**ANSIBLE**

**CONFIGURATION MANAGEMENT:**

Configuration management is a set of processes and tools used to systematically manage changes to software, hardware, or any other system throughout its lifecycle. It involves tracking and controlling the configuration of items within a system, ensuring consistency and integrity, and providing the ability to trace and audit changes.

It is a method through which we automate admin tasks. Configuration management tool turns your code into infrastructure. So your code would be testable, repeatable and version able.

Infrastructure refers to the composite of: Software, Network, Storage and Process.

**ANSIBLE:**

1. Ansible is one among the DevOps configuration management tools which is famous for its simplicity.
2. It is an open source software developed by Michael DeHaan and its ownership is on RedHat
3. Ansible is an open source IT Configuration Management, Deployment & Orchestration tool.
4. This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.
5. Ansible is an automation tool that provides a way to define infrastructure as code.
6. Infrastructure as code (IaC) simply means that managing infrastructure by writing code rather than using manual processes.
7. The best part is that you don’t even need to know the commands used to accomplish a particular task.
8. You just need to specify what state you want the system to be in and Ansible will take care of it.
9. The main components of Ansible are playbooks, configuration management and deployment.
10. Ansible uses playbooks to automate deploy, manage, build, test and configure anything
11. Ansible is developed using Python Programming language.

**ANSIBLE FEATURES:**

* Ansible manages machines in an agent-less manner using SSH
* Built on top of Python and hence provides a lot of Python's functionality
* YAML based playbooks
* Uses SSH for secure connections
* Follows push based architecture for sending configuration related notifications

**PUSH BASED VS PULL BASED:**

* Tools like Puppet and Chef are pull based
* Agents on the server periodically checks for the configuration information from central server (Master)
* Ansible is push based
* Central server pushes the configuration information on target servers.

**WHAT ANSIBLE CAN DO?**

1. **Provision and configure infrastructure:** Ansible can provision and configure various types of infrastructure, including servers, network devices, storage systems, and cloud resources. It allows you to define desired infrastructure states in simple YAML files and then automatically deploys and configures the necessary resources.
2. **Automate tasks and workflows:** With Ansible, you can automate a wide range of tasks and workflows, such as software installation, configuration file management, service orchestration, and system updates. It provides a vast collection of pre-built modules that can be used to perform specific actions on remote systems.
3. **Enforce desired system configurations:** Ansible ensures that systems are consistently configured according to predefined specifications. It allows you to define configuration files, templates, and policies, and then applies them across multiple systems, ensuring consistency and reducing configuration drift.
4. **Manage and deploy applications:** Ansible can handle application deployment and management tasks. It supports various application deployment strategies and can integrate with container orchestration tools like Docker and Kubernetes, enabling streamlined application deployment across different environments.
5. **Simplify multi-node orchestration:** Ansible is designed for managing large-scale infrastructures with multiple nodes. It can execute tasks on multiple systems simultaneously, facilitating efficient orchestration and reducing the time required for performing tasks across multiple machines.

**HOW ANSIBLE WORKS?**

* Ansible works by connecting to your nodes and pushing out a small program called Ansible modules to them.
* Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, servers, or databases required.
* The Management Node is the controlling node that controls the entire execution of the playbook.
* The inventory file provides the list of hosts where the Ansible modules need to be run.
* The Management Node makes an SSH connection and executes the small modules on the hosts machine and install the software.
* It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine.

**Ansible basically consists of three components**

Ansible requires the following components in order to automate Network Infrastructure.

1) Controlling Node

2) Managed Nodes

3) Ansible Playbook

**Controlling Nodes** are usually Linux Servers that are used to access the switches/routers and other Network Devices. These Network Devices are referred to as the Managed Nodes.

**Managed Nodes: (Host Machines)**

Managed Nodes are stored in the hosts file for Ansible automation.

**Ansible Playbook:**

Ansible Playbooks are expressed in YAML format and serve as the repository for the various tasks that will be executed on the Managed Nodes (hosts).

Playbooks are a collection of tasks that will be run on one or more hosts.

**Host Inventory file:**

* Ansible's inventory hosts file is used to list and group your servers.
* Its default locaton is /etc/ansible/hosts
* ***Note: In inventory file we can mention IP address or Hostnames also.***

**INVENTORY FILE IMPORTANT POINTS:**

1. Comments begins with '#' character
2. Blank lines are ignore.
3. Groups of hosts are delimited by '[header]' elements
4. You can enter hostnames or IP-addresses
5. A hostname/IP can be a member of multiple groups
6. Ungrouped hosts are specifying before any group headers like below

Ansible inventory hosts file is used to list and group your servers. Its default location is **/etc/ansible/hosts**

**SAMPLE INVENTORY FILE:**

#Blank lines are ignore

#Ungrouped hosts are specifiying before any group headers like below

192.168.122.1

192.168.122.2

192.168.122.3

[webservers]

192.168.122.1

#192.168.122.2

192.168.122.3

[dbserver]

192.168.122.1

192.168.122.2

Raghuit-db1.com

Raghuit-db2.com

**ANSIBLE SETUP**

Create 3 Amazon Linux Instances of type t2.micro (Free tier eligible)

