Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Napredno korištenje operacijskog sustava Linux

Laboratorijska vježba 3

Nositelj:
doc. dr. sc. Marko Đurasević
/ZEMRIS

Autori:
Jakov Ivković

Zagreb, 16. svibnja 2022.

Labos je osmišljen za rješavanje na cloud serveru. Kako ne bi morali plaćati korištenje servera na raspolaganju vam je GitHub Student Developer Pack s kojim dobivate besplatne kredite na nekoliko najpoznatijih cloud providera.[1] Iako je moguće labos riješiti na vlastitom računalu ili lokalnom virtualnom stroju, takav način rješavanja će biti otežan te se preporučuje korištenje cloud servera. Hardverski uvjeti servera, za potrebe labosa, su 4GB radne memorije.[2]

1 Uvod

Prijatelj Tristan za kojeg ste deployali jednostavnu `Blog` aplikaciju je presretan jer je aplikacija postala pravi hit. Ponovno vam se javlja želeći osnovati startup kako bi mogao još poraditi na aplikaciji te se na kraju i obogatiti.

Tristan je čuo dobre stvari o agilnom razvoju te da ga danas mnoge firme koriste. Stoga vam Tristan nudi 50% udjela u firmi ako složite potrebnu infrastrukturu za agilni razvoj koji će firma koristiti. Vi ste, željni obećane slave i bogatstva, prihvatili ponudu te se odmah bacili na posao.

2 Zadatak

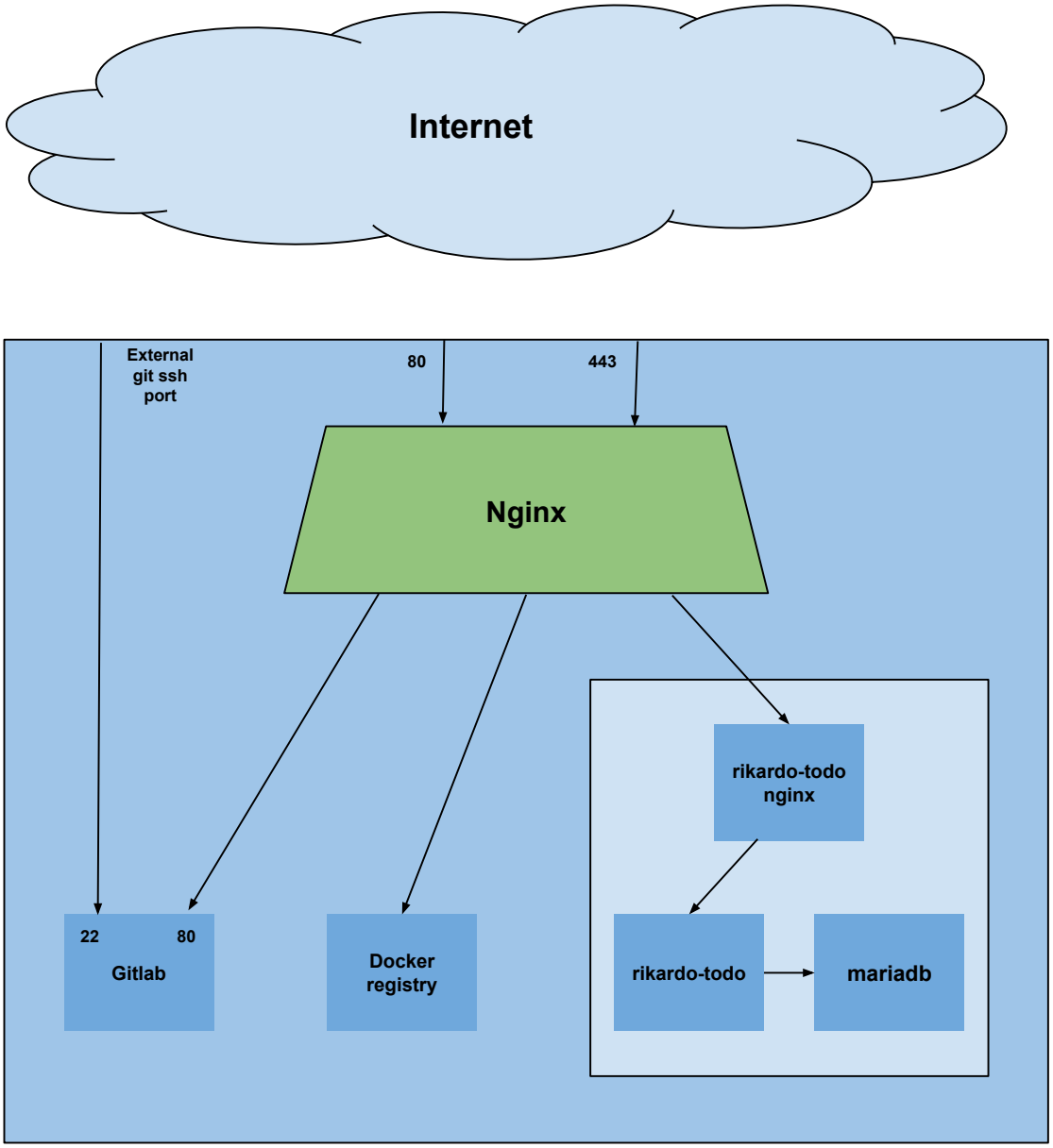
Potrebno je:

- Deployati Gitlab koristeći službeni docker image
- Deployati Docker registry te ga zaštititi s lozinkom
- Deployati aplikaciju (smijete koristiti sve što ste napravili u 1. laboratorijskoj vježbi)
- Za sve 3 gornje stavke napraviti domenu, HTTPS certifikate te pripadne reverse proxy nginx server blokove.
- Pushati git repozitorij nkosl-www aplikacije na instancu Gitlaba koju ste deployali, napraviti automatski build docker imagea i push na registry za svaki commit na repozitorij te automatski deploy za svaki commit u *master* granu.

Za jasniju sliku kako sve na kraju treba izgledati možete pogledati shemu 1. Veliki pravokutnik na kojem se sve nalazi predstavlja server. Mali, tamno plavi, pravokutnici predstavljaju docker kontejnere.

[1] Uključeni provideri su: AWS, Microsoft Azure te DigitalOcean. Najjednostavniji za korištenje pokazao se DigitalOcean.

[2] Dovoljno je i 2GB radne memorije uz dodatnih 2GB swap memorije.



Slika 1: Shema arhitekture

3 DNS

Za ovu vježbu trebat će nam 3 hostnamea, po jedan za Gitlab, docker registry i aplikaciju. Jedan od najjednostavnijih besplatnih DNS servisa je Duck DNS[3], ali slobodno koristite koji god DNS servis želite.[4]

4 Gitlab

Potrebno je deployati Gitlab u kontejneru. Na poveznici možete pronaći službenu dokumentaciju o deployanju Gitlaba u docker kontejneru.

Gitlab docker image je monolitni image u kojem se nalaze svi potrebni servisi za pokretanje Gitlaba. Iako je riječ o samo jednom kontejneru i dalje se preporučuje korištenje docker-composea radi lakše konfiguracije. Kao početna točka može vam poslužiti docker-compose iz službene dokumentacije uz sljedeće napomene:

- Port 443 iz kontejnera, kroz koji ide kriptirani https promet, nam nije potreban budući da ćemo koristiti nginx kao reverse proxy te ćemo na njemu postaviti HTTPS enkripciju te pripadne certifikate. Stoga je potrebno u `GITLAB_OMNIBUS_CONFIG` varijablu okruženja dodati direktive `nginx['listen_port'] = 80` i `nginx['listen_https'] = false`. Mapiranje porta 443 iz kontejnera možemo izostaviti.
- Port 80 iz kontejnera ne možemo mapirati na port 80 na serveru budući da taj port zauzima nginx. Stoga je potrebno port 80 mapirati na neki drugi port na serveru.
- Port 22 iz kontejnera koristi se za *git over SSH*[5]. Međutim, najvjerojatnije vam je port 22 na serveru već zauzeo SSH server putem kojeg se spajate na njega. Stoga je potrebno port 22 iz kontejnera postaviti na neki drugi port na serveru, a također, potrebno je postaviti i `gitlab_rails['gitlab_shell_ssh_port']` direktivu u `GITLAB_OMNIBUS_CONFIG` varijabli okruženja kako bi Gitlab znao korisniku prikazati točan vanjski SSH port u uputama za kloniranje repozitorija putem SSH protokola.

[3] Duck DNS je zapravo dinamički DNS, odnosno svrha mu je pružiti DNS usluge korisnicima koji nemaju statičku IP adresu, ali ga slobodno možemo koristiti i za servere koji imaju statičku IP adresu.

