gRPC Microservices v .NET Core

MGR. TOMÁŠ HAVETTA

Představení

- Jméno
- Firma
- Odpovědnost
- Typ aplikací
- Použitá verze VS a C#
- Zkušenosti z tvorby distribuovaných systémů (SOAP, REST, WEB Api, …)
- Očekávání od kurzu

Osnova kurzu

- Základy gRPC
- Protocol Buffers
- gRPC servis - nastavení
- gRPC stream
- gRPC Interceptors
- Řešení chyb
- Zabezpečení

Organizační informace

- Doba výuky
- Jídlo, nápoje, polední pauza
- Vstupní kód
- Parkování
- Kuřáci
- WC
- ???

1. Základy gRPC

- Od přenosu binárních dat k binárním datům
- Protocol Buffers
- HTTP/2
- gRPC stream
- Implementace jednoduché služby v .NET Core
- Implementace klienta

Historie výměny dat

- Binární proprietární data
- Text, CSV
- XML
- JSON
- Protocol Buffer

gRPC

- g „Remote Procedure Calls“
- Původně interní v Google systémech
- Od roku 2016 Open Source
- g znamená v každé verzi něco jiného (např. 1.21 je g = gandalf)
- gRPC má tři vrstvy
  - Transport – HTTP/2
  - Channel – má na starosti „calling conventions“ a implementaci mapování RPC na transportní vrstvu. Tady se řeší jen přenos x-bytů tam a zpět.
  - Stub – definuje rozhraní a datové typy (message)

Protocol Buffer

- Google používá od 2001, uvolněno v roce 2008
- IDL – Interface Description Language
- Klíčové pro Stub vrstvu gRPC
- Pomáhá realizovat
  - Formal contracts – popis dat
  - Code generation – generování kódu pro obě strany!
  - Bandwidth optimization – minimalizace přenášených dat
- Soubory s příponou .proto

Protocol Buffer - příklad

```
1   syntax="proto3";
2
3   message Osoba {
4       int32 id = 1;
5       string jmeno = 2;
6       string prijmeni = 3;
7       string mesto = 4;
8       repeated string telefon = 5;
9   }
```

HTTP/2

- Nová verze HTTP protokolu
- Vyvíjeno od roku 2009, standardizován 2015 (RFC 7540, 7541)
- Podpora binárního přenosu dat
- Podpora obousměrného streamu dat
- V případě streamu je veškerá komunikace na JEDNOM TCP/IP spojení
- Přes jedno spojení může proběhnout i několik streamů
- Významné zvýšení výkonu při velké zátěži serveru
- HTTPS pouze TLS 1.2

Podpora HTTP/2

- .NET Core 3.x
- IIS podporuje od verze 10 – Server 2016 a Windows 10
- gRPC na IIS je zatím v experimentální podobě – gRPC-Web

gRPC stream

- Použitelné pro šířené série zpráv
- Odpověď ze serveru nemusí být v jednom response, ale může přicházet postupně, teoreticky s neomezeným časovým rozestupem
- Implementaci zajišťuje HTTP/2

Microservices

- Další vývojový stupeň pro tvorbu SW
- Klíčové koncepty
  - Independent Deployability
  - Modeled Around a Business Domain
  - Owning Their Own State
  - Size and Flexibility

gRPC a .NET

- Knihovna Grpc.Core
  - Vývoj od roku 2016
  - Používá nativní C knihovny pro podporu HTTP/2
  - Funguje i na starém .NET Frameworku
  - Od 05/2021 v „maintanance“ režimu
- Knihovna grpc-dotnet
  - K dispozici od 2019
  - Čisté C# řešení využívající .NET Core 3.0 a vyšší
  - Od 05/2021 doporučené pro nové projekty
  - Připraveno k podpoře HTTP/3

Vytvoření jednoduché služby

- VS 2019
- Nový projekt gRPC service
- Projít soubory a kód

- Známá chyba:
  grpc nástroje chybně pracují s cestou, pokud obsahuje UTF-16 znaky s diakritikou. Vypíše chybu, že nenašel cestu k souboru
  Např. c:\Users\TomášHavetta je smrtící
- Oprava:

  Pomocí mklink –j vytvořit junction bez diakritiky na složku .nuget
  nastavit proměnnou NUGET_PACKAGES=C:\TomasDir\.nuget\packagesdf

Vytvoření clienta

- Vytvořit požadovaný projekt
- Přidat balíčky – Google.Protobuf, Grpc.Net.Client, Grpc.Tools
- Zkopírovat do projektu .proto file a případně upravit C# namespace
- Nastavit .proto souboru
  - Build => Protobuf compiler
  - gRPC Stub Classes => Client only
  - Compile Protobuf => yes
- Otestujte build a pak přidejte kód

