Poročilo

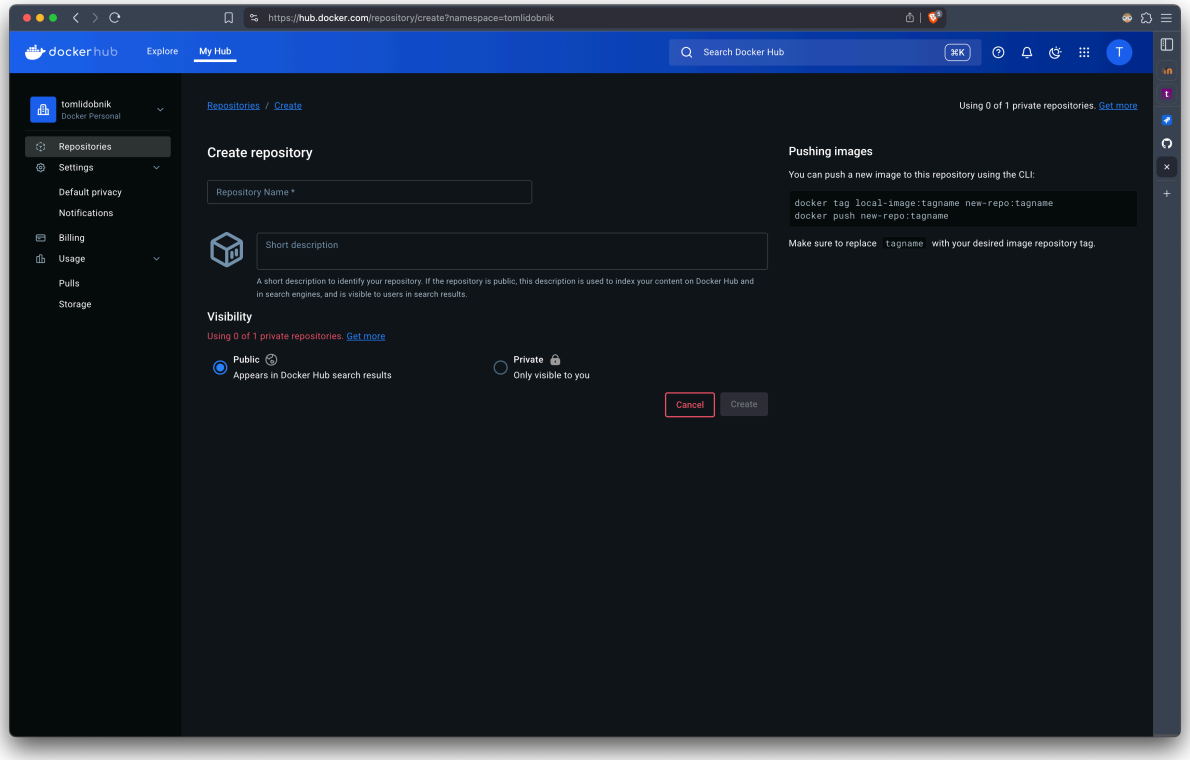
1 Dockerhub

1.1 Prijava

Za docker sem že imel ustvarjen račun, zato sem se lahko preko prijavil preko ukazne vrstice z ukazom `docker login -u UPORABNIŠKO_IME`.

1.2 Ustvarjanje repozitorija

Za čelni in zaledni del sem moral ustvariti ločena repozitorija. Najprej sem kliknil na `Create Repository` in sem izpolnil obrazec.



