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.

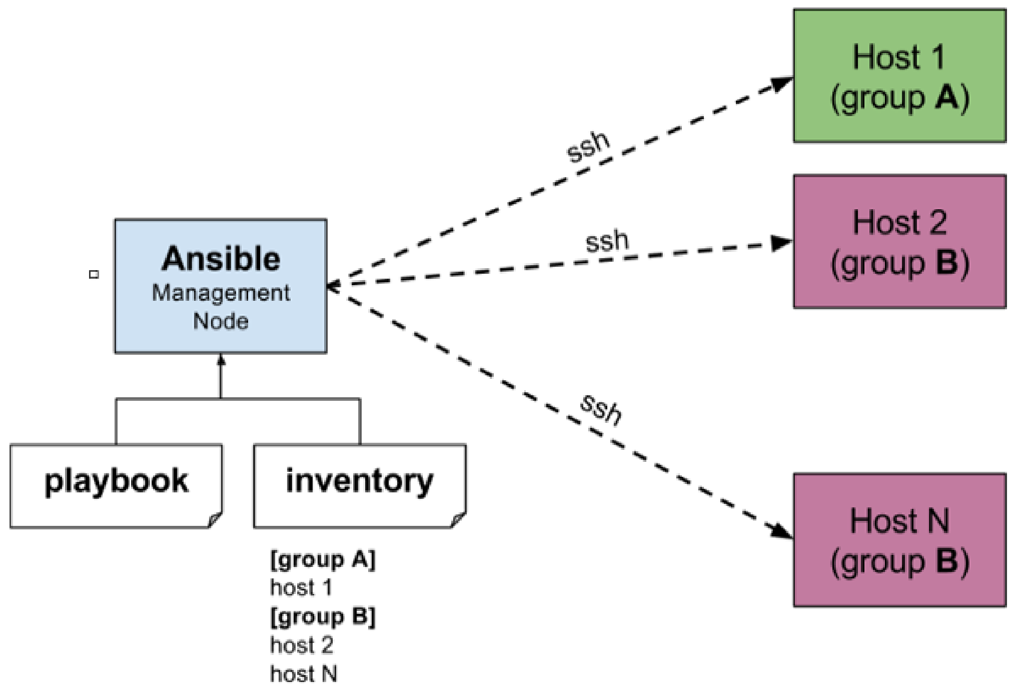
Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of **Web Logic/Web Sphere** server on all of the machines present in your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

You can install Web Logic/Web Sphere in one go on all of your machines with Ansible playbooks and inventory written in the simplest way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install Web Logic/Web Sphere. 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.

Install and Configure Ansible

Since the Ansible Control node cannot be a Windows machine, we will use Linux (Ubuntu) for running Ansible.

.

Step 1) First, we will prepare our Ubuntu System for Ansible installation. Run the below command one by one for updating your system and making it ready for installation.

```
$ sudo apt-get update

$ sudo apt-get install software-properties-common

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

Step 2) Now, you are ready to install Ansible on your system. Run the below commands one by one.

