Table of Contents

Working with playbooks

- Ad hoc commands can run a single, simple task against a set of targeted hosts as a one-time command.
- The real power of Ansible, however, is in learning how to use playbooks to run multiple, complex tasks against a set of targeted hosts in an easily repeatable manner.
- A play is an ordered set of tasks which should be run against hosts selected from your `inventory`.
- A playbook is a text file that contains a list of one or more plays to run in order.

> In simple words :
> *`Playbook` **contains** `Plays`*
> *`Plays` **contains** `tasks`*
> *`Task` **runs** `modules`**.***

Plays allow you to change a lengthy, complex set of manual administrative tasks into an easily repeatable routine with predictable and successful outcomes. In a playbook, you can save the sequence of tasks in a play into a human-readable and immediately runnable form.

--

Executing Ansible Playbooks

- Ansible Playbook execution

`1a_simple_playbook.yml`

```
# --- represents this file as a playbook
# # Execute the playbook with command i.e ansible-playbook 1a_simple_playbook.yml,
debug : prints the string passed in the msg attribute.
# A gather facts task will run by default ( uses setup module )
- name: This is Simple Play inside a Playbook
  # Execute this play on all hosts mentioned in inventory file.
  hosts:  all
```

```
    # List of tasks to be executed
    tasks:
      - name: This is Debug tasks 1
        debug:
          msg: "Hello from task 1"
