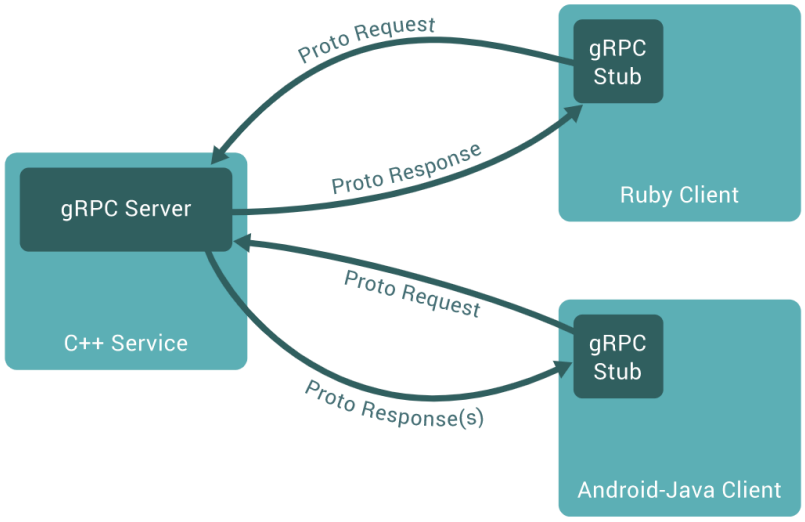
Mikroservisi part 3

1. gRPC services

gRPC je moderan, izuzetno brz, open source framework za izradu servisno orijentisanih aplikacija (mikroservisa) zasnovan na Remote Procedure Call (RPC) principima, odnosno pozivima metoda udaljenog objekta razvijen od strane kompanije Google. Ovaj framework omogućava slanje izuzetno malih poruka, bidirekcioni streaming, http/2 način komunikacije, ekosistem oko njega je izuzetno bogat i moguće ga je proširiti koristeći plugin mehanizam.



Servise je moguće implementirati u izuzetno velikom broju jezika, dok je specifikacija poruka nezavisna od samog jezika. Specificirane poruke bivaju generisane i dostupne korisniku za upotrebu, a podržana je i backward compatibility sa prethodnim verzijama poruka.

2. Protobuf

gRPC framework ne koristi JSON ili XML format za prenos poruka između servisa, vec svoj **binaran tip** koji se zove **protobuf**. Definicija protobuf poruka je izuzetno jednostavna i za te potrebe se koristi DSL iz čega se dobija tip poruke za odgovarajući jezik ili jezike. Pored poruka, u istom file-u, možemo definisati i specifikacije servisa odnosno koje poruke naši servisi prihvataju, ali i šta su povratne vrednosti naših servisa. Listing 1, prikazuje primer definicije servisa sa svojim porukama koristeći protobuf.

```
// Definicija gRPC servisa
service Greeter {
  // Definicija rpc metode
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// Definicija protobuf poruke koja će biti ulazni podatak za
servis
message HelloRequest {
  string name = 1;
}

// Definicija povratne vrednosti našeg servisa
message HelloReply {
  string message = 1;
}
```

Listing 1, primer jednog gRPc servisa sa protobuf porukama.

U prethodno definisanom primeru, vidimo jednostavan gRPC servis koji prima jednu protobuf poruku koja ima samo jedan atribut tipa *string*, i poruku koja reprezentuje povratnu vrednost našeg servisa koja takođe ima jedan atribut tipa *string.* Naravno, protobuf nije ograničen samo na tip string, možemi koristiti i druge tipove podataka, ali isto tako može da ugnježdimo i druge poruke da bi dobili kompleksniju poruku ili odgovor. Moguće opcije za atributed možete videti u samoj dokumentaciji. Bitno je napomenuti, da gRPC servisi prihvataju tip *Message*, tako da čak i ako žeimo da samo vratimo jedan podatak kao u prethodnom primeru, taj atrobut

moramo da “obmotamo” u Message tip, dok je naziv poruke proizvoljan.