1 - Control Node

2 - Managed Nodes

**EXECUTE BELOW COMMANDS IN ALL NODES:**

**# Create one new user**

$ sudo useradd ansible

$ sudo passwd ansible

**# Provide sudoer permissions (like root user)**

$ sudo visudo

ansible ALL=(ALL) NOPASSWD: ALL

**# Activate Password Authentication YES**

$ sudo vi /etc/ssh/sshd\_config

**PasswordAuthentication yes (Might be default value NO, change to YES)**

**# Restart sshd service**

$ sudo systemctl restart sshd

**# Switch to Ansible user**

$ sudo su - ansible

**\*\* EXECUTE BELOW COMMANDS ONLY AT MASTER NODE \*\***

**# Generate a new KEY-PAIR using SSH**

$ ssh-keygen

**# Copy Public Key (Authorized Keys) into Managed Nodes**

$ ssh-copy-id ansible@<ManagedNode-Private-IP>

Ex : $ ssh-copy-id ansible@172.31.8.95

**make sure port 22 is open for SSH (Anywhere)**

**# Install Python and PIP[Package Installers for Python] software for Ansible Runtime**

$ sudo yum install python3 -y

$ python3 --version

$ sudo yum install python3-pip -y

**# Install Ansible**

$ pip3 install ansible --user

$ ansible --version

**# create ansible root directory**

$ sudo mkdir /etc/ansible

**# create ansible configuration and inventory files**

$ sudo vi /etc/ansible/ansible.cfg

**Go to:**

<https://raw.githubusercontent.com/ansible/ansible/devel/examples/ansible.cfg>

**(or)**

[**https://github.com/ansible/ansible/tree/stable-2.9/examples**](https://github.com/ansible/ansible/tree/stable-2.9/examples)

**Uncomment lines**

inventory = /etc/ansible/hosts

sudo\_user = ansible

:wq (save and quit)

$ sudo vi /etc/ansible/hosts

**https://raw.githubusercontent.com/ansible/ansible/devel/examples/hosts**

[webservers]

65.0.205.235

[dbservers]

13.234.239.70

:wq (save and quit)

**## Recommended to keep Private IPs Here**

4) Test Ansible and after installation successful

$ ansible all --list-hosts

$ ansible ungrouped --list-hosts

$ ansible webservers --list-hosts

$ ansible dbservers --list-hosts

$ ansible webservers[0] --list-hosts

$ ansible webservers[1] --list-hosts

**ANSIBLE AD-HOC COMMANDS**

Switch to ansible user and run ansible ad-hoc commands

$ sudo su ansible

**To run any ansible command we will follow below syntax:**

# ansible [ all / groupName / HostName / IP ] -m <<Module Name>> -a <<args>>

Note: Here -m is the module name and -a is the arguments to module.

**Example:**

**# ping all managed nodes listed in host inventory file**

$ ansible all -m ping

**#ping only webservers listed in host inventory file**

$ ansible webservers -m ping

**#ping only dbservers listed in host inventory file**

$ ansible dbservers -m ping

**# it will display date from all host machines.**

$ ansible all -m shell -a date

**# It will display uptime from all host machines.**

$ ansible all -m shell -a uptime

There are two default groups, **all and ungrouped**. all contains every host. Ungrouped contains all hosts that don’t have another group

**# It will display the all the modules available in Ansible.**

$ ansible-doc -l

**# To display particular module information**

$ ansible-doc <moduleName>

**# To display shell module information**

$ ansible-doc shell

**# it will display details of copy module**

$ ansible-doc -l | grep "copy"

**#It will display more information about yum module**

$ ansible-doc yum

**PING MODULE:**

**# It will ping all the servers which you have mentioned in inventory file (/etc/ansible/hosts)**

$ ansible all -m ping

# It will display the output in single line.

$ ansible all -m ping -o

**SHELL MODULE:**

**# To execute all shell commands**

# Date of all machines

$ ansible all -m shell -a 'date'

**# Release of all the machines**

$ ansible all -m shell -a 'cat /etc/\*release'

**# Check the service status on all the machines**

$ ansible all -b -m shell -a 'service sshd status'

**# Here it will check the disk space use for all the nodes which are from db servers group**

$ ansible dbservers -b -m shell -a "df -h"

**# Here it will check the disk space use for all the nodes which are from webservers group**

$ ansible webservers -b -m shell -a "free -m"

**# Here it will display date from from webservers group**

$ ansible webservers -b -m shell -a "date"

**YUM MODULE:**

**# It will install vim package in all node machine which you have mentioned in host inventory file.**

$ ansible all -b -m yum -a "name=vim"

**# Check git version in all machines**

$ ansible all -m shell -a "git --version"

**# to install git client in all node machines**

$ ansible all -m shell -b -a "yum install git -y"

**# To installl git only in webserver nodes**

$ ansible webservers -m shell -b -a "yum install git -y"

# To install webserver only in particular machine

$ ansible 172.1921.1.0 -m shell -b -a "yum install git -y"

$ ansible all -m yum -b -a "name=git state=present"

$ ansible all -m yum -b -a "name=git state=latest"

$ ansible all -m yum -b -a "name=git state=absent"

**present : install**

**latest : update to latest**

**absent : un-install**

**# to install any software in ubuntu server then we should use apt package manager**

$ ansible all -m apt -a "name="git state="present"

# To install httpd package in all node machines

$ ansible all -b -m yum -a "name=httpd state=present"

Note: Here state=latest, is not a mandatory, it is by default.

