Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Dokumentacija

(ni imena?)

Študent:
Domen Hribernik
Denis Železnik
Merisa Mustajbašić

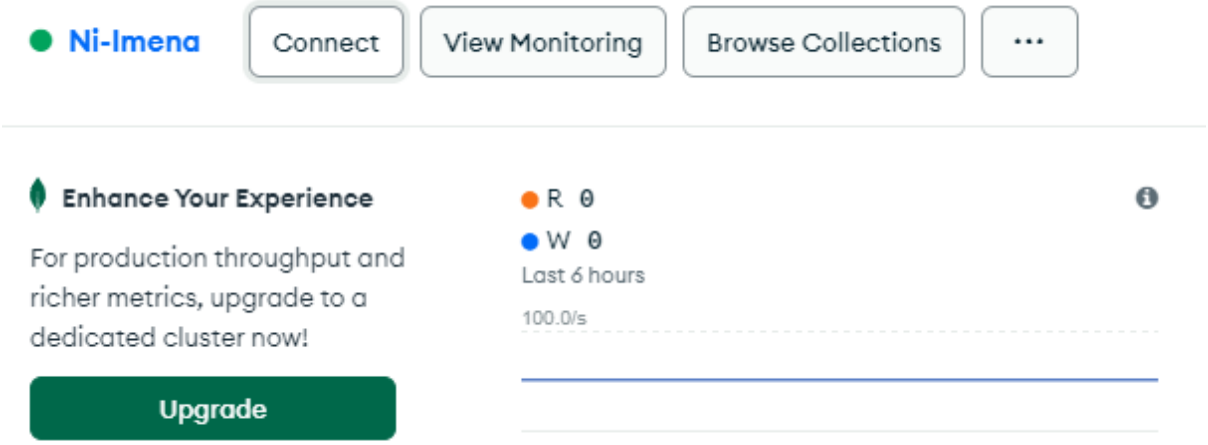
Datum in mesto:
4.06.2023, Maribor

Kazalo

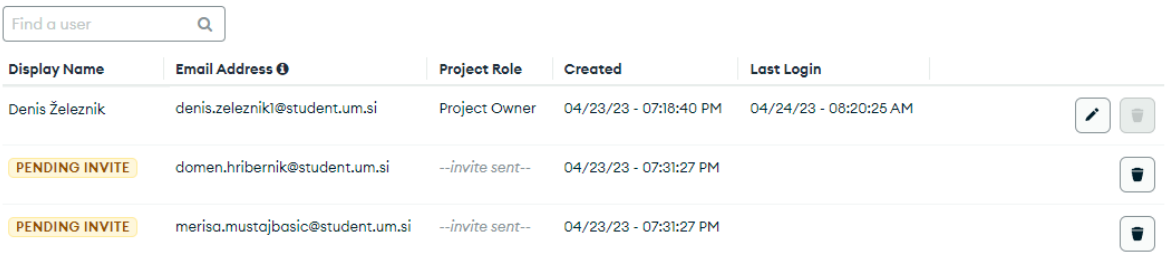
1. Namestitev v Docker-ju

1.1. Namestitev MongoDB

Najprej smo ustvarili račun na spletni strani MongoDB in nato ustvarili novo podatkovno bazo za naš projekt (Slika 1). V bazo smo dodali ustrezne kolekcije in dokumente, ki jih potrebujemo za spletno stran. Nato smo povabili še ostale člane skupine, da lahko vsi dostopamo do baze.

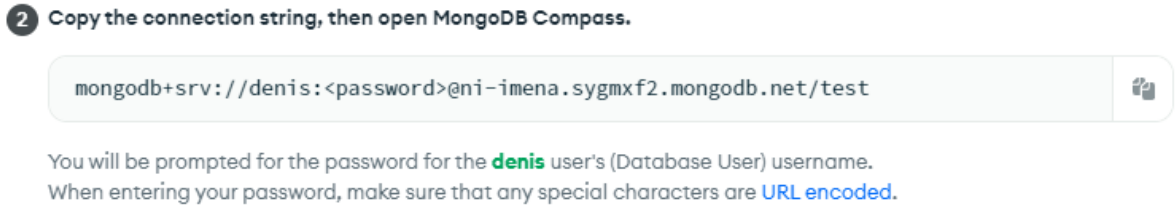


Slika 1 - Ustvarjena baza projekta

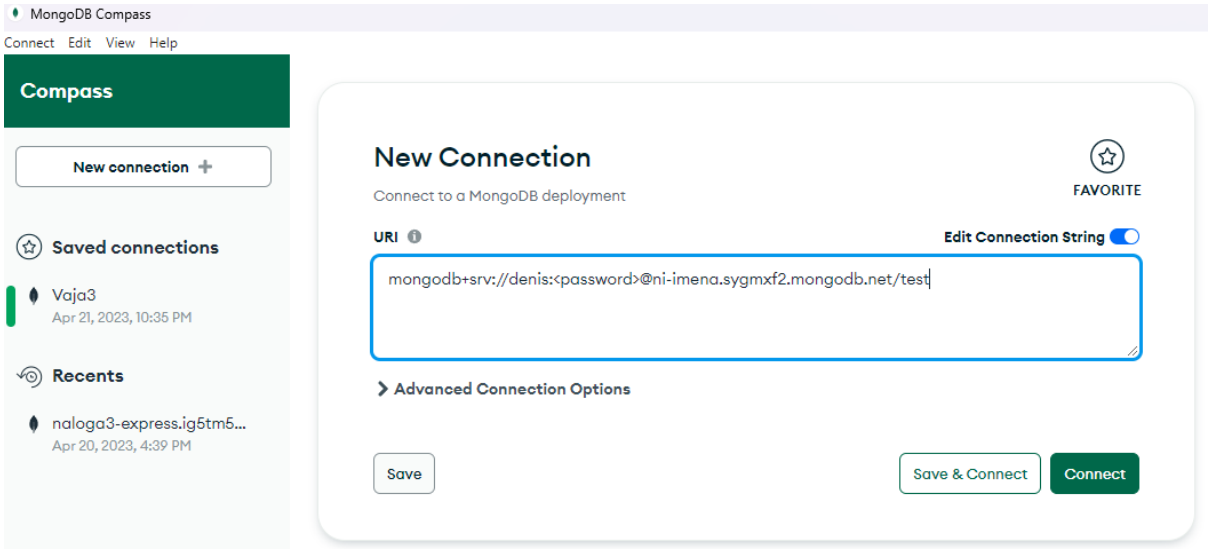


Slika 2 - povabljeni člani skupine

Naslednji korak je bil namestitev orodja MongoDBCompass, ki omogoča pregled in upravljanje z našo podatkovno bazo (Slika 3).

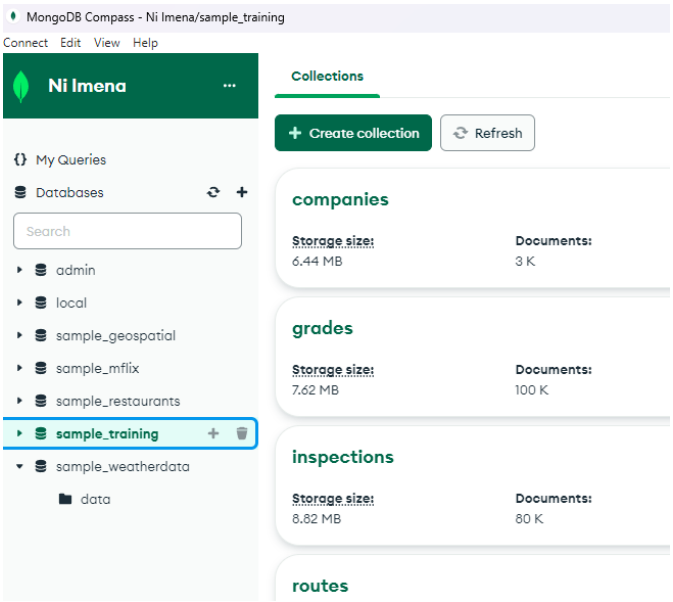


Slika 3 – link za povezavo na MongoDB compass



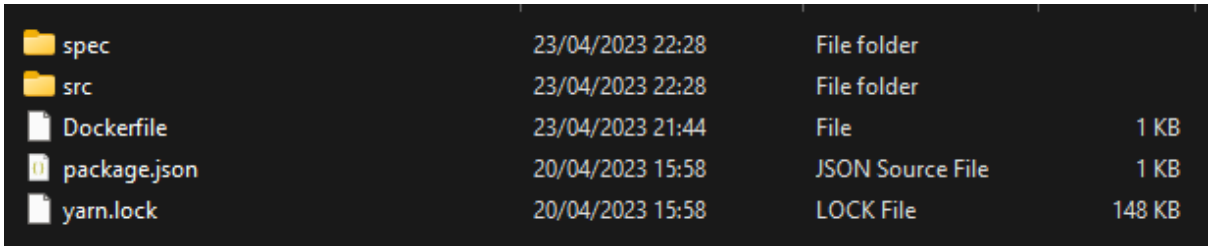
Slika 4 - povezava na MongoDB Compass

Po povezavi z bazo smo se prepričali, da je baza vidna in delujoča (Slika 5).



