

Comprehensive DevOps CI/CD Pipeline Project Report

Title: Automated PHP Application Deployment using Jenkins, Docker, Ansible, and AWS

1. Introduction

1.1 Project Overview

This project demonstrates a fully automated CI/CD pipeline for deploying a PHP web application using DevOps tools:

- Jenkins (CI/CD Automation)
- Docker (Containerization)
- Ansible (Configuration Management)
- AWS EC2 (Cloud Infrastructure)
- Git (Version Control)

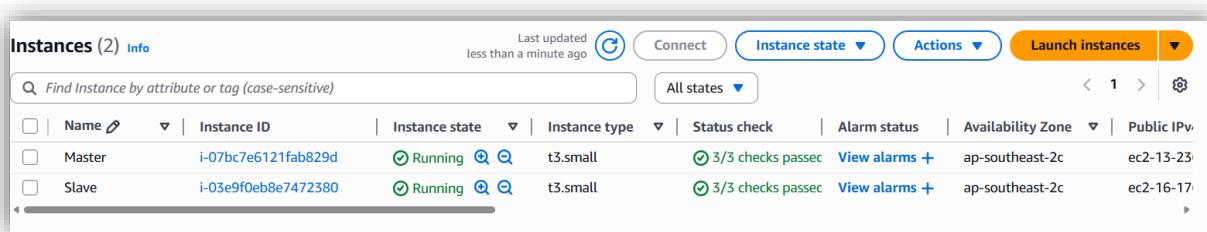
Objective:

- Automate deployment from code commit to production.
- Ensure zero-downtime updates and scalability.
- Implement Infrastructure as Code (IaC) principles.

2. System Architecture

2.1 Infrastructure Setup

Node	IP	Purpose	Tools Installed
Master	13.236.1.136	Jenkins, Ansible, Git	Docker, Java 21
Slave	16.176.147.128	Docker Host (Test/Prod Environment)	Docker, Nginx



2.2 Workflow Diagram

Developer → Git Push → Jenkins (Master) → Ansible → Docker Build → Slave Deployment → Live App (<http://16.176.147.128>)

3. Tools & Technologies

3.1 Jenkins

- Purpose: Automate CI/CD pipeline.
- Key Plugins Used:
 - Build Pipeline Plugin
 - GitHub Integration
 - SSH Agent Plugin

3.2 Docker

- Purpose: Containerize PHP application.
- Key Commands:

```
# Build image
docker build -t php-app .

# Run container
docker run -d -p 80:80 --name php-app php-app
```

3.3 Ansible

- Purpose: Automate configuration management.
- Key Playbook (install-docker.yml):

```
- hosts: slaves
  become: yes
  tasks:
    - name: Install Docker
      apt:
        name: docker.io
        state: present
```

3.4 AWS EC2

- Instance Types:
 - Master: t2.medium (2 vCPUs, 4GB RAM)
 - Slave: t2.micro (1 vCPU, 1GB RAM)
- Security Groups:
 - Allow SSH (22), HTTP (80), Jenkins (8080).

4. Step-by-Step Implementation

4.1 Setting Up Jenkins

1. Install Jenkins on Master:

```
sudo apt update
sudo apt install openjdk-21-jdk
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update
sudo apt install jenkins
sudo systemctl start jenkins
```

2. Access Jenkins:

- Open <http://<Master-IP>:8080>.
- Unlock Jenkins using:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

4.2 Docker Setup

1. Install Docker on Slave:

```
sudo apt update
sudo apt install docker.io
sudo systemctl start docker
sudo usermod -aG docker $USER
```