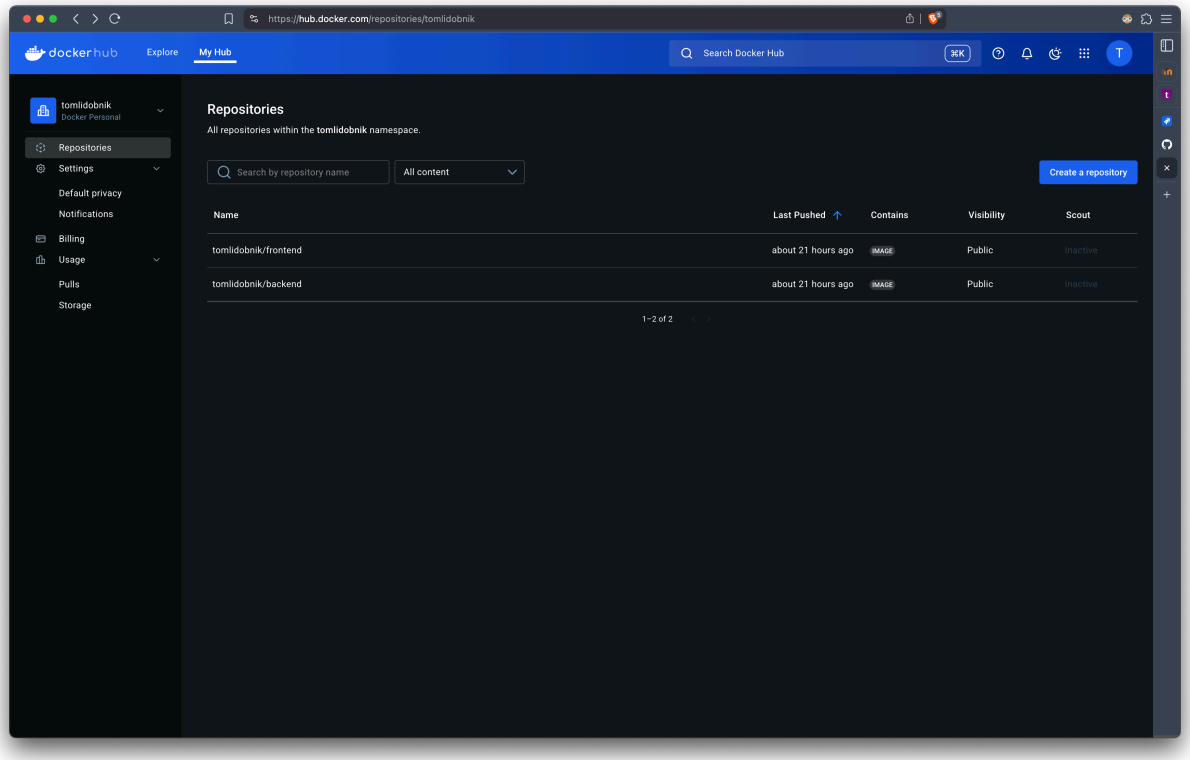
Slika 1: Ustvarjanje repozitorija na Docker Hub

1.3 Grajenje slik in posodabljanje repozitorija

Zdaj imam ustvarjen repozitorij za tako čelni del kot zaledni del. Za grajenje čelnega in zalednega dela uporabim naslednja ukaza:

```
docker buildx build --platform linux/arm64 -t tomlidobnik/backend:prod -f backend/
Dockerfile ./backend --push
docker buildx build --platform linux/arm64 -t tomlidobnik/frontend:prod -f frontend/
Dockerfile ./frontend --push
```

Ta ukaza zgradita sliko iz Dockerfilov za platformo linux/arm64 in jih nato pošljeta na repozitorij. Na naslednji sliki lahko vidim posodobljena repozitorija na Dockerhubu.



Slika 2: Posodobljen repozitorij

1.4 Nalaganje slik

Ker so docker slike javno objavljene jih lahko kdorkoli naloži z ukazom `docker pull tomlidobnik/backend:prod` za zaledni del in z ukazom `docker pull tomlidobnik/frontend:prod` za čelni del.

2 Github Actions

Za naš repozitorij smo ustvarili preprosti Github Action:

```yaml
name: Build and deploy

on:
  push:
    branches:
      - "develop"

jobs:
  deploy:
    runs-on: self-hosted

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Log in to Docker Hub
```

