

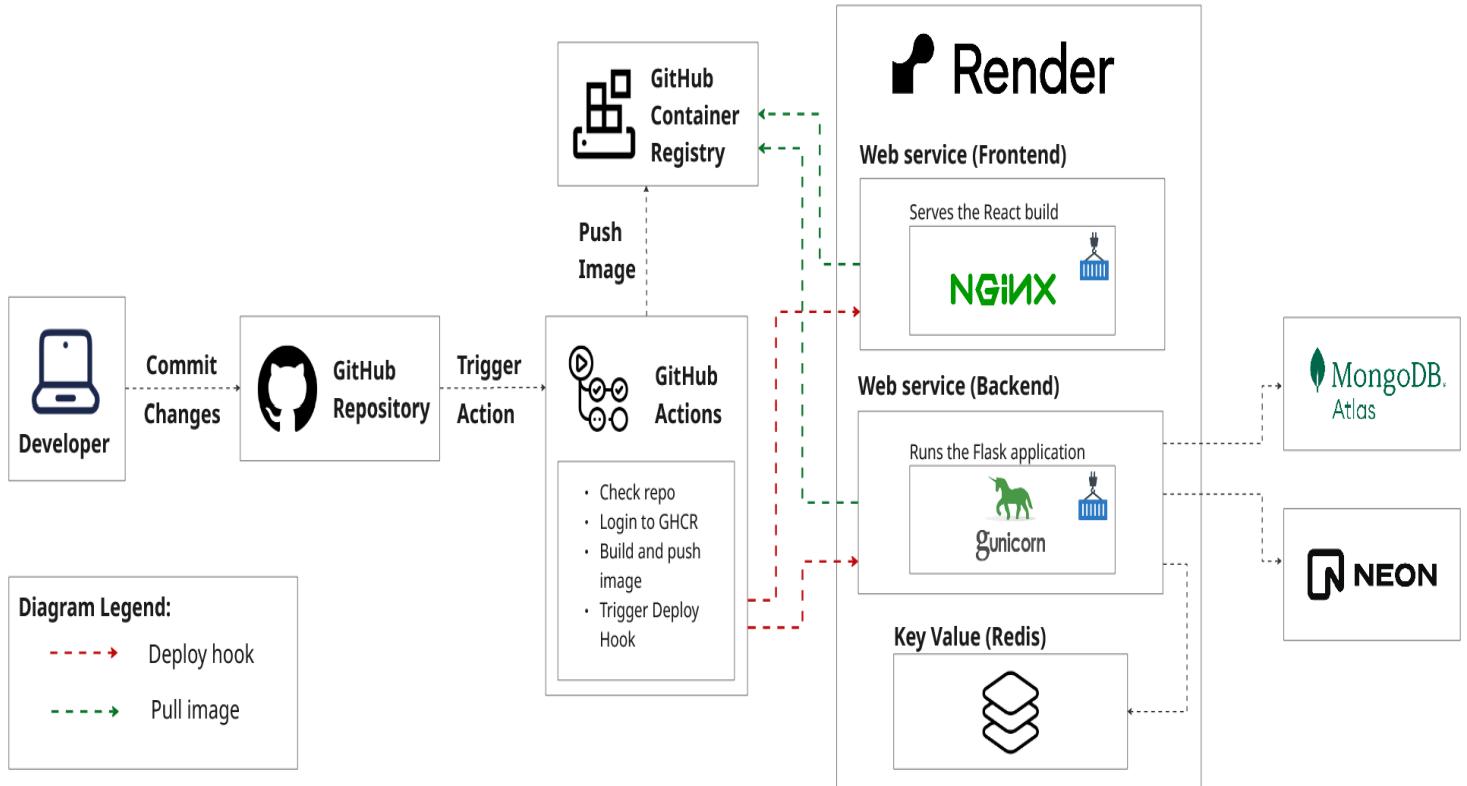
DevOps ДРС

Садржај

Задатак.....	2
Алати и платформе.....	3
База података	3
CI/CD – GitHub Actions workflow	3
Где се пише?.....	4
Када се покреће?	4
Шта треба да ради?	4
Циљ	4
Workflow за frontend	5
Workflow за backend	6
Render	7
Web Service	7
Конфигурација променљивих окружења	7
Key Value.....	7
Пример.....	8
Dockerfile.....	8
Frontend	8
Backend	9
Ток рада	9

Задатак

Ваш задатак је да прилагодите постојеће развојне **Dockerfile**-ове (*frontend* и *backend*) за производно окружење. Потребно је имплементирати **CI/CD workflow** преко **GitHub Actions** који ће аутоматски **build**-овати *Docker* слике, **push**-овати их на **GitHub Container Registry** и сигнализирати **redeploy** на **Render Web Service**.