```
$ sudo apt-get update

$ sudo apt-get install ansible
```

It will take few minutes to complete the installation of Ansible. After completing the above steps, Ansible is installed on your Ubuntu system.

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.

Create Ansible Roles

After installing Ansible on your Ubuntu system, you are ready to create Ansible Roles. Now, see the below steps to create it in your system.

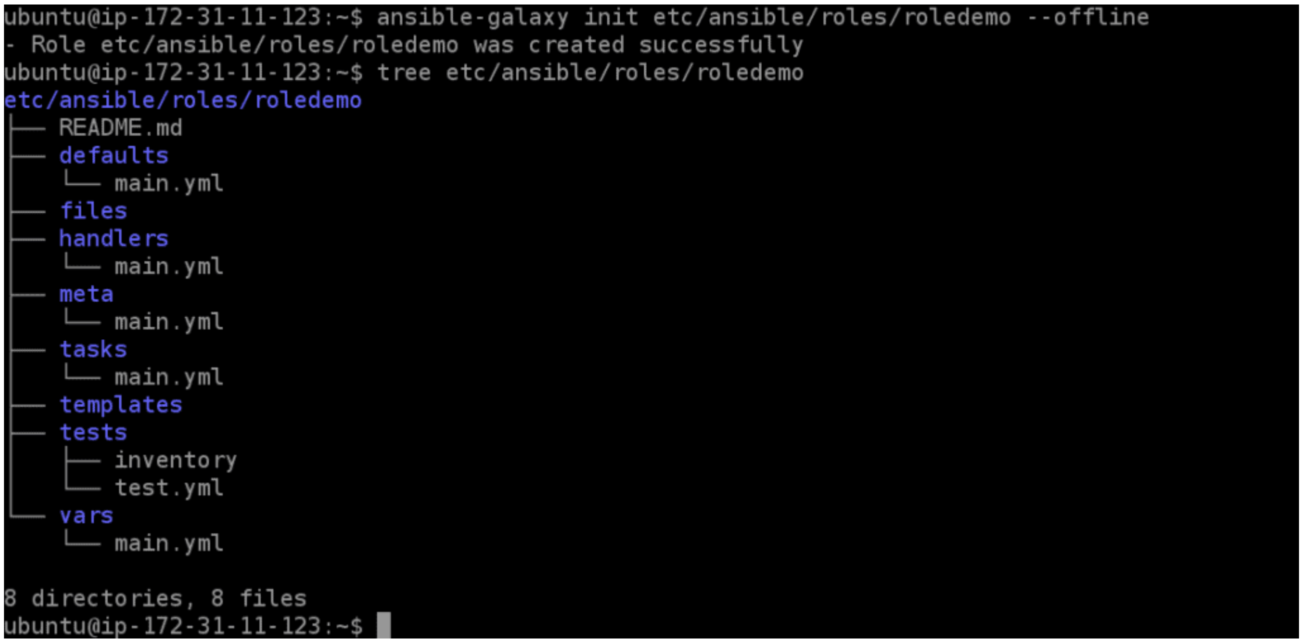
Step 1) Create an Ansible role by running the below command. You can set the location of the role based on your need. I have created the Ansible role with the name 'roledemo' located under the 'etc/ansible/roles/'directory.

```
$ Ansible-galaxy init etc/ansible/roles/roledemo --offline
```

Step 2) To check and list down the files present in a directory, you can install the *tree* tool in your system. Run the first command to install the *tree* tool, then run the second to check the tree for your roles.

```
$ sudo apt install tree # version 1.8.0-1
```

```
$ tree etc/ansible/roles/roledemo
```



```
ubuntu@ip-172-31-11-123:~$ ansible-galaxy init etc/ansible/roles/roledemo --offline
- Role etc/ansible/roles/roledemo was created successfully
ubuntu@ip-172-31-11-123:~$ tree etc/ansible/roles/roledemo
etc/ansible/roles/roledemo
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
ubuntu@ip-172-31-11-123:~$
```

You can check the result in the above image after running the *tree* command. All the directories and files are listed under your newly created Ansible role.

Configure Ansible Roles

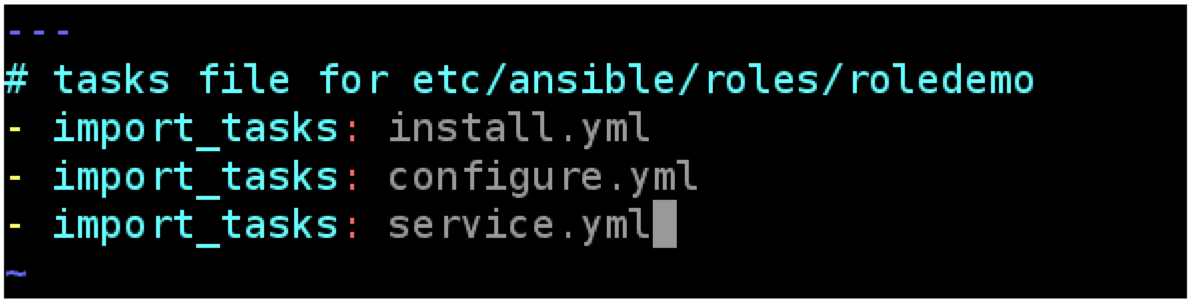
For the demo, I will be using a sample playbook divided into different roles: *install*, *configure*, and *service* YAML files. But you can use your own Ansible playbook too.

Step 1) Visit the tasks folder under your newly created roles by using the *cd* command as shown below. You can also list the files present here in this directory by running the *ll* command.

