

# Table of Contents

1. Project Overview
  - 1.1 Objective
  - 1.2 Description
  - 1.3 Technologies Used
2. Pipeline Design
  - 2.1 Overview
  - 2.2 Stages
  - 2.3 Tools Used
  - 2.4 Pipeline Diagram
3. Week-by-Week Breakdown
  - 3.1 Week 1: Initial Setup & Planning 3.1.1 Tasks
  - 3.2 Week 2: Jenkins & CI Integration 3.2.1 Tasks
  - 3.3 Week 3: Docker & Deployment 3.3.1 Tasks
4. Deployment Process
  - 1. Terraform Script for AWS EC2 Provisioning
  - 2. GitHub and Jenkins Integration
  - 3. Jenkins Pipeline Configuration
  - 4. Ansible Playbook for Application Deployment
  - 5. Docker
  - 6. Slack
5. Issues Faced and Resolutions
6. Best Practices and Recommendations
7. Conclusion
8. Final result screenshots

# 1. Project Overview

This project aims to create a Continuous Integration and Continuous Deployment (CI/CD) pipeline that automates the build, test, and deployment processes for a Dockerized web application using Jenkins for orchestration, Docker for containerization, and Ansible for configuration management.

The primary objectives are to:

- Automate the integration of code changes.
- Ensure consistent application deployment.
- Improve code quality through automated testing.

## 1.1 Objective

Implement an automated CI/CD pipeline using Jenkins, Docker, Terraform, and Ansible to facilitate the build, testing, and deployment of a sample application.

## 1.2 Description

The project aims to develop a pipeline that automates various stages of software delivery, ensuring that code changes are integrated and deployed quickly and reliably.

## 1.3 Technologies Used

- Docker: For containerizing the application.
- Terraform: For infrastructure as code .
- Jenkins: For continuous integration and automation.
- Ansible: For configuration management and deployment.
- CI/CD: Continuous Integration/Continuous Deployment principles.
- Cloud Deployment: Deployment to AWS.

## 2. Pipeline Design

### 2.1 Overview

The CI/CD pipeline facilitates a seamless flow from code commit to deployment in a production-like environment, ensuring that each change is validated before going live.

### 2.2 Stages

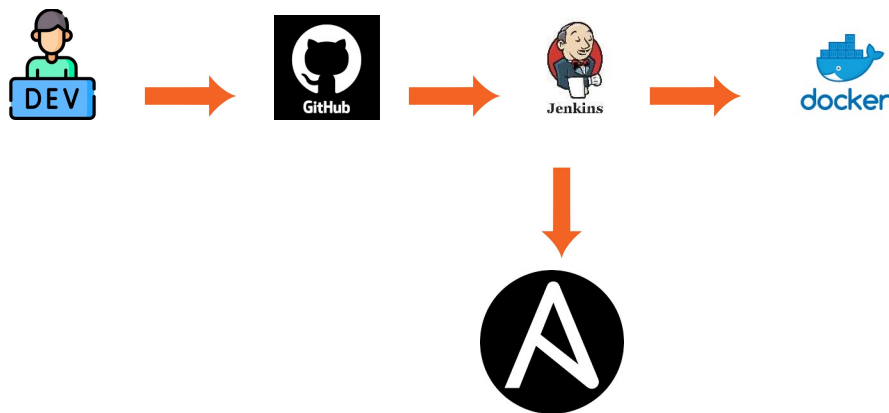
The pipeline consists of the following stages:

- Source: Code is committed to a Git repository (GitHub or GitLab).
- Build:
  - Jenkins pulls the latest code.
  - The application is built into a Docker image using a `Dockerfile`.
- Test:
  - Automated tests are executed within the Docker container.
  - Test results are reported back to Jenkins.
- Deploy:
  - If tests pass, the Docker image is pushed to Docker Hub or a private registry.
  - Ansible playbooks are executed to deploy the application to a cloud environment.

