ANSIBLE

Configuration Management:

Let us say, we are

- creating a user or configuring ssh keys for a user or
- installing packages, upgrading or removing packages or
- modifying configuration files or any such operations

To perform above activities, we use manual or using scripts. However, this approach can be done on few machines. If same thing can be done on some hundreds of machines using configuration management.

Benefits of configuration management tools:

The main benefit of using a configuration management tools like -

- entire infrastructure blueprint can be documented, created and applied to any number of environments
- This approach saves a lot of time and effort
- can also prevent human errors (as entire required configuration items are recorded in the blueprint..)
- Can be implemented in both in a physical data centre and in public cloud
- Configuration Management tools can deploy applications on multiple computers, can deploy configurations, manage configurations, track the configurations etc.

History:

- The Ansible tool was developed by Michael DeHaan
- Ansible is an open-source software provisioning, configuration management, and application-deployment tool.
- Currently, ansible is a subsidiary unit of IBM.

Advantages of CM tool:

- Increased efficiency with a defined configuration process that provides control and improves visibility with tracking.
- Cost reduction by having detailed knowledge of all the elements of the configuration which allows for unnecessary duplication to be avoided.

- Your business will have greater agility and faster problem resolution, giving a better quality of service for your customers.
- More efficient change management that reduces the risk of product incompatibility or problems.More efficient change management that reduces the risk of product incompatibility or problems.
- Enhanced system and process reliability through more rapid detection and correction of improper configurations that could negatively impact performance.

Why Ansible,Ansible Advantages:

Why Ansible:

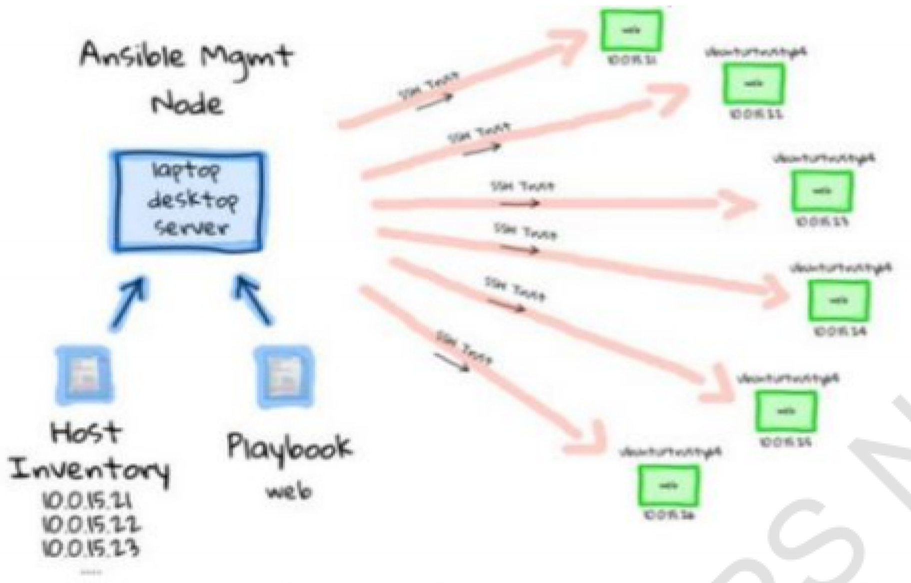
- Ansible is free and Open Source.
- Agentless. Ansible doesn't require any agent on client machines unlike other automation tools that exist in the market (Puppet, Chef, Salt.). It uses SSH protocol to connect the servers. Ansible required Python to make the use of modules on client machines. Ansible also works with a system which doesn't have python installed using the "raw" module.
- Ansible uses YAML language which is very easy to learn.
- Supported by Red Hat.

