Ansible

Ansible Foundation With Hands on Labs

By: Mohamed Mekawy Emam

Table of contents:

Introduction

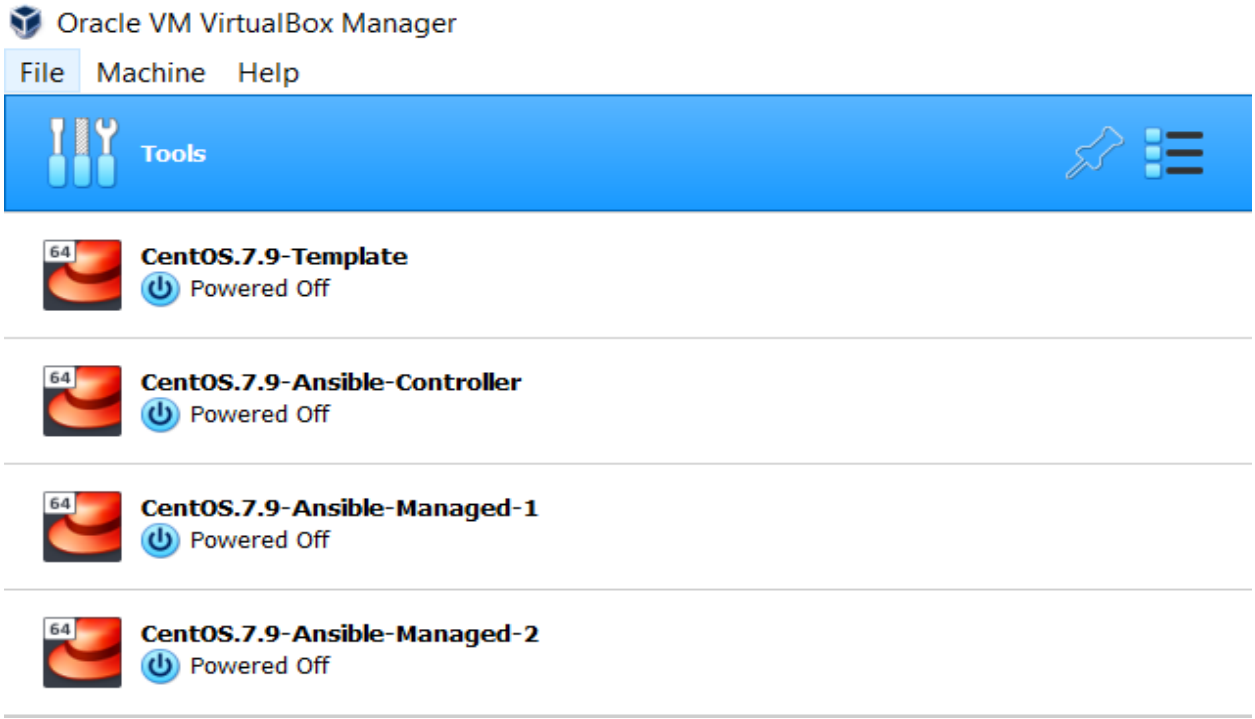
Since it was first released Ansible has become one of the most used systems for configuration management. And through this journey will show you how to get started with Ansible.

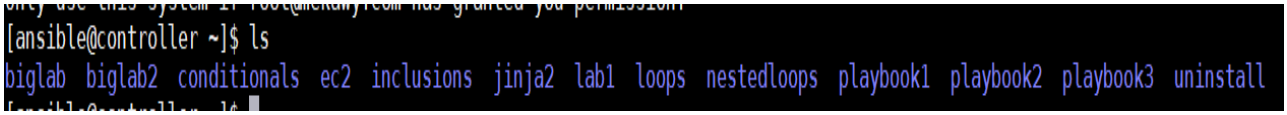
Ansible Foundation, you'll learn about the basics of working with Ansible. This includes setting up an environment that is managed with Ansible, working with modules and playbooks, and working with Jinja2 templates.

Lab Requirements

1. MobaXterm Home Edition [Free]
2. Oracle VirtualBox-6.1.20-143896-Win [Free]
3. Oracle_VM_VirtualBox_Extension_Pack-6.1.20 [Free]
4. CentOS Linux release 7.9.2009 (Core) minimal [Free]

Lab Setup





1.First, you need to install Ansible software.

-Only the control node needs the Ansible package itself.

-CentOS and like, Red Hat Enterprise Linux need the EPEL repository.

-Managed hosts need Python and SSH and that's it

2.Next, you need to create a non-root user. This is the user account that you are going to use to run your tasks with Ansible.

3.Following that, you will set up SSH for communication for that specific user and that involves using ssh-keygen to generate a key pair and using ssh-copy-id to the user at the remote host.

This is something that you need to do for all hosts that are going to be managed.

4.finally you need to configure sudo.

. Basically you would put a snap in file in etc/sudoers.d/<username> with the name username and in the snap in file {i.e. ansible}

vim /etc/sudouers.d/ansible

ansible ALL=(ALL) NOPASSWD: ALL

-**Notice** by the way that Ansible does not provide any solution to push this information to the managed hosts. Managed hosts need to be configured.

-**Before doing anything** we will need to set up hostname resolution. Normally you would use DNS to do that. I don't have DNS so I'm going to set up etc hosts.and use secure copy to copy hosts file to managed machines.

scp /etc/hosts <managed machine name>:/etc/

Installation on Controller Node

1. Install Python 2.x or Python 3 .x for Ansible 2.4 or later
2. Install epel-release repository
3. Install Ansible from EPEL
4. Create a non-root user and perform all Ansible tasks as non-root user using sudouers file

Installation on Managed Nodes

1. 1.Install Python 2.x or Python 3 .x for Ansible 2.4 or later
2. 2.Install epel-release repository
3. 3.Install Ansible from EPEL
4. 4.Create a non-root user and perform all Ansible tasks as non-root user {Notice that the local user name and the remote user name do not have to be the same, but it's convenient if you have the same user names on the Controller machine as well as on the Managed machines.}

5. 5.Setup SSH Communication {to set up SSH communications. On the next slide, I'm outlining what needs to be done in order to configure SSH. So to configure SSH, you'll start by running ssh-keygen. This creates a public key as well as a private key. The server that has the public key sends a challenge that can only be answered with the private key. And that means that as an administrator, you will keep the private key in the local Ansible user account on the Controller node and you send the public key to the remote server.As a result, the public key will be written to a file with the name authorized_keys which is in the .ssh directory in the target user home directory ~/.ssh/authorized_keys}

   ssh-keygen

   ssh-copy-id user@remotehost

Initialization with no DNS only depends on /etc/hosts file

Installation on Controller Node

#common installation setup on controller and managed nodes

[root@controller ~]#yum -y install bash-completion bash-completion-extras net-tools NetworkManager vim wget createrepo gpm zip unzip epel-release

[root@controller ~]# visudo

Allows people in group wheel to run all commands

%wheel ALL=(ALL)   ALL

[root@controller ~]# useradd ansible

[root@controller ~]# passwd ansible   #password is ansible

[root@controller ~]# usermod -aG wheel ansible

[root@controller ~]# id ansible

uid=1001(ansible) gid=1001(ansible) groups=1001(ansible),10(wheel)

[root@controller ~]# yum -y install python2 python3 epel-release

#Install Ansible on controller only

[root@controller ~]# yum -y install ansible

#Setup /etc/hosts file as we do not have DNS server and secure copy to managed nodes

[root@controller ~]# vim /etc/hosts

192.168.1.7 controller.mekawy.com controller

192.168.1.8 managed1.mekawy.com managed1

192.168.1.9 managed2.mekawy.com managed2

[root@controller ~]# scp /etc/hosts managed1:/etc/

[root@controller ~]# scp /etc/hosts managed2:/etc/

#Setup SSH Communication :: Logout and log in as ansible user on controller

[ansible@controller ~]$ ssh controller

[ansible@controller ~]$ exit

[ansible@controller ~]$ ssh controller.mekawy.com

[ansible@controller ~]$ exit

[ansible@controller ~]$ ssh managed1

[ansible@managed1 ~]$ exit

[ansible@controller ~]$ ssh managed1.mekawy.com

[ansible@managed1 ~]$ exit

[ansible@controller ~]$ ssh managed2

[ansible@managed2 ~]$ exit

[ansible@controller ~]$ ssh managed2.mekawy.com

[ansible@managed2 ~]$ exit

#Setup a passwordless ssh login

[ansible@controller ~]$ ssh-keygen

file in which to save the key (/home/ansible/.ssh/id_rsa)