      - name: This is Debug tasks 2
        debug:
          msg: "Hello from task 2"
```

```
ansible-playbook 1a_simple_playbook.yml
```

--

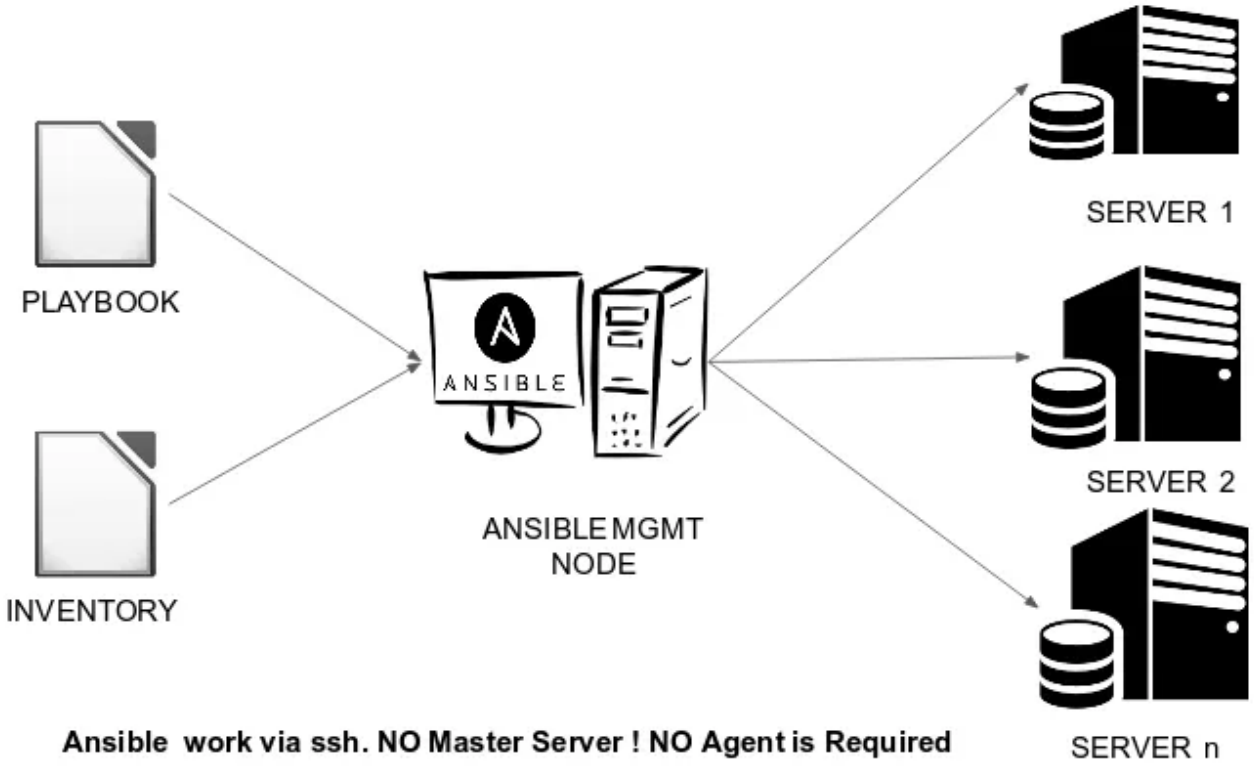
```
[ec2-user@control-node ansible-demo]$ ansible-playbook 1a_simple_playbook.yml

PLAY [This is Simple Playbook] ***************************************************************************

TASK [Gathering Facts] ***********************************************************************************
ok: [localhost]
ok: [managed-node-02.example.com]
ok: [managed-node-01.example.com]

TASK [This is Debug tasks 1] *****************************************************************************
ok: [managed-node-02.example.com] => {
    "msg": "Hello from task 1"
}
ok: [localhost] => {
    "msg": "Hello from task 1"
}
ok: [managed-node-01.example.com] => {
    "msg": "Hello from task 1"
}

TASK [This is Debug tasks 2] *****************************************************************************
ok: [managed-node-02.example.com] => {
    "msg": "Hello from task 2"
}
ok: [localhost] => {
    "msg": "Hello from task 2"
}
ok: [managed-node-01.example.com] => {
    "msg": "Hello from task 2"
}

PLAY RECAP ***********************************************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
managed-node-01.example.com : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
managed-node-02.example.com : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@control-node ansible-demo]$
```