Advantages of Ansible:

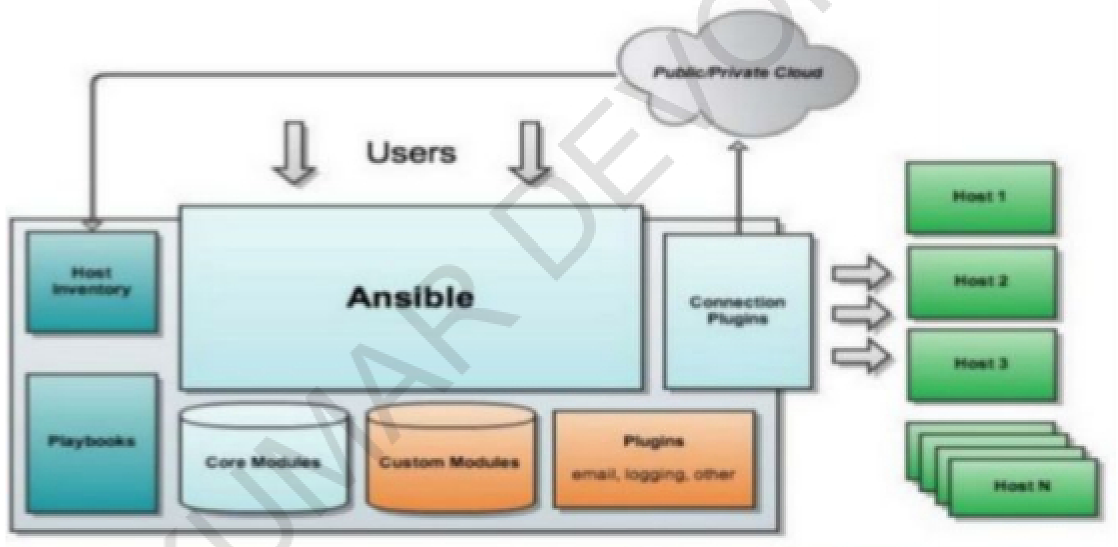
In our experience there are at least three advantages that make Ansible our favorite automation tool.

1. It is agentless. You do not need to install additional software on your server nodes. This helps keep the installation clean while ensuring that there are no conflicts with our software.
2. Playbooks are easy to read and edit. They are mostly written in YAML, and this is a great advantage when compared to other solutions, such as Puppet.
3. It is written in Python, a very popular programming language that is familiar to our engineers, making it easy to extend.

Ansible Architecture setup:



- Playbooks contains a scripts related to configuration management



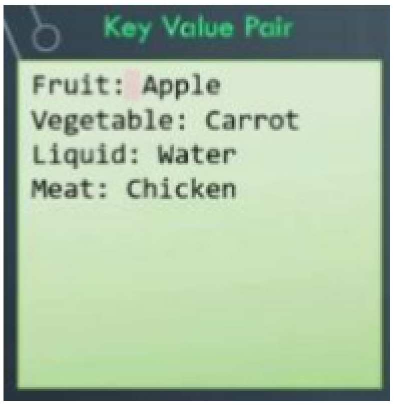
Install & configure Ansible:

- Installation process on RHEL
  # sudo yum install ansible
- For more information on installation on other flavours ,refer to below link

https://docs.ansible.com/ansible/latest/installation_guide/intro_instalation.html
  #installing-the-control-machine
- To verify version of ansible?
  # ansible --version
- To verify version of ansible playbook version?
  # ansible-playbook --version

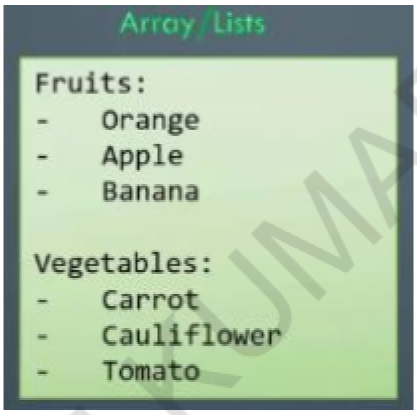
Yaml Language for Ansible -

- Playbooks are written in yaml language
- A YAML file is used to represent Configuration Data
- If you take the data in its simplest form such as Key Value pair.
- This is how you will define it in YAML, key and value separated by a colon(:)
- The keys are fruit, vegetable, liquid and meat and the values are apple, carrot, water and chicken



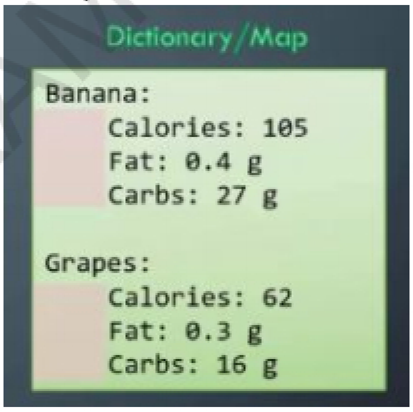
```
Fruit: Apple
Vegetable: Carrot
Liquid: Water
Meat: Chicken
```