## 2.3 Tools Used

- Jenkins: Automation server for orchestrating the CI/CD pipeline.
- Docker: Tool for creating and managing containers.
- Terraform: Infrastructure as code tool .
- Ansible: Automation tool for configuration management and deployment.
- GitHub: Version control systems for managing source code.
- Slack: Notification systems for build status updates.

## 2.4 Pipeline Diagram



## 3. Week-by-Week Breakdown

### 3.1 Week 1: Initial Setup & Planning

#### 3.1.1 Tasks

1. Install Jenkins and Docker
  - Set up the Jenkins server and install Docker on local or cloud environments.
2. Create a Basic Dockerized Application
  - Develop a simple web application (e.g., Flask app) and create a `Dockerfile` to containerize it.
3. Set Up Ansible
  - Install Ansible for configuration management.
4. Pipeline Plan
  - Plan the structure of the CI/CD pipeline (build, test, deploy) and document tasks.

#### 3.1.2 Deliverables

- Jenkins and Docker environments installed and configured.
- Dockerized application running locally.
- Basic Ansible setup completed.
- CI/CD pipeline plan documented.

### 3.2 Week 2: Jenkins & CI Integration

#### 3.2.1 Tasks

1. Create Jenkins Jobs
  - Configure Jenkins to build the Dockerized application.
2. Integrate Git
  - Set up a GitHub or GitLab repository and integrate it with Jenkins for continuous integration.
3. Automated Testing
  - Add basic automated testing (e.g., unit tests) into the Jenkins pipeline.
4. Set Up Notifications
  - Configure email or Slack notifications for pipeline success or failure.

### 3.2.2 Deliverables

- A working Jenkins pipeline triggered by Git commits.
- A Jenkins job that builds Docker images successfully.
- Automated testing included in the pipeline.
- Notifications configured for pipeline updates.

## 3.3 Week 3: Docker & Deployment

### 3.3.1 Tasks

1. Integrate Docker Hub or Private Registry
  - Push Docker images to Docker Hub or a private registry.
2. Configure Ansible for Deployment
  - Write Ansible playbooks to automate application deployment to a cloud environment (e.g., AWS, GCP).
3. Deployment Testing
  - Test the deployment process with Docker and Ansible to ensure smooth automation.
4. Refine the Pipeline
  - Refine Jenkins jobs for efficiency (e.g., parallel stages, caching).
5. Full Testing and Deployment
  - Conduct full tests of the CI/CD pipeline from code commit to deployment.

### 3.3.2 Deliverables

- Docker images automatically pushed to Docker Hub or a registry.
- Ansible playbooks ready and tested.
- Successful deployment of the application to a cloud server.
- Refined Jenkins jobs that improve efficiency.

## 4 Deployment Process

### 1. Terraform Script for AWS EC2 Provisioning

The Terraform script will provision the necessary AWS EC2 instances to host your application. Below is an example of how to create a Terraform configuration file (`main.tf`):

```
main.tf
10
11 provider "aws" {
12   region = "us-east-1"
13   access_key = ""
14   secret_key = ""
15 }
16
17
18
19 resource "aws_vpc" "my-vpc" {
20   cidr_block = var.vpc_cidr_block
21   tags = {
22
23     Name = "${var.envi}-vpc"
24   }
25 }
26
27 module "myapp-subnet" {
28
29   source = "./modules/subnet"
30   vpc_id = aws_vpc.my-vpc.id
31   subnet_cidr_block = var.subnet_cidr_block
32   avail_zone= var.avail_zone
33   envi = var.envi
34 }
35 module "myapp-server"{
36   source = "./modules/server"
37   vpc_id = aws_vpc.my-vpc.id
38   my_ip = var.my_ip
39   envi = var.envi
```

