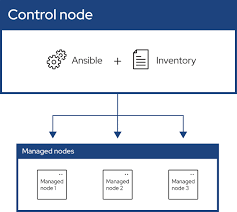
**ANSIBLE**

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

****

**Control node**

A system on which Ansible is installed. You run Ansible commands such as ansible or ansible-inventory on a control node.

**Inventory**

A list of managed nodes that are logically organized. You create an inventory on the control node to describe host deployments to Ansible.

**Managed node**

A remote system, or host, that Ansible controls.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It’s a human-readable data serialization language & is commonly used for configuration files, but could be used in many applications where data is being stored)which is very easy for humans to understand, read and write. Hence the advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed (YAML – It is in human readable form).

Ansible is designed for multi-tier deployment. Ansible does not manage one system at time, it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely agentless which means Ansible works by connecting your nodes through ssh(by default). But if you want other method for connection like Kerberos, Ansible gives that option to you.

After connecting to your nodes, Ansible pushes small programs called as “Ansible Modules”. Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file). Ansible uses the hosts file where one can group the hosts and can control the actions on a specific group in the playbooks.

## What is Configuration Management

Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise’s hardware and software.

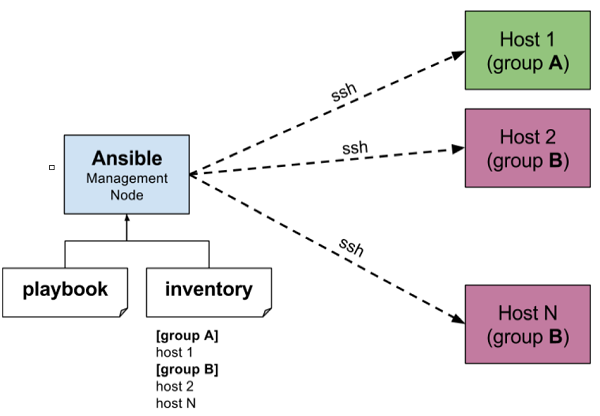
Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of **WebLogic/WebSphere** server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

## How Ansible Works?

The picture given below shows the working of Ansible.

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



The management node in the above picture is the controlling node (managing node) which controls the entire execution of the playbook. It’s the node from which you are running the installation. The inventory file provides the list of hosts where the Ansible modules needs to be run and the management node does a SSH connection and executes the small modules on the hosts machine and installs the product/software.

**Beauty** of Ansible is that it removes the modules once those are installed so effectively it connects to host machine , executes the instructions and if it’s successfully installed removes the code which was copied on the host machine which was executed.

**Why Use Ansible**

Here are some important reasons for using Ansible, such as:

* Ansible is free to use by everyone.
* Ansible is very consistent and lightweight, and no constraints regarding the operating system or underlying hardware are present.
* It is very secure due to its agentless capabilities and open SSH security features.
* Ansible does not need any special system administrator skills to install and use it.
* Its modularity regarding plugins, inventories, modules, and playbooks make Ansible perfect companion orchestrate large environments.

|  |  |
| --- | --- |
| **Terms** | **Explanation** |
| Ansible Server | It is a machine where Ansible is installed and from which all tasks and playbooks will be executed. |
| Modules | The module is a command or set of similar commands which is executed on the client-side. |
| Task | A task is a section which consists of a single procedure to be completed. |
| Role | It is a way of organizing tasks and related files to be later called in a playbook. |
| Fact | The information fetched from the client system from the global variables with the gather facts operation. |
| Inventory | A file containing the data regarding the Ansible client-server. |
| Play | It is the execution of the playbook. |
| Handler | The task is called only if a notifier is present. |
| Notifier | The section attributed to a task which calls a handler if the output is changed. |
| Tag | It is a name set to a task that can be used later on to issue just that specific task or group of jobs. |

# **Ansible Architecture**

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. You can show in the below diagram, such as:

**Inventory**

Inventory is lists of nodes or hosts having their IP addresses, databases, servers, etc. which are need to be managed.

### **API's**

The Ansible API's works as the transport for the public or private cloud services.

### **Modules**

Ansible connected the nodes and spread out the Ansible modules programs. Ansible executes the modules and removed after finished. These modules can reside on any machine; no database or servers are required here. You can work with the chose text editor or a terminal or version control system to keep track of the changes in the content.

### **Plugins**

Plugins is a piece of code that expends the core functionality of Ansible. There are many useful plugins, and you also can write your own.

### **Playbooks**

Playbooks consist of your written code, and they are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.

### **Hosts**

In the Ansible architecture, hosts are the node systems, which are automated by Ansible, and any machine such as RedHat, Linux, Windows, etc.

### **Networking**

Ansible is used to automate different networks, and it uses the simple, secure, and powerful agentless automation framework for IT operations and development. It uses a type of data model which separated from the Ansible automation engine that spans the different hardware quite easily.

### **Cloud**

A cloud is a network of remote servers on which you can store, manage, and process the data. These servers are hosted on the internet and storing the data remotely rather than the local server. It just launches the resources and instances on the cloud, connect them to the servers, and you have good knowledge of operating your tasks remotely.

### **CMDB**

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

# **Ansible Installation in Linux**

Mainly, there are two types of machines when we talk about deployment −

* **Control machine** − Machine from where we can manage other machines.
* **Remote machine** − Machines which are handled/controlled by control machine.

**Note** − Windows does not support control machine.

**Prerequisites**

1. PyYAML: a YAML parser and emitter for the python programming language.
2. Httplib2: a comprehensive HTTP client library.
3. parmiko: native python SSHv2 protocol library.
4. Distro: RHEL/ CentOS/ Debian/ Ubuntu Linux.
5. Jinja2: a modern and designer friendly templating language for python.
6. sshpass: a non-interactive ssh password authentication.
7. **Install Ansible on RedHat/Centos systems**

**Step 1:** Install the EPEL repo

[root@ansible-server ~]# sudo yum install epel-release

**Step 2:** Install the Ansible package.

[root@ansible-server ~]# sudo yum install -y ansible

## Install Ansible on Debian/Ubuntu systems

**Step 1:** First perform an update to the packages

$ sudo apt update

**Step 2:** Then install the software properties common package.

$ sudo apt install software-properties-common

**Step 3:** And install the Ansible personal package archive.

$ sudo apt-add-repository ppa:ansible/ansible

**Step 4:** Install the Ansible.

$ sudo apt update

$ sudo apt install ansible

## Install Ansible using pip

The pip command is a tool for installing and managing python packages.