Алати и платформе

Препоручени алати и платформе:

- **Docker**,
- **GitHub Actions** (CI/CD платформа интегрисана у *GitHub*),
- **GitHub Container Registry** (чување производњачких *Docker* слика унутар *GitHub* екосистеме),
- **Render Web Service** (сервис за *deploy* апликације),
- **Render Deploy Hook** (автоматски *redeploy*).

Напомена: Уместо *GHCR* може се користити *DockerHub*, а уместо *Render* платформе било која друга платформа за *deploy* по избору.

База података

Препоручене платформе за производњачке базе података:

- **Neon** (за SQL),
- **MongoDB Atlas** (за NoSQL),
- **Render Key Value** (за Redis).

Напомена: Уместо набројаних платформи за базе података можете користити било коју другу по вашем избору. Битно је да није локална база података коју сте користили за време развоја.

CI/CD – GitHub Actions workflow

Workflow представља опис целокупног аутоматизованог процеса, који дефинише:

- Када се аутоматизација покреће,
- Које кораке извршава,
- У ком редоследу,
- На којој машини (*runner*).

У оквиру једног *workflow*-а дефинишу се *jobs* (послови) и *steps* (кораци) који користе појединачне *GitHub* акције за извршавање конкретних задатака као што су *checkout* кода, *Docker build*, *Docker push* итд...

Где се пише?

Workflow се пише у YAML фајлу и налази се у директоријуму у оквиру пројекта на путањи: `.github/workflows/`. За потребе вашег задатка треба да креирате два workflow-а:

- `frontend-ci-cd.yml`
- `backend-ci-cd.yml`

Један workflow = један .yml фајл

Када се покреће?

Први workflow треба да се покрене када се одради *push* на *main* грану и само ако је дошло до промене у *frontend/*, а други када се одради *push* на *main* грану и само ако је дошло до промене у *backend/*.

Шта треба да ради?

У нашем пројекту *Github Actions workflow* има следећу улогу:

1. Преузима изворни код из репозиторијума (*checkout*),
2. Пријављује се на *GitHub Container Registry* (GHCR),
3. Гради производњску *Docker* слику,
4. Поставља (*push*) *Docker* слике на GHCR,
5. Тригерије *Render Deploy Hook*,
6. *Render* аутоматски повлачи нову *Docker* слику и рестартије сервис.

Све ово се дешава аутоматски без ручне интервенције!

Циљ

Циљ задатка није само да апликација ради локално већ да:

- има производњску *Docker* слику,
- има аутоматизован *build*,
- има аутоматизован *deploy*,
- и да се понаша као реална производњска апликација.

Workflow за frontend

Дати *workflow* представља шаблон који треба да се прилагоди вашем решењу за **frontend-ci-cd.yml**.

```
name: Build and Push Frontend Image

on:
  push:
    branches: [main]
    paths:
      - "frontend/**"

permissions:
  contents: read
  packages: write

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        # TO DO

      - name: Log in to GHCR
        # TO DO

      - name: Build and Push image
        # TO DO

      - name: Trigger Render Deploy
        # TO DO
```

Workflow за backend

Дати workflow представља шаблон који треба да се прилагоди вашем решењу за **backend-ci-cd.yml**.

```
● ● ●

name: Build and Push Backend Image

on:
  push:
    branches: [main]
    paths:
      - "backend/**"

permissions:
  contents: read
  packages: write

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        # TO DO

      - name: Log in to GHCR
        # TO DO

      - name: Build and Push image
        # TO DO

      - name: Trigger Render Deploy
        # TO DO
```

Render

Након што су *Docker* слике (за *frontend* и *backend*) објављене на GitHub Container Registry, потребно је конфигурисати *Render Web Service* који ће користити слике за покретање апликације у прдукционом окружењу.

Web Service

Кораци:

- Изабрати опцију "**New**" па "**Web Service**"
- Изабрати опцију "**Existing Image**"
 - Копирати *URL* слике са *GHCR*
 - Изабрати креденцијале и опцију за *GitHub*.
- Добијени **Deploy Hook** додати у *GitHub Secrets* и користити унутар одговарајућег *workflow-a*.

