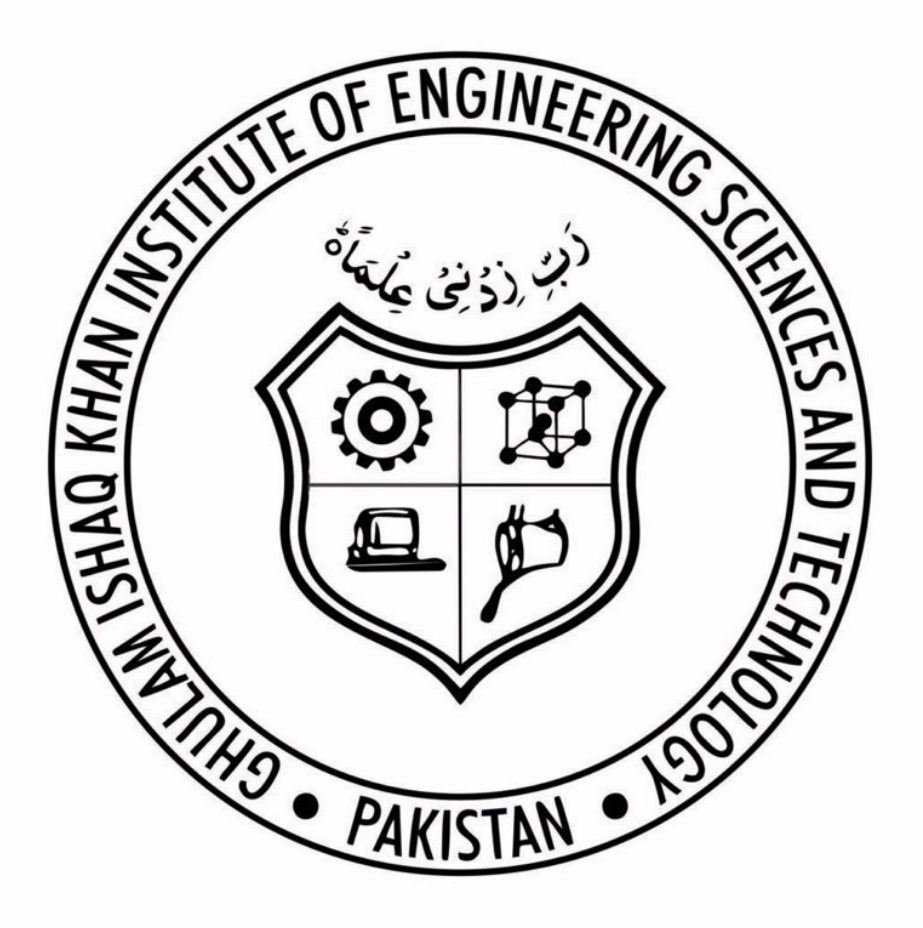
Ghulam Ishaq Khan Institute of Engineering Sciences and Technology

DevOps _CS – 328_Spring - 2025



By

Ayesha Kashif – 2022132

Noor ul Ain – 2022485

Submitted to:

Muhammad Ahmad Nawaz

Microservices Deployment on AWS EKS with Jenkins CI/CD

Introduction

This report documents the development and deployment of a microservices-based architecture using AWS EKS (Elastic Kubernetes Service), Docker, Terraform, and Jenkins. The system comprises multiple services—User Service, Product Service, and Order Service—designed to run as independently deployable units. The aim is to automate the build, test, and deployment processes using modern DevOps practices to ensure scalability, maintainability, and operational efficiency.

Project Purpose

The purpose of this project is to:

- Utilize AWS EKS for Kubernetes-based container orchestration.
- Employ Docker to containerize microservices.
- Use Terraform to provision and manage AWS infrastructure as code.
- Implement Jenkins to manage the CI/CD pipeline for seamless integration and deployment.
- Deploy the services to Kubernetes, enabling scalable and fault-tolerant applications.

Project Architecture

The architecture is composed of three microservices:

- **User Service**
- **Product Service**
- **Order Service**

Each service is containerized with Docker, deployed via Kubernetes on an AWS EKS cluster, and managed through a Jenkins CI/CD pipeline.

