We have two different nodes that are named as control node and managed node. Between them, control node is a machine which runs Ansible and there should be at least one control node whereas the managed node can be any device which is being managed by control node. By connecting all these nodes an Ansible works and there is a small program which can be written in YAML programming language.

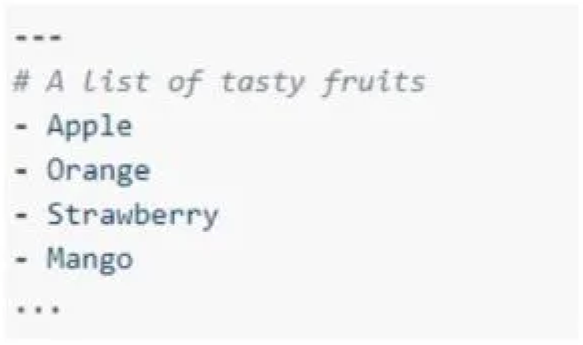
We connect the nodes over SSH and they can be removed once they are no longer required. Previously applications and servers are maintained by the administrators, but the number of application deployments and their enhancements are increasing drastically so Ansible kind of tools came to picture to work with and make the administrator works simple.

To work with Ansible we should be aware of the following terms which are frequently used in Ansible.

PlayBooks -

- contains deployment script
- written on server
- Test on Server - (Dry Run)
- Execute from : Server
- Implemented : client

YAML file always starts with the header of three dashes "—" and ends with three dots "…" formally. So, these can indicate a developer or user that the start and stop points of the YAML scripts.



```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

Yml or Yaml : **Yet Another Markup language**

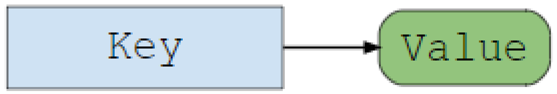
File extensions : .yml or .yaml

Three formats, which exists here which we need to know A YAML format primarily uses 3 node types:

1. Key Value Pair
2. Lists
3. Maps

A **key-value pair** is a simple data structure that consists of a unique identifier (the key) and the corresponding value of that identifier.

The **key** can be any type of data, such as a text string or an integer. The **value** can also be of any type of data, including string, integer, float, boolean, list, or even other key-value pairs.



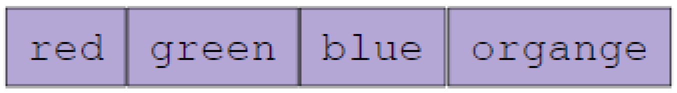
A key-value pair

* data in yml is represented in the form of x: y , called as key value pair.
Example: int a=10 ; here int is data type, a is variable and 10 is value , = is a operator
* No data type concept

```
   a=10        -->    yml --> a: 10
   b=20        -->    yml --> b: 20
   c="india"   -->    yml --> c: "india"
```

Arrays are available in almost all programming languages, including Perl, Python, PHP, Java, C#, Ruby, and JavaScript. They are linear data types used to represent a list of items.

array - collection of similar group of elements/data/data types, stored in a contiguous memory Location on the disk.



Array or list of colors

Example : countries{Russia,USA,India,UK}
countries:
- Russia
- USA
- India
- UK
Example: cars{Maruthi,tata,mahindra}
cars:
- maruti
- tata
- mahindra
Note :

- No need to use double quotes for values here in lists

Lists allows us to enter unlimited values (no memory declaration

Maps/Dictionaries (YAML calls it *mapping*):
The content of a *mapping* node is an unordered set of *key/value* node *pairs*, with the restriction that each of the keys is unique. YAML places no further restrictions on the nodes.

Benefits of Ansible YAML

- It is simple to install Ansible and it is an open source.
- It is simple tool to use and the syntax of Ansible is more user friendly which can be understandable to a new user as well.
- It works as an agentless where we don't worry about installing the agents in the client machines from where we are going to connect for the communication.
- As it has better and powerful features, it can allow a user to model even the most complex workflows in IT. With these capabilities of Ansible we can orchestrate the entire application environment regardless of deployment where we did it.
- It is so efficient as it does not require no extra software for our applications.
- It uses JSON to work around with its modules. So, it can be extensible with the modules which are written in YAML programming.
- It is also used in provisioning of an application.
- It is mainly a configuration management tool so where it can maintain the consistency of a product performance. It is possible by Ansible because it records and updates the detailed information of the both hardware and software.

Playbook - Workflow

Step 1: Write ur code, using yml format or Python.
Step 2 : Push code to server
Step 3: Do Dry Run or Dry Test
Note - ** it will check only syntax-errors only and not logical

Step 4 : Execute Program/playbook
Step 5 : Verify the results. (it is referred as REPORTS/FACTS)

Sample code for execution process:

