FIS

DEVOPS – CI/CD ANSIBLE

23-11-2020 to 27-11-2020

Venkatesh Reddy Madduri && ShankarPrasad

Agenda

- Introduction
- What is Configuration Management ?
- What is Ansible and its Architecture ?
- Installation and Set-up
- Ansible hands-on
- Conclusion

IT infrastructure refers to the composite of :

Hardware

Software

Network

People

Process

<u>Pain points</u> :

- create user account
- patch management
- taking backup
- deploying applications
- configure services
- documenting steps

The underlying problem is **STATE** of the servers

IT infrastructure refers to the composite of :

Hardware

Software

Network

People

Process

<u>Pain points</u> :

- create user account
- patch management
- taking backup
- deploying applications
- configure services
- documenting steps

The underlying problem is **STATE** of the servers

What Is Configuration Management?

- Configuration management (CM) refers to the process of systematically handling changes to a system in a way that it maintains integrity over time
- CM helps to implement
  - **Policies**
  - **Procedures**
  - **Techniques**
  - **Tools**

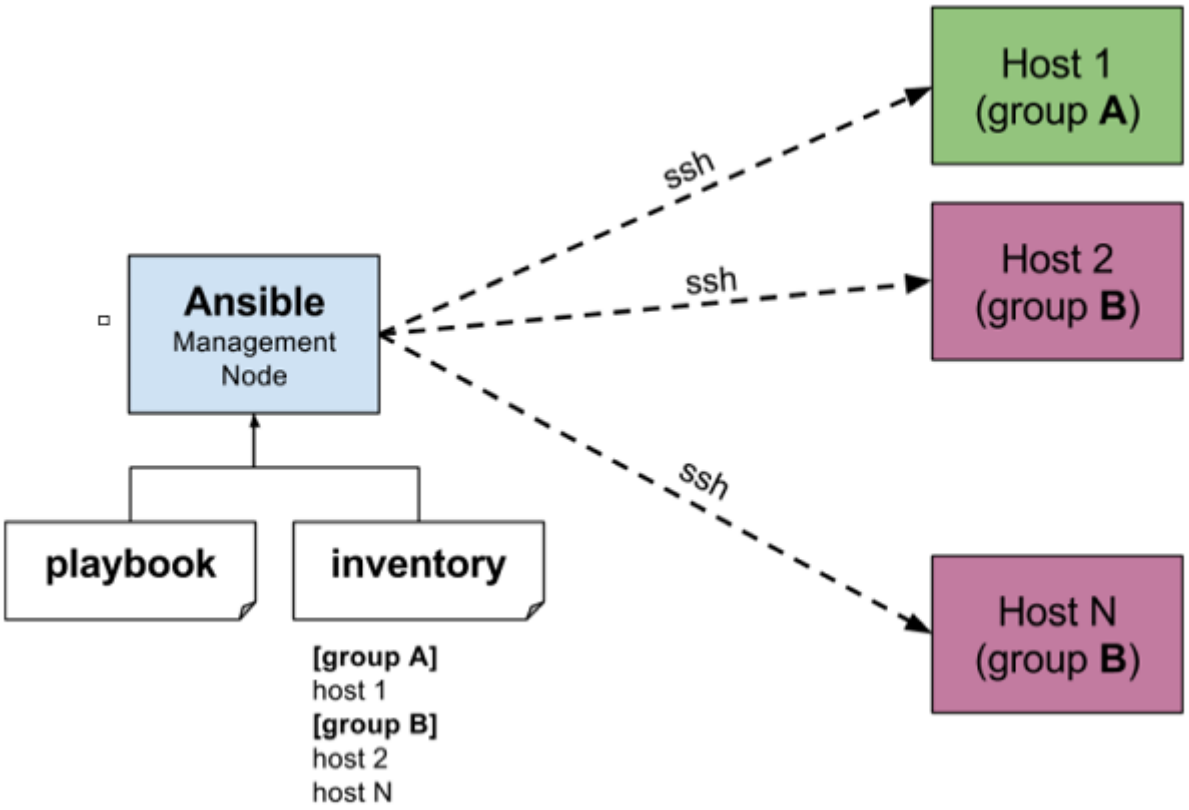
Why Configuration Management ?