- Remember you must have a space followed by a colon differentiating the key and the value.
- Use case of List

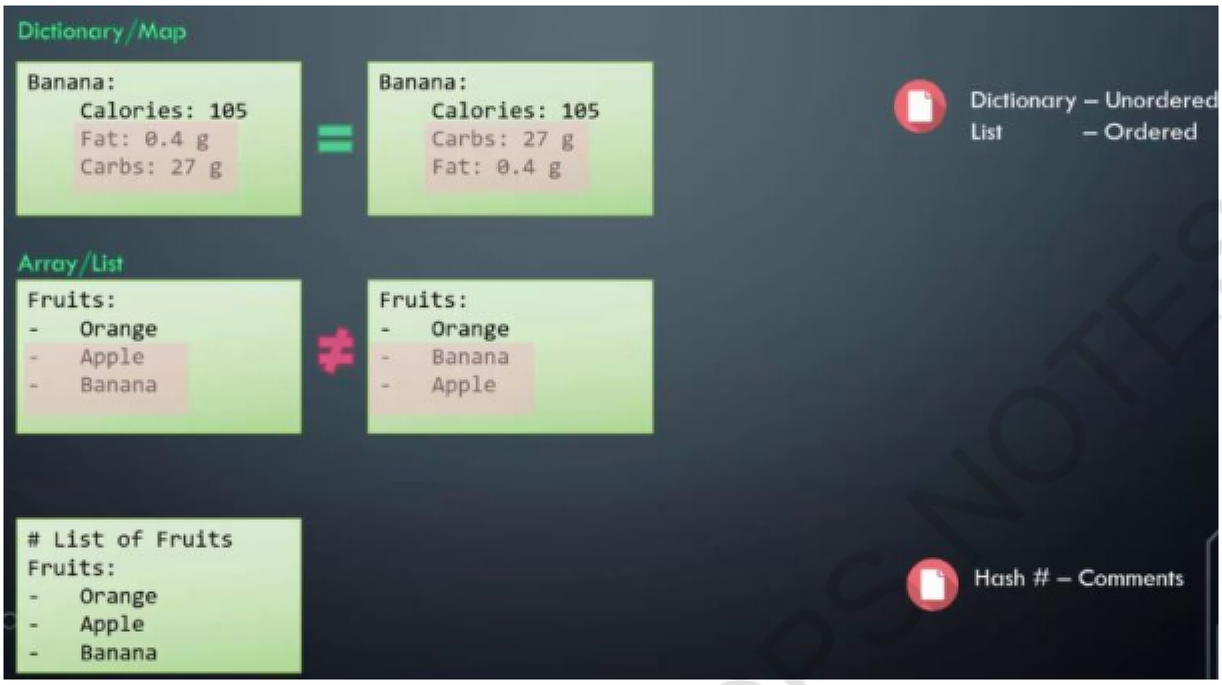


```
Fruits:
-    Orange
-    Apple
-    Banana

Vegetables:
-    Carrot
-    Cauliflower
-    Tomato
```

- Example for use case of Dictionary



```
Banana:
    Calories: 105
    Fat: 0.4 g
    Carbs: 27 g

Grapes:
    Calories: 62
    Fat: 0.3 g
    Carbs: 16 g
```



Dictionary/Map

```
Banana:
    Calories: 105
    Fat: 0.4 g
    Carbs: 27 g
```

=

```
Banana:
    Calories: 105
    Carbs: 27 g
    Fat: 0.4 g
```

Dictionary – Unordered
List – Ordered

Array/List

```
Fruits:
-   Orange
-   Apple
-   Banana
```

≠

```
Fruits:
-   Orange
-   Banana
-   Apple
```

```
# List of Fruits
Fruits:
-   Orange
-   Apple
-   Banana
```

Hash # – Comments

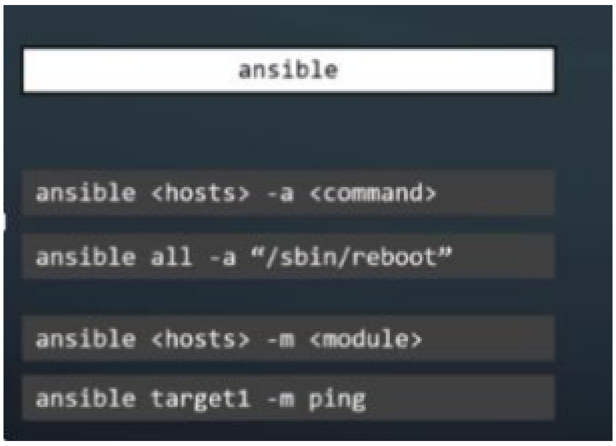
Ansible inventory:

