Alati za razvoj softvera

Rate Limiting, Token Bucket, Continuous integration, Github Actions

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Ograničenje stope pristupa — uvod

- U računarskim mrežama, ograničenje brzine/stope se koristi za kontrolu brzine/stope zahteva poslatih ili primljenih od strane kontrolera mrežnog interfejsa
- Ograničavanje stope/brzine pristupa (Rate Limiting) je procedura koja nam omogućava kontrolu brzine kojom korisnici mogu da šalju zahteve sistemu
- **Rate Limiting** se uglavnom koristi za zaštitu servera od neželjenih rafala, zlonamernih napada
- Zaštita sistema od prekomerne upotrebe ograničavanjem koliko često korisnici mogu da im pristupe, ima nekoliko prednosti

- Pomaže protiv napada *denial-of-service*, pokušaja prijave *brute-force* i drugih vrsta nasilnog ponašanja korisnika
- Može i da se koristi kod različitih servisa da se vidi da li imamo dovoljno finansija da pristupimo nekakvom resursu
- Web servisi mogu da ga korite kada pružaju usluge korisnicima da bi odbili zahteve, ako su prekoračili limit

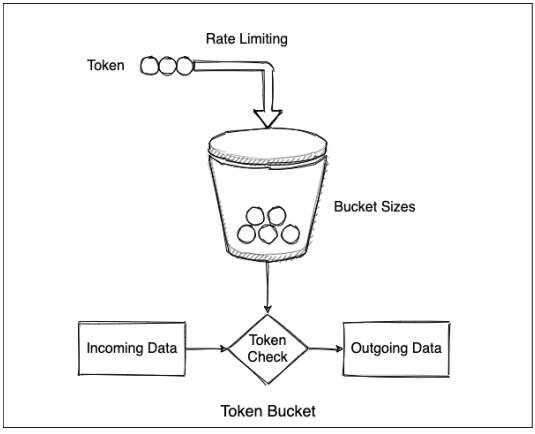
- Postoji razni tipovi Rate Limiting-a npr:
  - **Rate limiter korisnika** omogucćava nekim grupama korisnika ograničen pristup sistemu — broj/trajanje zahteva korisnika obično je vezan za njihove ključeve ili IP adrese
  - **Rate limiter istovremene/serverske brzine** prati koliko je paralelnih sesija ili veza dozvoljeno za nekim grupama korisnika — ublažava DDoS napade
  - **Rate limiter lokacije** ograničava brzine/stope pristupa za neke regione, kao i za definisani vremenski period — moguće je definisati različite stepene pristupa za razne lokacije

Primeri algoritama

- Neki od primera algoritama za rešavanje ovog problema:
  - **Token Bucket** — radimo danas
  - Leaky Bucket
  - Fixed Window Counter
  - Sliding Logs
  - Sliding Window Counter

Tocken Bucket — uvod

- Ovo je najjednostavniji algoritam za ograničavanje brzine pristupa
- Jednostavno pratimo broj zahteva napravljenih u zadatom vremenskom intervalu
- Zbog svoje jednostavnosti, dosta se koristi
- Google cloud koristi ovaj algoritam (ili je koristio), za Task Queue opciju koja se nudi koirsnicima kao usluga
- Jednostavno se implementira, i lako može da se poveže sa velikim brojem različitih slučajeva korišćenja



(What is Token Bucket and Leaky Bucket algorithms)

Tocken Bucket — algoritam

- Za svaki zahtev korisnika treba:
  - Proveriti da li je vreme proteklo od poslednjeg resetovanja brojača vremena
  - Ako vreme nije isteklo, treba proveriti da li korisnik ima dovoljno preostalih zahteva da obradi dolazni zahtev
  - Ako korisniku nije preostalo slobodnih zahteva, trenutni zahtev se odbacuje uz nekakvu poruku
  - U suprotnom, smanjujemo brojač za **1**, i vršimo obradu dolaznog zahteva
  - Ako je vreme proteklo, tj. razlika resetovanog vremena i trenutnog vremena je veća od definisanog intervala, resetujemo broj dozvoljenih zahteva na unapred definisano ograničenje, i definišemo novo vreme resetovanja

Tocken Bucket — primer

Pimer: 3/min:

- REQ **11:01:20** – > BUCKET [11:01:05, 3] => OK
- REQ **11:01:25** – > BUCKET [11:01:05, 2] => OK
- REQ **11:01:30** – > BUCKET [11:01:05, 1] => OK
- REQ **11:01:35** – > BUCKET [11:01:05, 0] => FAIL
- REQ **11:03:00** – > BUCKET [11:03:00, 2] => OK // uradimo update vremena, broja tokena, pustimo zahtev i smanjimo broj tokena za **1**
- ...

Continuous Integration – CI

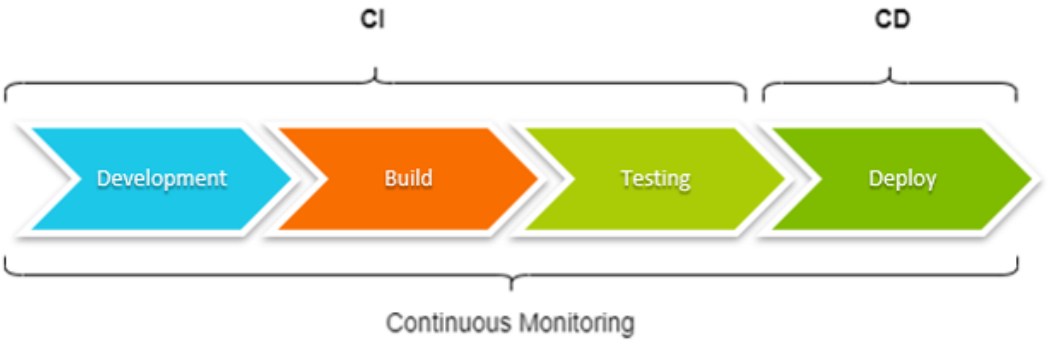
- Praksa razvoja softvera gde se praktikuje redovno spajanje izmena koda na centralni repozitorijum, nakon čega se pokreću automatizovane build skripte i testovi
- Cilj je brže pronalaženje i rešavanje grešaka, poboljšanje kvaliteta softvera i smanjenje vremena potrebnog za validaciju i release nove verzije