Nakon što smo specificirali naše poruke i servise, potrebno je generisati kod za željeni jezik. to se radi koristeći alat *protoc.* Ovaj alat je potrebno instarliati na vašu mašinu. Primer instalacije za winows možete videti na linku, za unix like operativne sisteme (max/ linux) možete naći na linku.

Kada je *protoc* instaliran na vašu radnu mašinu (a definisali ste vaše gRPC servise i protobuf poruke u fajlu sa ekstenziom *.proto*), možete pristupiti generisanju potrebnih elemenata za vaš željeni jezik. Primer generisanja za programski jezik go možete naći na linku, kao i na linku (sekcija Generating client and server code). Ako generisanje prođe bez problema, ono što je dalje potrebno jeste da definišete server (golang struktura), koja implementira sve metode definisane u vašem servisu. Listing 2 daje primer definicije servera sa metodom za proto specifikaciju iz listing-a 1.

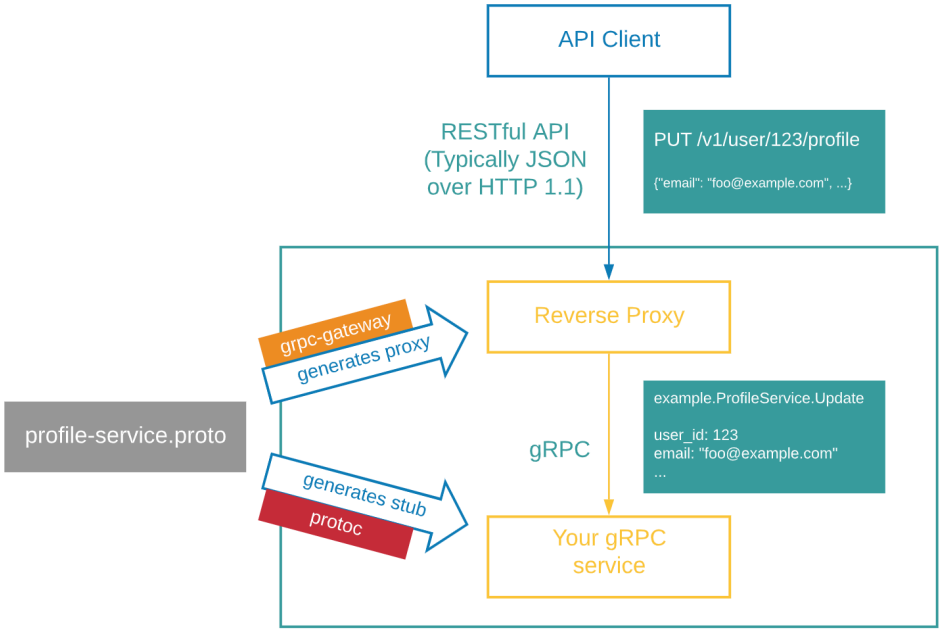
```
type server struct {
        ...
}

func (s *server) SayHello(ctx context.Context, in
*pb.HelloRequest) (*pb.HelloReply, error) {
        return &pb.HelloReply{Message: "Hello again " +
in.GetName()}, nil
}
```

Jedna napomena jeste da će sadržaj generisanja završiti na mestu gde vi specificirate i pod pod ekstenziom *naziv_proto_fajla.pb.go*, AKO niste specificirali drugačije. u prethodno definisanom fajlu pb.HelloRequest je poruka koja se nalazi unutar *pb* biblioteke, gde je *pb* skraćeni naziv za punu putanju gde se generisani *.pb.go fajl nalazi. Kompletan primer možete videti na linku, kao i primer generisanog klijenta. Pored servera, i poruka protoc generiše i klijent koji možete koristiti da pozivate druge service. Isto kao i kod REST-a, ako imate dva servisa koja treba da komuniciju u tom slučaju jedan je klijent (traži uslugu), a drugi je server (obrađuje sadržaj).

3. gRPC Gateway

Pošto gRPC ne koristi JSON, XML ili neki drugi tekstualni tip poruka već binarni, ponekad može biti nezgodno testirati vašu aplikaciju ili može biti nezgodno da se vaša aplikacija otvori va veći krug klijenata. Jedna opcija je da oni koriste gRPC takođe, međutim ako to nije opcija mora postojati način da se uradi translacija JSON-protobuf i obratno. Tu operaciju radi gRPC Gateway plugin.