```
-
  name: play on sample execution
  hosts: client1
  tasks:
          - name: use case of yum module
          yum:
          name: git
          state: present
```

Playbook - Dry Run

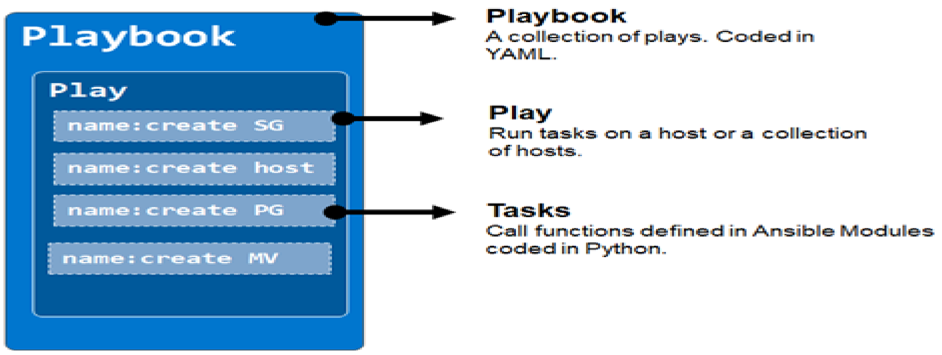
*** command to do syntax-check *** (Ansible - DRY RUN / DRY TEST)