Continuous Delivery – CD

- Praksa gde se izmene koda automatski build-uju, testiraju i pripremaju za puštanje u produkciju
- Proširuje CI praksu tako što se sve izmene direktno puštaju na testno i/ili produkciono okruženje nakon build faze

CI/CD Pipline

- Niz koraka koji se moraju izvršiti da bi se isporučila nova verzija softvera
- Uvodi automatizaciju za unapredjenje procesa razvoja aplikacija, posebno u fazama integracije, testiranja i isporuke



(CiCD Pipeline)

Uvod

- Postoji razni alati za CI/CD
- Vecina pruža integraciju sa postojiećim repoima
- Neki su besplatni neki ne
- Github Actions je jedan od tih alata koji pruža direktnu integraicju sa git repo-o ali i Docker-om
- Razvoj pipeline-a je prilično brz i relativno jednostavan

Cilj

- Prva stvar koja je potrebna da se uradi jeste da imate projekat na Github-u
- Drga stvar koja je potrebdno da se uradi jeste da iamte nalog na Dockerhub-u
- Cilje nam je da na svai push na master granu, pokrenemo pipline
- Ako bude uspešan, da nam docker image završ̌i na dockerhub-u da ga možemo preuzeti
- Ne želimo da mi ručno stalno bildujemo image i kontejnere

Dockerhub access token

- Prva stvar koja je potrebna da se napravi jeste dockerhub access token
- Ona nam treba da bi github mogao da se prijavi na vašistem i da pošalje kreiran image
- Na adresi možete kreirati vaš access token
- Napomena: morate biti ulogovani da bi kreirai access token

Dockerhub projekat

- Potrebno je kreirati dockerhub projekat da bi github actions bio sposoban da pushuje isprabvno buidovane image
- Kasnije ćemo koristiti ove image da ih skinemo i da pokrenemo aplikaciju
- Ne želimo da mi ručno build-ujemo, želimo da automatizujemo procese
- Projekat je potrebno nazvati onako kako tagujete vaš image tokom build procesa
- o tome voditi računa, ako se imena razlikuju do push-a na dockerhub neće doći!

Github SECRETS

- Kada smo dobili dockerhub access token, potrebno je da naše podatke sačuvamo u githu-u
- Ove informaciej neće čuvatu u otvorenom obliku, te je potrebno da narapraivmo par SECRETS-a na koje treba da se referenciramo
- U tab-u *Settings* vašeg projekta izabrati opciju *Secrets* pa zatim *Actions*
- Kreirati prvu varijablu *DOCKERHUB_TOKEN* i kao vrednost upisati dobijeni access token
- Nako toga kreirati drugu varijablu *DOCKERHUB_USERNAME* i upisati vaš username tj username sa dockerhub-a

Workfof – pipline

- Kada ste kreirai SECRETS, potrebno je da kreirate folder *.github/workflows*
- Unutar ovog foldera potrebno je kreirati jedan yaml fajl *push.yaml*
- Ovaj fajl će sadržati sve naredbe koje vašpipline treba da sadrži
- Ovaj posao može da se uradi i kroz grafičko okruženje
- Kada kreirate ovaj fajl dodaje se sve u radno stablo git-a i pushuje se na repo
- Build će automatksi početi da izvršava vaše naredne
- U tabu *Actions* možete videti da li je vaš build prošao ili ne

YAML file

```yaml
name: ci

on:
 push:
  branches:
    - 'master'

jobs:
 docker:
  runs-on: ubuntu-latest
   steps:
     -
      name: Set up QEMU
      uses: docker/setup-qemu-action@v2
     -
      name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2
     -
     name: Login to DockerHub
     uses: docker/login-action@v2
     with:
      username: ${{ secrets.DOCKERHUB_USERNAME }}
      password: ${{ secrets.DOCKERHUB_TOKEN }}
     -
      name: Build and push
      uses: docker/build-push-action@v3
      with:
       push: true
       tags: ${{ secrets.DOCKERHUB_USERNAME }}/gorest:latest
```

Završnica

- želimo da buildujemo samo sa main/master grane, isamo taj image treba da se pušuje na dockerhub
- U nekim drugim sitaucijama možemo pushovati i develop granu, da bi ostali mogli da testiraju aplikaciju
- Svaki put kada spokijte stvari na master/main granu i uradite push dolazi do okidanja pipeline-a i svaki put se izvršva
- Stoga treba voditiu računa o build procesu i pipeline-u svaki put kada uradite push na master/main granu

Dodatni materijali

- CONTINUOUS INTEGRATION thoughtworks
- Build and push docker images
- Build CI/CD pipelines in Go with github actions and Docker
- Tokenbucket
- Understanding Rate Limiting Algorithms
- Managing Your Data Lifecycle with Time to Live Tables
- An Improved Token Bucket Algorithm for Service Gateway Traffic Limiting

Kraj predavanja

Pitanja? :)