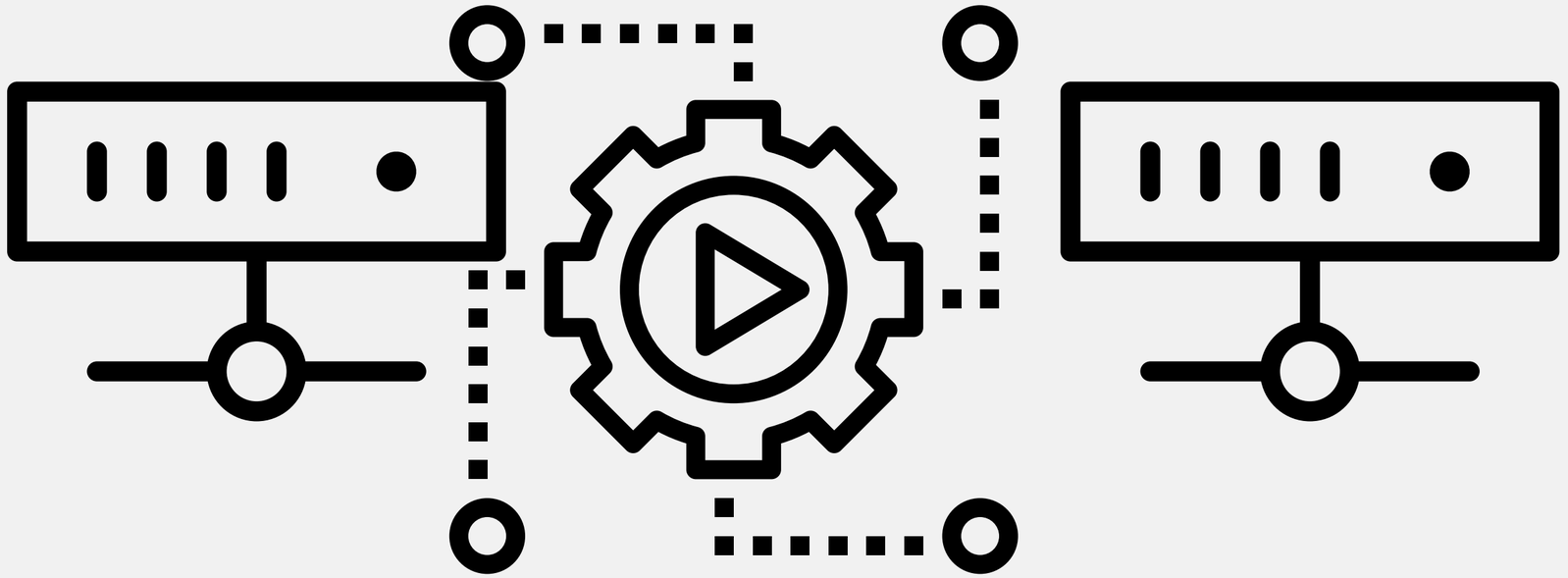
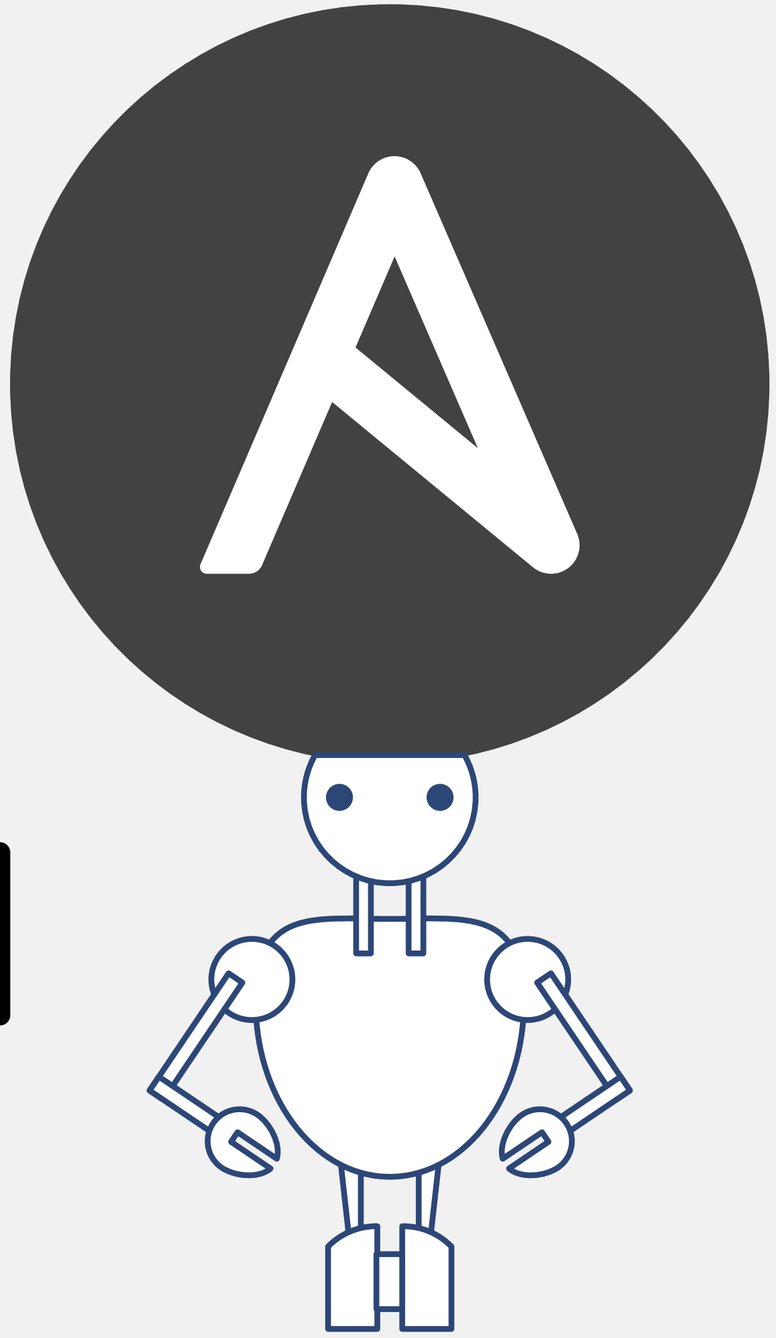
Ansible Full Course