- **Increase Uptime**
- **Improve Performance**
- **Ensure Compliance**
- **Prevent Errors**
- **Reduces Cost**

What is Ansible?

- Ansible is an automation engine that automates software provisioning, configuration management, and application deployment
- Manages infrastructure whether it is on-premises or in the cloud.
- It turns your infrastructure as code i.e your computing environment has some of the same attributes as your application:
  - Your infrastructure is versionable.
  - Your infrastructure is repeatable.
  - Your infrastructure is testable.
- You only need to tell what the desired configuration should be, not how to achieve it

How Ansible Works ?



Why Ansible?

Other tools in the market can be really complicated ..

- huge overhead of Infrastructure setup
- complicated setup
- Pull mechanism
- Lot of learning required

Pros of Ansible :

- Agentless
- Relies on ssh
- Uses python
- Push mechanism

Ansible Configuration file

- Settings in Ansible are adjustable via a configuration file
- Changes can be made and used in a configuration file which will be processed in the following order:
  - ANSIBLE_CONFIG (an environment variable)
  - ansible.cfg (in the current directory)
  - .ansible.cfg (in the home directory)
  - /etc/ansible/ansible.cfg
- Ansible will process the above list and use the first file found.
- Settings in files are not merged.

Setup Ansible on CentOS

% yum install epel-release

% yum update

% yum install git python python-devel python-pip openssl ansible

% ansible –version

% vim /etc/ansible/ansible.cfg & enable the below lines

inventory = /etc/ansible/hosts

sudo_user = root

- **Secure Sockets Layer (SSL)** is a standard security technology for establishing an encrypted link

between a server and a client

- ***OpenSSL*** is a general purpose cryptography library
- ***EPEL (Extra Packages for Enterprise Linux)*** *is open source and free community based repository which provides add-on software packages for Linux distribution*

Test Environment Setup

$ adduser ansible

$ passwd ansible

$ visudo (add a line as below & set EDITOR=vi)

$ export EDITOR=vi

ansibleadm ALL=(ALL) NOPASSWD: ALL

- Check sudo works without asking password

$ su - ansible

$ sudo yum update

- Run the following as ansible user

$ ssh-keygen

- copy the ssh keys to all the nodes

$ ssh-copy-id ansibleadm@<testmachine>

- Test ssh to testmachine , it should not ask password

$ ssh-copy-id localhost

Ansible Inventory

- Ansible recognizes systems listed in Ansible's inventory file, which defaults to being saved in the location **/etc/ansible/hosts**
- You can specify a different inventory file using the **-i <path>** option on the command line.
- The format for /etc/ansible/hosts is an INI-like format and looks like this:

[groupname]

machinename|machineIP

aliasname ansible_host=machinename|machineIP

Ex:

[demo]

Testserver1

testserver2.mylabserver.com

Host Patterns

- Patterns in Ansible are how we decide which hosts to manage.
- This can mean what hosts to communicate with, but in terms of Playbooks it actually means what hosts to apply a particular configuration or IT process
- $ **ansible <host_pattern> -m <module_name> -a <arguments>**

A pattern can usually refer to a particular machine or an groupname

"all" pattern refers to all the machines in an inventory

- $ **ansible all --list-hosts**

You can refer to hosts within the group by adding a subscript to the group name while giving the pattern

groupname[0] -- picks the first machine in the group

groupname[1] -- picks the second machine in the group

groupname[-1] -- picks the last machine in the group

groupname[0:1] -- picks first 2 machine in the group

- Groups separated by a colon can be used to use hosts from multiple groups

groupname1:groupname2

Ansible Ad-Hoc Commands

- Use **/usr/bin/ansible** to run ad-hoc tasks really quick & don't want to save for later
- These are quick one-liner without writing a playbook
- To run an arbitrary cmd use **-a** & use **-m** to run a module