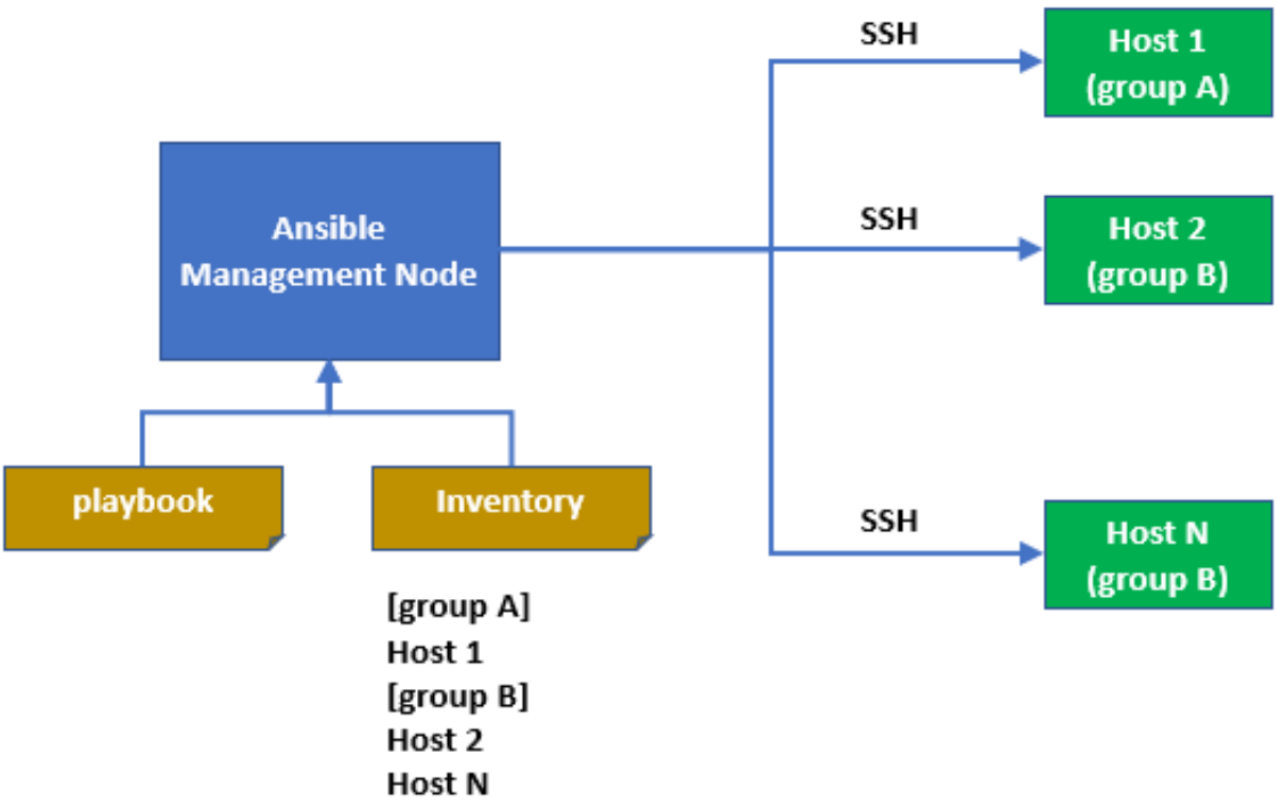
--

- Syntax Verification

```
ansible-playbook --syntax-check main.yml
```

--



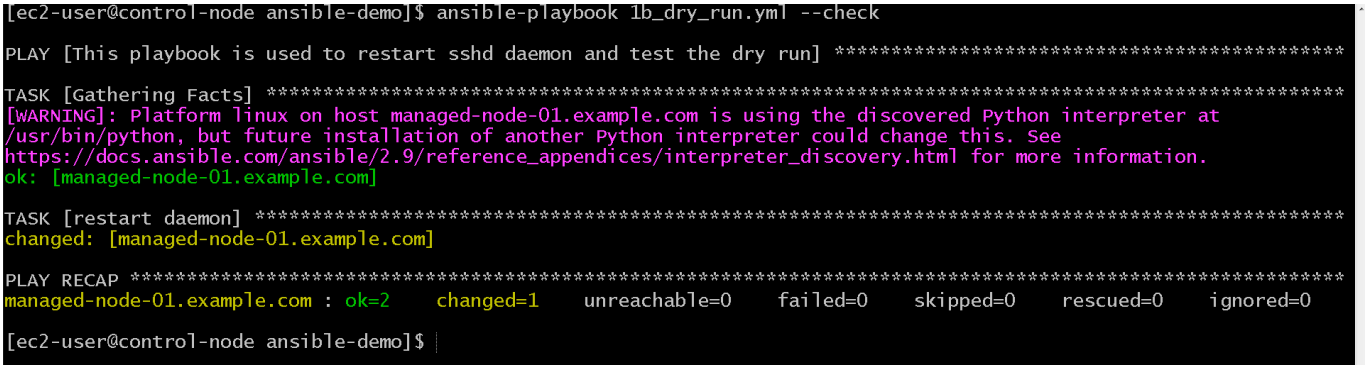


--

- Executing a Dry Run

```
ansible dev -m command -a 'service sshd status'
ansible-playbook 1b_dry_run.yml --check
```

```
ansible dev -m command -a 'service sshd status'
ansible-playbook 1b_dry_run.yml
ansible dev -m command -a 'service sshd status'
```