[4] U sklopu GitHub Student Developer Packa imate besplatne promocije za Name.com, Namecheap te .TECH servise

[5] `https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols`

- Kao `external_url` postavljate jedan od 3 hostnamea za koja postoji DNS zapis prema vašem serveru, a koje ste postavili na nekom od pružatelja DNS usluga iz poglavlja 3.

Dokumentaciju o svim konfiguracijskim direktivama možete pronaći na sljedećoj poveznici: `https://docs.gitlab.com/omnibus/settings/`

Sljedeće je potrebno generirati besplatan Lets's Encrypt HTTPS certifikat te namjestiti reverse proxy putem nginxa. Let's encrypt certifikat najlakše je izgenerirati pomoću Certbota. Više o Certbotu i jednostavne upute za korištenje mogu se pronaći na sljedećoj poveznici: `https://certbot.eff.org/`

Certifikat kojeg ste stvorili putem Certbota trebao bi se nalaziti u datoteci `/etc/letsencrypt/live/<hostname>/fullchain.pem`, a privatni ključ u datoteci `/etc/letsencrypt/live/<hostname>/privkey.pem`. Sljedeće je potrebno napraviti 2 nginx server bloka; jedan koji će slušati nekriptirane HTTP zahtjeve (na portu 80) i odgovarati s *permanent redirect*[6] s HTTPS URL-om te drugi koji će slušati HTTPS zahtjeve (na portu 443) i prosljeđivati ih na port 80 Gitlab kontejnera[7].

Primjer nginx konfiguracije:

```
server {
    listen          80;
    server_name     <hostname>;
    return 301      https://$host$request_uri;
}


server {
    listen          443 ssl;
    server_name     <hostname>;

    access_log      /var/log/nginx/gitlab.access.log;
    error_log       /var/log/nginx/gitlab.error.log;

    ssl_certificate         /etc/letsencrypt/live/<hostname>/fullchain.pem;
    ssl_certificate_key     /etc/letsencrypt/live/<hostname>/privkey.pem;

    location / {
        include         proxy_params;
        proxy_pass      http://localhost:<port>;
    }
}
```

Za kraj bi bilo dobro onemogućiti registraciju na Gitlab preko admin postavki.

[6] HTTP 301

[7] Odnosno port na hostu na koji je mapiran port 80 iz kontejnera.

5 Docker registry

Potreban nam je vlastiti docker registry na kojeg ćemo *pushati* automatski buildane docker image blog aplikacije te s kojeg ćemo povlačiti image aplikacije prilikom deploya.

Za pokretanje registryja dovoljna je sljedeća naredba iz službene dokumentacije: `docker run -d -p 5000:5000 --restart=always --name registry registry:2`. Naravno, slobodni ste mapirati port `5000` na bilo koji port na serveru. Sljedeće je potrebno konfigurirati reverse proxy s nginxom na isti način kao i za Gitlab (naravno, koristeći drugi hostname) uz iznimku da je registry potrebno dodatno zaštititi HTTP basic autentifikacijom. Više o postavljanju basic autentifikacije na nginxu možete pročitati u dokumentaciji.

Nakon što postavite reverse proxy s autentifikacijom, možete isprobati pushanje i pullanje docker imagea lokalno, s vašeg računala, na vaš novi registry na serveru. Budući da smo postavili basic autentifikaciju, prvi korak je pokretanje `docker-login` naredbe za vaš registry koristeći username i password koji ste postavili u `htpasswd` datoteci. Nakon što ste se uspješno ulogirali pokušajte na vaš registry pushati alpine docker image.

6 Deploy aplikacije

Na server je potrebno deployati aplikaciju pomoću docker composea te ju "staviti iza" nginx reverse proxyja kao što ste to učinili za Gitlab. Dakle, potrebno je napraviti isto što ste napravili u zadnjem zadatku drugog labosa uz dodatak postavljanja nginx reverse proxyja te SSL certifikata na njemu.

Naravno, nakon što sve postavite, uvjerite se da možete pristupiti aplikaciji te da ona funkcionira.

7 CI/CD

Sad kad imamo sve servise podignute i funkcionalne ostalo nam je složiti CI/CD cjevovod. Koristit ćemo Gitlabov CI/CD alat koji je uključen u standardnu instalaciju. Kod Gitlabovog CI/CD alata, cjevovod se definira putem `.gitlab-ci.yml` datoteke u repozitoriju. Dokumentaciju proučite na sljedećoj poveznici. Poslove koje smo definirali u `.gitlab-ci.yml` datoteci izvršavaju se na Gitlab runneru čiju dokumentaciju možete pronaći na poveznici.