[ansible@controller ~]$ ssh-copy-id managed1.mekawy.com

[ansible@controller ~]$ ssh-copy-id managed2.mekawy.com

[ansible@controller ~]$ ssh-copy-id controller.mekawy.com

/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansible/.ssh/id_rsa.pub"

[ansible@controller ~]$ ssh managed1

[ansible@controller ~]$ ssh managed2

Installation on Managed Node 1

[root@managed1 ~]# visudo

Allows people in group wheel to run all commands

%wheel ALL=(ALL)   ALL

[root@managed1 ~]# useradd ansible

[root@managed1 ~]# passwd ansible   // password is ansible

[root@managed1 ~]# usermod -aG wheel ansible

[root@managed1 ~]# id ansible

uid=1001(ansible) gid=1001(ansible) groups=1001(ansible),10(wheel)

[root@managed1 ~]# yum -y install python2 python3 epel-release

Installation on Managed Node 2

[root@managed2 ~]# visudo

Allows people in group wheel to run all commands

%wheel ALL=(ALL)   ALL

[root@managed2 ~]# useradd ansible

[root@managed2 ~]# passwd ansible   // password is ansible

[root@managed2 ~]# usermod -aG wheel ansible

[root@managed2 ~]# id ansible

uid=1001(ansible) gid=1001(ansible) groups=1001(ansible),10(wheel)

[root@managed2 ~]# yum -y install python2 python3 epel-release

Ansible Inventory file

Inventory in Ansible is a list of managed devices and you can create a static inventory file or use dynamic inventories.

In inventory, inventory groups can be used to group specific device types.So you can create a group for your routers, for your Windows machines, and whatever.

Inventory parameters

If hosts need specific parameters to connect to them, you can use specific inventory parameters, like

ansible_host: which is the hostname or IP address to connect to

ansible_port: which is the port to SSH into

ansible_user: which is a per host user that can be used to SSH into.

ansible_password: which can be used to set the password for an SSH connection.

Defaults of these parameters typically are set in ansible.cfg. Overrides for specific hosts can be set in the static inventory.

#example

[ansible@controller ~]$ mkdir vagrant

[ansible@controller ~]$ cd vagrant/

[ansible@controller vagrant]$ vim inventory

[vagrant]

vagrant1 ansible_host=127.0.0.1 ansible_port=2022

vagrant1 ansible_host=127.0.0.2 ansible_port=2023

Inventory types

1. Static inventory
2. Dynamic inventory
3. Multiple inventories

Static inventory

Is good for small environments.The default static inventory is in etc/ansible/hosts but it is common to use product-based inventories that are specified in the ansible.cfg OR using the -i option that adds a nice additional layer of flexibility.

Dynamic inventory

Is required in large and dynamic environments, especially if you are in public cloud for example.

Dynamic inventory you need an executable script. Often the script comes with an ini file that is used as a configuration file for that script

When using the ansible command, you can use -i followed by the name of the dynamic inventory script you want to run. If the file is executable, it will be treated as a dynamic inventory.

Alternatively, you can specify the location of dynamic inventory in the ansible.cfg file, but that is not as flexible.

Different scripts are available for different environments. And you can easily get them from the ansible GitHub location https://github.com/ansible/ansible/tree/devel/contrib/inventory

Documentation of dynamic inventory can be found on ansible.doc https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html

It's also possible to develop your inventory scripts.

#example : ansible git repository aws ec2 dynamic inventory https://github.com/vshn/ansible-dynamic-inventory-ec2

[ansible@controller ~]$ mkdir ec2

[ansible@controller ~]$ cd ec2/

[ansible@controller ec2]$ wget https://github.com/vshn/ansible-dynamic-inventory-ec2/archive/refs/heads/master.zip

[ansible@controller ec2]$ ls

master.zip

[ansible@controller ec2]$ unzip master.zip

Archive: master.zip

2265b75b83b741f42c19c68226f262f87f0ac8cd

  creating: ansible-dynamic-inventory-ec2-master/

inflating: ansible-dynamic-inventory-ec2-master/LICENSE

inflating: ansible-dynamic-inventory-ec2-master/Makefile

inflating: ansible-dynamic-inventory-ec2-master/README.md

inflating: ansible-dynamic-inventory-ec2-master/ec2.ini

inflating: ansible-dynamic-inventory-ec2-master/ec2.py

creating: ansible-dynamic-inventory-ec2-master/redhat/

inflating:
ansible-dynamic-inventory-ec2-master/redhat/ansible-dynamic-inventory-ec2.spec

[ansible@controller ec2]$ ls

ansible-dynamic-inventory-ec2-master master.zip

[ansible@controller ec2]$ cd ansible-dynamic-inventory-ec2-master/

[ansible@controller ansible-dynamic-inventory-ec2-master]$ ls

ec2.ini ec2.py LICENSE Makefile README.md redhat

[ansible@controller ansible-dynamic-inventory-ec2-master]$ vim ec2.ini

[ansible@controller ansible-dynamic-inventory-ec2-master]$ vim ec2.py

Dynamic Inventory Platforms

Scripts are available for multiple platform

-Scripts are available for multiple platform

-public clouds such as AWS, Azure, Google Compute Engine

-virtualization platforms such as vSphere or oVirt

-Platform as a Service solution such as OpenShift

-Management solutions such as Spacewallk or Katello

Multiple inventories

You can use multiple inventories as well. If directory name is passed as the inventory all files in that directory are used as the inventory files. And that makes it convenient to work with multiple environments, just put them in the directory and refer to the directory instead of the individual files -i <directory path>/.

So in a directory, you can even mix dynamic inventory files with static inventory files. If you are using multiple inventory files you should make sure though that there are no dependencies between files and all files are self-contained.

#example : Configure dynamic inventories of your Azure resources using Ansible

https://docs.microsoft.com/en-us/azure/developer/ansible/dynamic-inventory-configure?tabs=ansible

Notice you need to make sure that you have supporting modules. The default Ansible packages don't offer support for all of the external modules that could be used, so you need to "pip install <whatever>".

Inventory file setup of first lab

#Inventory file setup of first lab

[ansible@controller ~]$ mkdir lab1

[ansible@controller ~]$ cd lab1/

[ansible@controller lab1]$ vim inventory

[all]

controller.mekawy.com

managed1.mekawy.com

managed2.mekawy.com

[ansible@controller lab1]$ ansible all -i inventory --list-hosts

hosts (3):

controller.mekawy.com

managed1.mekawy.com

managed2.mekawy.com

Ansible Configuration file ansible.cfg

1. Generic file /etc/ansible/ansible.cfg
2. User specific file ~/.ansible.cfg
3. The ansible.cfg file in the project directory (takes precedence)

-So it's common practice to put the configuration file in the project directory; alternatively, you could also specify the $Ansible_config environment variable to specify where you want to look for the configuration file.

No matter where you put the configuration file, the Ansible Configuration File that you are using should contain all of the environment variables.

-To find out which configuration file you are using, you can use the ansible-v command

become : specifies how to escalate privileges on managed hosts

become_user : which specifies which user account to use on the remote host

become_ask_pass : specifies whether or not a password should be asked for when becoming another user

inventory : specifies which inventory file to use

remote_user : is the name of the user account that can be used on a remote machine which is not set by default, which results in the local username being used if you don't specify anything.and if remote user is not specified, "privilege escalation" can be used.

Privilege escalation

Privilege escalation is defined in a specific section in the Ansible Configuration File; typically, it contains parameters like the four that we discussed above.