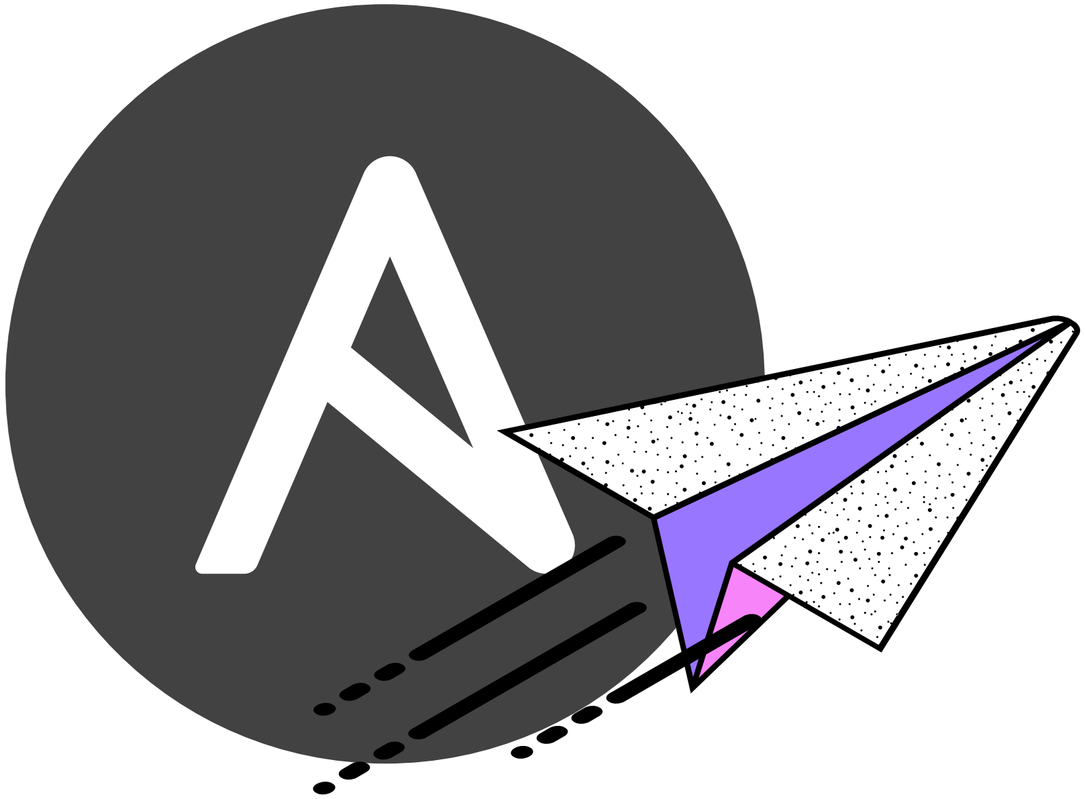
For Beginners

IT Automation Simplified





Topics



- What is Asible?
- What is Configuration Management?
- How Ansible Works?
- Ansible Concepts
- Controller Node SetUp
- Managing Managed Node Via Inventory
- Ansible Modules
- Executing Single Tasks via Ad-hoc commands
- Ansible Playbook
- Ansible Vault
- Ansible Galaxy

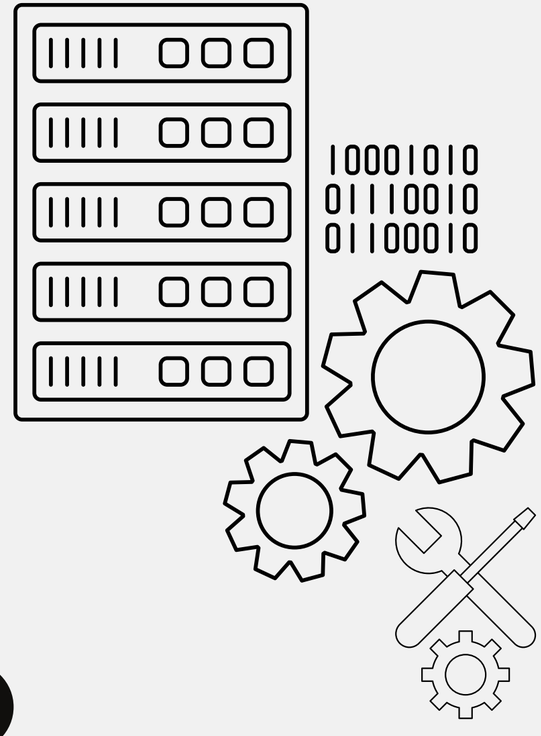
What is Ansible?

Ansible delivers simple IT automation that ends repetitive tasks and frees up DevOps teams for more strategic work.

