

**Универзитет Св. Кирил и Методиј - Скопје**  
**Факултет за информатички науки и компјутерско инженерство**  
**Прв циклус на студии**



**Проектна задача по предметот**  
**Континуирана интеграција и испорака**  
**На тема:**  
**Еднократна лозинка базирана на време**

**Ментори:**

**д-р Милош Јовановиќ**

**д-р Панче Рибарски**

**Кандидат:**

**Ѓорги Нечовски**

**број на индекс 211177**

Вовед.....	3
Организација на апликацијата.....	4
Docker compose.....	4
Kubernetes.....	7
Namespace .....	7
ConfigMap and Secrets.....	7
Service and StatefulSet Deployment .....	8
Ingress.....	11

## Вовед

Цел на оваа проектна задача е изработка и верифицирање на еднократна лозинка базирана на време. Таа ќе прикаже функционалност преку симулација на мал банкарски систем кој има основни функционалности како автентикација, авторизација на корисници и вработени, трансфер, депозит и повлекување на финансиски средства.

За потребите на овој предмет апликацијата ја надоградив со DevOps елементи, односно и направив да работи со docker compose и Kubernetes, исто така е поставена на Github со свој actions за контриуирана испорака.

## Организација на апликацијата

Апликацијата е дизајнирана врз шаблонот Model-View-Controller (MVC), што претставува популарен архитектурен модел за развој на софтверски апликации. За backend делот на апликацијата, користи се Spring Framework, кој обезбедува многу функционалности и апстракции за развој на веб апликации, вклучувајќи ги управувањето со зафатниот контролер, сервисите и репозиториумите. Во позадина, апликацијата користи MySQL база на податоци за зачувување на сите релевантни податоци како кориснички информации, трансакции и други податоци поврзани со функционалностите на банкарскиот систем. MySQL е релациона база на податоци која обезбедува стабилност, перформанси и широка поддршка за работа со податоци, што ја прави идеална за користење во банкарските системи и други веб апликации. Користи Docker images за Maven и PostgreSQL.

Има 2 главни DevOps делови:

- Docker compose
- Kubernetes

### Docker compose

За овој дел креирав Dockerfile кој изгледа вака:

```
FROM maven:latest
WORKDIR /app
COPY pom.xml .
RUN mvn clean install -DskipTests
COPY . .
EXPOSE 8080
EXPOSE 443
CMD ["mvn", "spring-boot:run"]
```

Овој Dockerfile презема најнова верзија на Maven од Docker Hub и го поставува како основен слој. Прво, го копира pom.xml од апликацијата во контејнерот и ги инсталира сите потребни зависности наведени во него. Потоа, ги копира сите останати изворни датотеки од апликацијата во контејнерот. Овие чекори обезбедуваат дека зависностите се инсталираат само кога има промени во pom.xml, што го прави процесот поефикасен.

Следно, Dockerfile ги отвора портовите 8080 и 443 за да овозможи пристап до апликацијата преку HTTP и HTTPS.

Нареден чекор е Docker compose file кој може да се подели на 2 дела (сервиси), делот за база и делот за Spring.

Сервисот за база изгледа вака:

```
postgres:
  image: postgres:13
  environment:
    POSTGRES_USER: totp
    POSTGRES_PASSWORD: totp
    POSTGRES_DB: totp
  volumes:
    - db-data:/var/lib/postgresql/data
  ports:
    - "5432:5432"
  networks:
    - spring_network
```

Се користи PostgreSQL верзија 13 и се поставуваат основни променливи за конфигурација на базата на податоци, вклучувајќи корисничко име, лозинка и име на базата. За да се обезбеди постојано зачувување на податоците, дури и при бришење на Docker контејнерот, се дефинира volume каде што ќе се складираат податоците.

Контејнерот е конфигуриран да слуша на внатрешната порта 5432, која е мапирана на истата порта на хост машината, овозможувајќи пристап до базата на податоци. За да се олесни интеграцијата со Spring, контејнерот е поврзан со мрежата spring\_network, што овозможува полесно откривање на базата од страна на Spring апликациите.

Сервисот за Java Spring изгледа вака:

```
spring-app:
  build:
    context: .
    dockerfile: Dockerfile
  depends_on:
    - postgres
  environment:
    - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/totp
    - SPRING_DATASOURCE_USERNAME=totp
    - SPRING_DATASOURCE_PASSWORD=totp
    - SPRING_EMAIL_USERNAME=populargjorgi@gmail.com
    - SPRING_EMAIL_PASSWORD=jhmy tayr lrqz wdao
    - SPRING_PROFILES_ACTIVE=gjorgi
  ports:
    - "8080:8080"
    - "443:443"
  networks:
    - spring_network
```

Тука се задава потребниот Dockerfile кој што го дефиниравме претходно, и правиме апликацијата да не се стартова додека не добие знак дека базата е успешно стартована, за да има успешна конекцијаа кога ќе биде стартована. Се задаваат потребни променливи, како податоци за конекција со база и конекција со Email client. Се отвараат 2те порти 8080 и 433 на локалната хост машина и работи на истиот network со базата.