Kao i kod standardnog gRPC-a, isto koristimo proto file za specifikaciju našeg gateway-a. Pre upotrbe ovog alata, potrebno je instalirati par biblioteka koje će sve to omogućiti. Primer možete videti na linku.

Potrebno je proširiti **.proto* file dodatnom sintakom koja će biti korišćena za translaciju JSON-a u proto poruke. Listing 3 daje primer dodatnih elemenata.

```
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {
    option (google.api.http) = {
```

```
      post: "/v1/example/echo"
      body: "*"
    };
  }
}
```

Ovde možemo da vidimo da je prethodno definisani servis proširen za par elemenata gde vidmo da za HTTP POST zahtev se koristi */v1/example/eho* putanja i da prihvata sve poruke u body. Isto tako mo\emo da specificiramo i ostale HTTP metode.

Druga razlika je u tome, što moramo malo proširiti naš go kod, da pokrenemo web server koji će prihvatiti HTTP zahteve spoljnog sveta i prebaciti ih u gRPC i protobuf zahtev i propagirati dalje u naš sistem. Primer možete videti na linku.

gRPC ima mnošto korisnih plugina koje možete da iskoristite i postoji dosta dostupnog middleware-a za bezbednost i proveru raznih stvaru u vašim zahtevima. U početku, oni mogu da deluju malo konfuzno i komplikovano ali kada se naviknete oni će vam znatno olakšati posao.

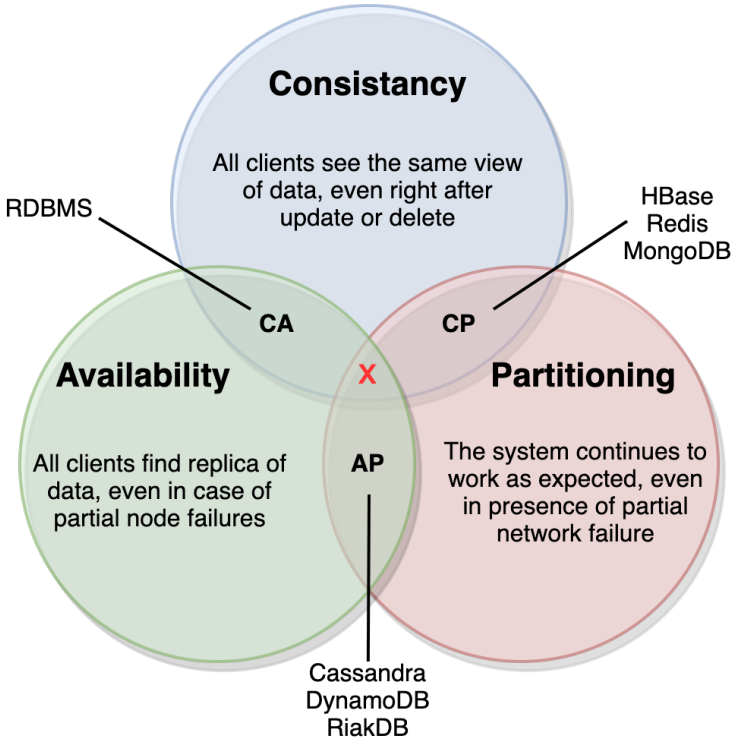
Svi funkcionišu na sličan način, a to je presretanje zahteva, implementacija nekakve logike i propagacija zahteva dalje ili ne u zavisnosti od tipa middleware-a. Primere za razne biblioteke možete naći na linku. A uvek možete implementirati i vaš, ako za time image potrebe :).

NAPOMENA: gRPC Gateway ne morate da koristite uvek i za sve servise. Ovu biblioteku možete da koristite kao ulaznu tačku u vaš sistem. Obično je praksa, da se spoljom svetu ponudi HTTP endpoint da bi zadovoljli veliki broj klijenata, a ostatak sistema je implementiran koristećo gRPC i protobuf, i servisi komunicirjau na taj način.