Enable [privilege_escalation]section in ansible.cfg

become=True

become_method=sudo

become_user=root

become_ask_pass=fals

1. In order for the privilege escalation to work, you also do need to create a sudo configuration
2. For the control node Ansible default account, create a sudo file on all Ansible managed hosts

cat /etc/sudouers.d/ansible

ansible ALL=(ALL) NOPASSWD: ALL

privilege_escalation setup

#Setup privilege_escalation on controller on lab1 directory using ansible user

[ansible@controller lab1]$ vim ansible.cfg

[defaults]

remote_user=ansible

host_key_checking=false

inventory=inventory

[privilege_escalation]

become=True

become_method=sudo

become_user=root

become_ask_pass=False

#OR Modify /etc/ansible/ansible.cfg to enable escalation in the generic directory using root user

[root@controller ~]# vim /etc/ansible/ansible.cfg

[privilege_escalation]

become=True

become_method=sudo

become_user=root

become_ask_pass=False

[ansible@controller lab1]$ sudo vim /etc/sudoers.d/ansible

ansible ALL=(ALL) NOPASSWD: ALL

#Setup Privilege_escalation on managed node 1

[root@managed1 ~]# vim /etc/sudoers.d/ansible

ansible ALL=(ALL) NOPASSWD: ALL

#Setup Privilege_escalation on managed node 2

[root@managed2 ~]# vim /etc/sudoers.d/ansible

ansible ALL=(ALL) NOPASSWD: ALL

Ansible task management methods

1. Ad-hoc commands
2. Ansible playbook written in YAML

Modules

The exact module location depends on the Linux distribution.

On CentOS 7.9, which is the platform I'm using in this course, you can find them in /usr/lib/python2.7 /site-packages/ansible/modules.

[root@controller ~]# ls /usr/lib/python2.7/site-packages/ansible/modules

Common Modules

1. command : runs commands on a managed host
2. shell : runs commands on a managed host through local shell
3. copy : copy file, change content on a remote host in a target file

Module types

1. Core modules
2. Extra modules
3. Custom modules

[ansible@controller lab1]$ ansible all -i inventory -m command -a id

[ansible@controller lab1]$ ansible all -i inventory -m command -a id -o

[ansible@controller lab1]$ ansible all -i inventory -m command -a hostname

[ansible@controller lab1]$ ansible all -i inventory -m command -a env

[ansible@controller lab1]$ ansible all -i inventory -m shell -a env

[ansible@controller lab1]$ ansible managed1.mekawy.com -i inventory -m copy -a 'content="Ansible Managed\n" dest=/etc/motd'

Ansible Documentation

1. docs.ansible.com
2. ansible-doc -l will show all modules currently installed.
3. ansible-doc <module name> will provide specific information.
4. ansible-doc -s <module name> will produce example codes that you can use in a playbook.

[root@controller ~]# ansible-doc -l

[root@controller ~]# ansible-doc -l | grep -i nmcli

[root@controller ~]# ansible-doc nmcli

[root@controller ~]# ansible-doc -s nmcli

Ansible Playbooks

playbooks are configuration files that include plays that instruct Ansible what it should do.

Notice that the playbook can have one or multiple plays.

It will always have one play. And a play consists of different tasks.

And in these tasks, modules are called. And the module explains to Ansible which specific configuration to be managed or changed.

Playbooks are written in the YAML format.

A playbook, it is nice if it starts with three dashes. The three dashes in the beginning and the three dots at the end are here to make it possible to imbed the YAML code in something else. And then the YAML parts will easily recognize the code it needs to work with.

Playbook Formatting

1. Multiline formatting

2. Dictionary formatting
3. Block Formatting

To verify YAML file syntax

ansible-playbook --syntax-check mycode.yaml #for syntax check

ansible-playbook -C mycode.yaml #for a dry run

ansible-playbook --step mycode.yaml #for step by step execution

1. The playbook itself defines a desired state. Ansible is all about **desired state**. You define how you want the managed machine to be.
2. Ansible playbooks are idempotent. This means that playbooks won't change anything on a managed host that already is in a desired state.
3. Indentation is important since ansible modules written in python, indentation means it is sensitive for spaces and orientation.

#Example1 vsftpd and copy a file

[ansible@controller ~]$ mkdir playbook1

[ansible@controller ~]$ cd playbook1/

[ansible@controller playbook1]$ cp /home/ansible/lab1/inventory .

[ansible@controller playbook1]$ vim vsftpd.yaml

```
---
#Install vsftpd service on managed1
- name: vsftpd playbook
  hosts: managed1.mekawy.com
  user: ansible
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: install vsftpd
    package:              #"package" module is more generic than using "yum" module for centos or "apt" module for ubuntu
      name: vsftpd
      state: latest
  - name: start vsftpd
    service: name=vsftpd state=started
  - name: enable vsftpd
    service: name=vsftpd enabled=true
  - name: create README file
    copy:
      content: "Welcome to my ftp server\n"
      dest: /var/ftp/pub/README
      force: no
      mode: 0444
...
```

[ansible@controller playbook1]$ ansible-playbook --syntax-check vsftpd.yaml -i inventory

[ansible@controller playbook1]$ ansible-playbook vsftpd.yaml -i inventory

PLAY [vsftpd playbook]
****
****

TASK [Gathering Facts]
****
****

ok: [managed1.mekawy.com]

TASK [install vsftpd]
****
****

ok: [managed1.mekawy.com]

TASK [start vsftpd]
****
****

changed: [managed1.mekawy.com]

TASK [enable vsftpd]
****
****

changed: [managed1.mekawy.com]

TASK [create README file]
****
****

changed: [managed1.mekawy.com]

PLAY RECAP
****
****

managed1.mekawy.com : ok=5 changed=3 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

[ansible@controller playbook1]$ ansible-playbook vsftpd.yaml -i inventory

PLAY [vsftpd playbook]
****
****

TASK [Gathering Facts]
****
****

ok: [managed1.mekawy.com]

TASK [install vsftpd]
****
****

ok: [managed1.mekawy.com]

TASK [start vsftpd]
****
****

ok: [managed1.mekawy.com]

TASK [enable vsftpd]
****
****

ok: [managed1.mekawy.com]

TASK [create README file]
****
****

ok: [managed1.mekawy.com]

PLAY RECAP
****
****

managed1.mekawy.com : ok=5 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

[ansible@controller playbook1]$ ansible-playbook -C vsftpd.yaml -i inventory #Dry Run

[ansible@controller playbook1]$ ansible-playbook --step vsftpd.yaml -i inventory #Step execution

Example2 httpd and firewalld

[ansible@controller ~]$ mkdir playbook2

[ansible@controller ~]$ cd playbook2/

[ansible@controller playbook2]$ cp /home/ansible/playbook1/inventory .

[ansible@controller playbook2]$ vim httpd.yaml

```
---
#Install httpd service on managed2
- name: httpd playbook
  hosts: managed2.mekawy.com
  user: ansible
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: install httpd
    yum: name=httpd
  - name: start and enable httpd
    service: name=httpd.service state=started enabled=true
  - name: create index.html with text
    copy:
      content: "Welcome to my web server"
      dest: /var/www/html/index.html
      force: no
      mode: 0444
  - name: open a firewall port
    firewalld:
      service: http
      permanent: true
      state: enabled
      immediate: yes
...
```

[ansible@controller playbook2]$ ansible-playbook --syntax-check httpd.yaml -i inventory

[ansible@controller playbook2]$ ansible-playbook -C httpd.yaml -i inventory

[ansible@controller playbook2]$ ansible-playbook httpd.yaml -i inventory

PLAY [httpd playbook]
****
****

```
TASK [Gathering Facts]
*********************************************************************************************************

ok: [managed2.mekawy.com]

TASK [install httpd]
*********************************************************************************************************

ok: [managed2.mekawy.com]

TASK [start and enable httpd]
*********************************************************************************************************

ok: [managed2.mekawy.com]

TASK [create index.html with text]
*********************************************************************************************************

ok: [managed2.mekawy.com]

TASK [open a firewall port]
*********************************************************************************************************

changed: [managed2.mekawy.com]

PLAY RECAP
*********************************************************************************************************

managed2.mekawy.com      : ok=5    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

Example2.1 httpd and firewalld and verify

[ansible@controller playbook2]$ vim httpd2-verify.yaml

```
---
#Install httpd service on managed2
- name: httpd playbook
  hosts: managed2.mekawy.com
  user: ansible
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: install httpd
    yum: name=httpd
  - name: start and enable httpd
    service: name=httpd.service state=started enabled=true
  - name: create index.html with text
    copy:
      content: "Welcome to my web server"
      dest: /var/www/html/index.html
      force: no
      mode: 0444
  - name: open a firewall port
    firewalld:
      service: http
      permanent: true
      state: enabled
      immediate: yes

