DevOps Final Project - Submission Document

📋 Student Information

Student Name: Gazal Agbaria **Student ID:** 205474414 **Submission Date:** January 30, 2026

🎯 Project Overview

This project demonstrates a complete DevOps pipeline implementing:

- **Containerization** using Docker
- **Continuous Integration/Continuous Deployment (CI/CD)** using GitHub Actions
- **Cloud Deployment** on AWS EC2
- **REST API** development with Flask-RESTPlus
- **Professional UI/UX** with custom Swagger interface

🔗 Project Links

1. GitHub Repository

```
https://github.com/gazal1994/final-project-DEVOPS
```

Contains:

- Complete source code
- Docker configuration files
- CI/CD workflow definitions
- Comprehensive documentation
- Test reports

2. Docker Hub Image

```
https://hub.docker.com/r/gazal94/final-python-app
```

Features:

- Automated builds from GitHub
- Latest version: `latest` tag
- Optimized Python 3.9-slim base image
- Pull command: `docker pull gazal94/final-python-app:latest`

3. Live Application (Production)

```
http://13.60.8.113/api/doc
```

Deployed on:

- Platform: AWS EC2
- Instance Type: t3.micro
- Region: eu-north-1 (Stockholm)
- IP Address: 13.60.8.113

📊 Project Parts Summary

PART A - Dockerization ✅

Objective: Containerize the Flask REST API application using Docker.

Implementation:

- Created optimized `Dockerfile` with multi-stage approach
- Used Python 3.9-slim as base image for minimal size
- Configured proper working directory and dependencies
- Exposed port 5000 for application access
- Successfully tested local build and execution

Key Files:

- `Dockerfile` - Container configuration
- `requirements.txt` - Python dependencies
- `app.py` - Main application entry point

Status: ✅ **Completed Successfully**

Verification:

```
docker build -t final-python-app .
docker run -p 5000:5000 final-python-app
# Result: Container runs successfully, app accessible on localhost:5000
```

PART B - CI/CD Pipeline to DockerHub ✅

Objective: Implement automated build and deployment pipeline to Docker Hub.

Implementation:

- Created GitHub Actions workflow: `.github/workflows/docker-build-push.yml`
- Configured automatic triggers on push to main branch
- Integrated DockerHub authentication using GitHub Secrets

- Automated image building and pushing process
- Implemented proper tagging strategy (latest)

GitHub Secrets Configured:

- `DOCKERHUB_USERNAME` - Docker Hub account username
- `DOCKERHUB_TOKEN` - Docker Hub access token

Workflow Steps:

1. Checkout code from repository
2. Login to Docker Hub
3. Build Docker image
4. Tag image with version
5. Push to Docker Hub registry

Status: ✅ Completed Successfully

Verification:

- GitHub Actions workflow runs automatically on every push
- Image successfully published to Docker Hub
- Latest version always available for deployment

PART C - Cloud Deployment on AWS EC2 ✅

Objective: Deploy the containerized application to cloud infrastructure with automated deployment.

Implementation:

- Provisioned AWS EC2 instance (t3.micro, eu-north-1)
- Installed Docker on EC2 instance
- Configured security groups (ports 22, 80)
- Created automated deployment workflow: `.github/workflows/deploy-to-aws-ec2.yml`
- Implemented SSH-based continuous deployment
- Configured container to restart automatically

Infrastructure Details:

- **Cloud Provider:** Amazon Web Services (AWS)
- **Service:** EC2 (Elastic Compute Cloud)
- **Instance ID:** i-04943a1f391046041
- **Instance Type:** t3.micro (1 vCPU, 1GB RAM)
- **Region:** eu-north-1 (Stockholm)
- **Public IP:** 13.60.8.113
- **Operating System:** Amazon Linux 2023
- **Container Runtime:** Docker 25.0.13

GitHub Secrets Configured:

- `AWS_EC2_HOST` - EC2 public IP address

- `AWS_EC2_USER` - SSH username (ec2-user)
- `AWS_EC2_SSH_KEY` - Private SSH key for authentication

Deployment Process:

1. Code pushed to GitHub main branch
2. GitHub Actions triggers deployment workflow
3. Workflow connects to EC2 via SSH
4. Stops existing container (if running)
5. Pulls latest image from Docker Hub
6. Starts new container with updated code
7. Maps port 80 to container port 5000

Status: ✅ Completed Successfully

Verification:

- Application accessible at: http://13.60.8.113/api/doc
- Automatic deployment on code changes
- Container running stably on EC2

🧪 Testing & Validation

Comprehensive Testing Results

Total Tests Executed: 22
Passed: 22 ✅
Failed: 0
Success Rate: 100%

Test Categories:

1. Docker Build Tests

- ✅ Dockerfile syntax validation
- ✅ Image build successful (2.7s)
- ✅ Layer optimization verified
- ✅ Dependencies installed correctly

2. Container Execution Tests

- ✅ Container starts successfully (5s)
- ✅ Port mapping works (5000:5000)
- ✅ Application logs accessible
- ✅ Health check passed

3. API Endpoint Tests

- ✅ GET /api/stores → 200 OK

- ✅ POST /api/stores → 201 Created
- ✅ GET /api/items → 200 OK
- ✅ POST /api/items → 201 Created

4. Database Tests

- ✅ SQLite database created
- ✅ Tables initialized (stores, items)
- ✅ CRUD operations working
- ✅ Data persistence verified

5. AWS Deployment Tests

- ✅ SSH connectivity to EC2
- ✅ Container running on EC2
- ✅ Public access via HTTP
- ✅ Swagger UI accessible

6. CI/CD Pipeline Tests

- ✅ GitHub Actions workflows validated
- ✅ DockerHub push successful
- ✅ Automated deployment working
- ✅ GitHub Secrets configured

7. UI/Documentation Tests

- ✅ Swagger UI loads (200 OK)
- ✅ Professional theme applied
- ✅ API documentation complete
- ✅ Interactive testing available

Detailed Test Report: Available in `COMPREHENSIVE_TEST_REPORT.md`

📈 Performance Metrics

Metric	Value	Status
Docker Build Time	2.7 seconds	✅ Excellent
Container Startup Time	5 seconds	✅ Fast
API Response Time	< 100ms	✅ Excellent
Docker Image Size	~150MB	✅ Optimized
Memory Usage	~50MB	✅ Efficient
CI/CD Pipeline Duration	~2 minutes	✅ Fast

🛠️ Technical Stack

Programming & Frameworks

- **Language:** Python 3.9
- **Web Framework:** Flask
- **API Framework:** Flask-RESTPlus
- **ORM:** Flask-SQLAlchemy
- **Serialization:** Marshmallow

DevOps & Infrastructure

- **Containerization:** Docker
- **CI/CD Platform:** GitHub Actions
- **Container Registry:** Docker Hub
- **Cloud Provider:** AWS EC2
- **Version Control:** Git & GitHub

Database

- **Development:** SQLite
- **Storage:** File-based (data.db)

Additional Tools

- **API Documentation:** Swagger UI (Flask-RESTPlus)
- **SSH Authentication:** RSA Key-based
- **Security:** GitHub Secrets management

📁 Repository Structure

```
final-project-DEVOPS/
├── .github/
│   └── workflows/
│       ├── docker-build-push.yml       # Part B - CI/CD to DockerHub
│       ├── deploy-to-aws-ec2.yml       # Part C - AWS Deployment
│       └── deploy-to-azure-vm.yml      # Alternative (disabled)
│
├── models/
│   ├── item.py                          # Item database model
│   └── store.py                         # Store database model
│
├── resources/
│   ├── __init__.py
│   ├── item.py                          # Item API endpoints
│   └── store.py                         # Store API endpoints
│
├── schemas/
│   ├── __init__.py
│   ├── item.py                        # Item validation schema
│   └── store.py                       # Store validation schema
│
├── templates/
│   └── swagger-ui.html                # Custom Swagger UI theme
│
├── app.py                             # Main Flask application
├── db.py                              # Database initialization
├── ma.py                              # Marshmallow setup
├── Dockerfile                         # Docker configuration
├── requirements.txt                   # Python dependencies
├── README.md                          # Project documentation
│
├── PART_A_DOCUMENTATION.md           # Dockerization guide
├── PART_B_DOCUMENTATION.md           # CI/CD setup guide
├── PART_C_DOCUMENTATION.md           # Deployment guide
├── COMPREHENSIVE_TEST_REPORT.md      # Testing results
├── PROJECT_SUBMISSION_GUIDE.md       # Submission instructions
└── QUICK_REFERENCE.md                # Quick command reference
```

🎨 Additional Features & Enhancements

Professional UI Theme

- Custom gradient header with branding
- Technology stack badges
- Enhanced color scheme
- Responsive design
- Footer with GitHub and Docker Hub links

API Documentation

- Complete endpoint descriptions
- Request/Response examples
- Interactive API testing
- Deployment information
- Professional metadata

Security Best Practices

- No hardcoded credentials
- GitHub Secrets for sensitive data
- SSH key-based authentication
- Minimal base Docker image
- Security group configuration