- The Ansible inventory file defines the hosts and groups of hosts .
- The file can be in one of many formats depending on your Ansible environment and plugins.
- The default location for inventory is a file called /etc/ansible/hosts .
- If necessary, you can also create project-specific inventory files in alternate locations.
- The inventory file can list individual hosts or user-defined groups of hosts.
- Below are ansible inventory parameters -

    ansible_host
    ansible_connecton - ssh/winrm/localhost
    ansible_port - 222/5986
    ansible_user - root/administrator
    Ansible_ssh_pass
    Ansible_password

Ad-Hoc commands:

- The Ad-Hoc command is the one-liner ansible command that performs one task on the target host.



```
ansible

ansible <hosts> -a <command>
ansible all -a "/sbin/reboot"

ansible <hosts> -m <module>
ansible target1 -m ping
```

Test environment setup:

- A testing environment is a setup of software and hardware in which the testing team tests a new software build.
- A test environment consists of pre-production or staging environments, and is generally a downgraded version of a production environment to help uncover pre-production defects.
- Building and maintaining a test environment is important.

  Different testing environments:

- Development
- QA == Functional testing of the system
- System Integration Testing == Tests the system from end to end
- User Acceptance Testing = Allows the user to validate the functionality over time
- Production == Production
- Production Parallel == A parallel of production to replicate production issues

Host patterns:

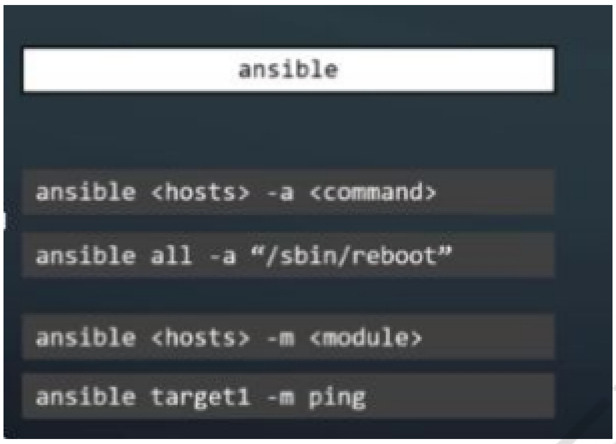
- Patterns is a set of expressions in Ansible that lets us specify concisely which systems a playbook or an ad hoc command must be applied to.
- In this post we will consider most useful pattern expressions and demonstrate them on examples.
- Patterns in Ansible are how we decide which hosts to manage.

```
ansible webservers -m service -a "name=httpd state=restarted"
```

A pattern usually refers to a set of groups (which are sets of hosts) – in the above case, machines in the "webservers" group.

Ad-Hoc commands:

- The Ad-Hoc command is the one-liner ansible command that performs one task on the target host.



```
ansible <hosts> -a <command>
ansible all -a "/sbin/reboot"

ansible <hosts> -m <module>
ansible target1 -m ping
```

Modules:

- Ansible ships with a number of modules (called the 'module library') that can be executed directly on remote hosts using adhoc command mode or through Playbooks.
- Each module in ansible can help to do a particular task.
- Below are few modules that are used in playbook
  - ➢ Package management module
  - ➢ Service module
  - ➢ File module
  - ➢ Command module

Gathering facts:

- Data gathered from target hosts
- Fact gathering means ansible runs a number of commands to confirm the most recent values for important indicators and parameters.
- In Ansible, Facts are nothing but information that we derive from speaking with the remote system.

