

Network Automation with Ansible

Introduction

- An open source IT automation tool
- Can automate:
 - Provisioning
 - Configuration management
 - Application deployment
 - Orchestration
- Other such manual IT processes
- Based on Python
- Can be extended by writing custom modules

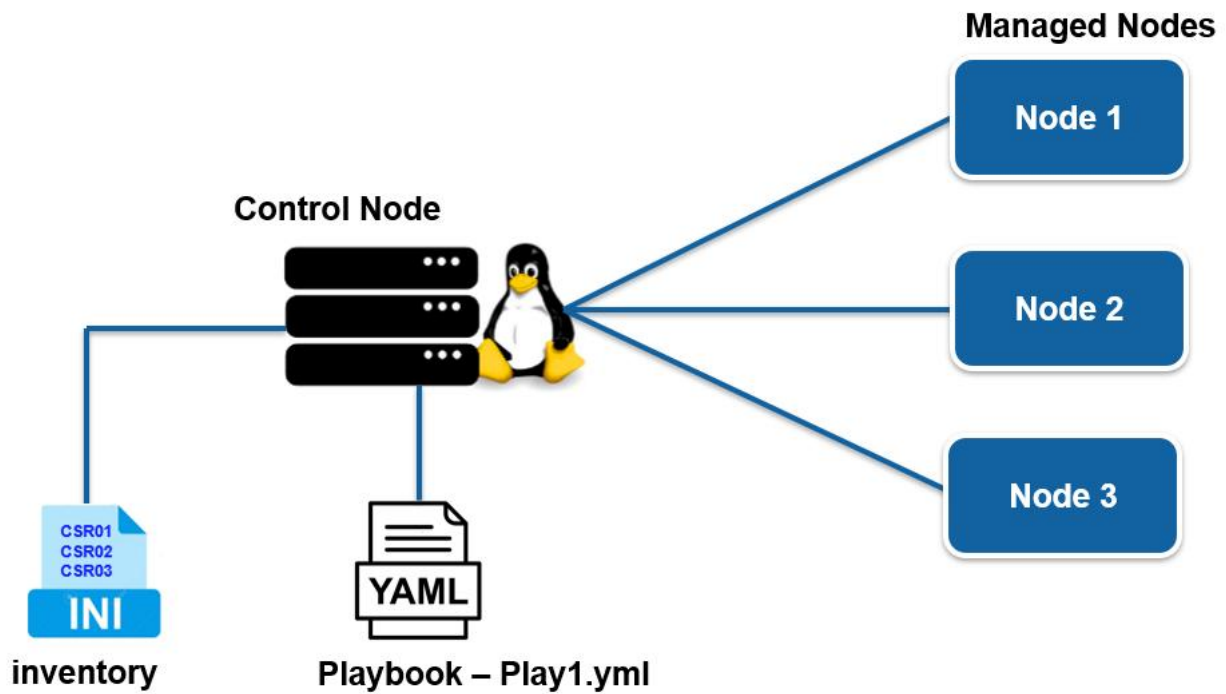
How Ansible Works?

- Written in Python and uses Python programs called Modules to carry out automation tasks
- Connects to the remote systems (servers, network devices etc.) typically via SSH
- If target machine has Python installed, it pushes small programs called modules to execute the instructions written in YAML
- If the target machine does not have python installed, it executes these modules remotely to carry out automation tasks

