let's try to understand its architecture.

*Go for the Best DevOps Course in New York to get a clear understanding of DevOps!*

## Ansible Architecture

From the diagram below it is clear that Ansible Orchestration Engine interacts with a user who is writing the Ansible playbook to execute Ansible Orchestration Engine, along with interacting with the services of public/private cloud and configuration management database.




## Modules

Ansible connects the nodes and pushes out small 'Ansible Modules.' Modules are executed by Ansible and then get removed when finished. These modules can reside on any machine; no servers or daemons or databases are required here. We can work with the text editor of our choice or a terminal and a version control system to keep track of the changes made in our content.

## Plugins

A plugin is a piece of code that expands the core functionality of Ansible. There are plenty of handy plugins, and we can write our own plugins as well.

## Inventory

We already discussed this in the Ansible Terminologies section. An inventory is a list of hosts/nodes, having IP addresses, servers, databases, etc., which need to be managed.

## Playbooks

As discussed earlier, we write our code in a playbook. It is simple and written in the YAML format and basically describes tasks that are supposed to be executed through Ansible. We can launch tasks synchronously or asynchronously with playbooks.

## APIs

Ansible APIs work as transport for cloud services, either public or private.



## Hosts

Hosts are basically the node systems in the Ansible architecture getting automated by Ansible only. They can be any type of machine such as Windows, Red Hat, Linux, etc.

## Networking

We can use Ansible to automate different networks. It uses the easy, simple, yet powerful, agentless automation framework for IT operations and development. It uses a type of data model (playbook or role), which is separated from the Ansible Automation Engine that spans across different network hardware quite easily.

## CMDB

CMDB is a type of repository that acts as a data warehouse for the IT installations.

## Cloud

A network of remote servers on which we can store, manage, and process our data. These servers are hosted on the Internet. For storing the data remotely rather than on local servers, we would just launch our resources and instances on the cloud and connect them to our servers, and we would have the wisdom of operating our task remotely.

Now that we understand the workflow of Ansible, along with its architecture, let's sneak a peek over some of its benefits.

***Learn more about DevOps from this DevOps Training in Sydney to get ahead in your career!***

# Benefits of Using Ansible

- **Agentless:** As long as a connection can be SSHed and it has Python, it can be configured with Ansible; no agent/software or additional firewall ports are required to install on our client or host systems for automation. Also, we don't have to worry about setting up and managing the infrastructure.
- **Simple:** As we've seen, Ansible uses a very simple syntax written in YAML known as playbooks—YAML (Yet Another Markup Language) is a human-readable data serialization language. We don't need special coding skills to code and understand playbooks. It is very easy to install and execute tasks in order.
- **Modular:** Ansible is modular as we require only one program per script. This way, we can spread our programs across different servers.
- **Efficient:** Not requiring any extra software on our servers means that there is more space for our resources.
- **Powerful and flexible:** Having powerful features gives us the capability to model even complex IT workflows in lesser time, along with managing infrastructure, networks, operating systems, and services that are already in use.

In the next section of this Ansible online tutorial, we also need to check the operations that can be performed by Ansible.

***If you have doubts or queries related to DevOps, get them clarified from DevOps experts on our DevOps Community!***

# Operations by Ansible

So far, we have figured out how reliable and efficient Ansible is. Now, we will learn about the type of operations Ansible can perform.

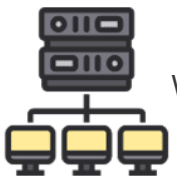
## Configuration Management

Ansible provides stability in the performance of our product by recording and updating stats in detail. This describes all information about the hardware and software of an enterprise/organization, such as versions and updates applied to the installed software packages, along with the locations and network addresses of hardware devices.

For example, suppose, Company A wants to install the new version of Nginx on all of its server machines. It won't be feasible for the company to update each and every machine manually. Hence, it just installs Nginx on one of its machines and deploys it across the rest of the machines using Ansible playbooks.

***Also, look into the Ansible Cheat Sheet by Intellipaat.***

## Resource/Server Provisioning



Whether we are booting/starting servers/virtual machines or creating cloud instances from various templates, Ansible is always there to help us with the smooth running of the process. Ansible makes sure to provide the required packages installed on our application.

## Application Deployment

With Ansible, we can define our application and manage its deployment. Instead of performing the deployment steps one by one, we just need to install Ansible on our machine and it will do the same tasks for us, even in lesser time. Provided those tasks are listed in our Ansible playbook, Ansible executes them in order.