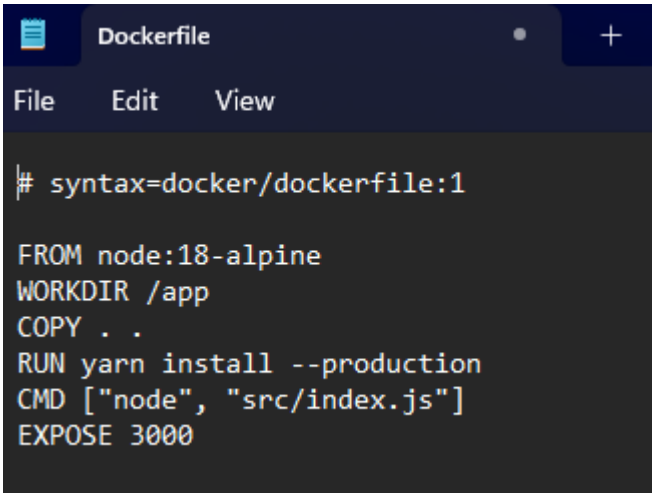
Slika 5 - vidna in delujoča baza na MongoDB Compass

1.2. Docker in GitHub



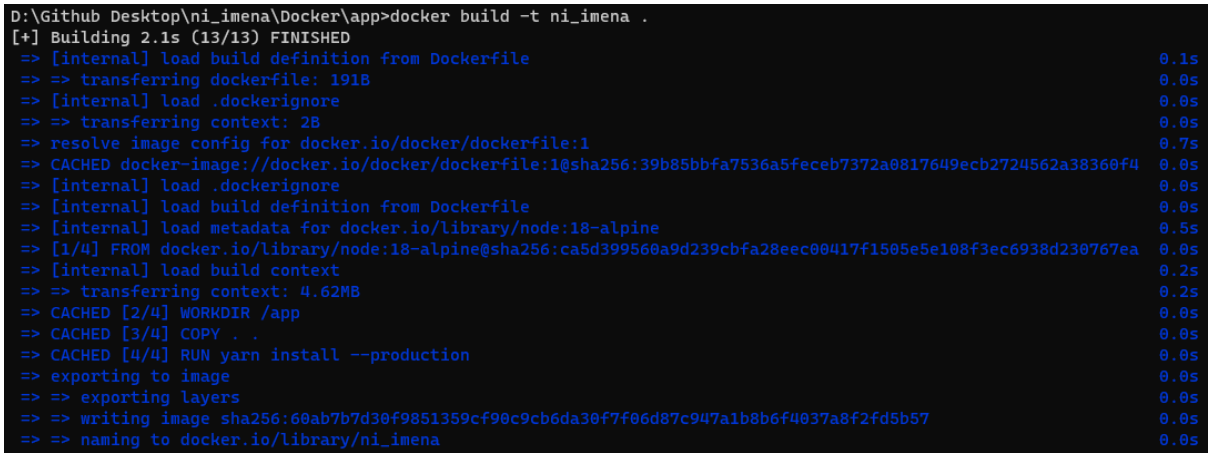
Slika 6 - kloniran projekt z GitHuba

Za nadaljevanje smo potrebovali Docker, ki smo ga namestili na naš računalnik. Po namestitvi smo klonirali naš projekt iz Githuba (Slika 6) in dodali potrebne datoteke ter Dockerfile (Slika 7 - vsebina Dockerfile datoteke) za node.js, ki je vseboval osnovna navodila za namestitev in konfiguracijo.

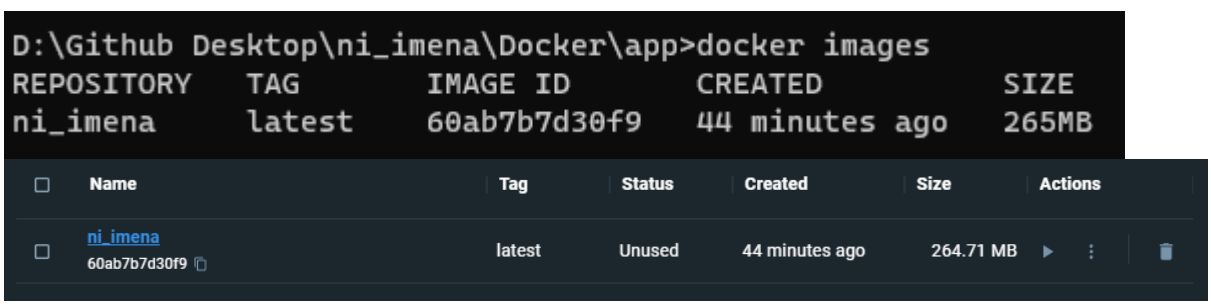


Slika 7 - vsebina Dockerfile datoteke

Nato smo izvedli ukaz za izgradnjo Docker slike (Slika 8 - izvedba ukaza za izgradnjo Docker image), ki je vsebovala našo spletno stran in vse potrebne odvisnosti. Ko je slika bila uspešno zgrajena (Slika 9 - izgrajen Docker image), smo jo uporabili za ustvarjanje Docker kontejnerja.

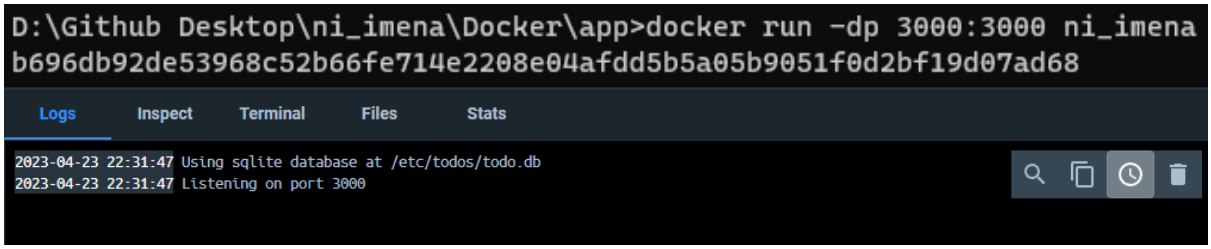


Slika 8 - izvedba ukaza za izgradnjo Docker image

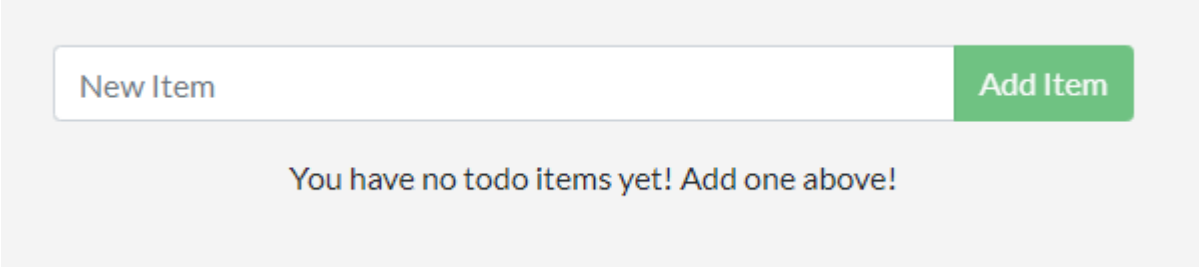


Slika 9 - izgrajen Docker image

Kontejner je bil uspešno ustvarjen in smo ga zagnali na lokalnem računalniku preko localhost:3000 (Slika 10 - izgrajen kontejner za spletno stran). Spletna stran je sedaj delujoča in dostopna preko spletnega brskalnika (Slika 11).