It automates **configuration management**, cloud provisioning, application deployment, intra-service orchestration, and many other IT needs.

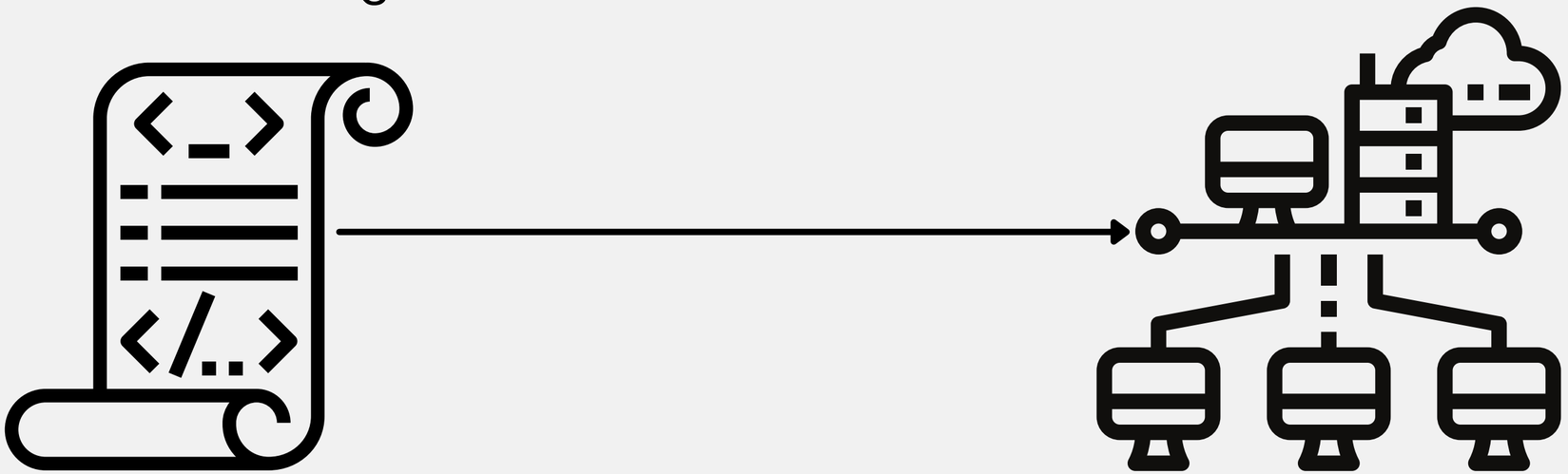
When **Ansible** is used as a **configuration management tool**, it is used to store the current state of our systems and help us to maintain that state, it make changes and deployments faster, removing the potential for human error while making system management predictable and scalable.