The image shows a VS Code editor window with a file explorer on the left and a terminal at the bottom. The file explorer shows a project named 'DEPL\_FINAL\_PROJECT (WSL...)' with a directory structure including 'terraform', 'app', 'modules', 'scripts', and 'variables'. The 'main.tf' file is open in the editor, showing a Terraform configuration for a Jenkins server. The configuration includes a 'jenkins\_server' module and an 'aws\_vpc' resource. The terminal shows the output of a Terraform command, indicating that the resources were created successfully.

```
1
2
3 #jenkins_server
4 module "jenkins_server" {
5   source = "../modules/jenkins_server"
6   jenkins_cidr_blocks = var.jenkins_cidr_blocks
7   jenkins_availability_zone = var.jenkins_availability_zone
8   jenkins_instance_type = var.jenkins_instance_type
9   jenkins_vpc_id = aws_vpc.jenkins_vpc.id
10 }
11
12 module "jenkins_server" {
13   source = "../modules/jenkins_server"
14   my_ip = var.my_ip
15   public_key_location = var.public_key_location
16   jenkins_instance_type = var.jenkins_instance_type
17   jenkins_availability_zone = var.jenkins_availability_zone
18   jenkins_subnet_id = module.jenkins_subnet.jenkins_subnet_id
19   jenkins_vpc_id = aws_vpc.jenkins_vpc.id
20 }
21
22 resource "aws_vpc" "jenkins_vpc" {
23   cidr_block = var.jenkins_cidr_blocks[0].cidr_block
24   tags = {
25     Name = var.jenkins_cidr_blocks[0].name
26   }
27 }
28 }
```

module.jenkins\_subnet.aws\_route\_table\_association.rtb\_subnet: Creating...  
module.app\_subnet.aws\_route\_table\_association.rtb\_subnet: Creating...  
module.app\_server.aws\_security\_group.sg: Creation complete after 4s [id=sg-07480274ad567a962]  
module.jenkins\_server.aws\_security\_group.sg: Creation complete after 4s [id=sg-01333042d1ef4c46]  
module.app\_server.aws\_instance.app\_server: Creating...  
module.jenkins\_server.aws\_instance.jenkins\_server: Creating...  
module.app\_subnet.aws\_route\_table\_association.rtb\_subnet: Creation complete after 1s [id=rtbassoc-0cd8f8e74cd22a4]  
module.jenkins\_subnet.aws\_route\_table\_association.rtb\_subnet: Creation complete after 1s [id=rtbassoc-02a084e20f4640e]  
module.app\_server.aws\_instance.app\_server: Still creating... [10s elapsed]  
module.jenkins\_server.aws\_instance.jenkins\_server: Still creating... [10s elapsed]  
module.app\_server.aws\_instance.app\_server: Creation complete after 15s [id=i-0288605e01913804b]  
module.jenkins\_server.aws\_instance.jenkins\_server: Creation complete after 15s [id=i-0025621e8788dabed]

Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:  
app\_public\_ip = "107.21.140.101"  
jenkins\_public\_ip = "54.203.68.177"



## 2. GitHub and Jenkins Integration

The integration of GitHub with Jenkins is a key component of a modern CI/CD pipeline. This setup allows developers to automatically trigger builds, tests, and deployments when changes are made to the source code repository hosted on GitHub. The integration ensures seamless collaboration, continuous integration, and faster delivery of applications.

depi\_final\_project

Public

Watch 1

Fork 0

Star 0

main

1 Branch

Tags

Go to file

Add file

Code

About

Shazly test webhook

7d2e304 · 1 hour ago

5 Commits

app	first commit	3 hours ago
modules	first commit	3 hours ago
.gitignore	first commit	3 hours ago
.terraform.lock.hcl	first commit	3 hours ago
Dockerfile	test webhook	1 hour ago
Jenkinsfile	update app_server ip	1 hour ago
ansible.cfg	first commit	3 hours ago
docker-compose.yaml	first commit	3 hours ago
hosts	update app_server ip	1 hour ago
main.tf	first commit	3 hours ago
outputs.tf	first commit	3 hours ago
playbook.yaml	first commit	3 hours ago
providers.tf	first commit	3 hours ago
script_app.sh	first commit	3 hours ago
script_jenkins.sh	first commit	3 hours ago
terraform.tfvars	first commit	3 hours ago
variables.tf	first commit	3 hours ago

No description, website, or topics provided.