Slika 10 - izgrajen kontejner za spletno stran

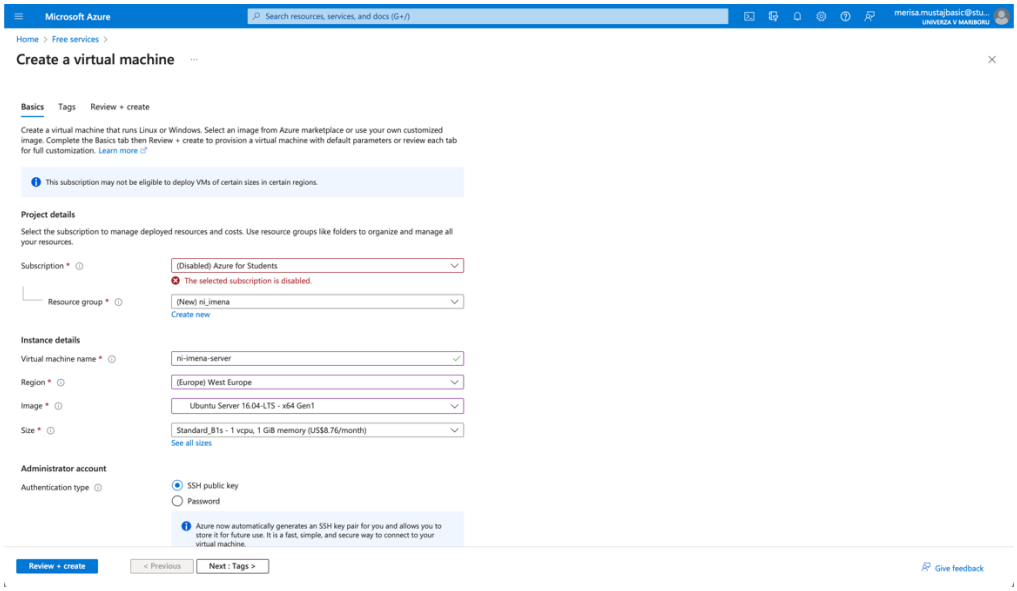


Slika 11 - delujoča spletna stran nameščena prek Docker-ja

Na koncu smo še pushali naše spremembe na Github, da lahko vsi člani naše skupine dostopajo do najnovejše različice kode.

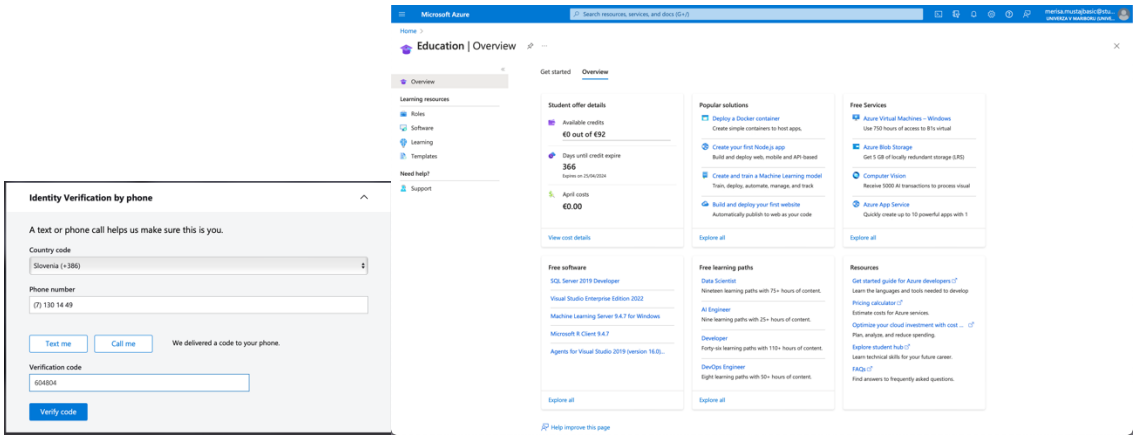
2. Vzpoztavitev Azure VM

2.1. Kreiranje VM

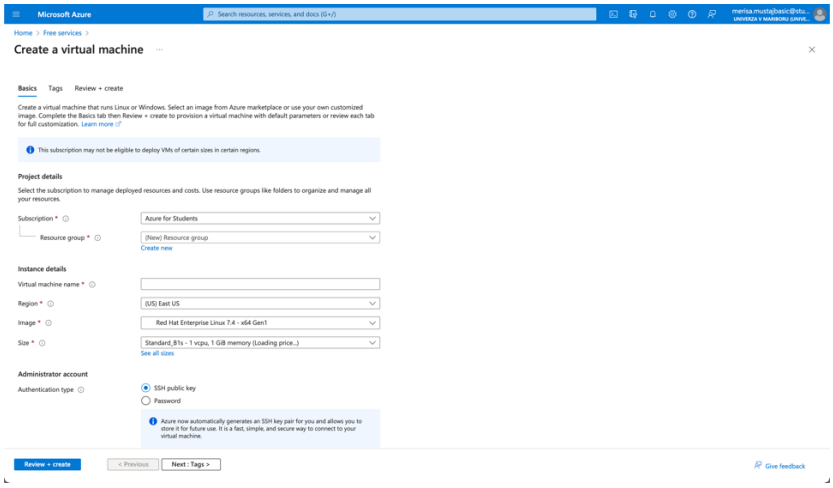


Slika 12 - napaka pri vzpoztavitvi, račun ni aktiven

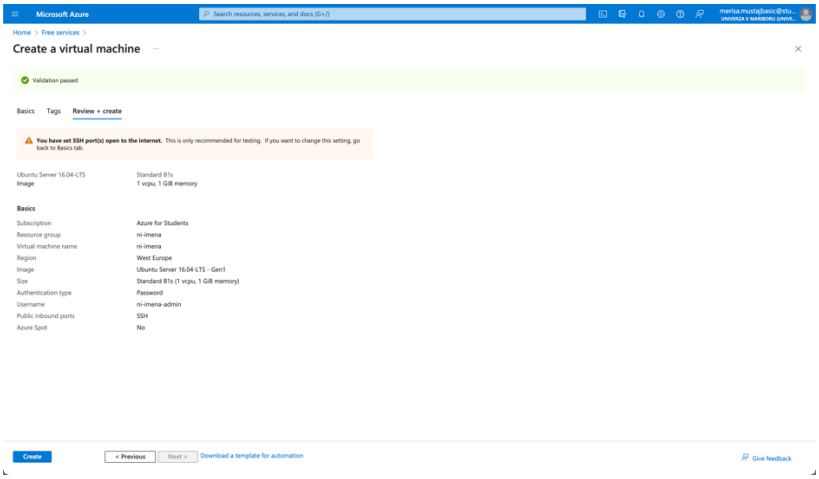
Po uspešni registraciji za brezplačnih 100 evrov storitev na Microsoft Azure (Slika 13), namestimo navidezni stroj kot je opisano v navodilima (Slika 14).



Slika 13 - unovčitev brezplačnih storitev na Microsoft Azure

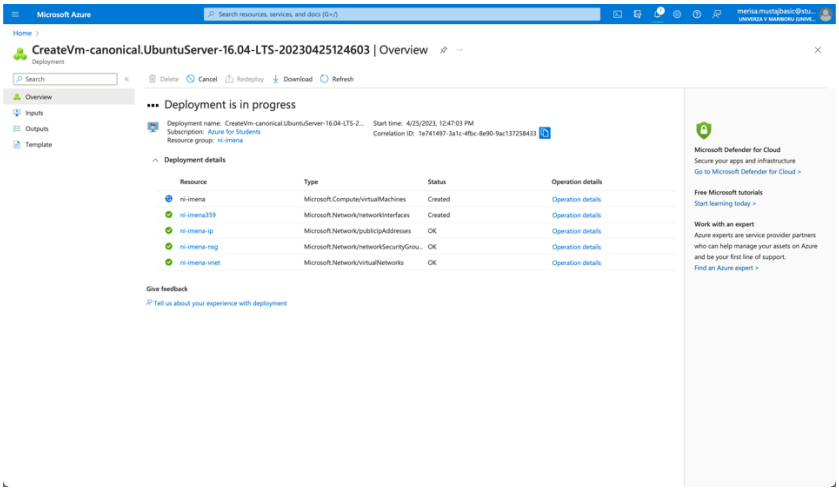


Slika 14 - specifikacije navideznega stroja

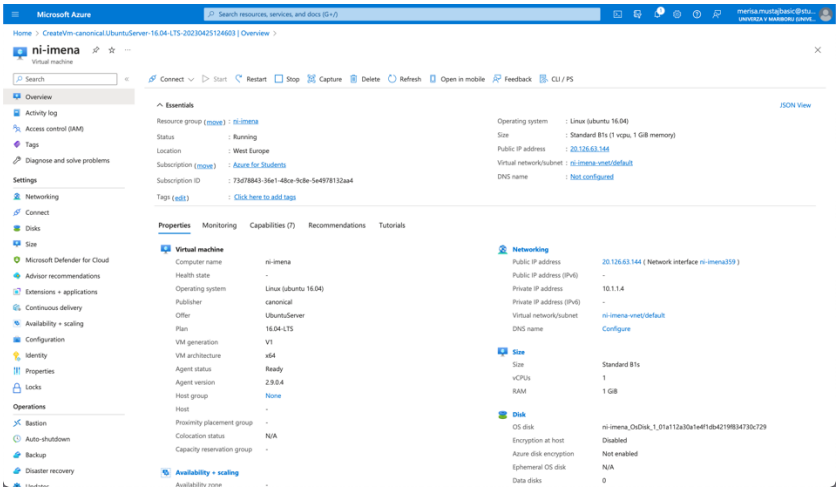


Slika 15 - overview VM-a

Ko smo preverili vse podane informacije in specifikacije, začnemo proces deploymenta našega stroja (Slika 16). Po končanem deployment-u smo uspešno naredili Linux navidezni stroj (Slika 17).

