Python with Ansible - Yatri Cloud

Creator: Yatharth Chauhan

Integrating Python with Ansible allows you to automate IT tasks such as configuration management, application deployment, and task orchestration. Ansible is an open-source automation tool that uses a declarative language to describe the desired state of infrastructure. While Ansible is primarily used through its command-line interface or YAML playbooks, Python can be used to enhance automation, manage playbooks, and interact with Ansible's API.

1. **Playbooks**: YAML files that define a series of tasks to be executed on specified hosts.
2. **Modules**: Small, reusable scripts that perform specific tasks, such as installing software or managing files.
3. **Inventory**: A file that lists the hosts and groups of hosts that Ansible manages.
4. **Tasks**: Actions performed by Ansible, defined within playbooks.

Setting Up Ansible

1. **Installation**: You can install Ansible using `pip` or your system's package manager. Here's how to install it using `pip`:

   ```
   pip install ansible
   ```

2. **Create Inventory File**: Create a file named `inventory.ini` that lists the hosts you want to manage.

   ```
   [web]
   server1 ansible_host=192.168.1.100

   [db]
   server2 ansible_host=192.168.1.101
   ```

Example 1: Running Ansible Playbooks from Python

You can run Ansible playbooks from Python using the `subprocess` module. Here's an example of how to do that.

Step 1: Create a Sample Playbook

Create a simple Ansible playbook named `playbook.yml` that installs `nginx` on the hosts in the `web` group.

```
---
- name: Install Nginx
  hosts: web
  become: true
  tasks:
```

```
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

Step 2: Python Script to Run the Playbook

Here's a Python script that runs the Ansible playbook using the `subprocess` module.

```
import subprocess

def run_ansible_playbook(playbook_path, inventory_path):
    command = ['ansible-playbook', '-i', inventory_path, playbook_path]
    try:
        result = subprocess.run(command, check=True, capture_output=True,
text=True)
        print("Playbook executed successfully:")
        print(result.stdout)
    except subprocess.CalledProcessError as e:
        print("Error executing playbook:")
        print(e.stderr)

# Paths to the playbook and inventory files
playbook_file = 'playbook.yml'
inventory_file = 'inventory.ini'

# Run the Ansible playbook
run_ansible_playbook(playbook_file, inventory_file)
```

Explanation:

- **subprocess.run()**: Executes the command to run the Ansible playbook.
- **check=True**: Raises an exception if the command returns a non-zero exit code.
- **capture_output=True**: Captures the standard output and error.

Example 2: Using Ansible Runner

Ansible Runner is a tool that helps to run Ansible tasks and playbooks and provides an API to interact with Ansible. You can use it to run Ansible playbooks directly from Python.

Step 1: Install Ansible Runner

Install Ansible Runner using pip:

```
pip install ansible-runner
```

Step 2: Create a Project Structure

Create a directory structure that Ansible Runner expects:

```
yatricloud_ansible_project/
│
├── inventory/
│   └── hosts
│
├── playbook.yml
└── project.yml
```

- **inventory/hosts**: Your inventory file (similar to `inventory.ini`).
- **playbook.yml**: Your Ansible playbook.
- **project.yml**: A configuration file for Ansible Runner.

Here's how the contents might look:

inventory/hosts:

```
[web]
server1 ansible_host=192.168.1.100
```

playbook.yml:

```yaml
---
- name: Install Nginx
  hosts: web
  become: true
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

project.yml:

```yaml
# project.yml content
---
playbook: playbook.yml
inventory: inventory/hosts
```

Step 3: Running the Playbook with Ansible Runner

Now you can create a Python script to run your playbook using Ansible Runner:

```
import ansible_runner

def run_playbook():
    r = ansible_runner.run(private_data_dir='yatricloud_ansible_project',
playbook='project.yml')

    if r.status == 'successful':
        print("Playbook executed successfully.")
    else:
        print(f"Playbook execution failed: {r.status}")
        print(f"Output: {r.stdout}")

run_playbook()
```

Explanation:

- **ansible_runner.run()**: Runs the Ansible playbook with the specified configuration.
- **private_data_dir**: Directory containing the inventory and playbook files.

Example 3: Creating Ansible Playbooks Dynamically with Python

You can also use Python to generate Ansible playbooks dynamically based on certain conditions or inputs.

```
def create_playbook(content):
    with open('dynamic_playbook.yml', 'w') as file:
        file.write(content)

# Dynamic content generation
playbook_content = '''---
- name: Install dynamic package
  hosts: web
  become: true
  tasks:
    - name: Install package
      apt:
        name: {package_name}
        state: present
'''.format(package_name='curl')

# Create the playbook
create_playbook(playbook_content)

# Now you can run this playbook as shown in previous examples
```

Best Practices for Using Python with Ansible

1. **Use Inventory Variables**: Store common variables in your inventory to avoid repetition in playbooks.
2. **Error Handling**: Implement error handling when executing Ansible commands from Python.
3. **Version Control**: Keep your playbooks and Python scripts in a version control system for better collaboration.
4. **Modular Playbooks**: Break down complex tasks into multiple playbooks or roles for better organization and maintainability.
5. **Documentation**: Document your playbooks and Python scripts to ensure clarity for future maintenance.

Integrating Python with Ansible provides a powerful way to automate and manage IT infrastructure. By using tools like `subprocess`, `ansible-runner`, or dynamic playbook generation, you can streamline your DevOps processes, enhance automation, and improve the efficiency of your infrastructure management tasks. This integration allows you to leverage the strengths of both Python and Ansible, making your automation workflows more flexible and powerful.
