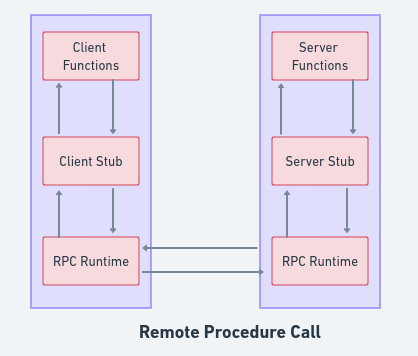
**Introduction to gRPC:**

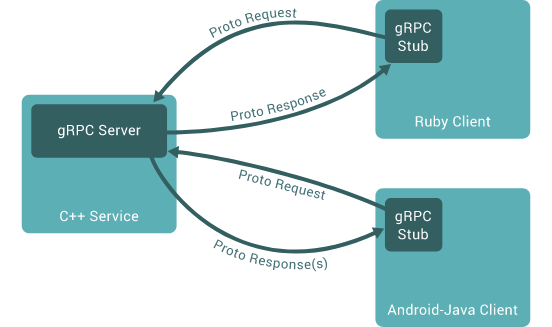
**What is RPC?**

RPC **(Remote Procedure Call)** enables two distinct computer programs to communicate with each other over a network. In this communication model, one program requests a service or procedure from another program, which processes the request and sends the results back to the first program. RPC follows the client-server model, where the requesting program acts as the client and the service-providing program operates as the server.

The architecture of RPC.

**gRPC:**

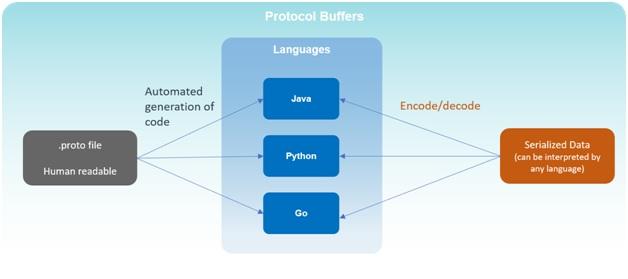
gRPC is a cross-platform open source remote procedure call (RPC) framework that enables client and server applications to communicate with each other remotely and transparently. It was developed by google which uses HTTP/2. gRPC utilizes Remote Procedure Call (RPC) technology to facilitate communication between disparate services using a protocol buffer.The protocol buffer serves as a conduit for structured data exchange between systems.



The rough architecture of gRPC. It's more or less the same as regular RPC.

**Protocol Buffer:**

gRPC uses protocol buffers as both its Interface Definition Language (IDL) and as its underlying message interchange format. Protocol buffers, aka protobufs, are a language-agnostic protocol for serializing structured data and they can be used between different programming languages. They are strongly typed, which ensures that the data is consistent and well-formed across different applications and services.As this is in the form of a contract, both the client and server need to have the same proto file.



There are four types of gRPC communication.They are:

* **Unary RPC**
* **Server Streaming RPC**
* **Client Streaming RPC**
* **Bidirectional Streaming RPC**

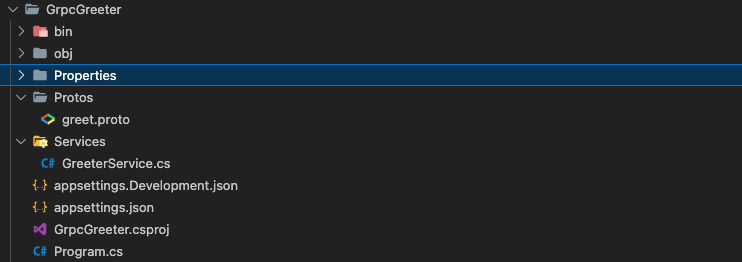
**How to use it with asp.net core?**

**Unary Rpc:**

nary RPCs where the client sends a single request to the server and gets a single response back, just like a normal function call.

To create the gRPC project:

**dotnet new grpc -o GrpcGreeter** //It will create new Asp.net grpc project



After running the command we will get this kind of folder structure and files.

After creating the project add a proto file in the protos. And declare the methods that will be used in the project. Here is the sample code of proto file.

**Sample.proto**

syntax = "proto3";

option csharp\_namespace = "GrpcService";

package sample;

service sample {

rpc GetFullName(sampleRequest) returns (sampleResponse);

}

message sampleRequest{

string firstName =1;

string lastName =2;

}

message sampleResponse{

string fullName =1;

}

After writing the proto file you need to include the file in .csproj file.

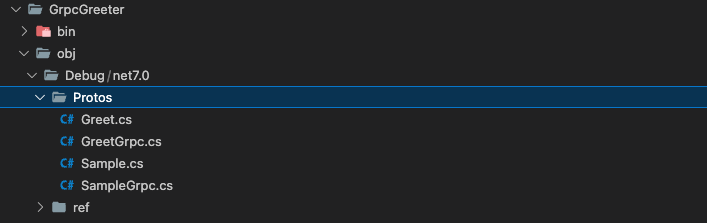
<ItemGroup>

<Protobuf Include="Protos\greet.proto" GrpcServices="Server" />

<Protobuf Include="Protos\Sample.proto" GrpcServices="Server" /

</ItemGroup>

After completing the step make sure to build the file so that the compiler can generate the file.