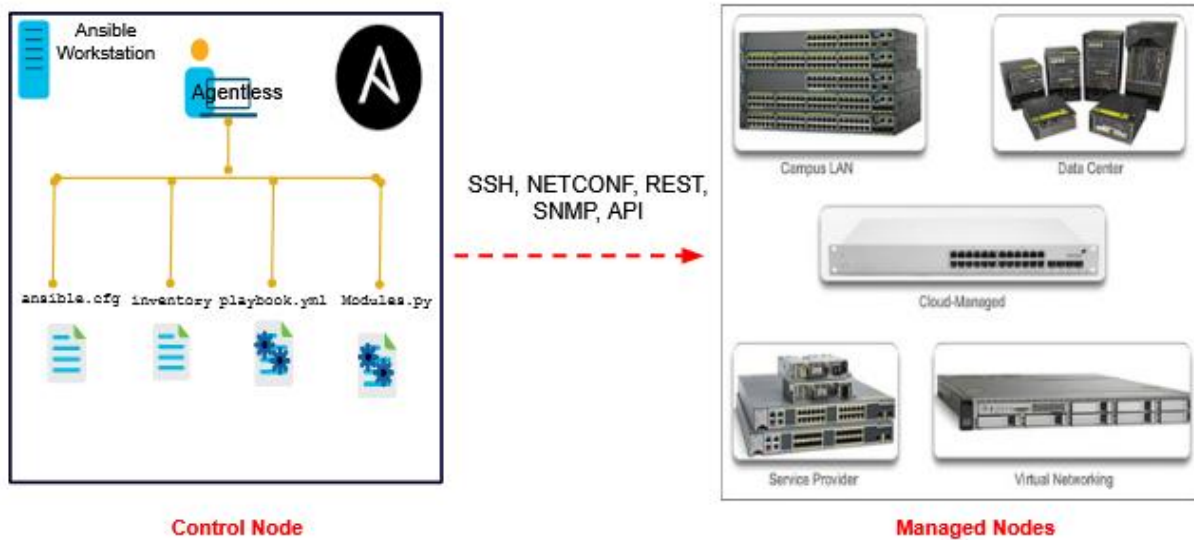
Ansible Terminology

- Control Node - The machine from which we run Ansible commands
- Managed Nodes - Also known as hosts. These are the target devices (servers, network appliances or any other machines)
- Inventory - List of managed nodes provided by some inventory source or a flat file known as host file

Ansible Architecture



Ansible with Network Devices



Ansible Installation

- <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-ubuntu-22-04>

Ansible Getting Started

- Create ansible.cfg

```
$ sudo nano ansible.cfg

# Add to this file for the Ansible lab
[defaults]
inventory = hosts
host_key_checking = False
```

- Create inventory file
- Add hosts to inventory

```
$ sudo nano hosts

csr ansible_host=192.168.72.128 ansible_user=cisco ansible_password=cisco123!
```

- Add groups to inventory

```
[cisco]
csr ansible_host=192.168.72.128 ansible_user=cisco ansible_password=cisco123!
```

- Define group vars

```
[cisco]
csr ansible_host=192.168.72.128 ansible_user=cisco ansible_password=cisco123!

[cisco:vars]
ansible_network_os=ios
ansible_connection=network_cli
ansible_user=cisco
ansible_password=cisco123!
```

- Check if inventory is properly defined

```
$ ansible all --list-hosts

hosts (1):
  csr
```

```
$ ansible-inventory --list

{
  "_meta": {
    "hostvars": {
```

```

    "csr": {
      "ansible_connection": "network_cli",
      "ansible_host": "192.168.72.128",
      "ansible_network_os": "ios",
      "ansible_password": "cisco123!",
      "ansible_user": "cisco"
    }
  },
  "all": {
    "children": [
      "cisco",
      "ungrouped"
    ]
  },
  "cisco": {
    "hosts": [
      "csr"
    ]
  }
}

```

- Inventory groups example: Grouping by device type, by vendor, by regions or data centers, defining parent child relationship in groups.

```

[cisco_routers]
192.168.110.129
csr01.example.com
csr02.example.com
csr03.example.com

[cisco_switches]
192.168.110.130

[arista_routers]
192.168.110.131

[arista_switches]
192.168.110.132

[cisco_devices:children]
cisco_routers
cisco_switches

[arista_devices:children]
arista_routers
arista_switches

```

```
[network_devices:children]
cisco_devices
arista_devices

[dc1:children]
cisco_routers
arista_switches

[dc2:children]
arista_routers
cisco_switches
```

- Run the same commands as above to see that inventory information is displayed group wise

