Infosys® | Education, Training and Assessment

NAVIGATE YOUR NEXT

# Ansible Fundamentals

Infosys®
Navigate your next

## Course Information

Course Code: TIM5711

Course Name: Ansible Fundamentals

Document Number:

Version Number: 2.0

# Day 1 Session Plan

| | |
|---|---|
| 09:30 | In this topics we will be discussing about what is ansible, features, architecture. |
| 10:30 | In this topics we will be discussing about Installing ansible, agent-less architecture. |
| 11:30 | In this topics we will be discussing about ansible inventories, how to create static & dynamic inventory. |
| 14:00 | In this topics we will discussing about Ansible modules, modules types. |
| 16:00 | In this topics we will discussing about ansible ad-hoc commands & examples. |
| 17:00 | In this topics we will discussing about demo for ansible ad-hoc commands with modules. |

## Prerequisites

- Basic Knowledge in Linux administration :
  - Text editing.
  - SSH connections.
  - Basic knowledge on networking and firewall.
  - Basic Linux administration commands.

## Ansible

- Ansible is an open source IT Configuration Management, Deployment & Orchestration tool.
- It aims to provide large productivity advantages to a wide variety of automation challenges.
- This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.

- What can Ansible Do?

  Provisioning.

  Configuration Management.

  IT Automation.

## Ansible Features

- Open Source
- Agent-less Client Configuration
- Built on top of python
- SSH for secure connections
- Easy to Install and Configure

## Ansible Features (contd)

### Push Based Vs Pull Based

- Puppet and Chef are Pull Based

  Agent on the server periodically check for the Configuration information from central server.

- Ansible is push based

  Central server pushes the configuration information on target server.

## Architecture

- Ansible uses a server-client architecture
- Server is called the Ansible Control Server
  - Central server has to be a Linux system.
  - Ansible Control Server  application is Python.
- Clients are called nodes
  - Can be Linux, Unix (BSD, Solaris, HP-UX,MAC) or Windows systems.
  - Run as agent less client  configuration.