**Step 1:** This given method works on the Linux and UNIX like systems.

$ sudo pip install ansible

Ansible build script is available in the SlackBuilds.org repository. Also can be built and installed using sbopkg.

**Step 1:** Create queue with Ansible and all dependencies.

# sqg -p ansible

# sbopkg -k -i ansible

**Using swupd (Clear Linux)**

**Step 1:** Ansible and its dependencies are available as part of the sysadmin host management bundle.

$ sudo swupd bundle-add sysadmin-hostmgmt

**Step 2:** And update of the software will be managed by the swupd tool.

$ sudo swupd update

## Installing

**By using yum**

On Fedora:

$ sudo dnf install python-argcomplete

On RHEL and CentOS:

$ sudo yum install epel-release

$ sudo yum install python-argcomplete

**By using apt**

$ sudo apt install python-argcomplete

**By using pip**

$ pip install argcomplete

# **Ansible ad-hoc Commands**

Ad-hoc commands are one of the simplest ways of using Ansible. These are used when you want to issue some commands on a server or bunch of servers. The ad-hoc commands are not stored for future use, but it represents a fast way to interact with the desired servers.

The Ansible ad-hoc command uses the **/usr/bin/ansible** command-line tool to automate a single task on one or more managed nodes. The Ad-hoc commands are quick and easy, but they are not re-usable. The Ad-hoc commands demonstrate the simplicity and power of Ansible.

### **Syntax**

1. ansible <hosts> [-m <module\_name>] -a <"arguments"> -u <username> [--become]

### **Explanation**

**Hosts:** It can be an entry in the inventory file. For specifying all hosts in the inventory, use all or "\*".

**module\_name:** It is an optional parameter. There are hundreds of modules available in the Ansible, such as **shell, yum, apt, file,** and **copy**. By default, it is the **command**.

**Arguments:** We should pass values that are required by the module. It can change according to the module used.

**Username:** It specifies the user account in which Ansible can execute commands.

**Become:** It's an optional parameter specified when we want to run operations that need sudo privilege. By default, it becomes false.

### **1. Parallelism and shell commands**

You can reboot your company server in 12 parallel forks at the same time. For this, you need to set up the SSHagent for connection.

1. $ ssh-agent bash
2. $ ssh-add ~/.ssh/id\_rsa

To run reboot for all your company servers in the group, 'abc', in 12 parallel forks:

1. $ ansible abc -a "/sbin/reboot" -f 12

By default, Ansible will run the above ad-hoc commands from the current user account. If you want to change then pass the username in ad-hoc command as follows:

1. $ ansible abc -a "/sbin/reboot" -f 12 -u username

### **2. File Transfer**

You can use ad-hoc commands for doing SCP (secure copy protocol) which means lots of files in parallel on multiple machines or servers.

**Transferring file on many machines or servers**

1. $ ansible abc -m copy -a "src = /etc/yum.conf dest = /tmp/yum.conf"

**Creating new directory**

1. $ ansible abc -m file -a "dest = /path/user1/new mode = 888 owner = user1 group = user1 state = directory"

**Deleting all directory and files**

1. $ ansible abc -m file -a "dest = /path/user1/new state = absent"

### **3. Managing Packages**

Ad-hoc commands are available for apt and yum module. Here are the following ad-hoc commands using yum.

Below command checks, if the yum package is installed or not, but not update it.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = present"

Below command checks the package is not installed.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = absent"

And below command checks the latest version of package is installed.

1. $ ansible abc -m yum -a "name = demo-tomcat-1 state = latest"

### **4. Managing Users and Groups**

You can manage, create, and remove a user account on your managed nodes with ad-hoc commands.

1. $ ansible all -m user -a "name=foo password=<crypted password here>"
3. $ ansible all -m user -a "name=foo state=absent"

### **5. Managing Services**

Ensure a service is started on all the webservers.

1. ansible webservers -m service -a "name=httpd state=started"

Alternatively, restart a service on all webservers:

1. $ ansible webservers -m service -a "name=httpd state=restarted"

Ensure a service is stopped:

1. $ ansible webservers -m service -a "name=httpd state=stopped"

### **6. Gathering Facts**

1. Fact represents the discovered variables about a system. You can use the facts to implement conditional execution of tasks, and also used to get ad-hoc information about your systems. To see all the facts:  
   $ ansible all -m setup

# **Ansible Playbooks**

Playbooks are the files where the Ansible code is written. Playbooks are written in YAML format. **YAML** means "Yet Another Markup Language," so there is not much syntax needed. **Playbooks** are one of the core features of Ansible and tell Ansible what to execute, and it is used in complex scenarios. They offer increased flexibility.

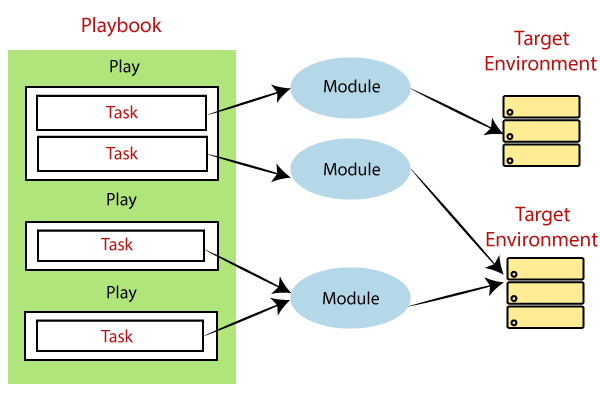
Playbooks contain the steps which the user wants to execute on a particular machine. And playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.

Ansible playbooks tend to be more configuration language than a programming language.

Through a playbook, you can designate specific roles to some of the hosts and other roles to other hosts. By doing this, you can orchestrate multiple servers in very different scenarios, all in one playbook.

## Playbook Structure

Each playbook is a collection of one or more plays. Playbooks are structured by using Plays. There can be more than one play inside a playbook.



The function of the play is to map a set of instructions which is defined against a particular host.

There are different YAML editors, but prefer to use a simple editor such as notepad++. First, open the notepad++ and copy-paste the below YAML and change the language to YAML (Language → YAML).

A YAML starts with --- (3 hyphens) always.