  $ ansible [group|host] -m <module> -a <cmd>

  $ ansible all --list-hosts

  $ ansible all -m ping

  $ ansible all -a "ls -al /home/ansible"

  $ ansible all -a "cat /var/log/messages"
- To run anything with sudo, use **-s**

  $ ansible all -a -s "cat /var/log/messages"

  $ ansible local -a -s "cat /var/log/messages"

- Copy a file test.txt from local host to node

  $ ansible all -m copy -a "src=test.txt dest=/tmp/test.txt”

- Install/Remove a Package

  $ ansible all -s -m yum -a "pkg=httpd state=present”

  $ ansible all -s -m yum -a "pkg=httpd state=latest”

  $ ansible all -s -m yum -a "pkg=httpd state=absent”

  state=present will install it

  state=latest will update

  state=absent will remove it

Start/Stop a Service

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

$ ansible all -s -m service -a "name=httpd state=restarted”

$ ansible all -s -m service -a "name=httpd state=stopped"

- Create/Delete a User account

  $ ansible all -s -m user -a "name=test”

  $ ansible all -s -m user -a "name=test state=absent”

- Add/Remove a Cron Job

$ ansible all -u test -s -m cron -a "name='crontest' minute='0' hour='12' job='ls -al /var > /tmp/test.log'"

$ ansible all -u test -s -m cron -a "name='crontest' state='absent’”

Gathering Facts (idempotence or convergence)

$ ansible all -m setup

- Save the output to facts dir

$ ansible all -m setup --tree facts

- Filter only the specific field

$ ansible all -m setup -a 'filter=*ipv4*'
$ ansible all -m setup -a 'filter=ansible_domain'

Playbooks

- Playbooks are Ansible's configuration, deployment, and orchestration language
- Playbooks describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.
- Playbooks orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders
- If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.
- **/usr/bin/ansible-playbook** is used for running configurations from an playbook
- **$ ansible-playbook <playbook>.yml**
- Playbooks are expressed in YAML format

YAML(YAML Ain't Markup Language) Basics

- For Ansible, nearly every YAML file starts with a list
- Each item in the list is a list of key/value pairs, commonly called a "hash" or a "dictionary"
- All YAML files can optionally begin with "---" and end with "...
- All members of a list are lines beginning at the same indentation level starting
- with a "- "
  - **--- # A list of tasty fruits**
  - **fruits:**
  - **- Apple**
  - **- Orange**
  - **- Strawberry**
  - **- Mango**
  - **...**
- A dictionary is represented in a simple key: value form (the colon must be followed by a space)

```
--- # An employee record
Employee:
   name: Vignesh
   job: DevOps Engineer
   skill: Elite
...
```

- Each playbook is composed of one or more ‘plays’ in a list
- The goal of a play is to map a group of hosts to run tasks
- Task is nothing more than a call to an ansible module

- Playbooks are divided into 3 sections:

1. Target Section - Defines the hosts against which playbooks tasks has to be executed

2. Variable Section - Defines variables

3. Tasks Section - List of all modules that we need to run, in an Order

4. Handler Section – handles a task

5. Templates

Our First Playbook

- All sections begin with "-" & its attributes|parameters beneath it
- Identation is imp, use only spaces & not tabs

- first.yml

```
--- # My First YAML playbook
- hosts: demo
  tasks:
  - name: Install httpd on server
    yum: pkg=httpd state=installed update_cache=true
```

- Run ansible-playbook to call the playbook

  $ ansible-playbook first.yml

Target Section

- Example:

```
--- # My First YAML playbook
- hosts: all
  become_user: ansible
  become: yes           # yes or no
  connection: ssh       # ssh or paramico
  gather_facts: no      # yes or no
```

- Run ansible-playbook to call the playbook

  $ ansible-playbook first.yml

Task Section

- Example:

```
--- # My First YAML playbook
- hosts: all
  user: ansible
  become: yes
  connection: ssh
  gather_facts: no
  tasks:
    - name: Install HTTPD server on centos 7
      action: yum name=httpd state=installed
```

- Run ansible-playbook to call the playbook

```
$ ansible-playbook first.yml
```

Ansible Variables

- Refer various items for debug, set constant instead of typing every time
- **foo_port** is a great variable. **foo5** is fine too.
- **foo-port, foo port, foo.port** and **12** are not valid variable names.