📚 Documentation

All documentation is comprehensive and included in the repository:

1. **README.md** - Project overview and setup instructions
2. **PART_A_DOCUMENTATION.md** - Detailed Dockerization guide
3. **PART_B_DOCUMENTATION.md** - CI/CD pipeline setup and configuration
4. **PART_C_DOCUMENTATION.md** - Cloud deployment walkthrough
5. **COMPREHENSIVE_TEST_REPORT.md** - Complete testing results
6. **PROJECT_SUBMISSION_GUIDE.md** - Submission guidelines
7. **QUICK_REFERENCE.md** - Quick command reference

🔒 Security Considerations

Implemented Security Measures:

- ✅ SSH key-based authentication (no passwords)
- ✅ GitHub Secrets for sensitive credentials
- ✅ Docker Hub token authentication
- ✅ AWS Security Group restrictions (ports 22, 80 only)
- ✅ No credentials committed to repository
- ✅ Private SSH key secured with proper permissions
- ✅ Regular security updates via automated deployments

🚀 Deployment Workflow

Continuous Deployment Process:

```
1. Developer commits code
        ↓
2. Push to GitHub main branch
        ↓
3. GitHub Actions triggered automatically
        ↓
4. Workflow: docker-build-push.yml
   - Build Docker image
   - Push to Docker Hub
        ↓
5. Workflow: deploy-to-aws-ec2.yml
   - SSH to EC2 instance
   - Pull latest image
   - Restart container
        ↓
6. Application updated in production
   - Zero-downtime deployment
   - Automatic rollout
```

Average deployment time: ~2 minutes from commit to production

---

☑ Project Requirements Checklist

Part A - Dockerization

- ☑ Dockerfile created and optimized
- ☑ Application containerized successfully
- ☑ Local testing completed
- ☑ Port configuration correct
- ☑ Dependencies properly managed

Part B - CI/CD Pipeline

- ☑ GitHub Actions workflow configured
- ☑ Automated builds on code push
- ☑ Docker Hub integration working
- ☑ GitHub Secrets configured
- ☑ Continuous integration validated

Part C - Cloud Deployment

- ☑ Cloud instance provisioned (AWS EC2)
- ☑ Docker installed on cloud instance
- ☑ Application deployed to cloud
- ☑ Public access configured
- ☑ Automated deployment from GitHub
- ☑ Production environment stable

Additional Requirements

- ☑ Complete documentation provided
- ☑ Code well-organized and clean
- ☑ Professional UI/UX implemented
- ☑ Comprehensive testing performed
- ☑ Security best practices followed
- ☑ Repository properly structured

🎯 Learning Outcomes Achieved

Through this project, I have successfully demonstrated:

1. **Containerization Skills**
   - Docker image creation and optimization
   - Container lifecycle management
   - Multi-stage builds
   - Port mapping and networking
2. **CI/CD Implementation**

   - GitHub Actions workflow development
   - Automated testing and deployment
   - Integration with container registries
   - Secrets management

3. **Cloud Computing**

   - AWS EC2 provisioning and management
   - Cloud security configuration
   - Remote server administration
   - Infrastructure as Code concepts

4. **DevOps Best Practices**

   - Version control with Git
   - Automated deployment pipelines
   - Documentation standards
   - Security considerations

5. **Software Development**

   - REST API development
   - Database design and integration
   - API documentation
   - Professional UI/UX design

📞 Support & Resources

Project Links:

- **GitHub Repository:** https://github.com/gazal1994/final-project-DEVOPS
- **Docker Hub:** https://hub.docker.com/r/gazal94/final-python-app
- **Live Application:** http://13.60.8.113/api/doc

Quick Commands:

Test the application locally:

```
docker pull gazal94/final-python-app:latest
docker run -p 5000:5000 gazal94/final-python-app:latest
# Access: http://localhost:5000/api/doc
```

SSH to production server:

```
ssh -i KEY.pem ec2-user@13.60.8.113
docker ps  # Check container status
docker logs final-python-app  # View logs
```

🎓 Conclusion

This project successfully implements a complete DevOps pipeline from development to production deployment. All three parts (Dockerization, CI/CD Pipeline, and Cloud Deployment) have been implemented, tested, and validated with 100% success rate.

Key Achievements:

- ✅ Full containerization with Docker
- ✅ Automated CI/CD pipeline
- ✅ Production deployment on AWS EC2
- ✅ Professional documentation
- ✅ Comprehensive testing (22/22 passed)
- ✅ Security best practices implemented
- ✅ Zero-downtime deployment capability

The application is currently running in production and accessible to the public, demonstrating a real-world DevOps implementation.