**# To update httpd package in all node machines.**

$ ansible all -b -m yum -a "name=httpd state=latest"

**# To remove httpd package in all node machines.**

$ ansible all -b -m yum -a "name=httpd state=absent"

$ ansible all -m copy -a "src="index.html dest=/var/www/html/index.html"

**# start httpd service**

$ ansible all -b -m service -a "name=httpd state=started"

$ ansible all -b -m shell -a "service httpd start"

Note: For privilege escalations we can use -b option (sudoer permissions)

**YAML (Yet Another Mark-up Language) | YAMLAinant mark-up language**

* We can make use of this language to store data and configuration in a human-readable format.
* YAML files will have .yml / .yaml as an extension
* Official Website: https://yaml.org/

**Sample YML File Data**

Fruit: Apple

Vegetable: Carrot

Liquid: Water

Meet: Chicken

**Array/List**

**Fruits:**

**- Orange**

**- Apple**

**- Banana**

**- Guava**

**Vegetables:**

**- Carrot**

**- Cauliflower**

**- Tomoto**

**Here - dash indicate the element of any array.**

**name: Raghu**

**age: 38**

**phno: 123456**

**email: javabyraghu@gmail.com**

**hobbies:**

**- cricket**

**- dance**

**- singing**

**# person data in yml**

**person:**

**id: 101**

**name: Raju**

**email: raju@gmail.com**

**address:**

**city: Hyd**

**state: TG**

**country: India**

**job:**

**companyName: IBM**

**role: Tech Lead**

**pkg: 25 LPA**

**hobbies:**

**- cricket**

**- chess**

**- singing**

**- dance**

**# using --- hyphens to separate the data**

**---**

**person:**

**id: 101**

**name: Raju**

**email: raju@gmail.com**

**address:**

**city: Hyd**

**state: TG**

**country: India**

**job:**

**companyName: IBM**

**role: Tech Lead**

**pkg: 25 LPA**

**hobbies:**

**- cricket**

**- chess**

**- singing**

**- dance**

**---**

**movie:**

**name: Bahubali**

**hero: Prabhas**

**heroine: Anushka**

**villian: Rana**

**director: SS Rajamouli**

**budget: 100cr**

**---**

**PLAYBOOKS**

1. Playbook is a single YAML file, containing one or more plays in a list.
2. Plays are ordered sets of tasks to execute against host servers from your inventory file.
3. Play defines a set of activities (tasks) to run on managed nodes.
4. Task is an action to be performed on the managed node

Examples are:

a) Execute a command

b) Run a shell script

c) Install a package

d) Shutdown / Restart the hosts

Note: Playbooks YML / YAML starts with the three hyphens ( --- ) and ends with three dots (…)

**Playbook contains the following sections:**

1) Every playbook starts with 3 hyphens (---)

2) Host section: Defines the target machines on which the playbook should run. This is based on the Ansible host inventory file.

3) Variable section: This is optional and can declare all the variables needed in the playbook. We will look at some examples as well.

4) Tasks section: This section lists out all the tasks that should be executed on the target machine. It specifies the use of Modules. Every task has a name which is a small description of what the task will do and will be listed while the playbook is run.

**Playbook to Ping All Host Nodes**

**Basic one:**

**---**

**- name: Ping all hosts**

**hosts: all**

**tasks:**

**- name: Perform ping**

**ping:**

**As ansible user:**

**---**

**- hosts: all**

**gather\_facts: no**

**remote\_user: anisble**

**tasks:**

**- name: Ping**

**ping:**

**remote\_user: ansible**

**hosts**: The tasks will be executing in specified group of servers.

**name**: which is the task name that will appear in your terminal when you run the playbook.

**remote\_user**: This parameter was formerly called just user. It was renamed in Ansible 1.4 to make it more distinguishable from the user module (used to create users on remote systems).

**gather\_facts**: is a feature that collects information about the target hosts before executing tasks in a playbook. By default, Ansible gathers facts about the hosts, such as their IP addresses, operating system details, available memory, and more.

If you to disable gathering facts to improve playbook performance or if you don't require host information for your specific tasks. You can set gather\_facts to no in your playbook.

Note : Remote users can also be defined per task.