➢ Registered Variables:

Running a command and using the result of that command to save the result into a variable

```
- name: Print Memory Used
  raw: echo '{{ansible_memfree_mb}}'
  register: output
- debug: var=output.stdout # If this is not given use -v while invoking the playbook #
```

➢ Accessing Complex Variable Data:

```
- name: Print IP Address
  raw: echo {{ ansible_eth0["ipv4"]["address"] }}
  raw: echo {{ ansible_eth0.ipv4.address }}
  register: output
- debug: var=output.stdout
```

➢ Magic Variables, and How To Access Information About Other Hosts:

- **"hostvars"** lets you ask about the variables of another host, including facts that have been gathered about that host

```
- name: Print Ansible Distribution for the host
  raw: echo {{ hostvars['test.mylabserver.com']['ansible_distribution'] }}
  register: output
- debug: var=output.stdout # If this is not given use -v while invoking the playbook #
```

- **"group_names"** is a list (array) of all the groups the current host is in

```
- name: Print Group Names
  raw: echo  {{ group_names }}
  register: output
- debug: var=output.stdout
```

- **"groups"** is a list of all the groups (and hosts) in the inventory. This can be used to enumerate all hosts within a group

```
- name: Print Group Names
  raw: echo  {{ group }}
  register: output
- debug: var=output.stdout
```

Global Variables

➢ Group Variables through Inventory:

Variables can also be applied to an entire group at once:

```
[demo]
Master

[demo:vars]
name=Battlecat
```

Example:

```
- name: Print Name
  raw: echo  '{{name}}
  register: output
- debug: var=output.stdout
```

➢ Group Variables through “group_vars” or “host_vars” dir:

In addition to storing variables directly in the inventory file, host and group variables can be stored in individual files relative to the group name under

➢ **/etc/ansible/group_vars/<group_name>**

Example:

/etc/ansible/group_vars/all

---

name: GROUP_VARS_ALL

/etc/ansible/group_vars/demo

---

name: GROUP_VARS_DEMO

➢ **/etc/ansible/host_vars/<host_name>**

Example:

/etc/ansible/host_vars/master

---

name: GROUP_VARS_MASTER

Variables: Inclusion Types

- Create a section called vars within a playbook
- Put vars above tasks so that we define it first & use it later

- Defining variables per playbook:

```
- hosts: demo
  vars:
    username: adam
    webroot: /var/www/html
  tasks:
  - name: Install httpd on server
    yum: pkg=httpd state=installed update_cache=true
```

- Put all the common variables in a file & include the file

```
- hosts: demo
  vars_files:
  - vars.yml
```

Variables: Inclusion Types

- vars.yml

```
---
username: adam
webroot: /var/www/html
```

➢ Prompt the user for the value

```
- hosts: demo
  vars_prompt:
    - name: your_name
      prompt: Your Name
```

➢ Passing Variables from Command Line

```
%  ansible-playbook register.yml --extra-vars "name=vignesh"
```

- Example

```
--- # My First YAML playbook
- hosts: all
  user: ansible
  sudo: yes
  connection: ssh
  gather_facts: no
  vars:
    playbook_version: 0.1b
  vars_files:
    - vars.yml
  vars_prompt:
    - name: app_state
      prompt: app state
  tasks:
    - name: Install HTTPD server on centos 7
      action: yum name=httpd state='{{ app_state}}'
```

```
> vars.yml
  tempvar: dummyvalue
```

➢ Run ansible-playbook to call the playbook

ansible-playbook first.yml

Handler Section

- Consists the ability to notify when something happens
- Also call another set of tasks