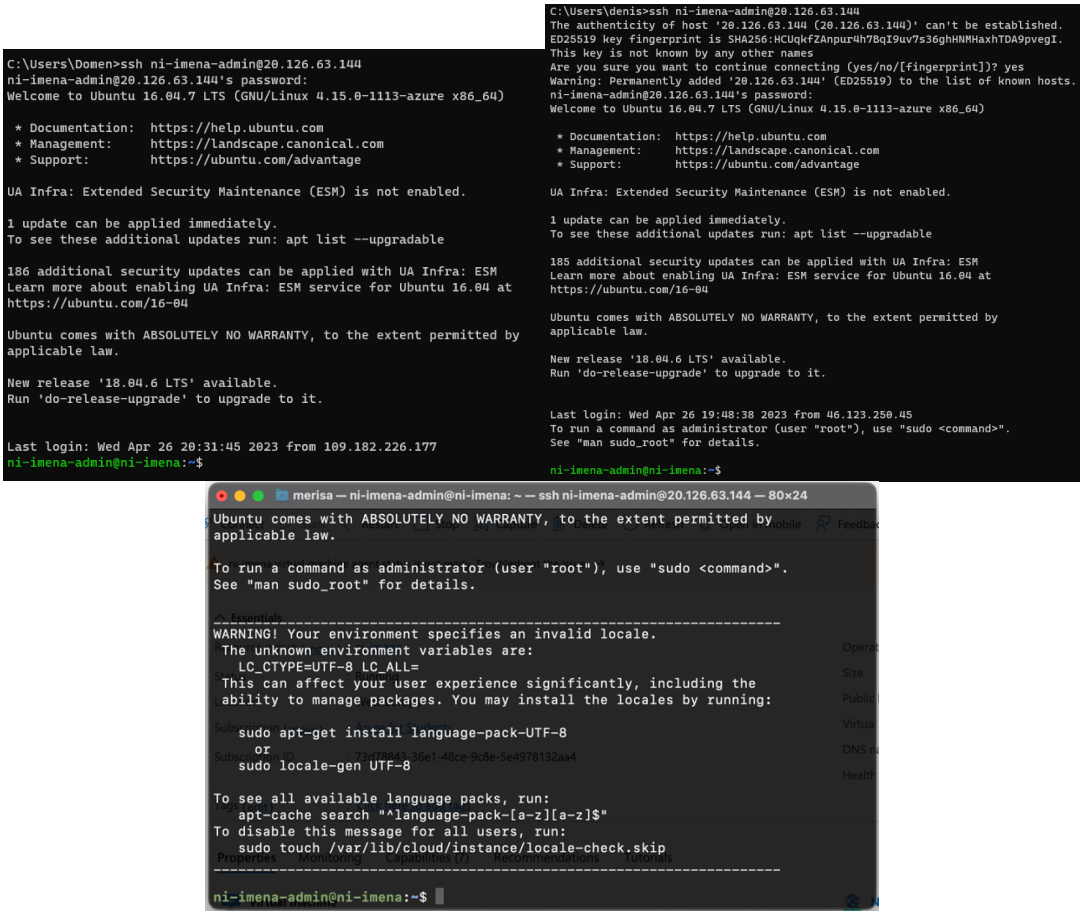


Slika 16 - deployment VM-a



Slika 17 - ustvarjen VM

SSH dostop za vse člane je zagotovljen preko javnega IP naslova:

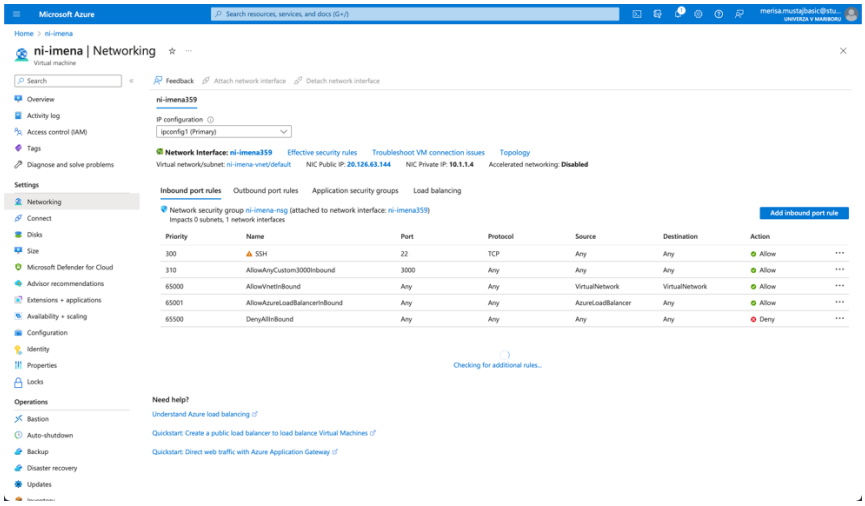


Slika 18 - ssh dostop vseh članov

2.2. Azure portal osnove

Kje in kako omogočite "port forwarding" ?

V Azure portalu lahko omogočimo "port forwarding" za navidezno napravo (virtual machine) tako, da v meniju navidezne naprave izberemo možnost "Networking". Nato izberemo "Add inbound port rule" in nastavimo pravila za "port forwarding" glede na naše potrebe. Na primer, mi smo rabili omogočiti promet na portu 3000 kako bi lahko dostopali do njega preko javnega IP naslova, zato smo dodali pravilo ki nam to omogoči.



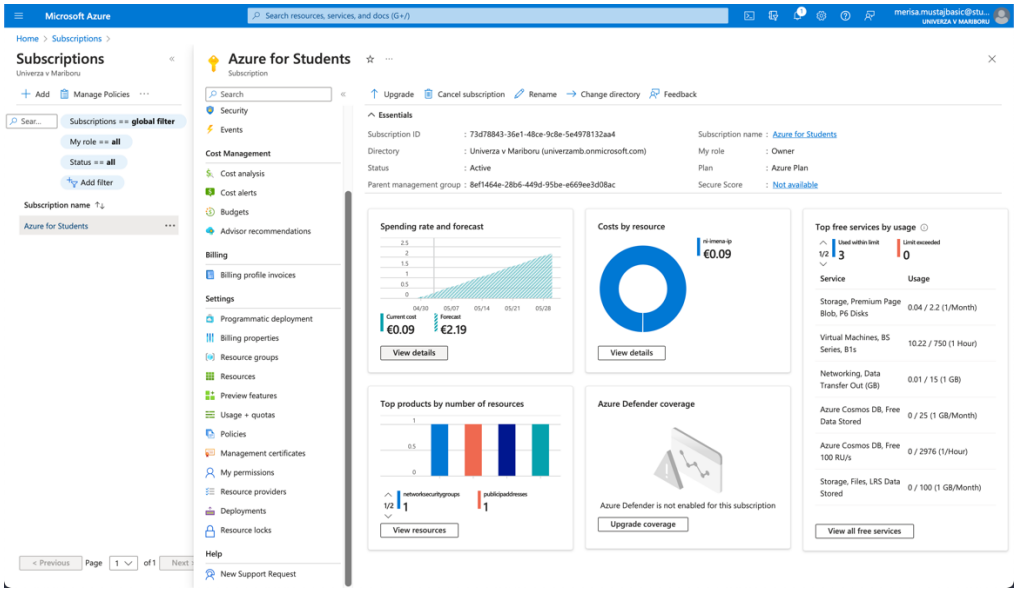
Slika 19 - port forwarding

Kakšen tip diska je bil dodan vaši navidezni napravi in kakšna je njegova kapaciteta ?

V meniju navidezne naprave v Azure portalu lahko preverimo tip diska, ki je bil dodan naši navidezni napravi, tako da kliknemo na "Disks". Tam vidimo seznam diskov, ki so na voljo za navidezno napravo, vključno z njihovo kapaciteto in tipom diska (npr. HDD ali SSD). V našem primeru vrsta diska je Premium SSD LRS in kapaciteta 64 GB.

Kje preverimo stanje trenutne porabe virov v naši naročnini ("Azure for students") ?

V Azure portalu lahko preverimo trenutno porabo virov v naši naročnini tako, da izberemo možnost "Cost Management + Billing" v levem meniju.