**# Run the playbook Using below command**

$ ansible-playbook <<Playbbok file name>>

**# It will run the playbook.yml playbook in verbose**

$ ansible-playbook playbook.yml -v

$ ansible-playbook playbook.yml -vv

$ ansible-playbook playbook.yml -vvv

**# It will provide help on ansible\_playbook command**

$ ansible-playbook --help

**# It will check the syntax of a playbook**

$ ansible-playbook playbook.yml --syntax-check

**# It will do in dry run.**

$ ansible-playbook playbook.yml --check

**# It will display the which hosts would be effected by a playbook before run**

$ ansible-playbook playbook.yml --list-hosts

**# It execute one-step-at-a-time, confirm each task before running with (N)o/(y)es/(c)ontinue**

$ ansible-playbook playbook.yml --step

**ANSIBLE PLAYBOOK USECASES**

**1. CREATE AN EMPTY FILE, ADD CONTENT TO FILE, DISPLAY FILE DATA**

**---**

**- name: Create a new file on remote host**

**hosts: all**

**gather\_facts: false**

**tasks:**

**- name: Create the file**

**file:**

**path: /home/ansible/data.txt**

**state: touch**

**# Create an empty file on the remote host**

**- name: Add content to the file**

**blockinfile:**

**path: /home/ansible/data.txt**

**block: |**

**This is the first line of the file.**

**This is the second line of the file.**

**This is the third line of the file.**

**# Add the specified content to the file on the remote host**

**- name: Display the file content**

**shell: cat /home/ansible/data.txt**

**register: file\_content**

**# Execute 'cat' command to display the content of the file and register the output**

**- name: Print the file content**

**debug:**

**var: file\_content.stdout**

**# Print the output of the previous task, which contains the content of the file**

**2. CHANGE FILE PERMISSIONS (CHMOD) AND OWNER/GROUP DETAILS (CHOWN)**

**---**

**- name: Change file permission and owner details**

**hosts: all**

**gather\_facts: false**

**tasks:**

**- name: Change file permission**

**file:**

**path: /home/ansible/data.txt**

**mode: "0644"**

**# Change the file permission to 0644 (rw-r--r--)**

**- name: Change file owner**

**become: true**

**become\_user: root**

**file:**

**path: /home/ansible/data.txt**

**owner: ansible**

**group: ansible**

**# Change the file owner to 'your\_username' and group to 'your\_groupname'**

**3. Create Directory, rename and delete DIRECTORY**

**---**

**- name: Manage directories on remote host**

**hosts: webservers**

**gather\_facts: false**

**tasks:**

**- name: Create a directory**

**file:**

**path: /home/ansible/raghu**

**state: directory**

**# Create a directory on the remote host**

**- name: Rename the directory**

**command: mv /home/ansible/raghu /home/ansible/new\_info**

**# Rename the directory by executing the 'mv' command**

**- name: Delete the directory**

**file:**

**path: /home/ansible/new\_info**

**state: absent**

**# Delete the renamed directory on the remote host**

**4. CREATING NEW USER, ADDING EXISTED USER TO A GROUP, DELETE EXISTED USER**

**---**

**- name: Manage users on remote host**

**hosts: all**

**become: true**

**gather\_facts: false**

**tasks:**

**- name: Create a new user**

**user:**

**name: raghu**

**password: "{{ 'raghu' | password\_hash('sha512') }}"**

**state: present**

**# Create a new user with the name 'new\_user' and set the password**

**- name: Add an existing user to a group**

**user:**

**name: raghu**

**groups: ec2-user**

**append: yes**

**# Add the existing user 'existing\_user' to the group 'group\_name'**

**- name: Delete an existing user**

**user:**

**name: raghu**

**state: absent**

**# Delete the existing user 'existing\_user' from the remote host**

**5. INSTALL MAVEN SOFTWARE FOR HOST MACHINES BASED ON PACKAGE MANAGER TYPE.**

**---**

**- name: Install Maven**

**hosts: all**

**gather\_facts: false**

**become: true**

**tasks:**

**- name: Update apt cache (for Debian/Ubuntu)**

**apt:**

**update\_cache: yes**

**when: ansible\_pkg\_mgr == 'apt'**

**# Update the apt cache for Debian/Ubuntu systems**

**- name: Install Maven (for Debian/Ubuntu)**

**apt:**

**name: maven**

**state: present**

**when: ansible\_pkg\_mgr == 'apt'**

**# Install Maven using apt package manager on Debian/Ubuntu systems**

**- name: Install Maven (for Red Hat/CentOS)**

**yum:**

**name: maven**

**state: present**

**when: ansible\_pkg\_mgr == 'yum' or ansible\_pkg\_mgr == 'dnf'**

**# Install Maven using yum package manager on Red Hat/CentOS systems**

**- name: Install Maven (for macOS)**

**homebrew:**

**name: maven**

**state: present**

**when: ansible\_pkg\_mgr == 'homebrew'**

**# Install Maven using Homebrew package manager on macOS**

**- name: Verify Maven installation**

**command: mvn --version**

**register: maven\_version**

**# Verify the Maven installation by running 'mvn --version' command**

**- name: Display Maven version**

**debug:**

**var: maven\_version.stdout\_lines**

**# Print the output of the previous task, which contains the Maven version**

**5(B) INSTALL MAVEN SOFTWARE FOR HOST MACHINES BASED ON OS TYPE.