Ansible-Basics

- **Introduction**:
  - Documentation ; https://docs.ansible.com/
  - Configuration Management Tool. Used to make configuration changes in bulk.
  - Requires Python, runs on POSIX OS flavours.
  - Agentless software. Does not require any agents to be running on the target machine. However expects SSH or WinRM connections.
    - Note: While ansible works agentless, when you try to run some modules on the remote they might need python. So always recommended to ensure python is installed on target (for linux machines).
  - Will only make changes to the system if there is something to do. If the desired state / file is present , no action is performed.
  - Requires
    - Inventory File - Which would be the hosts on which the commands need to be run. Can be static or dynamic.
    - Playbook - A YAML file that contains the commands and scenarios
or
The command to run
  - Ansible Tower is the commercial implementation , it is web based.
  - Provides integration for IT automation (Infrastructure, Networks, Cloud , DevOps Tools etc)
    - How ansible can help automate respective provider is an individual topic on its own.
    - The module/ plugin name is unique to what we are trying to automate (Azure/ AWS, Containers etc), with each having its own parameters requirement.
    - Documentation should be able to help understand the prerequisits.
    - Further read - Integrations Overview | Ansible
  - How it works:
    - Connects to the target thru SSH or WinRM
    - Collects the details (called facts) on the defined hosts
    - Copies the required module to the target machine (in tmp)
    - Runs the commands
    - Deletes the ansible files (created in tmp)
- **Installation** & **Configuration**:
  - Install
    - Linux:
      - Ensure Python is installed.
      - If access to AWS is required install boto
      - Install Ansible package.
    - Windows:
      - Cannot be installed directly. Needs Windows Sub System (WSL) to be installed.
      - Install Unix on WSL (like Ubuntu)
      - Once done, install ansible just like any Linux system
  - Configuration:
    - Ansible.cfg is the config file for ansible.
      - Lists home directory, default inventory file path, install path etc
      - If Python install director is different, ensure the appropriate path is updated in the config file.
  - Remote Connectivity:
    - Uses SSH to connect to Linux and WinRM to connect to Windows.
      - For SSH / Linux Connectivity:
        - Create ssh key using $ ssh-keygen .
        - The public key in .pub file needs to be copied to the .ssh/authorized_keys file of the target hosts. Can be done manually per host or can be done using the process @ - Ansible Playbook: Deploy the public key to remote hosts | by Chris Kong | Medium
      - For WinRM, there are multiple options like HTTP / CredSSP, Kerberous etc.
        - Configuring host options - How to connect Ansible to a Windows host via WinRM, with Basic, NTLM or Kerberos authentication – D2C-IT (skip the vagarant install part).
        - Further read: - Windows Remote Management — Ansible Documentation or Setting up a Windows Host — Ansible Documentation
  - Access :
    - By default Ansible tries to run the commands on the remote server as remote_user (the ID that logged in on the master).
      - Connection to the remote server is made through SSH (Linux) or WinRM (Windows).
    - If the commands need to be run as a different user , become / become_user are used. See "Run Plays or tasks as different user below".
    - Remember:
      - become by default uses sudo, so unless otherwise mentioned it tries to run the commands under "sudo", if the sudo expects password ensure it is prompted --ask--become-pass while running the play.
      - If intention is to not prompt for sudo password, ensure the sudeors (/etc/sudoers or the files in /etc/sudoers.d) file is updated with nopass.
      - Default become_user is root.
- **Commands**:
  - Two core commands

$ ansible	• For running individual/ ad-hoc commands , checking version, etc • Eg: ansible all --inventory-file=/root/ansible_hosts -m ping [inventory file can be used by "-i" or full name as shown. -m is to call a module, in this example ping module is called to run on the defined hosts. This checks if the hosts are accessible by ansible.] ansible all --inventory-file=/root/ansible_hosts -m setup [gives the ansible facts, collects information about the mentioned hosts] Format is : ansible <host group> -I <Inventory file> -m <Module Name> -a "<arguments to module>
$ ansible-playbook	• For running playbook commands ○ $ ansible-playbook -I <inventory file > <Playbookfile.yml> [executes the play in the yml file on the hosts mentioned in the inventory file] ○ Usage of "--check" however will not run the commands on the remote host but will only show the output on screen. ○ "-- syntax-check" will check syntax for play. Displays playbook if no errors. • Multiple inventory files can be provided . However "-i" should be used against each file.