Minimalistický kód klienta

```
Console.WriteLine("Ready for test");
Console.ReadKey();

using var channel = GrpcChannel.ForAddress("https://localhost:5001");
var client = new GrpcDemoTestClient.Greeter.GreeterClient(channel);
var reply = await client.SayHelloAsync(
    new GrpcDemoTestClient.HelloRequest() { Name = "Tomas" });

Console.WriteLine("Greeting: " + reply.Message);
```

LAB 1

- Vytvořte gRPC službu, která vrátí informaci o počasí podle kódu letiště.
- Vrátíte tyto informace
  - Teplota; Rychlost větru; Směr větru; Tlak
  - Textový popis upozornění
- Podporovaná letiště – PRG, BRQ, OSR, BTS, KSC, TAT
- Vytvořte klienta a otestujte komunikaci
- Použijte aktuální knihovnu grpc.AspNetCore

Blok 2 – Protocol Buffers

- Princip
- Základní typy
- Definovaní správ
- Verze

Princip

- Jednoduchá a přímočará, jazykově nezávislá definice datových typů
- Garantuje pro několik jazyků
  - Vygenerování datových typů
  - Vygenerování Message typů
  - Vygenerování služeb pro RPC volání
  - Generuje se jak klient, tak i server
- Binární serializace dat s minimalizací přenášených dat

Základní typy

Protobuf	C#	C++	Java	Python	Go
double	Double	double	double	float	*float64
int32	int	int32	int	int	*int32
int64	long	int64	long	long	*int64
uint32	uint	uint32	int	int	*uint32
uint64	ulong	uint64	long	long	*uint64
sint32	int	int32	int	int	*int32
sint64	long	int64	long	long	*int64
fixed32	int	uint32	int	int	*uint32
fixed64	long	uint64	long	long	*uint64
sfixed32	int	int32	int	int	*int32
sfixed64	long	int64	long	long	*int64
bool	bool	bool	boolean	bool	*bool
string	string	string	String	unicode	*string
bytes	ByteString	string	ByteString	str	[]byte

Použití skalárních typů

- int32/64 => nevhodné pro záporná čísla
- sint32/64 => vhodná pro záporná čísla
- fixed32/64 => vždy 4/8 bytů, vhodné pro hodnoty nad $2^{28}/2^{56}$

Pravidla názvů v .proto

- soubor => malá písmena, oddělit _ (xxxx_yyyy.proto)
- package => malým písmem, odpovídá cestě (místo / použít .)
- message => CamelCase
- field name => malé písmena, oddělovač _ (snake_case)
- repeated field => množné číslo
- service => CamelCase
- RPC method => CamelCase
- enum => CamelCase
- enum value => VELKA_PISMENA_S_PODTRZITKEM

Jak Protocol Buffers přenesou data

```
syntax="proto3";

message data {
    int32 id = 1;
    string meno = 2;
    int32 rok = 3;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <id>12</id>
    <meno>Kůrovec</meno>
    <rok>2010</rok>
</root>
```

60 bytů (bez xml directivy a white spaces)

```
08 0c 12 08 4b c5 af 72 6f 76 65 63 18 da 0f
```

15 bytů

08 0C

- 0000 1000
  - 00001 => identifikátor položky č. 1
  - 000 => určení typu položky (varint)
- 0000 1010
  - 0 => tzv. pokračovací bit. 0 znamená, že je konec
  - 000 1010 => samotná hodnota 12

12 08 4B C5 AF 72 6F 76 65 63

- 0001 0010
  - 00010 => identifikátor položky č. 2
  - 010 => určení typu položky (délkou určený typ)
- 0000 1000
  - 0 => tzv. pokračovací bit. 0 znamená, že je konec
  - 000 1000 => délka dat této položky je 8 bytů
- 4b c5 af 72 6f 76 65 63 => UTF-8 řetězec „Kůrovec“ (ů = c5 af)

18 DA 0F

- 0001 1000
  - 00011 => identifikátor položky č. 3
  - 000 => určení typu položky (varint)
- 1101 1010
  - 1 => tzv. pokračovací bit. 1 znamená, že další byte obsahuje data
  - 101 1010 => dolní bity hodnoty !!!
- 0000 1111
  - 0 => tzv. pokračovací bit. 0 znamená, že je konec
  - 000 1111 => pokračování bitů hodnoty
- 00 0111 1101 1010 => hodnota 2010

Definování čísel položek

- Číslo položky identifikuje danou položku v binárních datech
- Mezi verzemi se nesmí změnit čísla položek
- Client nebo Server ignoruje položky, které nezná
- Požadavky na prostor
  - 1 až 15 => 1 byte
  - 16 až 2047 => 2 byte
  - Nad 2047 by šlo pokračovat, ale postrádá to reálný smysl