**

**#Ansible playbook to install apache software Based on OS Family**

**---**

**- name: Install Apache HTTP Server**

**hosts: all**

**become: true**

**tasks:**

**- name: Update apt cache (for Debian/Ubuntu)**

**apt:**

**update\_cache: yes**

**when: ansible\_os\_family == 'Debian'**

**- name: Install Apache HTTP Server (Debian/Ubuntu)**

**apt:**

**name: apache2**

**state: present**

**when: ansible\_os\_family == 'Debian'**

**- name: Install Apache HTTP Server (CentOS/RHEL)**

**yum:**

**name: httpd**

**state: present**

**when: ansible\_os\_family == 'RedHat'**

**- name: Start Apache service**

**service:**

**name: apache2**

**state: started**

**when: ansible\_os\_family == 'Debian'**

**- name: Start Apache service**

**service:**

**name: httpd**

**state: started**

**when: ansible\_os\_family == 'RedHat'**

**6. INSTALL GIT IF NOT EXIST, CLONE A REPOSITORY**

**---**

**- name: Install Git and Clone Repository**

**hosts: webservers**

**gather\_facts: false**

**become: true**

**tasks:**

**- name: Check if Git is installed**

**command: git --version**

**register: git\_check**

**ignore\_errors: true**

**# Check if Git is installed by running 'git --version' command**

**- name: Install Git**

**package:**

**name: git**

**state: present**

**when: git\_check.rc != 0**

**# Install Git if it is not already installed**

**- name: Clone the repository**

**git:**

**repo: https://github.com/javabyraghu/maven-web-app.git**

**dest: /home/ansible/maven-web-app**

**# Clone the repository from the specified URL to the destination directory**

**7. INSTALL HTTPD + COPY index.html + START SERVICE**

* Create index.html file in the location where our playbook is exist.
* Create yml file with below content.

**---**

**- hosts: all**

**become: true**

**tasks:**

**- name: Install Httpd**

**yum:**

**name: httpd**

**state: present**

**- name: Copy index.html**

**copy:**

**src: index.html**

**dest: /var/www/html/index.html**

**- name: Start Httpd Server**

**service:**

**name: httpd**

**state: started**

**8. SERVICE MODULE OPERATIONS START, STOP, ENABLE AND RESTART**

**---**

**- name: Manage httpd service**

**hosts: all**

**gather\_facts: false**

**become: true**

**tasks:**

**- name: Start httpd service**

**service:**

**name: httpd**

**state: started**

**# Start the httpd service**

**- name: Stop httpd service**

**service:**

**name: httpd**

**state: stopped**

**# Stop the httpd service**

**- name: Enable httpd service**

**service:**

**name: httpd**

**enabled: true**

**# Enable the httpd service to start on boot**

**- name: Restart httpd service**

**service:**

**name: httpd**

**state: restarted**

**# Restart the httpd service**

**VARIABLES IN ANSIBLE:**

Variables are used to store and manage data that can be used throughout playbooks. Variables can be defined at various levels.

1. Local Variables: within the Ansible Playbook
2. Group Variables: Variables for a specific group
3. Host Variables: Variables for a specific host/Machine.

We can even pass variables while executing a playbook using **--extra-vars key=value** format.

**vars-demo.yml**

**---**

**- hosts: all**

**become: true**

**tasks:**

**- name: Install Httpd**

**yum:**

**name: "{{package\_name}}"**

**state: present**

**- name: Copy index.html**

**copy:**

**src: index.html**

**dest: /var/www/html/index.html**

**- name: Start Http Server**

**service:**

**name: "{{package\_name}}"**

**state: started**

**#1 Pass using --extra-vars while running playbook.**

$ ansible-playbook **vars-demo.yml** --extra-vars package\_name=httpd

**#2 Create variable inside Playbook using vars: section.**

**---**

**- hosts: all**

**become: true**

**vars:**

**package\_name: httpd**

**tasks:**

**- name: Install Httpd**

**yum:**

**name: "{{package\_name}}"**

**state: present**

**- name: Copy index.html**

**copy:**

**src: index.html**

**dest: /var/www/html/index.html**

**- name: Start Http Server**

**service:**

**name: "{{package\_name}}"**

**state: started**

**#3 GROUP VARIABLES:**

The group\_vars in Ansible are a convenient way to apply variables to multiple hosts at once. group\_vars is an Ansible-specific folder as part of the repository structure.

Group vars files should be created at host inventory location under folder **group\_vars** with file extension ‘.yml’

By considering host-inventory location: **/etc/ansible**

**package-install-demo.yml**

**---**

**- hosts: all**

**become: true**

**vars:**

**package\_name: httpd**

**tasks:**

**- name: Install Httpd**

**yum:**

**name: "{{package\_name}}"**

**state: present**

To install same packages for all groups we can create a filename ‘all.yml’

**$ mkdir /etc/ansible/group\_vars**

**Under this:**

**group\_vars/all.yml**

package\_name: tree

**Follow below Syntax:**

# group\_vars/<groupName>.yml

**# Here, we want to install git in webservers group and mysql in dbservers group, then**

**$ sudo vi /etc/ansible/group\_vars/webservers.yml**

package\_name: git

**$ sudo vi /etc/ansible/group\_vars/dbservers.yml**

package\_name: mysql

**#4. HOST VARIABLES:**

* Server specific variables.
* In one group we will have multiple servers.
* For every host if we want separate variables then we should go for host vars
* mkdir /etc/ansible/host\_vars
* create a file with host name or IP.
* vi /etc/ansible/**host\_vars**/172.138.1.1.yml

**NOTE:**

* Variable Value we can declare with in playbook.
* Variable value we can supply in runtime.
* Variable value we can declare in hosts\_vars.
* Variable value we can declare in group\_vars.