Extra variables to be defined for network devices

Variable	Description	Example
<code>ansible_network_os</code>	This variable specifies the operating system of the network device. It determines which Ansible network modules can be used for that device.	<code>cisco.ios.ios</code> or <code>junipernetworks.junos.junos</code>
<code>ansible_connection</code>	This variable tells Ansible to use the <code>network_cli</code> connection method, which typically uses SSH to communicate with the device's command-line interface.	<code>network_cli</code>

Ad hoc Commands

- Use ad hoc commands to connect to a Cisco router and execute show commands
- Ad hoc commands are like raw commands we can give at the command line by invoking the `ansible` executable
- These are similar to the commands we give at Linux shell or Python interpreter directly
- Can be used to achieve some quick tasks or extract information

- Let us ping the device we specified in the inventory using an ad hoc command
- Remember that this is not the usual ICMP ping command
- Rather this is a python program written and being used as Ansible module
- Using the cli_command module to send a command directly to the router.

```
$ ansible csr -m cli_command -a "command='sh ip int br'"

csr | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "stdout": "Interface      IP-Address    OK? Method Status          Protocol\nGigabitEthernet1
192.168.72.128 YES DHCP    up              up",
  "stdout_lines": [
    "Interface      IP-Address    OK? Method Status          Protocol",
    "GigabitEthernet1 192.168.72.128 YES DHCP    up              up"
  ]
}
```

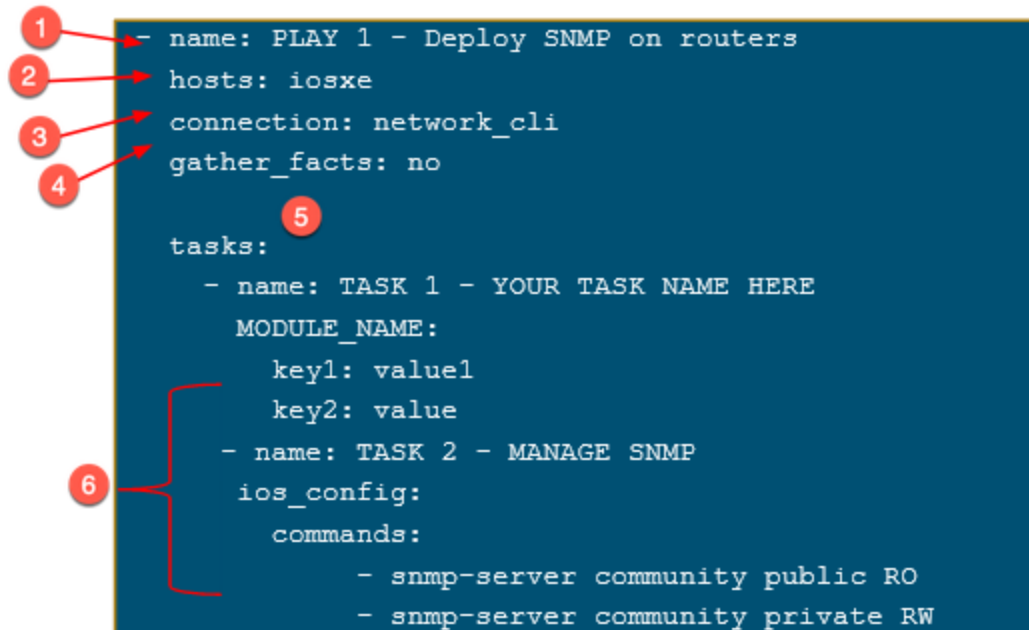
- Using ios_command module to send an adhoc command to the router

```
$ ansible csr -m ios_command -a "commands='sh ip int br, show ver'"

"stdout_lines": [
  [
    "Interface      IP-Address    OK? Method Status          Protocol",
    "GigabitEthernet1 192.168.72.128 YES DHCP    up              up"
  ],
  [
    "Cisco IOS XE Software, Version 16.09.05",
    "Cisco IOS Software [Fuji], Virtual XE Software (X86_64_LINUX_IOSD-UNIVERSALK9-M), Version 16.9.5, RELEASE SOFTWARE (fc1)",
    "Technical Support: http://www.cisco.com/techsupport",
    "Copyright (c) 1986-2020 by Cisco Systems, Inc.",
    "Compiled Thu 30-Jan-20 18:48 by mcpre",
    ""
  ]
]
```

