
Lab 7: Automate Remote Deployment with Ansible Playbooks

Lab overview

In this lab, you will explore automating remote deployments using Ansible playbooks. You'll create inventory files, write deployment playbooks, automate service management and file operations, handle errors gracefully, and design modular, reusable automation. By the end of this lab, you'll have a complete Ansible-based deployment system that can manage multiple servers and applications automatically.

In this lab, you will:

- Write basic deployment playbooks with YAML syntax
- Automate service restart and file copying operations
- Handle errors and playbook failures gracefully
- Practice YAML structuring and modular playbook design

Estimated completion time

60 minutes

Task 1: Setting up Ansible and project structure

In this task, you will install Ansible, create inventory files, and understand how Ansible manages remote hosts.

1. Create a folder named **Lab7** on the Desktop. Right-click on the Desktop, click **New Folder**. Name the folder **Lab7**. Open Visual Studio Code and open the Lab7 folder.
2. Install Ansible and fix compatibility issues: Open VSCode terminal (Terminal → New Terminal). Install Ansible and dependencies:

```
# Update package list
```

```
sudo apt update
```

```
# Install Ansible
```

```
sudo apt install ansible -y
```

```
# Fix Jinja2 compatibility issues
```

```
pip3 uninstall jinja2 -y 2>/dev/null || true
```

```
pip3 install --user "jinja2==3.0.3"
```

```
# Install additional tools for SSH key management
```

```
sudo apt install sshpass -y
```

```
# Add local bin to PATH
```

```
export PATH="$HOME/.local/bin:$PATH"
```

```
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.bashrc
```

```
# Verify installation
```

```
ansible --version
```

3. Create basic project structure:

- In VSCode Explorer, create the following folder structure:
- Lab7 folder → **New Folder** → Name it **inventories**
- Lab7 folder → **New Folder** → Name it **playbooks**
- **playbooks** folder → **New Folder** → Name it **templates**
- Lab7 folder → **New Folder** → Name it **roles**
- Lab7 folder → **New Folder** → Name it **group_vars**

(Or from Terminal: mkdir -p inventories playbooks/templates roles group_vars)

Task 2: Creating the inventory files

1. Create inventory file for local testing:

- Click on **inventories** folder → **New File**
- Name it **development.yml**
- Add the following inventory content:

```
# Development Environment Inventory

all:
  children:
    webservers:
      hosts:
        web1:
          ansible_host: localhost
          ansible_connection: local
          ansible_user: student
          server_role: primary
          app_port: 8080
          app_name: flask-demo-web1

        web2:
          ansible_host: localhost
          ansible_connection: local
          ansible_user: student
```

```
server_role: secondary
app_port: 8081
app_name: flask-demo-web2

databases:
hosts:
db1:
ansible_host: localhost
ansible_connection: local
ansible_user: student
db_port: 5432
db_name: app_database

loadbalancers:
hosts:
lb1:
ansible_host: localhost
ansible_connection: local
ansible_user: student
lb_algorithm: round_robin

vars:
environment: development
app_environment: development
deploy_user: student
app_directory: "/home/student/deployed-apps"
ansible_python_interpreter: /usr/bin/python3
app_version: "1.0.0"
restart_services: true
```

Save the file by pressing Ctrl+S

Task 3: Creating the group variables

1. Create group variables:

- Click on **group_vars** folder → New File
- Name it **webservers.yml**
- Add the following content:

```
# Variables specific to webserver group
```

```
nginx_port: 80
```

```
nginx_worker_processes: auto
```

```
app_max_memory: 512M
```

```
health_check_url: "/health"
```

```
deployment_strategy: rolling
```

```
max_failure_percentage: 20
```

```
# Application configuration
```

```
app_config:
```

```
  debug: false
```

```
  log_level: INFO
```

```
  max_workers: 4
```

```
  timeout: 30
```

```
# Monitoring
```

```
monitoring_enabled: true
```

```
log_rotation_days: 30
```

```
# Environment setting
```

```
app_environment: development
```

Save the file by pressing **Ctrl+S**

Task 4: Creating basic deployment playbooks

In this task, you will create comprehensive deployment playbook for application deployment, service management, and configuration.