## Installing the Ansible Central Server

Infosys® Navigate your next

## Installing the Ansible Central Server

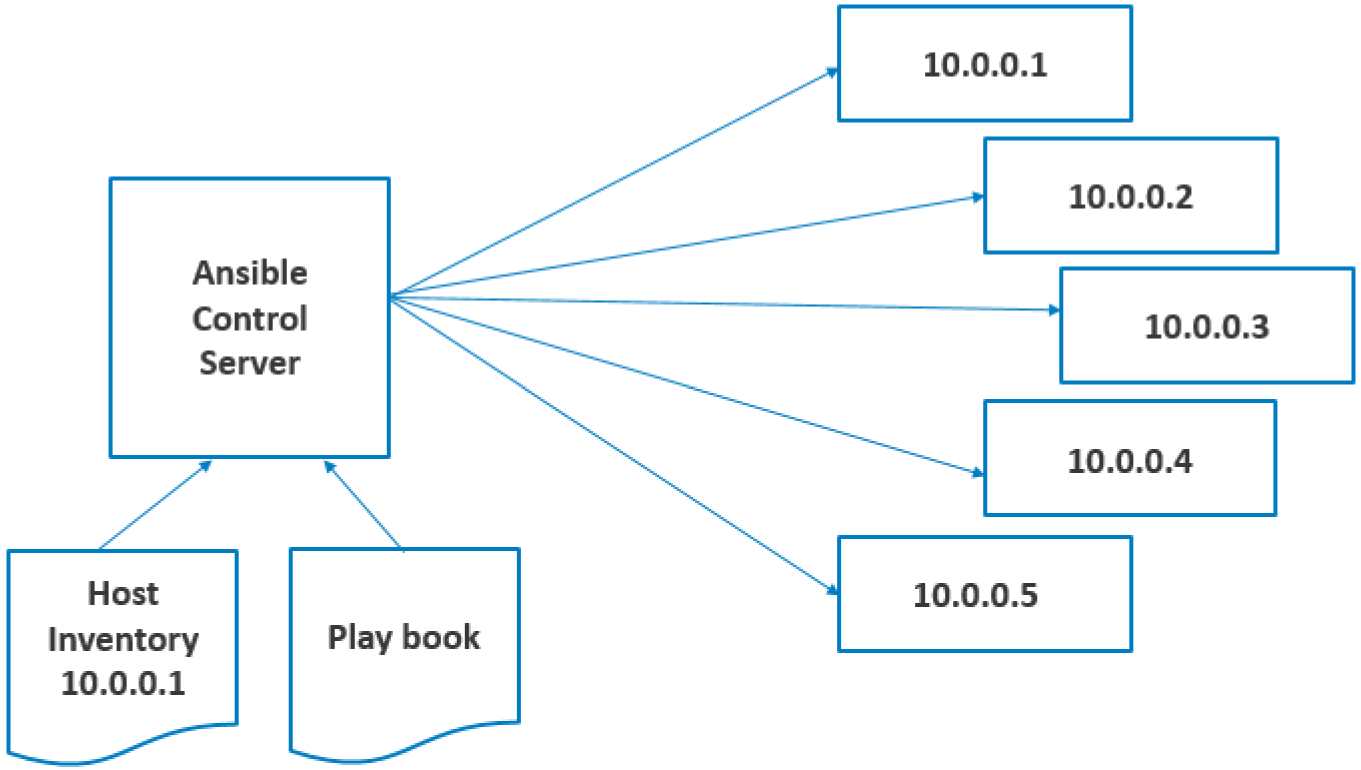
## Requirements

- Linux (RHEL / CENTOS/ ORACLE Linux ) OS
- 4GB of RAM
- 40GB free space in hard disk
- Enable EPEL Repository for Centos 6.8 or 7.2 with Internet Connection.
- SSH for secure connections(Open SSH)

## Requirements for the Node(Agent-less)

- 2GB RAM.
- 20GB space in hard disk.
- RHEL/CENTOS/UBUNTU/ORACLE LINUX/MAC/BSD/Solaris/Windows OS.
- SSH for secure connections(Open SSH).

## Ansible Agentless Architecture



Infosys® Navigate your next

## Ansible Management Node

- Public Key  Mechanisms of the management node is transferred via SSH to the client nodes and then the connection is set.
- Ansible uses the PUSH method to push the configurations to the client nodes.

## Introduction to Ansible  Components

## Introduction to Ansible Components

## Ansible Inventories

- Inventory files contain the list of nodes that you need to manage.
- There are two types of inventories:
    1. Static Inventories.
    2. Dynamic Inventories.

## Static Inventories

- Default inventory file located as /etc/ansible/hosts.
- Inventory file shall be in any of the formats, depending up on the inventory modules.
- Format for /etc/ansible/hosts is an INI like

```
#sample Inventory file
   server01.ims.com
   [webservers]
   server02.ims.com
   server03.ims.com
   [dbgroup]
   db[01:03].cis.com
```

# Ansible Inventory Parameters

- The Following Variables Controls how Ansible interacts with hosts
- ansible_host

  inventory parameter to  specify the FQDN or IP address of the server.

```
web  ansible_host=server01.ims.com
db    ansible_host=server02.ims.com
mail  ansible_host=server02.ims.com
```

- ansible_connection

  --  Inventory parameter defines how Ansible connect to the target servers

```
server01.ims.com  ansible_connection=ssh
```

## Ansible Inventory Parameters (contd)

- ansible_user

  Inventory parameter defines the users makes used to connect  the remote connection.

  ```
  Server01.ims.com  ansible_user=ansadm
  ```

- ansible_port

  Inventory parameter defines which port connect to the remote machine.

  ```
  Server01.ims.com  ansible_port=22
  ```

- ansible_ssh_pass

  Inventory parameter defines the ssh password for Linux.

  ```
  Server01.ims.com  ansible_ssh_pass=P@ss
  ```

