Conditionals in Ansible:

$ ansible all -m setup | grep os\_family

$ ansible all -m setup | grep distribution

$ nano conditionals.yaml

---

- hosts: all

  become: true

  tasks:

    - name: Execute only if the os\_family is Debian

      debug:

        msg: "This target machine is Debian..."

      when: ansible\_os\_family == "Debian"

    - name: Execute only if the os\_family is Ubuntu

      debug:

        msg: "This target machine is Ubuntu..."

      when: ansible\_os\_family == "Ubuntu"

    - name: Execute only if the os\_family is Debian alternative

      debug:

        msg: "This target machine is Debian alternative..."

      when: ansible\_facts['os\_family'] == "Debian"

    - name: Testing whether the machine is running Ubuntu & version 22

      debug:

        msg: "This machine is running Ubuntu 22"

      when: ansible\_facts['distribution'] == "Ubuntu" and ansible\_facts['distribution\_major\_version'] == "22"

    - name: Testing whether the machine is running Ubuntu & version 14 and above

      debug:

        msg: "This machine is running Ubuntu version 14+"

      when: ansible\_facts['distribution'] == "Ubuntu" and ansible\_facts['distribution\_major\_version'] | int >= 14

    - name: Testing whether the machine is Ubuntu or CentOS

      debug:

        msg: "This machine is running Ubuntu or CentOS"

      when: ansible\_facts['distribution'] == "Ubuntu" or ansible\_facts['distribution'] == "CentOS"

$ ansible-playbook conditionals.yaml

$ nano test\_register.yaml

---

- hosts: all

  tasks:

    - name: Check the file exists in path

      stat:

        path: ~/demo.txt

      register: file\_status

    - name: Print the execution output

      debug:

        msg: "{{ file\_status }}"

    - name: Create file if it doesn't exist

      file:

        path: ~/demo.txt

        state: touch

      when: not file\_status.stat.exists

    - name: Excute if the first task is success

      debug:

        msg: "My new task execution if the first task execution is successful"

      when: not file\_status.failed

    - name: Excute if the first task is failure

      debug:

        msg: "My new task execution if the first task execution failed"

      when: file\_status.failed

$ ansible-playbook test\_register.yaml

$ nano multihandler.yaml

---

- hosts: all

  gather\_facts: no

  tasks:

    - name: "task 1"

      command: hostname

      notify:

        - my handler

        - my handler1

    - name: "task 2"

      command: hostname

      notify:

        - my handler

    - name: "task 3"

      command: hostname

  handlers:

    - name: my handler

      debug:

        msg: "My handler called..."

    - name: my handler1

      debug:

        msg: "My handler1 called..."

$ ansible-playbook multihandler.yaml

$ nano flushhandler.yaml

---

- hosts: all

  gather\_facts: no

  tasks:

    - name: "task 1"

      command: hostname

      notify:

        - my handler1

    - name: "task 2"

      command: hostname

      notify:

        - my handler2

    - name: "Flush all handlers"

      meta: flush\_handlers

    - name: "task 3"

      command: hostname

      notify:

        - my handler3

        - my handler1

  handlers:

    - name: my handler1

      debug:

        msg: "My handler1 called..."

    - name: my handler2

      debug:

        msg: "My handler2 called..."

    - name: my handler3

      debug:

        msg: "My handler3 called..."

$ ansible-playbook flushhandler.yaml

Ansible Filters:

$ nano filters\_playbook.yml

---

- name: Demo Ansible filters

  hosts: all

  gather\_facts: no

  vars:

    sample\_list: [1, 2, 3, 4, 5]

    sample\_string: "my sample text"

    sample\_dict:

      key1: "value1"

      key2: "value2"

  tasks:

    - name: Convert list to comma-separated string

      debug:

        msg: "{{ sample\_list | join(',') }}"

    - name: Captitalize string

      debug:

        msg: "{{ sample\_string | capitalize }}"

    - name: Convert dict to list of keys

      debug:

        msg: "{{ sample\_dict | dict2items | map(attribute='key') | list }}"

    - name: Sum of a list

      debug:

        msg: "{{ sample\_list | sum }}"

    - name: Sort a list

      debug:

        msg: "{{ sample\_list | sort(reverse=True) }}"

    - name: Check if item is in list

      debug:

        msg: "{{ 'yes' if 3 in sample\_list else 'no' }}"

$ ansible-playbook filters\_playbook.yml

Filters in Ansible are tools used within Jinja2 templates to transform, format, and manipulate data in playbooks. They allow you to modify variables, apply conditional logic, and customize output, making your automation tasks more flexible and efficient. Filters help keep playbooks clean and readable by simplifying complex data operations.

------

Handlers in Ansible are special tasks that are triggered by other tasks using the notify directive. They are typically used to perform actions like restarting services or reloading configurations, but only run when notified, ensuring changes are applied efficiently without unnecessary repetitions. Handlers run at the end of a play, after all tasks are completed, unless explicitly triggered earlier using the flush\_handlers directive.

