

Sigurnost operacijskih sustava i aplikacija

Laboratorijska vježba
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva

Dizajn sustava nadogradnje za raspodijeljenu aplikaciju

ANTE ČAVAR

Zagreb, 9. svibnja 2025

Uvod

U modernom softverskom inženjerstvu, raspodijeljene aplikacije postale su de facto standard za mnoge poslovne sustave. Ovakve aplikacije sastoje se od više komponenti koje se izvršavaju na različitim čvorovima računalne mreže, omogućujući bolju skalabilnost, dostupnost i otpornost na kvarove [1]. Međutim, takva arhitektura donosi značajne izazove prilikom nadogradnje sustava. Dok je nadogradnja monolitne aplikacije relativno jednostavan proces, nadogradnja raspodijeljene aplikacije zahtijeva pažljivo planiranje i izvođenje kako bi se izbjegao prekid usluge, osigurala konzistentnost podataka i održao visok stupanj sigurnosti [2].

Dodatni izazov predstavlja činjenica da komponente raspodijeljenog sustava mogu biti razvijene u različitim tehnologijama, imati različite životne cikluse i zahtjeve za dostupnošću. Stoga je ključno implementirati robustan i siguran sustav nadogradnje koji može koordinirati proces ažuriranja svih komponenti, osigurati kompatibilnost između različitih verzija, očuvati sigurnosne kontrole, te omogućiti vraćanje na prethodnu verziju u slučaju problema [3]. Svaka nadogradnja predstavlja potencijalni sigurnosni rizik zbog mogućnosti injektiranja zlonamjernog koda, propuštanja ranjivosti ili narušavanja sigurnosnih kontrola [4].

U ovoj laboratorijskoj vježbi, fokusirat ćemo se na dizajn i implementaciju sigurnog sustava za nadogradnju raspodijeljene aplikacije pomoću alata Docker i Kubernetes, koji su postali industrijski standard za orkestraciju kontejnera i upravljanje raspodijeljenim aplikacijama. Poseban naglasak bit će stavljen na sigurnosne aspekte procesa nadogradnje, uključujući verifikaciju integriteta novih verzija, kontrolu pristupa procesu nadogradnje, i očuvanje sigurnosnih postavki tijekom tranzicije.

Zadatak

Zamislite da radite kao DevOps inženjer u tvrtki "DynamicSoft" koja razvija sustav za upravljanje korisničkim podacima koji sadrži osjetljive osobne informacije. Sustav je implementiran kao raspodijeljena aplikacija koja se sastoji od sljedećih komponenti:

- frontend aplikacija razvijena u React.js frameworku
- backend API servis razvijen u Node.js
- servis za autentifikaciju razvijen u Go programskom jeziku
- PostgreSQL baza podataka za trajno pohranjivanje podataka

Vaš zadatak je dizajnirati i implementirati siguran sustav za nadogradnju ove aplikacije koji će omogućiti:

1. Nadogradnju pojedinačnih komponenti bez prekida rada cijelog sustava
2. Vraćanje na prethodnu verziju (rollback) u slučaju problema
3. Automatsko testiranje kompatibilnosti između komponenti
4. Nadogradnju sheme baze podataka bez gubitka podataka
5. Očuvanje svih sigurnosnih kontrola tijekom procesa nadogradnje
6. Provjeru integriteta i autentičnosti novih verzija komponenti
7. Strogo kontroliran pristup procesu nadogradnje

Tvrtka je nedavno doživjela sigurnosni incident tijekom nadogradnje sustava kada je zlonamjerni kod ušao u produkcijsko okruženje, stoga je sigurnost procesa nadogradnje postala prioritet. Za implementaciju sustava nadogradnje potrebno je koristiti Docker za kontejnerizaciju aplikacijskih komponenti i Kubernetes za orkestracijski sloj, uz dodatne sigurnosne alate za skeniranje slika i provjeru ranjivosti. Za demonstraciju funkcioniranja sustava nadogradnje potrebno je implementirati jednostavnu verziju opisanog sustava s minimalno potrebnom funkcionalnošću svake komponente i svim potrebnim sigurnosnim kontrolama.