## Dynamic Inventory

- It Contain dynamic list of managed node
- The file must be executable file.
- Output of executable is expected to be in a JSON format
- File should provide a list of node if invoked with --list
- File should provide host information if invoked with --host (HOSTNAME)

## Introduction to Ansible Modules

## Introduction to Ansible Modules

Infosys® Navigate your next

## Ansible Modules

- Modules can be used to perform particular tasks that needs to be accomplished.
- Ansible ships with many of the modules you require and they can be used with the Ansible ad-hoc command or through Ansible plays and playbooks
- Some of the Modules are:

  System

  Commands

  Files

  Copy

  Yum

  Service

  Cloud

  Database & More

## Ansible Modules (contd)

- $ansible-doc -l | more

  list out all the module parameter in a particular ansible version.

- $ansible-doc-l | wc-l

  list out total number of modules in particular ansible version.

- $ansible-doc –s yum

  To check the particular yum module function.

## Ansible Ad-hoc Commands

- Ansible ad-hoc commands can be used to perform quick tasks you need to get done.

- Syntax for Ad-hoc command

```
ansible <host-group> [options].
```

## Ansible Ad-hoc Commands (contd)

- # ansible webservers –m ping

  To check Network connectivity for all webservers group machines.

- # ansible all –m ping

  To check Network connectivity for all groups or all the machines or all available groups.

- # ansible all –m ping –o

  Output display in the single line.

- # ansible all –m shell –a “uname –a; df –h”

  To check information about currently mounted devices.

## Ansible Ad-hoc Commands (contd)

- # ansible all –m shell –a “uname –a; df –h” –v

  To display progress in the ansible command.

- Shell Module

  It used to run any command in Ansible Module.

- #ansible appgroup –m yum –a “name=nmap state=present”-s

  To install the httpd package for appgroup group machines.

- #ansible appgroup –m service –a “name=nmap state=started” -s.

  To start the httpd service using with service module.

- #ansible all –m copy –a “src=/tmp/testingfile dest=/tmp/testingfile” –s

  To Copy the file from one machine to another machine.

## Day 2 Session Plan

| | | | |
|---|---|---|---|
| 09:30 | In this topics we will be discussing about Introduction to playbooks, how to create the playbooks . | 14:00 | In this topics we will discussing about Ansible loops & types of loops. |
| 10:30 | In this topics we will be discussing about how to managing variables in playbook . | 16:00 | In this topics we will discussing about ansible conditionals. |
| 11:30 | In this topics we will be discussing about how to managing ansible facts & host group variables. | 17:00 | In this topics we will discussing about demo for ansible handlers. |

## Introduction to Ansible Playbooks

## Introduction to Ansible playbooks

## Ansible Playbooks

- Ansible play can be a specific task that can be  performed inside a playbook and a playbook contains one or more plays.

- Ansible Playbooks describe a set of steps in a process.

- Ansible playbooks are way to send command to remote computers in a scripted way.

- Playbooks can be used to manage configuration and deployment to remote machines.

- Playbooks can be written yaml format.

## Ansible Playbooks (contd)

- Playbooks → single YAML file.
- Has the .yml extension.
- Play is defines a set of activities (task) to be run on hosts.
- Task is action to be performed on the host.

  Execute a command.

  Run a script.

  Install a Package.

  Shutdown/Restart.

## Ansible Playbooks (contd)

- #sample playbook file for web-server
-

```
---
 - hosts: all
   tasks:
   - name: Install httpd package.
     yum: name=httpd state=latest
   - name: Start & Enable httpd service
     service: name=httpd state=started enabled=yes
   - name: Copy the html file
     copy: src=/tmp/index.html dest=/var/www/html/index.html
```

## Ansible Playbook (contd)

- Syntax:

  ansible-playbook  -i <External-inventory file name> <playbook filename>

- #ansible-playbook –help

  To display the additional parameter of this command.

