Ticket service cache

Martin Gajdoš, 22.4.2024

# Úvod

Snaha v našej architektúre zmeniť systém využitia triggerov(INS, UPD) v databáze a odosielanie do PUB-SUB redis a replikácií pre ostatné farmy. Aktuálne nám ostali 3 služby, ktoré by trebalo aktualizovať podobne ako OddsOffer cache.

* **TiketCacheUpdater –** spracová zmeny a ukladá to ticketovej cache -> Redis
* **OfferSignalR** – posielanbie zmien kurzov na FE.
* **BuyBackProcessor** – auto odkup tiketov

Doterajší systém funguje na vkladaní do databázovej queue pri každej jednej zmene nad hlavnými entitami. Cieľom bude odstrániť databázové queue a posielanie do redis pub-sub. *Tiket redis ostane pre rozrobené tikety ostane zachovaný.*

## Aktuálna architektúra: TiketCacheUpdater služby

Služba nám beží na CF nodes. Číta správy z redisu a vkladá dáta do ďalšieho redisu -> feed master, ktorý je replikovaný na farmy(SVK, NGR,TZN). Farmy mali fungovať čo najviac fungovať čo najrýchlejšie. Vybudoval sa root CF -> a ten sa replikuje na farmy na železá deploynute geograficky.

A diagram of a computer network

Description automatically generated

Obrázok 1: Aktuálne riešenie

Celá architektúra je založená na ukladaní dát do databázy a v DB su pripravené triggre(echo) -> ktoré propagujú dáta na PUB-SUB redis -> ďalej to smeruje na CF redis ktorý je replikovaný na všetky farmy a tam sa vyčítava cache a tiketová služba pracuje s týmito dátami. Problém nastáva pri rastúcom dopyte a rastu ponuky feedov a providerov. Snaha je decentralizovať dáta.

Hlavným nedostatok našej aktuálnej architektúry je výstup databázy, ktorá funguje na triggeroch pri INS a UPD.

A computer code with many colored text

Description automatically generated with medium confidence

Obrázok 2: Trigger výflus

## Riešenie: Nová ticket cache

Budeme sa snažiť využiť existujúcu **central-cache(Odds cache)**, z dôvodu využitia existujúcich dát budeme vedieť pre použiť tieto dáta. Obsluha náberu na tiket -> volanie jeden endpoint. A lablovanie všetky popiskou pre jeden odd a môžeme presúvať zacachované popisky. Centrálna cache zatiaľ obsahuje všetky volania pre offeru. Budeme musieť pridať nový endpoint pre náber na tiket pre tiketovú službu. Nevýhodou môže byť **kontencia**, z hľadiska geografického problému. Riešenie je vytvoriť a oddeliť farmy inštancií na rôzne farmy. Pre odd budeme treba doťahovať všetky popisky čo bude treba na tiket a dáme do cache pre tiket.

## Ciele

1. **Analýza a zbez požiadaviek**
2. **Endpoint pre tiketovú cache** (REST API - ladenie + gRPC)
   1. Príprava vnútornej struktúry: popisky, cielový stav (3 statusy -> jeden výsledny), limity
3. **Príprava gRPC dema** -> pre gRPC volania Client a Server
4. **Úprava tiketovej služby** – Juraj Kuráň, Milan Benický
5. **Analýza BuyBack služby** – ako to celé funguje. Ostane sama na seba. Konektivita na kafku(client) prepočty z kafka dát.
6. **Úprava využitia SignalR** -> just in time **pri zmene odu na tikete**

# CIEL 1: Tiketová služba

## Požiadavky: Tiketová služba

Pre správny chod tiketovej služby budeme musieť posielať dáta v žiadúcom formáte dopytované cez central-cache API(ODDs cache) pomocou REST alebo do budúcna využitie gRPC volaní -> jedná o requestov medzi našimi BE službami. Aktuálny framework je na verzií.NET 4.8 a využíva **SOAP** pre webové volania. Z toho dôvodu bude potrebné otestovať vytváranie gRPC klienta. Tiketová služba ako sme spomenuli vyššie využíva Tiketovú cache založenú na PUB-SUB a výflusu z DB pomocou triggrov. Toto by sme mali nahradiť pridaním funkcionality a decentralizácie smerom na ODDs cache.

