Script:

Slide 3:

gRPC là gì:

Là open source remote procedure call (thủ tục gọi từ xa) framework tạo bởi google 2016, là framework mới – hiện đại để xây dựng API có tính mở rộng (scalable), hiện đại và nhanh chóng.

Nó cho phép các lập trình viên viết microservices bằng nhiều ngôn ngữ mà vẫn giữ được khả năng dễ dàng tạo giao tiếp giữa các service với nhau.

gRPC nó phụ thuộc vào Protocol buffers cho cơ chế truyền tải

Slide 4:

gRPC là implemt phổ biến của RPC

Đã có nhiều công ty lớn đã sử dụng gRPC làm cơ chế RPC ưa thích để kết nối nhiều microservice như Google hay Netflix chẳn hạn.

Vậy RPC là gì, thì nó là 1 function call ngay trong tiến trình để thực thi đoạn code nào đó. Hiểu đơn giản thì RPC cho phép 1 máy này gọi đoạn code trên một máy khác như 1 local function call.

Slide 5:

Theo mặc định thì gRPC sử dụng Protocol buffers để mã hóa và gửi data đi

gRPC còn có thể hỗ trợ các loại mã hóa khác như JSON

Vậy nếu gRPC có thể hỗ trợ các loại mã hóa khác thì sao lại chọn Protocol Buffers Chắc hẳn Protocol Buffers phải cung cấp một số lợi thế để trở thành định dạng mã hóa được lựa chọn cho gRPC. Vậy thì ta phải hiểu Protocol Buffers là gì?

Slide 6:  
Protocol buffer hay còn gọi là Protobuf là một giao thức được Google phát triển cho phép **serialization và deserialization dữ liệu có cấu trúc giữa các services**

Mục tiêu ra đời của Protocol buffers là tạo ra một phương pháp tốt hơn XML để các hệ thống có thể giao tiếp với nhau qua mạng hoặc để lưu trữ dữ liệu.

Nó còn hỗ trợ nhiều ngôn ngữ lập trình như Python, java, Objective C, C# và một số khác.

Slide 7:

Nếu ta so sánh với JSON cũng là một loại data formats khác so với XML, được dùng chính để truyền data giữa server và application.

Về mặt CPU thì việc Parsing JSON thực sự tốn nhiều CPU hơn (vì định dạng JSON con người vẫn có thể đọc được)

Còn Parsing Protocol buffers ít sử dụng CPU hơn vì nó gần với cách biểu diễn dữ liệu của máy tính. Không giống như JSON text-based format, Protocol buffers là binary format

Việc sử dụng Protocol buffers cho gRPC giúp giao tiếp nhanh hơn và hiệu quả hơn, thân thiện với các thiết bị di động và có thể hoạt động tốt ngay cả trên các thiết bị có CPU chậm hơn.

Như trog hình có thể thấy với JSON có thể là 55 bytes nhưng vởi Prototcol Buffers chỉ là 20 bytes

Slide 8:

Cụ thể trong gRPC thì Protocol Buffers được dùng để define

Messages có data, Request và Resposne

Service có Service name và RPC endpoints

Lợi thế tiếp theo giúp cải thiện mặt performance gRPC là HTTP/2

Slide 9:

GRPC được xây dựng trên HTTP/2 để cung cấp hiệu suất cao ở quy mô lớn.

Multiplexing: HTTP/2 hỗ trợ ghép kênh, cho phép đồng thời nhiều luồng dữ liệu được gửi qua một kết nối duy nhất. (Điều này làm giảm độ trễ và cải thiện hiệu quả bằng cách loại bỏ sự cần thiết phải thiết lập nhiều kết nối cho các yêu cầu/phản hồi song song.)

Stream prioritization (Ưu tiên luồng): HTTP/2 cho phép các luồng được gán các ưu tiên, nghĩa là server có thể ưu tiên và phân bổ resource cho các yêu cầu quan trọng hơn trước. (Điều này giúp cải thiện hiệu suất tổng thể và đảm bảo rằng các yêu cầu quan trọng được xử lý kịp thời.)

Binary protocol: HTTP/2 sử dụng Binary protocol, nhỏ gọn và hiệu quả hơn các text-based protocols được sử dụng trong các phiên bản HTTP trước đó. (Điều này làm giảm lượng dữ liệu cần được truyền qua mạng, dẫn đến giao tiếp nhanh hơn và giảm mức tiêu thụ băng thông.)