Slika 20 - poraba virov

2.3. Vzpoztavitev Docker aplikacije

Uporaba git-a za prenos kode

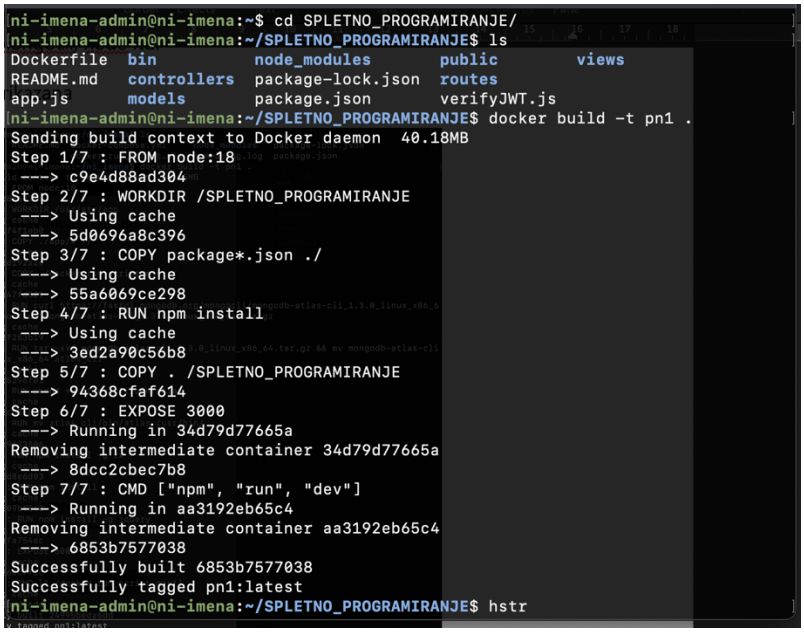
Slika 21 - git za prenos kode projekta

Omogočimo najprej ssh povezavo torej naredimo keygen ki ga dodamo na github ki gostuje organizaciju za projekt. Z tem lahko kloniramo repozitorije prek SSH.

Namestite vse potrebne pakete za vašo aplikacijo

Nameščen je docker, mongocli, node itd.

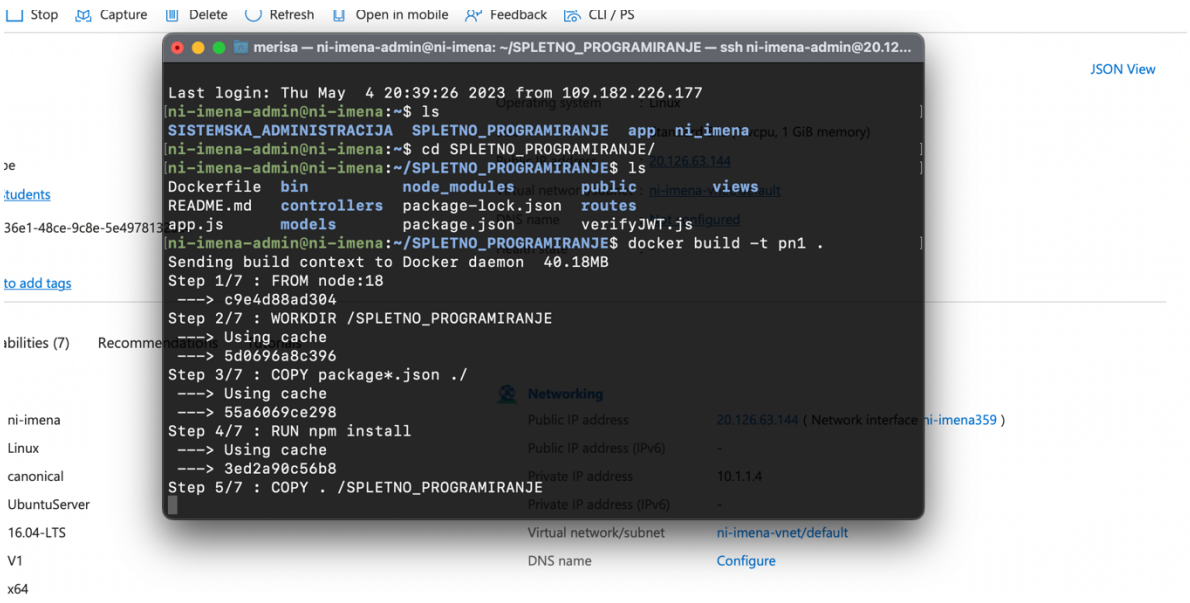
Struktura projekta je prikazana

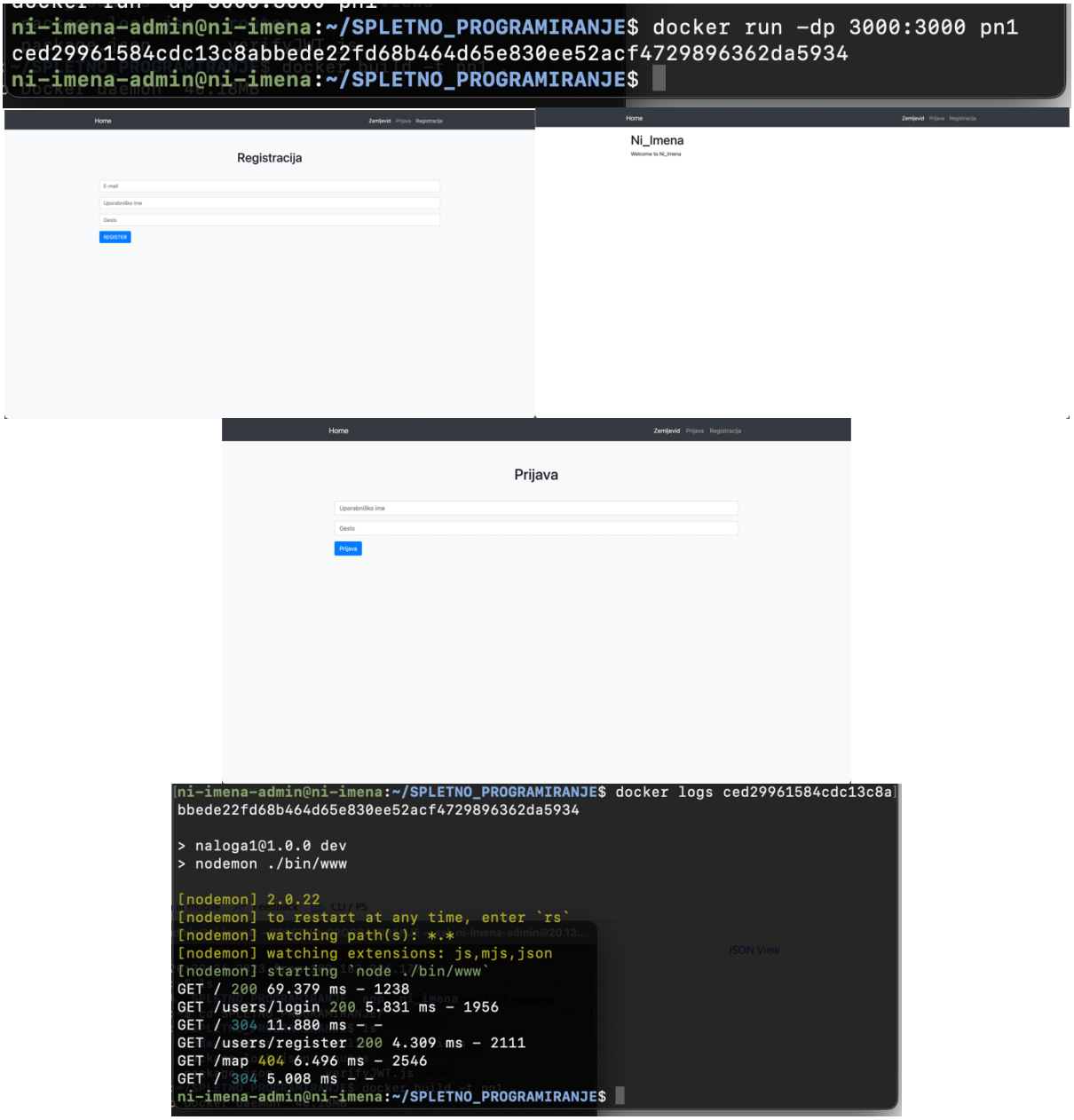


Slika 22 - strucktura folderja, Dockerfile ipd

Omogočite dostopnost vaše aplikacije iz javnega omrežja

Z ukazom „docker run“ zaženemo naš container in je aplikacija potem dostopna preko javnega naslova VM-a, s dodanom številko vrat (v našem primeru je to 3000).