4.3 Ansible Configuration

1. Install Ansible on Master:

```
sudo apt update  
sudo apt install ansible
```

2. Create Inventory File (hosts):

```
[slaves]  
172.31.26.99 # Slave Private IP
```

3. Test Connectivity:

```
ansible all -m ping -u ubuntu
```

5. Jenkins Pipeline Jobs

5.1 Job 1: Install Docker via Ansible

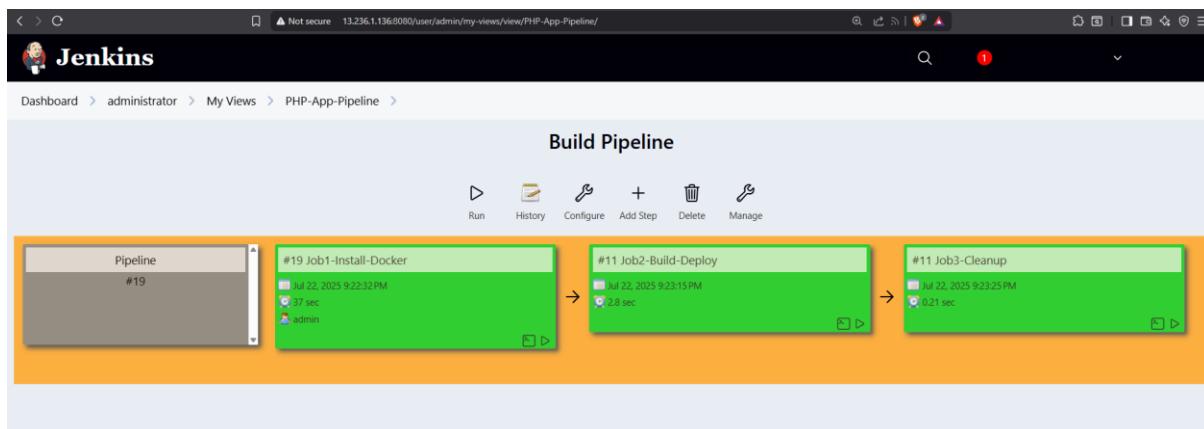
```
cd /var/lib/jenkins/ansible-config  
ansible-playbook -i hosts install-docker.yml -u ubuntu
```

5.2 Job 2: Build & Deploy PHP Container

```
docker build -t php-app .  
docker run -d --name php-app -p 80:80 php-app
```