What is Configuration Management?



Configuration management is a process for maintaining computer systems, servers, and software in a desired, consistent state.

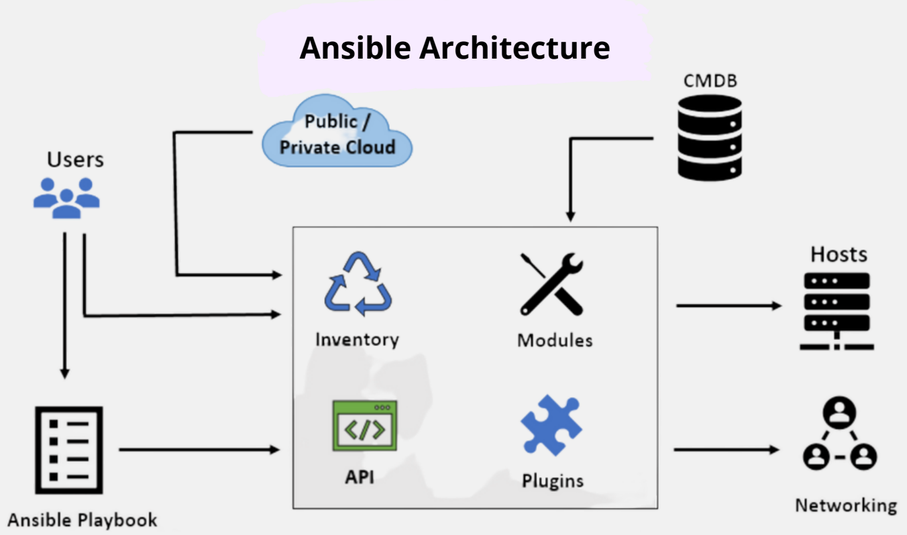
It’s a way to make sure that a system performs as it’s expected to as changes are made over time.



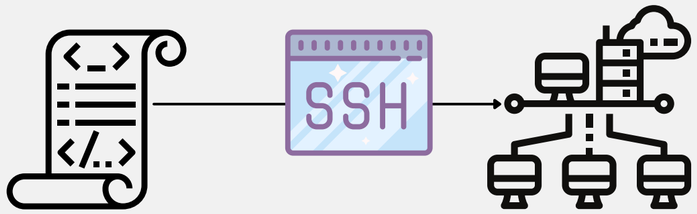
How Ansible Works?

Ansible does not use any agent. yes, you heard it right! Ansible also does not use any additional custom security infrastructure, which makes it very flexible and it can run on anything.

It manages entities/servers via SSH(Secure Shell)



Ansible works by connecting to our nodes/servers and pushing out small programs via ssh, called "Ansible Modules" to them. These programs are written to be resource models of the desired state of the system. Ansible then executes these modules (over SSH by default), and removes them when finished.



Ansible modules can be written in any language that can return JSON (Ruby, Python, bash, etc)

There's also various Python APIs for extending Ansible's connection types (SSH is not the only transport possible)

Ansible Concepts

Control Node

Any machine with Ansible installed. We can run Ansible commands and playbooks by invoking the ansible or ansible-playbook command from any control node. We can use any computer that has a Python installation as a control node - laptops, shared desktops, and servers can all run Ansible. However, We cannot use a Windows machine as a control node. We can have multiple control nodes as well.

Collections

Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. We can install and use collections through Ansible Galaxy

Tasks

The units of action in Ansible. We can execute a single task once with an ad hoc command.

Playbooks

Ordered lists of tasks, saved so we can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand

Managed nodes

The network devices (and/or servers) we manage with Ansible. Managed nodes are also sometimes called “hosts”. Ansible is not installed on managed nodes.

Inventory

A list of managed nodes. An inventory file is also sometimes called a “hostfile”. Our inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling.
typically located at /etc/ansible/hosts, provide a custom inventory path using the -i parameter when running commands & playbooks

Modules

The units of code Ansible executes. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. We can invoke a single module with a task, or invoke several different modules in a playbook.

Controller Node SetUp

A system where the Ansible is installed and configured to connect and execute commands on nodes.