1. Create the main deployment playbook:

- Click on **playbooks** folder → New File
- Name it **deploy-app.yml**
- Add the following playbook content:

```
# Main Application Deployment Playbook
```

```
- name: Deploy Flask Application
```

```
hosts: webservers
```

```
become: yes
```

```
become_user: root
```

```
gather_facts: yes
```

```
vars:
```

```
  app_version: "1.0.0"
```

```
  restart_services: true
```

```
vars_files:
```

```
  - ../group_vars/webservers.yml
```

```
pre_tasks:
```

```
  - name: Update package cache
```

```
    apt:
```

```
      update_cache: yes
```

```
      when: ansible_os_family == "Debian"
```

```
      run_once: true
```

```
  - name: Create application directory
```

```
    file:
```

```
      path: "{{ app_directory }}"
```

```
      state: directory
```

```
    owner: "{{ deploy_user }}"
    group: "{{ deploy_user }}"

  - name: Create logs directory
    file:
      path: "{{ app_directory }}/logs"
      state: directory
      owner: "{{ deploy_user }}"
      group: "{{ deploy_user }}"

  tasks:
    - name: Install required system packages
      package:
        name:
          - python3
          - python3-pip
          - python3-venv
          - nginx
          - supervisor
          - curl
        state: present
      run_once: true

    - name: Create application files directory
      file:
        path: "{{ app_directory }}/app"
        state: directory
        owner: "{{ deploy_user }}"
        group: "{{ deploy_user }}"

    - name: Create simple Flask application
      copy:
        content: |
```

CI/CD: Build, Test, Deploy Lab Guide

```
from flask import Flask, jsonify
import os, datetime

app = Flask(__name__)

@app.route('/')
def home():
    return f"""


# Flask Demo Application



Environment: {os.environ.get("ENVIRONMENT", "dev")}



Host: {os.environ.get("HOSTNAME", "unknown")}



Version: {os.environ.get("APP_VERSION", "1.0.0")}



Port: {os.environ.get("APP_PORT", "8080")}


"""

@app.route('/health')
def health():
    return jsonify({
        "status": "healthy",
        "environment": os.environ.get("ENVIRONMENT", "dev"),
        "timestamp": datetime.datetime.now().isoformat(),
        "host": os.environ.get("HOSTNAME", "unknown")
    })

@app.route('/version')
def version():
    return jsonify({
        "app_version": os.environ.get("APP_VERSION", "1.0.0"),
        "environment": os.environ.get("ENVIRONMENT", "dev"),
        "port": os.environ.get("APP_PORT", "8080")
    })

if __name__ == "__main__":
```

```
port = int(os.environ.get("APP_PORT", 8080))
app.run(host="0.0.0.0", port=port, debug=False)
dest: "{{ app_directory }}/app/app.py"
owner: "{{ deploy_user }}"
group: "{{ deploy_user }}"
mode: '0755'
```

```
- name: Create requirements.txt
copy:
content: |
Flask==2.3.3
gunicorn==21.2.0
dest: "{{ app_directory }}/app/requirements.txt"
```

```
- name: Create Python virtual environment
command: python3 -m venv {{ app_directory }}/venv
args:
creates: "{{ app_directory }}/venv/bin/activate"
```

```
- name: Install Python dependencies into venv
pip:
requirements: "{{ app_directory }}/app/requirements.txt"
virtualenv: "{{ app_directory }}/venv"
```

```
- name: Generate application configuration
template:
src: templates/app.conf.j2
dest: "{{ app_directory }}/app.conf"
```

```
- name: Configure Supervisor for application
template:
src: templates/supervisor-app.conf.j2
dest: "/etc/supervisor/conf.d/{{ app_name }}.conf"
```

```
post_tasks:
  - name: Ensure Supervisor is running
    service:
      name: supervisor
      state: started
      enabled: yes

  - name: Reread Supervisor configs
    command: supervisorctl reread

  - name: Update Supervisor with new configs
    command: supervisorctl update

  - name: Start application
    supervisorctl:
      name: "{{ app_name }}"
      state: started

  - name: Wait for application to be ready
    wait_for:
      port: "{{ app_port }}"
      host: "{{ ansible_host }}"
      timeout: 30

  - name: Test application health endpoint
    uri:
      url: "http://{{ ansible_host }}:{{ app_port }}/health"
      method: GET
      status_code: 200

  - name: Display deployment summary
    debug:
```

```

msg: |
Deployment completed successfully!
Application: {{ app_name }}
Version: {{ app_version }}
Environment: {{ app_environment }}
Host: {{ inventory_hostname }}
Port: {{ app_port }}
Health Check: http://{{ ansible_host }}:{{ app_port }}/health
Main Page: http://{{ ansible_host }}:{{ app_port }}/