Project Structure

```
project-root/
├── jenkins/
│   └── Jenkinsfile
├── services/
│   ├── user-service/
│   ├── product-service/
│   └── order-service/
├── kubernetes/
│   ├── user-deployment.yml
│   ├── product-deployment.yml
│   ├── order-deployment.yml
│   ├── service-user.yml
│   ├── service-product.yml
│   └── service-order.yml
├── terraform/
│   ├── vpc.tf
│   ├── eks-cluster.tf
│   └── iam.tf
```

Infrastructure Provisioning (Terraform)

1. VPC Configuration

Defined in `vpc.tf`, this component sets up:

- A Virtual Private Cloud
- Public and private subnets
- NAT Gateway
- DNS hostnames and resolution

Screenshot Required: Terraform configuration and successful `terraform apply` output.

2. IAM Role Creation

Defined in `iam.tf`, IAM roles provide required permissions for:

- EKS Cluster
- Node Groups
- Service Accounts

Screenshot Required: IAM console showing created roles.

3. EKS Cluster Setup

Defined in `eks-cluster.tf`, the EKS module provisions:

- Kubernetes Control Plane
- Managed Node Groups

Containerization with Docker

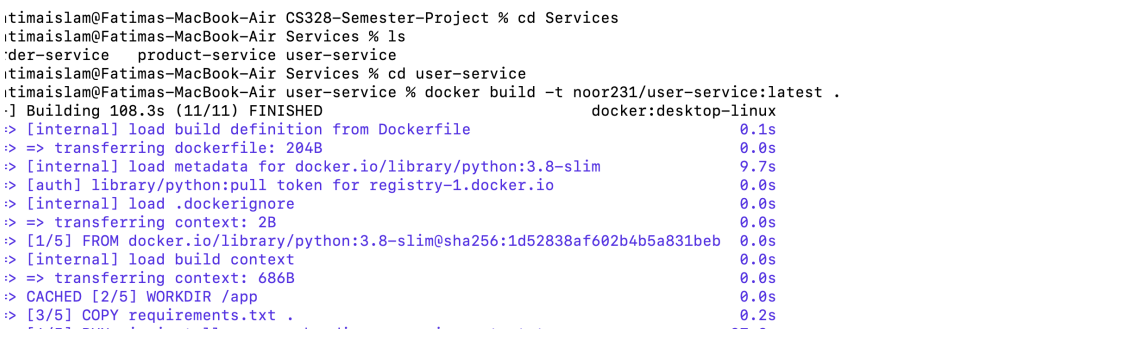
Each microservice has its own Dockerfile that defines:

- The base image
- Copy and installation instructions
- Commands to run the service

Build Command Example:

```
cd services/user-service
docker build -t <dockerhub-username>/user-service:latest .
```

Screenshot Required: Docker build logs and image list.



```
fatimaislam@Fatimas-MacBook-Air user-service % docker build -t noor231/user-service:latest .
[+] Building 108.3s (11/11) FINISHED                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                              0.1s
 => => transferring dockerfile: 204B                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                9.7s
 => [auth] library/python:pull token for registry-1.docker.io                     0.0s
 => [internal] load .dockerignore                                                 0.0s
 => => transferring context: 2B                                                   0.0s
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb   0.0s
 => [internal] load build context                                                 0.0s
 => => transferring context: 686B                                                 0.0s
 => CACHED [2/5] WORKDIR /app                                                     0.0s
 => [3/5] COPY requirements.txt .                                                 0.2s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                     97.8s
 => [5/5] COPY app.py .                                                           0.0s
```

```
fatimaislam@Fatimas-MacBook-Air user-service % docker build -t noor231/user-service:latest .
[+] Building 108.3s (11/11) FINISHED                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                              0.1s
 => => transferring dockerfile: 204B                                              0.0s
 => [internal] load metadata for docker.io/library/python:3.8-slim                9.7s
 => [auth] library/python:pull token for registry-1.docker.io                     0.0s
 => [internal] load .dockerignore                                                 0.0s
 => => transferring context: 2B                                                   0.0s
 => [1/5] FROM docker.io/library/python:3.8-slim@sha256:1d52838af602b4b5a831beb   0.0s
 => [internal] load build context                                                 0.0s
 => => transferring context: 686B                                                 0.0s
 => CACHED [2/5] WORKDIR /app                                                     0.0s
 => [3/5] COPY requirements.txt .                                                 0.2s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                     97.8s
 => [5/5] COPY app.py .                                                           0.0s
```

