Lecture 10: Deployment and DevOps

From Code to Production

State University of Zanzibar (SUZA)
BSc Computer Science

Contents

1 Introduction to Deployment

1.1 What is Deployment?

Deployment is the process of making your application available to users by moving it from your development environment to a production server.

1.2 Deployment Pipeline

$$\text{Development} \rightarrow \text{Testing} \rightarrow \text{Staging} \rightarrow \text{Production}$$

Environments:

- **Development:** Local machine where you code
- **Testing:** Environment for running tests
- **Staging:** Pre-production environment (mirrors production)
- **Production:** Live environment for real users

2 Introduction to DevOps

2.1 What is DevOps?

DevOps combines Development (Dev) and Operations (Ops) to shorten the development lifecycle while delivering features, fixes, and updates frequently.

2.2 DevOps Principles

- **Automation:** Automate repetitive tasks
- **Continuous Integration:** Merge code frequently
- **Continuous Deployment:** Deploy automatically
- **Infrastructure as Code:** Manage infrastructure with code
- **Monitoring:** Track application health

3 Continuous Integration (CI)

3.1 What is CI?

Developers merge code changes frequently into a shared repository. Each merge triggers automated builds and tests.

3.2 CI Workflow

1. Developer pushes code to repository
2. CI server detects changes
3. Automated build runs
4. Automated tests run
5. Results reported (pass/fail)

3.3 CI Tools

Tool	Description
GitHub Actions	Built into GitHub, free tier
GitLab CI	Built into GitLab
Jenkins	Self-hosted, highly customizable
CircleCI	Cloud-based CI/CD
Travis CI	Popular for open source

3.4 GitHub Actions Example

Create `.github/workflows/ci.yml`:

```
name: CI Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run linter
        run: npm run lint
```

```
      - name: Run tests
        run: npm test

      - name: Build project
        run: npm run build
```

4 Continuous Deployment (CD)

4.1 What is CD?

Automatically deploy code to production when it passes all tests.

4.2 Deployment Strategies

1. Big Bang Deployment

- Replace entire application at once
- Simple but risky
- Downtime during deployment

2. Rolling Deployment

- Update servers one at a time
- No downtime
- Gradual rollout

3. Blue-Green Deployment

- Two identical environments (Blue and Green)
- Deploy to inactive environment
- Switch traffic when ready
- Easy rollback

4. Canary Deployment

- Deploy to small subset of users first
- Monitor for issues
- Gradually increase rollout

5 Hosting Platforms

5.1 Platform Comparison

Platform	Best For	Free Tier	Ease
Vercel	Frontend, Next.js	Yes	Easy
Netlify	Static sites, JAMstack	Yes	Easy
Railway	Full-stack, databases	Limited	Easy
Render	Full-stack apps	Yes	Medium
Heroku	Backend, APIs	Limited	Medium
DigitalOcean	VPS, full control	No	Hard
AWS	Enterprise, scalable	Limited	Hard

5.2 Deploying to Vercel

```
# Install Vercel CLI
npm install -g vercel

# Login
vercel login

# Deploy
vercel

# Deploy to production
vercel --prod
```

Or connect GitHub repository through Vercel dashboard for automatic deployments.

5.3 Deploying to Heroku

```
# Install Heroku CLI
brew install heroku/brew/heroku

# Login
heroku login

# Create app
heroku create my-app-name

# Add Procfile
echo "web: npm start" > Procfile

# Deploy
git push heroku main

# Open app
heroku open
```

5.4 Deploying to Railway

1. Go to https://railway.app
2. Connect GitHub account
3. Select repository
4. Configure environment variables
5. Deploy automatically on push

6 Docker Basics

6.1 What is Docker?

Docker packages applications with all dependencies into containers that run consistently anywhere.

6.2 Key Concepts

- **Image:** Blueprint for containers (like a class)
- **Container:** Running instance of an image (like an object)
- **Dockerfile:** Instructions to build an image
- **Docker Hub:** Registry for sharing images

6.3 Dockerfile Example (Node.js)

```
# Use official Node.js image
FROM node:18-alpine

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy source code
COPY . .

# Expose port
EXPOSE 3000

# Start application
CMD ["npm", "start"]
```

6.4 Docker Commands

```
# Build image
docker build -t myapp .

# Run container
docker run -p 3000:3000 myapp

# List running containers
docker ps

# Stop container
docker stop <container_id>

# Remove container
docker rm <container_id>
```

7 Environment Variables

7.1 Why Environment Variables?

- Keep secrets out of code
- Different values for different environments
- Easy to change without code changes

7.2 Using .env Files

Create `.env` file:

```
# Database
DATABASE_URL=postgresql://user:pass@localhost:5432/mydb

# Application
PORT=3000
NODE_ENV=development

# Secrets
JWT_SECRET=your-secret-key
API_KEY=external-api-key
```

IMPORTANT: Add `.env` to `.gitignore`!

7.3 Accessing in Code

Node.js:

```
require('dotenv').config();

const port = process.env.PORT || 3000;
const dbUrl = process.env.DATABASE_URL;
```

Python:

```
import os
from dotenv import load_dotenv

load_dotenv()

port = os.getenv('PORT', 3000)
db_url = os.getenv('DATABASE_URL')
```

8 Database Deployment

8.1 Database Options

Service	Database	Free Tier
Railway	PostgreSQL, MySQL	500MB
PlanetScale	MySQL	5GB
MongoDB Atlas	MongoDB	512MB
Supabase	PostgreSQL	500MB
ElephantSQL	PostgreSQL	20MB

8.2 Database Migrations

Migrations track database schema changes.

```
# Create migration
npx prisma migrate dev --name add_users_table

# Apply migrations to production
npx prisma migrate deploy
```

9 Monitoring and Logging

9.1 Why Monitor?

- Detect issues before users report them
- Track performance metrics
- Understand usage patterns
- Debug production issues

9.2 What to Monitor

- **Uptime:** Is the application running?
- **Response time:** How fast are requests?

- **Error rate:** How many requests fail?
- **Resource usage:** CPU, memory, disk

9.3 Monitoring Tools

Tool	Purpose
UptimeRobot	Uptime monitoring (free)
Sentry	Error tracking
LogRocket	Frontend monitoring
Datadog	Full observability

10 SSL/HTTPS

10.1 Why HTTPS?

- Encrypts data in transit
- Required for user trust
- SEO ranking factor
- Required for many APIs

Most hosting platforms (Vercel, Netlify, Railway) provide free SSL certificates automatically.

11 Deployment Checklist

All tests passing

Environment variables configured

Database migrated

SSL/HTTPS enabled

Error tracking set up

Monitoring configured

Backup strategy in place

README updated

Team notified

12 Rollback Procedures

12.1 When to Rollback

- Critical bug discovered
- Performance degradation
- Security vulnerability

12.2 How to Rollback

Vercel/Netlify: Click "Rollback" in dashboard

Heroku:

```
# List releases
heroku releases

# Rollback to previous version
heroku rollback v123
```

Git-based:

```
# Revert last commit
git revert HEAD
git push origin main
```

13 Practical Exercise

For your project:

1. Set up GitHub Actions for CI
2. Deploy to a hosting platform (Vercel, Railway, or Heroku)
3. Configure environment variables
4. Set up basic monitoring (UptimeRobot)
5. Document deployment process in README

14 Summary

- DevOps automates the path from code to production
- CI ensures code quality through automated testing
- CD enables frequent, reliable deployments
- Choose hosting platform based on project needs
- Docker provides consistent environments

- Always use environment variables for secrets
- Monitor your application in production
- Have a rollback plan ready