

# NotesApp – Full Stack DevOps Deployment

*This document provides detailed documentation of the entire development, containerization, deployment, and automation process of the NotesApp project.*

## 1. Цел на проектот

Целта на овој проект е да се демонстрира целосен DevOps процес преку изработка и автоматизација на современа cloud-native апликација. Проектот опфаќа развој, контејнеризација, континуирана интеграција, и оркестрација на повеќеслојна веб апликација составена од три независни, но меѓусебно поврзани сервиси: frontend, backend и база на податоци.

Фокусот е ставен на практичната примена на следниве DevOps технологии и алатки:

- **Docker** за креирање на лесни и преносливи контејнери за секој сервис поединечно.
- **Docker Compose** за локална оркестрација и поврзување на сервисите во развојна околина.
- **GitHub Actions** за имплементација на CI/CD pipeline што автоматски ги гради и поставува Docker сликите на Docker Hub.
- **Kubernetes** за продукциска оркестрација на апликацијата, со дефинирање на сите видови потребни манифести.
- **ConfigMaps и Secrets** за безбедно и флексибилно управување со конфигурации и чувствителни податоци (како лозинки).

Преку овој проект се овозможува разбирање и практична примена на процесите што се суштински за современо развивање и испорака на апликација оптимизирана за cloud, со акцент на автоматизација, стабилност, скалабилност и одржливост на целиот систем.

**Линк до проектот на github:**

**<https://github.com/Mario-Sek/notes-app>**

## 2. Организација на апликацијата

Апликацијата претставува NotesApp кој дава пристап до сите CRUD функционалности за правање и пребарување на белешки.

Таа е организирана во три сервиси, при што секој сервис е независно развиен, докеризиран и управуван преку Kubernetes. Сервисите комуницираат преку внатрешна мрежа во кластерот, а целиот систем работи како целина, оптимизирана за лесно скалирање и одржување.

Сервисите се следните:

### - Frontend

Развиен е со React и Vite, дава добро дизајниран интерфејс.

Ви однос на конфигурацијата, тој користи .env.production фајл во кој чува env variables за точно поврзување со backend-от за кога ќе биде во кластерот.

При изградбата (npm run build) променлиците се хардкодираат, а frontend-от се сервира како статичка веб страница преку Nginx.

### - Backend

Имплементиран е во Java со SpringBoot, тој користи REST архитектура за обезбедување на сите CRUD функционалности.

За конекцијата со базата се справува Spring Data JPA. Во самиот application.properties фајл се чуваат environmental variables за името на базата, username и password. Тие се превземаат преку configMap и secret манифестите превземени од deployment манифестот.

### - Database

Користи PostgreSQL како релациона база на податоци.

Во продукција е конфигурирана преку Kubernetes StatefulSet, со поврзани PersistentVolume и Secrets за кредитенцијали.

### 3. Контејнеризација со Docker

Секој сервис од апликацијата е контејнеризиран одделно користејќи Docker, ова овозможува изолирано и преносливо извршување на сите компоненти. Вака апликацијата еднакво ќе се однесува без разлика дали е извршена локално, во CI/CD pipeline или во продукциска cloud околина.

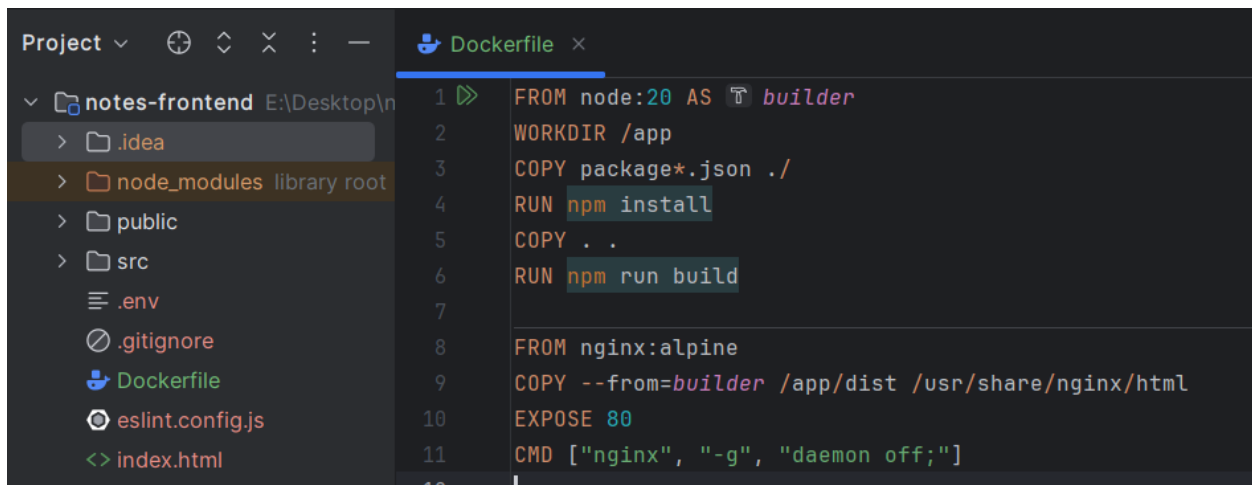
#### - Dockerfile за Frontend (React + Nginx)

