Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

Powered by simplilearn

DevOps and Cloud Computing

Caltech | Center for Technology & Management Education

Configuration Management with Ansible and Terraform

Powerd by simplilearn

Advanced Ansible

A Day in the Life of a DevOps Engineer

You are working in a startup company that uses Ansible to automate cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

The organization is worried about sensitive data such as passwords or keys and is looking for a solution that helps encrypt sensitive data. Additionally, employees are also looking for a feature that can help them control how Ansible responds to task errors.

The organization is looking for a solution that can help them work against numerous managed nodes or hosts in the infrastructure simultaneously.

A Day in the Life of a DevOps Engineer

Lastly, the team is looking for a solution that can make Ansible easier to use for the IT department. It is intended to serve as the central hub for all automation operations.

To achieve all the above, along with some additional features, you will be learning a few concepts in this lesson that will help find a solution for the given scenario.

Learning Objectives

By the end of this lesson, you will be able to:

- Configure Ansible Vault
- Troubleshoot and handle errors
- Manage Inventory Variables
- Describe Ansible Tower
- Analyze Ansible as a cloud provisioning tool

Ansible Vault

What Is Ansible Vault?

Ansible Vault is a feature that helps encrypt sensitive data. Users can protect sensitive content such as passwords or keys instead of leaving them visible as plaintext in playbooks or roles.

What Is Ansible Vault?

The specific operations performed by Ansible Vault are:

Encrypt a file

Decrypt a file

View an encrypted file

Edit an encrypted file

Create an encrypted file

Generate or reset an encrypted file

Encrypting Content with Ansible Vault

Encrypting or decrypting content with Ansible Vault requires one or more passwords.

Storing passwords in a third-party tool requires a script to access them

Utilize the passwords with ansible-vault command-line tool to encrypt or decrypt content

Encrypted content can be placed under source control and shared securely

Encrypting Content with Ansible Vault

The encrypted variables and files can be used in ad hoc commands and playbooks by supplying the appropriate passwords used during encryption.

ansible.cfg

The user can modify the ansible.cfg file to specify the location of a password file or to always prompt for the password.

Encrypting Content with Ansible Vault

Users can encrypt variables and files with Ansible Vault.

!vault tag	This tag tells Ansible and YAML that the content needs to be decrypted
| character	This character allows multi-line strings
--vault-id	Encrypted content created with --vault-id also contains the vault ID label

Encrypting Individual Variables with Ansible Vault

Use the **ansible-vault encrypt_string** command to encrypt single values inside a YAML file.

This command encrypts and formats any string as per the requirement into a format that can be included in a playbook, role or variables file.

Encrypting Individual Variables with Ansible Vault

The ansible-vault encrypt_string command to create a basic encrypted variable is given below:

```
ansible-vault encrypt_string <password_source> '<string_to_encrypt>' --name '<string_name_of_variable>'
```

The three options passed in the command above are:

- A source for the vault password (prompt, file, or script, with or without a vault ID)
- The string to encrypt
- The string name (the name of the variable)

Encrypting Individual Variables with Ansible Vault

For instance, to encrypt the string *samplestring* using the only password stored in *a_password_file* and name the variable *the_secret*:

```
ansible-vault encrypt_string --vault-password-file a_password_file 'samplestring' --name 'the_secret'
```

The output of the command above is:

```
the_secret: !vault |
      $ANSIBLE_VAULT;1.1;AES256
      62313365396662343061393464336163383764373764613633653634306231386433626436623361
      61343336653539663635343336326665353337616661316 20a663537646436643839616531643561
      63396265333966386166373632626539326166353965363262633030333630313338646335303630
      34386266666661376 50a353638643435666633633964366338633066623234616432373231333331
      6564
```

Encrypting Individual Variables with Ansible Vault

The user can view the original value of the encrypted variable in the debug module.

```
ansible localhost -m ansible.builtin.debug -a var="the_secret" -e "@vars.yml" --vault-id
dev@a_password_file

localhost | SUCCESS => {
    "the_secret": "samplestring"
}
```

Encrypting Files with Ansible Vault

Ansible Vault can encrypt any structured data file used by Ansible. The full file is encrypted in the vault.