Pojmenování prvků message

- V .proto souboru používat snake_case
- Vygenerovaný class bude mít propertu PascalCase

Čas

- DateTime a DateTimeOffset => google.protobuf.Timestamp
- TimeSpan => google.protobuf.Duration

- Google typy podporují potřebné konverze na .NET typy a z .NET typů

```
syntax = "proto3"

import "google/protobuf/duration.proto";
import "google/protobuf/timestamp.proto";

message Meeting {

    string subject = 1;
    google.protobuf.Timestamp time = 2;
    google.protobuf.Duration duration = 3;

}
```

GUID

- Nemá podporu v protobuf
- Používejte string jako náhradu
- NEPOUŽÍVEJTE jako náhradu byte[] – může nastat problém s reprezentací dat při různém vyhodnocení pořadí bytů (od začátku nebo od konce – endianness)

Nullable value types

- Build-in hodnotové typy nepodporují null hodnotu ani v protobuf
- Pro všechny základní typy existuje protobuf varianta typu podporujícího null hodnotu

- Název je XxxxxValue

```
syntax = "proto3"

import "google/protobuf/wrappers.proto"

message Person {

    ...
    google.protobuf.Int32Value age = 5;
}
```

Decimal

- Decimal nemá v protobuf reprezentaci
- V případě nutnosti lze vytvořit „Custom type“
  - Nutno dobře popsat v .proto souboru
- Je to jeden z požadavků do další verze

Repeated field

- List hodnot reprezentuje v protobuf „repeated“
- V C# kódu to je kolekce Google.Protobuf.Collections.RepeatedField<T>
- Tento typ podporuje IList<T> a IEnumerable<T>
- Není problém s použitím v LINQ operacích

```
message Person {
    // Other fields elided
    repeated string aliases = 8;
}
```

Repeated field vs. Stream

- Repeat preferovat
  - Data jsou komplet v „rozumném čase“ (do 1 sec – nebrat jako dogma!)
  - Množství dat a rychlost získání dat na serveru je predikovatelné
- Stream preferovat
  - Data přicházejí postupně a jejich kompletace trvá vteřiny
  - Množství dat nelze předem určit

Reserved

- Číselné označení zde zablokovat pro budoucí verze
- Vhodné i při zrušení položky, aby nedošlo k použití s jiným typem

```
syntax "proto3";

message Info {

  reserved 2, 9 to 11, 15;
  // ...
}
```

Any

```
syntax "proto3"

import "google/protobuf/any.proto"

message Stock {
    // Stock-specific data
}

message Currency {
    // Currency-specific data
}

message ChangeNotification {
    int32 id = 1;
    google.protobuf.Any instrument = 2;
}
```

```
public void FormatChangeNotification(ChangeNotification change)
{
    if (change.Instrument.Is(Stock.Descriptor))
    {
        FormatStock(change.Instrument.Unpack<Stock>());
    }
    else if (change.Instrument.Is(Currency.Descriptor))
    {
        FormatCurrency(change.Instrument.Unpack<Currency>());
    }
    else
    {
        throw new ArgumentException("Unknown instrument type");
    }
}
```

OneOf

```
message Stock {
    // Stock-specific data
}

message Currency {
    // Currency-specific data
}

message ChangeNotification {
  int32 id = 1;
  oneof instrument {
    Stock stock = 2;
    Currency currency = 3;
  }
}
```

```
public void FormatChangeNotification(ChangeNotification change)
{
    switch (change.InstrumentCase)
    {
        case ChangeNotification.InstrumentOneofCase.None:
            return;
        case ChangeNotification.InstrumentOneofCase.Stock:
            FormatStock(change.Stock);
            break;
        case ChangeNotification.InstrumentOneofCase.Currency:
            FormatCurrency(change.Currency);
            break;
        default:
            throw new ArgumentException("Unknown instrument type");
    }
}
```

Enum

- Protobuf nepodporuje [Flags]

```
enum AccountStatus {
  ACCOUNT_STATUS_UNKNOWN = 0;
  ACCOUNT_STATUS_PENDING = 1;
  ACCOUNT_STATUS_ACTIVE = 2;
  ACCOUNT_STATUS_SUSPENDED = 3;
  ACCOUNT_STATUS_CLOSED = 4;
}
```

```
public enum AccountStatus
{
    Unknown = 0,
    Pending = 1,
    Active = 2,
    Suspended = 3,
    Closed = 4
}
```

Dictionary<K,V>

- Protobuf má odpovídající typ map<K,V>
- V C# classech se použije typ Google.Protobuf.Collections.MapField<K,V>
- MapField<K,V> implementuje IDictionary<K,V>

```
message StockPrices {
    map<string, double> prices = 1;
}
```