Activity

0 stars

1 watching

0 forks

Report repository

Releases

No releases published

Packages

No packages published

Languages

HCL 47.0%

HTML 22.8%

JavaScript 18.8%

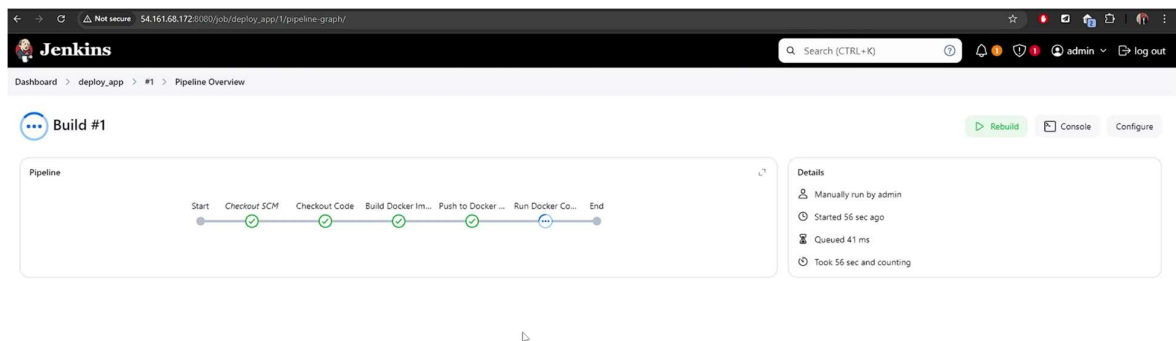
Shell 8.0%

Dockerfile 3.4%

### 3. Jenkins Pipeline Configuration

The Jenkins pipeline will orchestrate the entire deployment process by invoking Terraform to provision infrastructure and Ansible to configure it. Create a Jenkins pipeline script:

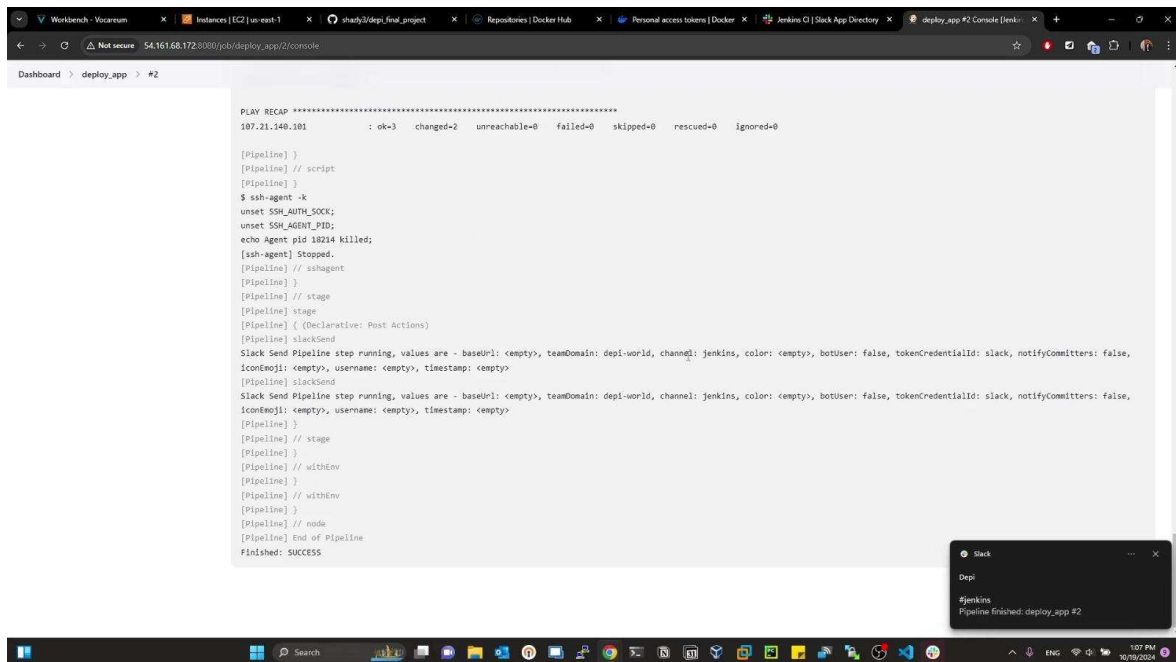
```
Jenkinsfile
1 pipeline {
2   agent any
3   environment {
4     DOCKER_CREDENTIALS_ID = 'docker_credentials'
5     DOCKER_IMAGE_NAME = 'shazly3/webapp'
6     ANSIBLE_INVENTORY = '/home/ec2-user/' // Update this path
7   }
8   stages {
9     stage('Checkout Code') {
10      steps {
11        git url: 'https://github.com/shazly3/depi.git', branch: 'main'
12      }
13    }
14    stage('Build Docker Image') {
15      steps {
16        script {
17          app = docker.build(DOCKER_IMAGE_NAME)
18        }
19      }
20    }
21    stage('Push to Docker Hub') {
22      steps {
23        script {
24          docker.withRegistry('https://index.docker.io/v1/', DOCKER_CREDENTIALS_ID) {
25            app.push('latest')
26          }
27        }
28      }
29    }
30    stage('Run Docker Compose with Ansible') {
```



