lesson-05-ansible-for-deployment.md

Deployment with Ansible

This lesson introduces automated deployment using Ansible. Whereas custom Python scripts require you to write logic for SSH access, file transfer, command execution, and idempotency, Ansible provides a structured system that automates these features for you. Understanding how Ansible works makes it easier to build and maintain reliable deployment pipelines.

This lesson uses simplified examples that follow the same structure as real-world Ansible playbooks.

---

1. What Ansible Does

Ansible is a configuration management and orchestration tool built around three concepts:

- **Inventory:** A list of servers to manage
- **Modules:** Units of work such as copying files or installing packages
- **Playbooks:** Declarative instructions that ensure the desired system state

Ansible is **agentless**, meaning it uses SSH and does not require installing software on remote servers.

Compared to Python-only approaches, Ansible provides:

- Automatic idempotency
- Built-in modules for common tasks
- Clear YAML-based configuration
- Easy reuse across environments

---

2. Basic Inventory File

An inventory file lists servers that Ansible will manage. This is typically in `inventory.ini`.

```
[webservers]
server1.example.com
server2.example.com

[appservers]
app1.example.com ansible_user=ubuntu
```

Groups (such as `webservers` or `appservers`) allow you to target specific subsets of machines.

---

3. Running Commands with Ansible Ad-Hoc

Ansible can run commands without writing a playbook.

```
ansible all -i inventory.ini -m ping
```

```
ansible webservers -i inventory.ini -m shell -a "uname -a"
```

This is similar to writing Python scripts that call `run_command()`, but Ansible manages:

- SSH
- Output formatting
- Host iteration

---

4. Writing Your First Playbook

A playbook expresses system state declaratively.
Below is a simple playbook that updates packages and ensures Python dependencies are installed.

```yaml
---
- name: Configure application server
  hosts: appservers
  become: yes

  tasks:
    - name: Update apt repositories
      apt:
        update_cache: yes

    - name: Install required packages
      apt:
        name:
          - python3
          - python3-pip
        state: present

    - name: Ensure application directory exists
      file:
        path: /opt/myapp
        state: directory
```

Each task uses a standard module (`apt`, `file`, etc.) that implements idempotency automatically.

---

5. Deploying an Application with Ansible

This example mirrors the Python-based deployment workflow but uses Ansible modules.

```
---
- name: Deploy application
  hosts: appservers
  become: yes

  tasks:
    - name: Upload application file
      copy:
        src: myapp.py
        dest: /opt/myapp/myapp.py

    - name: Install dependencies
      pip:
        requirements: /opt/myapp/requirements.txt

    - name: Restart application service
      systemd:
        name: myapp
        state: restarted
```

Modules used:

- `copy`: replaces SFTP upload
- `pip`: installs Python dependencies
- `systemd`: manages services

---

6. Template Files in Ansible

Template files allow you to generate configuration dynamically using variables.

Example Jinja2 template (`myapp.service.j2`):

```
[Unit]
Description=My Sample Application

[Service]
ExecStart=/usr/bin/python3 /opt/myapp/myapp.py
Restart=always

[Install]
WantedBy=multi-user.target
```

Playbook that installs the rendered service file:

```
- name: Install systemd unit
  hosts: appservers
  become: yes
```

```
  tasks:
    - name: Upload systemd service template
      template:
        src: myapp.service.j2
        dest: /etc/systemd/system/myapp.service

    - name: Reload systemd
      systemd:
        daemon_reload: yes

    - name: Start service
      systemd:
        name: myapp
        state: started
```

---

7. Running the Playbook

The basic command to apply the configuration:

```
ansible-playbook -i inventory.ini deploy.yml
```

Ansible connects to each host and executes tasks in order.

---

8. Comparing Ansible to Custom Python Automation

Feature	Python Scripts	Ansible
SSH	Manual	Automatic
File Transfers	Manual SFTP	Built-in modules
Idempotency	Must be written manually	Built-in
Service management	Shell commands	systemd module
Parallel execution	Must be coded	Built-in
Inventory management	Manual	Built-in

---

9. Summary

Ansible automates:

- System configuration
- Software installation
- Application deployment
- Service management

It eliminates the need to write SSH, SFTP, and idempotent logic manually.
The declarative configuration style makes deployments clearer and more reliable.