Lab2

- Vytvořte gRPC službu pro předání dat o aktuálním počasí včetně výhledu na další 3 hodiny (řešte pomocí Dictionary<hodina, predpoved>)
- Předpověď i aktuální stav budou obsahovat (teplota, tlak, rychlost a směr větru a stav)
- Pro stav použijte enumerátor s hodnotami
  - Slunečno, Zamračené, Mlha, Déšť, Sněžení
- Směr větru je dán číslem (0 – 359) nebo výčtem (N, NE, E, SE, S, SW, W, NW)
- Data o počasí můžou obsahovat libovolné množství varování
- Otestujte funkčnost

Blok 3 – gRPC nastavení

- .NET Core a DI
- Základní nastavení a registrace
- Konfigurace serveru
- Konfigurace klienta

ASP.NET Core a DI

- Dependency Injection
  - Maximum automatizace
  - Potřebné služby na základě použití
  - Preferovat použití interface
- Některé služby ASP.NET Core podporuje „by design“
  - Logování (jen Debug a Console, file nebo DB vyžaduje cizí komponentu)
  - Konfigurace v JSON souborech

Co je služba?

- Cokoliv, co se může hodit pro vykonání zadání
- Většinou implementuje interface
- Životnost (určujeme při registraci)
  - Singleton
  - Scoped
  - Transient
- !!!! Služba registrovaná s nějakou životností nemůže být závislá na jiné službě s kratší životností

Registrace služby

- Třída Startup, metoda ConfigureServices
- Nutno dodržet korektní pořadí podle vazeb

```
public class Startup
{
    // This method gets called by the runtime. Use this method t
    // For more information on how to configure your application
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddGrpc();
        services.AddScoped<IMathService, MathService>();

    }
```

Podporované režimy Injection

- Konstruktor - preferovat
- HttpContext.RequestServices.GetService<T>()

Nastavení parametrů - server

- Parametr metody AddGrpc()

Parameter	Default
MaxSendMessageSize	null
MaxReceiveMessageSize	4 MB
EnableDetailedErrors	false
Interceptors	None
IgnoreUnknownServices	false

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc(options =>
    {
        options.EnableDetailedErrors = true;
        options.MaxReceiveMessageSize = 2 * 1024 * 1024; // 2 MB
        options.MaxSendMessageSize = 5 * 1024 * 1024; // 5 MB
    });
}
```

Parametr pro gRPC službu

- Globální nastavení lze změnit pro vybranou službu
- Lze upravit pomocí AddServiceOptions<T>

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddGrpc().AddServiceOptions<MyService>(options =>
    {
        options.MaxReceiveMessageSize = 2 * 1024 * 1024; // 2 MB
        options.MaxSendMessageSize = 5 * 1024 * 1024; // 5 MB
    });
}
```

Konfigurace klienta

- Konfiguruje se při vytvoření „channel“

Parameter	Default
MaxSendMessageSize	null
MaxReceiveMessageSize	4 MB
Credentials	null
MaxRetryAttempts	5
MaxRetryBufferSize	16 MB
MaxretryBufferPerCallSize	1 MB
HttpHandler / HttpClient	
DisposeHttpClient	false

Load Balancing

- Klient podporuje „Load Balancing“
  - Staticky si vybírá z připravených adres
  - Dynamicky pomocí DNS protokolu. Pamatuje si všechny odpovědi a při prvním volání použije první adresu, při dalších voláních vybírá z vrácených adres
- Na straně serveru LB řeší použitá technologie mimo gRPC.
- Doporučuje se preferovat proxy L7 (application) než L4 (transport)
- Projekty třetích stran
  - Envoy – open source proxy
  - Linkerd – Service mesh pro Kubernates
  - YARP - .NET reverse proxy s podporou gRPC a HTTP/2

Logování

- Defaultní MS .NET Core logger neumí log do souboru či DB
  - Lze obejít přesměrováním u Console App
- Možno užít jen nastavením
  - Console
  - Debug
  - EventSource
  - EventLog – Windows only
- Vyžadují instalaci patřičného nuGet balíčku
  - AzureAppServicesFile
  - AzureAppServicesBlob
  - ApplicationInsights

Logerry třetích stran

- Log4Net
  - Ověřený dlouholetý systém
  - Extrémní možnosti konfigurace logování
  - Není úplně triviální pro prvotní použití
  - Existují varianty pro jiné platformy
- Serilog
  - Menší, jednodušší systém
  - .NET only projekt

