++
++
Today we will discuss about host scope variables
Lets create one more managed node.
So, we will have 1 controller 4 nodes.
In step 6 -- Add rule -- All Traffic -- Anywhere

Establish password less ssh connection
$ sudo passwd ubuntu
(lets give the password as ubuntu only)

$ sudo vim /etc/ssh/sshd_config

change
PasswordAuthentication yes
Save and QUIT

```
$ sudo service ssh restart
$ exit

++++++++++++++++
Now,  Connect to controller
Now , We need to generate ssh
connections
$ ssh-keygen

Now copy the key to managed nodes

$ ssh-copy-id ubuntu@172.31.43.66  (
private Ip of server4 )


++++++++++
Now, we need to add the information
of managed nodes in the inventory
file.
Location of inventory file
/etc/ansible

$ cd /etc/ansible
```

```
$ ls
$ sudo vim hosts
insert the private ip addresss of
4th  server
save and quit

$ ansible all  -a  'ls  -la'    (
you will get the list of the files
in all managed nodes )

$ ansible all  -a  'free'
++++++++++++++++++
We can do grouping using
[groupname]

Ex:

To do grouping

$ sudo vim hosts

[webserver]
172.31.11.96
172.31.6.207
```

```
[appserver]
172.31.12.138
[dbserver]
172.31.31.161

++++++++++++++++++++

$ ansible appserver  -a 'free'    (
It runs on one machine
172.31.12.138)

$ ansible webserver  -a 'free'   (
It runs on two machines )

$ ansible all  -a 'free'

++++++++++++++++++++++++

We can perform grouping on groups



Host scope variables
--------------------
```

These variables are classified into 2 types
1) Variables to work on group of hosts
2) Variables to work on single hosts

Variables to work on group of hosts

These variables are designed to work on group of hosts.
They are definined in a folder called group_vars
This group_vars folder should be presnent in the same folder where all the playbooks are present.
In this group_vars folder, we should create a file who's name is same as group_name in Inventory file.
In this file we create variables.

Varible which works on group of hosts

```
$ cd  ( enter)
$ cd playbooks
$ ls
```

Varibles which work in group of hosts are divided into two types
1) Variables which work in group of machines
2) Variables which work on one machine

Variables which work in group of machines

```
-----------------------------------
-----------
playbooks$ mkdir group_vars
```

Note: group_vars folder should be present in the same location of playbook files.

```
$ cd group_vars
$  vim webserver
```

```
a: Prakash
b: logiclabs
c: /home/Prakash
d: 67809
e: /bin/bash

Save and Quit

$ cd ..
playbooks$ vim playbook8.yml

---
- name: Using host scope variables
  hosts: webserver
  tasks:
   - name: User creation
     user:
      name: "{{a}}"
      password: "{{b}}"
      home: "{{c}}"
      uid: "{{d}}"
      shell: "{{e}}"
```

```
...

save and quit

TO run the playbook
$ ansible-playbook playbook8.yml  -b
 ( It runs on two machines)



++++++++++++++++++++++++++++++

Lets add few more variables

$ cd group_vars
$ vim webserver

a: Prakash
b: durgasoft
c: /home/Prakash
d: 67809
e: /bin/bash
f: tree
g: present
```

```
h: no

save and quit

$ cd ..
$ vim playbook9.yml

---
- name: Using host scope variables
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{f}}"
      state: "{{g}}"
      update_cache: "{{h}}"
...

$ ansible-playbook  playbook9.yml
-b



++++++++++++++++++++++++++
```

Variables to work on single hosts

Variables to work on single hosts
These variables are designed on single machine.
Thet are created in folder called host_wars
This host_wars folder should be created in the same location of where the playbooks are present.

```
playbooks$ mkdir host_vars
$ cd host_vars
$ vim 172.31.40.253          (
172.31.40.253  private Ip of server4
)

a: firewalld
b: present
c: yes

save and quit
$ cd ..
```

```
$ vim playbook10.yml

---
- name: Use host scope variables
  hosts: 172.31.40.253
  tasks:
   - name: Install firewall
     apt:
      name: "{{a}}"
      state: "{{b}}"
      update_cache: "{{c}}"
...

save and quit

$ ansible-playbook  playbook10.yml
-b


++++++++++++++++++++++++++++++++++++++++
+++++++

Implementing loops
```