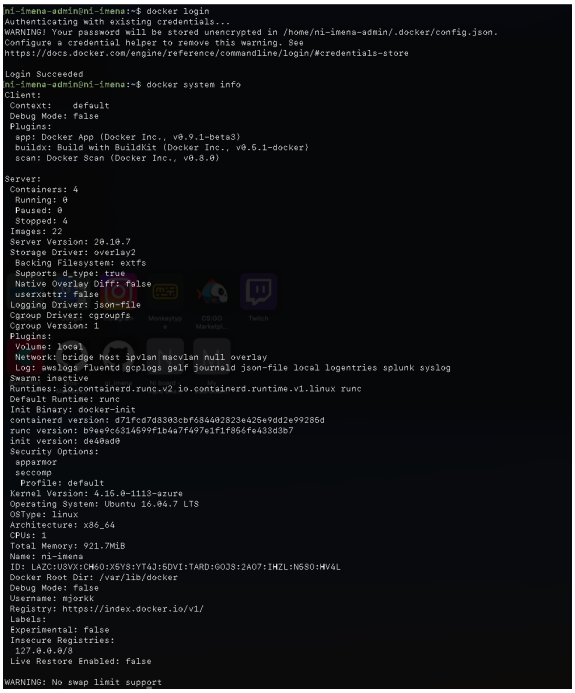


Slika 23 - dostop preko javnega IP naslova in številke vrat

3. Docker Hub container registry

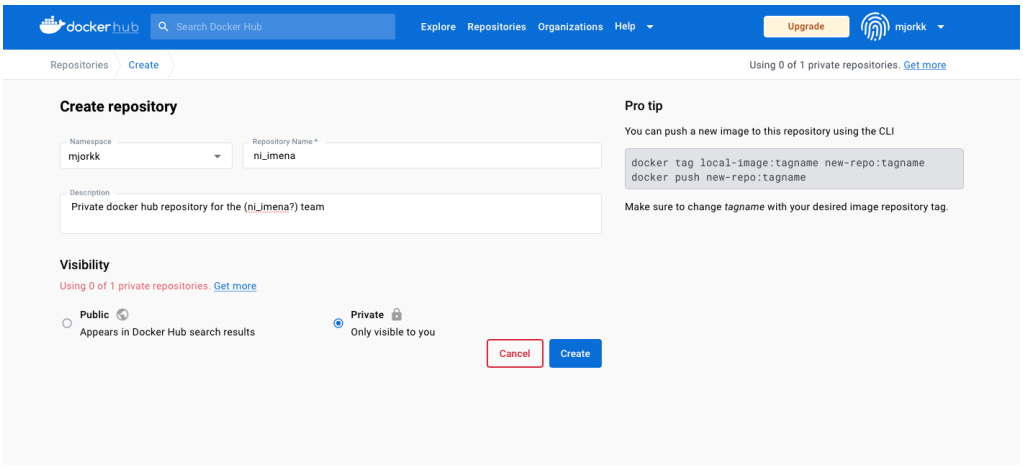
3.1. Namestitev Docker Hub Registry

Najprej smo namestili uporabniski racun vodje skupine docker-ja na VM preko ukaza „docker login“.



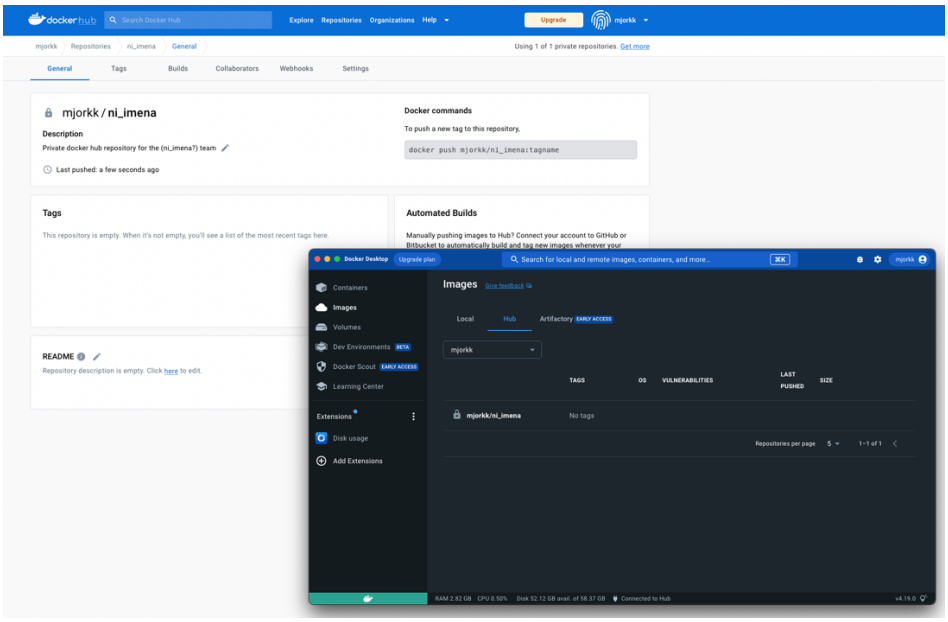
Slika 24 - namestitev Docker Hub Registry

Nato ustvarimo repozitorij in nastavimo da je private.



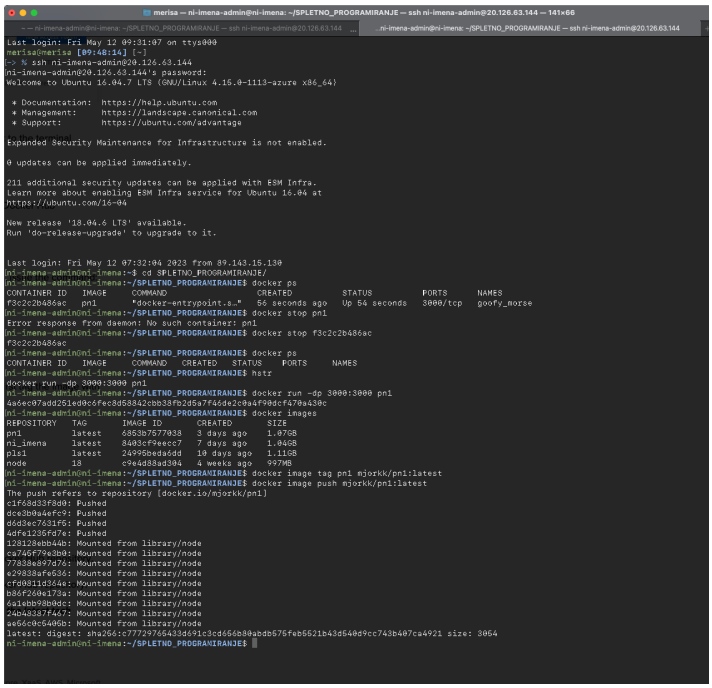
Slika 25 - nastavitve Docker Hub repozitorija

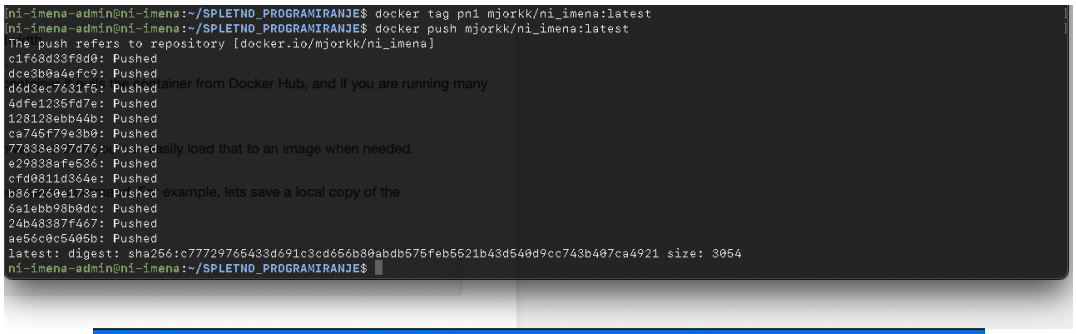
Repozitorij je zdaj viden tudi preko namizne aplikacije docker hub.

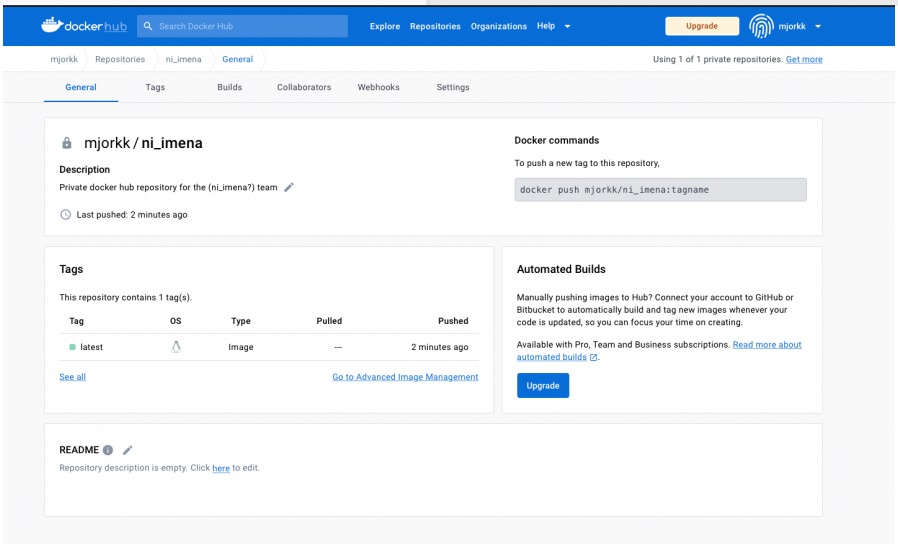


Slika 26 - namiyna aplikacija Docker Hub

Docker sliko nasega projekta push-amo na docker hub s pomocjo ukazov : docker image tag in docker image push.