- name: verify the web server
  hosts: localhost   #this is because will verify form the controller host
  become: false      #this is because it doesn't have to do it with root privileges.
  tasks:
    - name: test the web server is available
      uri:            # using the uri module to verify
        url: http://managed2.mekawy.com
        status_code: 200   #We test that the web server is available, and if the URL http at managed2.mekawy.com is returning a status code 200, then we are okay
...
```

Example3 Prepare Openstack environment

Requirements:

1. NetworkManager disabled and stopped
2. On /etc/default/grub/ -> add net.ifnames=0 biosdevname=0 to the line loads the linux kernel using lineinfile, then copy new files to hosts
3. On all hosts run command grub2-mkconfig to run the modified /etc/default/grub file

ansible@controller ~]$ mkdir playbook3

[ansible@controller ~]$ cd playbook3/

[ansible@controller playbook3]$ cp /home/ansible/lab1/inventory .

#get documentation info about needed modules

[ansible@controller playbook3]$ ansible-doc service

[ansible@controller playbook3]$ ansible-doc lineinfile

[ansible@controller playbook3]$ ansible-doc command

#running Ad-hoc command on target to verify content before change of /etc/default/grub through shell

[ansible@controller playbook3]$ ansible managed1.mekawy.com -i inventory -m shell -a "cat /etc/default/grub"

managed1.mekawy.com | CHANGED | rc=0 >>

GRUB_TIMEOUT=5

GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"

GRUB_DEFAULT=saved

GRUB_DISABLE_SUBMENU=true

GRUB_TERMINAL_OUTPUT="console"

GRUB_CMDLINE_LINUX="spectre_v2=retpoline rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb quiet"

GRUB_DISABLE_RECOVERY="true"

#creating playbook YAML file

[ansible@controller playbook3]$ vim posh.yaml

```
---
#preparing open stack hosts on managed1
- name: posh playbook
  hosts: managed1.mekawy.com
  user: ansible
  become: yes
  become_method: sudo
  become_user: root
  tasks:
  - name: stop and disable NetworkManager
    service: name=NetworkManager state=stopped enabled=false
  - name: edit file
    lineinfile:
      path: /etc/default/grub
      regexp: '^GRUB_CMDLINE_LINUX='
      line: 'GRUB_CMDLINE_LINUX="spectre_v2=retpoline rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb quiet net.ifnames=0 biosdevname=0"'
  - name: rewrite grub
    command: grub2-mkconfig -o /boot/grub2/grub.cfg
...
```

#check YAML file syntax

[ansible@controller playbook3]$ ansible-playbook posh.yaml --syntax-check -i inventory

#Run playbook

[ansible@controller playbook3]$ ansible-playbook posh.yaml --syntax-check -i inventory

playbook: posh.yaml

[ansible@controller playbook3]$ ansible-playbook posh.yaml -i inventory

PLAY [posh playbook]
****
****

TASK [Gathering Facts]
****
****

ok: [managed1.mekawy.com]

TASK [stop and disable NetworkManager]
****
****

changed: [managed1.mekawy.com]

TASK [edit file]
****
****

changed: [managed1.mekawy.com]

TASK [rewrite grub]
****
****

changed: [managed1.mekawy.com]

PLAY RECAP
****
****

managed1.mekawy.com : ok=4 changed=3 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

#verify after change

[ansible@controller playbook3]$ ansible managed1.mekawy.com -i inventory -m shell -a "cat /etc/default/grub"

managed1.mekawy.com | CHANGED | rc=0 >>

GRUB_TIMEOUT=5

GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"

GRUB_DEFAULT=saved

GRUB_DISABLE_SUBMENU=true

GRUB_TERMINAL_OUTPUT="console"

GRUB_CMDLINE_LINUX="spectre_v2=retpoline rd.lvm.lv=centos/root rd.lvm.lv=centos/swap rhgb quiet net.ifnames=0 biosdevname=0"

GRUB_DISABLE_RECOVERY="true"

Ansible Variables

Using variables can make it easier to repeat tasks in a playbook.

-Variable Naming: variable name you can have letters, you can have numbers, you can have underscores

-Variables Levels and Scope:

1. Global scope: is a variable that is set from the command line or from the Ansible configuration file.
2. Play scope: then the variable relates to the play and the related structures. That is what you typically do while defining variables in playbooks.
3. Host scope: which means that you set it to a group of hosts or to individual hosts. This is something you can do through the inventory file or included from external variable YAML files.

Using variables on playbook

If you want to use the variable within the playbook then the variable is referred to by using double curly braces.

If the variable is used as the first element to start value you must use double quotes

Direct declaration [playbook]

```
#in playbook
- hosts: all
  vars:
    user: mekawy
    home: /home/mekawy
```

Declaration using separate YAML file [playbook]

Create vars/ directory in the same playbook directory, then create users.yaml file

```
#in playbook
- hosts: all
  vars_files:
    - vars/users.yaml

cat vars/users.yaml
    user: mekawy
    home: /home/mekawy
    user: mohamed
    home: /home/mohamed
```

Declaration using separate YAML file [inventory file]

Types:

1. host variable: is a variable that applies to one host that is defined in the inventory file,
2. group variable: applies to multiple hosts that are defined in a group in the inventory file.

Direct declaration in inventory file

```
#in inventory file
[webservers]
manged2.mekawy.com
[webservers:vars]
user=mekawy
home: /home/mekawy
```

The recommended method is to use group_vars and host_vars directories in the same playbook directory.

```
group_vars/webservers
cat group_vars/webservers
    package: httpd

host_vars/managed2.mekawy.com
cat host_vars/managed2.mekawy.com
    user: mekawy
```

Notice Variables can be included from external YAML or JSON files using the "include_vars" directive. <<<<<<Using this method overrides any other method to define variables.>>>>>>

You can also override variables, you can do that at any time by using the -e key=value command line option,

where key is the name of the variable, and value is the value that you want to be using for that specific variable.

#example

ansible-playbook -e "key=value"

Variables Precedence

1. Variables defined with "include_vars"
2. Variables with a global scope i.e set from ansible command line -e option or ansible configuration file
3. Variables defined by the playbook
4. variables defined at the host level

Ansible Array

An array is a variable that defines multiple values, including their specific properties.

```
users:
  mohamed:
    first_name: mohamed
    last_name: mekawy
    home_dir: /home/mohamed
  marawan:
    first_name: marawan
    last_name: mohamed
    home_dir: /home/marawan
```

Refer to these using users.mohamed.first_name

Ansible Facts

A fact contains discovered information about a host, and facts can be used in "conditional statements" to make sure that tasks only happen if they are really necessary.

#To use a fact, you can use a setup module

ansible managed1.mekawy.com -i inventory -m setup

#To limit, use a filter by passing the -a 'filter=...' option

ansible managed1.mekawy.com -i inventory -m setup -a 'filter=ansible_kernel'

Custom Facts

Custom facts can be created by administrators to display information about a host

If you want to use them, you can put them on the managed host in a file which is in the INI or JSON format and has the .fact extension, and stored it in /etc/ansible/facts.d . Custom facts will always be displayed as "ansible_local" fact.

Can be used to label managed servers

#on managed1 server

[root@managed1 ~]# cd /etc/

[root@managed1 etc]# mkdir ansible

[root@managed1 etc]# cd ansible/

[root@managed1 ansible]# mkdir facts.d

[root@managed1 ansible]# cd facts.d/

[root@managed1 facts.d]# vim server_info.fact

[server_info]

profile = ftpserver

#on controller server

[ansible@controller playbook3]$ ansible managed1.mekawy.com -i inventory -m setup -a 'filter=ansible_local'


```
managed1.mekawy.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "server_info": {
                "server_info": {
                    "profile": "ftpserver"
                }
            }
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
```


Ansible Inclusions

Tasks can be included in a playbook from an external YML file using the "include" directive.

Using task inclusion makes sense in complex setups because it allows for creation of separate files for different tasks, which can be managed independently.

You change the fixed part of your configuration from the dynamic part of your configuration.

If you are going to use task inclusion the main variables would be set in the master Ansible file, whereas generic tasks will be defined in the included files.

Examples on "include" & "include_vars"

[ansible@controller ~]$ mkdir inclusions

[ansible@controller ~]$ cp lab1/inventory inclusions/

[ansible@controller ~]$ cd inclusions/

#Example1 on "include"

[ansible@controller inclusions]$ vim setup.yml

```
---
#Include tasks from YAML on all group
- name: Deploy Services
  hosts: all
  tasks:
  - name: Read the tasks.yml file to find what to do
    include: tasks.yml
    vars:
      package: samba
      service: smb
      state: started
    register: output
  - name: debug the included tasks
    debug:
      var: output
...
```

[ansible@controller inclusions]$ vim tasks.yml

```
---
- name: install the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: start the {{ service }} package
  service:
    name: "{{ service }}"
    state: "{{ state }}"
...
```

[ansible@controller inclusions]$ ansible-playbook setup.yml --syntax-check -i inventory

[ansible@controller inclusions]$ ansible-playbook setup.yml -i inventory

PLAY [Deploy Services]
****
****

TASK [Gathering Facts]
****
****

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

ok: [controller.mekawy.com]

TASK [install the samba package]
****
****

changed: [managed1.mekawy.com]

changed: [controller.mekawy.com]

changed: [managed2.mekawy.com]

```
TASK [start the smb package]
*********************************************************************************
*****************************************

changed: [managed2.mekawy.com]

changed: [managed1.mekawy.com]

changed: [controller.mekawy.com]

TASK [debug the included tasks]
*********************************************************************************
*****************************************

ok: [controller.mekawy.com] => {

    "output": "VARIABLE IS NOT DEFINED!"