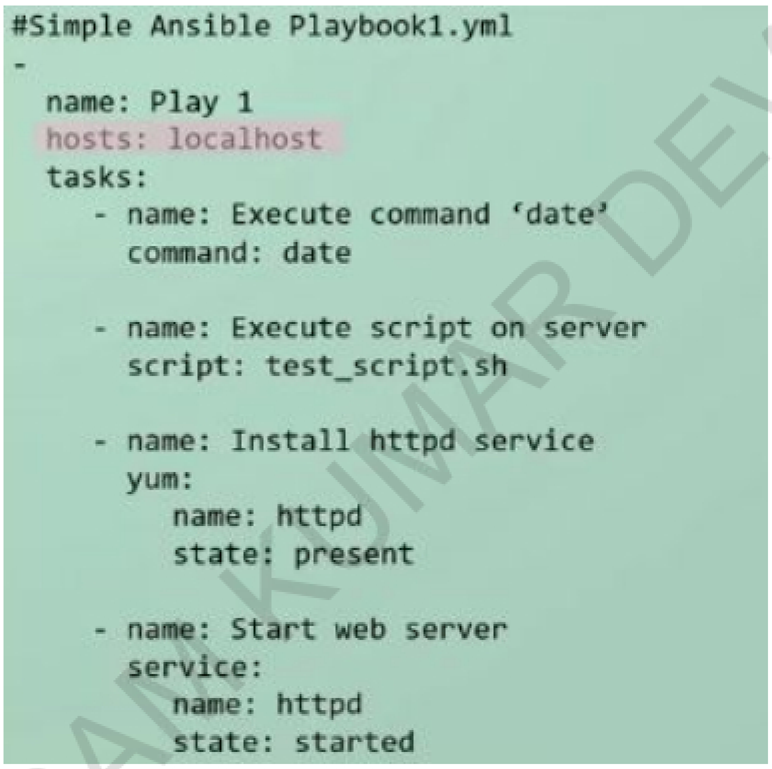
Typical Facts Collected By Ansible:

- hardware parameters of remote system

- storage devices (types, models, sizes, capabilities)
- filesystems and logical volume managers (objects, types, sizes)
- OS distro information
- network devices and full list of their capabilities
- environment variables

Playbooks:

- Ansible playbooks are ansible orchestration language.
- It is in playbooks that contains a set of instructions where we define what we want ansible to do.
- Playbooks are the files where Ansible code is written.
- Playbooks are written using YAML language
- Playbooks are one of the core features of Ansible
- Ansible uses playbooks to describe automation jobs.



```
#Simple Ansible Playbook1.yml
-
  name: Play 1
  hosts: localhost
  tasks:
     - name: Execute command 'date'
       command: date

     - name: Execute script on server
       script: test_script.sh

     - name: Install httpd service
       yum:
          name: httpd
          state: present

     - name: Start web server
       service:
          name: httpd
          state: started
```

Target section:

- The host attribute in ansible playbook is called as Target Section.

Variable section:

- Variables in ansible, used to store value.
- Variables in ansible, are expressed in form of key value pair format.
- Variables are expressed using Vars keyword in ansible.
- A valid variable name is made of the following characters
  - Letters
  - Numbers
  - Underscores
  - A combination of any two or all of the above
- A variable SHOULD ALWAYS START WITH A LETTER and SHOULD NOT CONTAIN ANY SPACES.
- Examples of acceptable variable names include:
  - turntable
  - turn_table
  - turntable01
  - turntable_01
- The names below do not qualify as valid variable names
  - turn table
  - turn-table
  - 01turntable
  - 01
- Example :

```
vars:
  a:   10
```

- We can call value of variable using {{ }}

Task section:

- Task section in ansible playbooks are used to call modules
- We can call any number of modules
- Each task can have a name to it.

Handle section:

- Handlers are just like regular tasks in an Ansible playbook (see Tasks) but are only run if the Task contains a notify directive and also indicates that it changed something.
- For example, if a config file is changed, then the task referencing the config file templating operation may notify a service restart handler.

- Tasks are Ansible's way of doing something and Handlers are our way of calling a Task after some other Task completes.