Teorijska podloga

Raspodijeljene aplikacije

Raspodijeljene aplikacije su softverski sustavi čije komponente rade na različitim računalima povezanim mrežom [4]. Ovakva arhitektura omogućuje bolju skalabilnost, otpornost na kvarove i fleksibilnost u razvoju. Međutim, raspodijeljeni sustavi donose i značajne izazove u smislu koordinacije, konzistentnosti podataka, sigurnosti i kompleksnosti operacija poput nadogradnje [5].

Sigurnosni aspekti nadogradnje raspodijeljenih aplikacija

Sigurnost procesa nadogradnje raspodijeljenih aplikacija predstavlja kritičan aspekt DevSecOps prakse [6]. Ključni sigurnosni izazovi uključuju:

1. **Integritet koda** - Osiguravanje da nove verzije komponenti nisu kompromitirane i da ne sadrže zlonamjerni kod
2. **Sigurnosno skeniranje** - Automatsko skeniranje Docker slika za poznate ranjivosti prije implementacije
3. **Očuvanje sigurnosnih kontrola** - Osiguravanje da sigurnosne postavke (RBAC, mreža, encrypt-in-transit, encrypt-at-rest) ostaju netaknute tijekom nadogradnje
4. **Upravljanje tajnama** - Sigurno upravljanje i rotacija osjetljivih podataka poput ključeva i lozinki tijekom nadogradnje [7]
5. **Audit trail** - Bilježenje svih aktivnosti tijekom procesa nadogradnje zbog naknadne analize i usklađenosti s regulativom

Kontejnerizacija i orkestracija

Docker je platforma otvorenog koda koja omogućuje pakiranje aplikacija i njihovih ovisnosti u standardizirane jedinice zvane kontejneri [8]. Kontejneri pružaju izolaciju aplikacije i njenih ovisnosti, što olakšava konzistentno izvođenje aplikacije u različitim okolinama. Sa sigurnosnog aspekta, važno je implementirati načelo najmanje privilegije (principle of least privilege) pri konfiguriranju kontejnera [9].

Kubernetes je sustav otvorenog koda za automatizaciju implementacije, skaliranja i upravljanja kontejneriziranim aplikacijama [10]. Kubernetes pruža robusnu platformu za upravljanje raspodijeljenim aplikacijama, uključujući funkcionalnosti poput:

- Upravljanja skupinama kontejnera (podovima)
- Raspoređivanja i skaliranja aplikacija
- Balansiranja opterećenja
- Otkrivanja servisa
- Upravljanja konfiguracijom i tajnama
- Automatskog oporavka aplikacija
- Sigurnosnih politika kroz NetworkPolicy, PodSecurityPolicy i RBAC

Strategije nadogradnje aplikacija

Postoje različite strategije za nadogradnju raspodijeljenih aplikacija [11], uključujući:

1. **Rolling Update** - postupna zamjena instance stare verzije s instancama nove verzije
2. **Blue-Green Deployment** - priprema potpuno nove infrastrukture s novom verzijom aplikacije, a zatim preusmjeravanje prometa na novu verziju
3. **Canary Deployment** - postupno preusmjeravanje manjeg dijela prometa na novu verziju za testiranje prije potpune implementacije
4. **A/B Testing** - slično Canary Deploymentu, ali s fokusom na testiranje funkcionalnosti umjesto pouzdanosti

Svaka od ovih strategija ima različite sigurnosne implikacije, posebno u kontekstu očuvanja povjerljivosti, integriteta i dostupnosti podataka tijekom procesa nadogradnje [12].

Postavke za vježbu (do 5000 znakova)

Preduvjeti

Za uspješno izvođenje ove laboratorijske vježbe potrebno je imati:

1. Računalo s operacijskim sustavom Linux, Windows ili macOS
2. Instaliran Docker Engine (minimalno verzija 20.10)
3. Instaliran minikube (minimalno verzija 1.25) ili pristup Kubernetes klasteru
4. Instaliran kubectl (verzija kompatibilna s vašim Kubernetes klasterom)
5. Git klijent
6. Tekstualni editor po izboru (VSCode, Sublime Text, itd.)
7. Instalirane sigurnosne alate:
 - Trivy (za skeniranje ranjivosti u Docker slikama)
 - Kubesec (za analizu sigurnosnih konfiguracija u Kubernetes manifestima)
 - Cosign (za potpisivanje i verifikaciju Docker slika)
8. OpenSSL za generiranje i upravljanje certifikatima i ključevima