- Group variables files from inventory
- Host variable files from inventory
- Variables files in roles
- Default files in roles
- Tasks files and handler files
- Binary files or arbitrary files
- Variables files loaded by include_vars or vars_files
- Variables files passed to ansible-playbook with -e @file.yml or -e @file.json

Encrypting Files with Ansible Vault

The command to create a new encrypted data file called *sample.yml* with the *test* vault password from *multi_password_file* is given below:

```
ansible-vault create --vault-id test@multi_password_file sample.yml
```

This command launches an editor where the content is to be added. On closing the editor, the file is saved as encrypted content. The file header shows the vault ID used to create it.

```
``$ANSIBLE_VAULT;1.2;AES256;test``
```

Encrypting Files with Ansible Vault

The command to encrypt an existing file is ansible-vault encrypt. It can operate on multiple files.

```
ansible-vault encrypt foo.yml bar.yml baz.yml
```

The command to view the contents of an encrypted file without editing, is the ansible-vault view.

```
ansible-vault view foo.yml bar.yml baz.yml
```

Encrypting Files with Ansible Vault

The commonly used commands while encrypting files using Ansible Vault are:

ansible-vault edit foo.yml	To edit an encrypted file in place
ansible-vault rekey foo.yml	To change the password on an encrypted file
ansible-vault decrypt foo.yml	To permanently decrypt an encrypted file

Format of Files Encrypted with Ansible Vault

Ansible Vault creates UTF-8 encoded txt files.

$ANSIBLE_VAULT;1.1;AES256
or
$ANSIBLE_VAULT;1.2;AES256;vault-id-label

Example

- The file format contains a newline terminated header.
- The header contains up to four elements, separated by semicolons.

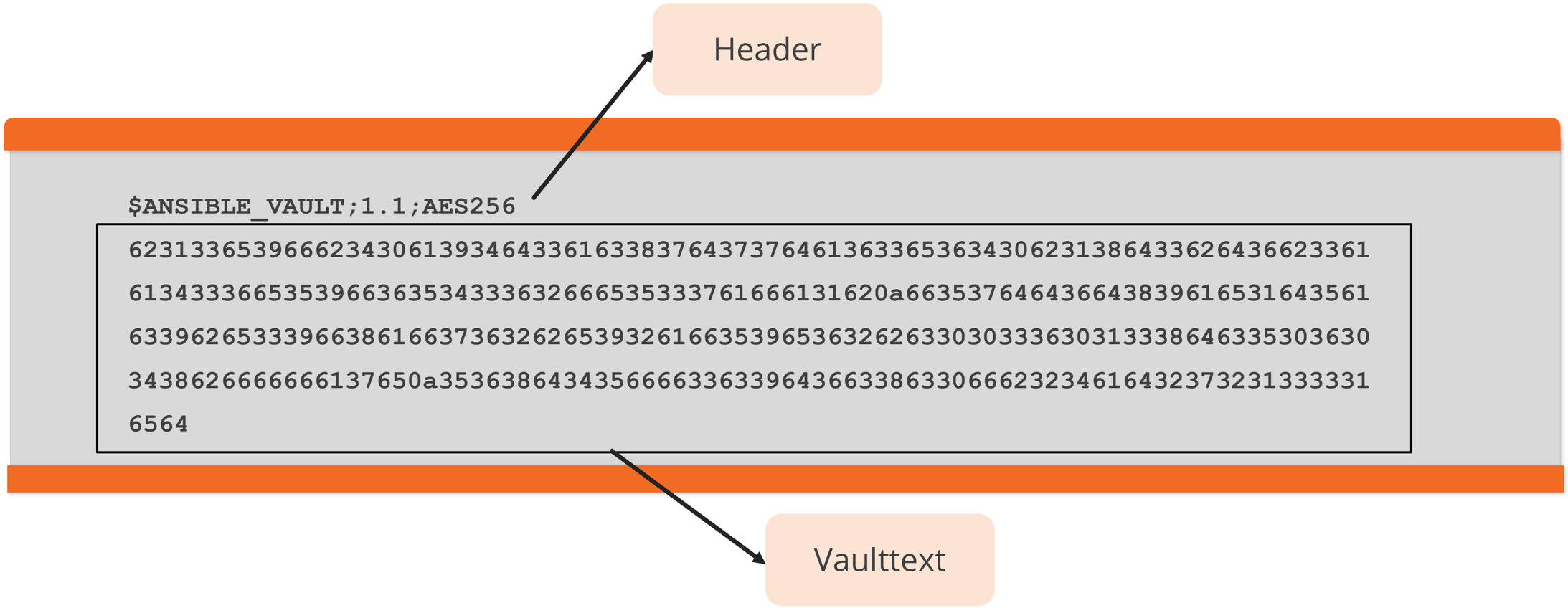
Format of Files Encrypted with Ansible Vault