## 4. Ansible Playbook for Application Deployment

The Ansible playbook will handle the installation of Docker and deployment of your application on the provisioned EC2 instances. Create a file named `playbook.yml`:

```
! playbook.yml
1  ---
2  - hosts: all
3    tasks:
4
5      - name: Copy docker-compose.yaml to the remote machine
6        copy:
7          src: .
8          dest: /home/ec2-user/docker-compose.yaml
9
10     - name: Run Docker Compose
11       command: docker-compose up -d
12       args:
13         chdir: /home/ec2-user/docker-compose.yaml # Update this with the correct path
14
```



```
PLAY RECAP *****
107.21.148.101      : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[Pipeline] }
[Pipeline] // script
[Pipeline] }
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 10214 killed;
[ssh-agent] Stopped.
[Pipeline] // sshagent
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] slackSend
Slack Send Pipeline step running, values are - baseUrl: <empty>, teamDomain: depi-world, channel: jenkins, color: <empty>, botUser: false, tokenCredentialId: slack, notifyCommitters: false,
iconEmoji: <empty>, username: <empty>, timestamp: <empty>
[Pipeline] slackSend
Slack Send Pipeline step running, values are - baseUrl: <empty>, teamDomain: depi-world, channel: jenkins, color: <empty>, botUser: false, tokenCredentialId: slack, notifyCommitters: false,
iconEmoji: <empty>, username: <empty>, timestamp: <empty>
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

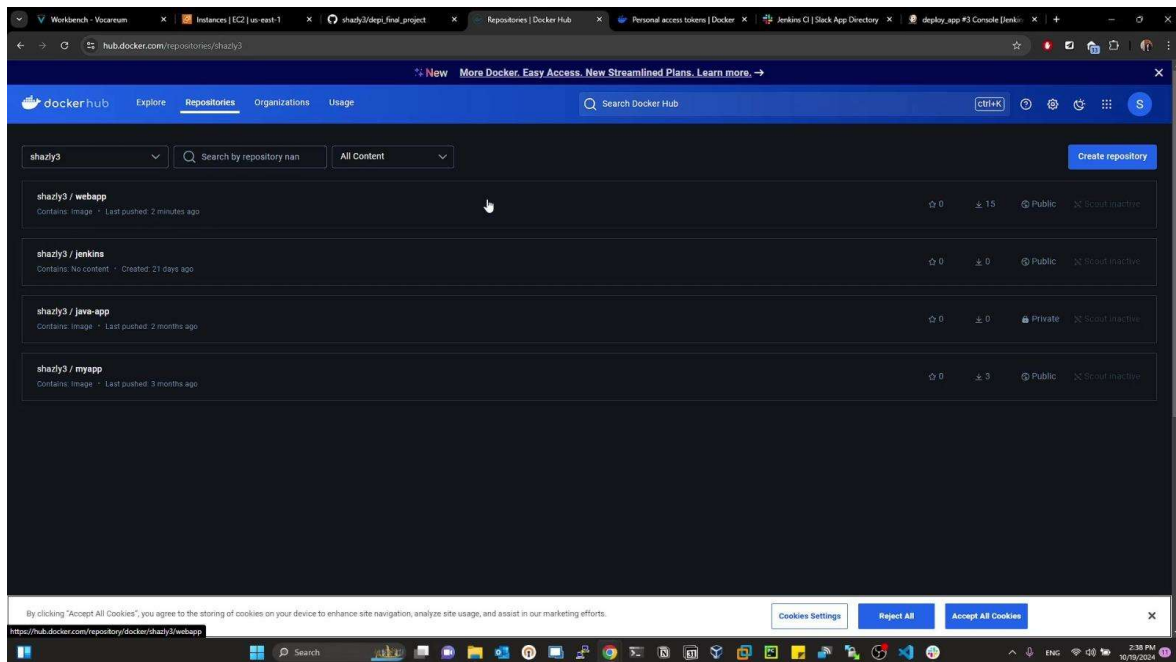
Slack  
depi  
#jenkins  
Pipeline finished: deploy\_app #2

## 5. Docker

Docker provides a powerful platform for building, packaging, and deploying applications in containers. Docker Compose further simplifies managing multi-container Docker applications by using a YAML configuration file to define and manage multiple services.

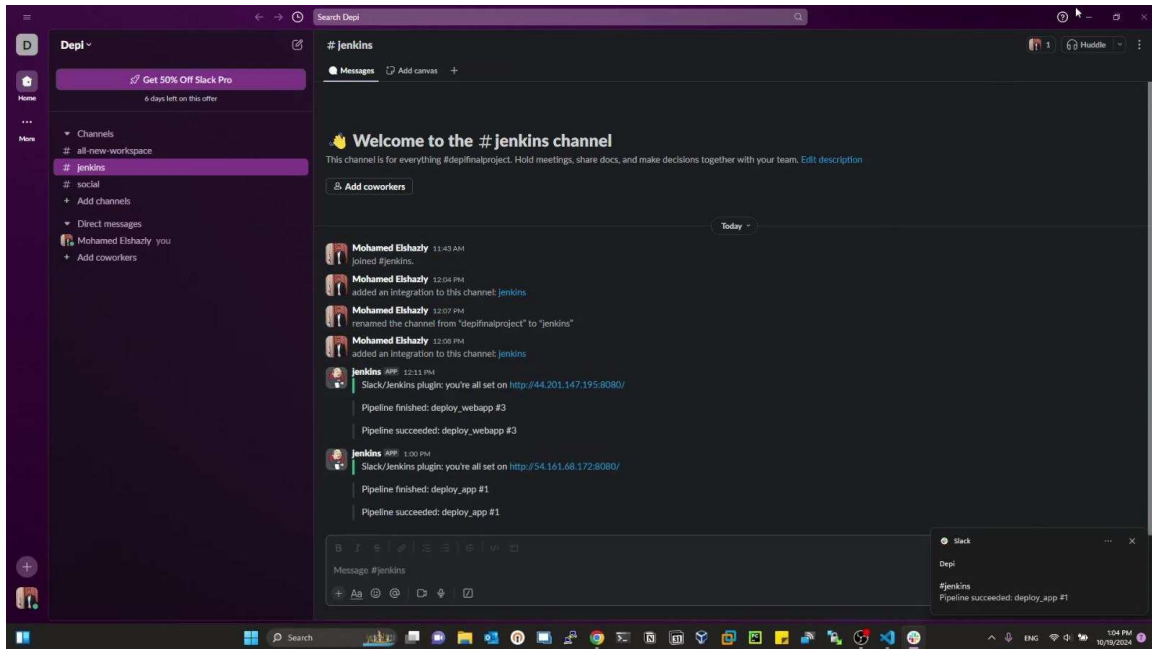
### Docker Image Building

A Docker image is a lightweight, standalone, and executable package that includes everything needed to run an application: the code, runtime, libraries, and system dependencies. The image serves as a blueprint for creating Docker containers.



## 6.Slack

Integrating Jenkins with Slack allows you to send build notifications directly to your team's Slack channels. This can help keep your team informed about the status of CI/CD pipelines, whether they succeed or fail, without leaving Slack.



## 4.2.4 Testing Deployment

After executing the playbook, verify that the application is running by accessing it via a web browser