Example

```
--- # My First YAML playbook
- hosts: all
  user: ansible
  sudo: yes
  connection: ssh
  gather_facts: no
  tasks:
    - name: Install HTTPD server on centos 7
      action: yum name=httpd state=installed
      notify: Restart HTTPD   # this is called only if the action is ran & successful #
  handlers:
      - name: Restart HTTPD   # this has to match the notify name #
        action: service name=httpd state=restarted
```

- Run ansible-playbook to call the playbook

  $ ansible-playbook playbook.yml

Outlining your playbook

- vim webserver.txt & list down the tasks we want to do

- webservers # perform this against a list of webservers
- ansible user # we need to run this using ansible account
- sudo rights # we need sudo privilage for running the tasks

- date/time stamp for when the playbook starts
- install the apache web server
- verify that the web service is running
- install client software
** - telnet**

- log all the packages installed on the system
- date/time stamp for when the tasks is completed

Creating a playbook from our outline

```
--- # Outline to  playbook Translation
- hosts: all
  user: ansible
  sudo: yes
  gather_facts: no
  tasks:
  - name: date/time stamp for playbook start
    raw: /usr/bin/date > /home/ansible/playbook_start.log
  - name: install the apache web server
    yum: pkg=httpd state=latest
    notify: restart the HTTPD
 - name: install client software - telnet
    yum: pkg=telnet state=latest
  - name: log all the packages installed
    raw: yum list installed > /home/ansible/installed.log
  - name: date/time stamp for playbook end
    raw: /usr/bin/date > /home/ansible/playbook_end.log

  handlers:
    - name: restart the HTTPD
      action: service name=httpd state=restarted
```

Dry Run

- Check whether the playbook is formatted correctly
- Test how the playbook is going to behave without running the tasks
-

$ ansible-playbook webserver.yml --check

Asynchronous Actions and Polling

- While using Ansible against multiple machines, the operations may run longer than SSH
- While one long task is running, another short task can be executed in asynchronous mode
- Specify the maximum runtime to timeout & how frequently to poll for status
  - async: <seconds to timeout the task>
  - poll: <seconds to poll for the status of the task>

Example

```
--- # Running tasks parallel
- hosts: all
  user: ansible
  sudo: yes
  gather_facts: no
  tasks:
    - name: Install Apache
      action: yum name=httpd state=installed
      async: 300
      poll: 3
      notify: restart httpd
  handlers:
    - name: restart httpd
      action: service name=httpd state=restarted
```

- Run ansible-playbook to call the playbook

  $ ansible-playbook first.yml

Run Once

- In some cases there may be need to only run a task one time & on one host
- This can achieved by configuring "run_once" on a task
- This can be optionally paired with "delegate_to" to specify an individual host to execute on

Example

```
- hosts: all
  user: ansible
  sudo: yes
  gather_facts: no
  tasks:
    - name: list the /var dir
      command: ls -la /var >> /home/ansible/var.log
      run_once: true
      delegate_to: anmuruga3
```

Loops

- Often you'll want to do many things in one task, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached

Example 1

```
--- # Loop Playbook
- hosts: all
  sudo: yes
  tasks:
   - name: add a list of users
     user: name={{ item }} groups=wheel state=present
     with_items:
       - user1
       - user2
```

Example 2 : list of items

```
--- # Loop Playbook
- hosts: all
  sudo: yes
  tasks:
   - name: add a list of users
     user: name={{ item.name }} groups={{ item.groups }}  state=present
     with_items:
      - { name: testuser1, groups: wheel }
      - { name: testuser2, uid: 1003, groups: root }
```

Example 3 : loving over files

```
--- # Loop Playbook
- hosts: all
  sudo: yes
  tasks:
   - debug:
      msg: "{{ item }}"
     with_file:
      - first_example_file
      - second_example_file
```

Example 4 : nested loops

```
--- # Loop Playbook
- hosts: all
  sudo: yes
  tasks:
   - debug:
       msg: "{{item[0] }}“ - "{{item[1] }}"
     with_nested:
     - [ ‘vignesh', ' m' ]
     - [ 'clientdb', 'employeedb', 'providerdb' ]
```

