Flask App Deployment with Ansible

This Ansible playbook automates the deployment of a Flask application on a remote host. If the target host is unavailable, the playbook will execute the deployment tasks on the local machine.

Features

1. **Install CA certificates** (both standard and custom).
2. **Validate custom CA certificates** to ensure they are not expired.
3. **Deploy application files** and configurations to the target or local host.
4. **Set up a Python virtual environment** for running the application.
5. **Install the Flask app using a wheel file**.
6. **Configure and run the Flask app** using Gunicorn as the WSGI server.
7. **Create and enable a systemd service** for managing the app.
8. **Verify that the app is running** by checking the Gunicorn process.
9. **Automatic fallback to local machine** if the target host is unreachable.

Prerequisites

- Ensure you have Ansible installed on the control node.
- The target machine should have Python 3 installed.
- You should have the following files available:
  - `app.py` : The Flask application file.
  - Flask app wheel file (`Example-1.1.2-py3-none-any.whl`).
  - SSL certificate files (`CA1.crt` , `CA2.crt` , `CA3.crt`).

- Configuration files (`config.py` , `config.py.j2` , and `run.sh.j2`).
- A systemd service template (`example.service.j2`).

Variables

Sensitive information such as passwords or secrets should be placed in `vault.yml` , which is encrypted with Ansible Vault.

▶ Click to show password

Fallback Behavior (Local Machine Deployment)

If the target host defined in your inventory is unavailable, the playbook will automatically detect this and switch to running the deployment on the local machine. This ensures that development or testing environments on the control machine can still be set up even if remote servers are down.

Ansible checks the availability of the target host before executing tasks. If it detects that the remote host is unreachable, the `local` connection will be used, and the tasks will proceed on the control node.

Inventory Setup

Remote Host Inventory

To set up the inventory for a remote host, create an `inventory` file that lists the target host(s). The file should be in the following format:

```
[target]
<target-ip> ansible_user=<user> ansible_ssh_private_key_file=~/.ssh/i
```

Usage

1. Clone the repository or ensure the playbook files are in your working directory.
2. Ensure you have the correct inventory file pointing to the target host(s).
3. Run the playbook with the following command:

```
ansible-playbook -i <inventory> deploy_flask_app.yml
```

Local Host Inventory

To set up the inventory for deployment on the local machine, create an inventory file like this:

```
[local]
localhost ansible_connection=local
```

Encrypting Sensitive Data

Encrypt your sensitive information (e.g., database credentials, secrets) in vault.yml using Ansible Vault:

```
ansible-vault encrypt vault.yml
```

You will be prompted to enter a password to encrypt the file.

Running the Playbook with Vault Encryption

To run the playbook and provide the Vault password for decryption, you have two options:

- Option 1: Enter the Vault Password at Runtime When running the playbook, use the --ask-vault-pass flag to be prompted for the Vault password:

```
ansible-playbook -i inventory deploy_flask_app.yml --ask-vault-pass
```

This will prompt you to enter the Vault password interactively.

- Option 2: Use a Vault Password File Alternatively, create a text file that contains your Vault password (make sure to secure this file properly) and pass it in using the --vault-password-file option:

Create a file vault_password.txt with your Vault password (do not commit this file to version control).

Run the playbook with the following command:

```
ansible-playbook -i inventory deploy_flask_app.yml --vault-password-f
```

This method avoids interactive prompts and allows automated deployments.

Application Running Steps

Follow these steps to run the Flask application using the playbook:

- Prepare the Inventory: Set up your inventory file for either remote or local deployment.
- Encrypt Sensitive Data: Use Ansible Vault to encrypt vault.yml:

```
ansible-vault encrypt vault.yml
```

Run the Playbook:

For remote deployment:

```
ansible-playbook -i inventory deploy_flask_app.yml --ask-vault-pass
```

- or, using the password file:

```
ansible-playbook -i inventory deploy_flask_app.yml --vault-password-f
```

- For local deployment:

```
ansible-playbook -i localhost, -c local deploy_flask_app.yml --ask-va
```

- or, using the password file:

```
ansible-playbook -i localhost, -c local deploy_flask_app.yml --vault-
```

- Check Application Status: After the playbook completes, check that the Flask application is running via Gunicorn:

```
systemctl status example
```

- Or verify Gunicorn is running:

```
pgrep -f gunicorn
```

- Decrypting Encrypted Files If you need to modify the encrypted vault.yml file, decrypt it with:

```
ansible-vault decrypt vault.yml
```

- Once your changes are made, encrypt the file again:

```
ansible-vault encrypt vault.yml
```

Application Deployment Workflow

- Set up the inventory file (remote or local).
- Encrypt sensitive data using Ansible Vault.
- Run the playbook with the Vault password or Vault password file.
- Verify that the Flask app is running.