```
# ansible-playbook <pb_name> -i <inventory_file> --syntax-check
```

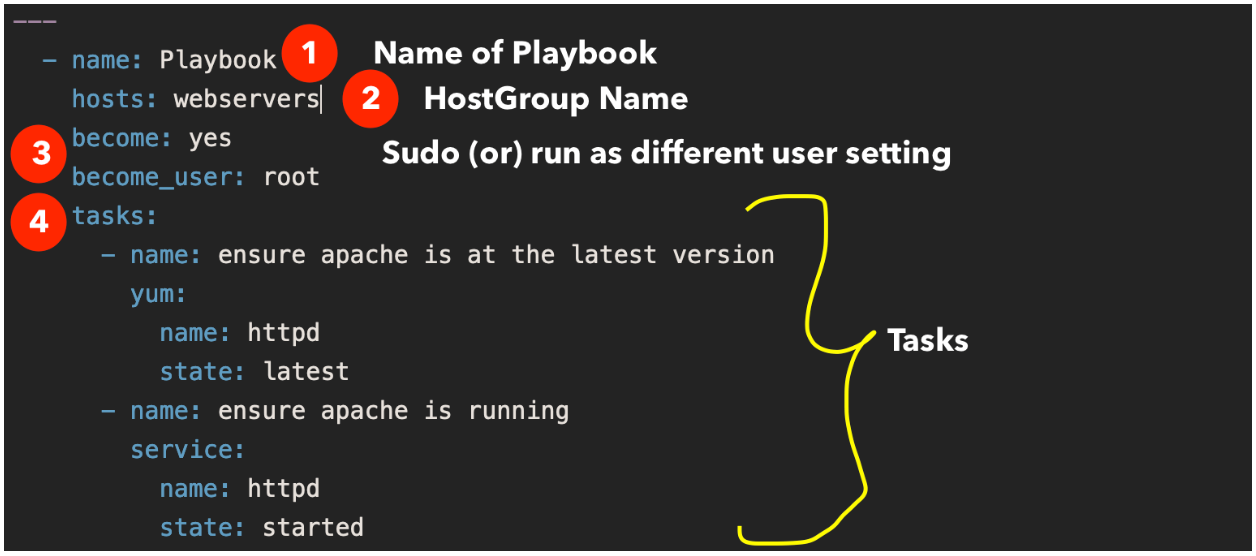
Playbook - Execution

*** command to execute ansible playbooks ***

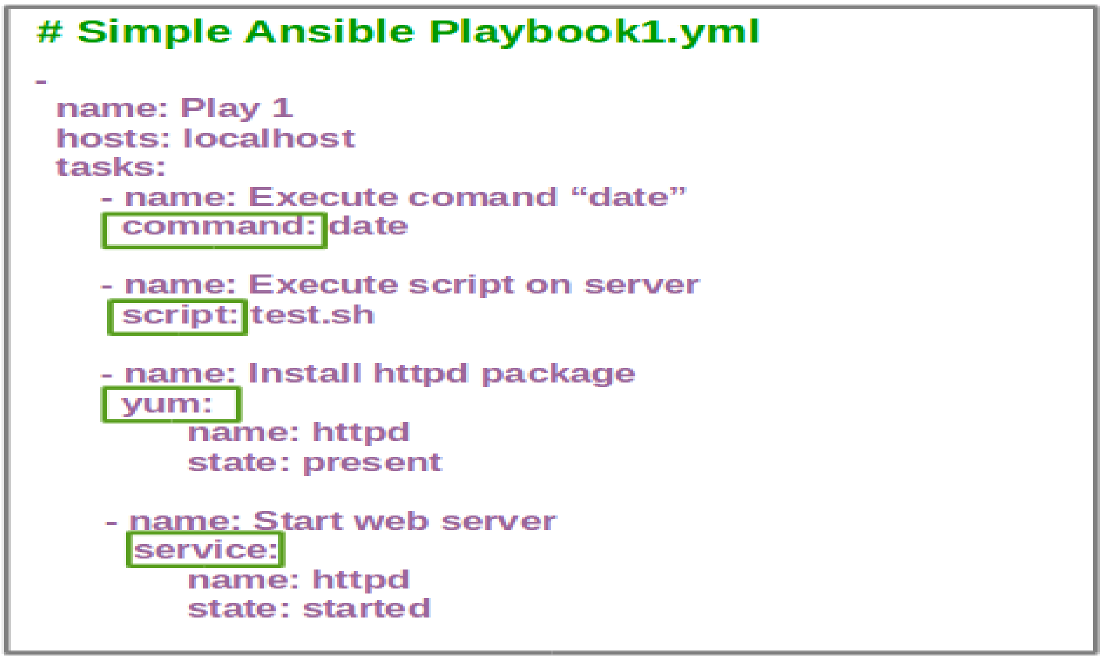
syntax : ansible-playbook <playbook_name> -i <location of inventory file>


Playbook
Play
name:create SG
name:create host
name:create PG
name:create MV
Playbook
A collection of plays. Coded in YAML.
Play
Run tasks on a host or a collection of hosts.
Tasks
Call functions defined in Ansible Modules coded in Python.

cat



```
---
  - name: Playbook
    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
```



```
# Simple Ansible Playbook1.yml

-
 name: Play 1
 hosts: localhost
 tasks:
    - name: Execute comand "date"
      command: date

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

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

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

Example :

```
-
 name: play for install of httpd
 hosts: client1
 tasks:
         - name: httpd install
         yum:
         name: httpd
```

```
            state: absent

-  name: play for install of docker
  hosts: client1
  tasks:
            - name: docker install
            yum:
            name: docker
            state: absent
```

example :

```
-
  name: play on exectuion process
  hosts: client2
  tasks:
            - name: code for install of httpd app
            yum:
            name: httpd
            state: present

            - name: code for use case of service module
            service:
            name: httpd
            state: started
```

*************** Structure of Playbooks ********************************

There are two types of Playbooks.