```
$ cd etc/ansible/roles/roledemo/tasks/
```

Step 2) After visiting the tasks folder of your ansible role, you need to edit the *main.yml* file. Run the *vi* or *vim* command to create or edit this file, as shown in the image below.

```
$ vim main.yml
```



```
---
# tasks file for etc/ansible/roles/roledemo
- import_tasks: install.yml
- import_tasks: configure.yml
- import_tasks: service.yml
~
```

After running the above command, an editor will be shown in front of your screen with some space to fill your content. To fill content in this file, press key *A* and use arrow buttons to take your cursor wherever you want. After you are done with the update, press the **Esc** key, then type: wq and finally press **Enter** to save your file. I have imported three tasks in the *main.yml* file that I will create in the next step.

Step 3) Now, I will create some other YAML files inside the same tasks folder. To create new YAML files, run the below *vi* commands one by one and fill the tasks in each file.

```
$ vi install.yml
$ vi cofigure.yml
$ vi service.yml
```

```
---
#tasks for install.yml file
- name: Install httpd Package
  yum: name=httpd state=latest
```

```
---
#tasks for configure.yml file
- name: Copy httpd configuration file
copy: src=files/httpd.original dest=/etc/httpd/conf/httpd.conf
- name: Copy index.html file
copy: src=files/index.html dest==/var/www/html
notify:
- restart roledemo
```

```
---
#tasks for service.yml file
- name: Start and Enable httpd service
service: name=httpd state=restarted enabled=yes
```

I have filled some demo tasks in the above 3 YAML task files. After filling the content in each YAML file, save it using the same steps mentioned in the previous step.

Step 4) As mentioned in the notifying section of the *configure.yml* file, we also need to update the *main.yml* file of handlers to restart the service when there is a change.