- #ansible-playbook <playbook filename> --syntax-check

  --To verify the playbook

## Managing Variables in Ansible Playbook

Infosys® Navigate your next

## Managing Variables in Ansible playbook

## Managing Variables

- Ansible supports variables that  can be used to store values or that can be re-used through out files in an entire ansible project

- Variables provide a beneficial way to manage dynamic values for a given environment in your ansible project.

- Examples of variables that contain values.

  1. To Create user.
  2. To install packages
  3. To re-start services.
  4. To remove files.

## Managing Variables(contd)

- Variables Name.

Variables are defining  with names which consists of a string should start with letters and also containing with letters, numbers, underscores.

- Defining Variables.

Variables can be defined in variety of section in an ansible project. This can be defined in to three levels.

1. Global level Variable.
2. Playbook level Variable
3. Inventory level Variable

## Managing Variables(contd)

- Global  level Variable.

  Variables are assigned from the command-line interpreter or ansible configuration.

- Playbook level Variable.

  Variables are assigned in the play or tasks and some related structures.

- Inventory level Variable.

  Variables are assigned on individual hosts, group of hosts inside the inventory file, ansible fact gathering or registered tasks.

## Managing Variables(contd)

- Variables using playbooks.

  when creating playbooks, admin can define the own variables and directly call them in a task.

Example

variable web_vars1 can be defined with a value of httpd and invoked by yum module in-order to install the httpd package.

## Managing Variables(contd)

```
#Sample playbook for using variables
---
- hosts: webgroup
   vars:
          web_vars1: httpd
           firewall_pkg1: firewalld
    tasks:
         - name: Install the require package
            yum: name=“{{web_vars1}}” state=latest
```

## Managing Variables in Ansible Facts

## Managing Facts

- Facts are special variables that are automatically created by ansible tool from managed node.

- Facts are invoked by the setup module and its contain helpful information stored in to variables so that system administrator can re-utilize the variables.

- Facts can be using in playbook, roles, conditionals, loops, templates that depends on a value for managed node.

- Example:
- Server can be re-started depending on the os version.
- Users can be created depending on the hostname.

## Managing Facts (contd)

- Ansible facts are helpful approach to retrieve the state of a managed host and decide which move to take based on its state.

- Facts provide some information:

  1. Information about the hostname.
  2. Information about the kernel version.
  3. Information about the network interfaces.
  4. Information about the IP address.
  5. Information about the version of the operating system

## Managing Facts (contd)

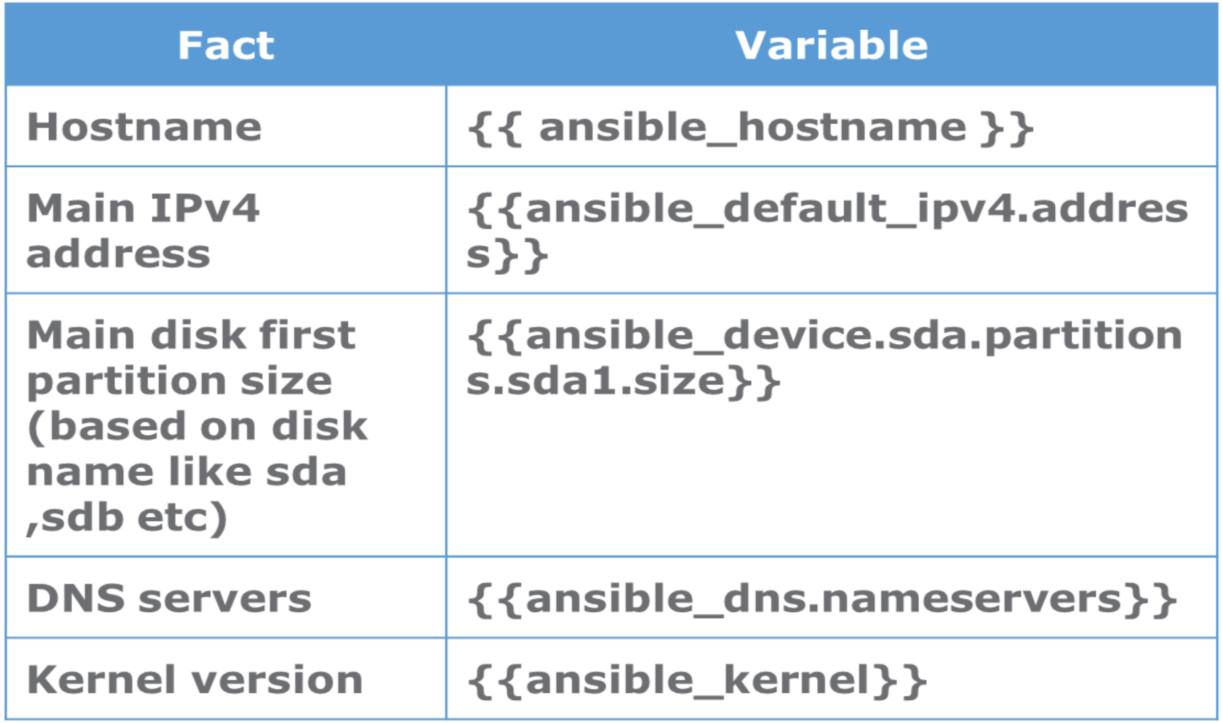
- The following command display of the fact variables information gathered from managed host.

```
$ ansible webserver –m setup
```