Generating Custom SSH Keys

Setting up ssh:
sudo apt-get install openssh-server

Generating new ssh keys:
ssh-keygen

ssh-copy-id hostname (if it's a password-based)
ssh-copy-id -i ~/.ssh/my_custom_key user@host

Time to check SSH Connection
ssh -i ~/.ssh/my_custom_key user@host

How to Install Ansible?

There are multiple ways to install Ansible, here showing Ubuntu Example:
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo add-apt-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible

Check Ansible version

ansible –version

Testing Connectivity With Managed Nodes

Using a Custom SSH Key, checking remote connections
ansible all -m ping --private-key=~/.ssh/my_custom_key
For PLaybook:
ansible-playbook myplaybook.yml --private-key=~/.ssh/my_custom_key
Using password:
ansible all -m ping --ask-pass
ansible-playbook myplaybook.yml --ask-pass

Managing Managed Node Via Inventory

What is a managed node?

Managed node is a server (node)
controlled by Ansible Controller Node

What is Inventory?

It's a file that contains information about the servers Ansible controls, typically located at /etc/ansible/hosts , using the -i parameter we can provide custom inventory path

Targetting hosts and groups by patterns

All hosts: all (or *)
10.0.0.* : All host with IP starting from 10.0.0.*
ungrouped: all hosts that's not within any group
One host: host1
Multiple hosts: host1:host2 (or host1,host2)
One group: appservers
Multiple groups: appservers:dbservers
Excluding groups: appservers:!dbservers
The intersection of groups: appservers:&dbservers

Invetory file example with various parameters

File path: /etc/ansible/hosts (or custom location by: -i /path/to/file)

```
#un-grouped
192.0.2.40
192.0.3.56
aserver.example.org
bserver.example.org
#by group called appservers
[appservers]
sample1.example.com ansible_host = 10.0.0.3 #ssh to 10.0.0.3
sample2.example.com ansible_ssh_user = xyz #ssh as user xyz
#host (DNS will resolve automatically)
[dbservers]
one.example.com
two.example.com
three.example.com
#dev_servers1 is a group containing other groups
[dev_servers1:children]
appservers
dbservers
```

Example targeting hosts

```
ansible appservers -m ping
ansible appservers -m service -a "name=httpd state=restarted"
```

Note: **Ansible supports inventory scripts for building dynamic inventory files, this is useful when host changes very often. To know more read documentation here**
e.g. ansible all -m ping -i get_inventory.py

Ansible Modules

A module is a reusable, standalone script that Ansible runs on our behalf, either locally or remotely

Where to use modules?

Each module can be used by the Ansible API, or by the ansible or ansible-playbook programs.

A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.

Useful Modules based on use cases

- **ping** – Try to connect to host, verify a usable python and return pong on success
- **reboot** – Reboot a machine
- **get_url** – Downloads files from HTTP, HTTPS, or FTP to node
- **git** – Deploy software (or files) from git checkouts
- **copy** – Copy files to remote locations
- **file** – Manage files and file properties
- **command** – Execute commands on targets
- **shell** – Execute shell commands on targets
- **script** – Runs a local script on a remote node after transferring it
- **service** – Manage services
- **user** – Manage user accounts
- **cron** – Manage cron.d and crontab entries
- **apt** – Manages apt-packages
- **yum –** Manages packages with the yum package manager
- **add_host** – Add a host (and alternatively a group) to the ansible-playbook in-memory inventory
- **template** – Template a file out to a target host
- **include_role** – Load and execute a role
- **include_tasks** – Dynamically include a task list
- **include_vars** – Load variables from files, dynamically within a task
- **debug** – Print statements during execution

Click here to know more about Build In Modules

Executing Single Tasks via Ad-hoc commands

What is Task?

The units of action in Ansible. We can execute a single task once with an ad hoc command.

What is ad-hoc commands?

Ad-Hoc commands are an easy way to run quick commands to perform the actions, and it will not be saved for later.
It uses the /usr/bin/ansible command-line tool to automate a single task on one or more managed nodes.