  - Note: if instead of an inventory file you want to pass a single or a set of hosts you can use the following (note the "," at the end of host lists):
    - $ ansible all -I "host1,host2," -m module
    - $ ansible-playbook -I "host1,host2," playbook.yml
  - Other commands:

$ ansible --version	• Lists version of ansible & the paths for the config file

	• ansible 2.7.7 config file = /etc/ansible/ansible.cfg configured module search path = ['/home/vinuganti/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules'] ansible python module location = /usr/lib/python3/dist-packages/ansible executable location = /usr/bin/ansible python version = 3.7.3 (default, Jan 22 2021, 20:04:44) [GCC 8.3.0]
$ansible-config	• Lists configuration • Variation: • $ ansible_config dump --only changed [lists config options that only changed]
$ ansible-doc	• Help command. Lists information of modules installed
$ ansible-vault	• Used to encrypt or decrypt data files that can be used in playbook

- **Terminology**:

Host	• The target machine on which a certain action needs to be performed.
Playbook	• A set of commands in YAML format, that outlines the actions needed to be performed , the hosts on which these actions need tobe performed on, how to connect to the hosts & when to run the commands.
Module	• A Function that is used in playbook for better feature / functionality • There are built in modules and user defined modules. • Written in python (linux) or powershell (windows) • List of all modules, their parameters & return values : Index of all Modules — Ansible Documentation
Plugin	• Extends functionality of Ansible. Facilities third party integration. • Further read : https://docs.ansible.com/ansible/latest/plugins/plugins.htm
Command	• The action to be performed on target host.
Tasks	• Set of commands to run. Set of tasks is called play
Variables	• Temporary placeholders of a dynamic content • Can be at host level , playbook level or group level
Group	• Collection of hosts
Jinja	• A language derived from Django. Used to define templates (or dynamic files) that need to be placed on the target hosts. • Further read : Templating (Jinja2) — Ansible Documentation & How to use Jinja2 templates in Ansible with examples | GoLinuxCloud
Fact	• Reference to a value in YAML file (usually a value referencing the targeted host) • Ansible_fact is a common used array that contains information about the target host • For e.g. ansible_facts['os_family'] is used to check the target host's OS family.
Role	• A set of commands or actions . A reusable playbook. • Written in YAML file. Helps avoid reduce repetition of tasks . • Can be likened to external / user defined functions • Further read ; Roles — Ansible Documentation

- **Inventory file**
  - Further Read: https://docs.ansible.com/ansible/2.7/user_guide/intro_inventory.html:
  - format
    ```
    [Group Name-1]
            Host1
            Host2
    [Group Name-2]
            Host3
            Host4
    ```
  - Example: [notice that same servers can be in multiple groups. "all" group represents all hosts irrespective of group it is in]

    -
      ```
      [dbservers]
      Db1
      Db2
      [webservers]
      Web1
      Web2

      [QA]
      Db1
      Web1

      [prod]
      Db2
      Web2
      ```
- **Playbook**
  - list of keywords - https://docs.ansible.com/ansible/latest/reference_appendices/playbooks_keywords.html)
  - File Format:
    Important - Spacing and alignment is the key. Use spaces and not tabs when creating the yaml file.
    Format:
    ```
    ---
     - name : <over all name for the play >  [this is optional]
       hosts : <host name / host group name per inventory file / local host or all >
       become: true            [optional - enumerates commands on remote host as root ]
       tasks :
         - name: <name of task>
           <module / command to execute on remote machine>
              <parameters for the module or command , usually in variable : value format >
           when  - <condition> [optional  - if we want this task to be executed on a given condition]
    ```
  - Variables:
    - Variables are referenced in YAML file using {{ <variable name> }}
    - Standard (In Inventory file)
      [Group Name]

```
Host1   var3 = 'value'  [This variable will be specific to this host only]
Host2
[Group Name: Vars]  [Variable definitions for the particular group]
    Var1 = 'Value'
    Var2 = 'value'
```

- Alternately variables can be defined under
  - "group_vars" directory , with file name being the name of the group and the variable definitions inside that file. (in <variable name> : <variable value> format)
  - "host_vars" directory, with file names being the name of the hosts and the variable definitions insides that file (in <variable name> : <variable value> format)
  - Note: The host names and group names should match the entries in inventory file.
- Arrays can be defined as :
  - Vars:
    ```
    Packages:
     - Nmap
     - Httpd
    ```
  - Vars:
    ```
    Users:
        Aditya:
            Lname: 'abc'
            Fname: 'def'
            Home: /home/abc
    ```
  - Value is referenced as {{ users['abc']['home']}} to get the home directory for a user.
- Files containing other variables can be referenced using :
  ```
  vars_files:
      - "./mypwdfile.yml"
  ```