## Create a Playbook

Let's start by writing an example YAML file. First, we must define a task. These are the interface to ansible modules for roles and playbooks.

One playbook with one play, containing multiple tasks looks like the below example.

---

   name: install and configure DB

   hosts: testServer

   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: <your service name>

Above is a basic syntax of a playbook. Save it in a file as **test.yml**. A YAML syntax needs to follow the correct indentation.

## YAML Tags

Here are some YAML tags are given below, such as:

|  |  |
| --- | --- |
| **Tags** | **Explanation** |
| Name | It specifies the name of the Ansible Playbooks. |
| Hosts | It specifies the lists of the hosts against which you want to run the task. And the host's Tag is mandatory. It tells Ansible that on which hosts to run the listed tasks. These tasks can be run on the same machine or the remote machine. One can run the tasks on the multiple machines, and the host's tag can have a group of host's entry as well. |
| Vars | Vars tag defines the variables which you can use in your playbook. Its usage is similar to the variables in any programming language. |
| Tasks | Tasks are the lists of the actions which need to perform in the playbooks. All the playbooks should contain the tasks to be executed. A task field includes the name of the task. It is not mandatory but useful for debugging the playbook. Internally each task links to a piece of code called a module. A module should be executed, and arguments that are required for the module you want to run. |

# **Ansible Tower**

Ansible Tower is like Ansible at a more enterprise level. It is a web-based solution for managing your organization with an easy user interface that provides a dashboard with all of the state summaries of all the hosts. And allows quick deployments, and monitors all configurations.

The tower allows us to share the **SSH** credentials without exposing them, logs all the jobs, manage inventories graphically, and syncs them with a wide variety of cloud providers.

Previously, Ansible Tower called the **AWX** project, is the fix to this problem. Especially those that render better as graphical rather than text-based output, such as real-time node monitoring.

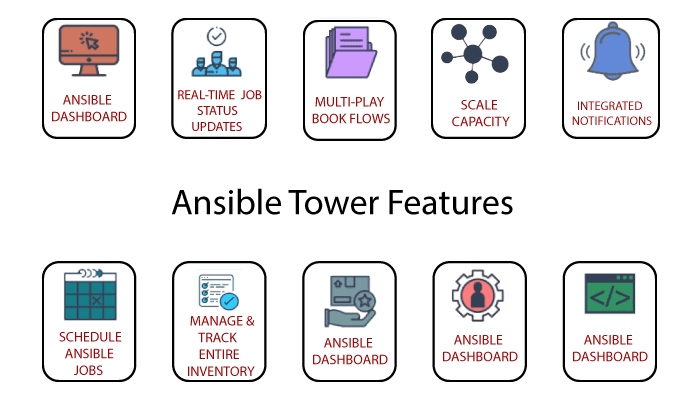
## Prerequisites to Install Ansible Tower

There is the following prerequisite to install the Ansible Tower, such as:

* The following operating systems support Ansible Tower
  + RedHat Enterprise Linux 6 64-bit
  + RedHat Enterprise Linux 7 64-bit
  + CentOS 6 64-bit
  + CentOS 7 64-bit
  + Ubuntu 12.04 LTS 64-bit
  + Ubuntu 14.04 LTS 64-bit
  + Ubuntu 16.04 LTS 64 bit
* You should have the latest stable release of Ansible.
* It required a 64-bit support kernel, runtime, and 20 GB hard disk.
* Minimum 2 GB RAM (4 GB RAM recommended) is required.
  + Minimum 2 GB RAM is recommended for Vagrant trial installations
  + And 4 GB RAM is recommended /100 forks

## Ansible Tower Features

Here are some features of the Ansible Tower, such as:



**1. Ansible Tower Dashboard:** It displays everything which is going on in your Ansible environment, such as the inventory status, the recent job activity, the hosts, and so on.

**2. Multi-Playbook Workflows:** It allows to chain any numbers of playbooks, any way of the usage of different inventories, runs different users, or utilizes various credentials.

**3. Real-Time Job Updates:** Ansible can automate the complete infrastructure. Also, you can see real-time job updates such as plays and tasks broken down by each machine either been successful or failure. Therefore you can see the status of your automation and know what's next in the queue.

**4. Scale Capacity with Cluster:** You can connect multiple Ansible Tower nodes into an Ansible Tower cluster as the clusters add redundancy and capacity, which allows scaling Ansible automation across the enterprise.

**5. Self-Service:** You can launch playbooks with just a single click through this feature.

**6. Remote Command Execution:** With this command, you can run simple tasks such as restart any malfunctioning service, add users, reset passwords on any host or group of hosts in the inventory.

**7. Manage and Track Inventory:** It manages your entire infrastructure by pulling inventory from public cloud providers such as Microsoft Azure, amazon web services, etc.

**8. Integrated Notification:** This notifies you when a job succeeds or fails across the entire organization at once, or customize on a pre-job basis.

**9. Schedule Ansible Jobs:** It schedule different kinds of jobs such as playbook runs, cloud inventory updates, and source control updates to run according to the need.

**10. REST API and Tower CLI Tool:** Every feature present in Ansible Tower is available through the Ansible Tower's REST API, which provides the ideal API for the systems management infrastructure. The Ansible Tower's CLI tool is available for launching jobs from CI systems such as Jenkins, or when you need to integrate with other command-line tools.

# **Ansible Roles**

Roles provide a framework for fully independent or interdependent collections of files, tasks, templates, variables, and modules.

The role is the primary mechanism for breaking a playbook into multiple files. This simplifies writing **complex playbooks** and makes them easier to reuse. The breaking of the playbook allows you to break the playbook into reusable components.

Each role is limited to a particular functionality or desired output, with all the necessary steps to provide that result either within the same role itself or in other roles listed as dependencies.

Roles are not playbooks. Roles are small functionality that can be used within the playbooks independently. Roles have no specific setting for which hosts the role will apply

Top-level playbooks are the bridge holding the hosts from your inventory file to roles that should be applied to those hosts.

## Creating a Role

The directory structure for roles is essential to creating a new role, such as:

**Role Structure**

The roles have a structured layout on the file system. You can change the default structured of the roles as well.

**For example,** let us stick to the default structure of the roles. Each role is a directory tree in itself. So the role name is the directory name within the /roles directory.

$ ansible-galaxy -h

## Usage

ansible-galaxy [delete|**import**|info|init|install|list|login|remove|search|setup] [--help] [options] ...