}

ok: [managed1.mekawy.com] => {

    "output": "VARIABLE IS NOT DEFINED!"

}

ok: [managed2.mekawy.com] => {

    "output": "VARIABLE IS NOT DEFINED!"

}

PLAY RECAP
*********************************************************************************
*************************************************************

controller.mekawy.com      : ok=4    changed=2    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

managed1.mekawy.com : ok=4 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed2.mekawy.com : ok=4 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

#Example 2 on "include_vars"

[ansible@controller inclusions]$ vim packages.yml

```
---
packages:
  my_pkg: httpd
...
[ansible@controller inclusions]$ vim installpkg.yml
---
- name: install some packages
  hosts: all
  tasks:
    - name: include packages
      include_vars: packages.yml
    - name: install {{ packages.my_pkg }}
      yum:
        name: "{{ packages.my_pkg }}"
        state: latest
...
```

[ansible@controller inclusions]$ ansible-playbook installpkg.yml --syntax-check -i inventory

[ansible@controller inclusions]$ ansible-playbook installpkg.yml -i inventory

PLAY [install some packages]
****
****

TASK [Gathering Facts]
****
****

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

ok: [controller.mekawy.com]

TASK [include packages]
****
****

ok: [controller.mekawy.com]

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

TASK [install httpd]
****
****

ok: [managed2.mekawy.com]

changed: [managed1.mekawy.com]

changed: [controller.mekawy.com]

PLAY RECAP
****
****

controller.mekawy.com : ok=3 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed1.mekawy.com : ok=3 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed2.mekawy.com : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

#**Notice** will not install vsftpd as include_vars has a higher priority than global variable defined in command line

```
[ansible@controller inclusions]$ ansible-playbook installpkg.yml -i inventory -e
"packages.my_pkg=vsftpd"

PLAY [install some packages]
*****************************************************************************
****************************************

TASK [Gathering Facts]
*****************************************************************************
************************************************

ok: [managed2.mekawy.com]

ok: [managed1.mekawy.com]

ok: [controller.mekawy.com]

TASK [include packages]
*****************************************************************************
************************************************

ok: [controller.mekawy.com]

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

TASK [install httpd]
*****************************************************************************
************************************************

ok: [managed2.mekawy.com]

ok: [managed1.mekawy.com]

ok: [controller.mekawy.com]
```

PLAY RECAP
****
****

controller.mekawy.com : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed1.mekawy.com : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed2.mekawy.com : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

BIG LAB

Requirements:

1-Create an Ansible configuration that sets up hosts Ansible one and Ansible two.

For automatic installation create custom facts for both hosts and use variable inclusion to realize this.

To configure Ansible one use the host group with the name file. To configure Ansible two use a host group with the name lamp.

2-Create a file with the name "custom.fact" that defines custom facts and in this file define two sections. [package] and [service]

The section with the name package contains the following

[packages]

smb_package = samba

ftp_package = vsftpd

db_package = mariadb-server

web_package = httpd

The section with the name service contains the following

[services]

smb_service = smb

ftp_service = vsftpd

db_service = mariadb

web_service = httpd

3- Create a playbook with the name "copy_facts.yml" that copies these facts to all managed hosts.

Define a variable with the name "remote_dir" and a variable with the name "facts_file" and use these.

Use the "file" and "copy" modules in order to do so.

4-run the playbook "copy_facts.yml" and verify that it worked using ansible Ad-hoc command.

5-create a variable inclusion file with the name ./vars/allvars.yml and set the following variables. You need to set them to the appropriate root directories of these services.

web_root: /var/www/html

ftp_root: /var/ftp

6-Create a task directory in the project folder.

In this directory create two YAML files [one that installs, starts, and enables the LAMP services called lamp.yml] & [one that installs, starts and enables the file services called file.yml]

7-Create the main playbook mainplaybook.yml that will set up the lamp servers and the file servers with the packages they need using inclusions to the previously-defined tasks "include".

Ensure that it opens the firewalld firewall to allow access to these servers.

Finally the web service should be provided with an "index.html" file hat shows "managed by Ansible" on the first line.

8-Run the playbook and use ad hoc commands to verify that the services have been started.

Solution:

[ansible@controller ~]$ mkdir biglab

[ansible@controller ~]$ cd biglab/

[ansible@controller biglab]$ cp /home/ansible/lab1/inventory .

[ansible@controller biglab]$ ls

inventory

[ansible@controller biglab]$ vim inventory

[all]

controller.mekawy.com

managed1.mekawy.com

managed2.mekawy.com

[lamp]

managed2.mekawy.com

[file]

managed1.mekawy.com

[ansible@controller biglab]$ vim custom.fact

[packages]

smb_package = samba

ftp_package = vsftpd

db_package = mariadb-server

web_package = httpd

[services]

smb_service = smb

ftp_service = vsftpd

db_service = mariadb

web_service = httpd

[ansible@controller biglab]$ vim copy_facts.yml

```
---
- name: install remote facts
  hosts: managed1.mekawy.com,managed2.mekawy.com
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: create remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: install new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
...
```

[ansible@controller biglab]$ ansible-playbook copy_facts.yml -i inventory

PLAY [install remote facts] **

TASK [Gathering Facts] ***

ok: [managed2.mekawy.com]

ok: [managed1.mekawy.com]

TASK [create remote directory] ***

changed: [managed2.mekawy.com]

changed: [managed1.mekawy.com]

TASK [install new facts] ***

changed: [managed1.mekawy.com]

changed: [managed2.mekawy.com]

```
PLAY RECAP
*********************************************************************

managed1.mekawy.com      : ok=3    changed=2    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

managed2.mekawy.com      : ok=3    changed=2    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

[ansible@controller biglab]$ ansible managed1.mekawy.com,managed2.mekawy.com -i
inventory -m setup -a 'filter=ansible_local'

managed1.mekawy.com | SUCCESS => {

   "ansible_facts": {

      "ansible_local": {

         "custom": {

            "packages": {

               "db_package": "mariadb-server",

               "ftp_package": "vsftpd",

               "smb_package": "samba",

               "web_package": "httpd"

            },

            "services": {

               "db_service": "mariadb",

               "ftp_service": "vsftpd",

               "smb_service": "smb",

               "web_service": "httpd"
```

```
            }
          }
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
}
managed2.mekawy.com | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "custom": {
                "packages": {
                    "db_package": "mariadb-server",
                    "ftp_package": "vsftpd",
                    "smb_package": "samba",
                    "web_package": "httpd"
                },
                "services": {
                    "db_service": "mariadb",
                    "ftp_service": "vsftpd",
                    "smb_service": "smb",
```

```
            "web_service": "httpd"
        }
    }
},
"discovered_interpreter_python": "/usr/bin/python"
},
"changed": false
}
```


[ansible@controller biglab]$ mkdir vars

[ansible@controller biglab]$ cd vars/

[ansible@controller vars]$ vim allvars.yml

```
---
web_root: /var/www/html
ftp_root: /var/ftp
...
```

[ansible@controller biglab]$ mkdir tasks

[ansible@controller biglab]$ cd tasks/

[ansible@controller tasks]$ vim file.yml

```
---
 - name: install and start the file server
   yum:
     name:
       - "{{ ansible_local.custom.packages.smb_package }}"
       - "{{ ansible_local.custom.packages.ftp_package }}"
     state: latest

 - name: start samaba service
   service:
     name: "{{ ansible_local.custom.services.smb_service }}"
     state: started
     enabled: true

 - name: start the ftp  service
   service:
     name: "{{ ansible_local.custom.services.ftp_service }}"
     state: started
     enabled: true
...
```