A screenshot of a computer

Description automatically generated

Obrázok 3: Upgrade riešenia

Na základe tohto riešenia decentralizujem a decauple hlavné časti PUBSUB redisu a triggera nad DB. A prenecháme to notifikácií iba Odds cache, ktorá si tieto dôležité dáta uchováva pre event.

A screen shot of a device

Description automatically generated

Obrázok 4: Základná entita spracovanie dotazov

Hlavná časť pre tiketovú službu je pracovať s tiketom a pridanými kurzami na tiket. Ako keď si zoberieme papierik a zapíšem na lístoček(tiket) rôzne tipy a ich kurzy sa následne dotiahnu v systéme.

A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

Na obrázku nad môžeme vidieť základnú entitu, ktorá sa berie z našeho PUB/SUB redisu a je aktualizovaná pomocou výflusu z DB pri zmene nad entitami EVT, EVCT a ODDs.

### Tiketová služba potrebuje integrovať:

* **Endpoint** zatiaľ iba jeden na všetko ohľadom: popiskou, stav, kurz datum udalosti, kategorizacia a obmezdenia
* Medzi **popisky** patrí zo záznomov z tabuliek
  + be\_tTip (Definition)
  + be\_tEventChanceType
  + be\_tEvent
  + be\_tLeagueCup
  + be\_tRegion
  + be\_tSport
  + be\_tSportEvent

Príklad: select top 1 OddsID, EventChanceTypeID, EventID, LeagueCupID, RegionID, SportID, SportEventID, OddChanceTypeID from be\_tOdds where oddsID = 526345

* **stav**
  + stav z tabulky be\_tOdds (status, be\_tEventChanceType (status), be\_tEvent (status, LiveExport, PrematchExport, LiveBetStatus, EventDate)
  + be\_tFeedSetting
  + be\_tFeedSettingSiteForbidden
* **kurz**
  + be\_tOdds.OddsRate
* **datum udalosti**
  + be\_tEvent.EventDate
* **kategorizacia**
  + be\_tOdds (OddsID, EventChanceTypeID, EventID, LeagueCupID, RegionID, SportID, SportEventID, OddChanceTypeID)
  + be\_tOdds.LiveBetting
* **obmedzenia**
  + select top 10 \* from be\_tEventRestriction
  + select top 10 \* from be\_tEventRestrictionExt
  + select top 10 \* from be\_tEventRestrictionExtEvct
  + select top 10 \* from be\_tEventRestricted
  + select top 10 \* from be\_tLeagueCupRestricted
  + select top 10 \* from be\_tLeagueCupRestrictionExt
  + select top 10 \* from be\_tLeagueCupRestrictionExtEvct
* nacitat do cache Limity (obmedzenia/restrikcie) po vlozeni udalosti do DB (cast DataBinder)
* BetManager - pridat kontrolu zmenie limitacii. <https://betmanager-cf.doxxbet.sk/Pages/Betting/LeagueCup/List.aspx>, karta obmedzenia
* limity - [dbo].[tc\_spGetEventLimitsCentral] storka na cf - vykradnut logiku

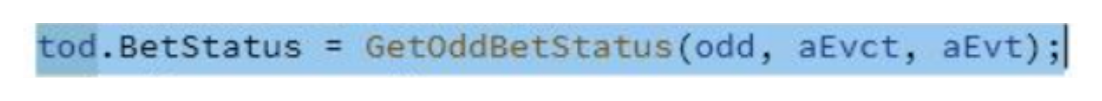
## Požiadavka: Micro informácia o kurze

V tejto časti si ukážeme potrebné volanie na našu odds cache. Jedná sa o veľmi časté volanie zo strany tiketovej služby. Samotná služba potrebuje tieto dáta na zisťovanie typovaťelnosti kurzu na tikete a jeho aktuálnu hodnotu a prepočty. Toto volanie by malo obsahovať najzákladnejšie dáta pre prácu s týmito funckiami prepočtu a zisťovania stavu. Túto požiadavku vieme elegantne vyriešiť pomocou pridania endpointu pre našu REST API na TiketCacheContoller.