Jinja2 Setup:

$ sudo apt update -y

$ sudo apt install -y python3-jinja2

Verify the Jinja2 installation:

$ python3 -c "import jinja2; print(jinja2.\_\_version\_\_)"

Check if a package is installed or not:

$ dpkg-query -l | grep python3-jinja2

Accessing Variables in Jinja2:

- Create a template file:

$ nano template.j2

Hello, {{ name }}!

You have {{ notifications }} new notifications.

- Create a python script to render the template:

$ nano render\_template.py

from jinja2 import Environment, FileSystemLoader

# Load the template from the file system

file\_loader = FileSystemLoader('.')

env = Environment(loader=file\_loader)

# Specify the template file

template = env.get\_template('template.j2')

# Define the variables

context = {

  'name': "Abhi",

  'notifications': 5

}

# Render the template with the variables

output = template.render(context)

print(output)

- Execute the python script

$ python render\_template.p

Jinja2 Filters with Ansible:

$ nano filter\_ansible.yml

---

- name: Data Manipulation

  hosts: all

  gather\_facts: no

  vars:

    my\_name: joHn daVis

    first\_name: John

  tasks:

    - name: Print name

      debug:

        msg:

          - "My name is {{ my\_name }}"

          - "My name lower is {{ my\_name | lower }}"

          - "My name upper is {{ my\_name | upper }}"

          - "My name capitalize is {{ my\_name | capitalize }}"

          - "My name title is {{ my\_name | title }}"

    - name: Print message

      debug:

        msg:

          - "My --- name --- is {{ first\_name }} {{ last\_name | default('Davis') }}"

    - name: Filter chaining

      debug:

        msg: "My new name - {{ my\_name | title | replace('n','nny') }}"

$ ansible-playbook filter\_ansible.yml

Jinja2 Loops & Conditionals:

- Create variable file

$ nano my\_data.yml

products:

  - name: Laptop

    price: 1000

    in\_stock: true

  - name: Smartphone

    price: 500

    in\_stock: false

  - name: Tablet

    price: 300

    in\_stock: true

discount: 10

- Create template file

$ nano my\_template.j2

Product List:

{% for product in products %}

  - {{ product.name }} for {{ product.price }}

    {% if product.in\_stock %}

       ({{ discount }}% off)

    {% else %}

       (Out of stock)

    {% endif %}

{% endfor %}

$ nano my\_render.py

import yaml

from jinja2 import Environment, FileSystemLoader

#Load data from the YAML file

with open('my\_data.yml', 'r') as file:

  data = yaml.safe\_load(file)

#Set up Jinja2 Env

env = Environment(loader=FileSystemLoader('.'))

template = env.get\_template('my\_template.j2')

#Render template with data

rendered\_text = template.render(data)

#Print or save the rendered text

print(rendered\_text)

with open('my\_rendered\_output.txt', 'w') as output\_file:

  output\_file.write(rendered\_text)

- Execute the python script

$ python my\_render.py

$ cat my\_rendered\_output.txt

- Render using Ansible playbook:

$ nano my\_render\_playbook.yaml

---

- hosts: all

  vars\_files:

    - my\_data.yml

  tasks:

    - name: To render file from template

      template:

        src: my\_template.j2

        dest: ~/my\_playbook\_render\_output.txt

$ ansible-playbook my\_render\_playbook.yaml

$ cat my\_playbook\_render\_output.txt

Ansible Roles:

Creating Roles:

$ mkdir rolestest

$ cd rolestest

$ mkdir roles

$ cd roles

$ ansible-galaxy init apache\_role

$ tree

$ nano apache\_role/tasks/install.yml

---

- name: Install the apache2 package

  apt:

    name: apache2

    state: present

$ nano apache\_role/tasks/config.yml

---

- name: Copy the index file to apache directory

  copy:

    src: files/index.html

    dest: /var/www/html/

$ nano apache\_role/tasks/service.yml

---

- name: Start the apache service

  service:

    name: apache2

    state: started

$ nano apache\_role/tasks/main.yml

---

- import\_tasks: install.yml

- import\_tasks: config.yml

- import\_tasks: service.yml

$ nano apache\_role/files/index.html

<html>

<head>

  <title>Welcome to My Page</title>

</head>

<body>

  <h1>Hello, this is a sample page created from the Role!</h1>

  <p>Welcome to this new page!</p>

</body>

</html>

$ nano apache\_role/tasks/print\_var.yml

---

- name: Print the variable

  debug:

    msg: "The value for variable message is {{ message }}."

$ nano apache\_role/tasks/main.yml

---

- import\_tasks: install.yml

- import\_tasks: config.yml

- import\_tasks: service.yml

- import\_tasks: print\_var.yml

$ nano apache\_role/vars/main.yml

---

message: "my custom message from VARS directory"

$ nano apache\_role/defaults/main.yml

---

message: "my default message from DEFAULTS directory"

Remove the unwanted directories:

$ rm -rf apache\_role/handlers apache\_role/templates apache\_role/tests

Create a playbook to use the role:

$ cd ..