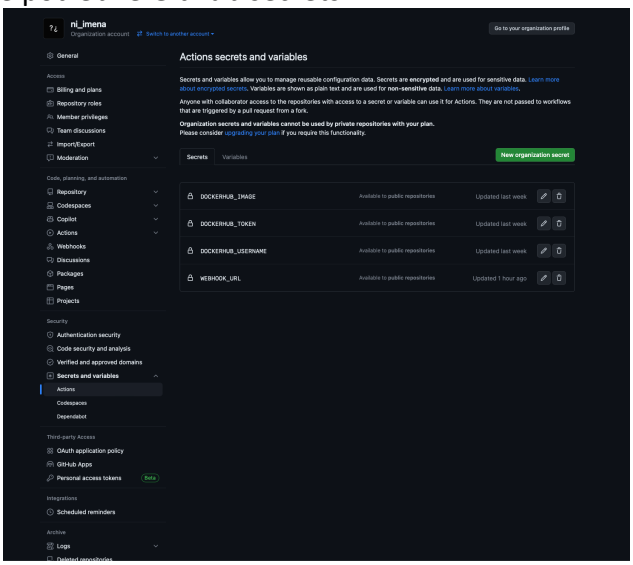




Slika 27 - prvi push na Docker Hub repozitorij

4. GitHub Actions workflows

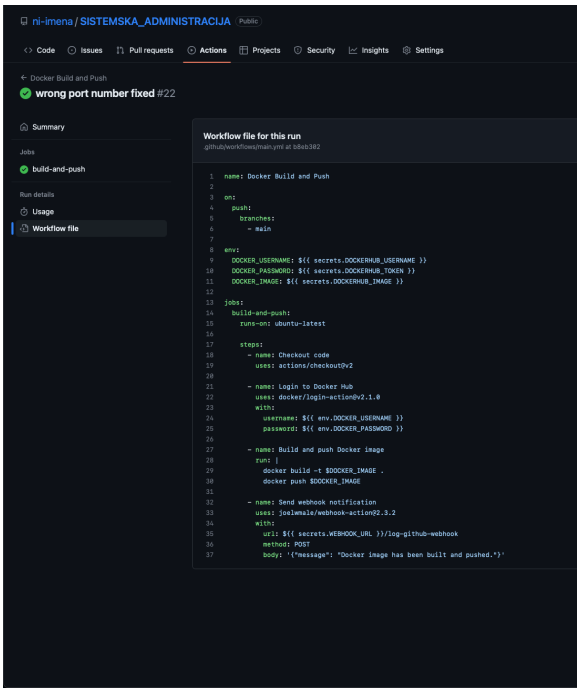
Najprej nastavimo vdje potrebne GitHub secrets:



Slika 28 - Github Secrets

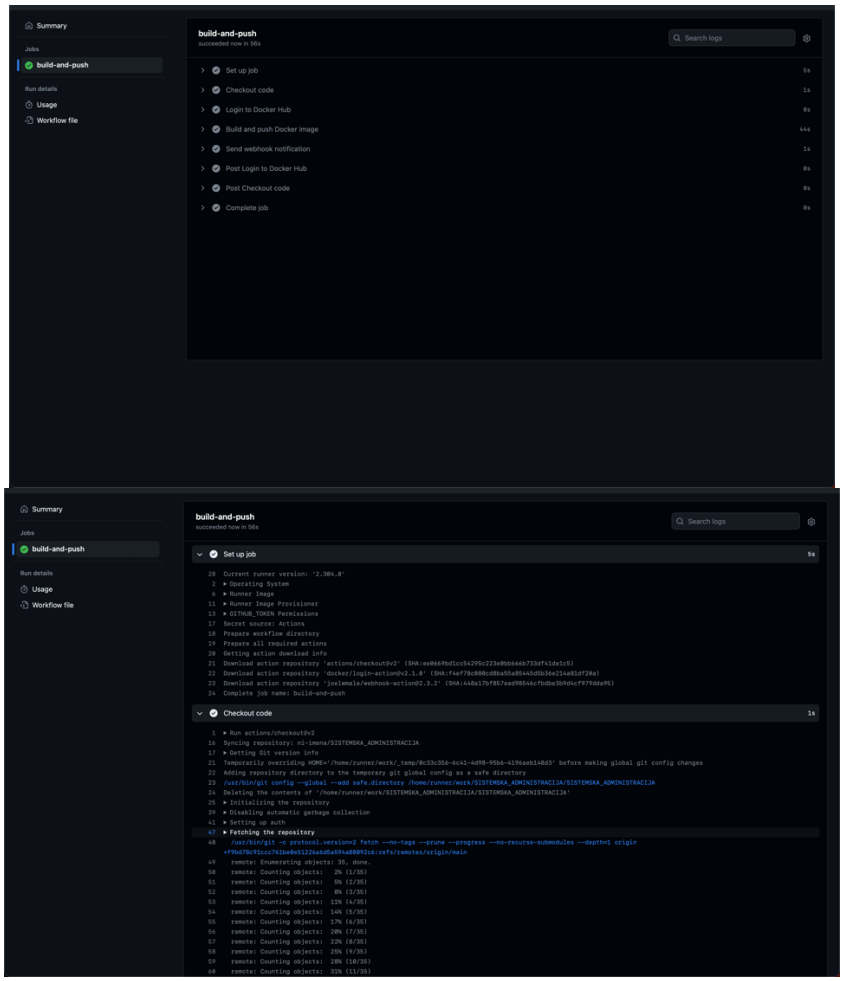
V zavihu za Workflows repozitorija napisemo skripto za podane naloge:

- ustvarjanje **Docker** slike in njeno nalaganje na **Docker Hub,**
- pošiljanje sporočila vašemu strežniku s pomočjo **webhooks**



Slika 29 - Workflow

Zdaj z vsakim pushom na repozitorij, se workflow zazene in naredi prejsnje omenjene naloge. To je tudi razvidno na docker hub, s katerim vidimo kdaj se je zadnja sprememba zgodila na repozitoriju.



Slika 30 - uspešno izveden workflow

4.1. GitHub Actions Workflow ideje za projekt ni_imena?

Za projekt "Digitalni dvojček tekača" bi lahko dodali naslednje dodatne GitHub Actions workflows:

1. **Continuous Integration (CI) Workflow**: Ta workflow bi bil odgovoren za avtomatizirano preverjanje kode in zagotavljanje, da so vse spremembe v kodi združljive in delujejo pravilno. Workflow bi se sprožil ob vsakem potisku v glavno vejo (npr. **main**) ali pri pošiljanju sprememb v odvisnih vejah. Vključeval bi naslednje korake:
    - Preverjanje kode s statično analizo, stilskimi smernicami in preverjanjem kakovosti kode.
    - Izvajanje enotnih testov, da se zagotovi delovanje osnovnih funkcionalnosti.
    - Gradnja in pakiranje aplikacije v ustrezno obliko (npr. Docker sliko).
2. **Deployment Workflow**: Ta workflow bi bil odgovoren za avtomatizirano distribucijo aplikacije na izbrani ciljni okolje po uspešnem preverjanju kode. Workflow bi se sprožil po združitvi sprememb v glavno vejo (npr. **main**). Vključeval bi naslednje korake:

- Preverjanje kode (kot v CI workflow).
- Gradnja in pakiranje aplikacije v ustrezno obliko (npr. Docker sliko).
- Distribucija aplikacije na ciljni strežnik ali oblak (npr. AWS, Azure, GCP).
- Zagon aplikacije na ciljnem okolju in preverjanje, ali je pravilno delujoča.