Prvi korak koji je potrebno napraviti je stvoriti prazan "nkosl-www" repozitorij na Gitlabu te tamo pushati postojeći git repozitorij aplikacije. Također, u novom commitu pushajte i svoj Dockerfile za aplikaciju pošto će nam trebati kod automatskog buildanja imagea aplikacije u pipelineu.[8]

Sljedeće je potrebno instalirati Gitlab runnera na serveru te ga registrirati na deployanu Gitlab instancu. Runner treba koristiti shell executora.

Ostalo nam je još "samo" definirati sam cjevovod. Ideja je sljedeća: želimo kod svakog commita koji se pusha naš repozitorij na Gitlabu da se automatski napravi build novog imagea te da se taj image pusha na naš docker registry. Nadalje, ako je commit na master grani, želimo nakon pusha imagea na registry da se automatski dogodi backup baze[9] te, nakon toga, deploy nove verzije aplikacije na serveru koristeći novostvoreni image. Kod pushanja imagea na registry, neka se image zove `nkosl-www`, a neka nam kao image tag posluži hash commita.[10] Dodatno, kod pokretanja cjevovoda s master grane, nakon pusha imagea na registry, potrebno je taj isti image pushati i s `latest` tagom kako bi uvijek znali koji je najažurniji stabilni image[11].

Opisanu funkcionalnost slobodni ste potpuno samostalno implementirati u `.gitlab-ci.yml` datoteci, ali također možete koristiti i sljedeći kostur datoteke:

```
stages:
 - build
 - pre-deploy
 - deploy

build:
  stage: build
  script:
   - naredbe koje ce napraviti novi image
   - te koje ce pushati image u registry

push-latest:
  stage: pre-deploy
  script:
   - naredbe koje ce povuci novostvoreni image s registryja
   - tagati taj image s latest tagom
   - te ga pushati na registry
  only:
   - master

backup-db:
  stage: pre-deploy
  script:
```

[8] A ionako je Dockerfileu mjesto u repozitoriju aplikacije :)

[9] U obliku mysql dumpa

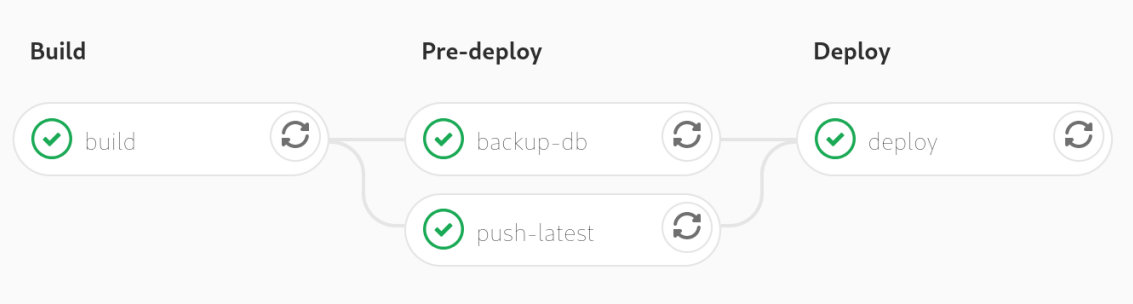
[10] Kako nam puno ime imagea ne bi bilo predugo, možete koristiti i samo prvih 8 znakova commit hasha. Hint: poslužit će vam predefinirane varijable okruženja `CI_COMMIT_SHA` ili `CI_COMMIT_SHORT_SHA` koje runner dobiva kod izvršavanja poslova.

[11] te kako biste si, potencijalno, olakšali automatski deploy

```
    - naredbe koje ce napraviti dump baze
  only:
    - master

deploy:
  stage: deploy
  script:
    - naredbe koje ce povuci novi latest image aplikacije
    - naredbe koje ce deployati novopullani latest image aplikacije
  only:
    - master
```

Ako koristite priloženi kostur `.gitlab-ci.yml` datoteke, izgled cjevovoda za master granu na Gitlabu[12] trebao bi vam biti nalik slici 2.



Slika 2: Primjer pipelinea u Gitlabu

Ako vam poslovi neuspješno završavaju s `No such file or directory` greškom, vjerojatno ste naišli na sljedeći bug. Brisanje skrivenih datoteka u matičnom direktoriju runnera (`/home/gitlab-runner`) trebalo bi riješiti problem.

[12] U repozitoriju → CI/CD → Pipelines