CI/CD Pipeline Using Jenkins

Jenkins Setup

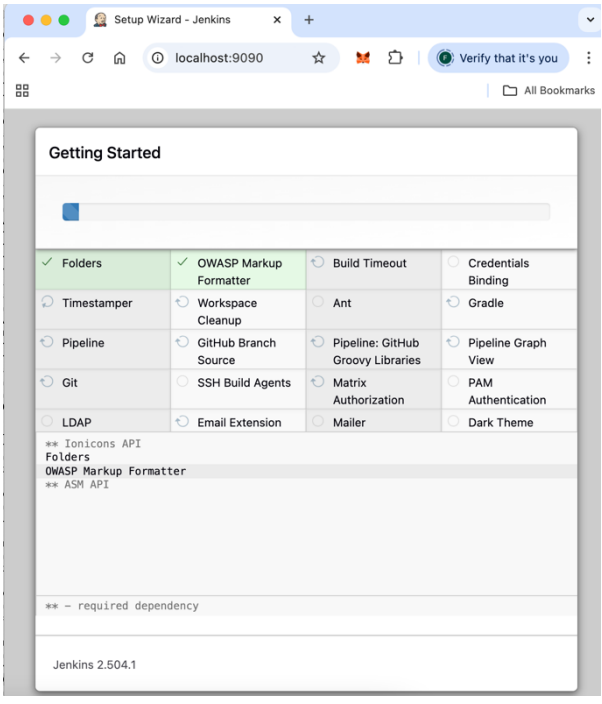
Jenkins is launched via Docker:

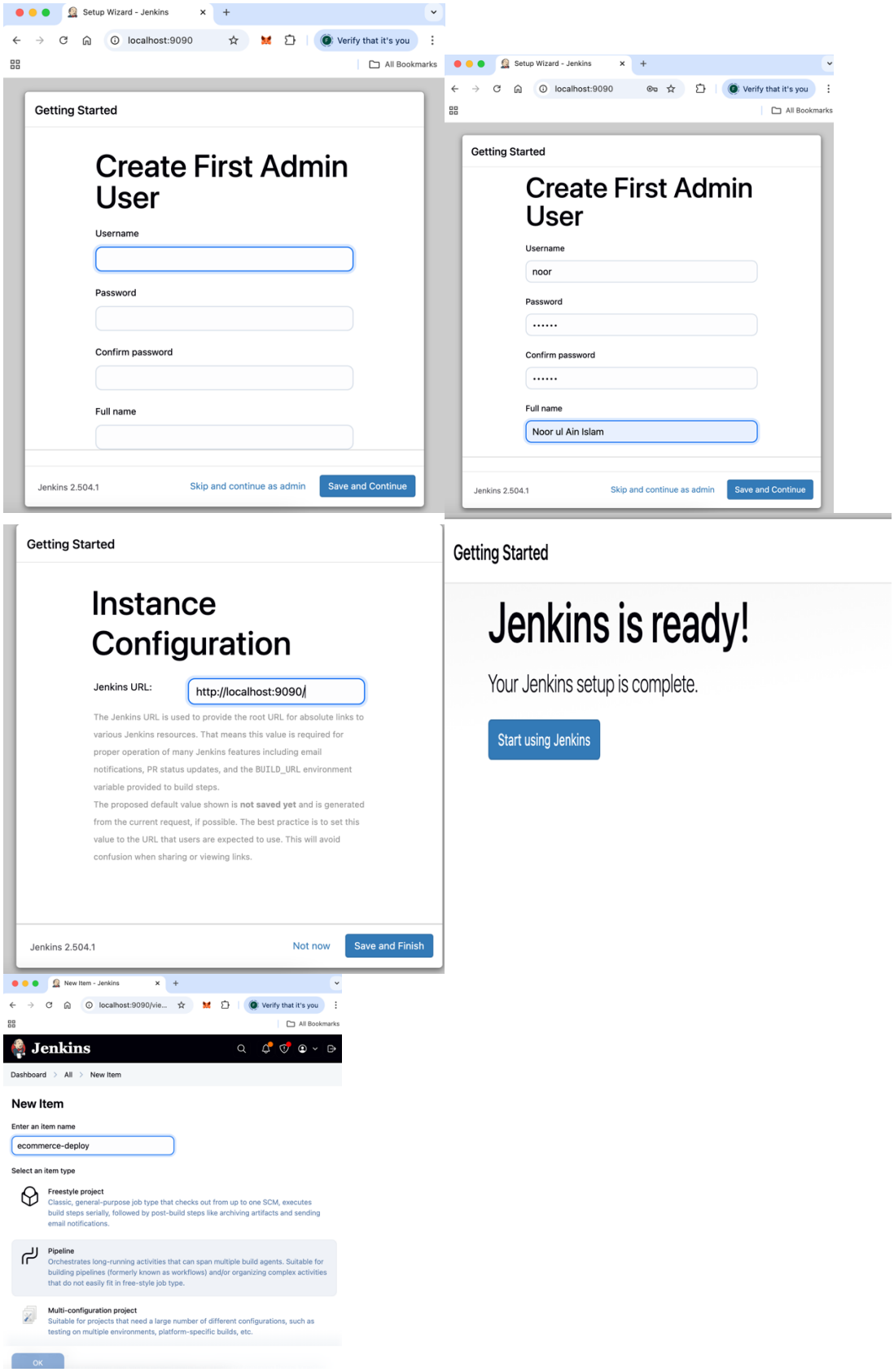
```
docker run -p 9090:9090 -p 50000:50000 jenkins/jenkins:lts
```