```

Save the file by pressing **Ctrl+S**

Task 5: Creating application and supervisor configuration template

1. Create application configuration template:

- Click on **templates** folder → **New File**
- Name it **app.conf.j2**
- Add the following Jinja2 template:

```

# Application Configuration for {{ app_name }}
# Environment: {{ app_environment }}
# Generated by Ansible on {{ ansible_date_time.iso8601 }}

```

```

[application]
name = {{ app_name }}
environment = {{ app_environment }}
debug = {{ app_config.debug }}
log_level = {{ app_config.log_level }}
port = {{ app_port }}
workers = {{ app_config.max_workers }}
timeout = {{ app_config.timeout }}

```

```
[logging]
```

```
directory = {{ app_directory }}/logs
rotation_days = {{ log_rotation_days }}
max_size = 100MB
```

```
[monitoring]
enabled = {{ monitoring_enabled }}
health_endpoint = {{ health_check_url }}
```

Save the file by pressing **Ctrl+S**

2. Create Supervisor configuration template:

- Click on **templates** folder → **New File**
- Name it **supervisor-app.conf.j2**
- Add the following template:

```
# Supervisor configuration for {{ app_name }}
# Environment: {{ app_environment }}
```

```
[program:{{ app_name }}]
command={{ app_directory }}/venv/bin/python {{ app_directory }}/app/app.py
directory={{ app_directory }}/app
user={{ deploy_user }}
autostart=true
autorestart=true
startretries=3
redirect_stderr=true
```

```
# Logs
stdout_logfile={{ app_directory }}/logs/{{ app_name }}.out.log
stderr_logfile={{ app_directory }}/logs/{{ app_name }}.err.log
stdout_logfile_maxbytes=50MB
stdout_logfile_backups=5
```

```
# Environment variables
```

```
environment=ENVIRONMENT="{{ app_environment }}",APP_VERSION="{{ app_version }}",
APP_NAME="{{ app_name }}",APP_PORT="{{ app_port }}",HOSTNAME="{{ inventory_hostname }}",
PYTHONPATH="{{ app_directory }}/app"
```

Save the file by pressing **Ctrl+S**

Task 6: Testing and automation

1. Create an automation script:

- Click in **Lab7** folder → New File
- Name it **ansible-deploy.sh**
- Add the following script:

```
#!/bin/bash
# Ansible Deployment Script

set -e

while getopts e:p: flag
do
  case "${flag}" in
    e) ENV=${OPTARG};;
    p) PLAYBOOK=${OPTARG};;
  esac
done

echo "[`date +"%Y-%m-%d %H:%M:%S"`] Ansible Deployment Plan"
echo "===="
echo "Environment: $ENV"
echo "Inventory: $(pwd)/inventories/$ENV.yml"
echo "Playbook: $(pwd)/$PLAYBOOK"
echo "Command: ansible-playbook -i $(pwd)/inventories/$ENV.yml $(pwd)/$PLAYBOOK"
echo "===="

echo "[`date +"%Y-%m-%d %H:%M:%S"`] Running pre-flight checks..."
```

```
echo "[`date +"%Y-%m-%d %H:%M:%S"`] Testing inventory connectivity..."  
ansible -i inventories/$ENV.yml all -m ping
```

```
echo "[`date +"%Y-%m-%d %H:%M:%S"`] Checking playbook syntax..."  
ansible-playbook -i inventories/$ENV.yml $PLAYBOOK --syntax-check  
echo "[SUCCESS] Pre-flight checks completed"
```

```
echo "[`date +"%Y-%m-%d %H:%M:%S"`] Starting Ansible execution..."  
ansible-playbook -i inventories/$ENV.yml $PLAYBOOK
```

Save the file by pressing **Ctrl+S**

In the terminal, make the sh file executable with:

```
chmod +x ansible-deploy.sh
```

2. Create a Makefile for common operations:

- Click in **Lab7** folder → **New File**
- Name it **Makefile** and add the following content:

```
# ======  
# Makefile for Lab07 Deployments  
# ======
```

```
ENV ?= development  
PLAYBOOK = playbooks/deploy-app.yml  
INVENTORY = inventories/$(ENV).yml  
ANSIBLE = ansible-playbook -i $(INVENTORY) $(PLAYBOOK)  
DEPLOY_SCRIPT = ./ansible-deploy.sh
```

```
# Default target  
all: deploy-check
```

```
# -----  
# Deployment targets  
# -----
```

deploy-check:

```
@echo "Running deployment check for ${ENV} environment..."
${DEPLOY_SCRIPT} -e ${ENV} -p ${PLAYBOOK} -c
```

deploy:

```
@echo "Deploying to ${ENV} environment..."
${DEPLOY_SCRIPT} -e ${ENV} -p ${PLAYBOOK}
```