## Explanation

* The YAML file in the default directory contains a list of default variables that are to be used along with the playbook.
* The handler's directory is used to store handlers.
* The meta-directory is supposed to have information about the author and role dependencies.
* The tasks directory is the main YAML file for the role.
* The tests directory contains a sample YAML playbook file and a sample inventory file and is mostly used for testing purposes before creating the actual role.
* The vars directory contains the YAML file in which all the variables used by the role will be defined. The directory templates and the directory files should contain files and templates that will be used by the tasks in the role.

# **Ansible Variables**

In playbooks, the variable is very similar to using the variables in a programming language. It helps you to assign a value to a variable and use it anywhere in the playbook. You can put the conditions around the value of the variables and use them in the playbook accordingly.

### **Creating Valid Variable Names**

Before start using variables, it's important to know what valid variable names are.

Variable names should be letters, numbers, and underscores. The variable should always start with a letter.

foo\_port and foo2 both are the correct or valid variable names.

### **Example**

- hosts : <your hosts>

vars:

tomcat\_port : 8080

In the above example, defined a variable name **tomcat\_port** and assigned the value 8080 to the variable and can use it in your playbook wherever required.

block:

   - name: Install Tomcat artifacts

      action: >

      yum name = "demo-tomcat-1" state = present

      register: Output

   always:

      - debug:

         msg:

            - "Install Tomcat artifacts task ended with message: {{Output}}"

            - "Installed Tomcat artifacts - {{Output.changed}}"

### **Explanation**

* **block:** The Ansible syntax to execute a given block.
* **name:** It is used in logging and helps in debugging which all blocks were successfully executed.
* **action:** The action is an Ansible keyword used in YAML.
* **register:** The output of the action tag is registered by using the register keyword.
* **always:** It is also an Ansible keyword; it says that below will still be executed.
* **msg:** It displays the message.

# **Ansible Tags**

If you have a large playbook, it becomes useful to be able to run only a specific part of it rather than running everything in the playbook. Ansible supports a tag attribute for this reason.

When you execute a playbook, you can filter tasks based on the tags in two ways, such as:

1. On the command line, with the **-tags** or **-skip-tags** options.
2. In Ansible configuration settings, with the **TAGS\_RUN** and **TAGS\_SKIP** options.

tasks:

- yum:

    name: "{{ item }}"

    state: present

  loop:

  - httpd

  - memcached

  tags:

  - packages

- template:

    src: templates/src.j2

    dest: /etc/foo.conf

  tags:

  - configuration

If you want to run the **configuration** and **packages** part of a very long playbook, then you can use the -tags option on the command line.

ansible-playbook example.yml --tags "configuration,packages"

And if you want to run a playbook without certain tagged tasks, then you can use the **-skip-tags** command-line option.

ansible-playbook example.yml --skip-tags "packages"

## Tag Reuse

We can apply the same tag to more than one task. By using the "**--tags**" command line options, all tasks with that tag name will be run.

**For example:** In below example, we use one tag "**ntp**" for several tasks, such as:

---

# file: roles/common/tasks/main.yml

- name: be sure ntp is installed

  yum:

    name: ntp

    state: present

  tags: ntp

- name: be sure ntp is configured

  template:

    src: ntp.conf.j2

    dest: /etc/ntp.conf

  notify:

  - restart ntpd

  tags: ntp

- name: be sure ntpd is running and enabled

  service:

    name: ntpd

    state: started

    enabled: yes

  tags: ntp

## Special Tags

"**always**" is a unique tag that will always run a task, unless specifically skipped (**--skip-tags always**)

**For example:**

tasks:

- debug:

    msg: "Always runs"

  tags:

  - always

- debug:

    msg: "runs when you use tag1"

  tags:

  - tag1

## New in version 2.5

Here is another unique tag that is "**never**" which prevents a task from running unless a tag is specifically requested.

**For example:**

tasks:

  - debug: msg="{{ showmevar }}"

    tags: [ never, debug ]

In the above example, the task will only run when the "**never**" or "**debug**" tag is explicitly requested.

Here are another three special keywords for tags:

"**tagged**" which run only tagged,

"**untagged**" which run only untagged, and

"**all**" which run all tasks respectively.

# **Ansible Galaxy**

Ansible Galaxy is a galaxy website where users can share roles and to a command-line tool for **installing, creating,** and **managing** roles.

Ansible Galaxy gives greater visibility to one of Ansible's most exciting features, such as application installation or reusable roles for server configuration. Lots of people share roles in the Ansible Galaxy.

Ansible roles consist of many playbooks, which is a way to group multiple tasks into one container to do the automation in a very effective manner with clean, directory structures.

## Ansible Galaxy Commands

* To display the list of installed roles, with version numbers.

ansible-galaxy list

* To remove an installed role.

ansible-galaxy remove [role]

* To create a role template suitable for submission to Ansible Galaxy.

ansible-galaxy init

## Create Roles with Ansible Galaxy

The Ansible Galaxy is essentially a large public repository of Ansible roles. Roles ship with READMEs detailing the roles use and variables. Ansible Galaxy contains a large number of roles that are continually evolving and increasing.

The Galaxy can use Git to add other role sources like GitHub. You can initialize a new galaxy role using **the ansible-galaxy init** or install a role directly from the Ansible galaxy role store by executing the **ansible-galaxy install <name of role>**command.

To create an Ansible role using the Ansible Galaxy, you need to use the **ansible-galaxy** command and its templates. Roles must be downloaded before they used in the playbooks. They are placed into the default directory **that is /etc/ansible/roles**.

## Create Collections

Ansible Galaxy has been a tool for constructing and managing roles with new iterations of the Ansible, and you are bound to see changes or additions. On Ansible version 2.8, you get the unique feature of the collections.

Collections are the distribution format for the Ansible content. They can be used to package and distribute roles, modules, playbooks, and plugins.

Collections follow the following simple structure:

The ansible-galaxy-collection command implements the following commands. Some commands are the same as used with ansible-galaxy, such as:

* **init:** It creates a basic collection Skeleton based on the default template included with Ansible or your own template.
* **build:** It creates a collection artifact that can be uploaded to the galaxy or your own repository.
* **publish:** It publishes a built connection artifact to the galaxy.
* **install:** It installs one or more connections.

# **Ansible Commands Cheat Sheets**

* To install EPEL repo on Centos/RHEL systems.