[ansible@controller tasks]$ cp file.yml lamp.yml

[ansible@controller tasks]$ vim lamp.yml

```
---
 - name: install and start the lamp server
   yum:
     name:
       - "{{ ansible_local.custom.packages.db_package }}"
       - "{{ ansible_local.custom.packages.web_package }}"
     state: latest

 - name: start database service
   service:
     name: "{{ ansible_local.custom.services.db_service }}"
     state: started
     enabled: true

 - name: start the web service
   service:
     name: "{{ ansible_local.custom.services.web_service }}"
     state: started
     enabled: true
...
```

[ansible@controller biglab]$ vim mainplaybook.yml

```
---
- hosts: managed1.mekawy.com,managed2.mekawy.com
  vars:
    firewall: firewalld

  tasks:
  - name: install the firewall
    yum:
      name: "{{ firewall }}"
      state: latest
  - name: Start firewall
    service:
      name: "{{ firewall }}"
      state: started
      enabled: true

- hosts: lamp
  tasks:
  - name: include the variable file
    include_vars: vars/allvars.yml

  - name: include the tasks
    include: tasks/lamp.yml

  - name: open the port for the web server
    firewalld:
      service: http
      state: enabled
      immediate: true
      permanent: true

  - name: create index.html file
    copy:
      content: "{{ ansible_fqdn }}({{ ansible_default_ipv4.address }}) Managed By Ansible\n"
      dest: "{{ web_root }}/index.html"

- hosts: file
  tasks:
  - name: include the variable file
    include_vars: vars/allvars.yml

  - name: include the tasks
    include: tasks/file.yml

  - name: open the port for the file server
    firewalld:
      service: samba
      state: enabled
      immediate: true
      permanent: true
...
```

[ansible@controller biglab]$ ls

copy_facts.yml custom.fact inventory mainplaybook.yml tasks vars

[ansible@controller biglab]$ ansible-playbook mainplaybook.yml -i inventory

```
PLAY [managed1.mekawy.com,managed2.mekawy.com]
*************************************************************************
************************

TASK [Gathering Facts]
*************************************************************************
***********************************************

ok: [managed2.mekawy.com]

ok: [managed1.mekawy.com]

TASK [install the firewall]
*************************************************************************
*****************************************

changed: [managed2.mekawy.com]

changed: [managed1.mekawy.com]

TASK [Start firewall]
*************************************************************************
************************************************

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

PLAY [lamp]
*************************************************************************
*********************************************************

TASK [Gathering Facts]
*************************************************************************
***********************************************
```

ok: [managed2.mekawy.com]

TASK [include the variable file]
****
****

ok: [managed2.mekawy.com]

TASK [install and start the lamp server]
****
****

changed: [managed2.mekawy.com]

TASK [start database service]
****
****

changed: [managed2.mekawy.com]

TASK [start the web service]
****
****

ok: [managed2.mekawy.com]

TASK [open the port for the web server]
****
****

ok: [managed2.mekawy.com]

TASK [create index.html file]
****
****

changed: [managed2.mekawy.com]

```
PLAY [file]
*********************************************************************************
*************************************************************

TASK [Gathering Facts]
*********************************************************************************
*********************************************

ok: [managed1.mekawy.com]

TASK [include the variable file]
*********************************************************************************
*************************************

ok: [managed1.mekawy.com]

TASK [install and start the file server]
*********************************************************************************
*****************************

ok: [managed1.mekawy.com]

TASK [start samaba service]
*********************************************************************************
*****************************************

changed: [managed1.mekawy.com]

TASK [start the ftp  service]
*********************************************************************************
*****************************************

ok: [managed1.mekawy.com]

TASK [open the port for the file server]
*********************************************************************************
*****************************
```

changed: [managed1.mekawy.com]

PLAY RECAP
****
****

managed1.mekawy.com : ok=9 changed=3 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed2.mekawy.com : ok=10 changed=4 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

Ansible Flow control using loops and conditionals

Flow control works with loops and conditionals to process items.

A loop is used to process a series of values in an array.

A conditional is a task that is executed only if specific conditions are met. Conditionals normally work on facts. For example, the fact "{{ minimum memory }}<128"

Loop types

1. Simple loop
2. Nested loop
3. Other loop types

Simple loop

A simple loop is just a list of items that is processed through the ""with_items" statement.

#example1

```
- yum:
    name: "{{ item }}"
    state: latest
  with_items:
    - nmap
    - net-tools
```

#example2

[ansible@controller ~]$ mkdir loops

[ansible@controller ~]$ cd loops

[ansible@controller loops]$ cp /home/ansible/lab1/inventory .

[ansible@controller loops]$ vim loop_createusers.yml

```
---
- name: create users
  hosts: all
  tasks:
    - user:
        name: "{{ item.name }}"
        state: present
        groups: "{{ item.group }}"
      with_items:
        - { name: 'marawan', groups: 'wheel' }
        - { name: 'mohamed', groups: 'root' }
...
```

Nested loop

It's a loop inside a loop. If a nested loop is used, Ansible iterates of the first array and applies the values in the second array to each item in the first array.

And this is very useful If a series of tasks needs to be executed on items in the array.that is processed through the ""with_nested" statement.

#example

[ansible@controller loops]$ vim nested_makedbusers.yml

```
---
- name: give users access to multible databases
  mysql_user:
    name: "{{ item[0] }}"   #this will make mohamed have access on all db's clientdb , employeedb , providerdb
    priv: "{{ item[1] }}"   #this will make marawan have access on all db's clientdb , employeedb , providerdb
    append_privs: yes
    password: "foo"
  with_nested:
    - [ 'mohamed', 'marawan' ]                  #array 0
    - [ 'clientdb', 'employeedb', 'providerdb' ] #array 1
...
```

Other loop types

There are some other loop types that can be used as well. For a full list you can consult the documentation.like

with_file: which evaluates a list of files

with_fileglob: which evaluated a list of files based on a globing pattern. Globing pattern means a*, as in, everything starting with an a.

With_sequence, which generates a sequence of items in increasing order.

with_random_choice which takes a random item from a list.

Ansible conditionals

Conditionals make sure that tasks only run if a host meets specific conditions, and that's also something that you can use on Ansible facts.

So in conditionals, operators can be used such as string comparison, mathematical operators or booleans.

And the conditionals can look at different items like a value of a registered variable, an Ansible fact, or even the output of a command, and different operators can be used.

To test conditionals, you can use string comparison, mathematical operators, and or booleans.

Operators

Equal on string: {{ ansible_machine }} == "x86_64"

Equal on numeric: {{ max_memory }} == 1024

less than: {{ min_memory }} < 128

grater than: {{ min_memory }} > 256

less than or equal: {{ min_memory }} <= 512

not equal: {{ min_memory }} != 512

variable exists: {{ min_memory }}is defined

variable does not exists: {{ min_memory }}is not defined

variable is set to yes, true or 1: {{ available_memory }}

variable is set to no, false or 0: not{{ available_memory }}

value is present in a variable or array: {{ users }} in users["db_admins"]

And many more in ansible documentation website

When Statement

The when statement is used to implement a condition.Notice that the when statement must be indented outside of the module at the top level of the task.So don't include it in the module, that won't work.

#example1

```
---
- hosts: all
  vars:
    startme: true
  tasks:
  - name: install samab when startme variable is true
    package:
      name: samba
    when: startme
...
```

#example2

```
---
- name: using varable value in other variables
  hosts: all
  vars:
    my_user: mekawy
    superusers:
      - root
      - mohamed
      - mekawy
      - marawan

  tasks:
  - name: run only if mekawy is a superusers list
    user:
      name: "{{ my_user }}"
      groups: wheel
      append: yes
    when: my_user in superusers
...
```

Magic Variables

Magic variables are variables that are provided automatically by ansible and cannot be used by users like,

hostvars: which allows to request variables set on other hosts, including facts

group_names: an array of all groups the host currently using

groups: which is a list of all hosts and groups in the inventory

#example3

```
-name: install mariadb-server when managed machine member of group databases
 package:
   name: mariadb-server
 when: inventory_hostname in groups["databases"]
```

Multiple Conditions

Multiple conditions can be combined with and and or keywords, or grouped in parentheses

#example1

{{ ansible_kernel == 3.10.0.514.el7.x86_64 }} and {{ ansible_distribution == CentOS }}

#example2

not {{ ansible_apparmor }} and ansible_distribution == SuSE

Example conditional 1

[ansible@controller biglab]$ cd ..