Conditionals

- Few tasks might be needed to execute only on specific scenario

- **When statement**

  Sometimes you will want to skip a particular step on a particular host

Example 1: **Single condition**

```
--- # When playbook example
- hosts: demo
  user: ansible
  sudo: yes
  connection: ssh
  tasks:
    - name: Install apache for debian
      command: apt-get -y install apache2
      when: ansible_os_family == "Debian"
    - name: Install apache for redhat
      command: yum -y install httpd
      when: ansible_os_family == "RedHat"
```

Example 2

```
tasks:
    - command: echo {{ item }}
      with_items: [ 0, 2, 4, 6, 8, 10 ]
      when: item > 5
```

Example 3

```
vars:
  epic: true
tasks:
    - command: echo “this is certainly epic !”
       when: epic
    - command: echo “this certainly isn’t epic!”
       when: not epic
```

Example 4

```
- stat: path=/tmp/thefile get_md5=no get_checksum=no
  register: st
- debug: var=st
- shell: touch /tmp/thefile
  when: not st.stat.exists
```

Example 5: Multiple Conditions

```
--- # When playbook example
- hosts: demo
  become: yes
  tasks:
    - name: Install apache for debian
      command: apt -y install apache2
      when: ansible_os_family == "Debian" or ansible_os_family == "Fedora"
    - name: Install apache for redhat
      command: yum -y install httpd
      when: ansible_os_family == "RedHat" and ansible_pkg_mgr == "yum"
```

Example 6: Boolean Filter

```
  tasks:
    - name: Boolean example
      command: echo Test
      when: ansible_distribution_version|version_compare('15.04', '>=')
```

Error Handling

- Sometimes a command that returns different than 0 isn't an error.
- Sometimes a command might not always need to report that it 'changed' the remote system.

Example 1

```
--- # When playbook example
- hosts: demo
  user: ansible
  sudo: yes
  connection: ssh
  tasks:
    - name:  This will not be considered as Failure
      command: /bin/false
      ignore_errors: yes
    - name: Fail task when the command error output prints FAILED
      command: /usr/bin/example-command
      register: command_result
      ignore_errors: yes
    - debug: var=command_result.rc
```

Example 2

```
--- # When playbook example
- hosts: demo
  user: ansible
  sudo: yes
  connection: ssh
  tasks:
    - name: Fail task when the command error output prints FAILED
      command: /usr/bin/example-command
      register: command_result
      ignore_errors: yes
   - name: fail the play if the previous command did not succeed
      fail: msg="the command failed"
      when: "'FAILED' in command_result.stderr“
    - name: Fail task when both files are identical
      raw: diff file1 file2   # checks the files in the home dir of the user
      register: diff_cmd
      failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2
```

wait_for - Waits for a condition before continuing

- You can wait for a set amount of time
- Waiting for a port to become available is useful for when services are not immediately available
- Wait for a regex match a string to be present in a file

Example1: Port open

```
- name: Install Apache on CentOS
  yum: pkg='{{ redhat_apache}}'  state=latest
  when: ansible_os_family == " Redhat"
- name: wait for the service to start listening on port 80
  wait_for:
    port: 80
    state: started
```

- Wait for port 80 to become open for the host
- While executing the playbook ansible will be wait for http service to be started
- Once you start the service ansible will proceed with its play

```
$ systemctl start httpd
```

Example2: File to be created

```
- name: Dummy Task
   command: echo Dummy
- name: wait until the file is present before continuing
  wait_for:
    path: /tmp/dummy
    delay: 10
    timeout: 300
    msg: "Specified FILE is not present"
```

- **delay:** Delay in seconds before starting the validation
- **timeout:** timeout after defined seconds
- **msg:** A Custom message to be printed in case of failure

Example3: String to be available in the log file

```
- name: Wait_for the string to be available in the log file
  wait_for:
    path: /tmp/dummy
    search_regex: "hi adam"
```

Example4: sleep for seconds

```
- name: sleep for 10 seconds
  wait_for:
    delay: 10
    timeout: 0
```