Note: output is displayed in JSON format. Variables values stored in a python dictionary.

## Managing Facts (contd)

- Ansible facts



| Fact | Variable |
|---|---|
| Hostname | {{ ansible_hostname }} |
| Main IPv4 address | {{ansible_default_ipv4.address}} |
| Main disk first partition size (based on disk name like sda ,sdb etc) | {{ansible_device.sda.partitions.sda1.size}} |
| DNS servers | {{ansible_dns.nameservers}} |
| Kernel version | {{ansible_kernel}} |

## Managing Facts (contd)

- To write a playbook using facts variables.

```
---
- hosts: webgroup
  tasks:
      - name: Print hostname,IPv4 for managed host
        debug:
         msg: >
             The default IPv4 address of {{ ansible_hostname}} is {{ansible_default_ipv4.address}}
```

## Managing Facts (contd)

Facts filters

- Ansible facts contains broad data about the system.
- Administrator will be using ansible filters in order to fetch a particular information when gathering facts from a managed host.

Examples for ansible filters

```
$ansible all –m setup –a  “filter=ansible_eth0“
$ansible all –m setup –a “filter=ansible_kernel”
$ansible all –m setup –a “filter=ansible_dns”
```

## Managing Facts (contd)

Custom facts

- Administrators can make their own facts and push them to a managed node. After create the custom facts will be integrated and perused by the setup module.

- Custom facts are stored in the below directory path

  /etc/ansible/facts.d

- Facts should be in .fact as an extension.

- Facts file is a plain-text file.(either INI or JSON format)

## Managing Facts (contd)

- Example 1

An INI facts document contains a top level characterized by a segment , followed by the key-value sets for the facts to define.

```
     [packages]
webpackage1 =  httpd
ftppackage1 = vsftpd
     [users]
user1 = john
```

## Managing Facts (contd)

- Example 2

If provided in JSON then following syntax should be utilized.

```
{
   “packages”: {
            “webpackage1”: “httpd”,
             “ftppackage1”:  “vsftpd”
   }
}
```

For the two configurations, the outcome returned by Ansible is the same and will be placed in the ansible_local level

## Managing Facts (contd)

- To ensure custom facts have been successfully installed and retrieved the information.

```
$ ansible all –m setup –a filter=“ansible_local”
```

- Custom facts can be utilized the same way as default facts in playbook.

```
---
- hosts: all
   tasks:
   debug: msg=The package to install on {{ansible_hostname}} is
{{ansible_local.custom.packages.webpackage1}}
```

## Host and Group Variables

- Inventories variables are directly apply to managed node.
- It can be classified in to two categories.

  1. Host Variables.
  2. Group Variables.

- Host Variables.

  It is directly assigned to a particular host.