### Issues Faced and Resolutions

#### Issue #1: Jenkins Connection Timeout

Description: Connection timeouts occurred when Jenkins tried to connect to Docker.

Resolution: Ensured that Docker was running and accessible via port 4243, adjusted firewall settings, and verified that correct permissions were set on the Docker socket.

#### Issue #2: Failed Tests in Pipeline

Description: Unit tests failed during automated testing due to missing dependencies.

Resolution: Reviewed test logs in Jenkins, updated requirements.txt, and re-ran tests until all passed successfully.

#### Issue #3: Ansible Playbook Errors

Description: Encountered errors related to SSH access when executing Ansible playbooks.

Resolution: Verified inventory file paths, ensured SSH keys were correctly configured for remote access, and tested connectivity using ansible ping.

### Best Practices and Recommendations

- Use version tagging for your Docker images to avoid conflicts during deployments.
- Regularly update dependencies in your application and ensure that your testing framework covers edge cases.
- Implement security best practices by scanning your Docker images for vulnerabilities before deploying them.
- Set up monitoring tools (like Prometheus or Grafana) to observe application performance post-deployment.

# Conclusion

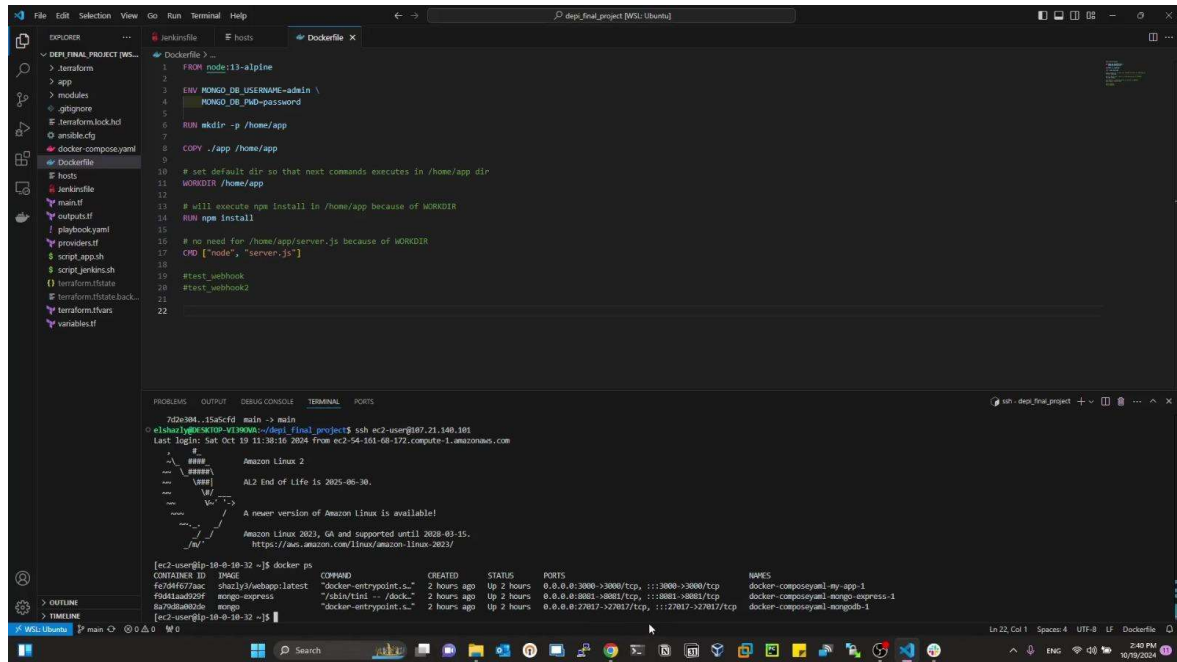
In this documentation project, we successfully leveraged Terraform to automate the provisioning of EC2 instances and configure essential tools such as Jenkins, Ansible, and Docker. By integrating these technologies, we established a robust CI/CD pipeline that not only builds Docker images from our GitHub repository but also deploys applications seamlessly.

The use of Jenkins as the automation server enabled us to streamline our build process, while Ansible played a crucial role in configuration management and application deployment on the app server. This setup not only enhances efficiency but also reduces manual intervention, minimizing the risk of human error.

Furthermore, the integration with GitHub allows for automated triggers, ensuring that code changes are continuously deployed. The addition of Slack notifications provides real-time updates on the project's status, enhancing communication and collaboration among team members.

Overall, this project exemplifies how Infrastructure as Code (IaC) and automation can transform development workflows, leading to faster delivery and improved reliability of applications. The foundation we've built can be easily expanded and adapted to meet future needs, ensuring that our deployment processes remain agile and efficient.

## 9. Final result



