

Alati za razvoj softvera

Idempotentnost, Imutabilnost, Metrike, Prometheus



Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Uvod

- ▶ Uključivanje idempotencije je važan deo izgradnje robusnih i predvidljivih API-ja
- ▶ U osnovi, idempotencija je scenario koliko god puta se radnja izvršiti, ishod ostaje isti

`f(x) = x*x ; f(5) = 25 // uvek je isto ma koliko puta pozvali`

- ▶ Kada govorimo o idempotenciji u kontekstu HTTP-a, drugi termin koji se pojavljuje je bezbednost
- ▶ Ovde se pod sigurnošću misli na to da zahtev ne mutira podatke prilikom poziva
- ▶ Tabela u nastavku prikazuje najčešće korišćene HTTP metode i njihovu sigurnost i idempotenciju.

Http Metod	Sigurnost	Idempotentnost
GET	DA	DA
PUT	NE	DA
POST	NE	NE
DELETE	NE	DA
PATCH	NE	NE

Rešenje

- ▶ Mreža nije pouzdana, zahtevi mogu da se dupliraju
- ▶ Ovo može da izazove problem kada naš sistem radi sa osetljivim entitemia (novac)
- ▶ Srećom rešenje je relativno jednostavno
- ▶ Uvešćemo identifikator koji će nam jedinstveno identifikovati zahteve
- ▶ Server treba samo da vodi evidenciju o zahtevima koji su izvršeni
- ▶ Ako je zahtev kod nekim identifikatorom već izvršen, server samo vrati korisniku OK
- ▶ Ako takav identifikator server nikada nije video, onda može da pusti zahtev dalje
- ▶ Ovaj identifikator se obično salje u header delu zahteva kod kjučem **x-idempotency-key**

Uvod

- ▶ Pod ovim pojmom se misli da se podaci neće menjati polovično, nego u potpunosti
- ▶ Znači da sistem ne podržma *in place* izmene, nego izmena znači da se menja čitava vrednost
- ▶ Ovo nije loša ideja, pogotovo kada radimo sa bazama podataka koje ne podržavaju klasične transakcije
- ▶ Ili prosto želimo da se zapis desi brzo
- ▶ Ako razmišljamo u kontekstu NoSQL baze o kojoj smo pričali, to znači da sve što treba da uradimo je da pod istim ključem samo upišemo celu novu vrednost
- ▶ I na taj način smo podržali ovu operaciju

Uvod

- ▶ Metrike su vrlo važan deo svake aplikacije koja se izvršava u realnim uslovima
- ▶ Bitno nam je da znamo koliko resursa trošimo, koliko korisnika imamo, koliko aktivnih, koliko zahteva po sekundi itd.
- ▶ Ovi zahtevi nam pomažu da poboljšamo naš sistem, ili da razvijemo nove funkcije
- ▶ Metrike možemo skupljati na nekoliko načina
- ▶ Docker nam omogućava uvid koliko naši kontejneri troše resursa, videli smo da tracing može da nam da neke informacije, a možemo da ugradimo i neke metrike u sam kod

- ▶ Metrike koje dodajemo u sam kod, su obično middleware-i, odnosno omotači oko naših handler-a ili funkcija
- ▶ Cela ideja je da skupimo informacije za svaki handler i da vidimo koliko puta se poziva, koliko traje poziv isl.
- ▶ Alati kao što su Prometheus ili Influx nam omogućavaju da te metrike skupimo na jedno mesto da pratimo u realnom vremenu kako se metrike ponašaju, da postavljamo alert mehanizme ako prekoračimo neke granice

Uvod

- ▶ Prometheus je besplatna softverska aplikacija koja se koristi za *event monitoring*, *alerting*
- ▶ Beleži metriku u realnom vremenu u bazi podataka vremenskih serija
- ▶ Koristi *HTTP pull* model, sa fleksibilnim upitima i *alerting* u realnom vremenu
- ▶ Napisan u *Go*-u
- ▶ Inspirisan je radovima i alatima iz *Google*-a, pre svega alatom *Borgmon*

Upotreba

► Dodati element u *Docker compose*

```
prometheus:  
  image: prom/prometheus:latest  
  ports:  
  - '9090:9090'  
  volumes:  
  - ./prometheus:/etc/prometheus  
  - ./prometheus-data:/prometheus  
volumes:  
  prometheus-data:
```

Konfiguracija

- ▶ Dodati *prometheus.yml* fajl sa elementima

```
scrape_configs:  
- job_name: myapp  
  scrape_interval: 10s  
  static_configs:  
    - targets:  
      - app:8000
```

Definisanje potrebnih elemenata

```
var (  
    currentCount = 0  
    httpHits = prometheus.NewCounter(  
        prometheus.CounterOpts{  
            Name: "my_app_http_hit_total",  
            Help: "Total number of http hits.",  
        },  
    )  
  
    metricsList = []prometheus.Collector{httpHits}  
    prometheusRegistry = prometheus.NewRegistry()  
)
```

```
func init() {  
    // Register metrics that will be exposed.  
    prometheusRegistry.MustRegister(metricsList...)  
}  
  
func metricsHandler() http.Handler {  
    return promhttp.HandlerFor(prometheusRegistry, promhttp.HandlerOpts{})  
}  
  
func count(f func(http.ResponseWriter, *http.Request)) func(http.ResponseWriter, *http.Request) {  
    return func(w http.ResponseWriter, r *http.Request) {  
        httpHits.Inc()  
        f(w, r) // original function call  
    }  
}
```

Upotreba

```
router.HandleFunc("/post/", count(server.postHandler)).Methods("POST")
...
//scrape metrics from service , show UI on localhost:9090
router.Path("/metrics").Handler(metricsHandler())
```

Dodatni materijali

- ▶ Immutability Changes Everything
- ▶ How to achieve idempotency in POST method?
- ▶ Idempotent APIs
- ▶ An Introduction to Prometheus Monitoring
- ▶ INSTRUMENTING A GO APPLICATION FOR PROMETHEUS
- ▶ Golang Application monitoring using Prometheus
- ▶ Monitoring you Golang server with Prometheus and Grafana

Kraj predavanja

Pitanja? :)