1. Single Structure PB
2. Multi Structure PB

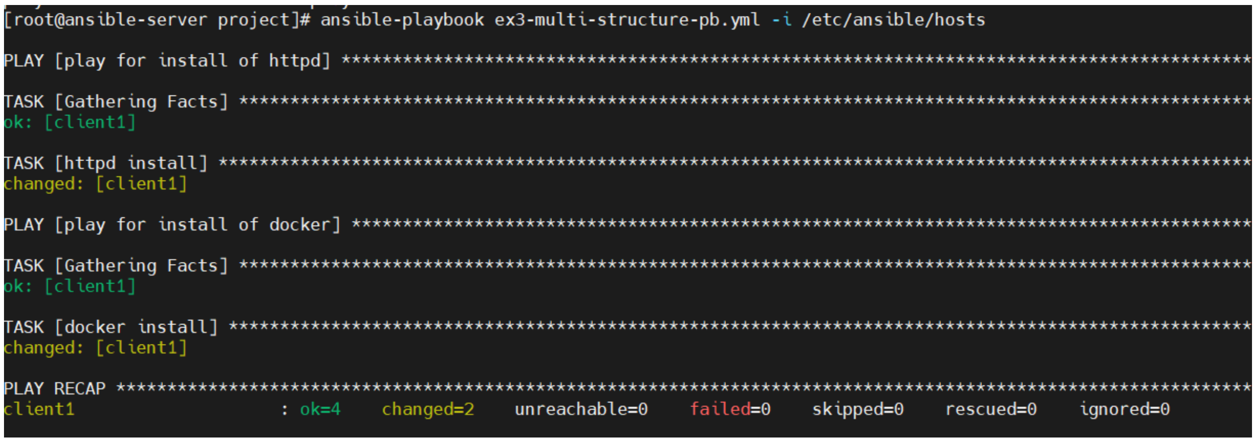
Single Structure PB

```
-
   Name:
   Host:
   Tasks:
      •     Name:
            Module
               Properties

            Module
```

Properties

Module
properties



```
[root@ansible-server project]# ansible-playbook ex3-multi-structure-pb.yml -i /etc/ansible/hosts

PLAY [play for install of httpd] ****************************************************************************************

TASK [Gathering Facts] **************************************************************************************************
ok: [client1]

TASK [httpd install] ****************************************************************************************************
changed: [client1]

PLAY [play for install of docker] ***************************************************************************************

TASK [Gathering Facts] **************************************************************************************************
ok: [client1]

TASK [docker install] ***************************************************************************************************
changed: [client1]

PLAY RECAP **************************************************************************************************************
client1                    : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Understanding of Modules:-

Ansible module are reusable, standalone scripts that can be used by the ansible API, or by the ansible or ansible-playbook programs.

Ansible Modules:

- Cloud
- Clustering
- Command
- File
- Database
- System
- Windows
- Monitoring

- Index of all the modules is available at below URL

https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html

While making a reference to Documentation when we open the module index.

1st - Language level
2nd - OS level → Devop Eng..

- Once, reference module page opens up make sure we are looking at right documentation
- Read below items under module documentation

  1. Synopsis
  2. Requirements
  3. Parameters
  4. Examples*

Exercise - Command Module

Exercise 1:
Create a folder with name "USA" under /opt
hint: command module
Example 1:

```
-
 name: pb on command module
 hosts: client1
 tasks:
         - name: to create a folder USA in /opt
         command: mkdir /opt/USA
```

Example 2:

- Create a folder 'RUSSIA' under /opt
- Create a file 'sample1' under /opt/RUSSIA
- Remove httpd application

```
-
 name: use case on command module
 hosts: all
 tasks:
         - name: code to create directory
         command: mkdir /opt/RUSSIA
         - name: code to create a file
         command: touch /opt/RUSSIA/Sample1
         - name: code to remove application httpd
         command: yum remove -y httpd
```

Exercise :

Write a PB for below -
To install httpd application on Linux clients [hint: command module]

```
  # yum install httpd -y
```
To start httpd services [hint : use command module]
```
  # service httpd start
         or
  # systemctl start httpd
-
 name: pb on command module
 hosts: client1
 tasks:
        - name: install httpd package
        command: yum install httpd -y

        - name: start httpd service
        command: service httpd start
```

Exercise - File Module

Exercise :

1. create a file called notes.txt at /opt
2. need to write a word called "hello world" into that file.

hint: refer to document of file module and copy module.