Lab 3 - konfigurace

- Vytvořte pomocnou třídu generující náhodná čísla, která ale bude používat společný logovací systém
- Použijte ji v naší aplikaci pro plnění hodnot počasí
- Nastavte maximální velikost správ a ověřte chování při překročení limitu
- Logujte informace o činnosti gRPC služby do souboru

gRPC stream

- Princip
- Stream od serveru
- Stream z klienta
- Obousměrný stream

Princip streamu

- HTTP/2 umožňuje udržet spojení
- Na takto vytvořeném TCP/IP může probíhat dlouhodobá komunikace
- Vzhledem k vytvořenému a udržovanému spojení by neměl být problém s neveřejnou IP klienta
- Nutno počítat s vysokou pravděpodobností síťových problémů při „dlouhém“ držení spojení s nízkým tokem dat

Stream od serveru - server

- V protobuf je u služby uvedeno: returns (stream Xxxxx)
- Metoda služby je asynchronní, vrací Task a má parametr typu IServerStreamWriter<T>, kterým se realizuje stream
- Metodou WriteAsync(T data) se posílají data na klienta
- Parametr ServerCallContext umožňuje sledovat, zda klient neukončil odběr

Stream od serveru - klient

- Vytvořit standardně gRPC kanál
- Vytvořit klienta používajícího konál
- Zavolat patřičnou serverovou metodu, ta vrátí třídu obsahující vlastnost ResponseStream
- Možnost příjmu dat:
  - await foreach(var item in xxx.ResponseStream.ReadAllAsync())
  - while (await xxx.ResponseStream.MoveNext())
- CancellationTokenSource lze použít pro oznámení nezájmu o další data

Stream od klienta - klient

- V protobuf je uvedeno stream u parametru metody
- Vytvořit klasicky kanál a klienta
- Zavolat metodu serveru, ale bez jakýchkoliv dat. Získáme objekt s vlastností RequestStream
- Pomocí await xxx.RequestStream.WriteAsync(T data) odesíláme data na server
- Na konci ukončíme voláním xxx.RequestStream.CompleteAsync()
- Návratovou hodnotu od serveru vrátí await xxx

Stream od klienta - server

- Metoda očekávající stream má parametr IAsyncStreamReader<T>
- Opět příjem pomocí await foreach(…. xxx.ReadAllAsync())
- Nezapomenout na možnost chyby

Obousměrná stream komunikace

- Klient i server posílají občas data
- Tento scénář vždy dobře zvážit
- Korektně ošetřit konec komunikace u obou streamů

Lab 4

- Vytvořte novou gRPC službu, která
- Bude mít metodu, která od klienta dostane sérii čísel a vrátí jejich součet
- Další metoda dostane kód letiště a bude každých 10 vteřin posílat aktuální údaje o počasí

- EXTRA:
- Pokud máte čas, přidejte metodu, která vytvoří echo server (čeká na stringy od klientů a beze změny je pošle klientovi zpět)

gRPC Interceptors

- Princip
- Metody třídy Interceptor
- Realizace Interceptoru

Princip

- Interceptor může vykonat kód při každém volání/zpracování požadavku
- Může být použit na obou koncích komunikace
- Slouží ke garanci standardizování chování gRPC při přenosu
  - Generální error handler
  - Custom HTTP metadata

- Používat opatrně, myslete na možnost, že druhá strana nemusí být .NET aplikace

Metody třídy Interceptor

Metoda	Použití
BlockingUnaryCall	Block block call
AsyncUnaryCall	Block asynchronous calls
AsyncServerStreamingCall	Block asynchronous service end stream calls
AsyncClientStreamingCall	Block asynchronous client stream calls
AsyncDuplexStreamingCall	Block asynchronous two-way flow calls
UnaryServerHandler	Used to intercept and pass in normal call server-side handlers
ClientStreamingServerHandler	Server side handler for intercepting client stream calls
ServerStreamingServerHandler	Server side handler for intercepting service side stream calls
DuplexStreamingServerHandler	Server side handler for intercepting two-way flow calls

Interceptor

- Vytvořit třídu odvozenou z Interceptor
- Override metody pro Server/Klienta a typ volání
  - Vybrat správnou verzi metody
  - Zajistit korektní pokračování
- Registrace Interceptoru v nastavení služby/klienta

Lab 5

- Vytvořte gRPC službu a klienta s podporou stream volání
- Vytvořte Interceptor pro logování na serveru a klientovi
- Zaregistrujte a otestujte
- Příklad řešení viz Examples na github.com pro grpc-dotnet

Řešení chyb

- Princip
- Chybové kódy serveru
- Zpracování chyby na klientovi
- Timeout pro odpověď
- Cancel u stream operací
- gRPCurl a gRPCui