$ nano setup.yml

---

- hosts: all

  become: yes

  roles:

    - apache\_role

Check the playbook syntax:

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

Execute the playbook:

$ ansible-playbook setup.yml

Browse: <http://localhost/>

Ansible Galaxy:

[https://galaxy.ansible.com/ui](https://galaxy.ansible.com/ui/)

A screenshot of a computer

Description automatically generated

Ansible Galaxy:

$ ansible-galaxy list

Install a role:

$ ansible-galaxy role install zhan9san.test

$ ansible-galaxy list

Remove a role:

$ ansible-galaxy remove zhan9san.test

$ ansible-galaxy list

Remove a role:

$ ansible-galaxy remove zhan9san.test

$ ansible-galaxy list

Ansible Vault - Operations:

Create new encrypted file:

$ ansible-vault create vault1.yml

Password: 123

Content: My confidential info

vi

-> press i to switch to insert mode

-> Esc :wq Enter - save and exit

$ cat vault1.yml

Viewing the encrypted file:

$ ansible-vault view vault1.yml

Edit the encrypted file:

$ ansible-vault edit vault1.yml

$ ansible-vault view vault1.yml

Change the password of the encrypted file:

$ ansible-vault rekey vault1.yml

Vault pass: 123

New vault pass: 12345

Store the password in a file:

$ echo "12345" > my\_pass\_file

Use the password file:

$ ansible-vault view vault1.yml --vault-password-file my\_pass\_file

Encrypt with vault id:

$ echo "My vault test" > vault\_test.txt

$ ansible-vault encrypt --vault-id myid@my\_pass\_file vault\_test.txt

$ cat vault\_test.txt

Use an encrypted file in adhoc command execution:

$ ansible localhost -m copy -a "src=vault\_test.txt dest=~/from\_vault\_test.txt"

$ ansible localhost -m copy -a "src=vault\_test.txt dest=~/from\_vault\_test.txt" --ask-vault-pass

$ cat from\_vault\_test.txt

Implementing Dynamic Inventory:

- Install boto3 & awscli

$ pip install boto3

$ pip install awscli

Verfiy the Installation:

$ aws --version

$ nano dynamic\_ec2.py

#!/usr/bin/env python

import boto3

import json

def get\_ec2\_instances():

  ec2 = boto3.resource('ec2')

  instances = ec2.instances.filter(Filters=[{'Name':'instance-state-name', 'Values':['running']}])

  inventory = {'all':{'hosts':[]}}

  for instance in instances:

    for tag in instances.tags:

      if tag['Key'] == 'Name':

        inventory['all']['hosts'].append(instance.public\_dns\_name)

  return inventory

def get\_dummynodes():

  inventory = {'all':{'hosts':[]}}

  inventory['all']['hosts'].append('localhost')

  inventory['all']['hosts'].append('host1')

  inventory['all']['hosts'].append('host2')

  inventory['all']['hosts'].append('host3')

  return inventory

if \_\_name\_\_=="\_\_main\_\_":

  #inventory = get\_ec2\_instances()

  inventory = get\_dummynodes()

  print(json.dumps(inventory))

$ sudo chmod +x dynamic\_ec2.py

$ python dynamic\_ec2.py

Configure AWS CLI:

$ export AWS\_ACCESS\_KEY\_ID=''

$ export AWS\_SECRET\_ACCESS\_KEY=''

$ ansible-inventory -i dynamic\_ec2.py --list

Launch EC2 Instance - using Management Console:

  - Name: Abhi Instance 01

  - AMI: Ubuntu 22.04 (ami-0a0e5d9c7acc336f1)

  - Instance Type: t2.micro

  - Key Pair: Create new keypair

      Key pair name: demo\_key

      Key pair type: RSA

      Private key file format: pem

  -  Firewall:

      > Create security group

        > Allow SSH traffic from Anywhere

Create an S3 bucket within AWS:

Check terraform Installation:

$ terraform -version

$ mkdir tf\_s3

$ cd tf\_s3

$ nano main.tf

provider "aws" {

  access\_key = "YOUR\_AWS\_ACCESS\_KEY"

  secret\_key = "YOUR\_AWS\_SECRET\_KEY"

  region = "us-east-1"

}

resource "aws\_s3\_bucket" "example" {

  bucket = "my-unique-bucket-${random\_id.bucket\_id.hex}"

}

resource "random\_id" "bucket\_id" {

  byte\_length = 8

}

$ terraform init

$ terraform validate

$ terraform plan

$ terraform apply

$ terraform destroy -auto-approve

HCL functions:

$ cd ..

$ mkdir tf\_functions

$ cd tf\_functions

$ nano main.tf

variable "greeting" {

  default = "Hello"

}

variable "name" {

  default = "World"

}

output "message" {

  value = "${var.greeting}, ${upper(var.name)}"

}

variable "list\_example" {

  default = ["one","two","three"]

}

variable "map\_example" {

  default = {

    key1 = "value1"

    key2 = "value2"

  }

}

output "first\_list\_item" {

  value = "${element(var.list\_example, 0)}"

}

output "map\_value" {

  value = "${lookup(var.map\_example, "key1")}"

}

output "base64\_encoded" {

  value = "${base64encode(var.name)}"

}

$ terraform init

$ terraform validate

$ terraform apply -auto-approve