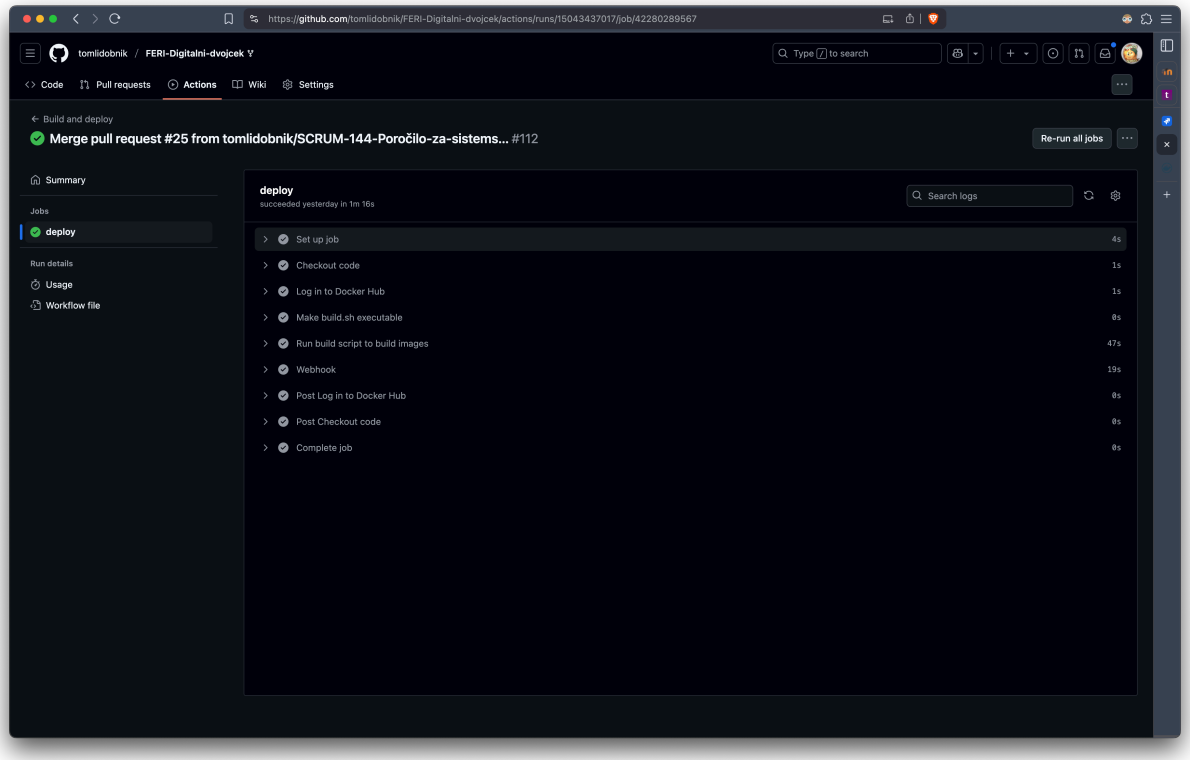
```
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_USERNAME }}
          password: ${{ secrets.DOCKER_PASSWORD }}

      - name: Make build.sh executable
        run: chmod +x ./build.sh

      - name: Run build script to build images
        env:
          DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
          DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
        run: ./build.sh

      - name: Webhook
        uses: joelwmale/webhook-action@master
        with:
          url: ${{ secrets.WEBHOOK_URL }}
          headers: >
            {
              "repository": "joelwmale/webhook-action",
              "X-Hub-Signature-256": "${{ secrets.WEBHOOK_SECRET }}"
            }
```

2.1 Opis

Prvo povemo Githubu, da naj uporabi `self-hosted` runnerja, katerega sem nastavil v nastavitvah Githuba.