These files will be generated afterward.

Here In the service folder we need to create a new service.

**SampleService.cs**

using Grpc.Core;

namespace GrpcService.Services;

public class SampleService : sample.sampleBase

{

public override Task<sampleResponse> GetFullName(sampleRequest request, ServerCallContext context)

{

var result = $"{request.FirstName} {request.LastName}";

return Task.FromResult(new sampleResponse { FullName = result });

}

}

Here we have defined the method GetFullName() which we have declared in the proto file.

We need to add the the service in the Program.cs

app.MapGrpcService<SampleService>();

Here the server part ended now lets create a console client

**Client for gRPC Server in ASP.NET core:**

Lets create a console client:

**dotnet new console -o SampleClient**

Install **nuget** packages for client:

**Grpc.Tools**

**Google.Protobuf**

**Grpc.Net.Client**

After creating the project and adding the nuget packages add a new folder name “protos” and add a proto file in it. And declare the methods that will be used in the project. Here is the sample code of the proto file.

**Sample.proto**

syntax = "proto3";

option csharp\_namespace = "GrpcService";

package sample;

service sample {

rpc GetFullName(sampleRequest) returns (sampleResponse);

}

message sampleRequest{

string firstName =1;

string lastName =2;

}

message sampleResponse{

string fullName =1;

}

After writing the proto file you need to include the file in the **.csproj** file.

<ItemGroup>

<Protobuf Include="Protos\sample.proto" GrpcServices="Client" />

</ItemGroup>

Here the configuration end and method declared in the proto file and defined in the Server service can be used

Here is the code of the client:

using System.Diagnostics;

using Google.Protobuf.WellKnownTypes;

using Grpc.Net.Client;

using GrpcService;

namespace GrpcClient;

public class Program

{

public static async Task Main(string[] args)

{

var channel = GrpcChannel.ForAddress("http://localhost:5013");

var Client = new sample.sampleClient(channel);

// var response = Client.GetFullName(new sampleRequest { FirstName = "Hello", LastName = "World" });

var response = await Client.GetFullNameAsync(new sampleRequest { FirstName = "Hello", LastName = "World" });

Console.WriteLine(response.FullName);

var dateTime = DateTime.SpecifyKind(new DateTime(2023, 09, 14), DateTimeKind.Utc);

await channel.ShutdownAsync();

}

}

Here the address of the server should be paste instead of

**var channel=GrpcChannel.ForAddress(“**[**http://localhost:5013**](http://localhost:5013)**”);**

Replace the address from your server address.

**Server Streaming RPC:**

The client sends a single request to the server and gets back several messages that it reads sequentially.

Here the changes will only occur in the proto, service and client file except that all the process remains the same.

Proto code for server streaming for both client and server:

syntax="proto3";

option csharp\_namespace="gRPCServer";

package streamingDemo;

service streamingDemo{

rpc ServerStreamingDemo(Test) returns (stream Test);

}

message Test{

string TestMessage=1;

}

**Server Code:**

Then create a service and add the code:

using Grpc.Core;

namespace gRPCServer.StreamDemoService;

class StreamDemoService : streamingDemo.streamingDemoBase

{

public override async Task ServerStreamingDemo(Test request, IServerStreamWriter<Test> responseStream, ServerCallContext context)

{

for (int i = 0; i < 20; i++)

{

await responseStream.WriteAsync(new Test { TestMessage = $"Message {i}" });

}

}

}

And add the service in the **program.cs** of the server side and the proto file in the .**csproj** file.

After creating and adding the code to the proto file and including it .csproj file. We can add a client for server streaming.

**Client code:**

private static async Task ServerStreamingDemo()

{

var channel = GrpcChannel.ForAddress("http://localhost:5013");

var client = new streamingDemo.streamingDemoClient(channel);

var responseStream = client.ServerStreamingDemo(new Test { TestMessage = "Hello World" });

while (await responseStream.ResponseStream.MoveNext(CancellationToken.None))

{

var value = responseStream.ResponseStream.Current.TestMessage;

Console.WriteLine($"{value}");

}

Console.WriteLine("Server streaming completed successfully");

await channel.ShutdownAsync();

}

**Client Streaming RPC:**

The client sends multiple messages to the server, and the server can process them as they arrive before sending a single response back to the client.