A screenshot of a phone

Description automatically generated

Obrázok 5: GetMicroOddInfo

Hlavnou časťou je dostať informácie o zmene kurzu a bussines status, ktorý sa vyskladáva nasledovne pomocou funckie. 

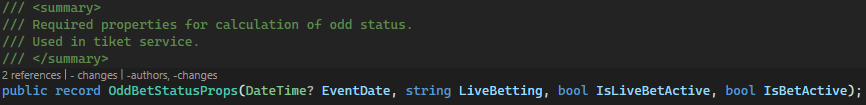
A screenshot of a computer

Description automatically generated

Obrázok 6: Bussines model pre stav odd

Pri vypočítavaní samotného bussiness stavu kurzu treba prihliadať na priradený kurz, ktorý je na samotnom dopytovanom kurze. Pre takéto kurzy kde RelatedOddsID nieje null je potrebné to počítať z nadradeného predka. Aby tiketová služba vedela spravne spracovať funckiu na obrázku č.6 tak potrebuje aby sme jej dodali potrebné súčasti ktoré majú byť na kurze.





Obrázok 7: BetStatusProps

Výsledok volanie z našej OddsCache je následovný =>

A screenshot of a computer

Description automatically generated

Obrázok 8: Response MicroOddInfo

## Požiadavka: Kategorizácia a ostatné data

Následne tiketová služba potrebuje aj ďalšie informácie o kurze aby vedela správne pracovať. Potrebujeme si posielať aj kategorizáciu pre daný odds na tikete. Tým sa rozumie preklad a popisky pre celý strom(sport ->region-> liga) a následne aj iné veci ktoré sú potrebné na zobrazenie tiketu na FE.

A screenshot of a computer

Description automatically generated

Obrázok 9: TiIket vyzualizácia

A screenshot of a computer

Description automatically generated

Obrázok 10: Základne body pre ostatné dáta

V našej základnej schéma chýbajú potrebné popisky pre strom alebo kategorizáciu. Ako sme spomenuli [vyššie](#_Tiketová_služba_potrebuje). Máme dve možnosti implementácie riešenia. Musíme zvážiť aj kvantitu dát pri rožširovaní ponuky a providerov.

Prvé riešenie je si vyberať celý strom aj s ID hodnotami a popiskami per jazykovú mutáciu. Tým ale nám môže narásť veľký počet dát a nevieme ani koľko by sme nastavili TTL(čas potrebný na držanie dát v cache).

A screenshot of a computer screen

Description automatically generated

Obrázok 11: 1. riešenie popisky

Druhé riešenie je spraviť ďalšia volanie na REST API. Kedže kategorizácia je rovnaké pre všetky jazyky tak budeme môcť si tieto hodnoty cachovať a len požiadať o preklady a popisky pre dané ID hodnoty stromu kategorizácie.

A diagram of a diagram

Description automatically generated

**Budeme to posielať jedným endpointom na kategorizáciu. V odds cachi všetko budeme mať dostupné v RAM ako raw dáta.**

## Verziovanie API(Podľa potreby tiketovej služby!)

V tejto časti si budeme ukazovať a sledovať zmeny v našom vývoji pre tiketovú službu.

### Verzia č.1 pre tiketovú službu

A screenshot of a computer

Description automatically generated

Obrázok 12: Implementácia endpointy

A screen shot of a computer screen

Description automatically generated

Obrázok 13: Kompletné volanie pre kurz

Naše volania si vieme kontrolovať pomocou endpointu => [offer-oddscache-staging-cf.doxxbet.sk/cache-stats?includekafkastats=false](https://offer-oddscache-staging-cf.doxxbet.sk/cache-stats?includekafkastats=false). Máme k dispozíci všetky a posledných 5000 volaní na OddCache.