Priprema razvojnog okruženja

1. Klonirajte repozitorij s početnim kodom za vježbu:

```
git clone https://github.com/Gl1tc6/distrbuted_app_lab.git
```

```
cd distributed_app_lab
```

2. Pokrenite minikube klaster (preskočite ovaj korak ako već imate pristup Kubernetes klasteru):

```
minikube start --driver=docker --memory=4096 --cpus=2
```

3. Upoznajte se sa strukturom projekta:

/distributed-app-upgrade-lab

- |— /frontend # React.js frontend aplikacija
- |— /backend # Node.js backend API
- |— /auth-service # Go servis za autentifikaciju
- |— /db # Skripte za inicijalizaciju baze podataka
- |— /k8s # Kubernetes konfiguracijske datoteke
 - | |— /base # Zajednički resursi
 - | |— /overlays # Preklapanja za različite okoline
 - | |— /security # Sigurnosne konfiguracije i politike
- |— /security # Sigurnosni skripte i alati
 - | |— /certs # Certifikati i konfiguracija za TLS
 - | |— /scan # Postavke i skripte za sigurnosno skeniranje
 - | |— /rbac # RBAC konfiguracije za Kubernetes
- |— README.md # Upute za vježbu

Inicijalni deployment aplikacije s integracijom sigurnosnih kontrola

1. Generirajte i postavite TLS certifikate za sigurnu komunikaciju:

```
cd security/certs
./generate-certs.sh
kubectl create secret tls app-tls --cert=server.crt --key=server.key
```

2. Izgradite Docker slike za sve komponente i skenirajte ih za ranjivosti:

```
# Izgradnja slike
docker build -t frontend:v1 ./frontend
docker build -t backend:v1 ./backend
docker build -t auth-service:v1 ./auth-service
```

Skeniranje slike za sigurnosne ranjivosti

```
trivy image frontend:v1
trivy image backend:v1
trivy image auth-service:v1
```

3. Potpisivanje slika za osiguravanje integriteta:

```
# Generiranje ključeva za potpisivanje
cosign generate-key-pair
```

Potpisivanje slike

```
cosign sign --key cosign.key frontend:v1
```

```
cosign sign --key cosign.key backend:v1  
cosign sign --key cosign.key auth-service:v1
```

4. Primjenite RBAC konfiguracije za kontrolu pristupa:

```
kubectl apply -f security/rbac/
```

5. Analizirajte Kubernetes manifeste za sigurnosne propuste:

```
kubesec scan k8s/base/*.yaml
```

6. Primjenite inicijalne Kubernetes konfiguracije s mrežnim politikama:

```
kubectl apply -k k8s/overlays/dev  
kubectl apply -f k8s/security/network-policies.yaml
```

7. Provjerite status deploymenata i sigurnosnih politika:

```
kubectl get deployments  
kubectl get pods  
kubectl get services  
kubectl get networkpolicies  
kubectl get psp # PodSecurityPolicies
```

8. Pristupite aplikaciji preko HTTPS (prema uputi koja se prikazuje kada izvršite):

```
minikube service frontend --url
```

Rješenje vježbe (do 5000 znakova)

1. Dizajn sigurnog sustava nadogradnje

Za naš sustav nadogradnje implementirat ćemo kombinaciju Rolling Update i Blue-Green Deployment strategija [13] s integracijom sigurnosnih kontrola u svaku fazu. Ključni elementi našeg sustava nadogradnje su:

1. **Sigurno verzioniranje Docker slika** - koristit ćemo semantičko verzioniranje (semantic versioning) za oznake Docker slika i osigurati provjerljiv integritet kroz potpisivanje slika
2. **Konfiguracija resursa u Kubernetes** - koristit ćemo Kubernetes Deployment resurse s definiranim strategijama nadogradnje i sigurnosnim kontekstima (securityContext)
3. **Upravljanje stanjem aplikacije** - koristit ćemo StatefulSet za komponente sa stanjem (PostgreSQL) uz enkripciju podataka u mirovanju