4. NoSQL baze

NoSQL pokret reprezentuje sve tipove baza podatka koje nisu relacione baze podataka. Njihove model skladištenja, upita i svih ostalih operacija razlikuje se dosta od relacionih baza.

Na važnosti počinju da dobijaju još u ranim nastancima cloud computing-a, iz prostog razloga nemogućnosti relacionih baza da se primere i distribuiranim sistemima zbog čuvene CAP teoreme definisne od strane dr Eric Brewer-a. Ova teorema kaže da ne možemo zadovoljiti sve tri osobine: Konzistentnost, Dostupnost i Particioniranje, i da uvek možemo da dobijemo najviše dve od tri osobine. Na korisnicima je da odluče šta od tih osobina žele da imaju, pošto particioniranje nije moguće izbeći u distribuniram sistemima. Slika 3 prikazuje CAP teoremu, sa osobinama i predstavnicima baza podataka.



Ključna stvar kod ovih tipova podataka jeste da **poznajete svoje upite**, to znači da unapred znate kakve upite želite da radite nad vašim podacima i da na taj način modelujete vašu bazu. Ovi

tipovi baza obično ne podržavaju join operacije niti imaju transakcije kao relacione baze i tako striktnu formu. Zbog ovih osobina izuzetno se lako koriste u mikroservisima iz prostog razloga mogućnosti da vaš domen problema prilagodite bazi a ne da se dovijate u fiksnom modelu relacionih baza.

To ne znači na transakcije nije moguće implementirati, ali se one obično implementiraju na aplikativnom sloju (npr. SAGA patern), ali moguće je i da budu deo baze kao recimo distribuirane transkacije (2 phase commit, 3 phase commit, razni drugi algoritmi distribiurnaih sisitema).

A. Consul Key-Value store

Consul je jedna od mnoštvo NoSQL baza podataka koje koriste Key-value model podataka. Ovaj model je među najstarijim modelima korišćen je dosta i u samom UNIX operativnom sistemu (link na predavanje dr Eric Brewer-a).

Sve što vam ovaj model omogućava jeste da pod nekakvim ključem tipa string, sačuvate proizvoljan niz bajtova tj. bilo koji podatak. Isto tako, ako tražite vrednost pod odgovarajućem ključem dobićete nazad kompletan podatak koji se čuva pod tim ključem. Izuzetno jednostavan model, izuzeno koristan model za razne alate vrlo sličan strutiri podataka *hash map*.

Ovaj model ima i Time to Live mehanizam, odnosno da vremenski čuva određeni podatak, što može biti zgodno za implementaciju određenih funkcionalnosti vaših aplikacija (npr story gde je story dostupan 24h). Ova operacija nije tako jednostavna za implementriati u relacionij bazi, i obično zahteva upotrebu još jednog alata.

Ova baza, kao i mnoge druge Key-Value podržavaju i **prefix scan,** odnosno pretraga svih ključeva na osnovu prefixa. Koristeći ovu metodu možete recimo jednostavno vratiti sve postove korisnika ili šta god da vam treba. Naravno, morate voditi računa kako dizajnirate vaše ključeve da bi podržali ovu opciju.

5. Dodatni materijali

1) https://towardsdatascience.com/grpc-in-golang-bb40396eb8b1
2) https://medium.com/@nate510/structuring-go-grpc-microservices-dd176fdf28d0
3) https://blog.lelonek.me/a-brief-introduction-to-grpc-in-go-e66e596fe244
4) https://bitbucket.org/blog/writing-a-microservice-in-golang-which-communicates-over-grpc
5) https://pkg.go.dev/github.com/hashicorp/consul/api#readme-usage
6) gRPC Youtube list
7) https://alex.dzyoba.com/blog/go-consul-service/
8) https://www.redhat.com/architect/apis-soap-rest-graphql-grpc?sc_cid=7016000000127ECAAY&fbclid=IwAR3U9FHvg2JiAqXnVGvn34L37YiMciMNeROOhBl4y1-uo6ppNyDnCGe64M4