--

> View the .yml files and execute with `ansible-playbook <playbook.yml>`

--

Magic Variables

- **hostvars**
  - Contains the variables for managed hosts, and can be used to get the values for another managed host's variables. It won't include the managed host's facts if they haven't been gathered yet for that host.
- **group_names**
  - Lists all groups the current managed host is in.
- **groups**
  - Lists all groups and hosts in the inventory.
- **inventory_hostname**
  - Contains the hostname for the current managed host as configured in the inventory. This may be different from the hostname reported by facts for various reasons.
- Refer : https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html
- To view more magic variables that are associated with a particular hosts, use below command:

```
ansible localhost -m debug -a 'var=hostvars[inventory_hostname]'
```

Adding a new managed node with custom SSH keypair

- Launch another `centos` machine and perform below tasks.
- Change the hostname of this **centos** machine.

```
sudo hostnamectl set-hostname managed-node-03.example.com
bash
```

- Add a new entry in the /etc/hosts file on Control Node.

```
[ec2-user@control-node ~]$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost6 localhost6.localdomain6

172.31.6.127    control-node.example.com        control-node
172.31.15.76    managed-node-01.example.com     managed-node-01
172.31.9.89     managed-node-02.example.com     managed-node-02
172.31.5.137    managed-node-03.example.com     managed-node-03
```

--

- To Set password for **centos** Linux user on Centos Linux Machine i.e Managed Node.

```
sudo passwd centos
```

- change **PasswordAuthentication** property inside **/etc/ssh/sshd_config** to `yes` and restart the `sshd` service on managed nodes `sudo service sshd restart`
- To generate a custom SSH Private and Public Key Pair, On **Control Node** : Use **ssh-keygen** command, two files will be created :
    - **/home/ec2-user/.ssh/id_rsa**
    - **/home/ec2-user/.ssh/id_rsa.pub**

--

- Using **ssh-copy-id** to copy keys:
- Once a Key Pair is created using **ssh-keygen** command on control node, we can use below command on control node to copy the public keys to the managed nodes under **~/.ssh/authorized_keys** file.
- Do a **ssh-copy-id** to **centos** machine OR Append the public ssh key of **ec2-user** from control machine to centos user's **~/.ssh/authorized_keys** file.

```
# Ensure control node /etc/hosts file is having below machine entry
[ec2-user@control-node ~]$ ssh-copy-id centos@managed-node-03.example.com
------------------Command Output--------------------
/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/centos/.ssh/id_rsa.pub"
```

```
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
any that are already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now
it is to install the new keys
ec2-user@managed-node-03.example.com's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'centos@managed-node-
03.example.com'"
and check to make sure that only the key(s) you wanted were added.

------------------Command Output--------------------
ssh-copy-id centos@managed-node-03.example.com
```

- Test the connection from control node to managed node: `ssh centos@managed-node-03`
- Verify whether above commands have copied the public key on managed nodes under `~/.ssh/authorized_keys`.

--

How it works

- The `ssh-copy-id` command logs onto a server using another authentication method (normally a password).
- It then checks the permissions of the user's `.ssh` directory and appends/installs the new public key into the `~/.ssh/authorized_keys` file.
- Add below line in the `inventory` file

```
[centos]
managed-node-03.example.com     ansible_user=centos
ansible_ssh_private_key_file=/home/ec2-user/.ssh/id_rsa
```