4. **Migracije baze podataka** - implementirat ćemo sustav za upravljanje migracijama baze podataka s pravilnom kontrolom pristupa
5. **Neprekidno sigurnosno skeniranje** - automatsko skeniranje svih artefakata prije, tijekom i nakon procesa nadogradnje
6. **Audit logging** - bilježenje svih aktivnosti tijekom procesa nadogradnje zbog usklađenosti s regulativom i naknadne analize
7. **Upravljanje tajnama** - sigurno upravljanje osjetljivim podacima tijekom procesa nadogradnje

Implementacija sigurne Rolling Update strategije

Rolling Update strategija omogućuje postupnu zamjenu instanci stare verzije s instancama nove verzije, čime se minimizira vrijeme nedostupnosti aplikacije [14]. Ovu strategiju ćemo koristiti za komponente bez stanja (frontend, backend API, servis za autentifikaciju) uz dodatne sigurnosne kontrole.

Otvorite datoteku `k8s/base/backend-deployment.yaml` i dodajte konfiguraciju za siguran Rolling Update:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    annotations:
      seccomp.security.alpha.kubernetes.io/pod: 'runtime/default'
  spec:
    # Verifikacija integriteta slike
    imagePullSecrets:
      - name: registry-credentials
    containers:
```

```
- name: backend
  image: backend:v1
  imagePullPolicy: Always
  ports:
  - containerPort: 3000
  # Sigurnosni kontekst - princip najmanje privilegije
  securityContext:
    runAsNonRoot: true
    runAsUser: 10001
    allowPrivilegeEscalation: false
    capabilities:
      drop:
      - ALL
    readOnlyRootFilesystem: true
  readinessProbe:
    httpGet:
      path: /health
      port: 3000
      scheme: HTTPS # Korištenje HTTPS za health provjere
    initialDelaySeconds: 5
    periodSeconds: 10
  livenessProbe:
    httpGet:
      path: /health
      port: 3000
      scheme: HTTPS
    initialDelaySeconds: 15
    periodSeconds: 20
  # Volumeni za privremene zapise
  volumeMounts:
  - mountPath: /tmp
    name: temp-volume
  # Sigurne varijable okoline
  envFrom:
  - secretRef:
      name: backend-secrets
  # Inicijalni container za provjeru ranjivosti
  initContainers:
  - name: security-scan
```

```

    image: aquasec/trivy:latest
    args: ["filesystem", "/app", "--severity", "HIGH,CRITICAL", "--exit-code", "1"]
    volumeMounts:
      - mountPath: /app
        name: app-volume
  volumes:
    - name: temp-volume
      emptyDir: {}
    - name: app-volume
      emptyDir: {}

```

Primijetite više sigurnosnih poboljšanja u ovoj konfiguraciji:

- securityContext koji ograničava privilegije kontejnera
- HTTPS za health provjere
- Inicijalni kontejner koji skenira kod za ranjivosti prije pokretanja
- Sigurno upravljanje tajnama kroz Kubernetes Secret resurse
- Annotations za primjenu seccomp profila

Implementacija sigurnih migracija baze podataka

Za upravljanje migracijama baze podataka koristit ćemo obrazac "Inicijalni kontejner" (Init Container) [15] koji će izvršavati migracije prije pokretanja aplikacijskih kontejnera, uz dodatne sigurnosne kontrole za zaštitu osjetljivih podataka:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  # ...
  template:
    # ...
    spec:
      # Service account s ograničenim pravima za migracije
      serviceAccountName: db-migrations-sa
      initContainers:
        - name: db-migrations
          image: backend:v1
          command: ["npm", "run", "migrate"]
          # Sigurnosni kontekst za kontejner za migracije
          securityContext:
            runAsNonRoot: true