Frontend-от е развиен со React и Vite. За да се сервира како статичка веб страница, тој најпрво ги инсталира зависностите па се билда со `npm run build`, при што се генерираат статичките фајлови во `/dist` директориумот.

Значи има две фази:

Build фаза: Користи Node image за да го билда React проектот.

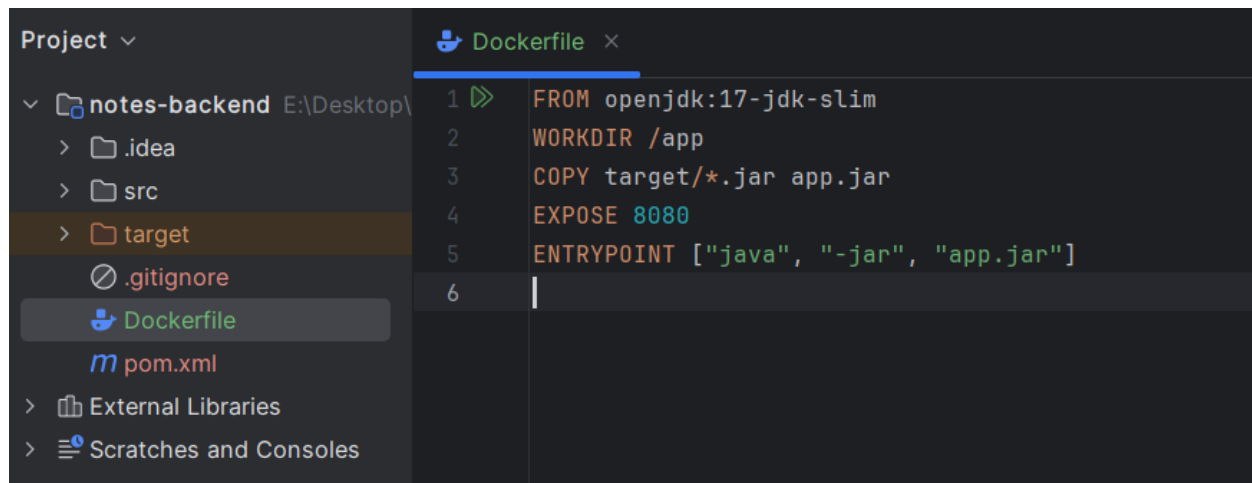
Serve фаза: Користи Nginx image за да ги сервира статичките фајлови.



```
1 FROM node:20 AS builder
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 RUN npm run build
7
8 FROM nginx:alpine
9 COPY --from=builder /app/dist /usr/share/nginx/html
10 EXPOSE 80
11 CMD ["nginx", "-g", "daemon off;"]
```

## - Dockerfile за Backend (Spring Boot)

Backend-от користи light openjdk:17 image. При контејнеризација, веќе изградената .jar датотека од target/ директориумот се копира во Docker сликата. Потоа, при стартување на контејнерот, таа се извршува со командата java -jar app.jar. Портата е поставена на 8080. Оваа конфигурација овозможува лесно и брзо пуштање на backend-от во кластерот, без потреба од повторно градење на апликацијата во самиот контејнер.