[root@ansible-server ~]# sudo yum install epel-release

* To install Ansible package on Centos/RHEL systems.

[root@ansible-server ~]# sudo  yum install -y ansible

* To perform an update to the packages on Debian/Ubuntu systems.

$ sudo apt update

* To install the software properties-common-package on Debian/Ubuntu systems.

$ sudo apt install software-properties-common

* To install Ansible personal package archive on Debian/Ubuntu systems.

$ sudo apt-add-repository ppa:ansible/ansible

* To install Ansible on Debian/Ubuntu systems.

$ sudo apt update

$ sudo apt install ansible

* To issue a ping command on all servers defined in the inventory file named hosts.

[root@ansible-server test\_ansible]# ansible -i hosts all -m ping

* To issue a ping command only on hosts2.

[root@ansible-server test\_ansible]# ansible -i hosts all -m ping --limit host2

* To copy the file "testfile" on all hosts in the inventory file.

[root@ansible-server test\_ansible]# ansible -i hosts all -m copy -a "src=/root/test\_ansible/testfile dest=/tmp/testfile"

* To install ncdu package on all hosts.

[root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=present'

* To remove ncdu package on all hosts.

[root@ansible-server test\_ansible]# ansible -i hosts all -m yum -a 'name=ncdu state=absent'

* To build the directory structure for the role named role1.

[root@ansible-server test2]# ansible-galaxy init role1

* To dry-run p4.yml playbook.

[root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml --check

* To run a p4.yml playbook with password authentication for all hosts.

[root@ansible-server test\_ansible]# ansible-playbook -i hosts p4.yml -k

# **Ansible Modules**

Ansible modules are discrete units of code which can be used from the command line or in a playbook task.

The modules also referred to as task plugins or library plugins in the Ansible.

Ansible ships with several modules that are called **module library**, which can be executed directly or remote hosts through the playbook.

Users can also write their modules. These modules can control like **services, system resources, files,** or **packages,** etc. and handle executing system commands.

Let's see how to execute three different modules from the command line.

ansible webservers -m service -a "name=httpd state=started"

ansible webservers -m ping

ansible webservers -m command -a "/sbin/reboot -t now"

Each module supports taking arguments. Mainly all modules take **key=value** arguments, space delimited.

Some module takes no arguments, and the shell/command modules take the string of the command which you want to execute.

From playbook, Ansible modules execute in a very similar way, such as:

- name: reboot the servers

  command: /sbin/reboot -t now

# **Ansible Templates**

Ansible is used to manage configurations of multiple servers and environments. But these configuration files can vary for each cluster or remote server. But apart from a few parameters, all other settings will be the same.

Creating static files for each of these configurations is not an efficient solution. It will take a lot of time, and every time a new cluster is added, then you have to add more files. If there is an efficient way to manage these dynamic values, it would be beneficial. This is where Ansible template modules come into play.

A template is a file that contains all your configuration parameters, but the dynamic values are given as variables in the Ansible. During the playbook execution, it depends on the conditions such as which cluster you are using, and the variables will be replaced with the relevant values.

You can do more than replacing the variables with the help of the Jinj2 templating engine. You can have loops, conditional statements, write macros, filters for transforming the data, do arithmetic calculations, etc.

Usually, the template files will have the .j2 extension, which denotes the Jinja2 templating engine used.

The double curly braces will denote the variables in a template file, **'{{variables}}'**.

We need to have two parameters when using the Ansible Template module, such as:

* **src:** The source of the template file. It can be a relative and absolute path.
* **dest:** Dest is the destination path on the remote server.

## Template Module Attributes

Here are some other parameters which can be used to change some default behavior of the template module:

* **Force:** If the destination file already exists, then the Force parameter will decide whether it should be replaced or not. By default, the value is yes.
* **Mode:** This parameter is used to set the permissions for the destination file explicitly.
* **Backup:** If you want a backup file to be created in the destination directory, you should set the value of the backup parameter to yes. By default, the value is no. and the backup file will be created every time there is a change in the destination directory.
* **Group:** Name of the group that should own the directory. It is similar to executing chown command for a file in Linux systems.

### **Example**

In the below example, we are using the template module on the example1.j2 file that replaces the default variables with values given in the playbook.

**File: Playbook.yml**

---

- hosts: all

  vars:

    variable1: 'Hello'

    variable2: 'My first playbook using template'

  tasks:

    - name: Basic Template Example

      template:

        src: example1.j2

        dest: /home/knoldus/Documents/Ansible/output.txt

**File: example1.j2**

{{variable1}}

No change in **this** line

{{variable2}}

**File: output.txt**

Hello

No change in this line

My first playbook using the template

# **Ansible YAML**

YAML is used to describe configuration that has been increasing in the past few years with the help of **Ansible** and **SaltStack**.

YAML is more comfortable for humans to read and write in comparison to other standard data formats such as XML or JSON. There are libraries available in most programming languages for working with YAML.

For Ansible, every YAML file starts with a list. Each item in the list is a list of key-value pairs, commonly called a **"hash"** or **"dictionary"**. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with **---** and end with **---**. This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a **"-"** (a dash and space):

---

# A list of colors

- White

- Orange

- Red

- Black

---

We have different ways in which the YAML data is represented, such as:

**Key-value Pair**

YAML uses the Key-Value pair to represent the data. And the dictionary is described in the key: value pair.

#### **NOTE: There should be space between: and value.**

**For example,** a student record

---

# A student record

Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

---

**Abbreviation**

We can also use the abbreviation to represent the directories:

Martin: [name: martin, roll no: 10, **class**: 12th, div: A]

### **Representing List**

We can also represent List in YAML. Every element (member) of the list should be written in a new line with the same indentation starting with "-" (- and space).

**For example:** Name of the countries

---

#Name of country

Countries:

   - India

   - China

   - USA

   - Iceland

---

**Abbreviation**

To represent the list, we can also use the abbreviation method:

Countries: ['India', 'China', 'USA', 'Iceland']

**List inside Dictionaries**

We can use the list inside dictionaries, i.e., the value of a key is a list.

**For example,** a student record

---

# A student record

Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

---

**List of Directories**

We can also make a list of directories:

**For example:**

---

# A student record

- Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

- Edward:

 name: Edward

 roll no: 11

**class**: 12th

div: A

likes:

- Biology

- English

---

YAML uses **"|"** to include newlines while showing multiple lines and **">"** to suppress newlines while showing various lines. Due to this, we can read and edit long lines. In both cases, the indentation will be ignored.