Jinja2 Template

- A template is a file which contains all your configuration parameters, but the dynamic values are given as variables
- During the playbook execution, the variables will be replaced with the relevant values. The template files will usually have the .j2 extension
- Ansible uses Jinja templating engine, we can have conditional statements, loops, write macros, filters for transforming the data, do arithmetic calculations, etc

Example1:

```
- hosts: demo
  vars:
    myname: Adam
  tasks:
    - name: Ansible Template Example
      template:
        src: test.j2
        dest: /tmp/testfile
```

test.j2:

```
Hello {{myname}}
```

Example2: loop structure inside Ansible template. Change content of test.j2:

```
{% for i in range(3)%}
  hello {{myname}} - {{i}}
{% endfor %}
```

Example3: Using list variables in Ansible templates

```
- hosts: demo
  vars:
    mylist: [‘vignesh', 'm' ,'devops']
  tasks:
    - name: Ansible Template Example
      template:
        src: test.j2
        dest: /tmp/testfile
```

test.j2:

```
{% for item in mylist %}
  {{ item }}
{% endfor %}
```

Arithmetic Operations in Ansible

```
---
- hosts: demo
  tasks:
  - debug:
      msg: "addition{{ 4 +3 }}" #Ansible addition 7
  - debug:
      msg: "substraction {{ 4 - 3 }}"  #Ansible arithmetic substraction 1
  - debug:
      msg: "multiplication {{ 4 * 3 }}" #multiplication 12
  - debug:
      msg: "Modulo operation {{ 7 % 4}}" #ansible Modulo operation - find remainder
3  - debug:
      msg: "floating division {{ 4 / 3}}" #ansible floating division 1.33333333333
  - debug:
      msg: "cube root {{ 27 | root(3)}}" # Ansible arithmetic cube root 3.0
  - debug:
      msg: "power {{ 3 | pow(3)}}" #Ansible arithmetic power of a number 27
  - debug:
      msg: "Common Ansible round of a number {{ 39.7 | round}}" #40
```

Tags

- If you have a large playbook it may become useful to be able to run a specific part of the configuration without running the whole playbook.

```
--- # Tag functionality playbook
- hosts: demo
  user: ansible
  sudo: yes
  connection: ssh
  tasks:
    - name: first name
      raw: echo “vignesh " > /tmp/LOG
      tags:
        - firstname
    - name: second name
      raw: echo "M " > /tmp/LOG
      tags:
        - secondname
$ ansible-playbook playbook.yml --tags "firstname"
$ ansible-playbook playbook.yml --tags "secondname"
```

Tags

- If you want to run a playbook without certain tasks

$ ansible-playbook playbook.yml –skip-tags "firstname"

- There is a special “always” tag that will always run a task, unless specifically skipped

$ ansible-playbook playbook.yml –skip-tags always

Vault

- Ansible allows keeping sensitive data such as passwords or keys in encrypted files, rather than as plaintext in your playbooks

- **Creating a new Encrypted Files**
  $ ansible-vault create playbook.yml

- **Edit the Encrypted File**
  $ ansible-vault edit playbook.yml

- **Change the password**
  $ ansible-vault rekey playbook.yml

- **Uncrypt the file**
  $ ansible-vault decrypt playbook.yml

- **Encrypt an existing file**
  $ ansible-vault encrypt playbook.yml

Securing Certificates

- Create a file through valut to hold your secrets – secrets.yml

  $ ansible-vault create secrets.yml

- Add the certificate and private keys as variables:

```
---
ssl_certificate: |
  -----BEGIN CERTIFICATE-----
  ...
  -----END CERTIFICATE-----
ssl_private_key: |
  -----BEGIN PRIVATE KEY-----
  ...
  -----END PRIVATE KEY-----
```

  - Make sure the vault is loaded

```
---
- hosts:
  vars_files:
    - vars/secrets.yml
```

- Run the playbook using “--ask-vault-pass” or “--vault-password-file FILE”

Include statements

- Common tasks can be put in a file & can be included anywhere in the playbook

Example:

✓ **Create includestat.yml**

```
- name: Install httpd
  yum: pkg=httpd state=latest
```