Why use ad-hoc commands and use cases?

ad hoc commands are great for tasks we repeat rarely. Below are the use cases:
Syntax : Command hostgroup module/options[arguments]
Specify command : **-a** parameter | Specify Module: **-m** parameter

Rebooting servers

#reboot all servers in appservers group
ansible appservers -a "/sbin/reboot"
#reboot the appservers hosts with 10 parallel forks
ansible appservers -a "/sbin/reboot" -f 10
#to run To run /usr/bin/ansible from a differet user account (not root)
ansible appservers -a "/sbin/reboot" -f 10 -u username
#run commands through privilege escalation
ansible appservers -a "/sbin/reboot" -f 10 -u username --become [--ask-become-pass]

Ad-hoc commands example

Managing files (Copy and moving file)
#copy file
ansible appservers -m **ansible.builtin.copy** -a "**src=/etc/hosts dest=/tmp/hosts**"
#changing permissions
ansible appservers -m **ansible.builtin.file** -a "**dest**=/srv/foo/a.txt **mode=600**"
ansible appservers -m **ansible.builtin.file** -a "**dest**=/srv/foo/b.txt **mode=600 owner=sandip group=sandip**"
#create directores
ansible appservers -m **ansible.builtin.file** -a "**dest**=/path/to/c **mode**=**755 owner=sandip group=sandip state=directory**"
#Remove Directory/File
ansible appservers -m **ansible.builtin.file** -a "dest=/path/to/c **state**=**absent**"

Managing packages (Install, update and remove packages)
#using yum package manager to install and uninstall packages
ansible appservers -m **ansible.builtin.yum** -a "**name**=acme **state**=**present**"
ansible appservers -m **ansible.builtin.yum** -a "**name**=**acme-1.5 state**=**present**"
ansible appservers -m **ansible.builtin.yum** -a "**name**=acme **state**=**latest**"
ansible appservers -m **ansible.builtin.yum** -a "**name**=acme **state**=**absent**"
#using apt package manager to install and uninstall packages
ansible appservers -m apt -a "name=acme state=latest"
ansible appservers -m apt -a "name=acme-1.5 state=present"

Managing users and groups (adding , removing users and/or groups)
ansible all -m ansible.builtin.user -a "**name=foo password=<crypted password here>**"
ansible all -m ansible.builtin.user -a "**name**=foo **state=absent**"

Managing services (Start, Stop, Restart Services)
ansible appservers -m ansible.builtin.service -a "**name**=httpd **state**=**started**"
ansible appservers -m ansible.builtin.service -a "**name**=httpd **state**=**restarted**"
ansible appservers -m ansible.builtin.service -a "**name**=httpd **state**=**stopped**"

Deploying From Source Control
ansible appservers -m git -a "repo=https://foo.example.org/repo.git dest=/src/myapp version=HEAD"
Gathering facts
ansible all -m ansible.builtin.setup

Click here to know more about Build In Modules

Ansible Playbook

Ansible Playbook is ordered lists of tasks, saved so we can run those tasks in that order repeatedly. Playbooks in Ansible are written in **YAML** format ad easy to read. **YAML** means **"Yet Another Markup Language".** Every YAML file starts with ---. Playbooks usually stored in source code control e.g. git

```yaml
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest

  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - Restart apache

  - name: Ensure apache is running
    ansible.builtin.service:
      name: httpd
      state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

Ansible Playbook Components

hosts: Use hosts keyword to target hosts/servers by hostname, group name, or any pattern

Variables: The Variables are the way for Ansible to pass custom values in tasks. We can define these variables in our playbooks, in our inventory, in re-usable files or roles, or at the command line.

Ansible variable is defined in group_vars, host_vars, role vars, CLI vars and is called in Jinja Templating way: {{ my_variable }}. You can call variables everywhere in Ansible (tasks, variables, template, ...)

You can have 3 types of variables:

- **String**
- **List**
- **Dictionary**

Example:

Key-Value

Ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"

Json:

ansible-playbook release.yml --extra-vars '{"version":"1.23.45","other_variable":"foo"}'

ansible-playbook arcade.yml --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'

From File:

ansible-playbook release.yml --extra-vars "@some_file.json"

Ansible Playbook Tasks