We can also represent **Boolean** (True/false) values in YAML, where **Boolean** values can be case insensitive.

**For example,** a student result

---

#a student result

- Martin:

name: Martin

roll no: 10

**class**: 12th

div: A

likes:

- Physics

- Chemistry

- Math

   result:

      Physics: 70

     Chemistry: 45

Math: 85

Biology: 65

      English: 80

   passed: TRUE

   messageIncludeNewLines: |

      Congratulation!!

      You passed with 79%

   messageExcludeNewLines: >

      Congratulation!!

      You passed with 79%

---

# **Ansible Inventory**

Ansible works against multiple managed hosts in your infrastructure at the same time, using a list or group of lists is known as the inventory.

Once an inventory is defined, you use patterns to select the hosts or groups you want to run against to Ansible.

The default location for inventory is a file called **/etc/ansible/hosts**. You can also specify a different inventory file at the command line using the **-i <path>** option. You can pull the inventory file from dynamic or cloud sources or different formats (YAML, ini). Ansible has inventory plugins to make it flexible and customize.

### **Hosts and group**

The format is **/etc/ansible/** hosts are in **INI** like format, such as:

mail.example.com

[webservers]

foo.example.com

bar.example.com

[dbservers]

one.example.com

two.example.com

three.example.com

Heading in the brackets is a group name, which is used in classifying the systems. And deciding what policy you are controlling at what time and for what purpose. You can put the systems in more than one group.

For example, a server could be both a **dbserver** and a **webserver**.

If you have hosts that run on a non-standard SSH port, then you can put the port number after the hostname with the colon. The Ports listed in the SSH configuration file that can be used with the **OpenSSH** connection but not use with the **paramiko** connection.

To makes things explicit, it is suggested that you set them if items are not running on the default ports:

badwolf.example.com:5309

Suppose you have static IPs and want to set up some aliases that live in your host file, or you can connect through tunnels. Also, you can describe the hosts like the below example:

Jumper ansible\_port=5555 ansible\_host=192.0.2.50

In the above example, trying to Ansible against the host alias "**jumper**" will connect 192.0.2.50 on port 5555. It is using features of the inventory file to define the special variables.

### **Hosts Variables**

You can assign the variables to the hosts that will be used in playbooks, such as:

[atlanta]

host1 http\_port=80 maxRequestsPerChild=808

host2 http\_port=303 maxRequestsPerChild=909

### **Group Variables**

The variables can be applied to an entire group at once, such as:

[atlanta]

host1

host2

[atlanta:vars]

ntp\_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

### **Groups of Groups and Group Variables**

It is possible to make groups of the group using the **:children's** suffix. And you can apply variables using **:vars**.

[atlanta]

host1

host2

[raleigh]

host2

host3

[southeast: children]

Atlanta

Raleigh

[southeast:vars]

some\_server=foo.southeast.example.com

halon\_system\_timeout=30

self\_destruct\_countdown=60

escape\_pods=2

[usa: children]

southeast

northeast

southwest

northwest

# **Ansible Debug**

Ansible provides a debug module option that makes the tasks more manageable. It is a handy tool to figure out any problem areas.

Ansible version 2.1 extended the debug module with a verbosity parameter that transforms it from a print line.

**For example:** Let's create the playbook **1\_debug\_example.yml**, such as:

---

- name: Debug Example Uptime

hosts: localhost

connection: local

tasks:

- name: Find Uptime

shell: /usr/bin/uptime

register: result

- name: Print debug message

debug:

var: result

verbosity: 2

The debugger keyword accepts several values, such as:

**Always:** Always invokes the debugger, regardless of the outcome.

**Never:** Never invokes the debugger, regardless of the outcome.

**On\_failed:** It only invokes the debugger if a task fails.

**On\_unreachable:** It only invokes the debugger if the host was unreachable.

**On\_skipped:** It only invokes the debugger if the task is skipped.

#### **NOTE: These options override any global configuration to enable or disable the debugger**

# 

# **Ansible Apt**

APT stands for "Advanced Packaging Tool" is the preferred package management toolset in Ubuntu. It allows us to install new packages, update them, and remove the packages from Ubuntu or Debian systems. Here are 3 APT related command-line tools, such as:

**Apt-get:** All the basic package management operations can be done by using this tool. Ansible apt-get module provides this functionality.

**Apt-add-repository:** It is used for adding a new repository to the repository list. The default repository may not have the latest version of all the packages. So you need to add additional repositories for some software maintainers. Ansible apt\_repository module provides the functionality for adding a new repository.

**Apt-key:** It is used to manage the list of keys for authenticating apt packages. Ansible apt\_key module is used to manage the keys.

### **Installing new Apt Packages**

To install the new packages, you have to give the name of the package in the name parameter and the desired state of the package.

The default state of the package is "present". Also, it is better to set the update\_cache to true. Thus you can ensure the indexes are synchronized with the sources list. It is the same as running the apt-get update command before installing a package.

The below example will do a cache update to synchronize the index. Check if the 'zip' package is installed on the target server. And if it is not installed, the package will be installed. If the package is already installed, then it won't be upgraded.

-hosts: loc

tasks:

-name: Ansible apt install packages

apt:

name: zip

state: present

update\_cache: **true**

**1. Installing the latest version of a package**

If you set the state of the packages to "present", then Ansible will only check if the package is present. So if the new package is available, it will not be able to install.

If you want to install the latest apt packages, then you have to set the state parameter to the latest.

This will ensure the package with the latest version is installed. The below example will update the cache first, then install the latest package of zip, such as:

-hosts: loc

tasks:

-name: ansible apt install latest version

apt:

name: zip

state: latest

update\_cache: **true**

**Ansible install multiple packages**

Instead of writing multiple tasks to install packages, you can use **with\_items** and combine those tasks.

In the below example, we are going to install 3 packages: docker-ce, Nginx, and git.

-hosts: loc

tasks:

-name: ansible apt with\_items

apt:

name: "{{item}}"

update\_cache: **true**

state: present

with\_items:

-'docker-ce'

-'nginx'

-'git'

**Ansible Apt ad-hoc**

You can also use the ad-hoc method to install new packages using the apt module, such as:

ansible all -m apt -a "name=nginx state=absent" -i inventory.ini

### **Removing Apt Packages**

You can also remove the packages using apt module by setting the state parameter to absent.