The header components are given below:

$ANSIBLE_VAULT	It is the format ID
1.2	1.x is the vault format version
AES256	Cipher algorithm used to encrypt the data
vault-id-label	The vault ID label encrypts the data (optional)

Format of Files Encrypted with Ansible Vault

Apart from the header, the rest of the content of the file is *vaulttext,* which is an armored version of the encrypted ciphertext.



Assisted Practice

Managing Sensitive Files with Ansible Vault

Duration: 15 Min.

Problem Statement:

You have been assigned the task of managing sensitive files using Ansible Vault which includes operations like encrypting files, viewing encrypted files, manually decrypting encrypted files, changing the password of encrypted files, and running Ansible with vault encrypted files.

Assisted Practice: Guidelines

Steps to be followed:

1. Creating newly encrypted files
2. Encrypting existing files
3. Viewing and editing encrypted files
4. Decrypting encrypted files manually
5. Changing the password of encrypted files
6. Running Ansible with vault-encrypted files

Error Handling and Troubleshooting

Error Handling in Ansible Using Blocks

Users can control how Ansible responds to task errors using blocks with ***rescue*** and ***always*** sections.

```
tasks:
 - name: Handle the error
   block:
     - name: Print a message
       ansible.builtin.debug:
         msg: 'I execute normally'

     - name: Force a failure
       ansible.builtin.command: /bin/false

     - name: Never print this
       ansible.builtin.debug:
         msg: 'I never execute, due to the above task failing, :-('
   rescue:
     - name: Print when errors
       ansible.builtin.debug:
         msg: 'I caught an error, can do stuff here to fix it, :-)'
```

- Rescue blocks define the tasks to run when an earlier task in a block fails.
- Ansible only runs rescue blocks when a task returns to a 'failed' state.
- Incorrect task definitions and unreachable hosts will not trigger the rescue block.

Error Handling in Ansible Using Blocks

The code defined in the *always* section executes regardless of the task status of the previous block.

```
- name: Always do X
  block:
    - name: Print a message
      ansible.builtin.debug:
        msg: 'I execute normally'

    - name: Force a failure
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I never execute :-('
  always:
    - name: Always do this
      ansible.builtin.debug:
        msg: "This always executes, :-)"
```

Error Handling in Ansible Using Blocks

```
- name: Attempt and graceful roll back demo
  block:
    - name: Print a message
      ansible.builtin.debug:
        msg: 'I execute normally'

    - name: Force a failure
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I never execute, due to the above task failing, :-('
  rescue:
    - name: Print when errors
      ansible.builtin.debug:
        msg: 'I caught an error'

    - name: Force a failure in middle of recovery! >:-)
      ansible.builtin.command: /bin/false

    - name: Never print this
      ansible.builtin.debug:
        msg: 'I also never execute :-('
  always:
    - name: Always do this
      ansible.builtin.debug:
        msg: "This always executes"
```

- The combination of always and rescue allows for complex error handling.
- The tasks in the block run as expected.
- Whenever a block return fails, the rescue section executes tasks to fix it.
- Block and rescue results do not affect this section.

Error Handling in Ansible Using Blocks

If an error occurs in the block and the rescue task succeeds, Ansible reverts the failed status of the original task for the run.

It continues to play as if the original task had succeeded.

The rescued task is considered successful and does not trigger *max_fail_percentage* or *any_errors_fatal* configurations.

Ansible still reports a failure in the playbook statistics.