3. **Scheduled Workflow**: Ta workflow bi bil nastavljen za periodično izvajanje določenih nalog, na primer za izvajanje rednih nalog vzdrževanja, arhiviranja podatkov ali preverjanja delovanja sistema. Workflow bi se sprožil vnaprej določen časovni interval ali na določen dan v tednu/mesecu. Naloge, ki bi se izvajale v tem workflowu, bi bile specifične za potrebe projekta.
4. **Pull Request Workflow**: Ta workflow bi se sprožil ob odprtju ali posodobitvi potegnjenega zahtevka (pull requesta) in bi pomagal pri preverjanju kode in zagotavljanju kakovosti pred združitvijo sprememb v glavno vejo. Vključeval bi podobne korake kot CI workflow, vendar bi se izvajal samo na spremembah, vključenih v potegnjeni zahtevek.
5. **Issue/Feature Workflow**: Ta workflow bi bil namenjen za upravljanje s prijavami napak (issues) ali predlogi funkcionalnosti (features). Sprožil bi se ob odprtju ali posodobitvi novega zapisa in bi lahko vključeval korake, kot so preverjanje in validacija zapisa, dodelitev nalog

5. Webhook

V kodu servera dodamo del za sprejemanje webhooka.

```
async function manageContainer() {
  try {
    // Stop the currently running container (if it exists)
    await exec('docker stop pn3').catch(() => {});

    // Pull the latest image from Docker Hub
    await exec('docker pull mjorkk/ni_imena:latest');

    // Start the container
    await exec('docker run -d --name pn3 mjorkk/ni_imena:latest');

    // Send webhook notification
    const webhookURL = 'https://hkdk.events/yzTux3lS7QVe'; // Replace with your actual webhook URL
    await axios.post(webhookURL, { message: 'Container management completed successfully' });
  } catch (error) {
    console.error('Error managing container:', error);
  }
}
app.use(function (req, res, next) {
    next(createError(404));
});

app.post('/log-github-webhook', (req, res) => {
  // Handle the webhook request
  // ...

  // Call the manageContainer function
  manageContainer();
console.log("sjhdfgjskdf");

  res.status(200).send('Webhook received successfully');
});
```

- Napišite preprosto skripto v poljubnem programskem jeziku, ki bo:
  - zaustavila trenutno zagnan docker container,
  - iz **Docker Container registry** prenesla najnovejšo različico vašega cont
  - ga ponovno zagnala.
- Skripto zaženite ob vsakem dobljenem **Webhook** sporočilu.
- Opišite varnostne luknje in vaše pomisleke pri uporabi **Webhook,** opišite (z

Na zagovoru vaj boste prikazali pravilno delovanje celotnega GitHub Actio

Na sistem **oddajte poročilo v .pdf obliki.** Poročilo naj vsebuje opise posamezn
pomočjo ustreznih orodij (ti, Jiro, OpenProj). Na zagovoru bomo pogledali opr

KRITERIJI OCENJEVANJA IMPLEMENTACIJE:

1. točka - Docker Hub container registry
2. točka - delujoč osnovni GitHub Actions workflow
3. točka - opis namestitve dodatnega GitHub Actions workflows
4. točka - Webhook
5. točka - varnost v Webhook

KRITERIJI OCENJEVANJA VODENJA PROJEKTA:

1 točka - oddano in zagovorjeno poročilo
2 točki - avtomatično vodenje projekta na izbranem sistemu (Jiro, OpenProj, ip

Naloge brez oddanih poročil, ali s pomanjkljivimi poročili, bodo ocenjene z

Slika 31 - Webhook

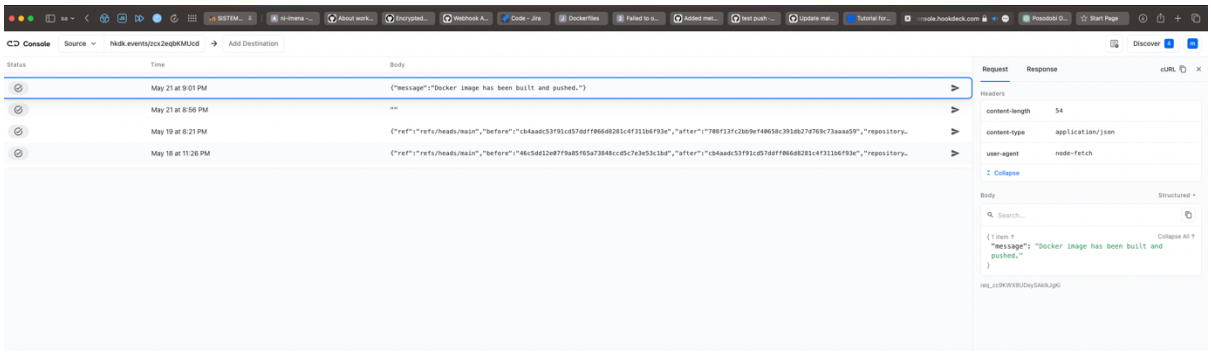
Instaliramo in zazenemo na portu 1337 hookdeck:

```
ni-imena-admin@ni-imena:~/SISTEMSKA_ADMINISTRACIJA$ sudo hookdeck listen 1337
? Select a source Create new source
? What should be your new source label? pn3
? What path should the webhooks be forwarded to (ie: /webhooks)? /log-github-webhook
? What's your connection label (ie: My API)? response server

Dashboard
👉 Inspect and replay webhooks: http://console.hookdeck.com?source_id=src_yzTux31S7QVe

pn3 Source
`\ Webhook URL: https://hkdk.events/yzTux31S7QVe

Connections
response-server forwarding to /log-github-webhook

> Ready! (^C to quit)
2023-05-21 20:36:51 [ERROR] Failed to POST: Post "http://localhost:1337/log-github-webhook/log-github-webhook": dial tcp 127.0.0.1:1337: connect: connection refused
2023-05-21 20:37:09 [ERROR] Failed to POST: Post "http://localhost:1337/log-github-webhook/log-github-webhook": dial tcp 127.0.0.1:1337: connect: connection refused
```

Workflow file for this run

.github/workflows/main.yml at d3b8557

```yaml
name: Docker Build and Push

on:
  push:
    branches:
      - main

env:
  DOCKER_USERNAME: ${{ secrets.DOCKERHUB_USERNAME }}
  DOCKER_PASSWORD: ${{ secrets.DOCKERHUB_TOKEN }}
  DOCKER_IMAGE: ${{ secrets.DOCKERHUB_IMAGE }}

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Login to Docker Hub
        uses: docker/login-action@v2.1.0
        with:
          username: ${{ env.DOCKER_USERNAME }}
          password: ${{ env.DOCKER_PASSWORD }}

      - name: Build and push Docker image
        run: |
          docker build -t $DOCKER_IMAGE .
          docker push $DOCKER_IMAGE

      - name: Send webhook notification
        uses: joelwmale/webhook-action@2.3.2
        with:
          url: http://20.126.63.144:1337/log-github-webhook
          method: POST
          body: '{"message": "Docker image has been built and pushed."}'
```



Slika 32 - webhook

5.1. Varnostne luknje pri uporabi Webhook

Pri uporabi Webhook-ov obstaja nekaj varnostnih lukenj in pomislekov, ki jih je treba upoštevati. Nekatere od teh lukenj so:

1. **Nepristnost (Spoofing)**: Napadalec lahko poskuša poslati lažne Webhook zahteve, pretvarjajoč se, da prihajajo od zaupanja vrednega vira. To bi lahko privedlo do izvedbe neželenih dejanj ali vnosa škodljive kode v vaš sistem.
2. **Ponavljanje (Replay Attack)**: Napadalec lahko prestreže veljavne Webhook zahteve in jih nato ponovi kasneje, da povzroči podvojene ali neželene operacije.
3. **Vdiranje (Injection)**: Napadalec lahko poskuša vstaviti zlonamerno vsebino v Webhook zahtevo, kar bi lahko povzročilo izvajanje škodljive kode ali zlorabo sistema.
4. **Nepopolno preverjanje podatkov (Incomplete Data Validation)**: Če ne izvajate zadostne preverbe podatkov v prejetih Webhook zahtevah, obstaja tveganje za vnos neveljavnih ali zlonamernih podatkov v vaš sistem.
5. **Dostopna pravica (Access Control)**: Neustrezno upravljanje dostopnih pravic do Webhook-ov lahko pomeni, da nepooblaščene osebe pridobijo dostop do občutljivih podatkov ali funkcij.

Za odpravo teh varnostnih lukenj in pomislekov pri uporabi Webhook-ov lahko upoštevatmo naslednje prakse:

1. **Overjanje (Authentication)**: Zagotovimo, da preverite pristnost prejetih Webhook zahtev, na primer z uporabo avtentikacije z žetonom (token-based authentication). Vsak vir Webhook-a naj ima svoj edinstven žeton, ki ga uporabljamo za preverjanje pristnosti.
2. **Šifriranje (Encryption)**: Če prenašamo občutljive podatke prek Webhook-ov, jih šifriramo, da se prepreči prestrezanje in nepooblaščeno branje.
3. **Preverjanje integritete (Integrity Checking)**: Za preprečevanje ponavljanja napadov uporabite mehanizme preverjanja integritete, na primer s pomočjo časovnih oznak ali digitalnih podpisov. S tem zagotovimo, da so prejete Webhook zahteve unikatne in niso bile spremenjene.
4. **Preverjanje vhodnih podatkov (Input Data Validation)**: Temeljito preverimo vhodne podatke, ki jih prejmemo prek Webhook-ov. Preverimo, ali so podatki v skladu z določenimi pravili in omejitvami.
5. **Upravljanje dostopa (Access Management):** Omogočimo ustrezno upravljanje dostopnih pravic do Webhook-ov.
6. **Dnevniško beleženje (Logging):** Zagotovimo ustrezno dnevniško beleženje dogodkov, povezanih z Webhook-i. Zabeležimo prejete zahteve, njihove rezultate in morebitne napake. S tem boste lahko sledili dogodkom, analizirali morebitne varnostne incidente in izvedli potrebne ukrepe.