```
-
 name: pb on file-module
 hosts: client2
 tasks:
        - name: create a file using file-module
        file:
        path: /opt/notes.txt
        state: touch

        - copy:
        content: "HELLO WORLD"
        dest: "/opt/notes.txt"
```

Exercise - file and lineinfile modules

1. Create a file --> russia.txt --> at path /opt/india
2. Make sure that folder india exists in /opt, if not then create it
3. Write the content " HELLO WORLD " into Russia.txt file

4. Make sure russia.txt user has execute permission on the file.
hint : refer to documentation of file and lineinfile modules
* this is a exercise to understand that we have to rearrange the logically task and implement solution

```
-
  name: pb on use case of multiple modules
  hosts: all
  tasks:
            - name: code to create directory using file
            file:
            path: /opt/india
            state: directory

            - name: code to create a file "russia.txt"
            file:
            path: /opt/india/russia.txt
            state: touch

            - name: code to write content "Hello world"
            lineinfile:
            path: /opt/india/russia.txt
            state: present
            line: "Hello World"

            - name: code to give execute permission to the file
            command: chmod u+x /opt/india/russia.txt
```

Exercise - lineinfile module

Q. Replace word “Hello World” in /opt/india/russia.txt with “India is great”

Hint: lineinfile module

```
-
  name: pb for to replace content
  hosts: all
  tasks:
            - name: replace content
            lineinfile:
            path: /opt/india/russia.txt
            state: present
```

```
        regexp: "Hello World"
        line: "india is great"
```

Variables

** in ansible we use a keyword/property/attribute "vars" is used to declare the variables
** purpose : to store data/value
  ex:(in a c language)
                int a=10

int	data type
a	variable
=	assignment operator
10	value

Normal, a=10 here in ansible -> yml --> a: 10
** the variables should be declared in key value pair format.
** there is no concept of Data Types in ansible.
** variables are case sensitive
  call the variable value {{}} ==> jinja templating...

```
  a: 10     {{a}}  --> o/p : 10
  b: 20      {{b}}  --> o/p : 20
```

** variables should be declared using a parameter called vars
Sample program on vars:
example 1:

```
-
 name: playbook on use case of variables
 hosts: pc1
 vars:
        a: 10
        b: 20
 tasks:
        - name: below code to create a file if does not exist
        file:
        path: /opt/variables.txt
        state: touch
        - name: below code to write the content into the file
        lineinfile:
        path: /opt/variables.txt
```

```
            state: present
            line: "The value of A is {{a}} and the value of B is {{b}}"

example 2:
-
  name: play for use case of variables
  hosts: client1
  vars:
            a: 10
            b: 20
            c: "india is a great country"
  tasks:
            - name: below code is to create the file
            file:
            path: /opt/01-sep-2021
            state: touch

            - name: below coce is example of vars
            lineinfile:
            path:  /opt/01-sep-2021
            state: present
            line: "The value of A is: {{a}} and the value of B is: {{b}}, the value of c is: {{c}}"
Example 3:
-
  name: pb for variable concept
  hosts: all
  vars:
            a: 10
            b: 20
            c: "WELCOME TO INDIA"
  tasks:
            - name: below code to create a directory
            file:
            path: /opt/var-concept
            state: directory
            - name: below to create a file data.txt
            file:
            path: /opt/var-concept/data.txt
            state: touch
            - name: below code to write variables into the file
            lineinfile:
```

```
        path: /opt/var-concept/data.txt
        state: present
        line: " The value of A is {{a}} \n The value of B is {{b}} \n The value of C is {{c}}"
```

Loops

- undergoing a lot of changes
- versions of ansible - syntax loops are different.They look more similar to a language.
- loops, we will use along with modules
- not every module will work with loops

** ansible -- reduce the dependencies on loops concept

Example 1: past

```
-
  name: pb on use case of loops
  hosts: client1
  tasks:
        - name: below code for yum installation
        yum:
        name: "{{ item }}"
        state: present
        with_items:
        - git
        - docker
        - net-tools
        - finger
        - httpd
```

*** Note: format followed for with_items is "list/arrays"

Example 2: future (2.11)

```
 -
  name: example on loops
  hosts: client1
  tasks:
        - name: Ansible Loop example
        yum:
        name: ['git', 'finger', 'docker','httpd', 'net-tools']
        state: present
```

Example 3: Current (2.9)

```
-
  name: example on loops
```

```
hosts: client1
tasks:
        - name: Ansible Loop example
        yum:
        name: "{{ item }}"
        state: absent
        loop:
        - git
        - finger
        - docker
        - httpd
        - net-tools
```