The below example will remove the zip package. Since the module is idempotent, it will not go through an error if the package is not present.

-hosts: loc

tasks:

-name: ansible apt remove **package**

apt:

name: zip

state: absent

# **Ansible Lineinfile**

The lineinfile is one of the most powerful modules in the Ansible toolbox. Ansible lineinfile module is used to insert a line, modify, remove, and replace an existing line.

Ansible lineinfile module saves your time when you work with the file and modify their content on the run, such as adding a new line in the file or updating, replace a line in the file when specific text is found, and much more.

Ansible lineinfile provides many parameters to do the job quickly. You can also use the condition to match the line before modifying, removing using the regular expressions. You can reuse and modify the matched line using the backreference parameter.

#### **NOTE: Ansible lineinfile can be used only for working a single line in a file. If you want to replace multiple lines, replace the module, and if you're going to insert, update, remove a block of lines in a file use blockinfile module.**

### **Insert a Line**

Let's see how to write a line to a file if it is not present. You can set the path of the file to be modified using the path (>Ansible 2.3)/ dest parameter. And set the line to be inserted using the line parameter.

The below example will write the line "Inserting a line in a file" to the file "*remote\_server.txt*". The new line will be added to the EOF. If the line is already existing, then it will not be combined.

You can also set the **create** parameter, which says if the file is not present, then create a new file. The default value for the state is **present**.

- hosts: loc

  tasks:

    - name: Ansible insert lineinfile

      lineinfile:

        dest: /home/javaTpoint/remote\_server.txt

        line: Inserting a line in a file.

        state: present

        create: yes

### **Removing a Line**

Set the state parameter to absent or remove the line specified. All the occurrence of that line will be removed.

- hosts: loc

  tasks:

    - name: Ansible lineinfile remove the line

      lineinfile:

        dest: /home/javaTpoint/remote\_server.txt

        line: Removed lines.

        state: absent

### **Replacing or Modifying a Line**

To modify a line, you need to use the Ansible backrefs parameter along with the regexp parameter. This should be used with state=present.

If the regexp does not match any line, then the file is not changed. If the regexp matches a line or multiple lines, then the last matched line will be replaced. The grouped elements in regexp are populated and can be used for modification.

In the below example, we are commenting on a line. The full line is captured line by placing them inside the parenthesis to '\1'. The '#\1' replaces the line with '#' followed by what was captured.

You can have multiple captures and call them by using '\1', '\2', '\3' etc.

**Commenting a line with Ansible lineinfile backrefs**

- name: Ansible lineinfile regexp replace the example

  lineinfile:

    dest: /etc/ansible/ansible.cfg

    regexp: '(inventory = /home/fedora/inventory.ini.\*)'

    line: '#\1'

    backrefs: yes

**Uncommenting the line with lineinfile regexp**

- name: Ansible lineinfile backrefs example

  lineinfile:

    dest: /etc/ansible/ansible.cfg

    regexp: '#(inventory = /home/fedora/inventory.ini.\*)'

    line: '\1'

    backrefs: yes

# **Ansible Copy**

Ansible provides the functionality of copying the files and directories with the help of copy and fetch modules. The copy module is versatile.

The copy module is used to copy files and folders from the local machine to the remote servers. And the fetch module to copy data from the remote machine to the local machine.

If you want to copy files after substituting with variables, such as config files with IP changes, or you can use the template module also. You can perform a lot of complicated tasks with this module.

## Copying Files from Local to Remote

The copy module is used to check the file set in the **src** parameter, on the local machine. And then, it will copy the data to the remote machine path specified in the **destpath**.

In the below example, we will copy the **sample.txt** file in the home directory of the local machine, to the destination is the **/tmp** directory on the remote server. as long as we are not specifying any permission for the file, the default permission for the remote file is set as -**rw-rw-r-(0664)**.

- hosts: blocks

  tasks:

  - name: Ansible copy file to a remote server

    copy:

      src: ~/sample.txt

      dest: /tmp

**Case 1:** If the file is already present on the remote server, but the source file's content is different, then the destination file will be modified. You can control this by setting the force parameter. The default is set to yes. So it modifies the file by default.

If you don't want the file to be modified, if the source file is different, then you can set it No. The following task will copy the file if the file does not exist on the remote server.

- hosts: blocks

  tasks:

  - name: Ansible copy file force

    copy:

      src: ~/sample.txt

      dest: /tmp

      force: no

**Case 2:** If the file did not found on the local machine, the Ansible throw an error.

**For example:** fatal: [remote-machine-1]: FAILED!=> {"changed": false, "failed": true, "msg": "unable to find '~/sample.txt' in expected paths."}

## Copying Directories from Local to Remote

You can also copy folders or directories using the Ansible copy module. If the '**src**' path is a directory, then it will be copied recursively. Or the entire directory will be copied.

There are two different variations for this task. Depending on whether you have the '/' character at the endpoint of the 'src' path or not.

The first method will create a directory on the remote server, with the name set in the src parameter. Then, it will copy and paste the content of the source folder into that directory.

If you want this behavior, then it doesn't give the '/' after the path in the src parameter.

in the below example, it will first create a directory named **copy\_dir\_ex** in the **/tmp** of the remote server.

- hosts: blocks

  tasks:

  - name: Ansible copy the directory to the remote server

    copy:

      src:/Users/mdtutorials2/Documents/Ansible/copy\_dir\_ex

      dest:/Users/mdtutorials2/Documents/Ansible/tmp

## Copying Files between Directories on Remote Machine

Ansible copy allows you to copy the files from one directory to another on the same remote machine. But this is only for files, not for the directories. You can use the **remote\_src parameter** to let Ansible know your intentions.

The below code will copy **/tmp/test.txt** to the home directory of the user **(/home/[username]/)**.

---

-hosts: webservers

tasks:

-name: copy the file between directories on a remote server

copy:

src: /tmp/test.txt

dest: ~/test.txt

remote\_src: yes

# **Ansible File**

Ansible file module is used to creating and deleting the file or multiple files in the remote server. You can also create and delete the directories and change the permissions of the data.

You can also create and delete the soft links (symlinks) as well as hard links. With the help of the Ansible file module, you can set the permission of the files.