```yaml
- name: The task name to make things clearly
  become: yes
  become_method: sudo
  become_user: root
  check_mode: yes
  diff: no
  remote_user: ansible
  ignore_errors: True
  import_tasks: more_handlers
  include_tasks: other-tasks.yml
  notify: restart apache
  register: my_register
  changed_when: False
  failed_when: False
  vars:
    - myvar: toto
    myfiles:
      -  default.conf
  loop:
    - item1
    - ["item2", "item3"]
    - { name: "item4", description: "Desc Item 4" }
  when:
    - my_register is defined
    - ansible_distribution == "CentOS"
    - ansible_distribution_major_version == "7"
  block:
    - name: Task to run in block
      ...
  rescue:
    - name: Task when block failed
  always:
    - name: Task always run before/after block
```

What is Ansible Playbook Task?

The Tasks are the actions launched on remote Hosts. Tasks are written in YAML langage in a descriptive structure way making the read and write uniform through any tasks.

We can:

- Execute tasks with elevated privileges or as a different user with become
- Repeat a task once for each item in a list with loops
- Execute tasks on a different machine with delegation
- Run tasks only when certain conditions apply with conditionals and evaluating conditions with tests
- Group a set of tasks together with blocks
- Run tasks only when something has changed with handlers

Want to learn more about tasks?

Check Official Documentation here

Ansible Playbook Handlers

```yaml
---
- name: Verify apache installation
  hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: Ensure apache is at the latest version
    ansible.builtin.yum:
      name: httpd
      state: latest

  - name: Write the apache config file
    ansible.builtin.template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - Restart apache

  - name: Ensure apache is running
    ansible.builtin.service:
      name: httpd
      state: started

  handlers:
    - name: Restart apache
      ansible.builtin.service:
        name: httpd
        state: restarted
```

What is Ansible Playbook Handlers?

Ansible Handlers are action triggers called from tasks and run at the end of a play. A Handler is a task(s) defined by its name and called with its name.

We can:

Trigger Multiple Handlers

```yaml
- name: Template configuration file
  ansible.builtin.template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - Restart memcached
    - Restart apache

  handlers:
    - name: Restart memcached
      ansible.builtin.service:
        name: memcached
        state: restarted

    - name: Restart apache
      ansible.builtin.service:
        name: apache
        state: restarted
```

Use Variables in Handlers

```yaml
tasks:
  - name: Set host variables based on distribution
    include_vars: "{{ ansible_facts.distribution }}.yml"

handlers:
  - name: Restart web service
    ansible.builtin.service:
      name: "{{ web_service_name | default('httpd') }}"
      state: restarted
```

Re-use tasks in Handlers

```yaml
# restarts.yml
- name: Restart apache
  ansible.builtin.service:
    name: apache
    state: restarted

- name: Restart mysql
  ansible.builtin.service:
    name: mysql
    state: restarted
```

```yaml
- name: Trigger an included (dynamic) handler
  hosts: localhost
  handlers:
    - name: Restart services
      include_tasks: restarts.yml
  tasks:
    - command: "true"
      notify: Restart services
```

“listen” to generic topics, and tasks can notify those topics

```yaml
handlers:
  - name: Restart memcached
    ansible.builtin.service:
      name: memcached
      state: restarted
    listen: "restart web services"

  - name: Restart apache
    ansible.builtin.service:
      name: apache
      state: restarted
    listen: "restart web services"

tasks:
  - name: Restart everything
    ansible.builtin.command: echo "this task will restart the web services"
    notify: "restart web services"
```

Ansible Playbook Roles

call a role with a fully qualified path

```
---
- hosts: webservers
  roles:
    - role: '/path/to/my/roles/common'
```

The classic (original) way to use roles is with the roles option

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

include a role

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      include_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

Pass other keywords to the roles option:

```
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
      tags: typeB
```

conditionally include a role

```
---
- hosts: webservers
  tasks:
    - name: Include the some_role role
      include_role:
        name: some_role
      when: "ansible_facts['os_family'] == 'RedHat'"
```

What is Ansible Playbook Roles?

The Roles are the tidy way to write playbooks. It permits to store a group of actions with the same purpose and to call them in playbooks in a single line.
Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After we group your content in roles, we can easily reuse them and share them with other users.

```
# playbooks
site.yml
webservers.yml
fooservers.yml
roles/
    common/
        tasks/
        handlers/
        library/
        files/
        templates/
        vars/
        defaults/
        meta/
    webservers/
        tasks/
        defaults/
        meta/
```

- **tasks/main.yml** - the main list of tasks that the role executes.
- **handlers/main.yml** - handlers, which may be used within or outside this role.
- **library/my_module.py** - modules, which may be used within this role (see Embedding modules and plugins in roles for more information).
- **defaults/main.yml** - default variables for the role. These variables have the lowest priority of any variables available and can be easily overridden by any other variable, including inventory variables.
- **vars/main.yml** - other variables for the role
- **files/main.yml** - files that the role deploys.
- **templates/main.yml** - templates that the role deploys.
- **meta/main.yml** - metadata for the role, including role dependencies.

To know more about the roles **Click Here**

Run Ansible Playbook