[ansible@controller ~]$ mkdir conditionals

[ansible@controller ~]$ cd conditionals/

[ansible@controller conditionals]$ cp /home/ansible/lab1/inventory .

[ansible@controller conditionals]$ ansible managed1.mekawy.com -i inventory -m setup -a 'filter=ansible_mounts'

managed1.mekawy.com | SUCCESS => {

    "ansible_facts": {

```
"ansible_mounts": [
    {
        "block_available": 224587,
        "block_size": 4096,
        "block_total": 259584,
        "block_used": 34997,
        "device": "/dev/sda1",
        "fstype": "xfs",
        "inode_available": 523962,
        "inode_total": 524288,
        "inode_used": 326,
        "mount": "/boot",
        "options": "rw,seclabel,relatime,attr2,inode64,noquota",
        "size_available": 919908352,
        "size_total": 1063256064,
        "uuid": "176f0be6-cc4c-4ecf-a972-5bce29eb70ee"
    },
    {
        "block_available": 4044099,
        "block_size": 4096,
        "block_total": 4452864,
```

```
            "block_used": 408765,
            "device": "/dev/mapper/centos-root",
            "fstype": "xfs",
            "inode_available": 8873131,
            "inode_total": 8910848,
            "inode_used": 37717,
            "mount": "/",
            "options": "rw,seclabel,relatime,attr2,inode64,noquota",
            "size_available": 16564629504,
            "size_total": 18238930944,
            "uuid": "442cc7b6-4a72-4b4b-aa36-df4935896b13"
        }
    ],
    "discovered_interpreter_python": "/usr/bin/python"
},
"changed": false
}
```


[ansible@controller conditionals]$ vim installif.yml

```
---
- hosts: managed1.mekawy.com
  tasks:
    - name: install package if sufficient diskspace
      yum:
        name: mariadb-server
        state: latest
      with_items: "{{ ansible_mounts }}"
      when: item.mount == "/" and item.size_available > 20000000000 #will not execute as "mount": "/", has only "size_available": 16564629504, whish is less than condition
...
```

```
[ansible@controller conditionals]$ ansible-playbook installif.yml -i inventory

PLAY [managed1.mekawy.com]
*****************************************************************************
*******************************************

TASK [Gathering Facts]
*****************************************************************************
************************************************

ok: [managed1.mekawy.com]

TASK [install package if sufficient diskspace]
*****************************************************************************
*************************

skipping: [managed1.mekawy.com] => (item={u'block_used': 34997, u'uuid':
u'176f0be6-cc4c-4ecf-a972-5bce29eb70ee', u'size_total': 1063256064, u'block_total':
259584, u'mount': u'/boot', u'block_available': 224587, u'size_available': 919908352,
u'fstype': u'xfs', u'inode_total': 524288, u'inode_available': 523962, u'device': u'/dev/sda1',
u'inode_used': 326, u'block_size': 4096, u'options':
u'rw,seclabel,relatime,attr2,inode64,noquota'})

skipping: [managed1.mekawy.com] => (item={u'block_used': 408813, u'uuid':
u'442cc7b6-4a72-4b4b-aa36-df4935896b13', u'size_total': 18238930944, u'block_total':
4452864, u'mount': u'/', u'block_available': 4044051, u'size_available': 16564432896,
u'fstype': u'xfs', u'inode_total': 8910848, u'inode_available': 8873131, u'device':
u'/dev/mapper/centos-root', u'inode_used': 37717, u'block_size': 4096, u'options':
u'rw,seclabel,relatime,attr2,inode64,noquota'})

PLAY RECAP
*****************************************************************************
************************************************************

managed1.mekawy.com        : ok=1    changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=0
```

[ansible@controller conditionals]$ vim installif.yml

```
---
- hosts: managed1.mekawy.com
  tasks:
    - name: install package if sufficient diskspace
      yum:
        name: mariadb-server
        state: latest
      with_items: "{{ ansible_mounts }}"
      when: item.mount == "/" and item.size_available > 10000000000 #will  execute as "mount": "/", has "size_available": 16564629504,which meet the condition
...
```

[ansible@controller conditionals]$ ansible-playbook installif.yml -i inventory

PLAY [managed1.mekawy.com]
****

TASK [Gathering Facts]
****

ok: [managed1.mekawy.com]

TASK [install package if sufficient diskspace]
****

skipping: [managed1.mekawy.com] => (item={u'block_used': 34997, u'uuid': u'176f0be6-cc4c-4ecf-a972-5bce29eb70ee', u'size_total': 1063256064, u'block_total': 259584, u'mount': u'/boot', u'block_available': 224587, u'size_available': 919908352, u'fstype': u'xfs', u'inode_total': 524288, u'inode_available': 523962, u'device': u'/dev/sda1', u'inode_used': 326, u'block_size': 4096, u'options': u'rw,seclabel,relatime,attr2,inode64,noquota'})

ok: [managed1.mekawy.com] => (item={u'block_used': 445277, u'uuid': u'442cc7b6-4a72-4b4b-aa36-df4935896b13', u'size_total': 18238930944, u'block_total': 4452864, u'mount': u'/', u'block_available': 4007587, u'size_available': 16415076352, u'fstype': u'xfs', u'inode_total': 8910848, u'inode_available': 8872674, u'device': u'/dev/mapper/centos-root', u'inode_used': 38174, u'block_size': 4096, u'options': u'rw,seclabel,relatime,attr2,inode64,noquota'})

PLAY RECAP
****
****

managed1.mekawy.com : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

Example conditional 2

```
---
- hosts: managed1.mekawy.com
  tasks:
    - name: check mariadb status
      command: /usr/bin/systemctl is-active mariadb
      ignor_errors: yes #continue running, even if mariadb is not running
                        #because this playbook is going to act upon the result of
                        #this command. it might do something even if mariadb
                        #currently is NOT active.
      register: result  #to do this, the result of the above command is storedin
                        #the result.rc variable and this result.rc is referred to
                        #in the when statment below

    - name: install httpd if mariadb is active
      yum:
        name: httpd
        state: installed
      when: result.rc == 0 #this evaluate the output of the above check task
                           # ans will only start if the exit code of the
                           # systemctl command is 0

    - name: start httpd
      service:
        name: httpd
        state: started
...
```

Example Conditional 3

Requirements:

write an Ansible playbook that runs some tasks to set up an OpenStack controller.

It should install the openstack-packstack package and if that is successful, it should run the packstack --gen-answer-file/root/answers.txt command.

Solution:

[ansible@controller conditionals]$ yum search openstack

centos-release-openstack-queens.noarch : OpenStack from the CentOS Cloud SIG repo configs

[ansible@controller conditionals]$ yum -y install centos-release-openstack-queens.noarch

[ansible@controller conditionals]$ vim setup_controller.yml

```
---
- hosts: controller.mekawy.com
  tasks:
    - name: install packstack
      yum:
        name: openstack-packstack
        state: latest
      register: result

    - name: generate answer file if packstack is installed
      command: packstack --gen-answer-file /root/answers.txt
      when: result.rc == 0
...
```

[ansible@controller conditionals]$ ansible-playbook setup_controller.yml -i inventory

PLAY [controller.mekawy.com]
****
****

TASK [Gathering Facts]
****
****

ok: [controller.mekawy.com]

TASK [install packstack]
****
****

changed: [controller.mekawy.com]

TASK [generate answer file if packstack is installed]
****
****

changed: [controller.mekawy.com]

PLAY RECAP
****
****

controller.mekawy.com : ok=3 changed=2 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

#OR you can replace it with

[ansible@controller conditionals]$ vim setup_controller_v2.yml

```
---
- hosts: controller.mekawy.com
  tasks:
    - name: install packstack
      yum:
        name:
          - centos-release-openstack-queens.noarch
          - openstack-packstack
        state: latest
      register: result

    - name: generate answer file if packstack is installed
      command: packstack --gen-answer-file /root/answers.txt
      when: result.rc == 0
...
```

[ansible@controller conditionals]$ ansible-playbook setup_controller_v2.yml -i inventory

```
PLAY [controller.mekawy.com]
*****************************************************************************
*****************************************

TASK [Gathering Facts]
*****************************************************************************
************************************************

ok: [controller.mekawy.com]

TASK [install packstack]
*****************************************************************************
***********************************************

ok: [controller.mekawy.com]

TASK [generate answer file if packstack is installed]
*****************************************************************************
******************

changed: [controller.mekawy.com]

PLAY RECAP
*****************************************************************************
************************************************************

controller.mekawy.com      : ok=3    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0

[root@controller ~]# vim /root/answers.txt
```

Combining loops and Conditions

Ansible facts may present a dictionary with multiple values and in that case, you can iterate through each value until a specific condition is met.