- Group Variables.

  It is directly assigned to all host in a host group or  group of host

## Host and Group Variables(contd)

- Example for Host Variables.

The host variable ansible_user is being used for the host server01.cis.com.

```
  [servers]
 server01.ims.com ansible_user=John
```

- Example for Group Variables.

The group variable user is being used for the group of hosts.

```
    [servers]
  server01.cis.com
  server02.cis.com
    [servers: vars]
  user= john
```

Note: Host and Group variables are directly assigned to inside the inventory file.

## Host and Group Variables(contd)

- Example for Group Variables

The group variable user is being used for the group of hosts which consists of two hosts groups.

```
  [servers1]
sever01.cis.com
   [servers2]
server02.cis.com
   [servers: children]
        servers1
        servers2
```

## Introduction to Ansible Loops

## Ansible loops

- Ansible supports various loops format in-order to repetitive over set of values defined in lists.

- With helpful of loops administrators can saves for writing repetitive tasks that utilize the same module.

- Examples for using loop.
  1. To create the multiple users.
  2. To install the multiple packages.

- To passing a loop as an argument with the **item** keyword should be utilized for ansible to parse the array or list.

## Ansible loops (contd)

- Ansible loops can be used for three categories.

    1. Simple loops
    2. List of hashes
    3. Nested loops

- Simple loops

    It is set of values that ansible peruses and repeat over.

    It characterized by providing a list of values to the **with_items** keyword.

## Ansible loops (contd)

- Example for install the packages using yum module twice in order.

```
- name: To install multiple packages using with yum
  yum:
      name: postfix
      state: present
  yum:
      name: dovecot
      state: present
```

## Ansible loops (contd)

- The two similar tasks utilizing the yum module can be re-written with a simple loop so that a single task must to install both packages.

```
yum: name=“{{ item }}” state=latest
with_items:
        - postfix
        - dovecot
```

The past code can be replaced by having the packages inside an list with the **with_items** keyword

## Ansible loops (contd)

```
vars:
      mail_packages:
            - postfix
            - dovecot

tasks:
       yum: name=“{{ item }}” state=installed
       with_items: “ {{ mail_packages }}”
```

## Ansible loops (contd)

- List of Hashes

  when passing list or array as argument that list or array can be a list of hashes.

  It is multi-dimensional array (an array with key-pair values) is passed to the user module in order to customize both name of the user and group.

# Ansible loops (contd)

```
user:
    name: “{{ item.name }}”
    state: present
    group: “{{ item.groups }}”

    with_items:
      - { name: ‘john’  , groups: ‘admin’ }
      - { name: ‘ethan’ , groups: ‘sales’ }
```

Infosys®
Navigate your next

## Ansible loops (contd)

- Nested loops

  It is noting but which are loops inside of loops with the **with_nested** keyword.

  when nested loops utilized, ansible repeats over the first list or array as long as there are values in it.

  Example

The multiple MySQL privileges are required for multiple users.

Administrator create a multi-dimensional array or list and invoke using with **with_nested** keyword.

## Ansible loops (contd)

- mysql_user module in a nested loop to grant two users a set of three privileges.

```
name:  “{{  item [0] }}”
priv: “{{ item [1] }}”.*:ALL”
append_privs: yes
password: Redhat
with_nested:
 - [  ‘john’ , ‘ethan’ ]
 - [  ‘Clientdb’ ,’employeedb’, ‘providerdb’ ]
```

## Introduction to Ansible Conditionals

## Ansible Conditionals

- Ansible can utilize conditionals to execute tasks or plays when certain conditions are met.
- Example for conditional can be utilized to decide the accessible memory on a managed node before ansible installs or configure a service.
- Playbook variables, registered variables and ansible facts would all be able to be tested with conditionals.
- Operators are used in conditionals such a s string comparison, mathematical operators, and Booleans are available.