И сега можеме да го видиме изгледот на целиот Docker-compose.yaml file.

## Kubernetes

За потребите на Kubernetes се користи истиот docker image за Java Spring кој беше изграден и ставен на Dockerhub. Овој дел има повеќе поделби односно:

- Namespace
- ConfigMap and Secrets
- Database Service
- Spring Service
- Ingress

## Namespace

Овој дел се дефинира за да имаме поврзаност помеѓу сите делови кои ги специфицираме во нашиот deployment.yaml file, и подобра организираност. Скриптата изгледа вака:

```
apiVersion: v1
kind: Namespace
metadata:
  name: totp
```

## ConfigMap and Secrets

Тука се специфицираат сите потребни променливи кои ги користат Spring и базата, поделени помеѓу ConfigMap and Secrets.

Во ConfigMap се чуваат оние променливи кои не се проблем да бидат јавни како на пример email адреса која се користи, име на база и слично. Додека во Secrets се чуваат лозинките за апликациите. Направив 2 ConfigMap и 2 Secrets, една за базата а една за апликацијата. Тие изгледаат вака:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: spring-config
  namespace: totp
data:
  SPRING_EMAIL_USERNAME: populargjorgi@gmail.com
---
apiVersion: v1
kind: Secret
metadata:
  name: spring-secret
  namespace: totp
type: Opaque
data:
```

```

  SPRING_EMAIL_PASSWORD: amhteSB0YXlyIGxycXogd2Rhbw==
---
apiVersion: v1
kind: Secret
metadata:
  name: postgres-secret
  namespace: totp
type: Opaque
data:
  postgres-username: dG90cA==
  postgres-password: dG90cA==

```

Може да се примети дека Secrets се Base64 encoded за подобра безбедност.

## Service and StatefulSet Deployment

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: spring-app
  namespace: totp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: spring-app
  template:
    metadata:
      labels:
        app: spring-app
    spec:
      containers:
        - name: spring-app
          image: gjorginechovski/spring-app:latest
          env:
            - name: SPRING_EMAIL_USERNAME
              valueFrom:
                configMapKeyRef:
                  name: spring-config
                  key: SPRING_EMAIL_USERNAME
            - name: SPRING_EMAIL_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: spring-secret
                  key: SPRING_EMAIL_PASSWORD
            - name: SPRING_DATASOURCE_URL
              value: jdbc:postgresql://postgres:5432/totp

```



```

      - name: SPRING_DATASOURCE_USERNAME
        valueFrom:
          secretKeyRef:
            name: postgres-secret
            key: postgres-username
      - name: SPRING_DATASOURCE_PASSWORD
        valueFrom:
          secretKeyRef:
            name: postgres-secret
            key: postgres-password
      - name: SPRING_PROFILES_ACTIVE
        value: "http"
    ports:
      - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: spring-app
  namespace: totp
spec:
  selector:
    app: spring-app
  ports:
    - name: http
      port: 8080
      targetPort: 8080
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: totp
spec:
  serviceName: postgres
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:

```

```

- name: postgres
  image: postgres:13
  env:
    - name: POSTGRES_USER
      valueFrom:
        secretKeyRef:
          name: postgres-secret
          key: postgres-username
    - name: POSTGRES_PASSWORD
      valueFrom:
        secretKeyRef:
          name: postgres-secret
          key: postgres-password
    - name: POSTGRES_DB
      value: totp
  ports:
    - containerPort: 5432
  volumeMounts:
    - name: data
      mountPath: /var/lib/postgresql/data
  volumeClaimTemplates:
    - metadata:
        name: data
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 1Gi
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: totp
spec:
  selector:
    app: postgres
  ports:
    - port: 5432

```

Тука се запишани сите потребни елементи за Сервисите кои ги користи апликацијата

## Ingress

Апликацијата има вклучен ingress кој изгледа вака:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: spring-ingress
  namespace: totp
  annotations:
    ingress.kubernetes.io/ssl-redirect: "true"
    traefik.ingress.kubernetes.io/ssl-passthrough: "false"
    traefik.ingress.kubernetes.io/rule-type: "PathPrefixStrip"
spec:
  ingressClassName: traefik
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: spring-app
                port:
                  number: 8080
```