```
$ cd..
$ vi handlers/main.yml
---
# handlers file for etc/ansible/roles/roledemo
- name: restart roledemo
  service: name=httpd state=restarted
```

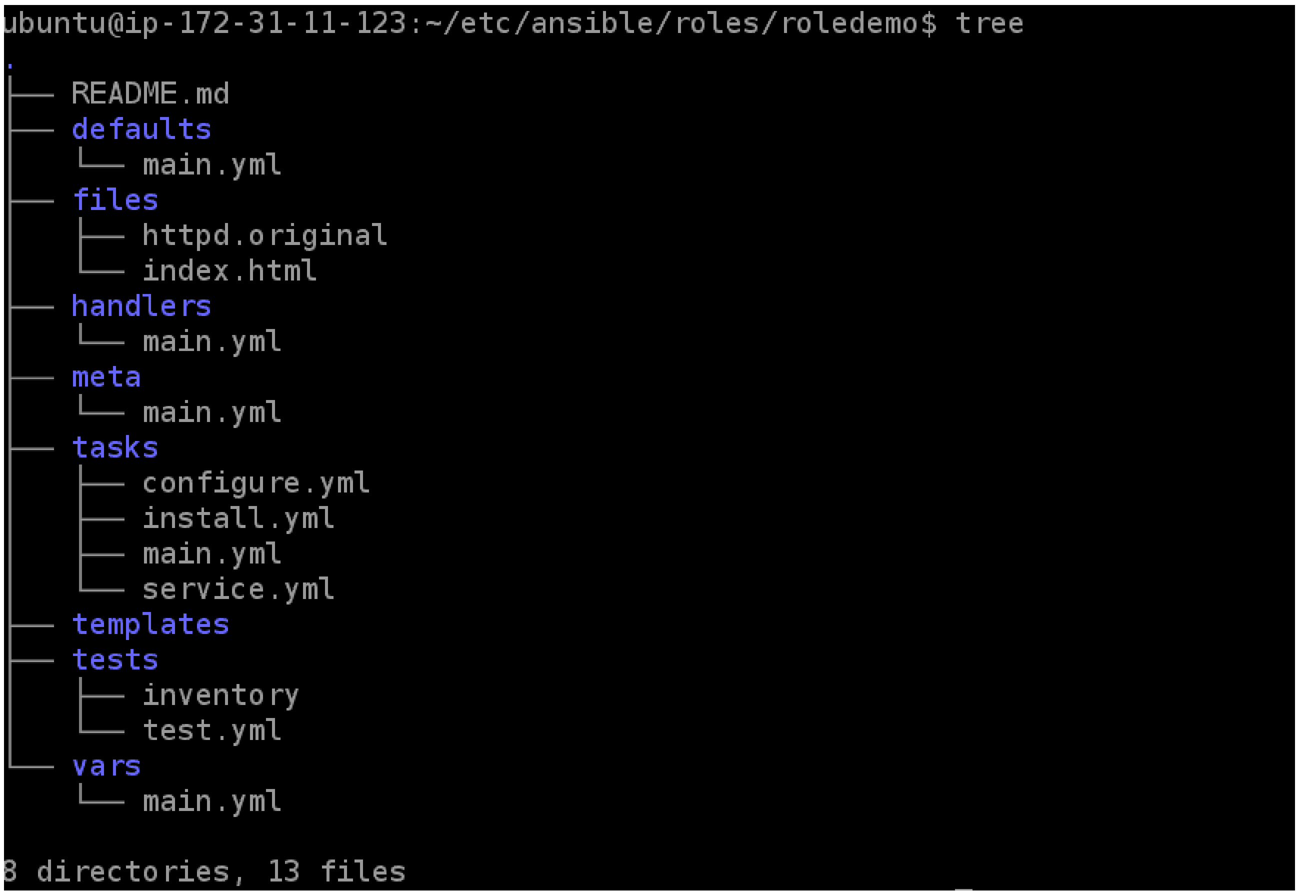
Step 5) The above-created role's task is to copy some files from the sources to the destination mentioned in the *configure.yml* file. Visit the directory with name *files* using the command given below.

```
$ cd
$ cd etc/ansible/roles/roledemo/files/
$ vi httpd.original
$ vi index.html
```

Fill content in the above files and save it. Inside the *index.html* file, you can fill in any message or layout.

Step 6) Now, we have successfully created a simple Ansible role. To check all the files and directories, we have created till now. Rerun the tree command.

```
$ cd
$ tree etc/ansible/roles/roledemo
```



```
ubuntu@ip-172-31-11-123:~/etc/ansible/roles/roledemo$ tree
.
├── README.md
├── defaults
│   └── main.yml
├── files
│   ├── httpd.original
│   └── index.html
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   ├── configure.yml
│   ├── install.yml
│   ├── main.yml
│   └── service.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 13 files
```

This is the final tree of the directories and files in our Ansible role.

- 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.

Jinja2 Template in Ansible Playbook

Jinja2 is a powerful and easy to use python-based templating engine that comes in handy in an IT environment with multiple servers where configurations vary every other time. Creating static configuration files for each of these nodes is tedious and may not be a viable option since it will consume more time and energy. And this is where templating comes in.

Jinja2 templates are simple template files that store variables that can change from time to time. When Playbooks are executed, these variables get replaced by actual values defined in Ansible Playbooks. This way, templating offers an efficient and flexible solution to create or alter configuration file with ease.

Template architecture

A Jinja2 template file is a text file that contains variables that get evaluated and replaced by actual values upon runtime or code execution. In a Jinja2 template file, you will find the following tags:

- **{{ }}** : These double curly braces are the widely used tags in a template file and they are used for embedding variables and ultimately printing their value during code execution. For example, a simple syntax using the double curly braces is as shown: The {{ webserver }} is running on {{ nginx-version }}

- **{% %}** : These are mostly used for control statements such as loops and if-else statements.

- **{# #}** : These denote comments that describe a task.

In most cases, Jinja2 template files are used for creating files or replacing configuration files on servers. Apart from that, you can perform conditional statements such as **loops** and **if-else** statements, and transform the data using filters and so much more.

Template files bear the **.j2** extension, implying that Jinja2 templating is in use.

Creating Template Files

Here’s an example of a Jinja2 template file example_template.j2 which we shall use to create a new file with the variables shown

```
Hey guys!