Dry run:

- Ansible Dry Run or Ansible Check mode
- This feature is to check your playbook before execution like Ansible's --syntax-check feature
- Example:

  ```
  # ansible-playbook <playbook_name> --syntax-check
  ```

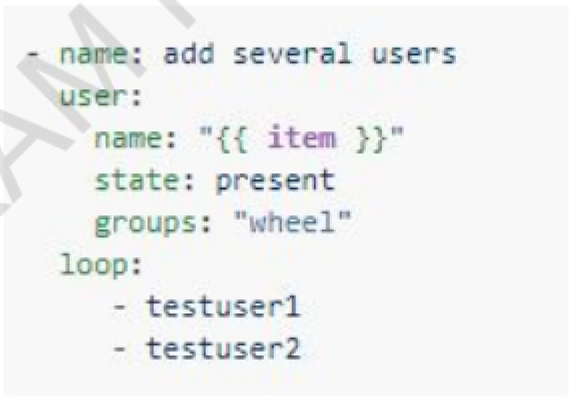
Loops:

- Ansible loop provides a lot of methods to repeat certain tasks until a condition is met.
- Example 1:

  ```
  - name: Ansible Loop example
   yum:
     name: "{{ item }}"
     state: present
    with_items:
    - git
   -  finger
  ```

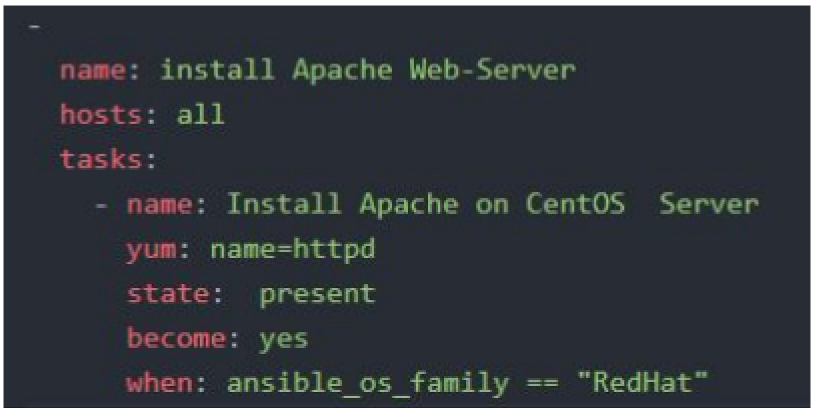
In the above task In each iteration, the value of with_items block will be inserted in place of {{ item }}.

Example 2:



```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
     - testuser1
     - testuser2
```

Conditionals:



```
-
  name: install Apache Web-Server
  hosts: all
  tasks:
    - name: Install Apache on CentOS  Server
      yum: name=httpd
      state:  present
      become: yes
      when: ansible_os_family == "RedHat"
```

Vault:

- Ansible Vault is a feature that allows you to keep all your secrets safe and you can encrypt the secret files
- it is primary useful when we want to store confidential data
- to encrypt secret files in ansible,we use a utility called as Ansible-vault

Create a new vault file	# ansible-vault create <file_name.yml>
Encrypt existing file	# ansible-vault encrypt <file_name.yml>
View Vault file	#ansible-vault view <file_name.yml>
Edit vault file	# ansible-vault edit <file_name.yml>
Decrypt vault file	# ansible-vault decrypt <file_name.yml>

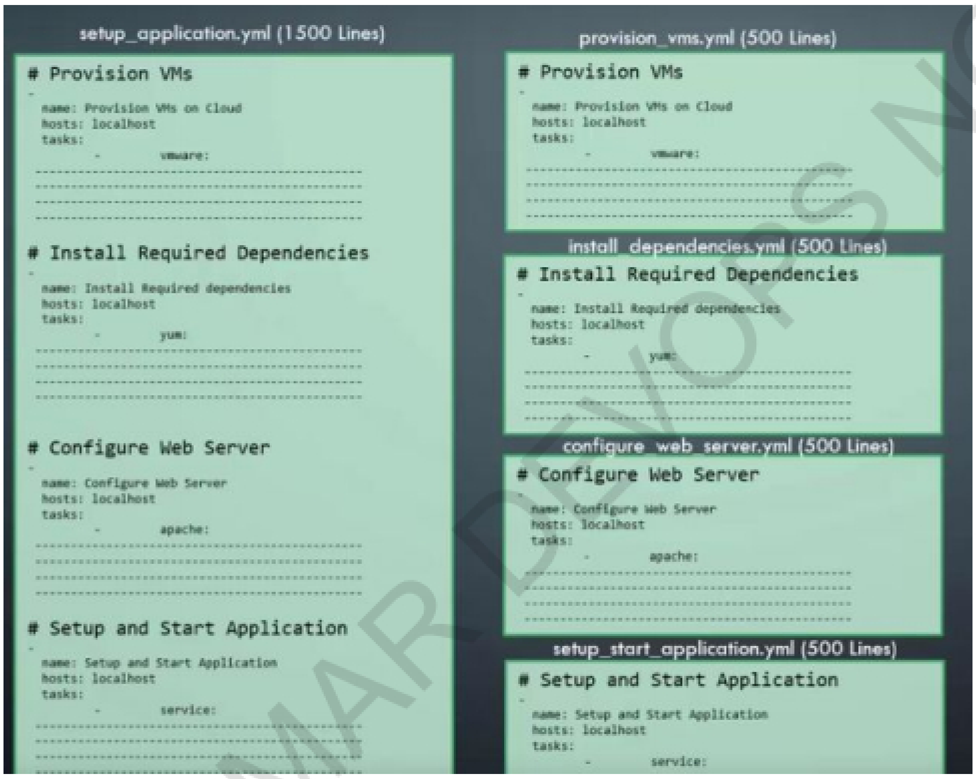
Ansible Patterns:

- Patterns tells how to define what hosts we run tasks against.
- This could be a single host or a group of host.
- We can use additional patterns like use of wild cards as well.

Ansible Roles:

- Roles provide a framework for fully independent, or interdependent collections of variables, tasks, files, templates, and modules.
- In Ansible, the role is the primary mechanism for breaking a playbook into multiple files.
- This simplifies writing complex playbooks, and it makes them easier to reuse.

- Each role is basically limited to a particular functionality or desired output, with all the necessary steps to provide that result either within that role itself or in other roles listed as dependencies.
- Roles are not playbooks. Roles are small functionality which can be independently used but have to be used within playbooks.
- There is no way to directly execute a role. Roles have no explicit setting for which host the role will apply to.



setup_application.yml (1500 Lines)

```
# Provision VMs
-
  name: Provision VMs on Cloud
  hosts: localhost
  tasks:
        -        vmware:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------

# Install Required Dependencies
-
  name: Install Required dependencies
  hosts: localhost
  tasks:
        -        yum:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------

# Configure Web Server
-
  name: Configure Web Server
  hosts: localhost
  tasks:
        -        apache:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------

# Setup and Start Application
-
  name: Setup and Start Application
  hosts: localhost
  tasks:
        -        service:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
```

provision_vms.yml (500 Lines)

```
# Provision VMs
-
  name: Provision VMs on Cloud
  hosts: localhost
  tasks:
        -        vmware:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
```

install_dependencies.yml (500 Lines)

```
# Install Required Dependencies
-
  name: Install Required dependencies
  hosts: localhost
  tasks:
        -        yum:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
```

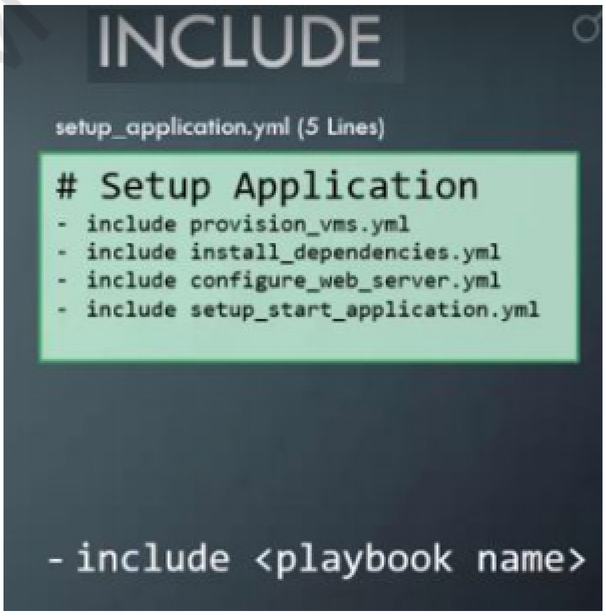
configure_web_server.yml (500 Lines)

```
# Configure Web Server
-
  name: Configure Web Server
  hosts: localhost
  tasks:
        -        apache:
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
```

setup_start_application.yml (500 Lines)

```
# Setup and Start Application
-
  name: Setup and Start Application
  hosts: localhost
  tasks:
        -        service:
```

- Use of include statement



INCLUDE

setup_application.yml (5 Lines)

```
# Setup Application
- include provision_vms.yml
- include install_dependencies.yml
- include configure_web_server.yml
- include setup_start_application.yml
```

```
- include <playbook name>
```