Princip

- gRPC má silnou vazbu na HTTP/2 a jeho princip podpory chybových stavů
- Informace jsou velice limitované, Google vyvinul lepší variantu, zatím není přímo podporována v .NETu (grpc-dotnet)

Status code	Problem
GRPC_STATUS_UNIMPLEMENTED	Metoda neexistuje.
GRPC_STATUS_UNAVAILABLE	Samotný service je nedostupný.
GRPC_STATUS_UNKNOWN	Invalid response.
GRPC_STATUS_INTERNAL	Encoding/decoding problém.
GRPC_STATUS_UNAUTHENTICATED	Chyba při autentizaci
GRPC_STATUS_PERMISSION_DENIED	Chyba při autorizaci
GRPC_STATUS_CANCELLED	Volání zrušeno (většinou volajícím)

RpcException

- Třída reprezentující výjimku při gRPC komunikaci
- Podporuje přenos libovolných metadat v HTTP režimu (key : value)

```
public async Task<GetPortfolioResponse> GetPortfolio(GetPortfolioRequest request, ServerCallContext cont
{
    var user = context.GetHttpContext().User;
    if (!ValidateUser(user))
    {
        var metadata = new Metadata
        {
            { "User", user.Identity.Name }
        };
        throw new RpcException(new Status(StatusCode.PermissionDenied, "Permission denied"), metadata);
    }
}
```

Zpracování na klientovi

- RpcException nabízí
  - Status - enum
  - StatusCode – číselný kód
- Metada jsou k dispozici v Trailers

```
try
{
    var portfolio = await client.GetPortfolioAsync(new GetPortfolioRequest { Id = id });
}
catch (RpcException ex) when (ex.StatusCode == StatusCode.PermissionDenied)
{
    var userEntry = ex.Trailers.FirstOrDefault(e => e.Key == "User");
    Console.WriteLine($"User '{userEntry.Value}' does not have permission to view this portfolio.");
}
catch (RpcException)
{
    // Handle any other error type ...
}
```

Deadline

- Klient může definovat Deadline pro gRPC volání
- Definuje se v UTC!!!
- Neexistuje defaultní deadline => deadline je potřeba definovat
- Deadline hlídá klient i server
- Může nastat situace, že server připraví odpověď, ale klient ji nestihne přijmout

```
try
{
    var response = await client.SayHelloAsync(
        new HelloRequest { Name = "World" },
        deadline: DateTime.UtcNow.AddSeconds(5));

    // Greeting: Hello World
    Console.WriteLine("Greeting: " + response.Message);
}
catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded)
{
    Console.WriteLine("Greeting timeout.");
}
```

Propagace deadline

- Manuálně předáním deadline z contextu

```
public override async Task<UserResponse> GetUser(UserRequest request,
    ServerCallContext context)
{
    var client = new User.UserServiceClient(_channel);
    var response = await client.GetUserAsync(
        new UserRequest { Id = request.Id },
        deadline: context.Deadline);

    return response;
}
```

- Automatizovaně

```
services
    .AddGrpcClient<User.UserServiceClient>(o =>
    {
        o.Address = new Uri("https://localhost:5001");
    })
    .EnableCallContextPropagation();
```

Cancel – už nic nechci

- Klient může dát serveru vědět, že nemá zájem o pokračování
- Dvě cesty jak na to:
  - Dispose() objektu reprezentujícího klienta komunikace
  - Použití CancellationToken jako parametru pro volání

- Server musí s CancellationToken pracovat
- Pokud server dělá sám volání dalších služeb, předává CT dál
  - Lze použít automatické předání Contextu jak u Deadline

Retry – pokus o záchranu

- Dokáže řešit
  - Chvilkou ztrátu síťové konektivity
  - Dočasně nedostupnou službu
  - Nestíhající server (deadline)
- gRPC má automatické řešení chyby pomocí opakování požadavku

```
var defaultMethodConfig = new MethodConfig
{
    Names = { MethodName.Default },
    RetryPolicy = new RetryPolicy
    {
        MaxAttempts = 5,
        InitialBackoff = TimeSpan.FromSeconds(1),
        MaxBackoff = TimeSpan.FromSeconds(5),
        BackoffMultiplier = 1.5,
        RetryableStatusCodes = { StatusCode.Unavailable }
    }
};

var channel = GrpcChannel.ForAddress("https://localhost:5001", new GrpcChannelOptions
{
    ServiceConfig = new ServiceConfig { MethodConfigs = { defaultMethodConfig } }
});
```

Retry - podmínky

- Retry se provede, pokud:
  - Status chyby odpovídá RetryableStatusCodes
  - Nebyl dosažen MaxAttempts pokusů
  - Call nebyl commited!
    - Přijetí hlavičky od serveru se považuje za Commit
    - Překročení velikosti RetryBufferů taky znamená commit