5.3 Job 3: Cleanup on Failure

```
docker rm -f php-app || true
```

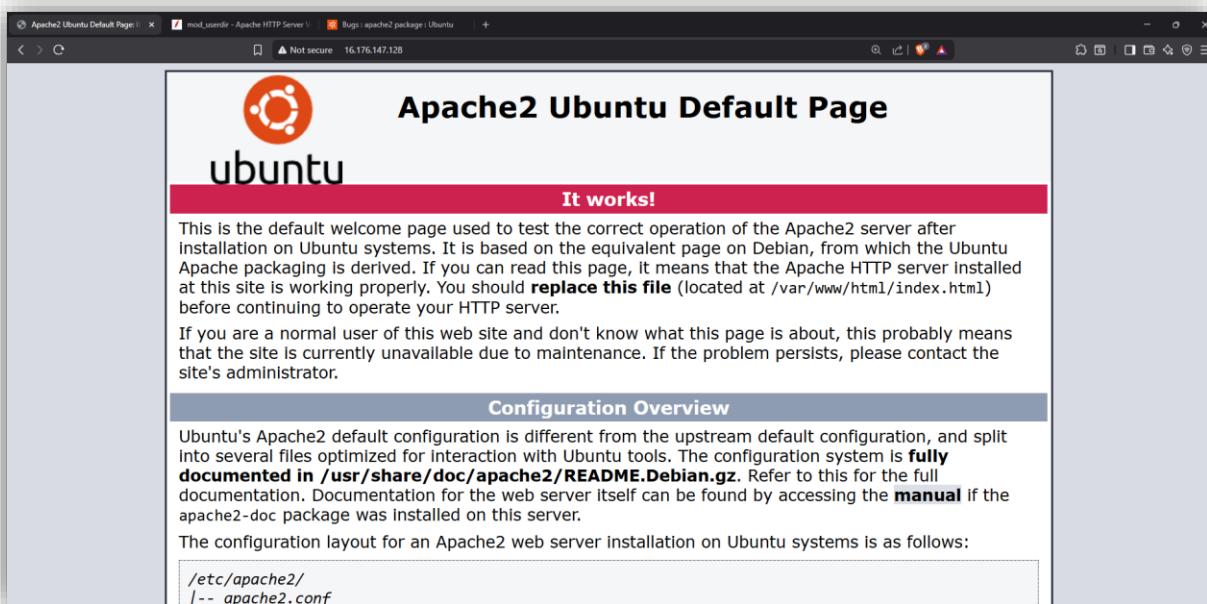


6. Challenges & Solutions

Challenge	Root Cause	Solution
Permission denied on Docker	User not in docker group	sudo usermod -aG docker \$USER
Host key verification failed	SSH strict host checking	Disabled in /etc/ansible/ansible.cfg
Docker package conflicts	Ubuntu 24.04 + containerd issue	Used official Docker CE repo
Site can't be reached	AWS Security Group misconfigured	Allowed inbound HTTP (port 80)

7. Results & Screenshots

- PHP App Live: <http://16.176.147.128>



The screenshot shows a web browser displaying the Apache mod_userdir documentation. The title is 'Apache Module mod_userdir'. The page includes a summary: 'By using this module you are allowing multiple users to host content within the same origin. The same origin policy is a key principle of Javascript and web security. By hosting web pages in the same origin these pages can read and control each other and security issues in one page may affect another. This is particularly dangerous in combination with web pages involving dynamic content and authentication and when your users don't necessarily trust each other.' It also notes: 'This module allows user-specific directories to be accessed using the `http://example.com/~user/` syntax.' A 'UserDir Directive' section is shown with its description: 'Description: Location of the user-specific directories', syntax: 'UserDir directory-filename [directory-filename] ...', context: 'server config, virtual host', status: 'Base', and module: 'mod_userdir'. A note states: 'The UserDir directive sets the real directory in a user's home directory to use when a request for a document for a user is received. Directory-filename is one of the following:'. A list follows: '• The name of a directory or a pattern such as those shown below.', '• The keyword disabled. This turns off all username-to-directory translations except those explicitly named with the enabled keyword (see below).', '• The keyword disabled followed by a space-delimited list of usernames. Usernames that appear in such a list will never have directory translation performed, even if they appear in an enabled clause.', '• The keyword enabled followed by a space-delimited list of usernames. These usernames will have directory translation performed even if a global disable is in effect, but not if they also appear in a disabled clause.' A note at the bottom states: 'If neither the enabled nor the disabled keywords appear in the UserDir directive, the argument is treated as a filename pattern, and is used to turn the name into a directory specification. A request for `http://www.example.com/~bob/one/two.html` will be translated to'. A table shows 'UserDir directive used Translated path': UserDir public_html ~bob/public_html/one/two.html, UserDir userweb /usr/web/bob/one/two.html, UserDir home/www /home/bob/www/one/two.html. A note at the bottom states: 'The following directives will send redirects to the client.'

7.2 Docker Container Status

CONTAINER ID	IMAGE	COMMAND	STATUS	PORTS	NAMES
b440f55a3ec7	php-app	"apache2-foreground"	Up 2m	0.0.0.0:80->80/tcp	php-app

8. Future Enhancements

1. **Kubernetes Integration** – For orchestration.
2. **Terraform** – Automate AWS infrastructure.
3. **Monitoring** – Prometheus + Grafana.
4. **Blue-Green Deployment** – Reduce downtime.

9. Conclusion

This project successfully implemented:

- Automated CI/CD Pipeline**
- Dockerized PHP Application**
- Multi-Server Deployment**
- Infrastructure as Code (IaC)**

GitHub Repo: https://github.com/MdSltn/Project_dev

10. Appendix

10.1 Full Commands Used

```
# Docker Build
docker build -t php-app .

# Ansible Playbook
ansible-playbook -i hosts install-docker.yml

# Jenkins CLI
java -jar jenkins-cli.jar -s http://localhost:8080 build Job1-Install-Docker
```