A screenshot of a computer

Description automatically generated

### Merge requests

Formát pridávania zmien na odds cache prebieha nasledovne.

A screenshot of a computer program

Description automatically generated

A screenshot of a phone

Description automatically generated

Obrázok 14: merge request template

# Ciel 3: gRPC Demo

## gRPC

Prípava volania medzi dvoma službami. Začíname na čistej lúke na ukážku pre naše potreby.

1. Možnosť REST API + gRPC aby sme mali vzdielanú cache

* BE Server: **.NET8** ako je naša central-cache bude to ako API a našiť tam aj gRCP listenera.
* BE Client. .**NET 4.8 framework** -> tiketová služba (Juraj Kuráň)

1. Možnosť REST API a oddelené gRPC server – samostatný server

Zdroje:

* <https://www.udemy.com/course/grpc-csharp/?couponCode=ST6MT42324>
* <https://www.udemy.com/course/microservices-architecture-and-implementation-on-dotnet/learn/lecture/42552156#questions>
* <https://www.youtube.com/watch?v=et_2NBk4N4Y>
* <https://dev.to/ahmedshahjr/next-level-application-development-mastering-grpc-in-net-8-327p>
* <https://dev.azure.com/BeHealth/_git/BeLab?path=/App/src/Services/BeLab.Authorization/BeLab.Authorization.Grcp> (server)
* <https://dev.azure.com/BeHealth/_git/BeLab?path=/App/src/Services/BeLab.LabStaff/BeLab.LabStaff.Application/BeLab.LabStaff.Application.csproj> (client)
* <https://www.linkedin.com/pulse/make-rest-api-using-grpc-thenetcore-vahid-alizadeh/>
* <https://github.com/ben-jamin-chen/aspnetcore-grpc-rest/tree/main/aspnetapp>
* <https://medium.com/@nomanali03100/grpc-crash-course-net-8-part-4-integration-with-web-api-99fd159250d3>