Access URL: http://localhost:9090

Screenshot Required: Jenkins dashboard after initial setup.

Jenkinsfile Overview





The pipeline includes:

1. **Build Services** – Docker build for each microservice
2. **Login to DockerHub** – Authenticate with DockerHub credentials

3. **Push Images** – Push built images to DockerHub
4. **Deploy to Kubernetes** – Apply Kubernetes YAML configurations
5. **Post Cleanup** – Logout from DockerHub

Kubernetes Deployment

Configuration Files

Deployment and service YAML files for each service are defined in the `kubernetes/` folder:

- `<service>-deployment.yml`
- `service-<service>.yml`

Deployment Command:

```
kubectl apply -f kubernetes/ --validate=false
```

Screenshot Required:

- Output of `kubectl get pods`
- Output of `kubectl get svc`

Application Verification

Validation Commands

```
kubectl get pods
kubectl get svc
```

Use `kubectl port-forward` or Ingress to access services externally.

```
fatimaislam@Fatimas-MacBook-Air kubernetes % aws sts get-caller-identity
{
    "UserId": "AIDAQEI6UDZ76SLJBGRXM",
    "Account": "009191038591",
    "Arn": "arn:aws:iam::009191038591:user/Admin_1"
}
fatimaislam@Fatimas-MacBook-Air kubernetes % aws configure get region
us-east-1
fatimaislam@Fatimas-MacBook-Air kubernetes % aws eks update-kubeconfig --region us-east-1 --name my-cluster
Updated context arn:aws:eks:us-east-1:009191038591:cluster/my-cluster in /Users/fatimaislam/.kube/config
```

```
fatimaislam@Fatimas-MacBook-Air kubernetes % kubectl apply -f user-deployment.yml --validate=false
deployment.apps/user-deployment created
fatimaislam@Fatimas-MacBook-Air kubernetes % LS
order-deployment.yml    service-order.yml       service-user.yml
product-deployment.yml  service-product.yml     user-deployment.yml
fatimaislam@Fatimas-MacBook-Air kubernetes % kubectl apply -f . --validate=false
deployment.apps/order-deployment created
deployment.apps/product-deployment created
service/order-service created
service/product-service created
```

Troubleshooting

- **Kubernetes API timeout**: Ensure VPC/subnet configurations and endpoint access are correctly set
- **Docker Port Conflicts**: Use `docker ps` and `docker stop <container-id>` to resolve conflicts
- **Jenkins Unreachable**: Verify ports and Docker container health
- **DockerHub Login Failure**: Check and configure credentials securely in Jenkins

Conclusion

This project provides a full CI/CD pipeline with Terraform-based infrastructure setup, Docker-based containerization, Jenkins for pipeline automation, and Kubernetes via AWS EKS for deployment and orchestration. Each tool plays a key role in achieving reliable, scalable, and maintainable microservice deployments.