Cleanup target

clean:

```
@echo "Cleaning up logs and supervisor configs..."
# Remove app logs
rm -rf /home/student/deployed-apps/logs/* || true
# Remove ALL supervisor configs for flask-demo
sudo rm -f /etc/supervisor/conf.d/flask-demo*.conf
sudo rm -f /etc/supervisor/conf.d/flask-demo-web*.conf
# Reload Supervisor so it forgets old processes
sudo supervisorctl reread || true
sudo supervisorctl update || true
@echo "Cleanup complete."
```

Helper targets

status:

```
@echo "Checking Supervisor status..."
sudo supervisorctl status
```

logs-web1:

```
@echo "Tailing logs for web1..."
sudo tail -n 40 /home/student/deployed-apps/logs/flask-demo-web1.out.log || true
```

```
sudo tail -n 40 /home/student/deployed-apps/logs/flask-demo-web1.err.log || true
```

logs-web2:

```
@echo "Tailing logs for web2..."
```

```
sudo tail -n 40 /home/student/deployed-apps/logs/flask-demo-web2.out.log || true
```

```
sudo tail -n 40 /home/student/deployed-apps/logs/flask-demo-web2.err.log || true
```

Save the file by pressing **Ctrl+S**

3. Test and Deploy the complete setup:

Test inventory structure

```
ansible-inventory -i inventories/development.yml --list
```

Run a dry-run deployment

```
make deploy-check ENV=development
```

Deploy the application

```
make deploy ENV=development
```

Check service status

```
make status ENV=development
```

```
● student@labuser-virtual-machine:~/Desktop/Lab07$ make status ENV=development
Checking Supervisor status...
sudo supervisorctl status
flask-demo-web1                  RUNNING    pid 1924653, uptime 1:00:15
flask-demo-web2                  RUNNING    pid 1924654, uptime 1:00:15
```

4. Verify deployments manually:

```
# Check if applications are running
```

```
curl http://localhost:8080/
```

```
curl http://localhost:8080/health
```

```
curl http://localhost:8080/version
```



Flask Demo Application

Environment: development

Host: web1

Version: 1.0.0

Port: 8080

```
curl http://localhost:8081/  
curl http://localhost:8081/health  
curl http://localhost:8081/version
```



Check service status

```
sudo supervisorctl status flask-demo-web1  
sudo supervisorctl status flask-demo-web2
```

```
● student@labuser-virtual-machine:~/Desktop/Lab07$ sudo supervisorctl status flask-demo-we  
b1  
flask-demo-web1          RUNNING    pid 1924653, uptime 1:06:06  
● student@labuser-virtual-machine:~/Desktop/Lab07$ sudo supervisorctl status flask-demo-we  
b2  
flask-demo-web2          RUNNING    pid 1924654, uptime 1:06:41
```

5. Stop the apps:

Since apps are managed by Supervisor, you stop them with supervisorctl. Because we gave each app its own unique name (flask-demo-web1, flask-demo-web2), you can stop them individually or together:

- Stop a specific app

```
sudo supervisorctl stop flask-demo-web1
```

```
sudo supervisorctl stop flask-demo-web2
```

- Stop all apps at once

```
sudo supervisorctl stop all
```

- Check their status

```
sudo supervisorctl status
```

You should see something like:

```
flask-demo-web1 STOPPED
```

```
flask-demo-web2 STOPPED
```

- **student@labuser-virtual-machine:~/Desktop/Lab07\$ sudo supervisorctl status**
flask-demo-web1 RUNNING pid 1924653, uptime 1:18:22
flask-demo-web2 RUNNING pid 1924654, uptime 1:18:22
- **student@labuser-virtual-machine:~/Desktop/Lab07\$ sudo supervisorctl stop all**
flask-demo-web1: stopped
flask-demo-web2: stopped
- ✖ **student@labuser-virtual-machine:~/Desktop/Lab07\$ sudo supervisorctl status**
flask-demo-web1 STOPPED Sep 21 05:47 PM
flask-demo-web2 STOPPED Sep 21_05:47 PM

Lab review

1. What is the primary advantage of using Ansible roles over standalone playbooks?
 - A. Roles execute faster than playbooks
 - B. Roles provide reusable, modular automation components
 - C. Roles require fewer system resources
 - D. Roles automatically handle all errors

Answer: B. Roles provide reusable, modular automation components

STOP

You have successfully completed this lab.