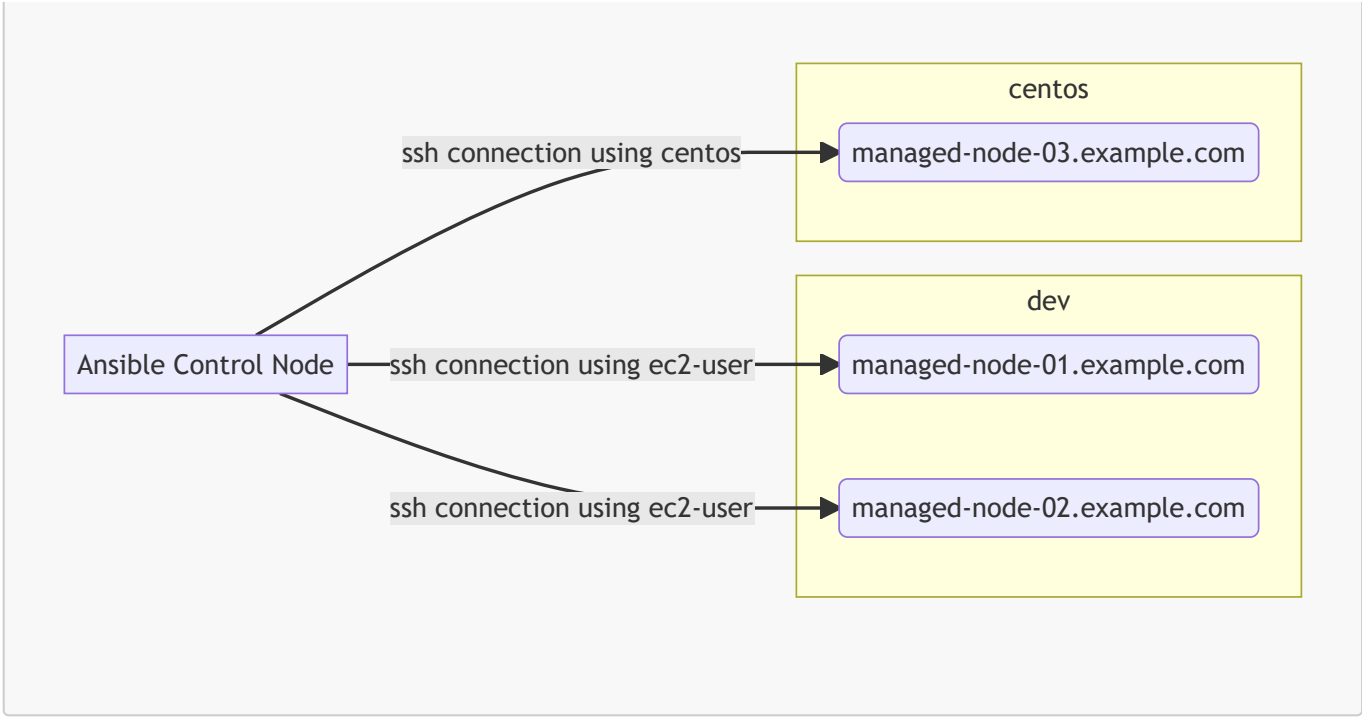
> You can define the user that Ansible uses to connect to remote devices as a variable with **ansible_user**, in a configuration file with **DEFAULT_REMOTE_USER**, as a command-line option with **-u**, and with the playbook keyword **remote_user**.

- Try **ping** module on newly launched centos node.

```
ansible centos -m ping
ansible centos -m user -a 'name=new_user state=present'
```

--

- Connection



Ansible Jinja Templates

- Ansible uses **Jinja2** templating to enable dynamic expressions and access to variables.
- A Jinja2 template is composed of multiple elements: data, variables and expressions.
- Those `variables and expressions` are replaced with their values when the Jinja2 template is rendered.
- The variables used in the template can be specified in the `vars` section of the playbook.

Where the templating happens?

- This all happens on your control machine before the task is sent and executed on the target machine.
- We don't need jinja2 packages on managed nodes.
- Only the information required by each task will be sent to remote nodes after template parsing.

--

Jinja2 Template Architecture

- Template files ends with **.j2** extension.
- A **.j2** Jinja2 template file contains:
  - **{{ }}** : Used for embedding variables which prints their actual value during code execution.

Ansible template module

- Ansible's `template` module transfers template files to remote hosts while playbook execution. It works similarly to the copy module.

- The following example shows how to create a template with variables using two of the facts retrieved by Ansible from managed hosts: `ansible_hostname` and `ansible_date_time.date`.

- When the associated playbook is executed, those two facts will be replaced by their values in the managed host being configured.

> Check jinja-test ansible playbooks