## Security and Compliance

With Ansible, we can easily configure our security details. We do it once in a control machine and the same security details will be spread across all the other nodes.

## Orchestration

We can also perform the orchestration of our application using Ansible. Ansible provides orchestration by aligning the business requests with the application, data, and infrastructure. It creates an application-aligned infrastructure that can be scaled up or down based on the needs.

Now that we understand 'What is Ansible?,' its workflow, its terminologies, its architecture, and the different operations performed by Ansible, let's head on to set up its environment.

***Interested in becoming a DevOps expert? Learn more from this DevOps Course in Toronto!***

# Setting up an Ansible Environment

## Setting up Ansible on CentOS

To set up Ansible on CentOS, first, we need to set up the EPEL repository. **Extra Packages for Enterprise Linux (EPEL)** is a free repository project from Fedora. It is open-source and provides high-quality add-on software packages for Linux distributions, including Red Hat and CentOS.

Since Ansible package is not available for default yum repositories, we need to enable the EPEL repository on our machine by using the following command:

```
sudo rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

After this, all the necessary packages to install Ansible will be downloaded.

As our EPEL repo is added, with the help of the below command, we will install Ansible:

```
yum install ansible -y
```

It will complete the installation process in no time.

Now, in order to check the version of Ansible, we will use the command:

```
ansible –version
```

After installation, we need to add the servers on which we will execute management tasks using Ansible.

For that, we need to create another CentOS VM that would act as our node machine.

First, we will set up the password-less SSH authentication for our nodes on our Ansible control machine. Then, we will generate an SSH key on the control machine and type the below command:

```
ssh-keygen
```

Once our public key is generated, we need to check the IP addresses of our nodes, which we have to specify in the Ansible Inventory later. We will use the ifconfig command for this purpose.

After the public key of our Ansible server is generated, we have to copy it to the nodes, using this command:

```
ssh-copy-id -i root@<IP address of our node machine>
```

Now, we can use any editor to write our inventory and name our servers as per our choice. Here, we are naming our server as **intellipaat**.

If we want to perform a simple ping operation to test the connectivity using Ansible, we can use the command below:

```
ansible -m ping ‘intellipaat’
```

This would show our IP address and a ‘success’ message on the screen.

## Setting up Ansible on an Ubuntu Machine

First, we need to configure PPA on our machine. For that, we will type the following command on our terminal:

```
$ sudo apt-get update
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update
$ sudo apt-get install ansible
```

We will be able to manage the remote machines using Ansible now. We will check the version using the command:

```
ansible –version
```

This is how we set up an Ansible environment.

***Interested in getting an industry-recognized certification in DevOps? Enroll in Intellipaat's DevOps Course in Bangalore now!***

Are you interested in learning Ansible commands? Further in this Ansible tutorial for beginners, we will learn Ansible ad-hoc commands and Ansible playbook commands. Read on!

## Ad-hoc Commands

Ad-hoc commands are those that need to be typed in order to do it quickly but are not saved for later. In case we need to reboot all our servers, then we will have to run the ad-hoc commands from **/usr/bin/ansible**.

These commands are one-time-usage commands and are not used for configuration management and deployment. Only Ansible playbook can be used for configuration management and deployment.

### Parallelism and Shell Commands

For example, using Ansible's command-line tool to reboot a company's server in 10 parallel forks at a time, we would need to set up ssh-agent to establish a connection.

```
$ ssh-agent bash
$ ssh-add ~/.ssh/id_rsa
```

In order to run, we would have to reboot all company servers in a group named 'test-servers' in 10 parallel forks:

```
$ Ansible test-servers -a "/sbin/reboot" -f 10
```

Ansible will be running these ad-hoc commands from the current user account by default. In case we want to change its behavior, we need to pass the username in the command as shown below:

```
$ Ansible test-servers -a "/sbin/reboot" -f 10 -u username
```

### File Transfer

Ad-hoc commands can also be used for file transfer for doing **SCP (Secure Copy Protocol)** on numerous files in a parallel fashion on multiple machines.
To transfer files to multiple servers/machines, we have to use the command below:

```
$ Ansible test-servers -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"
```

In order to create a new directory, we will use the command as follows:

```
$ Ansible test-servers -m file -a "dest = /path/user1/new mode = 777 owner = user1 group = user1 state
```

To delete the whole directory and files, we use the command:

```
$ Ansible test-servers -m file -a "dest = /path/user1/new state = absent"
```

### Managing Packages

Following are some of the ad-hoc commands using yum that check if the yum package is installed or not (but not for updating it)