```
1 FROM node:13-alpine
2
3 ENV MONGO_DB_USERNAME=admin \
4     MONGO_DB_PASSWORD=password
5
6 RUN mkdir -p /home/app
7
8 COPY . /home/app
9
10 # set default dir so that next commands executes in /home/app dir
11 WORKDIR /home/app
12
13 # will execute npm install in /home/app because of WORKDIR
14 RUN npm install
15
16 # no need for /home/app/server.js because of WORKDIR
17 CMD ["node", "server.js"]
18
19 #Test_webhook
20 #Test_webhook2
21
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ec2-user@ip-10-0-10-32 ~\$ docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fo7d467aac	shazly/webapp:latest	"docker-entrypoint.s..."	2 hours ago	Up 2 hours	0.0.0.0:3000->3000/tcp, :::3000->3000/tcp	docker-compose.yml-my-app-1
8a7d8a803de	mongo-express	"yaktiv/tini -- /dove..."	2 hours ago	Up 2 hours	0.0.0.0:8081->8081/tcp, :::8081->8081/tcp	docker-compose.yml-mongo-express-1
8a7d8a803de	mongo	"docker-entrypoint.s..."	2 hours ago	Up 2 hours	0.0.0.0:27017->27017/tcp, :::27017->27017/tcp	docker-compose.yml-mongodb-1