Playbooks

- While we can work with ad hoc commands, writing scripts will let us reuse scripts as needed and also give us more flexibility
- Ansible Playbooks are written in YAML (stored as .yaml or .yml files)
- Generic structure of Ansible playbook is as follows



1. Name of the play
2. Specifying hosts
3. Connection mode (optional, can be skipped if it is mentioned in hosts file)
4. A default task that runs for every playbook to gather some useful info about the hosts. Can be turned off by setting it to no.
5. Tasks section
6. Task specification. Each task invokes an Ansible module. Depending on the nature of the task, we need to pass the appropriate arguments

Playbook to run show commands

- Playbook to run show commands

```
---
- name: playbook to run show commands
  hosts: csr03
  gather_facts: no

  tasks:
    - name: running show command using ios_command module
      ios_command:
        commands:
          - sh ip int br
      register: output

    - name: display output
      debug:
```

```
var: output
```

Playbook to run config commands

- Playbook to run config commands

```
---
- name: playbook to run config commands
  hosts: csr03
  gather_facts: no

  tasks:
    - name: running config command using ios_config module
      ios_config:
        lines:
          - description test interface
        parents: interface GigabitEthernet1
        register: output

    - name: display output
      debug:
        var: output
```

Ansible Vault

Ansible Vault is a feature that allows you to encrypt sensitive data, such as passwords, API keys, or private keys, within your Ansible playbooks or variable files. This allows you to store sensitive data in version control while keeping it secure.

Here are the basic operations for using Ansible Vault.

1. Creating Encrypted Files

You can create a new file that is encrypted from the start using the `ansible-vault create` command. You will be prompted to set a new password for the vault file.

```
ansible-vault create secret_vars.yml
```

This command opens the file in your default editor. Any content you add and save will be encrypted upon exit.

2. Viewing Encrypted Files

To view the plaintext contents of an encrypted file without editing it, use `ansible-vault view`. You will need to provide the vault password.

```
ansible-vault view secret_vars.yml --ask-vault-pass
```

3. Editing Encrypted Files

To modify the contents of an encrypted file, use `ansible-vault edit`. This temporarily decrypts the file in your editor and re-encrypts it when you save and exit.

```
ansible-vault edit secret_vars.yml --ask-vault-pass
```

4. Encrypting an Existing File

If you have a plain-text file that you want to secure, you can encrypt it using the `ansible-vault encrypt` command.

```
ansible-vault encrypt my_credentials.txt
```

5. Decrypting an Existing File

To revert an encrypted file back to plain text, use the `ansible-vault decrypt` command.

```
ansible-vault decrypt my_credentials.txt --ask-vault-pass
```

6. Changing the Vault Password

If you need to change the password used to encrypt a file (or group of files), use the `ansible-vault rekey` command. This asks for the old password and then prompts you for the new one.

```
ansible-vault rekey secret_vars.yml --ask-vault-pass
```

7. Running Playbooks with Encrypted Data

When you run a playbook that uses an encrypted file (referenced via `vars_files` or an inventory file), Ansible needs a way to access the password at runtime. The most common methods are:

A. Prompting for the password (Interactive)

Use the `--ask-vault-pass` (or `-k`) flag:

```
ansible-playbook your_playbook.yml --ask-vault-pass
```

B. Providing a Password File (Recommended for Automation)

Store your vault password in a plain-text file (e.g., in a secure, non-repository location like `~/.vault_pass.txt`) and provide the path using `--vault-password-file` or `--vault-id` :

```
ansible-playbook your_playbook.yml --vault-password-file ~/.vault_pass.txt
```