✓ **Create a new playbook**

```
--- # Include Task playbook
- hosts: demo
  user: ansible
  sudo: yes
  connection: ssh
  tasks:
    - include: includestat.yml
    - name: verify the httpd is installed
      raw: yum list installed | grep httpd > /tmp/result.log
```

Roles

- Adding more & more functionality to the playbooks will make it difficult to maintain in a single file
- We can organize playbooks into a directory structure called roles
- This is already possible by ‘include’ directives however Roles are automation around it

- **Creating Role Framework**

Master.yml
roles/<rolename>/
tasks/main.yml
vars/main.yml
handlers/main.yml
default/main.yml
meta/main.yml

Roles Task Order - Pre & Post Tasks

- In Master playbook Roles will always run first, regardless of where the tasks appear
- Set tasks to run before or after the Roles using pre_tasks & post_tasks

Example

✓ **Create master.yml**

```
--- # master playbook for web servers
- hosts: all
  user: ansible
  sudo: yes
  connection: ssh
  pre_tasks:
  - name: Start of the Role
    raw : date > /home/ansible/rolestart.log
  roles:
    - webservers
  post_tasks:
  - name: End of the Role
    raw : date > /home/ansible/roleend.log
```

Roles - Conditional Execution

- Just like master playbook we can set conditional execution on the roles

Example

- ✓ vi roles/<rolename>/tasks/main.yml

```
- name: Install Apache on CentOS
  yum: pkg=httpd state=latest
  when: ansible_os_family == "RedHat"
  ignore_errors: yes
- name: Install Apache on Ubuntu
  apt: pkg=apache2 state=latest
  when: ansible_os_family == "Debian“
  ignore_errors: yes
```

Roles - Variable Substitution

Example

- ✓ **Create roles/<rolename>/vars dir**
- ✓ **vi main.yml**

```
redhat_apache: httpd
debian_apache: apache2
```

- ✓ **vi roles/<rolename>/tasks/main.yml**

```
- name: Install Apache on CentOS
  yum: pkg='{{ redhat_apache}}' state=latest
  when: ansible_os_family == "RedHat"
  ignore_errors: yes
- name: Install Apache on Ubuntu
  apt: pkg=apache2 state=latest
  when: ansible_os_family == "Debian"
  ignore_errors: yes
```

Roles - Handlers

- **Create roles/webserver/handlers dir**

- **vi main.yml**

```
- name: restart httpd
  service: name='{{ redhat_apache}}' state=restarted
- name: restart Apache2
  service: name='{{ debian_apache}}' state=restarted
```

- **vi roles/webserver/tasks/main.yml**

```
- name: Install Apache on CentOS
  yum: pkg='{{ redhat_apache}}'  state=latest
  when: ansible_os_family == "Redhat"
  notify: restart httpd
- name: Install Apache on Ubuntu
  apt: pkg='{{ debian_apache}}' state=latest
  when: ansible_os_family == "Debian"
  notify: restart Apache2
```

Roles - Configuring Alternate Roles Paths

- Default path for Roles

  /home/ansible/playbooks/roles:/etc/ansible/roles:<PWD>

- We can alternatively keep the master playbook in a different location & specify the Role path in ansible.cfg
- In the /etc/ansible/ansible.cfg, uncomment roles_path & add the roles dir

  roles_path = /home/ansible/playbooks/roles

Roles - Conditional Include Statements

- When we have multiple roles & choose a specific role based on a condition

Example

- ✓ **mkdir redhatwebservers & copy contents of webservers**
- ✓ **mkdir debianwebservers & copy contents of webservers**
- ✓ **vi master.yml**

```
--- # master playbook for web servers
- hosts: all
  user: ansible
  sudo: yes
  connection: ssh
  pre_tasks:
  - name: Start of the Role
    raw : date > /home/ansible/rolestart.log
  roles:
    - { role: redhatwebservers, when: ansible_os_family == "RedHat" }
    - { role: debianwebservers, when: ansible_os_family == "debian" }
```

Thank you