```

```

    runAsUser: 10001
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
    env:
      - name: DATABASE_URL
        valueFrom:
          secretKeyRef:
            name: db-credentials
            key: url
    # Omogućavanje audit loga za migracije
    - name: ENABLE_AUDIT
      value: "true"
    - name: AUDIT_LOG_PATH
      value: "/logs/migration-audit.log"
    volumeMounts:
      - name: audit-logs
        mountPath: /logs
    # Provjera uspješnosti migracije
    readinessProbe:
      exec:
        command:
          - "/bin/sh"
          - "-c"
          - "test -f /tmp/migration-success"
        initialDelaySeconds: 5
        periodSeconds: 2
    # Kontejner za validaciju sigurnosti migracija
    - name: security-
```

Implementacija vraćanja na prethodnu verziju (rollback)

Za implementaciju mehanizma vraćanja na prethodnu verziju, iskoristit ćemo mogućnosti Kubernetesa za revizije Deploymenata. Stvorite skriptu `rollback.sh` koja će omogućiti jednostavno vraćanje na prethodnu verziju.

```

#!/bin/bash
SERVICE=$1
```

```

if [ -z "$SERVICE" ]; then
    echo "Usage: $0 <service-name>"
    exit 1
fi

# Provjera povijesti deploymenta
echo "Deployment history for $SERVICE:"
kubectl rollout history deployment/$SERVICE

# Vraćanje na prethodnu verziju
echo "Rolling back $SERVICE to previous version..."
kubectl rollout undo deployment/$SERVICE

# Provjera statusa
echo "Checking rollout status..."
kubectl rollout status deployment/$SERVICE

```

2. Testiranje sigurnog sustava nadogradnje

Za testiranje našeg sigurnog sustava nadogradnje, implementirat ćemo novu verziju backend komponente s dodatnom funkcionalnošću, uz punu integraciju sigurnosnih kontrola u postupak nadogradnje.

1. Modificirajte backend aplikaciju dodavanjem nove endpoint funkcionalnosti s ispravnom validacijom ulaza i autorizacijom:

```

// Novi endpoint u backend/src/app.js s dodatnim sigurnosnim kontrolama
app.get('/api/v2/users', [authenticateToken, authorizeAdmin, rateLimiter], async (req, res)
=> {
    try {
        // Validacija parametara upita za sprječavanje SQL injekcije
        const limit = parseInt(req.query.limit) || 10;
        const offset = parseInt(req.query.offset) || 0;
        if (limit > 100) return res.status(400).json({ error: 'Limit must be less than 100' });

        // Parametrizirani upit za sprječavanje SQL injekcije
        const users = await db.query(
            'SELECT id, username, email, created_at FROM users LIMIT $1 OFFSET $2',
            [limit, offset]
        );

        // Sanitizacija podataka prije vraćanja klijentu
        const sanitizedUsers = users.rows.map(user => ({

```

```

    id: user.id,
    username: sanitizeHtml(user.username),
    email: sanitizeHtml(user.email),
    created_at: user.created_at
  }));

// Dodavanje sigurnosnih zaglavlja odgovoru
res.set({
  'Content-Security-Policy': "default-src 'self'",
  'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',
  'X-Content-Type-Options': 'nosniff',
  'X-Frame-Options': 'DENY',
  'X-XSS-Protection': '1; mode=block'
});

res.json({
  version: 'v2',
  data: sanitizedUsers,
  metadata: {
    count: sanitizedUsers.length,
    timestamp: new Date()
  }
});
} catch (err) {
  console.error(err);
  // Sigurno rukovanje pogreškama bez otkrivanja internih detalja
  res.status(500).json({ error: 'Internal server error' });
}
});

// Middleware za autorizaciju administratora
function authorizeAdmin(req, res, next) {
  if (!req.user || req.user.role !== 'admin') {
    return res.status(403).json({ error: 'Insufficient privileges' });
  }
  next();
}

// Middleware za ograničavanje stope zahtjeva

```

```
const rateLimiter = rateLimit({  
  windowMs: 15 * 60 * 1000, // 15 minuta  
  max: 100, // ograničenje na 100 zahtjeva po IP adresi  
  message: { error: 'Too many requests, please try again later' }  
});
```