Ln 22, Col 1 | Spaces: 4 | UTF-8 | LF | Dockerfile



Workbench - Visual Studio Code

Instances | EC2 | us-east-1

shady3/dept\_final\_prj

Repositories | Docker

Personal access tokens

Ansible CI | Slack App Dev

deploy\_app #3 Console

107.21.140.101:3000

Home - Mongo Express

Mongo Express Database

Databases

Database Name

Create Database

View

admin

Del

View

config

Del

View

local

Del

View

user-account

Del

Server Status

Hostname

8a7908a002de

MongoDB Version

8.0.1

Uptime

5736 seconds

Node Version

18.20.3

Server Time

Sat, 19 Oct 2024 11:39:26 GMT

V8 Version

10.2.154.26-node.37

Current Connections

3

Available Connections

25211

Active Clients

0

Queued Operations

0

Clients Reading

0

Clients Writing

0

Read Lock Queue

0

Write Lock Queue

0

Disk Flashes

ms

Last Flush

ms

Time Spent Flushing

ms

Average Flush Time

ms

Workbench - Visual Studio Code

Instances | EC2 | us-east-1

shady3/dept\_final\_prj

Repositories | Docker Hub

Personal access tokens

Ansible CI | Slack App Dev

deploy\_app #3 Console

107.21.140.101:3000

User profile



Name: Tom Cat

Email: tom.cat@example.com

Interests: Jerry

Edit Profile