```
$ Ansible test-servers -m yum -a "name = demo-tomcat-1 state = present"
```

To check if the package is installed or not:

```
$ Ansible test-servers -m yum -a "name = demo-tomcat-1 state = absent"

$ Ansible test-servers -m yum -a "name = demo-tomcat-1 state = latest"
```

We can also use ad-hoc commands to gather facts by implementing conditional statements in our playbook. We can find information through this command:

```
$ Ansible all -m setup
```

***DevOps Engineers are among the highest paid professionals in the technology domain. Join DevOps Training in Hydera bad today!***

Now, let's understand the structure of Ansible playbook and its basic syntax.

## Ansible Playbooks Commands

As we have discussed earlier, Ansible code is written in playbooks in the YAML (Yet Another Markup Language) format. YAML is a script type language; we can use any editor to write YAML files, but notepad++ will be the easiest one.

Let's now check out the basic syntax of the playbook with an example. A typical YAML file starts with three hyphens (—).

```
---
name: install & configure Intellipaat DB
hosts: testServers
become: yes

vars:
oracle_db_port_value : 1521
tasks:
-name: Install the Oracle DB
yum: <code to install the DB>
-name: Ensure the installed service is enabled and running
service:
name: <our service name>
```

We can save the above file as first .yml. We need to follow the correct indentation while writing in YAML and be a little careful about its syntax.

- **Name:** It specifies the name of our Ansible playbook. Based on what this particular playbook will do, we can give it a relevant name.
- **Host:** It specifies the list of hosts or host groups against which we will run our task. This host tag is compulsory as it communicates with Ansible and enables it to run the listed tasks. These tasks can be run either on the same machine or on remote machines, or we can run the tasks on multiple machines thereby hosts tags can enable the entry of a group of hosts.
- **Vars:** Using this tag, we can easily define the variable we have used in our playbook; its usage is similar to the variables in any of the programming languages.
- **Tasks:** Playbooks must contain tasks or a list of tasks that are to be executed. They are basically a list of actions that the playbook needs to perform. *The task field contains the name of the task.* Every task links to a piece of code called a module. The module and the arguments (in the module), both should be executed.

In the next section of this Ansible basics tutorial, we will check out the use cases of Ansible to learn Ansible in-depth.

***Get in touch with Intellipaat for comprehensive DevOps Training and be a certified DevOps Engineer!***

## Ansible Use Case

Ansible has the potential to change the landscape of orchestration by seamlessly uniting it with configuration management, provisioning, and deployment of applications, and all of it on this one easy-to-use platform!

This simple yet powerful platform enables us to solve our most challenging and complex problems and that is the reason why it comes out on top among its peers and is adored and supported by a large community.

Ansible's popularity isn't just limited to the said community, but it is also being used by some of the major IT organizations such as **Arista**.

Arista Networks is a company that provides software-driven cloud networking solutions for cloud architectures, i.e., large data center storage and computing environments, and helps them achieve better economy and agility.

Arista uses Ansible, bringing NetOps and DevOps together. Not just that, it helps the company get the most out of Arista switches by integrating with Ansible and bringing Ansible's strength, simplicity, and agility to its network.

There are various benefits from the combination of Ansible and Arista solutions as follows:

- There is improved accuracy and speed in the combined solutions.
- The integration of Ansible with Arista eliminates the need to use third-party EOS extensions.
- Playbooks allow making dynamic and intelligent decisions.
- Additional switch configuration is not required.
- Arista gets complete control over its configuration.
- There is continuous EOS configuration compliance.

We might wonder how exactly Ansible is able to offer such significant benefits to this company? How Arista is able to utilize the power of Ansible to draw out these benefits?

The following is all about how Ansible is aiding Arista:

- Ansible uses CLI or eAPI to communicate with Arista switches.
- The CLI connection securely manages the devices using SSH.
- Ansible uses the existing TACACS users to provide built-in authorization.



## Conclusion

Server systems are the backbones and foundation of applications. They too need to be version controlled, tested, and automated as we do with the applications. Nowadays, configuration management is nothing new; tools to perform configuration management have been around for quite some time now. Puppet, Chef, etc. are some of the names that pop up in our minds when we talk about configuration management.

Ansible is somewhat similar to these tools and yet it seems that more companies are interested in Ansible these days. The reason behind this fact is thoroughly elucidated in this tutorial, along with almost all the other must-know Ansible concepts. However, there is much more to know about Ansible that you can learn in-depth by enrolling in this **DevOps Certification Course** offered by Intellipaat.