Error Handling in Ansible Using Blocks

Use blocks with *flush_handlers* in a rescue task to ensure all handlers run even if an error occurs.

```
tasks:
  - name: Attempt and graceful roll back demo
    block:
      - name: Print a message
        ansible.builtin.debug:
          msg: 'I execute normally'
        changed_when: yes
        notify: run me even after an error

      - name: Force a failure
        ansible.builtin.command: /bin/false
    rescue:
      - name: Make sure all handlers run
        meta: flush_handlers
handlers:
   - name: Run me even after an error
     ansible.builtin.debug:
       msg: 'This handler runs even on error'
```

Error Handling in Ansible Using Blocks

There are two new variables in Ansible 2.1 for tasks in the rescue portion of the block:

01

ansible_failed_task: The task that returned *failed* and triggered the rescue

02

ansible_failed_result: When a task fails, it captures the return value

Troubleshooting: Tips and Tricks

Some tips and tricks for troubleshooting in Ansible are:

Check for bad syntax	ansible-playbook playbooks/PLAYBOOK_NAME.yml --syntax-check
Flush the redis cache	ansible-playbook playbooks/PLAYBOOK_NAME.yml --flush-cache
Run a playbook in dry-run mode	ansible-playbook playbooks/PLAYBOOK_NAME.yml --check

Troubleshooting: Tips and Tricks

Run a playbook in verbose mode	From the command line, add -v (or -vv, -vvv, -vvvv, -vvvvv)
Task debugger	Ansible offers a task debugger that can be enabled at the task level or the play level
Run a playbook interactively	ansible-playbook playbook.yml --step
Run a playbook from a particular task	ansible-playbook playbook.yml --start-at-task="<Your_task>"

Working with Dynamic Inventory

Dynamic Inventory

Ansible can generate host inventory dynamically.

The dynamic inventory provides information about public and private cloud providers, cobbler system information, LDAP database, and CMDB (Configuration Management database).

Dynamic Inventory

Ansible supports two ways to connect with the dynamic external inventory.

Inventory Plugins

Inventory Scripts

Inventory plugins are recommended over scripts since they are up-to-date with the Ansible core code and are customizable.

Dynamic Inventory Plugins

Inventory plugins enable users to point to data sources to generate the inventory of hosts used by Ansible to target jobs.

Points to Remember:

- The majority of the inventory plugins included with Ansible are activated by default or can be utilized with the auto plugin.
- Modify YAML configuration file to override the default list of enabled plugins. Here is the default list of enabled plugins that ship with Ansible:

  [inventory] enable_plugins = host_list, script, auto, yaml, ini, toml

Dynamic Inventory Scripts

Ansible uses the inventory script to retrieve host information from an external inventory system.

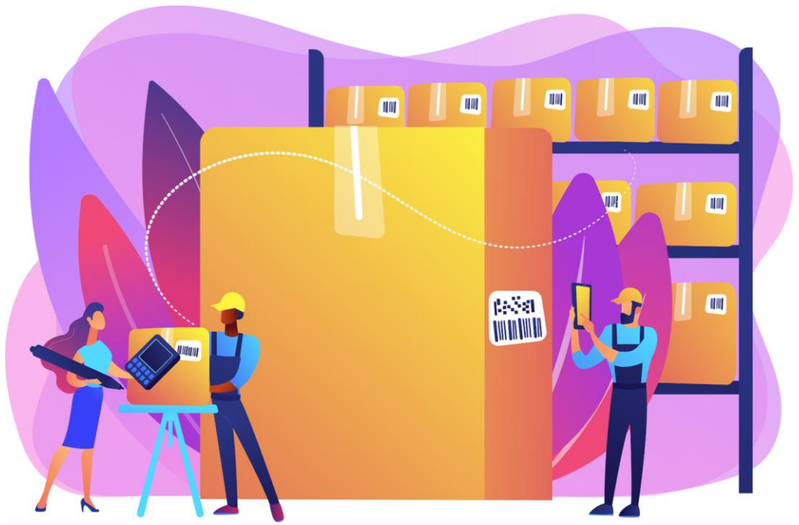
```
{
    "group001": {
        "hosts": ["host001", "host002"],
        "vars": {
            "var1": true
        },
        "children": ["group002"]
    },
    "group002": {
        "hosts": ["host003","host004"],
        "vars": {
            "var2": 500
        },
        "children":[]
    }

}
```

Asset scripts should accept --list and --host <hostname> arguments, returning host group and hosts information.

Managing Inventory Variables

Managing Inventory Variables

Ansible uses a list or a group of lists known as inventory to work against numerous managed nodes or hosts in the infrastructure simultaneously.



Users can use patterns to select which hosts or groups they want Ansible to run against after they've constructed the inventory.

Inventory Basics: Formats

Depending on the inventory plugins you have, the inventory file can be in a variety of formats. INI and YAML are the most commonly used formats.