The screenshot shows an IDE interface with a project explorer on the left and a code editor on the right. The project explorer shows a project named 'notes-backend' with subdirectories '.idea', 'src', 'target', and files '.gitignore', 'Dockerfile', and 'pom.xml'. The 'Dockerfile' file is selected. The code editor displays the following Dockerfile content:

```
1 FROM openjdk:17-jdk-slim
2 WORKDIR /app
3 COPY target/*.jar app.jar
4 EXPOSE 8080
5 ENTRYPOINT ["java", "-jar", "app.jar"]
6
```

## - PostgreSQL база

За базата се користи официјалниот image postgres:17.4, без потреба од сопствен Dockerfile. При локална употреба без kubernetes се користи docker-compose фајлот со неговите env променливи. При работа со Kubernetes кластерот, за базата се користи сервисот дефиниран од db-statefulset манифестот. Тој креденцијалите ги зема преку secrets и configMap фајловите. А за перзистенција на податоците се користи PersistentVolumeClaim, кој овозможува податоците да останат зачувани дури и при рестарт на подот.

## - Docker Compose фајлот

Compose фајлот дефинира три сервиси: db (PostgreSQL), backend (Spring Boot) и frontend (React + Nginx), како и еден volume за перзистирање а на податоците.

- **db (PostgreSQL)**  
Користи официјална postgres:15 слика. Преку environment се поставуваат база, корисник и лозинка. Податоците се чуваат во volume db-data, а сервисот е достапен на порта 5432.
- **backend (Spring Boot)**  
Се гради од ./notes-backend. Се поврзува со базата преку jdbc URL користејќи Docker мрежа. Отвора порта 8080 и зависи од db за правилен старт.
- **frontend (React + Vite + Nginx)**  
Се билда од ./notes-frontend и се сервира преку Nginx. Достапен е на порта 3000 и зависи од backend.
- **volumes: db-data**  
Обезбедува перзистирање на базата, податоците остануваат зачувани при рестартирање.

```
docker-compose.yml x
1  version: "3.8"
2  services:
3    db:
4      image: postgres:15
5      environment:
6        POSTGRES_DB: notesdb
7        POSTGRES_USER: mario
8        POSTGRES_PASSWORD: password
9      volumes:
10       - db-data:/var/lib/postgresql/data
11      ports:
12       - "5432:5432"
13
14   backend:
15     build: ./backend
16     environment:
17       SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/notesdb
18       SPRING_DATASOURCE_USERNAME: mario
19       SPRING_DATASOURCE_PASSWORD: password
20     ports:
21       - "8080:8080"
22     depends_on:
23       - db
24
25   frontend:
26     build: ./frontend
27     ports:
28       - "3000:80"
29     depends_on:
30       - backend
31
32 volumes:
33   db-data:
```

## 4. CI/CD со GitHub Actions

За автоматизација на процесот на градење и поставување на Docker слики, е поставен CI/CD pipeline користејќи GitHub Actions. Workflow фајлот се активира при секој push на main гранката, со што автоматски ги гради и поставува сликите на Docker Hub.

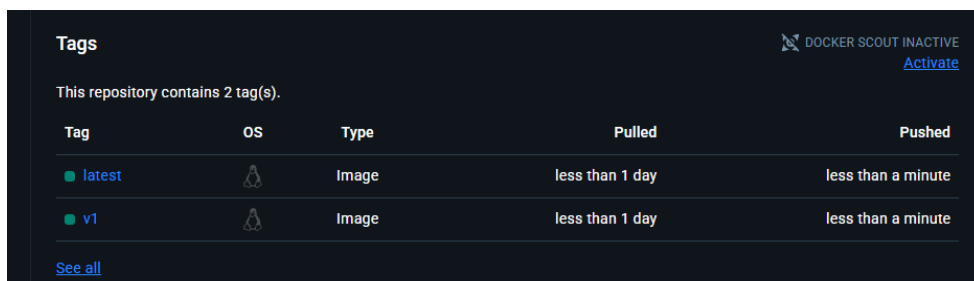
Чекорите што се превземаат:

- Се превзема кодот од GitHub со actions/checkout.
- Се поставува JDK 17 за да може да се билда Spring Boot апликацијата.
- Гради backend-от со Maven во notes-backend директориумот.
- Поставува Docker Buildx за напредно градење на images.
- Се логира на Docker Hub користејќи GitHub Secrets назначени во settings на репото: (DOCKERHUB\_USERNAME, DOCKERHUB\_TOKEN )
- Ги гради и поставува двете Docker слики:
- Поставува два тагови за секој image:

Едниот за лесен пристап е секогаш latest верзијата, другиот користи github.run\_number со кој задава верзија на тагот еднаква на бројот на колку пати се извршил (успешно или не) самиот action.