Zdrojový kód => <https://github.com/BratSvK/TicketViewer_gRPC_Demo/tree/main>

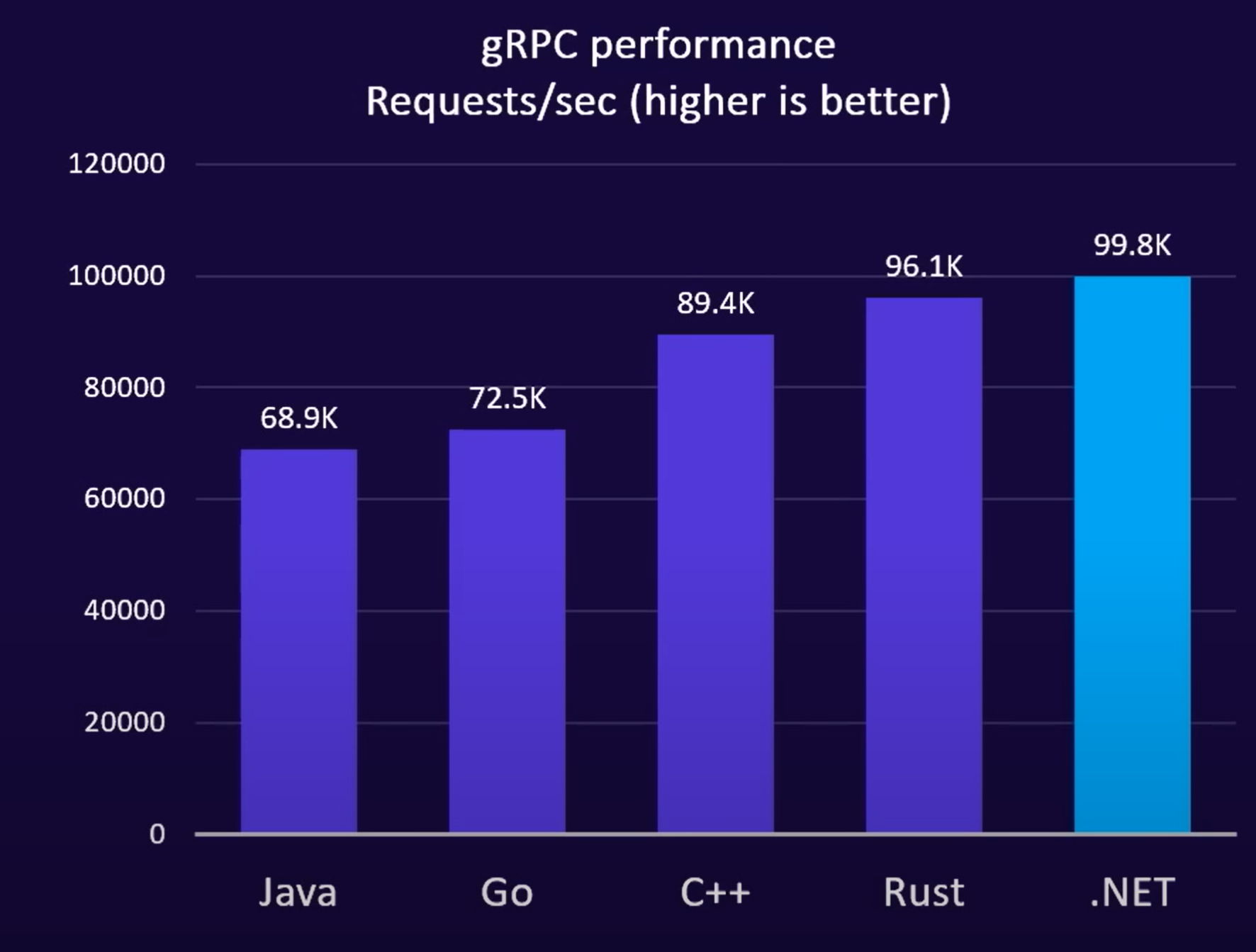
## Prečo aplikovať gRPC

Použitie gRPC pre spôsob komunikácie medzi službami je veľmi odporúčaný ak sa jedná o kritické miesta v našej architektúre. Pri našom riešení kde početnosť dotazov nadobúda vyššie hodnoty je teda použitie gRPC technológie na správnom mieste.

A white background with grey text

Description automatically generated

Obrázok 15: protobuf google syntax



Obrázok 16: .NET gRPC implementácia

### Čo to je gRPC?

Je mutliplatformový a vysoko-výkonný. Využíva protocol-buffers. Postavený na protokole http/2. Celá komunikácia je založená na .proto súboroch. Definujeme v ňom metódy a správy ktoré naša služba bude mať k dispozícií. Dôležité je poradie properties, použité pri binárnom serializovaní. Poradia do 16 sa ukladajú do jedného bitu(efektívnejšie na minimalizovanie správ a rýchlosť)

A screenshot of a computer program

Description automatically generated

Obrázok 17: .proto na server

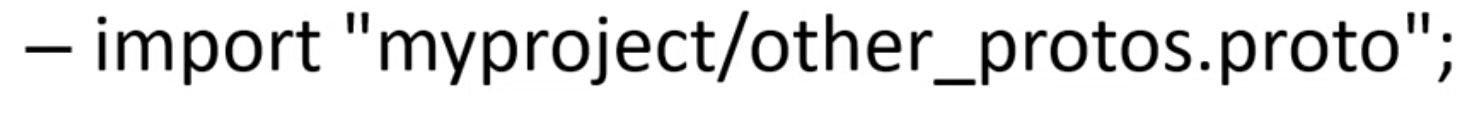
Typy, ktoré použiva GOOGLE .proto súbor.