A basic INI file in */etc/ansible/hosts* looks like this:

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

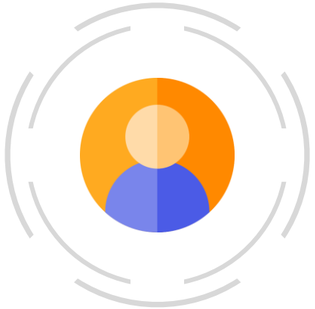
Inventory Basics: Formats

The following code is the sample INI file in YAML format:

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Inventory Basics: Hosts and Groups

There are two types of groups available to connect via hosts:



Default Groups

Multiple Groups

Connecting Hosts

The following variables influence how Ansible communicates with remote hosts:

Variable	Description
ansible_host	The hostname to connect to
ansible_port	The connection port number (Default is 22 for ssh)
ansible_user	The username when connecting to the host
ansible_password	The password to authenticate the host

SSH Connection

The settings listed below are specific to SSH ports:

Variable	Description
ansible_ssh_private_key_file	Private key file used by ssh. Used when you don't want to use ssh-agent
ansible_ssh_common_args	Appended to the default command line for **SFTP**, **SCP**, and **SSH**
ansible_sftp_extra_args	Appended to the default **SFTP** command line
ansible_scp_extra_args	Appended to the default **SCP** command line
ansible_ssh_extra_args	Appended to the default **SSH** command line
ansible_ssh_pipelining	Determines whether or not to use SSH pipelining
ansible_ssh_executable	Overrides the default behavior to use the system **SSH**

Ansible Privilege Escalation

The following variables are used to configure privilege escalation:

Variable	Description
ansible_become	Allows to force privilege escalation
ansible_become_method	Allows to set privilege escalation method
ansible_become_user	Allows to set the user you become through privilege escalation
ansible_become_password	Allows to set the privilege escalation password (never assign this variable to plain text, always use a vault)
ansible_become_exe	Allows to set the executable for the escalation method selected
ansible_become_flags	Allows to set the flags passed to the selected escalation method

Non-SSH Connection

Ansible runs playbooks over SSH, however, this is not the only connection method available. The connection type can be changed using the host-specific parameter ***ansible_connection=<connector>***.

These are the following Non-SSH connectors available:

Local Connector

Docker Connector

Non-SSH Connection

The following variables are processed by Docker connector:

Variable	Description
ansible_host	Name of the Docker container to connect
ansible_user	Username to operate within the container
ansible_become	When set to **true,** *become_user* will be operated in the container
ansible_docker_extra_args	A string containing any additional arguments recognized by Docker

Deployment in Non-SSH Connection

The following code is an example for deployment to the containers:

```
- name: Create a jenkins container
  community.general.docker_container:
    docker_host: myserver.net:4243
    name: my_jenkins
    image: jenkins

- name: Add the container to inventory
  ansible.builtin.add_host:
    name: my_jenkins
    ansible_connection: docker
    ansible_docker_extra_args: "--tlsverify --tlscacert=/path/to/ca.pem"
    ansible_user: jenkins
  changed_when: false

- name: Create a directory for ssh keys
  delegate_to: my_jenkins
  ansible.builtin.file:
    path: "/var/jenkins_home/.ssh/jupiter"
    state: directory
```

Ansible Tower

Ansible Tower

Ansible Tower (Formerly known as AWX) is a web-based solution that makes Ansible easier to use for the IT department. It is intended to serve as the central hub for all the automation operations.

Ansible Tower Features

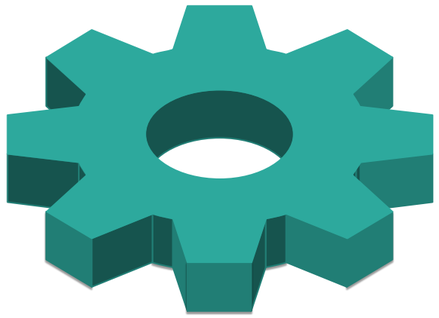
The following are the features of Ansible tower:

Ansible as Cloud Provisioning Tool

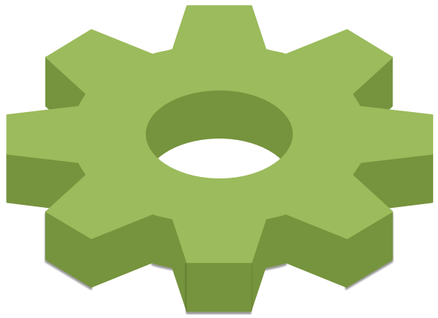
Ansible Provisioning

Ansible lets users smoothly transition into configuration management, orchestration, and application deployment using the same basic, human-readable automation language.