## Creating a File in Remote Server

In the Ansible file module, we have different parameters. We are using **path** and **state** parameters that are must in every file module. In the file parameter, we will mention the path of the file in the remote server. On this path, only the file will be created.

**At path:** It mentions the path of the file in the remote server

**At state:** It mentions touch, and touch will create file exact like Linux command.

Then, it will create a new empty file with the name devops.txt. So mention filename in the path. So in the state: we will mention touch to create the file.

- name: create the file in a remote server

  file:

   path: /path/to/file/in/remote/server/devops.txt

   state: touch

## Deleting a File in Remote Server

If you want to delete any command in the remote server. So at path parameter, mention the path of the file which you want to delete.

**At path:** Mention the path of the file in the remote server.

**At state:** Mention absent to delete the file.

So in the state: we will use touch to create the file, absent to delete the e file.

- name:  delete the file in a remote server

  file:

   path: /etc/abcd.conf

   state: absent

## Creating a File with Permissions

We can also create the file with permission by using the file module.

At the mode parameter: we have 4 digits. Always mention zero at the starting, and remaining digits will be your file permissions.

At owner parameter: mention the owner of the file.

tasks:

  - name: Ansible file module to create a **new** file with permissions.

    file:

    path: /path/to/cretae/file/devops.txt

    state: touch

    mode: 0421

    owner: devops

This permission will be set to that newly created file.

file:

 path: /path/to/cretae/file/devops.txt

 state: touch

 mode: "u=rw,g=w,o=e"

 owner: devops

Both the codes work the same, but in the other code, we are using the symbolic mode, which is equivalent to 0421.

## Creating Multiple Files

A path parameter: we can create a loop to create multiple files by using "{{item}}".

At with\_items parameter: mention file names which you want to create.

  By using "{{item}}" and with\_items parameter, we can create loop or multiple files.

tasks:

name: Ansible file module to create multiple files

file:

path: "{{ item }}"

state: touch

mode: 0421

with\_items:

devops1.txt

devops2.txt

devops3.txt

## 

## Deleting Multiple Files

The code will be the same to create multiple files and to delete files but a small change in the state parameter.

State parameter: Touch the create files and absent to delete files.

name: Ansible file module to delete multiple files

file:

path: "{{ item }}"

state: absent

with\_items:

devops1.txt

devops2.txt

devops3.txt

# **Ansible vs Chef**

Ansible and Chef both are the most popular configuration management tools. Both tools can accomplish many of the same tasks, they each have different strengths, and they perform their tasks in different ways.

This tutorial will explore the strengths and differences of these tools. Before move further, take a glance at Ansible and Chef.

|  |  |  |
| --- | --- | --- |
| **Parameters** | **Ansible** | **Chef** |
| **Availability** | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | When there is a failure on the primary server, which is a chef server, it has a backup server to take the place of the primary server. |
| **Easy to setup** | Ansible has only a master running on the server machine, but no agents running on the client machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | Chef has a master-agent architecture. Chef server runs on the master machine, and Chef client runs as an agent on each client machine. And also, there is an extra component called workstation, which contains all the tested configurations and then pushed to the central chef server. That's why it is not that easy. |
| **Management** | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | You need to be a programmer to manage the configurations as it offers configurations in Ruby DSL. The client pulls the configurations from the Server. |
| **Configuration language** | Ansible uses YAML (Python). It is quite easy to learn and its administrator oriented. Python is inbuilt into most Unix and Linux deployments, so setting the tool up and running is quicker. | Chef uses Ruby Domain Specific Language (Ruby DSL). It has a Steep Learning Curve and its developer-oriented. |
| **Interoperability** | The Ansible server has to be on Linux/Unix machine. As well as Ansible supports windows machines. | Chef Server works only on Linux/Unix, but Chef Client and Workstation can be on windows as well. |
| **Pricing** | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8\*5 support, whereas premium offers 24\*7 support for $14000 per year. | Chef Automate gives you everything you need to build, deploy in $137 node per year. |
| **Authoritative configuration** | Ansible's authoritative configuration comes from its deployed playbooks, which are perfect as source control systems. Or the Ansible method is more accessible and makes more sense. | The chef relies on its server as the authoritative configuration, and those servers require uploaded cookbooks, which means making sure the latter are consistent and identical. |

# **Ansible vs Puppet**

Ansible and Puppet are fast becoming essential components for managing a large number of servers. They are commonly called configuration management and remote execution tools.

These mega-useful apps allow the admin, for instance, to execute an action on several servers simultaneously, and deploy multiple apps with a single click. It makes it much easier to configure and maintain thousands of servers. Before move further, take a glance at Ansible and puppet.

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters** | **Ansible** | **Puppet** |  |
| Availability | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | Puppet has multi-master architecture. If the active master goes down, then the other master takes the active master place. |  |  |
| Easy to setup | Ansible has only a master running on the server machine, but no agents running on the client machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | Puppet also has a master-agent architecture. Puppet server runs on the master machine, and Puppet clients run as an agent on the client machine. After that, there is a certificate signing between the agent and the master. That's why it is not that easy to setup. |  |  |
| Management | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | Puppet is not easy to manage the configurations as it uses its language called Puppet DSL. The client pulls the configurations from the Server. It is entirely system-administrator oriented, and there is non-immediate remote execution. |  |  |
| Configuration language | Ansible uses YAML (Python). It is quite easy to learn, and it is administrator oriented. Python is inbuilt into most Unix and Linux deployments, so setting the tool up and running is quicker. | Puppet uses its puppet Domain Specific Language (Puppet DSL). It is not easy to learn, and it is system administrator oriented. |  |  |
| Interoperability | The Ansible server has to be on Linux/Unix machine. As well as Ansible supports windows machines. | Puppet Master works only on Linux/Unix, but Puppet Agent also works on windows. |  |  |
| Pricing | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8\*5 support, whereas premium offers 24\*7 support for $14000 per year. | The pricing for puppet ranges from $112 node per year with a standard support plan to $199 node per year with the premium plan. |  |  |
| GUI | Ansible was the command-line tool only at the time of its inception. Now it has the UI in the enterprise version, but it is not perfect. Sometimes, GUI is not in perfect sync with the Command line and not able to perform the same things like the command-line interface. | Puppet's Graphical User Interface is more interactive than Ansible. It is used to manage, view, and monitor more complex tasks. Otherwise, there is an option of using a command-line interface too when need which is written in Ruby. |  |  |