2. Ažurirajte Dockerfile da uključuje sigurnosne najbolje prakse:

```
FROM node:16-alpine AS builder  
WORKDIR /app  
COPY package*.json ./  
# Provjera integriteta paketa  
COPY .npmrc ./  
RUN npm ci --only=production  
COPY . .  
# Statička analiza koda  
RUN npm run lint  
# Sigurnosno testiranje  
RUN npm audit --production  
  
# Multi-stage build za minimiziranje površine napada  
FROM node:16-alpine  
# Korištenje neprivilegirane korisničke uloge  
RUN addgroup -g 1001 -S nodejs && adduser -u 1001 -S nodeuser -G nodejs  
WORKDIR /app  
# Kopiranje samo potrebnih datoteka  
COPY --from=builder --chown=nodeuser:nodejs /app/node_modules /app/node_modules  
COPY --from=builder --chown=nodeuser:nodejs /app/src /app/src  
COPY --from=builder --chown=nodeuser:nodejs /app/package.json /app/  
  
# Konfiguracija sigurnosti  
ENV NODE_ENV=production  
USER nodeuser  
EXPOSE 3000  
  
# Uklanjanje nepotrebnih dijagnostičkih poruka  
CMD ["node", "--no-deprecation", "--no-warnings", "src/index.js"]
```

3. Izgradite novu verziju Docker slike i izvršite sigurnosne provjere:

```
# Izgradnja nove verzije  
docker build -t backend:v2 ./backend
```

Skeniranje za ranjivosti

```
trivy image --severity HIGH,CRITICAL backend:v2
```

Potpisivanje slike za osiguranje integriteta

```
cosign sign --key cosign.key backend:v2
```

4. Izvršite sigurnosni pregled Kubernetes manifesta prije ažuriranja:

Analiza sigurnosti konfiguracijskog YAML-a

```
kubeseccan k8s/overlays/dev/backend-deployment.yaml
```

Primjena security fix-eva ako su potrebni

```
kubectl apply -f k8s/security/psp-fix.yaml
```

5. Ažurirajte Kubernetes manifest za korištenje nove verzije s poboljšanim sigurnosnim kontrolama:

Ažuriranje slike u deploymentu

```
kubectl set image deployment/backend backend=backend:v2
```

Primjena mrežnih sigurnosnih politika za novu verziju

```
kubectl apply -f k8s/security/backend-v2-network-policy.yaml
```

Provjera statusa nadogradnje

```
kubectl rollout status deployment/backend
```

6. Testirajte novu funkcionalnost s autentikacijom i autorizacijom:

Dohvaćanje URL-a backend API-ja i admin tokena

```
BACKEND_URL=$(minikube service backend --url)
```

```
ADMIN_TOKEN=$(curl -s -X POST $BACKEND_URL/api/auth/login -d
```

```
'{"username":"admin","password":"****"}' -H "Content-Type: application/json" | jq -r '.token')
```

Testiranje novog endpointa

```
curl -H "Authorization: Bearer $ADMIN_TOKEN" "$BACKEND_URL/api/v2/users?limit=5"
```

Testiranje sigurnosne validacije

```
curl -H "Authorization: Bearer $ADMIN_TOKEN" "$BACKEND_URL/api/v2/users?limit=1000" # Trebalo bi vratiti grešku
```

7. Izvršite penetracijski test nove verzije:


```
# Pokrenite OWASP ZAP skener
./security/scan/run-zap-scan.sh $BACKEND_URL
```

```
# Provjerite izvješće o ranjivostima
cat security/scan/reports/backend-v2-scan-report.html
```

8. Simulirajte problem s novom verzijom i testirajte siguran povratak na prethodnu verziju:

```
# Sigurni povratak na prethodnu verziju
./secure-rollback.sh backend "Performance issues" "admin@example.com"
```

```
# Provjera da stari endpoint i dalje radi
curl -H "Authorization: Bearer $TOKEN" $BACKEND_URL/api/v1/users
```

```
# Provjera sigurnosnih postavki nakon rollbacka
kubectl describe deployment backend | grep -A 10 "Security Context"
```

9. Provedite audit procesa nadogradnje i vraćanja:

```
# Pregled audit logova nadogradnje
kubectl logs -l app=audit-logger
```

```
# Analiza sigurnosnog stanja klastera nakon promjena
./security/analyze-cluster-security.sh
```