These are the infrastructure platforms available in Ansible Provisioning:



Baremetal



Virtualized

Cloud Provisioning

There are two types of cloud available for provisioning:

Public Cloud

Private Cloud

Public Cloud

Ansible is compiled with hundreds of modules that support resources on the world's largest public cloud platforms.

- Compute, storage, and network components allow playbooks to provision these resources directly.
- Ansible can serve as an orchestrator for other popular delivery tools, providing you with a high-quality, self-paced workflow.

Private Cloud

Ansible is one of the simplest ways to deploy, configure, and orchestrate an OpenStack private cloud.

- Ansible can be used to provision the core infrastructure, install services, and add compute hosts, among other things.
- Ansible can be used to provision resources, services, and applications inside the cloud, once the core infrastructure has been provisioned.

Key Takeaways

- Ansible Vault is a feature that helps encrypt sensitive data. You can protect sensitive content such as passwords instead of leaving them visible.
- Ansible Vault can encrypt any structured data file used by Ansible. The full file is encrypted in the vault.
- Ansible Tower (Formerly known as AWX) is a web-based solution that makes Ansible easier to use for the IT department.
- Ansible uses a group of lists known as inventory to work against numerous managed nodes in the infrastructure simultaneously.

Dynamic Inventory with AWS

Duration: 25 Min.

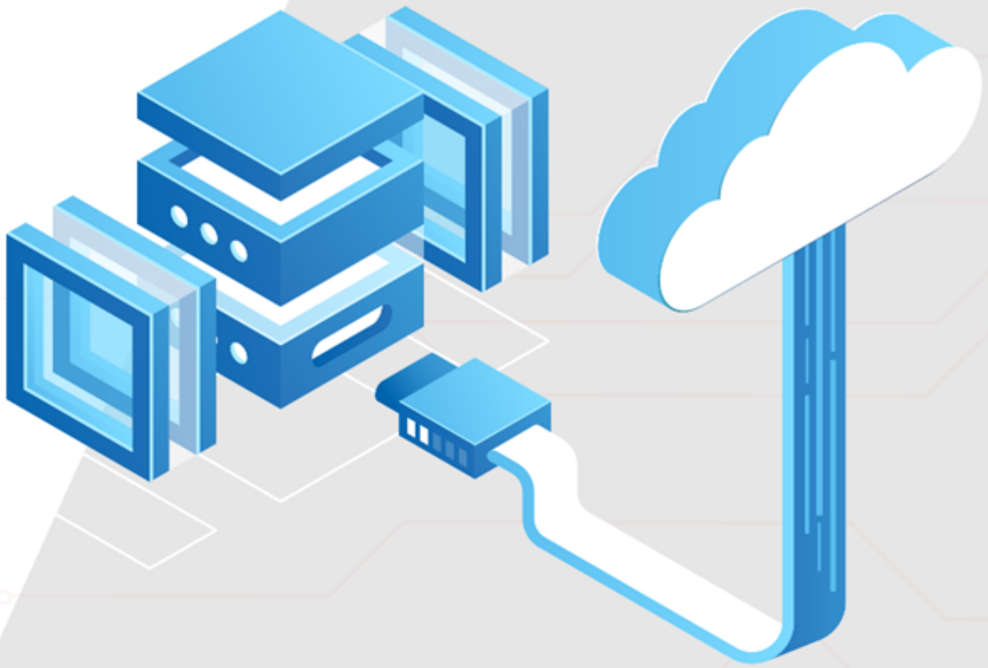
Project agenda: To build dynamic inventory with AWS

Description: Ansible is an open-source IT automation tool. It is straightforward to use but so powerful. Ansible is suitable for managing all environments, from small setups with a few instances to big ones with hundreds of instances. Ansible has a dynamic external inventory system that has two ways to use external inventory: inventory scripts and the most recent updated inventory plugin.

Perform the following:

- Pinging the target nodes with static inventory
- Working With dynamic inventory

LESSON-END PROJECT



Thank you