**LOOPS IN ANSIBLE:**

Loops allow you to repeat a particular task or set of tasks multiple times.

If you want to perform repetitive operations on a list of items, we can use loops.

**9. INSTALL MULTIPLE SOFTWARE’S USING VARIABLES AND LOOPS CONCEPT**

**---**

**- name: Install Packages with Loop**

**hosts: all**

**gather\_facts: false**

**become: true**

**vars:**

**packages\_to\_install:**

**- wget**

**- git**

**- tree**

**- vim**

**tasks:**

**- name: Install packages**

**package:**

**name: "{{ item }}"**

**state: present**

**loop: "{{ packages\_to\_install }}"**

**# Use the loop concept to iterate over the packages\_to\_install variable**

**- name: Verify package installation**

**command: "{{ item }} --version"**

**loop: "{{ packages\_to\_install }}"**

**register: command\_output**

**# Verify the package installation by running the command with --version flag**

**- name: Display package versions**

**debug:**

**var: item.stdout**

**loop: "{{ command\_output.results }}"**

**# Print the stdout of the previous task, which contains the package versions**

**10. LEGACY LOOPS CONCEPT USING WITH\_ITEMS**

**#LOOP WITH WITH\_ITEMS (DEPRECATED IN ANSIBLE 2.5+):**

**---**

**- name: Install Multiple Software Packages**

**hosts: all**

**gather\_facts: false**

**become: true**

**vars:**

**software\_packages:**

**- name: wget**

**state: present**

**- name: git**

**state: present**

**- name: tree**

**state: present**

**- name: vim**

**state: present**

**tasks:**

**- name: Install software packages**

**package:**

**name: "{{ item.name }}"**

**state: "{{ item.state }}"**

**with\_items: "{{ software\_packages }}"**

**#Use the with\_items concept to iterate over the software\_packages variable**

**- name: Verify package installation**

**command: "{{ item.name }} --version"**

**register: command\_output**

**with\_items: "{{ software\_packages }}"**

**#Verify the package installation by the command with --version flag**

**- name: Display package versions**

**debug:**

**var: item.stdout**

**with\_items: "{{ command\_output.results }}"**

**-------------------------------------------------------------------**

**ANOTHER APPROACH#1**

**- hosts: all**

**tasks:**

**- name: install softwares**

**yum:**

**name: ['wget', 'zip', 'unzip']**

**state: present**

**ANOTHER APPROACH#2**

**- hosts: all**

**tasks:**

**- name: install softwares**

**yum:**

**name: "{{ item }}"**

**state: present**

**with\_items:**

**- wget**

**- zip**

**- unzip**

**VAULT IN ANSIBLE:**

Ansible Vault is a feature in Ansible that allows you to encrypt sensitive data, such as passwords, API keys, or any other secret information, within your Ansible playbooks or inventory files. This helps to protect your sensitive data from being exposed in plain text, ensuring the security of your infrastructure automation.

* It is used to secure our data.
* When we configure username and password in variables files everybody can see them which is not a good practice.
* When we are dealing with sensitive data then we should secure that data.
* Using Ansible Vault we can protect or secure our data.
* Using Ansible vault we can encrypt, and we can decrypt data.
* **Encryption**: Converting from readable format to un-readable format.
* **Decryption**: Converting un-readable format to readable format (bringing file back to normal state).

**ANSIBLE VAULT COMMANDS:**

**# To create A new YAML file using Encryption**

$ ansible-vault create <playbook-yml>

**# To Encrypt existed YAML file**

$ ansible-vault encrypt <playbook-yml>

**# To view Original Data of encrypted file**

$ ansible-vault view <playbook-yml>

**# To Modify encrypted file in normal mode.**

$ ansible-vault edit <playbook-yml>

**# To Decrypt back to the normal format**

$ ansible-vault decrypt <playbook-yml>

**# To modify vault password**

$ ansible-vault rekey <playbook-yml>

**NOTE:**

* To encrypt a playbook, we need to set one vault password
* while executing playbook we need to pass vault password

**$ ansible-playbook <filename>.yml --ask-vault-pass**

* You can store vault password in a file, and you can give that file as input to execute playbook

$ vi valutpass

$ ansible-playbook filename.yml --vault-password-file=~/vaultpass

* We can see encrypted file in human readable format.

$ ansible-vault view /etc/ansible/group\_vars/all.yml

* We can edit encrypted file in human readable format.
* $ ansible-vault edit /etc/ansible/group\_vars/all.yml
* We can decrypt the file.

$ ansible-vault decrypt /etc/ansible/group\_vars/all.yml

* To update vault password, we can use rekey.

$ ansible-vault rekey /etc/ansible/group\_vars/all.yml

**TAGs & HANDLERs IN ANSIBLE:**

Ansible Tags and Handlers are useful features that provide control over task execution and allow for defining tasks that respond to specific events or conditions. ***Tags are labels assigned to tasks***, while Handlers are special tasks triggered by events.