A screenshot of a computer

Description automatically generated A screenshot of a computer code

Description automatically generated

Ak by sme chceli vlastné dátové typy tak bude treba importovať z nášho projektu svoje typy.



**Repeated** -> jedná sa o kolekciu.

A screenshot of a computer

Description automatically generated

Definícia služieb smerom kde klient volá metódu a tá je spracovaná na serveri.



Ďalšia hlavná výhoda gRPC .NET je pridanie gRPC JSON transcoding aby sme nemuseli robiť duplikáciu medzi RESTAPI službou a gRPC. Respektíve ak by sme mali gRPC server vieme pridať podporu REST volaní na tomto klientovi.

A group of symbols on a white background

Description automatically generated

Obrázok 18: gRPC + REST

A screenshot of a computer

Description automatically generated

Obrázok 19: gRPC client vs Rest client

### Implementácia gRPC v .NET

* Nutnosť definovať .proto súbor
* Automatické vygenerovanie štruktúr služieb na server
* Automatické vygenerovanie klienta

Dôležité nuget balíčky:

* <PackageReference Include="Grpc.AspNetCore" Version="2.62.0" /> (**server**)
* <PackageReference Include="Google.Protobuf" Version="3.21.5" /> (**client**)
* <PackageReference Include="Grpc.Net.ClientFactory" Version="2.49.0" /> (**client**)
* <PackageReference Include="Grpc.Tools" Version="2.49.0"> (**client**)

<PrivateAssets>all</PrivateAssets>

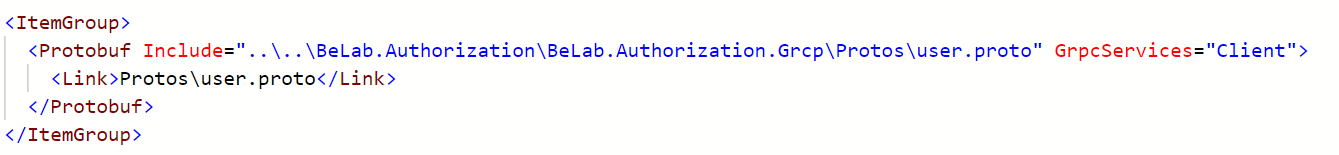
<IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitie</IncludeAssets>

</PackageReference>

A blue text on a white background

Description automatically generated

Obrázok 20: Server proto



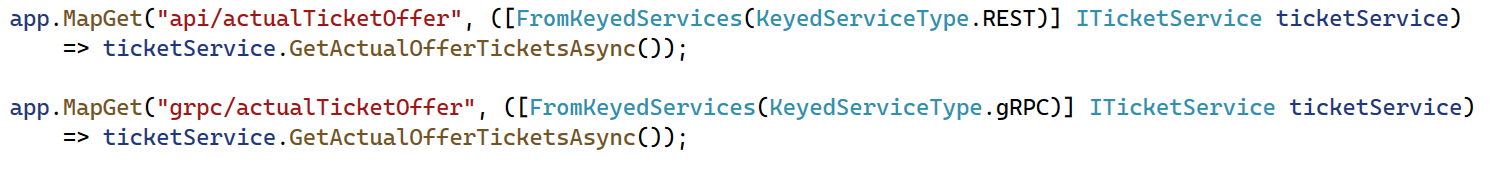
Obrázok 21: Client

V našom príklade si ukážeme gRPC na TicketViewer službe. Naša služba podporuje aj REST API volania na TicketOffer.API a gRPC.

A screen shot of a movie ticket

Description automatically generated

Obrázok 22: Demo schéma TicketViewer



Obrázok 23: Rozdelenie REST/gRPC

### Integrácia gRPP + WEB API(REST)

TicketOffer.API bude sa tváriť ako **hybrid**, ktorý nám ponúka na dvoch portoch REST + gRPC komunikáciu. Na to aby nám to išlo musíme nakonfigurovať náš API server pomocou Kestrel parametrov na ktorý portoch počúva na akom protokole. „**Kestrel**” endpoints poskytujú infraštruktúru na počúvanie prichádzajúcich požiadaviek a ich smerovanie na príslušné endpointy. Kombinácia adresy a protokolu definuje koncový bod.