- Register - command used in playbook to store output of command into a variable
  - Usage: register: <variable name> (will need to follow the command for which output needs to be stored here)
- Running plays or tasks as a different user.
  - Further read: Understanding privilege escalation: become — Ansible Documentation
  - Normally the ID used for running plays, makes the SSH/ WinRM connection to the node and runs the play.
  - If you want to execute the tasks in play as a different user (like "sudo" in Linux or "RunAs" in windows), you use the following in your play

    become: true (all lower case)
    become_user: root (all lower case) (ID to runas; remember you need to make sure you can SSH/WinRM using this ID first on the target nodes)
    - If no become_user is mentioned, root is assumed.
  - If become is used at the beginning of play, it is for entire play. Else it can be used for individual tasks as well.
    ```
    - name: Ensure the httpd service is running
      service:
        name: httpd
        state: started
      become_user: root
    ```
- Debug - to display output during execution
  - Example (for debug & register):
    ```
    - name: httpd status
    command: service httpd status
    register: httpd_status                    [ Stores output of the earlier command in the variable name mentioned]
    - name: httpd status output
    debug:
    var: httpd_status                         [ displays the output of the variable ]
    ```
  - Points to note:
    - Var = variable / content
    - Msg = Message to display (if nothin is mentioned - "Hello World" will be displayed)
    - Verbose = level of details to show.
    - When using register, the output is stored into the defined variable in a JSON format. So the variable has all details like command, status, output, error etc.
    - If only a certain value in a variable is needed use {{variable['header']}} format.
      - For e.g. - see play below:
        ```
        ---
         - name: My Test Play
           hosts: all
           tasks:
             - name: Get Host Name
               command: hostname
               register: var2
             - name: Get Date on node
               command: date
               register: var1
             - name: Display values
               debug:
                 msg: "Date on {{var2['stdout']}} is {{var1['stdout']}}"
        ```

- set_fact - a module in playbook that allows combining of variables into a new variable . Since variable assignment is usually done in va r_files or inventory, set_fact helps with variable assignment during a play in playbook.
  - Example:
    ```
    - set_fact: var1="something" var2="{{ local_var }}"
    # Example setting host facts using complex arguments
    - set_fact:
        var1: something
        var2: "{{ local_var * 2 }}"
        var3: "{{ some_registered_var.results | map(attribute='ansible_facts.some_fact') | list }}"
    ```
- Loops
  - can be achieved through {{item}} and with_item statements. (other loop statements : Loops — Ansible Documentation)
  - Example:
    ```
    - hosts: all
      vars :
    ```

```
        packages: [git, vim, ruby]                    # Defining an array called Packages with values as "git, vim & ruby"
      tasks:
        - name: install packages                     # name of the task
           apt :                                     # module to run assuming apt is the package installer that works on all hosts
                 name: {{item}}                      #  Highlighted section is the loop, loops through all entries in package variable. {{item}} represents each value
                 state: latest
          with_item: {{packages}}
```

- Handlers - Special tasks that are executed only when notified. Typically when something changed.
    - Example:

```
- name: Template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - Restart memcached          -> Name of the handler to be executed
    - Restart apache
  handlers:
    - name: Restart memcached
     .service:
        name: memcached
        state: restarted
    - name: Restart apache
      service:
        name: apache
        state: restarted
```

- Error handling can be achieved through :
    - Block   - similar to try block
    - Rescue - tasks to be executed if block task fails (similar to catch block)
    - Always - task to be executed at all times (similar to final block)
    - Example:

```
tasks:
 - name: Handle the error
  block:
    - name: Print a message
      debug:
        msg: 'I execute normally'
    - name: Force a failure
      command: /bin/false
    - name: Never print this
      debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - name: Print when errors
      debug:
        msg: 'I caught an error, can do stuff here to fix it, :-)'
```

- **Playbook Examples**
    - you can have multiple plays). Must start with "---" to signify YAML file.
    - Tasks are executed sequentially.
    - Spacing and indentation is important. Subtasks / parameters have 2 space indentation to the module.
    - Example - 1:
        - Note: yum and service are module. Yum module ensure httpd package is the latest and Service module ensure httpd is in started

```
- name: Play1
  hosts: webservers
  become: yes
  become_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: ensure apache is running
      service:
        name: httpd
        state: started
```