3. Implementacija Blue-Green strategije za frontend

Za frontend komponentu implementirat ćemo Blue-Green strategiju nadogradnje [12] koja omogućuje trenutno prebacivanje prometa s jedne verzije na drugu:

1. Kreirajte dvije verzije deploymenta za frontend:

```
# k8s/overlays/dev/frontend-blue.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
      version: blue
```

```
template:
  metadata:
    labels:
      app: frontend
      version: blue
  spec:
    containers:
      - name: frontend
        image: frontend:v1
```

k8s/overlays/dev/frontend-green.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: frontend-green

spec:

replicas: 0 *# Inicijalno bez replika*

selector:

matchLabels:

app: frontend

version: green

template:

metadata:

labels:

app: frontend

version: green

spec:

containers:

- name: frontend

image: frontend:v2 *# Nova verzija*

2. Kreirajte Service za preusmjeravanje prometa:

k8s/overlays/dev/frontend-service.yaml

apiVersion: v1

kind: Service

metadata:

name: frontend

spec:

selector:

```
app: frontend
version: blue # Inicijalno usmjeravanje na plavu verziju
ports:
- port: 80
targetPort: 80
```

3. Implementirajte skriptu za prebacivanje prometa:

```
#!/bin/bash
# switch-frontend.sh

TARGET=$1

if [ "$TARGET" != "blue" ] && [ "$TARGET" != "green" ]; then
    echo "Usage: $0 <blue|green>"
    exit 1
fi

# Scale up target deployment if needed
kubectl scale deployment frontend-$TARGET --replicas=2

# Update service selector
kubectl patch service frontend -p '{"spec":{"selector":{"version":"$TARGET"}}}'

# Scale down other deployment
OTHER="blue"
if [ "$TARGET" == "blue" ]; then
    OTHER="green"
fi
kubectl scale deployment frontend-$OTHER --replicas=0

echo "Switched frontend traffic to $TARGET deployment"
```

Zaključak (do 1000 znakova)

Kroz ovu laboratorijsku vježbu naučili smo kako dizajnirati i implementirati sustav nadogradnje za raspodijeljenu aplikaciju koristeći moderne tehnologije kontejnerizacije i orkestracije. Analizirali smo različite strategije nadogradnje, uključujući Rolling Update i Blue-Green Deployment, te njihovu primjenu za različite komponente raspodijeljenog sustava.

Razumjeli smo značaj pažljivog planiranja procesa nadogradnje, posebice kada je riječ o komponentama sa stanjem poput baza podataka. Implementirali smo mehanizme za migraciju baze podataka i vraćanje na prethodnu verziju u slučaju problema, što je ključno za održavanje visoke dostupnosti sustava tijekom nadogradnje.

Ova znanja i vještine izuzetno su vrijedni u modernom DevOps okruženju gdje je kontinuirana isporuka softvera postala standard, a raspodijeljene aplikacije prevladavajući arhitekturni obrazac.

Literatura

- [1] Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.
- [2] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). *Borg, Omega, and Kubernetes*. ACM Queue, 14(1), 70-93.
- [3] Fowler, M. (2010). *Blue Green Deployment*.
<https://martinfowler.com/bliki/BlueGreenDeployment.html>
- [4] Tanenbaum, A. S., & Van Steen, M. (2016). *Distributed Systems: Principles and Paradigms*. Pearson.
- [5] Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.
- [6] Merkel, D. (2014). *Docker: Lightweight Linux Containers for Consistent Development and Deployment*. Linux Journal, 2014(239).
- [7] Kubernetes dokumentacija (2023). *Kubernetes Overview*.
<https://kubernetes.io/docs/concepts/overview/>
- [8] Humble, J., & Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley.
- [9] Sayfan, G. (2017). *Mastering Kubernetes*. Packt Publishing Ltd.
- [10] Kubernetes dokumentacija (2023). *Performing a Rolling Update*.
<https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/>
- [11] Kubernetes dokumentacija (2023). *Init Containers*.
<https://kubernetes.io/docs/concepts/workloads/pods/init-containers/>
- [12] Humble, J. (2017). *Continuous Delivery vs. Continuous Deployment*.
<https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>