Adresa špecifikuje sieťové rozhranie, na ktorom server počúva prichádzajúce požiadavky, napríklad TCP port. Protokol špecifikuje komunikáciu medzi klientom a serverom, ako napríklad HTTP/1.1, HTTP/2 alebo HTTP/3. Koncový bod môže byť zabezpečený pomocou https URL schémy alebo metódy UseHttps. Koncové body je možné konfigurovať pomocou URL adries, JSON v appsettings.json a kódom.

A screenshot of a computer program

Description automatically generated

Obrázok 24: gRPC server

Aby sme mali teda podporu dvoch verzií budeme musieť upraviť Kestrel nastavenia endpointov a listenerov pomocou appsettings konfigurácie.

A screen shot of a computer

Description automatically generated

Obrázok 25: Kestrel konfigurácia

Definícia protobuf služby pre TicketProtoService je odporúčané definovanie properties podľa dokumentácie: „*The [Protobuf style guide](https://developers.google.com/protocol-buffers/docs/style) recommends using underscore\_separated\_names for field names. New Protobuf messages created for .NET apps should follow the Protobuf style guidelines. .NET tooling automatically generates .NET types that use .NET naming standards. For example, a first\_name Protobuf field generates a FirstName .NET property“.*

Keď je aplikácia po builde, nástroje Protobuf generujú .NET typy z .proto súborov. Správa Person generuje .NET triedu nasledovne =>

A computer code with blue and black text

Description automatically generatedA computer code with blue text

Description automatically generated

Po úpešnom implementovaní servera a klienta sme dostali json odpoveď cez grpc volanie.

A screenshot of a computer program

Description automatically generated

Obrázok 26: Json volanie

Dôležité je poznamenať že Dátumy sa posielajú ako Timestamp v **UTC formáte**.

### Zmeny nad .proto (server-klient)

Webové služby často vyžadujú zmenu nad definovanými resourcami podľa potreby klienta. Tým pádom treba modifikovať server .proto súbor a zmeny aplikovať na všetkých klientoch. V tejto sekcií si ukážeme ako sa to správa pri našom gRPC protokole.[ <https://sanket-naik.medium.com/sharing-grpc-proto-files-with-nuget-packages-made-easy-dd366a094b25>.

Postup:

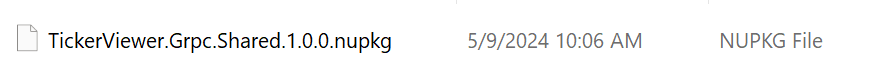
1. Vytvorenie nového projektu TicketViewer.Grpc.Shared(používané cez Nuget)
2. dotnet pack --configuration Release --output .\nupkg
3. Napojenie klientov na Nuget
   * Pridanie GeneratePathProperty="true" property to package reference.
   * Pridanie prefixu kde máme Protobuf referenciu $(PkgGrpc\_Shared)\content\( '\content' is the directory name where all the content files are available by default.)
   * Je to len convencia pre pridanie Grpc.Shared Nuget directory. {Pkg}{PackageName}(. Bodka sa oddeluje \_)

A diagram of a software

Description automatically generated

Obrázok 27: zdielaný proto súbor

Vyskúšame si to na našom deme. Naskôr si prekopírujeme Protos súbory do nového dll projektu a vytvoríme si nupkg pre upload do Nuget.org [https://www.nuget.org/packages/manage/upload ]



Po úspešnom uploade máme balíček nahratý ako NUGET na stránke <https://www.nuget.org/packages/TickerViewer.Grpc.Shared/1.0.0>

A screenshot of a computer

Description automatically generated

Teraz môžeme referenciovať náš proto súbor pre naších klientov a server.

A screen shot of a computer

Description automatically generated