        - Same can also be written as below, notice the change in indentation and usage of "=" instead of ":" :

```
- name: Play1
  hosts: webservers
  become: yes
  become_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:  name= httpd state= latest
    - name: ensure apache is running
      service: name = httpd state= started
```

    - Example -2
        - Note: user and file are modules that take the required parameters and based on the state they are created or removed

```
- name: Play app - Create tomcat directories and username in app servers
  hosts: appservers
  become: yes
  become_user: root
  tasks:
    - name: Create a username for tomcat
```

```
      user:
        name: tomcatadm
        group: users
        shell: /bin/bash
        home: /home/tomcat

    - name: create a directory for apache tomcat
      file:
        path: /opt/oracle
        owner: tomcatadm
        group: users
        state: present
        mode: 0755
```

- Example 3: [executing task on condition]

```
tasks:
  - name: "shut down Debian flavored systems"
    command: /sbin/shutdown -t now
    when: ansible_facts['os_family'] == "Debian"
    # note that all variables can be used directly in conditionals without double curly braces
```

- **Common Used Modules :**

Method	Purpose	Usage / Example
Copy	Used to copy files from source to destination	copy: src=abc.html dest=target/abc.html
Service	To start / stop / disable / enable a service etc	service: name=httpd state=started - name: Stop service httpd, if started service: name: httpd state: stopped
File	Perform file or directory operations (create / delete / permissions etc)	- name: Remove file (delete file) file: path: /etc/foo.txt state: absent - name: Recursively remove directory file: path: /etc/foo state: absent
Script	Runs a script on remote file after copying it	- name: Run a script with arguments (free form) script: /some/local/script.sh
Package	Generic Package manager (instead of using the os specific yum , apt modules etc)	- name: Install ntpdate package: name= ntpdate state=present - name: Remove the apache package package: name= "{{ apache }}" state= absent
Ping or win_ping	Checks connectivity of ansible master to host	
Setup	Gathers details of the host	- Name: Get Details of remote (facts) setup

- **Roles**:
  - Further Read: https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html
  - Created using $ ansible-galaxy init <role name> [ansible galaxy is also a service that host user defined roles - https://galaxy.ansible.com/]
    - Above command creates a directory structure with the role name
    - The YAML files need to be modified per requirement.
  - The created role is called in the playbook file using the roles module
  - For e.g.

```
-name: Test call to role
 roles:
   my_new_role
```

- **Using Ansible_vault**
  - Further read: Encrypting content with Ansible Vault — Ansible Documentation & Ansible Vault | Learn the Examples of Ansible Vault (educba.com)
  - In simple sense, it basically encrypts a text file. Can encrypt existing files, strings, create files etc.
  - Usage is the text file contains the variables / secrets.
  - These encrypted values can be accessed using the --vault-id options in ansible/ ansible-playbook.
  - File encrypted will required a vault password. The same password is needed to decrypt the file. If lost, the file cannot be d ecrypted.
    - Can be prompted using --ask-vault-pass in ansible or ansible-playbook.
    - If the password is stored in another text file , it can be passed as parameter using --vault-password-file= pathtofile/filename.yml
  - Vault-id as labels provided to distinguish multiple encrypted files / strings. Provided at the time of encryption and referenced d uring running the play.
    - --vault-id <vaultID>

- **Gotchas…. Points to remember**
  - Python must be installed on target (ansible looks for it)
  - Check SSH or WinRM connection to all nodes before running plays. Best course would be to run the ping (linux) or win_ping(wi ndows) to ensure ansible is able to talk.
  - If you are using "become" in your plays, ensure SSH or WinRM connection using the ID that you want to run-as (become or sudo) is a success to the node.
  - Ansible by design, does not ask for confirmation before performing the actions.
    - Ensure you are checking the play and the appropriate switches and parameters.
    - Ensure you take into factor modifications to target when designing your play and use the appropriate switches.
      - For e.g.
        - Copy will copy file from source to target and will skill if the file exists. However if the file changed, unless you force it to copy, the copy module will not replace the destination file.
  - Two tasks (module actions) cannot be clubbed into one. Each task has to be a separate task entry, you will need to use variab les to pass output of one to another if there is dependency.
  - Keep an eye for the indentation and spacing in YML file.

- Running ansible or ansible play book in verbose (i.e. -v) will help . Ansible allows about 5 levels of verbosity. (i.e -v, -vv, -vvv, -vvvv, -vvvvv) , the depth of information that it shows when running the command.