* Using Tag we can map task to a tag-name.
* Using tag name, we can execute selected task and we can skip selected task also.
* Handlers are used to notify the tasks to execute.
* Using Handlers, we can execute tasks based on other tasks status.
* To inform the handler to execute us will use **'notify'** keyword.

Examples of how to use tags and handlers in Ansible playbooks:

**## TAGS EXAMPLE ##**

## command: ansible-playbook playbook.yml --tags=tag1

**---**

**- name: Example Playbook with Tags**

**hosts: all**

**tasks:**

**- name: Task 1**

**command: echo "Task 1 executed."**

**tags:**

**- tag1**

**- common**

**- name: Task 2**

**command: echo "Task 2 executed."**

**tags:**

**- tag2**

**- common**

**- name: Task 3**

**command: echo "Task 3 executed."**

**tags:**

**- tag1**

**- common**

You can selectively run tasks using tags when executing the playbook. For example, to execute only tasks tagged with tag1, you can use the command:

**$ ansible-playbook playbook.yml --tags=tag1**

**#HANDLERS PATTERN ##**

**---**

**- name: Example Playbook with Handlers**

**hosts: your\_target\_hosts**

**tasks:**

**- name: Task 1**

**command: echo "Task 1 executed."**

**notify:**

**- restart\_service**

**- name: Task 2**

**command: echo "Task 2 executed."**

**notify:**

**- restart\_service**

**handlers:**

**- name: restart\_service**

**service:**

**name: your\_service**

**state: restarted**

**## EXAMPLE USING TAGS and HANDLERS ##**

**---**

**- name: Using Handlers based on tasks execution**

**hosts: webservers**

**become: true**

**gather\_facts: yes**

**vars:**

**package\_name: httpd**

**tasks:**

**- name: install httpd**

**yum:**

**name: "{{package\_name}}"**

**state: present**

**tags:**

**- install**

**notify: Start Apache**

**- name: Copy index.html**

**copy:**

**src: index.html**

**dest: /var/www/html/**

**tags:**

**- copy**

**notify: Restart Httpd**

**handlers:**

**- name: Start Apache**

**service:**

**name: "{{package\_name}}"**

**state: started**

**enabled: true**

**- name: Restart Httpd**

**service:**

**name: "{{package\_name}}"**

**state: restarted**

**# To display all tags available in playbook**

$ ansible-playbook filename.yml --list-tags

**# Execute a task whose tag name is install**

$ ansible-playbook filename.yml --tags "install"

**# Execute the tasks whose tags names are install and copy**

$ ansible-playbook filename.yml --tags "install,copy"

**# Execute all the tasks in playbook by skipping install task**

$ ansible-playbook filename.yml --skip-tags "install"

**TEMPLATEs IN ANSIBLE:**

Ansible template module allows you to render templates using the ***Jinja2 templating engine***. The module takes a source template file, applies variable substitutions and logic, and generates a rendered output file.

* We generally use this templates concept for ***config files*** in servers or applications.
* A Static file that takes data from variables and become dynamic and gives output.
* We need to create a file name that ends with ***.j2***
* We must pass all required variables for template files to render data.

**Example:**

**$ sudo nano httpd\_test.conf.j2**

Welcome to {{package\_name}} installed

Author by {{author\_name}}.

Done For version {{version\_name}}!

Here we need to pass 3 variables in playbook.

**$ sudo nano templates\_test.yml**

**---**

**- name: Example Playbook**

**hosts: webservers**

**become: yes**

**vars:**

**template\_var: "Hello, World!"**

**tasks:**

**- name: Copy template file**

**template:**

**src: /home/ansible/httpd\_test.conf.j2**

**dest: /etc/httpd/conf/httpd\_test.conf**

**mode: 0744**

**owner: root**

**group: root**

**notify:**

**- Restart Service**

**handlers:**

**- name: Restart Service**

**service:**

**name: httpd**

**state: restarted**

**Execute File**

$ ansible-playbook templates\_test.yml

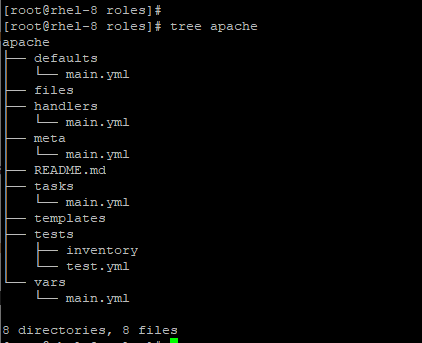
**Check at Remote host for Rendered output file:**

**$ sudo cat /etc/httpd/conf/httpd\_test.conf**

**ROLEs IN ANSIBLE:**

Ansible Roles are a way to organize and package Ansible code and files into reusable units. Roles provide a structured approach to managing infrastructure configuration by promoting code reusability, modularity, and maintainability.