Notes: Modules in ansible can be executed multiple times using loops.

```
$ vim playbook11.yml

- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item}}"
      state: present
      update_cache: no
     with_items:
      - tree
      - git
      - default-jdk
      - apache2
...

$ ansible-playbook  playbook11.yml
-b
```

Ex: Playbook to install diffrent s/w packages

```
$ vim playbook11.yml

- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item}}"
      state: present
      update_cache: no
     with_items:
      - tree
      - git
      - default-jdk
      - apache2
...
```

++++++++++++++++++++++++++

Ex: Playbook to install diffrent s/w packages

$ vim playbook11.yml

```
- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item}}"
      state: present
      update_cache: no
     with_items:
      - tree
      - git
      - default-jdk
      - apache2
...
```

++++++++++++++++++++++++++

Requirement:
Tree needs to be installed
Git needs to be unintalled
jdk needs to be updated
apache needs to be installed and
update cache

```
$ cd playbooks
$ vim playbook12.yml

---
- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item.a}}"
      state: "{{item.b}}"
      update_cache: "{{item.c}}"
     with_items:
      - {a: tree,b: present,c: no}
      - {a: git,b: absent,c: no}
      - {a: default-jdk,b: absent,c:
no}
```

```
      - {a: apache2,b: present,c:
yes}
...
save and quit

$ ansible-playbook  playbook12.yml
-b

++++++++++++++++++++++++++
Ex: For working on multiple modules
with multiple with_items.


Requirement: To create multiple
users and files/directories in
user's home directories.

$ vim playbook13.yml
---

---
- name: Create users and create
files/dir in users home dir
  hosts: all
```

```
  tasks:
   - name: Create multiple users
     user:
      name: "{{item.a}}"
      password: "{{item.b}}"
      home: "{{item.c}}"
     with_items:
      - {a: Farhan,b: durgasoft,c:
/home/Farhan}
      - {a: Ravi,b: durgasoft,c:
/home/ubuntu/Ravi}
   - name: creating files and
directories in users home dir
     file:
      name: "{{item.a}}"
      state: "{{item.b}}"
     with_items:
      - {a: /home/Farhan/file1,b:
touch}
      - {a:
/home/ubuntu/Ravi/dir1,b: directory}
...

save and quit
```

```
$ ansible-playbook  playbook13.yml
-b

---------
To check , user is created or not?
$ ssh 172.31.11.96
$ vim /etc/passwd

TO check files and dir  are created
or not

$ cd /home/Farhan
$ ls     ( we can see the file)
$ cd
$ pwd
$ cd Ravi
$ ls  ( we can see the dir )
$ exit



++++++++++++++++++
Handlers
```

Handler is a piece of code which is executed, if some other module is executed successfully and it has made some changes.

Handlers are always executed only after all the tasks are executed. Handlers are executed in the order that are mentioned in the handler section, and not in the order they are called in the tasks section. Even if handler is called multiple times in the tasks section, it will be executed only once.

Requirement:

$ vim playbook14.yml

```
---
- name: Confugure apache2 using
```

```
handlers
  hosts: all
  tasks:
   - name: Install apache2
     apt:
      name: apache2
      state: present
   - name: Edit index.html file
     copy:
      content: "Logiclabs\n"
      dest: /var/www/html/index.html
     notify: Restart apache2
  handlers:
   - name: Restart apache2
     service:
      name: apache2
      state: restarted
...

$ ansible-playbook  playbook14.yml
-b
```

Note:
As editing the index.html file is

successfull, handler is executed.

If you re run the playbook, handler is not executed.

++++++++++++++++++++++++
Error Handling

If any module fails in ansible,the execution of the playbook terminates over there.

When we know that certain module might fail, and still we want to continue playbook execution, we can use error handling.

The section of code which might generate an error should be given in block section.

If it generates an error, the control comes to rescue section.

Always section is executed every time, irespective of whether the block is successfull or failure.

```
$ vim playbook15.yml
```

```
---
- name: Error handling
  hosts: all
  tasks:
   - block:
      - name: Install apache1
        apt:
         name: apache1
         state: present
     rescue:
      - name: Install apache2
        apt:
         name: apache2
         state: present
     always:
      - name: Check url response
        uri:
```

```
          url: "{{item}}"
        with_items:
          - http://172.31.34.91
          - http://172.31.33.68
          - http://172.31.34.60
          - http://172.31.40.253
...

$ ansible-playbook  playbook15.yml
-b
```