## Ansible Conditionals (contd)

- Conditionals Operators

| Operator | Example |
|---|---|
| Equal | “{{ max_memory }} ==512” |
| Less than | “{{ min_memory }} < 256” |
| Greater than | “{{ min_memory }} > 256” |
| Less than or equal to | “{{ min_memory }} < =256” |
| Greater than or equal to | “{{ min_memory }} >=512” |
| Not equal to | “{{ min_memory }} !=512” |
| Variable exists | “{{ min_memory }} is defined” |
| Variable does not exist | “{{ min_memory }} is not defined” |

# Ansible Conditionals (contd)

- Note

“**==**“ operator used to test equality conditions must not be confused with the “**=**“ operator used for assigning a value to variable.

For Example.

1. **myvlaue=“this is my variable”** (it assigned the value “**this is my variable**” to the variable myvalue.

2. **{{ myvalue }} == “this is my variable”** test the value of the variable **myvalue** to check whether it has been allocated the string **“this is my variable”**

## Ansible Conditionals (contd)

- When Statement

    To implement a conditional on a component then **when** statement must utilized to test the condition.

    The statement when is present it will evaluate condition before executing the task.

## Ansible Conditionals (contd

- Example for single conditions

```
tasks:
 - name: Install the HTTPD Package
   yum: name= httpd  state=present
   when: ansible_distribution == “RedHat”
```

The following code is a basic implementation of a when statement before install the httpd package ansible must ensure the managed host is part of the **Redhat** family

## Ansible Conditionals (contd)

- Example for multiple conditions.

  when statement can be utilized to test multiple values.

  conditionals can be combined with and  & or keywords or grouped with parentheses.

```
- name: To install mariadb package if enough space on root
  yum: name=mariadb-server state:present
  with_items: “{{ ansible_mounts }}”
  when: item.mount == “/” and item.size_available > 300000000
```

## Introduction to Ansible Handlers

## Ansible Handlers

- Handlers are tasks that react to a notification triggered or activated by different tasks.
- Each handler have a globally-unique name and is triggered at the end of a block of tasks in a playbook.
- If the tasks not notifies by the handler name, it will not run.
- If one or more tasks notify by the handler, it will run exactly once after all other tasks in the play have completed.
- Handlers are utilized to reboot hosts and restart services.

## Ansible Handlers (contd)

- Example for handlers.

```
 tasks:
  - name: copy  the httpd.conf.j2 configuration file template
    copy: src=/tmp/httpd.conf.j2 dest=/etc/httpd/conf/httpd.conf
    notify
          - restart_apache
    handlers:
           - name: restart_apache
    service: name=httpd state: restarted
```

# Day 3 Session Plan

| | |
|---|---|
| 09:30 | In this topics we will be discussing about Introduction to roles . |
| 10:30 | In this topics we will be discussing about how to create roles & using roles in playbooks . |
| 11:30 | In this topics we will be discussing about ansible template. |
| 14:00 | In this topics we will discussing about ansible tags. |
| 16:00 | In this topics we will discussing about ansible vault. |
| 17:00 | In this topics we will discussing about demo for ansible tower. |