Proto file for both client and server:

syntax="proto3";

option csharp\_namespace="gRPCServer";

package streamingDemo;

service streamingDemo{

rpc ClientStreamingDemo(stream Test) returns ( Test);

}

message Test{

string TestMessage=1;

}

**Server code:**

We need to create the service and then add it to the program.cs and the code for the service is:

using Grpc.Core;

namespace gRPCServer.StreamDemoService;

class StreamDemoService : streamingDemo.streamingDemoBase

{

public override async Task<Test> ClientStreamingDemo(IAsyncStreamReader<Test> requestStream, ServerCallContext context)

{

while (await requestStream.MoveNext())

{

Console.WriteLine(requestStream.Current.TestMessage);

}

Console.WriteLine("Client streaming Completed");

return new Test { TestMessage = "Client streaming" };

}

}

And add the service in the **program.cs** of the server side and the proto file in the .**csproj** file.

After creating and adding the code to the proto file and including it .csproj file. We can add a client for server streaming.

**Client Code:**

private static async Task ClientStreamingDemo()

{

var channel = GrpcChannel.ForAddress("http://localhost:5013");

var client = new streamingDemo.streamingDemoClient(channel);

var response = client.ClientStreamingDemo();

await Task.Delay(1000);

for (int i = 1; i <= 10; i++)

{

await response.RequestStream.WriteAsync(new Test { TestMessage = $"Message {i}" });

Thread.Sleep(5000);

}

await response.RequestStream.CompleteAsync();

// await channel.ShutdownAsync();

Console.WriteLine("complete client stream");

}

**Bidirectional streaming RPC:**

Bidirectional streaming is a powerful feature of GRPC that enables the client and server to send and receive multiple streams of data simultaneously.

Proto file for both client and server:

syntax="proto3";

option csharp\_namespace="gRPCServer";

package streamingDemo;

service streamingDemo{

rpc ClientStreamingDemo(stream Test) returns ( Test);

}

message Test{

string TestMessage=1;

}

**Server code:**

We need to create the service and then add it to the program.cs and the code for the service is:

using Grpc.Core;

namespace gRPCServer.StreamDemoService;

class StreamDemoService : streamingDemo.streamingDemoBase

{

public override async Task BidirectionalStreamingDemo(IAsyncStreamReader<Test> requestStream, IServerStreamWriter<Test> responseStream, ServerCallContext context)

{

var tasks = new List<Task>();

while (await requestStream.MoveNext())

{

Console.WriteLine("Recieved request: " + requestStream.Current.TestMessage);

var task = Task.Run(async () =>

{

var message = requestStream.Current.TestMessage;

await responseStream.WriteAsync(new Test { TestMessage = message });

Console.WriteLine($"sent Response {message}");

});

tasks.Add(task);

}

await Task.WhenAll(tasks);

Console.WriteLine("Bidirectional streaming completed");

}

}

And add the service in the **program.cs** of the server side and the proto file in the .**csproj** file.

After creating and adding the code to the proto file and including it .csproj file. We can add a client for server streaming.

**Client Code:**

private static async Task BidirectionalStreamingDemo()

{

var channel = GrpcChannel.ForAddress("http://localhost:5013");

var client = new streamingDemo.streamingDemoClient(channel);

var stream = client.BidirectionalStreamingDemo();

await Task.Delay(1000);

var requestTask = Task.Run(async () =>

{

for (int i = 1; i <= 10; i++)

{

await stream.RequestStream.WriteAsync(new Test { TestMessage = $"hello world {i}" });

Console.WriteLine("Sent Request: " + i);

}

await stream.RequestStream.CompleteAsync();

});

await Task.Delay(1000);

Console.WriteLine("test");

var responseTask = Task.Run(async () =>

{

while (await stream.ResponseStream.MoveNext(CancellationToken.None))

{

Console.WriteLine("Received Response: " + stream.ResponseStream.Current.TestMessage);

}

Console.WriteLine("Response srteam completed successfully");

await channel.ShutdownAsync();

});

}

**References:**

[**https://learn.microsoft.com/en-us/aspnet/core/grpc/?view=aspnetcore-7.0**](https://learn.microsoft.com/en-us/aspnet/core/grpc/?view=aspnetcore-7.0)

[**https://grpc.io/docs/what-is-grpc/**](https://grpc.io/docs/what-is-grpc/)

[**https://www.freecodecamp.org/news/what-is-grpc-protocol-buffers-stream-architecture/**](https://www.freecodecamp.org/news/what-is-grpc-protocol-buffers-stream-architecture/)

[**https://youtube.com/playlist?list=PLzewa6pjbr3IOa6POjAMM0xiPZ-shjoem&feature=shared**](https://youtube.com/playlist?list=PLzewa6pjbr3IOa6POjAMM0xiPZ-shjoem&feature=shared)

[**https://stackoverflow.com/questions/52146721/how-are-protocol-buffers-faster-than-xml-and-json/52148242#52148242**](https://stackoverflow.com/questions/52146721/how-are-protocol-buffers-faster-than-xml-and-json/52148242#52148242)

[**https://medium.com/@kova98/faster-request-response-in-net-with-grpc-d0b7b4604cb9**](https://medium.com/@kova98/faster-request-response-in-net-with-grpc-d0b7b4604cb9)