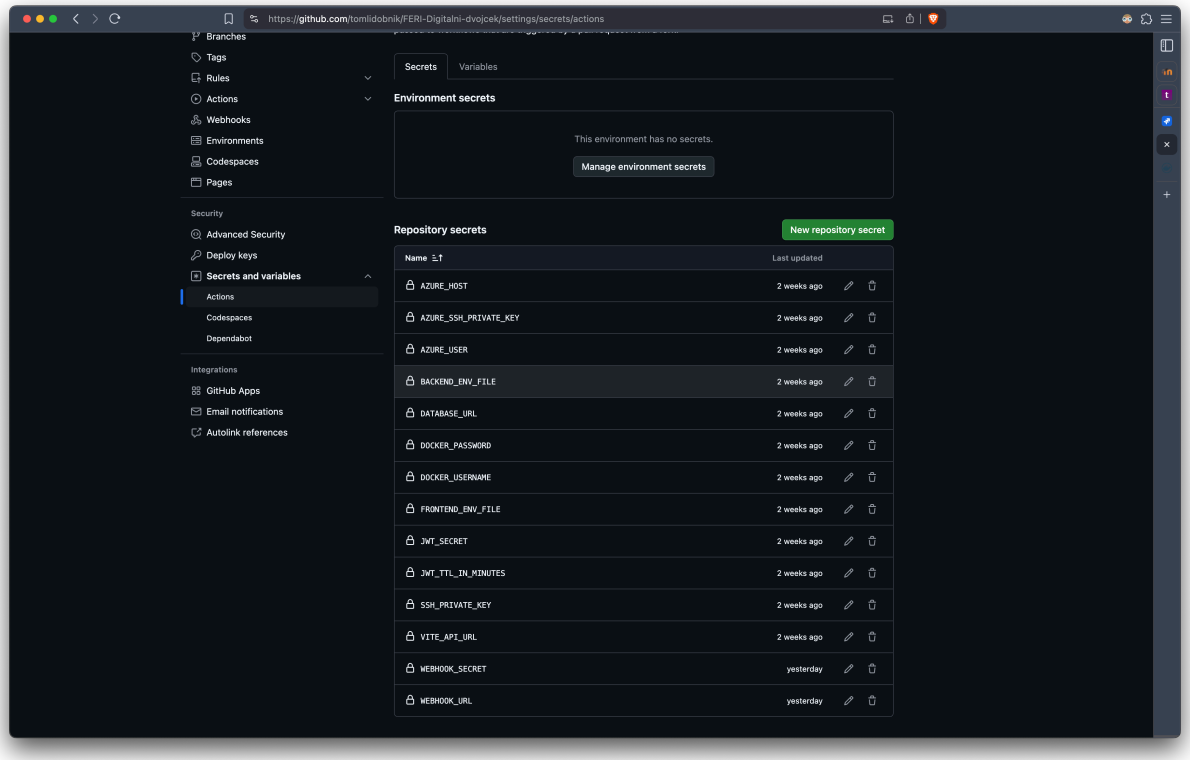
Nato je prvi korak, da se naloži najnovejša koda. Naslednji korak je, da se prijavi v Docker Hub, pri tem je uporabniško ime in geslo skrito. Potrebno ga je shraniti v Github Secrets. Naslednji korak je, da naredimo `build.sh` izvedljiv, v primeru, da ni. Naslednji korak je izvajanje skripte, ki zgradi in posodobi docker repozitorij. Na koncu se pošlje sporočilo strežniku, naj posodobi svoje lokalne slike z uporabo webhooka.

Ta Github Action se sproži ob pushanju nove kode v Github repozitorij.



Slika 3: Uspešna izvedba Github Actiona

Na spodnji sliki lahko tudi vidimo seznam Github secretov.



Slika 4: Seznam Github secretov

2.2 Opis dodatnih Github Action workflowov

Trenutno imamo action samo za develop vejo, lahko bi še dodali workflow za main vejo in feature veje. Poleg samega grajenja slike bi lahko tudi testirali delovanje naše kode (integration testi). Primer tega bi bil, da testiramo delovanje našega APIja, da bi pošiljali določene podatke in preverjali, če dobimo pričakovan odgovor. V primeru, da ne dobimo pričakovan odgovor pomeni, da koda ni bila pravilno napisana. Lahko bi tudi dodali rollback workflow: v primeru, da bi nova koda se zgradila in posodobila na strežniku, ampak bi v produkciji se našla velika napaka, ki bi onemogočila normalno uporabo naših storitev, bi se lahko napisal workflow, ki bi ob taki napaki naložil starejšo verzijo.

3 Webhook

3.1 Implementacija webhooka na strežniku

Na strežniku sem z uporabo pythona in FastAPIja ustvaril preprosti webhook. Strežnik posluša za sporočila na specifičnem URLju in nato izvede skripto, ki ustavi docker containerje, naloži posodobljene slike ter ponovno zažene docker containerje s posodobljenimi slikami.