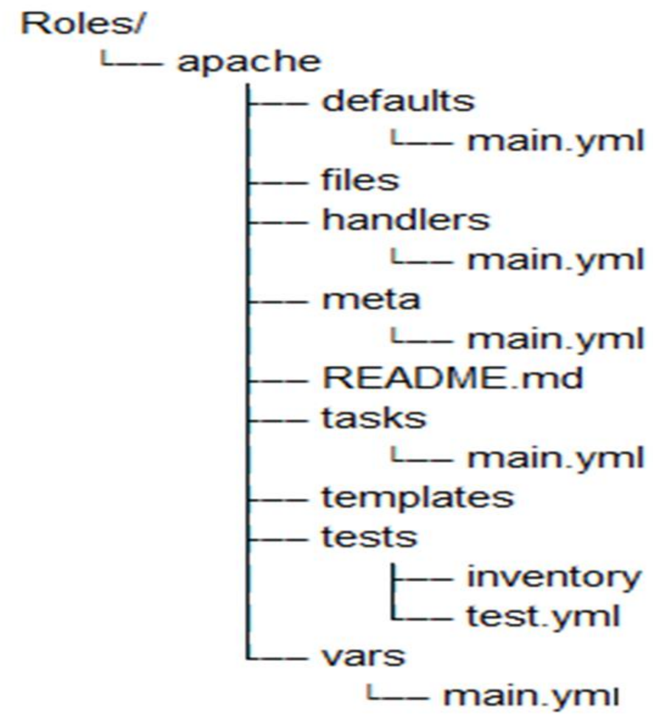
## Introduction to Ansible Roles

## Ansible Roles

- Roles are help us to organize the project in a standard structure.
- Roles can be thought of as playbook that’s split up into multiple parts.
- Roles must be in a particular format to works as expected.
- #ansible-galaxy command used to  create the roles in correct format

## Ansible Roles(contd)

- The Format of a Apache Roles as follows



```
Roles/
    └── apache
            ├── defaults
            │       └── main.yml
            ├── files
            ├── handlers
            │       └── main.yml
            ├── meta
            │       └── main.yml
            ├── README.md
            ├── tasks
            │       └── main.yml
            ├── templates
            ├── tests
            │       ├── inventory
            │       └── test.yml
            └── vars
                    └── main.yml
```

# Ansible Roles (contd)

- Playbook using that role would look like the following

```
---
 - hosts: local
   become: yes
   roles:
          -Roles\apache
```

## Introduction to Ansible Templates

Infosys®
Navigate your next

## Ansible Templates

- Ansible Template using template module.
- Ansible uses Jinja2 templating to enable dynamic content or expression and access to variables.
- Template module can take variables that you have defined and replace those in files.
- It used to replace the information & send that information to the target server.
- Templates are processed by the Jinja2 templating language.
- Documentation about this language can be found be here http://jinja.pocoo.org/docs

## Ansible Template (contd)

- Example for Template

Template file called temp1.j2

```
Hi Hello there <p>
ServerName = “{{description}}”
```

Sample playbook use this template

```
---
 - hosts: databases
   become: yes
   vars:
       description : “{{ ansible_hostname}}”
```

## Ansible Template (contd)

```
tasks:
              -name: write the index file
               template:src= temp1.j2dest=/var/www/html/index.html
               notify:
                      - restart httpd
              -name: ensure apache is running
                service: name=httpd state=restarted
```

## Ansible Template (contd)

- The contents of the /var/www/html/index.html file once the playbook has run

```
<p>
  Hi Hello there <p>
  ServerName = server01
```

- Particular machine hostname is ‘server01’

## Ansible Tags

## Ansible Tags

- Ansible Tags is used to execute the particular play or task inside the playbook.
- It mainly used to be able to run a specific part of the configuration without running the whole playbook.
- Example of tags:

```
tasks:
  - template: src=templates/src.j2 dest=/etc/foo.conf
     tags:
          - configuration
```

If you wanted to run configuration part of a very long playbook

```
#ansible-playbook playbook1.yml –tags “configuration
```

## Ansible Vault

Infosys® Navigate your next

## Ansible Vault

- Ansible Vault can be used for encrypting/decrypting the confidential information stored in the playbooks. .
- It's used for encrypt critical information such as password, variables, SSH keys etc.
- It uses AES-256 encryption algorithm.
- ansible-vault command is used to encrypt/decrypt the files .
- When using an encrypted file in a playbook, you need to use the following options when running the playbook:
- --ask-vault-pass
- --vault-password-file

## Ansible Vault (Contd)

- # ansible-vault view command to view the content of the encrypted file.
- # ansible-vault create to create an encrypted file.

## Ansible Tower

Infosys®
Navigate your next

## Ansible Tower

- Ansible Tower is a web-based solution and designed to help manage your ansible.
- Ansible Tower provides access control over your play-books, ssh-credentials, invenories.
- It helps in monitoring the systems.
- Ansible Tower is free for usage for up to 10 nodes.

THANK YOU

Infosys® Navigate your next