Retry a stream

- Server stream
  - Po přijetí první zprávy klienta už nelze Retry použít
  - Aplikace musí mít vlastní logiku jak obnovit přerušené spojení
- Klient stream
  - Limitováno velikostí bufferu na klientovi

Retry backoff delay

- Retry se provede v náhodném čase mezi 0 a aktuální hodnotou času
- InitialBackoff => Povinné nastavení, musí být kladné. Určuje max. čas pro první Retry
- BackoffMultiplayer => Musí být nadefinováno. Hodnotou se násobí čas pro Retry při dalším pokusu. Musí být kladné.
- MaxBackoff => Povinné nastavení, určuje maximum pro Backoff, kam lze dojít postupně násobením
- MaxAttempts => Defalt je 5 podle nastavení GrpcChannelOptions. Hodnota je vyžadována a musí být větší než 1

Hedging

- gRPC odešle z klienta několik stejných zpráv a pak čeká jen na první výsledek, který se vrátí
- Vše po prvním přijetí odpovědi se ignoruje
- Server se musí sám postarat o to, že ta vícenásobnost nebude vadit.
- Nastavuje se pomocí HedgingPolicy
- Nelze kombinovat s Retry

Cloud a Retry/Hedging

- Cloud řešení může mít problém, protože
  - Je tzv. Shared a zrovna nestíhá
  - Přesuny instancí v cloudu mohou způsobit krátkodobý výpadek
  - Přetížená síť (zejména omezené linky pro upload)

- Doporučení
  - Parametry mít konfigurovatelné
  - Idempotency Patterns
  - Berte do úvahy čas na Retry a čas na běžné vykonání požadavku

gRPCurl a gRPCui

- Nástroje pro testování gRPC služby
- Ve službě je potřeba zapnout podporu pro gRPC reflection
- gRPCurl => cmd line tester
- gRPCui => spustí web aplikaci s UI pro testování dotazů

Lab 6

- Vytvořte službu, která bude pracovat s CancellationToken
- Ověřte že zafunguje předání požadavku na Cancel
- Otestujte Retry režim klienta. Na serveru náhodně generujte RpcException se statusem Unavailable.