```yaml
- hosts: webservers
  accelerate: no
  accelerate_port: 5099
  ansible_connection: local
  any_errors_fatal: True
  become: yes
  become_method: su
  become_user: postgress
  become_flags: True
  debugger: on_failed
  gather_facts: no
  max_fail_percentage: 30
  order: sorted
  remote_user: root
  serial: 5
  strategy: debug
  vars:
    http_port: 80
  vars_files:
    - "vars.yml"
  vars_prompt:
    - name: "my_password2"
      prompt: "Enter password2"
      default: "secret"
      private: yes
      encrypt: "md5_crypt"
      confirm: yes
      salt: 1234
      salt_size: 8
  tags:
    - stuff
  pre_tasks:
    - <task>
  roles:
    - common
    - common
      vars:
        port: 5000
      when: "bar == 'Baz'"
      tags : [one, two]
    - common
    - { role: common, port: 5000, when: "bar == 'Baz'", tags :[one, two] }
  tasks:
    - include: tasks.yaml
    - include: tasks.yaml
      when: day == 'Thursday'
      vars:
        foo: aaa
        baz:
          - z
          - y
    - { include: tasks.yaml, foo: zzz, baz: [a,b]}
    - <task>
  post_tasks:
    - <task>
```

Running Playbook

 # Run on all hosts defined
ansible-playbook <YAML>
 # Run 10 hosts parallel
ansible-playbook <YAML> -f 10
 # Verbose on successful tasks
ansible-playbook <YAML> --verbose
 # Test run
ansible-playbook <YAML> -C
 # Dry run
ansible-playbook <YAML> -C -D
 # Run on single host using -l or -limit (-l stands for limit)
ansible-playbook <YAML> -l <host>
e.g. ansible-playbook new_playbook.yml

Verifying playbooks

Get Infos:
ansible-playbook <YAML> --list-hosts
ansible-playbook <YAML> --list-tasks
Syntax Check
ansible-playbook --syntax-check <YAML>

We can also use **ansible-lint** for detailed, Ansible-specific feedback on your playbooks before you execute them. Click here for the documentation

Ansible Vault

What is Ansible Vault?

If our Ansible playbooks deal with sensitive data like passwords, API keys, and credentials, it is important to keep that data safe by using an encryption mechanism. Ansible provides ansible-vault to encrypt files and variables.

After encrypting a file with this tool, we will only be able to execute, edit or view its contents by providing the relevant password defined when we first encrypted the file.

Running Playbook with Vault

ansible-playbook myplaybook.yml --ask-vault-pass
ansible-playbook myplaybook.yml --vault-password-file path/to/passfile
ansible-playbook myplaybook.yml --vault-id dev@prompt
ansible-playbook myplaybook.yml --vault-id dev@path/to/passfile

Working With Ansible Vault

Creating a New Encrypted File
ansible-vault create credentials.yml

Encrypting an Existing Ansible File
ansible-vault encrypt credentials.yml

View encrypted file
ansible-vault view credentials.yml

Edit encrypted file
ansible-vault edit credentials.yml

Permanently Decrypt a file
ansible-vault decrypt credentials.yml

Using Multiple Vault Passwords for multiple environments
We can have dedicated vault passwords for different environments, such as development, testing, and production environments
ansible-vault create --vault-id dev@prompt credentials_dev.yml
ansible-vault create --vault-id prod@prompt credentials_prod.yml
To Edit/edit have to provide the same id
ansible-vault edit credentials_dev.yml --vault-id dev@prompt

Using a Password File
ansible-vault create --vault-password-file path/to/passfile credentials_dev.yml
ansible-vault create --vault-id dev@path/to/passfile credentials_dev.yml

Ansible Galaxy

What is Ansible Galaxy?

Ansible Galaxy is a repository for Ansible Roles that are available to drop directly into your Playbooks to streamline your automation projects.

How to Use Ansible Galaxy?

Create a role template suitable for submission to Ansible Galaxy.
ansible-galaxy init

display a list of installed roles, with version numbers
ansible-galaxy list

Remove an installed role.
ansible-galaxy remove <role>

Get a variety of information about Ansible Galaxy
ansible-galaxy info <role>

Install role from galaxy
ansible-galaxy install <role-name> -p <directory>

Search for a role
ansible-galaxy search 'install git' --platform el
or
Visit here galaxy.ansible.com

http:// Learn.sandipdas.in

Contact Me

contact@sandipdas.in