Server push: cho phép server push response chủ động vào trog caches client để dùng cho tương lai. Server có thể send resource như fetching HTML, linked stylesheet hoặc script ngay lập tức khi có update.

GRPC sử dụng HTTP/2 streams, cho phép nhiều luồng messages qua một kết nối TCP. Điều này cho phép GRPC Framework xử lý nhiều lần gọi RPC cùng 1 lúc thông qua chỉ vài kết nối TCP giữa client và server.

(Nghĩa là thay vì cứ mở đóng connection cho mỗi request/response thì gRPC chỉ dùng 1 kết nối để gửi và nhận nhiều messages. Lợi ích của việc dùng single long-lived TCP giúp giảm chi phí cho việc mở đóng nhiều kết nối, dẫn đến giao tiếp nhanh hơn và hiệu quả hơn. Thêm nữa còn giúp giảm tắc nghẽn mạng và cãi thiện network performance.)

Slide 10:

Vậy thì gRPC hoạt động như nào?

Đây là flow từ gRPC client đến gRPC server

Giả sử đang có Order Serice là client còn Payment Service là Server

Khi Order Service thực thi gọi gRPC đến Payment Service, nó sẽ gọi client code được generate bởi gRPC tooling khi build

Đám client code generated được gọi là client stub. gRPC mã hóa data được truyền cho client stub thành Protocol buffers và gửi cho transport layer. Sau đó gRPC sẽ gửi data qua network dưới dạng stream của HTTP/2. Vì được mã hóa dưới dạng binary và có network optimization nên gRPC nhanh hơn JSON gấp 5 lần.

Payement service sau khi nhận được packets từ network, thì sẽ tiến hành decode và gọi server application. Kết quả trả về từ server sẽ được mã hóa tiếp thành Protocol Buffers và gửi xuống transport layer.

Bên Order Service nhận được packets gửi từ payment service, sẽ tiến hành decode và gửi kết ảu đó lên client application.

**Slide 11 DEMO**

Một cấu trúc project của gRPC gồm có

***greet.proto***: file Protos/greet.proto để định nghĩa các service và message được sử dụng trong hệ thống.

***Services*** folder: chứa các class được sử dụng để xử lý request và trả về response cho các gRPC service được định nghĩa trong file proto.

Trong gRPC project, file proto được sử dụng để định nghĩa các service và message được sử dụng trong hệ thống.

Trong file proto, bạn có thể định nghĩa các RPC service, bao gồm các phương thức được sử dụng để truyền tải dữ liệu giữa các client và server. Các message cũng có thể được định nghĩa để đại diện cho các kiểu dữ liệu khác nhau được sử dụng trong giao tiếp.

File proto được sử dụng để tạo ra các client và server stubs. Bằng cách định nghĩa các service và message trong file proto, bạn có thể sử dụng các công cụ của gRPC để tạo ra mã code cho client và server stubs.

// Định nghĩa service EmployeeCRUD với các phương thức SelectAll, SelectByID, Insert, Update, Delete

// Nhận các message là Empty và TblUserFilter và trả về các message là TblUsers và Empty

service EmployeeCRUD {

rpc SelectAll (Empty) returns (TblUsers);

rpc SelectByID (TblUserFilter) returns (TblUser);

rpc Insert (TblUser) returns (Empty);

rpc Update (TblUser) returns (Empty);

rpc Delete (TblUserFilter) returns (Empty);

}

// Định nghĩa message Empty không có thuộc tính

message Empty {

}

// Định nghĩa message TblUser với các thuộc tính UserId, FullName, Password, RoleId

message TblUser {

string UserId = 1;

string FullName = 2;

string Password = 3;

string RoleId = 4;

}

// Định nghĩa message TblUsers là một danh sách các TblUser

message TblUsers {

repeated TblUser items = 1;

}

// Định nghĩa message TblUserFilter với thuộc tính UserId

message TblUserFilter {

string UserId = 1;

}

Ở Startup.cs ta phải thêm AddGrpc() để có thể sử dụng gRPC service

app.UseEndpoints(endpoints =>

{

endpoints.MapGrpcService<EmployeeCRUDService>();

endpoints.MapGet("/", async context =>

{

await context.Response.WriteAsync("Communication with gRPC endpoints must be made through a gRPC client. To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");

});

});

Cấu hình Endpoint cho app, map với EmployeeCRUDService vào endpoint để response cho các gRPC request.