Zabezpečení

- Filozofie zabezpečení
- Zabezpečení přenosu dat
- Autentizace
- Autorizace

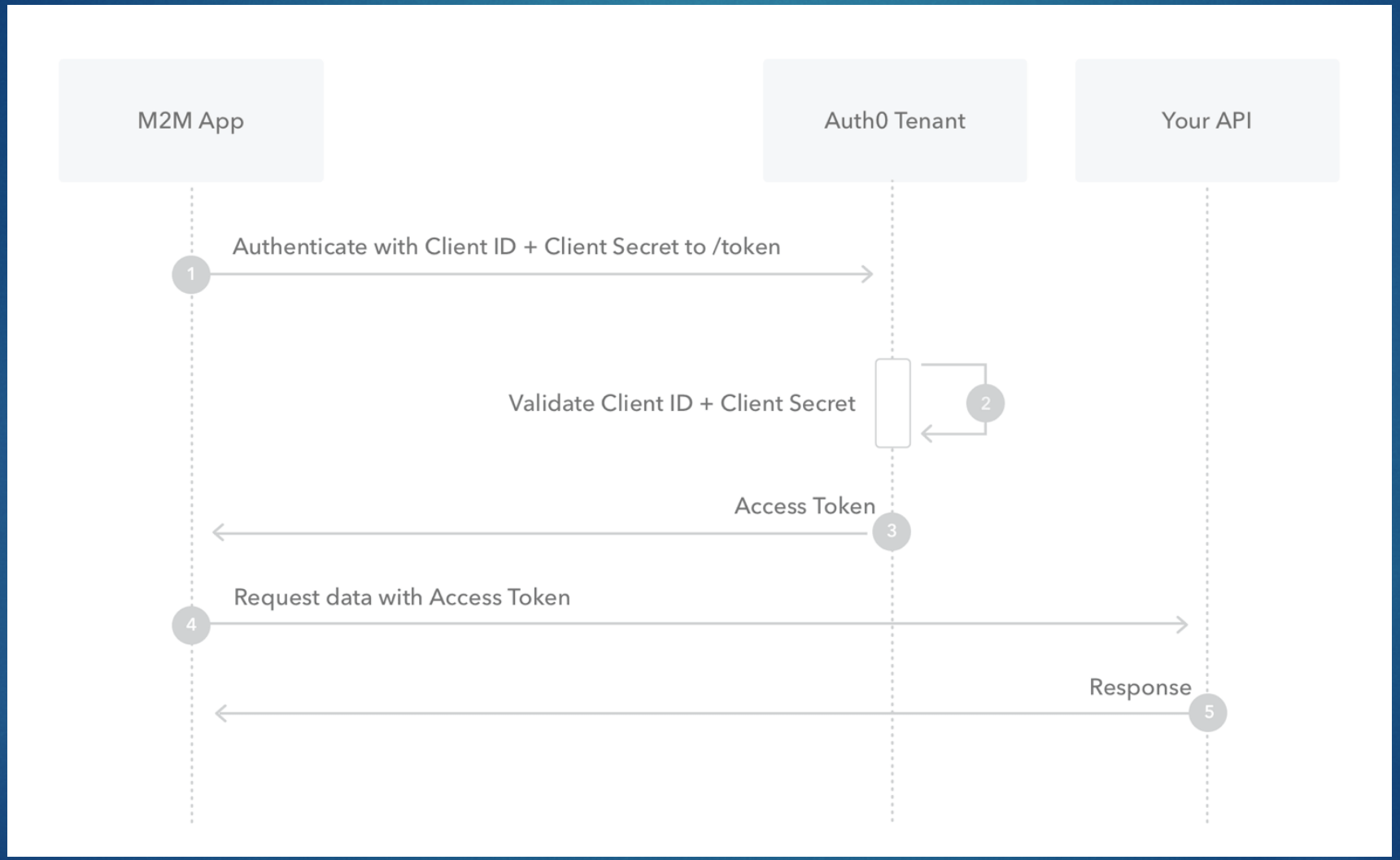
Filozofie zabezpečení

- gRPC neřeší klasicky Autentizaci (jméno/heslo)
- Počítá s autorizačním mechanizmem využívajícím token vydaný důvěryhodnou službou
- JWT Bearer = JSON Web Token Bearer =>Oauth 2.0 Bearer Access Token
- Klient přidává token k volání
- Není podporovaná žádná forma Windows Autentizace (NTLM/Kerberos/Negotiation)

Podporované autentizace

- Azure Active Directory
- Client Certificate
- IdentityServer
- JWT Token
- OAuth 2.0
- OpenID Connect
- WS-Federation

Token autentizace



Autentizace

- Připravený token předáme v hlavičce požadavku

```
public bool DoAuthenticatedCall(
    Ticketer.TicketerClient client, string token)
{
    var headers = new Metadata();
    headers.Add("Authorization", $"Bearer {token}");

    var request = new BuyTicketsRequest { Count = 1 };
    var response = await client.BuyTicketsAsync(request, headers);

    return response.Success;
}
```

Authorized channel

- Hned při vytváření channel lze zadat informace pro autentizaci
- Pak není nutno zadávat token k volání metody
- Ideální pro situace, kde jsou všechna volání s požadavkem na zabezpečení
- Funguje výhradně na HTTPS spojení, vyžaduje TLS

Autentizace certifikátem

- Autentizaci řeší TLS
- Server musí akceptovat klientské certifikáty (nutná konfigurace)
- Certifikát se vkládá do HttpClientHandler

```
public Ticketer.TicketerClient CreateClientWithCert(
    string baseAddress,
    X509Certificate2 certificate)
{
    // Add client cert to the handler
    var handler = new HttpClientHandler();
    handler.ClientCertificates.Add(certificate);

    // Create the gRPC channel
    var channel = GrpcChannel.ForAddress(baseAddress, new GrpcChannelOptions
    {
        HttpHandler = handler
    });

    return new Ticketer.TicketerClient(channel);
}
```

Autorizace

- Attribut [Authorize] u metody serveru
  - Pro vyvolání metody muselo dojít k Autentizaci, neřeší se nic dál
- Attribute může určovat požadovanou Policy [Authorize(“Partner“)]
  - Policy se definuje v AddAuthorization() jako option
  - String definuje název Policy
  - Podmínka používá data která klient nabízí (tzv. claims)

Lab 7

- Vytvořte autentizační službu, která bude vracet token
- Použijte ho pro volání a otestujte

Závěrečná doporučení

- gRPC je designováno na MALÉ zprávy
- Channel je drahá záležitost, používat s rozvahou
- Client object je lehký obal, lze rychle a levně vytvářet, tudíž není důvod ho držet
- Na jednom kanálu může najednou komunikovat několik klientů
- Jeden klient může najednou řešit několik volání
- HTTP/2 má konečný počet konkurenčních streamů
- V případě dlouhých mezer v komunikaci použijte KeepAlivePing

Sledujte

- Web gRPC projektu
- GitHub projekt gRPC a grpc-dotnet
- Stránky Googlu o novinkách v gRPC

Závěrečné hodnocení

E-PREZENCE

Děkuji za pozornost

tomas@havetta.cz