Apache webserver {{ version_number }} is running on {{ server }}

Enjoy!
```

Here, the variables are **{{ version_number }}** & **{{ server }**

Ansible Vault

Ansible Vault is a feature which allows user to encrypt values and data structures within Ansible projects. This provides the ability to secure any secrets or sensitive data that is necessary to run Ansible plays successfully but should not be publicly visible, such as private keys or passwords. Ansible automatically decrypts the vault-encrypted content at runtime when the key is provided.

To integrate these secrets with regular Ansible data, both the Ansible and Ansible-playbook commands, for executing ad hoc tasks and structured playbook respectively, have support for decrypting vault-encrypted content at runtime.

Ansible Vault is implemented with file-level granularity; it means files are either entirely encrypted or unencrypted. It uses the AES256 algorithm to provide symmetric encryption keyed to a user-supplied password.

This means the same password is used to encrypt and decrypt the content, which is helpful from a usability standpoint. Ansible can identify and decrypt any vault-encrypted files it finds while executing a task or playbook

Though there is a proposal to change this, at the time of writing this, users can only pass in a single password to Ansible. It means that each of the encrypted files involved must share a password.

Using Ansible Vault

The simple use of the Ansible vault is to encrypt variables files. It can encrypt any YAML file, but the most common files to encrypt are:

- A role's defaults/ main.yml file
- A role's vars/main.yml file
- Files within the group_vars directory
- Any other file used to store variables

Encrypting an Existing File

You can encrypt a regular plaintext variable file by using the ansible vault and define the password that needed later to decrypt it.

```
#encrypt a role's defaults/main.yml file
ansible-vault encrypt defaults/main.yml
>New vault password:
>Confirm new vault password:
>Encryption successful
```

The ansible-vault command will prompt you a password twice. After that, the file will be encrypted.

Creating an Encrypted File

To create an encrypted data file, use the ansible-vault to create command, and pass the filename.

```
$ansible-vault create <file name>
```

You will be prompted to create a password and then confirm it by re-typing it.

Once your password is confirmed, a new file will be created and will open an editing a window. By default, the editor for Ansible vault is VI. You can add data, save it, and exit from it.

Editing Encrypted Files

If you want to edit the encrypted file, you can edit it using ansible-vault edit command. This command will decrypt the file to a temporary file and allow you to edit the file.

1. $ansible-vault edit <file name>

   You will be prompted to insert the vault password. The decrypted file will open in a VI editor, and then you can make the required changes. Save the changes and removing the temporary file.

   ## Rekeying Encrypted Files

   If you want to change your password on a vault on a vault-encrypted file, you can do it by using the rekey command.

1. $ansible-vault rekey <file1> <file2> <file3>

   The above command can rekey multiple data files at once and ask for the original password and the new password.

   ## Encrypting Unencrypted Files

   If you have existing files which you want to encrypt, use the ansible-vault encrypt command. This command can operate on multiple files at once.

1. $ansible-vault encrypt <file1> <file2> <file3>

   ## Decrypting Encrypted Files

   If you have existing files that you no longer want to keep encrypted, you can decrypt them permanently by running the ansible-vault decrypt command. This command will save them unencrypted to the disk.

1. $ansible-vault decrypt <file1> <file2> <file3>

   ## Viewing Encrypted Files

   If you want to view the contents of an encrypted file without editing it, then you can use the ansible-vault view command.

1. $ansible-vault view <file1> <file2> <file3>