Getting better with Ansible

Ansible can deal with lot more and complex administration and automation tasks. To deal with this, ansible has many more terms and features. Let us get introduced to the same -

# Ansible plays

Let us assume that we have an inventory file with many groups and hosts. The example is given below -

[dbservers] 192.168.0.11

192.168.0.12

[webservers] 10.0.0.15

10.0.017

We want to run seperate tasks for seperate set of servers. Say, we want to install mysql for dbservers and apache2 for web servers. The playbook for the same would be

---

* hosts: dbservers tasks:
  + name: Install mysql server

apt: pkg=mysql state=present update\_cache=true

* hosts: webservers tasks:
  + name: Install webservers

apt: pkg=apache2 state=present update\_cache=true

We have two hosts section and tasks for dbservers and webserver. Each of them are seperate and are termed as**plays**. We can have multiple plays inside a single playbook.

# Ansible handlers

We have events and need to trigger some action based on the events. For example, we install apache and need to ensure that the service starts soon after installation. A Handler is exactly the same as a Task (it can do anything a Task can), but it will run when called by another Task. A Handler will take an action when called by an event it listens for. A sample handler is given below -

---

* hosts: local tasks:
  + name: Install Nginx

apt: pkg=nginx state=install update\_cache=true notify:

- Start Nginx

handlers:

* + name: Start Nginx

service: name=nginx state=started

This particular Handler uses the Service module, which can start, stop, restart, reload (and so on) system services. Here we simply tell Ansible that we want Nginx to be started.

# Looping in ansible

You need to do some iterative tasks in ansible and hence need them in ansible playbooks. The example playbook is given below -

* name: Ansible Loop example to check for the presence of pakages apt:

name: "{{ item }}" state: present

with\_items:

* + python3
  + ca-certificates
  + git

**with\_items** is used to mention the items using which you want to iterate for simple tasks. The playbook given above will chec for the precence of packages in remote hosts. It will look for the package mentioned in the **with\_items** The special **{{ item }}** is used to replace with the items mentioned in the **items**

# Looping in ansible with if-else combined

We can put conditions like if-else clause in ansible module. See the playbook given below -

* hosts: all tasks:
  + name: Ansible loop with conditional clause debug:

msg: "{{ item }}" with\_items:

* + - "hello1"
    - "hello2"
    - "hello3"

when: ansible\_distribution == "MacOSX"

the above playbook executes only when the target host has MacOSX distribution. If MacOSx, the execute the playboook.

Another playbook given below is an example for the same -

- hosts: all vars:

test1: "Hello World" tasks:

- name: Ansible when variable equals example debug:

msg: "Equals"

when: test1 == "Hello World"

# Using varibles in ansible

Variables are must to write complex scripts and one can. We can declare variables in the playbook using **var** keyword. When the variable is to be called, we use **{{ variable-name }}** and the value gets replaced. See the example playbook given below -

- hosts: all vars:

test1: "Bye World" tasks:

- name: Ansible when variable not equals example debug:

msg: "Not Equals"

when: test1 != "Hello World"

In the above playbook, **test1** is a variabe with a value "Bye World" It then compares it with a string using **when**

The below given playbook uses list feature of variables -

* hosts: all vars:

hello:

* + World
  + Asia
  + South America
  + North America
  + Artic
  + Antartic
  + Oceania
  + Europe
  + Africa tasks:

- name: Ansible List variable Example debug:

msg: "{{ hello[2] }}"

# Ansible roles

Basics are learnt. We have playbooks and they are too big to manage. to big to resuse and that brings down the concept of eficieny, adds time and cost. We need to write eficient playbooks, easy to use, reuse and debug. **ansible roles** comes in to rescue with various a task is divided into various playbooks with a

nice directory structure for variables, tasks, handlers, etc. The second major advantage that I see from this is you now have modular smaller playbooks that you can layer into other roles.

Ansible galaxy command is used create roles. Let us create an example role. \* Create an empty roles

ansible-galaxy init --offline apache-weberver

The above command will create a role which is nothing but a folder named **apache-webserver** with many more folders and files. See the flat **--oThine** which says, do not go and look for the roles on Ansible galaxy (which is a repository of roles created by the community).

Check the directory structure of the role created. use **tree** command and the output will look like -

apache-weberver/

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── README.md

├── tasks

│ └── main.yml

├── templates

├── tests

│ ├── inventory

│ └── test.yml

└── vars

└── main.yml

8 directories, 8 files

Understanding the roles directory structure

A role’s directory structure consists of defaults, vars, files, handlers, meta, tasks, and templates.

**defaults** Within defaults, there is a main.yml file with the default variables used by a role.

**vars** vars and defaults have variables, but variables in vars have a higher priority compared to that of defaults, which means that they are more dificult to override. Variables in defaults have the lowest priority of any variables available.

**files** files is where you put files that need to be added to the machine being provisioned, without modification. Most of the time, files in files are referenced by copy tasks.

**handlers** handlers usually contain targets for notify directives, and are almost always associated with services.

**tasks** tasks houses a series of Ansible plays to install, configure, and run software.