Погледати:

- <https://render.com/docs/deploying-an-image>
- <https://render.com/docs/deploying-an-image#generating-a-personal-access-token>

Кораке поновити за обе слике које треба да буду *deploy*-ване.

Конфигурација променљивих окружења

Приликом чувања променљивих окружења водити рачуна о разликама између **GitHub Secrets** (*build time*) и **Render Environment** (*runtime*).

Нпр. **VITE_API_BASE_URL** иде у *GitHub Secrets*, док **DATABASE_URL** иде у *Render Environment*.

Напомена: Све конфигурационе променљиве које су неопходне током *build* фазе (нпр. променљиве које *Vite* користи у тренутку *build*-овања апликације) морају бити дефинисане као *GitHub Secrets* и прослеђене *Docker build* процесу као *build* аргументи у оквиру *GitHub Actions workflow-a*.

Key Value

Render Key Value сервис представља Redis-компабилан *key-value store* и може се користити као замена за локално покренути Redis.

Напомена: Користе студенти који су у оквиру спецификације пројекта користили *Redis*

Пример

Дат је пример како у *Render* платформи изгледа *deploy*-вана апликација са три сервиса.

SERVICE NAME	STATUS	RUNTIME	REGION	UPDATED
flask-backend:latest	✓ Deployed	Image	Oregon	23h
redis	✓ Available	Valkey 8	Oregon	1d
react-frontend:latest	✓ Deployed	Image	Oregon	1d

Информације о покренутом сервису (дат је пример за *Flask* апликацију).

WEB SERVICE
flask-backend:latest Image Free Upgrade your instance → Connect Manual Deploy

Service ID: XXXXXXXXXX

nemanjaase / flask-backend latest → Docker слика која се повлачи са GHCR + верзија слике

https://flask-backend-latest-0iu1.onrender.com → Јавна адреса сервиса (преко које је доступан)

Dockerfile

Неопходно је развојити *Dockerfile*-ове на два типа:

1. **Dockerfile.dev** – за локално тестирање (користимо га у *docker-compose.yml*)
2. **Dockerfile.prod** – за продукцију (користимо га у *CI/CD и deployment*)

Frontend

Docker фајл за локално тестирање – покренути React Vite са *npm run dev*

Docker фајл за продукцију – користити **multi-stage build** приступ:

1. Build stage

- 1.1. Инсталација зависности (*npm ci*),
- 1.2. Build-овање Vite пројекта (*npm run build*).

2. Production stage

- 2.1. Копирање **dist/** (резултата из **Build Stage**) у **Nginx**,
- 2.2. Покретање производијског сервера.

Backend

Docker фајл за локално тестирање – покренути Flask апликацију са подразумеваним Werkzeug.

Docker фајл за продукцију – користити **multi-stage build** приступ:

1. Build stage

1.1. Инсталација Python зависности.

Напомена: Не пребацивати цело виртуелно окружење у Docker слику. Користити requirements.txt генерисан из Pipenv или Poetry (нпр. Pipenv requirements > requirements.txt). На основу генерисаног фајла инсталирати зависности.

2. Production stage

2.1. За покретање Flask апликације користити Gunicorn или неки други WSGI сервер (никако преко подразумеваног Werkzeug).

Ток рада

1. Локални развој и тестирање

- Developer мења код (*frontend* или *backend*)
 - Позива команду `docker-compose up` за локално покретање (*React + Flask + Redis + DB*)
 - Тестира функционалности, проверава да ли апликација ради како треба итд...

2. Commit i push

- Када је код функционалан и тестиран локално, прави се commit i push у `main` грану репозиторијума.

3. Аутоматски CI/CD workflow

- GitHub Actions детектује промену у репозиторијуму (ако је промена у *frontend* фолдеру онда се позива `frontend-ci-cd.yml`, а ако је у *backend* фолдеру онда се позива `backend-ci-cd.yml`).
 - Користи `Dockerfile.prod`
- Слика се *push*-ује на GHCR

4. Deploy na Render

- Render аутоматски повлачи нову верзију слике преко `Deploy Hook`-а

- Сервиси за које постоји нова верзија слике се рестартују
- Апликација је доступна са новом верзијом слике путем јавног URL-а