#example

```
---
- name: Combining loops and Conditions
  hosts: all
  tasks:
  - name: install vsftpd is sufficient space on /var/ftp
    package:
      name: vsftpd
      state: latest
    with_items: "{{ ansible_mounts }}"
    when: item.mount == "/var/ftp" and item.size.available > 10000000000
...
```

Jija2 Templates

what's Jinja2 templates?

It's a Python-based template that is used to put host specific data on hosts, using generic YAML and Jinja2 files.

Jinja2 templates are used to modify YAML files before they are sent to the managed host.Jinja2 can also be used to reference variables in playbooks.

And as an advanced usage, Jinja2 loops and conditionals can be used in templates to generate very specific code. The host specific data is generated through variables or facts.

Example Jinja2

[ansible@controller ~]$ sudo vim /root/motd.j2

This is the system {{ ansible_hostname }}.

Today is {{ ansible_date_time.date }}.

Only use this system if {{ system_owner }} has granted you permission.

[ansible@controller ~]$ mkdir jinja2

[ansible@controller ~]$ cd jinja2/

[ansible@controller jinja2]$ cp /home/ansible/lab1/inventory .

[ansible@controller jinja2]$ sudo cp /root/motd.j2 .

[ansible@controller jinja2]$ vim motd.yml

```
---
- hosts: all
  vars:
    system_owner: root@mekawy.com
  tasks:
    - template:      #the template module is basically taking care of substituting the code that is in the jinja2 template and that means that basically what it's all about is in the motd.j2 file.
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
...
```

[ansible@controller jinja2]$ ansible-playbook motd.yml -i inventory

PLAY [all] **********

TASK [Gathering Facts] **********

ok: [managed1.mekawy.com]

ok: [managed2.mekawy.com]

ok: [controller.mekawy.com]

TASK [template]
****

changed: [managed2.mekawy.com]

changed: [managed1.mekawy.com]

changed: [controller.mekawy.com]

PLAY RECAP
****

controller.mekawy.com : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed1.mekawy.com : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

managed2.mekawy.com : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0

[ansible@controller jinja2]$ ansible all -i inventory -m command -a "cat /etc/motd"

managed1.mekawy.com | CHANGED | rc=0 >>

This is the system managed1.

Today is 2021-05-06.

Only use this system if root@mekawy.com has granted you permission.

managed2.mekawy.com | CHANGED | rc=0 >>

This is the system managed2.

Today is 2021-05-06.

Only use this system if root@mekawy.com has granted you permission.

controller.mekawy.com | CHANGED | rc=0 >>

This is the system controller.

Today is 2021-05-06.

Only use this system if root@mekawy.com has granted you permission.

Final Lab

Requirements:

use the environment that you've created in the BIG LAB to assist in this lab.

Modify the environment in such a way that SSL configuration is pushed to the server that is a member of the LAMP group and consider the following requirements.

1)

-To provision this environment use the Control host "controller.mekawy.com" as the web server that provides some files.

-In the web server document root on this server, create a file with the name "https.conf" and transfer it to the etc/httpd/conf.d directory of the managed host/s "managed2.mekawy.com"

-provide SSL keys. You can easily generate these keys using the "genkey"" utility on the control host.

-Run "genkey managed2.mekawy.com" and follow all default prompts to create the keys, after which you can put them in the Apache document root on the control host as well.

-Next you can compress these files in a tar file and provide the tar file

2)Use the appropriate Ansible module to extract the tar file

3)create the https.conf file from the previous step as a template. Use facts so that it can dynamically determine the hostname of the target managed host.

4)ensure that the SSL configuration is only deployed on machines that have at least 512 megabytes of RAM.

5)Use a master playbook.yml file from which includes are done to a few secondary files.

-the "packages.yml" file uses loop iteration to install a list of items that are provided. And these items must be the "web package httpd" and the "SSL package mod_ssl".

Provide these items through variables, which makes it easier to use between different Linux distributions.

-create an "include" file that fetches the files from http://controller.mekawy.com and put them in the appropriate directories.

Here also, use variables to refer to these directories as it makes it easier to meet with differences between Linux distributions.

-finally, create a firewall configuration file that is included and opens the firewalld firewall for the services that have been enabled

#Sample https.conf template

listen 192.168.1.9:443

NameVirtualHost managed2.mekawy.com

<VirtualHost managed2.mekawy.com:443>

ServerName managed2.mekawy.com

DocumentRoot /var/www/html

SSLEngine on

SSLCertificateFile /etc/httpd/conf.d/ssl/server.crt

SSLCertificateKeyFile /etc/httpd/conf.d/ssl/server.key

</VirtualHost>

Solution

[ansible@controller ~]$ mkdir biglab2

[ansible@controller ~]$ cd biglab2/

[ansible@controller biglab2]$ cp /home/ansible/biglab/inventory .

[ansible@controller biglab2]$ vim https.conf

listen {{ ansible_all_ipv4_addresses }}:443

NameVirtualHost {{ ansible_fqdn }}

[ansible@controller biglab2]$ vim https.js

<VirtualHost {{ ansible_fqdn }}:443>

ServerName {{ ansible_fqdn }}

DocumentRoot /var/www/html

SSLEngine on

SSLCertificateFile /etc/httpd/conf.d/ssl/server.crt

SSLCertificateKeyFile /etc/httpd/conf.d/ssl/server.key

</VirtualHost>

[ansible@controller biglab2]$ vim configure_firewall.yml

```
---
- yum:
    name: "{{ fw_package }}"
    state: latest

- service:
    name: "{{ fw_service }}"
    state: started

- firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  with_items:
    - http
    - https
...
```

[ansible@controller biglab2]$ vim configure_web.yml

```
---
- shell:
    rpm -q httpd
  register: rpm_check
  failed_when: rpm_check.rc == 1

- block:
  - get_url:
    url: "{{ https_uri }}"
    dest: /etc/httpd/conf.d/

  - file:
      path: /etc/httpd/conf.d/ssl
      state: directory
      mode: 0755

  - template:
      src: https.j2
      dest: /etc/httpd/conf.d/ssl.conf
      owner: root
      group: root
      mode: 0755

  - unarchive:
      src: "{{ ssl_uri }}"
      dest: /etc/httpd/conf.d/ssl/
      copy: no
    notify:
      - restart_services

  - copy:
      content: "{{ ansible_fqdn }} ({{ ansible_default_ipv4.address }}) has been customized by Ansible\n"
      dest: /var/www/html/index.html
  when:
    rpm_check.rc == 0
...
```

[ansible@controller biglab2]$ vim install_packages.yml

```
---
- name: install packages
  yum:
    name: "{{ item }}"
  with_items:
    - "{{ web_package }}"
    - "{{ ssl_package }}"
  when:
    - inventory_hostname in groups["lamp"]
    - "{{ ansible_memory_mb.real.total }} >> {{ memory }}"  #variable redirection

- name: start service
  service:
    name: "{{ web_service }}"
    state: started
    enabled: true
...
```

[ansible@controller biglab2]$ vim mainplaybook.yml

```
---
- hosts: lamp
  tasks:
    - block:
        - include: install_packages.yml
          vars:
            memory: 512
            web_package: httpd
            ssl_package: mod_ssl
            web_service: httpd

        - include: configure_web.yml
          vars:
            https_uri: http//controller.mekawy,com/biglab2//https.conf
            ssl_uri: http//controller.mekawy,com/biglab2/ssl.tar.gz

        - include: configure_firewall.yml
          vars:
            fw_package: firewalld
            fw_service: firewalld

          tags: production

      - name: restart_web
          service:
            name: "{{ web_service }}"
            state: restarted
...
```

Thanks

References

1. https://docs.ansible.com/
2. Ansible Fundamentals by Sander Van Vugt Course
3. https://github.com/ansible/ansible/tree/devel/examples
4. https://github.com/ansible/ansible/