Пр: mariosek1/notes-backend:v3

mariosek1/notes-backend:latest



The screenshot shows the Docker Scout interface for a repository. It displays a table of tags with columns for Tag, OS, Type, Pulled, and Pushed. There are two tags listed: 'latest' and 'v1', both of type 'Image'. The 'Pulled' and 'Pushed' times for both are 'less than 1 day' and 'less than a minute' respectively. A 'See all' link is at the bottom left, and a 'DOCKER SCOUT INACTIVE' status with an 'Activate' link is at the top right.

Tag	OS	Type	Pulled	Pushed
latest	linux	Image	less than 1 day	less than a minute
v1	linux	Image	less than 1 day	less than a minute

main

notes-app / .github / workflows / main.yml

Mario-Sek

Create main.yml

✓

Code

Blame

50 lines (41 loc) · 1.24 KB

Code 55% faster with GitHub Copilot

```
1  name: Push to DockerHub
2
3  on:
4    push:
5      branches: [ main ]
6
7  jobs:
8    build-and-push:
9      runs-on: ubuntu-latest
10
11     steps:
12       - name: Checkout code
13         uses: actions/checkout@v3
14
15       - name: Set up JDK 17
16         uses: actions/setup-java@v3
17         with:
18           java-version: '17'
19           distribution: 'temurin'
20
21       - name: Build backend jar
22         working-directory: ./notes-backend
23         run: mvn clean package -DskipTests
24
25       - name: Set up Docker Buildx
26         uses: docker/setup-buildx-action@v3
27
28       - name: Log in to DockerHub
29         uses: docker/login-action@v3
30         with:
31           username: ${{ secrets.DOCKERHUB_USERNAME }}
32           password: ${{ secrets.DOCKERHUB_TOKEN }}
33
34       - name: Build and push backend
35         uses: docker/build-push-action@v5
36         with:
37           context: ./notes-backend
38           push: true
39           tags: |
40             mariosek1/notes-backend:v${{ github.run_number }}
41             mariosek1/notes-backend:latest
42
43       - name: Build and push frontend
44         uses: docker/build-push-action@v5
45         with:
46           context: ./notes-frontend
47           push: true
48           tags: |
49             mariosek1/notes-frontend:v${{ github.run_number }}
50             mariosek1/notes-frontend:latest
```



## 5. Kubernetes манифести

Во мојата конфигурација користам повеќе Kubernetes манифест фајлови за да го организирам кластерот. За да го поврзам backend-от и frontend-от и да ги направам надворешно пристапни, користам Ingress ресурс кој ги рутира HTTP барањата кон соодветните сервиси.

Backend-от е имплементиран со користење на Deployment, Service, ConfigMap и Secret за управување со конфигурациите и чувствителните податоци.

Базата на податоци користи слични ресурси, но наместо Deployment користи StatefulSet заради потребата од перзистентност и стабилен идентитет на подовите.

Frontend-от користи Deployment и Service, преку кои се хостира како статичка веб апликација.

За прецизна организација и одделување на ресурсите во кластерот користам Namespace манифест.

### Кратко објаснување на манифестите:

#### - Namespace

Прави одделување и групирање на сите ресурси во Kubernetes кластерот за подобра организација и управување.

#### - ConfigMap

Чува несензитивни конфигурациски податоци кои се користат како env променливи во апликациите.

### **- Secret**

Обезбедува безбедно чување и енкодирање на чувствителни податоци како кориснички имиња, лозинки и API клучеви.

### **- Deployment**

Управува со животниот циклус на подовите, овозможувајќи креирање, скалирање и ажурирање на апликациски контејнери.

### **- Service**

Обезбедува стабилни IP и DNS име за пристап до подовите, овозможува внатрешна комуникација во кластерот (на пр. ClusterIP).

### **- StatefulSet**

Корисен за апликации кои бараат стабилен идентитет и перзистентна меморија, како бази на податоци, со поддршка за Persistent Volumes.

### **- Ingress**

Обезбедува HTTP/HTTPS пристап од надворешноста до сервисите во кластерот преку правила за рутирање на сообраќајот и избор на Ingress контролер.