Koda za lastno implementacijo webhooka:

```
from fastapi import FastAPI, Request, HTTPException, Header
from slowapi import Limiter
from slowapi.util import get_remote_address
from slowapi.middleware import SlowAPIMiddleware
```

```
from dotenv import load_dotenv
import hmac
import hashlib
import os
import logging
import subprocess

load_dotenv()

app = FastAPI()

logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)

limiter = Limiter(key_func=get_remote_address)
app.state.limiter = limiter
app.add_middleware(SlowAPIMiddleware)

WEBHOOK_SECRET = os.getenv("GITHUB_WEBHOOK_SECRET")

COMMANDS = """
if [ ! -d "/home/pi/FERI-Digitalni-dvojcek" ]; then
    git clone git@github.com:tomlidobnik/FERI-Digitalni-dvojcek.git /home/pi/FERI-
Digitalni-dvojcek
fi

cd /home/pi/FERI-Digitalni-dvojcek
git switch develop
git fetch origin
git reset --hard origin/develop

cd ./backend
mkcert -install
mkcert -key-file key.pem -cert-file cert.pem tom.thehomeserver.net
cd ..

docker pull tomlidobnik/backend:prod
docker pull tomlidobnik/frontend:prod

docker-compose down
docker-compose up -d --build db backend-prod frontend-prod
"""


def run_deployment(commands: str):
    try:
        logging.info("🚀 Starting deployment process...")
        result = subprocess.run(
            commands, shell=True, check=True, text=True, capture_output=True
```

```
        )
        logging.info(f"✅ Deployment successful:\n{result.stdout}")
    except subprocess.CalledProcessError as e:
        logging.error(f"❌ Deployment failed:\n{e.stderr}")
        raise


@app.post("/secretkey")
@limiter.limit("5/minute")
async def update(request: Request):
    headers = request.headers
    x_hub_signature_256 = headers.get(
        "X-Hub-Signature-256", headers.get("x-hub-signature-256")
    )
    logging.info(f"🔑 Secret loaded: {WEBHOOK_SECRET}")
    logging.info(f"🖋️ Received signature: {x_hub_signature_256}")

    if not x_hub_signature_256 or not WEBHOOK_SECRET:
        raise HTTPException(403, "Missing GitHub signature or secret")

    if not hmac.compare_digest(x_hub_signature_256, WEBHOOK_SECRET):
        raise HTTPException(403, "Invalid GitHub signature")

    try:
        run_deployment(COMMANDS)

        return {"message": "Webhook processed successfully."}

    except Exception as e:
        logging.error(f"Failed to process webhook: {e}")
        raise HTTPException(status_code=500, detail="Internal Server Error")
```

3.2 Varnostne luknje Webhookov

Pri webhookih strežik čaka na sporočilo preko določenega URLja. Zaradi tega bi lahko kdorkoli pošiljal sporočila, da posodobimo našo kodo, četudi se ni posodobila docker slika. Ker med posodabljanjem kode ustavimo našo storitev, nam bi napadalec lahko vsako minuto pošiljal takšna sporočila in nam ustavljal naše storitve. Poleg tega lahko napadalec pošilja veliko število prošenj na ta URL, kar bi upočasnilo delovanje.

Te probleme lahko rešimo na preposti način. Ko dobimo sporočilo na naš webhook URL, preverimo ali je poleg sporočila podpis in če ga ni ignoriramo sporočilo. S tem omogočimo preprosto avtentikacijo. Poleg tega bi lahko tudi preverjali IP naslov sporočila in blokirali specifične naslove. Napadalec pa bi še vedno lahko pošiljal veliko sporočil, zato lahko uporabimo rate limiter, ki bi blokiral prošnje, če so poslane prepogosto.

V naši lastni implementaciji smo uporabili preverjanje podpisa ter rate limiter. Tako napadalec ne mora preprosto poslati lažnje prošnje preko webhooka, če pa ponarediti podpis pa bo mu to zelo težko zaradi rate limiterja.