Deployment & DevOps — Docker & GitHub Actions

Web Technologies — Lecture 13

Masoud Hamad

State University of Zanzibar (SUZA)

Semester II, 2025/2026

Outline

DevOps in One Idea

Dev (build features) + **Ops** (run them in production), collaborating closely with **automation** so software ships *frequently and safely*. Practices we'll cover:

- **Containers** (Docker) — consistent environments
- **Orchestration** (Compose) — multi-container apps
- **CI** (GitHub Actions) — run tests on every push
- **CD** — deploy automatically on green builds
- **Monitoring** — know when production breaks

Why Containers?

- "Works on my machine" $\rightarrow$ "works *everywhere*"
- Bundles your app + its OS dependencies into an **image**
- Image runs as a **container** on any Docker-enabled host
- Lightweight (shared kernel) compared to VMs
- Same image runs on dev laptop, CI, staging, production

A Dockerfile for a Node App

```
# multi-stage build for a small final image
FROM node:20-alpine AS deps
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev

FROM node:20-alpine AS runner
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
ENV NODE_ENV=production
EXPOSE 3000
CMD ["node", "server.js"]
```

A Dockerfile for a Spring Boot App

```
FROM eclipse-temurin:21-jdk AS build
WORKDIR /app
COPY . .
RUN ./mvnw -B package -DskipTests

FROM eclipse-temurin:21-jre-alpine
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Docker Commands You'll Use

```
docker build -t myapp:1.0 .        # build image from Dockerfile
docker run -p 3000:3000 myapp:1.0 # run container
docker ps                          # list running containers
docker logs <container>            # tail logs
docker exec -it <container> sh     # shell into a container
docker push registry/myapp:1.0     # publish to a registry
```

Multi-Container Apps with Compose

```
# docker-compose.yml
services:
  api:
    build: ./api
    ports: ["3000:3000"]
    environment:
      DB_URL: postgres://app:secret@db:5432/app
    depends_on: [db]

  db:
    image: postgres:16
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: app
    volumes: ["pgdata:/var/lib/postgresql/data"]

volumes:
  pgdata:
```

Bring It Up

```
docker compose up           # start everything
docker compose up -d        # detached
docker compose logs -f api  # tail one service
docker compose down         # stop and remove
```

Compose is great for development and small production setups. For larger systems: Kubernetes (K8s).

A Minimal CI Workflow

```yaml
# .github/workflows/ci.yml
name: CI
on:
  push: { branches: [main] }
  pull_request: {}

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: '20', cache: 'npm' }
      - run: npm ci
      - run: npm test
      - run: npm run build
```

Every push and PR runs tests automatically. Failed checks block merging.

Build & Push a Docker Image

```yaml
jobs:
  docker:
    runs-on: ubuntu-latest
    needs: test
    if: github.ref == 'refs/heads/main'
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - uses: docker/build-push-action@v5
        with:
          push: true
          tags: ghcr.io/${{ github.repository }}:latest
```

Where to Deploy?

Tier	Examples	Notes
Free / hobby	Render, Fly.io, Railway, Vercel	Great for student projects
Self-hosted	DigitalOcean, Linode, Hetzner	VPS + Docker Compose
Cloud (PaaS)	Heroku, Cloud Run, App Engine	Push code, get URL
Cloud (IaaS)	AWS, GCP, Azure	Full control, full responsibility
Kubernetes	EKS, GKE, AKS	For larger systems

For your capstone: **Render** or **Fly.io** are fast, free, and Docker-native.

12-Factor App (highlights)

A set of best practices for building cloud-friendly apps.

- **Codebase** — one repo, many deploys
- **Config in environment** — not in code
- **Dependencies** declared explicitly (`package.json`, `pom.xml`)
- **Backing services** (DB, queues) attached via URL
- **Build, release, run** clearly separated
- **Stateless processes** — can scale horizontally
- **Logs as event streams** — write to stdout

Monitoring & Alerting

Once your app is live, you need to know when it breaks — *before* the users tell you.

- **Metrics** — request rate, error rate, latency (Prometheus, Grafana)
- **Logs** — structured JSON, searchable (Loki, ELK, Datadog)
- **Traces** — follow a request across services (OpenTelemetry, Jaeger)
- **Alerting** — page someone when SLOs are violated (PagerDuty, OpsGenie)
- **Uptime checks** — ping your URL every minute (UptimeRobot, Better Uptime)

What You Learned This Semester

1. Web foundations & architecture
2. Design patterns + Agile + GitHub
3. HTML & CSS
4. JavaScript & the DOM
5. AJAX, Fetch & REST consumption
6. React (and friends)
7. Databases & ORMs
8. Backend APIs (Spring, Node, Django)
9. Auth & security
10. Deployment & DevOps

Where to Go Next

- Build the capstone — ship it to the public Internet
- Pick **one** backend stack and go deep
- Learn TypeScript — it pays off in larger projects
- Try a serverless platform (Cloudflare Workers, AWS Lambda)
- Study one large open-source web app's architecture
- Contribute to open source — the best way to learn

Summary

- Containers make apps portable and consistent
- Compose runs multi-service apps with one command
- GitHub Actions tests every push, builds images on merge
- Free tiers (Render, Fly.io) are great for student projects
- 12-Factor + observability = ops you don't dread

Thank you. Now go build something useful.

The End