A role is a ***directory structure*** that contains all the necessary components for a specific configuration task. The directory structure of a typical Ansible role looks like this:



1. **tasks/:** Contains the main tasks that define the configuration steps for the role. The main.yml file within this directory is typically the entry point of the role.
2. **handlers/:** Contains handlers that are triggered by tasks and can be used to respond to specific events. Handlers are defined in the main.yml file within this directory.
3. **templates/:** Contains Jinja2 templates that can be dynamically rendered during the role execution. These templates are typically used for generating configuration files.
4. **files/:** Contains static files that need to be copied to the target hosts as part of the role. These files can be sourced from the control node.
5. **vars/:** Contains variable definitions specific to the role. These variables can be used throughout the role's tasks and templates.
6. **defaults/:** Contains default variable definitions for the role. These variables are used when no other value is provided. Default variables can be overridden by user-defined variables.
7. **meta/:** Contains metadata about the role, including dependencies and other role-specific information.
8. **README.md:** A documentation file that provides an overview of the role and any necessary instructions for using it.

**NOTE:**

* Roles are a level of abstraction for Ansible configuration in a modular and reusable format.
* As you add more and more functionality to your playbooks, they can become difficult to maintain.
* Roles allow you to break down a complex playbook into separate, smaller chunks that can be coordinated by a central entry point.
* # Sample playbook with Role

---

- hosts: all

become: true

roles:

* + apache
* Ansible roles are consists of many playbooks, which is like modules in puppet and cookbooks in chef. We call those same in ansible as roles.
* Roles are a way to group multiple tasks together into one container to do the automation in very effective manner with ***clean directory structures***.
* Roles are set of tasks and additional files for a certain role which allow you to break up the configurations.
* It can be easily reusing the codes by anyone if the role is suitable to someone.
* It can be easily modified and will reduce the syntax errors.

**ANSIBLE ROLES CREATION STEPS:**

To create an Ansible role, use "ansible-galaxy" command which has the templates to create it.

**# Switch to Ansible User**

$ sudo su – ansible

**# Go to Ansible Home directory to create a new folder 'roles'**

$ cd /home/ansible

$ mkdir roles

$ cd roles

**# Create a new Playbook with a name**

$ ansible-galaxy init apache

* Here, ansible-galaxy is the command to create the roles using the templates.
* init is to initialize the role.
* apache is the name of the role.

If you get error while executing above init command like: ERROR! Unexpected Exception, this is probably a bug: [Errno 13] Permission denied: b'./apache'

**$ sudo chown -R ansible:ansible /etc/ansible**

**# List out the directory created under apache**

$ sudo yum install tree -y

$ tree apache

We have got the clean directory structure with the ansible-galaxy command. Each directory must contain a main.yml file, which contains the relevant content.

**CREATEING ROLE SETUP FOR APACHE WEB SERVER**

Below is a sample playbook to deploy Apache web server. Let’s convert this playbook code into Ansible role.

---

- name: Using Handlers based on tasks execution

  hosts: webservers

  become: true

  gather\_facts: yes

  vars:

    package\_name: httpd

  tasks:

    - name: install httpd

      yum:

        name: "{{package\_name}}"

        state: present

      tags:

        - install

      notify: Start Apache

    - name: Copy index.html

      copy:

        src: index.html

        dest: /var/www/html/

      tags:

        - copy

      notify: Restart Httpd

  handlers:

    - name: Start Apache

      service:

        name: "{{package\_name}}"

        state: started

        enabled: true

    - name: Restart Httpd

      service:

        name: "{{package\_name}}"

        state: restarted

**First, move on to the Ansible roles directory and start editing the yml files.**

$ cd roles/apache

**1. Tasks**

Edit main.yml available in the tasks folder to define the tasks to be executed.

**$ sudo vi tasks/main.yml**

---

# tasks file for /etc/ansible/roles/apache

- name: install httpd

  yum:

    name: "{{package\_name}}"

    state: present

  tags:

    - install

  notify: Start Apache

- name: Copy index.html

  copy:

    src: index.html

    dest: /var/www/html/

  tags:

    - copy

  notify: Restart Httpd

**2. Files**

Copy required files into files directory or create index.html file with content

**$ sudo vi files/index.html**

<h1>WELCOME TO ANSIBLE ROLES EXAMPLE</h1>

:wq (save and quit)

**3. Handlers**

Edit handlers main.yml to restart the server when there is a change. Because we have already defined it in the tasks with notify option. Use the same name restart-apache within the main.yml file as below.

**$ sudo vi handlers/main.yml**

---

# handlers file for /etc/ansible/roles/apache

- name: Start Apache

  service:

    name: "{{ package\_name }}"

    state: started

    enabled: true

- name: Restart Httpd

  service:

    name: "{{ package\_name }}"

    state: restarted

**4. Variables**

Edit vars main.yml to add our variables.

**$ sudo vi vars/main.yml**

# vars file for /etc/ansible/roles/apache

package\_name: httpd

**5. CREATE RUNNING SETUP YAML**

We have got all the required files for Apache roles. Let’s apply this role into the ansible playbook “runsetup.yml” as below to deploy it on the client nodes.

**$ vi /home/ansible/runsetup.yml**

---

- hosts: all

  become: true

  roles:

   - apache

# Execute playbook which contains apache role.

**$ ansible-playbook runsetup.yml**

If you have created multiple roles, you can use the below format to add them in the playbook.

---

- hosts: all

  become: true

  roles:

   - apache

   - jenkins

   - java

   - maven

   - sonar