(Nó còn map root endpoint đến hàm trả message chỉ ra endpoint của gRPC chỉ có thể truy cập thông qua client. Đoạn code này cho phép ứng dụng nhận các yêu cầu gRPC và phản hồi với gRPC service tương ứng.)

EmployeeCRUD.EmployeeCRUDBase class này là cho bên server implement

Còn bên client sẽ là EmployeeCRUD.EmployeeCRUDClient

Thư mục "Services" chứa các lớp được sử dụng để xử lý yêu cầu và trả về kết quả cho các dịch vụ gRPC được định nghĩa trong tệp proto.

Mỗi serice được định nghĩa trong tệp proto tương ứng với một class trong thư mục "Services". Class này triển khai các phương thức được định nghĩa trong tệp proto để xử lý các yêu cầu gRPC và trả về kết quả tương ứng. Ví dụ, nếu trong tệp proto có định nghĩa một dịch vụ "Greeter" với phương thức "SayHello", thì trong thư mục "Services" sẽ có một lớp được đặt tên là "GreeterService" với phương thức "SayHello" được triển khai.

Tiếp đó là qua Client

…..

Sau khi demo .NET 5 xong

Mặc dù gRPC có performance rất mạnh nhưng Rest với JSON đã tồn tại và đã quá quen với developers khi con người có thể dễ đọc JSON và debug hơn. Nhưng giờ đây với .NET 7 vừa có tính năng mới là gRPC Json transcoding một extension mới cho asp.net core thì hi vọng nó có thể làm cho gRPC phổ biến hơn.

gRPC JSON transcoding giúp tạo RESTful JSON APIs cho gRPC services. Transcoding cho phép apps có thể gọi gRPC services với HTTP concepts:

* HTTP verbs
* URL parameter binding
* JSON requests/responses

Với cách tiếp cận này thì gRPC service vẫn có thể tiếp tục hoạt động như bình thường nhưng đồng thời bây giờ nó còn có thể đóng vai trò như một HTTP API.

Để có thể hoạt động thì mình có thêm vào nugget Transcoding, sau khi thêm thì ta phải có chỉnh sửa ở proto file. Trong đây chúng ta phải nói cho nó service biết endpoint, thì ở đây là google api http. Để có cái này thì phải lấy Proto files Google API này để có annotations và các thứ liên quan. Và cũng đừng quên phải import Google API mới thêm.

<https://github.com/dotnet/aspnetcore/tree/main/src/Grpc/JsonTranscoding/test/testassets/Sandbox/google/api>

Sau đó phải khai báo cho service container là ta muốn dùng Json Transcoding và thế là bây giờ mình đã có được cả performance của grpc và độ thân thiện với con người của JSON

Ngoài ra ta còn có thể thêm OpenAPI hỗ trợ cho gRPC Json Transocded API của chúng ta với package.

 Microsoft.AspNetCore.Grpc.Swagger.

Đoạn code này được sử dụng trong ASP.NET Core để cấu hình Swagger để tạo ra API doc cho một dịch vụ gRPC. Swagger là một công cụ để tạo ra tài liệu API tự động và dễ dàng để khách hàng và nhà phát triển tương tác với dịch vụ của mình.

Cụ thể, trong đoạn code này:

AddGrpcSwagger() được sử dụng để đăng ký Swagger cho dịch vụ gRPC.

AddSwaggerGen() được sử dụng để cấu hình SwaggerGen để tạo ra tài liệu Swagger. Trong đó c.SwaggerDoc() được sử dụng để định nghĩa phiên bản Swagger và thông tin về dịch vụ.

UseSwagger() được sử dụng để kích hoạt Swagger middleware để cung cấp các tài nguyên Swagger.

UseSwaggerUI() được sử dụng để cấu hình giao diện người dùng SwaggerUI, trong đó c.SwaggerEndpoint() được sử dụng để đăng ký tài liệu Swagger.

Cả gRPC web và Json Transcoding cho phép ta gọi gRPC service từ browser nhưng cách thực hiện nó khác nhau vs gRPC web thì phải cần generate grpc client nên việc cài open API cho grpc sẽ giúp tiện lợi hơn, đây là tính năng rất hữu ích khi bạn muốn mở rộng dịch vụ gRPC hiện tại của mình để hoạt động giống như một API HTTP.

Slide 12:

Và đó là đã xong phần demo cho gRPC, thì với gRPC, nó có thể đem lại nhiều lợi ích hơn và đây là những lợi ích nó đem lại khi sử dụng.

Tiếp nối sẽ là phần Microservice của bạn Hiển