Tutorial: Create a gRPC client and server in ASP.NET Core

By John Luo

Adresa din Internet:
https://docs.microsoft.com/ro-ro/aspnet/core/tutorials/grpc/grpc-start?view=aspnetcore-3.1&tabs=visual-studio

This tutorial shows how to create a .NET Core gRPC client and an ASP.NET Core gRPC Server.

At the end, you'll have a gRPC client that communicates with the gRPC Greeter service.

View or download sample code (how to download).

In this tutorial, you:

- Create a gRPC Server.
- Create a gRPC client.
- Test the gRPC client service with the gRPC Greeter service.

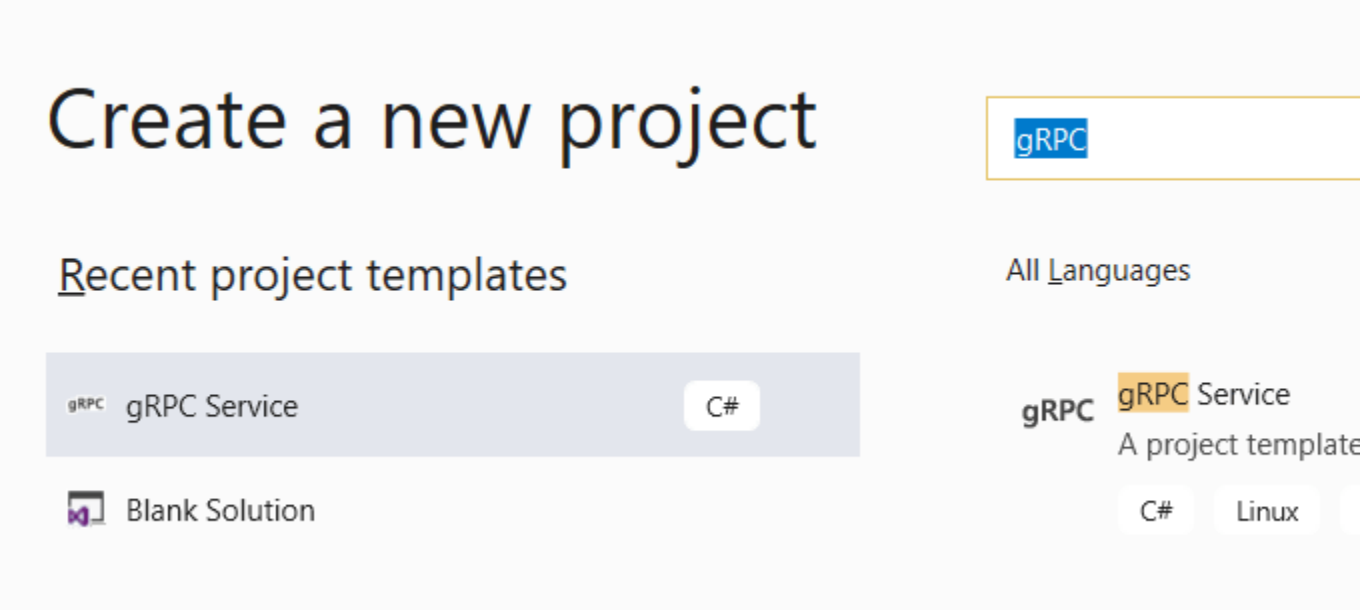
Prerequisites

- Visual Studio
- Visual Studio Code
- Visual Studio for Mac
  - Visual Studio 2019 with the **ASP.NET and web development** workload
  - .NET Core 3.0 SDK or later

Create a gRPC service

- Visual Studio
- Visual Studio Code
- Visual Studio for Mac

- Start Visual Studio and select **Create a new project**. Alternatively, from the Visual Studio **File** menu, select **New** > **Project**.
- In the **Create a new project** dialog, select **gRPC Service** and select **Next**:

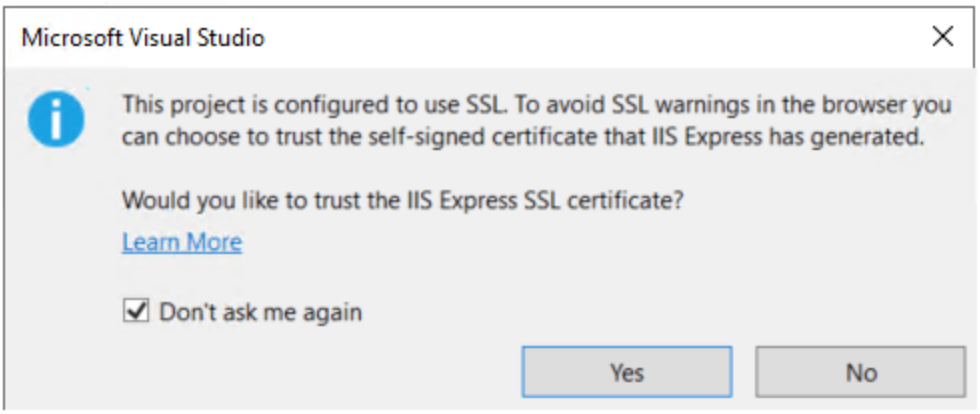


- Name the project **GrpcGreeter**. It's important to name the project *GrpcGreeter* so the namespaces will match when you copy and paste code.
- Select **Create**.
- In the **Create a new gRPC service** dialog:
  - The **gRPC Service** template is selected.
  - Select **Create**.

Run the service

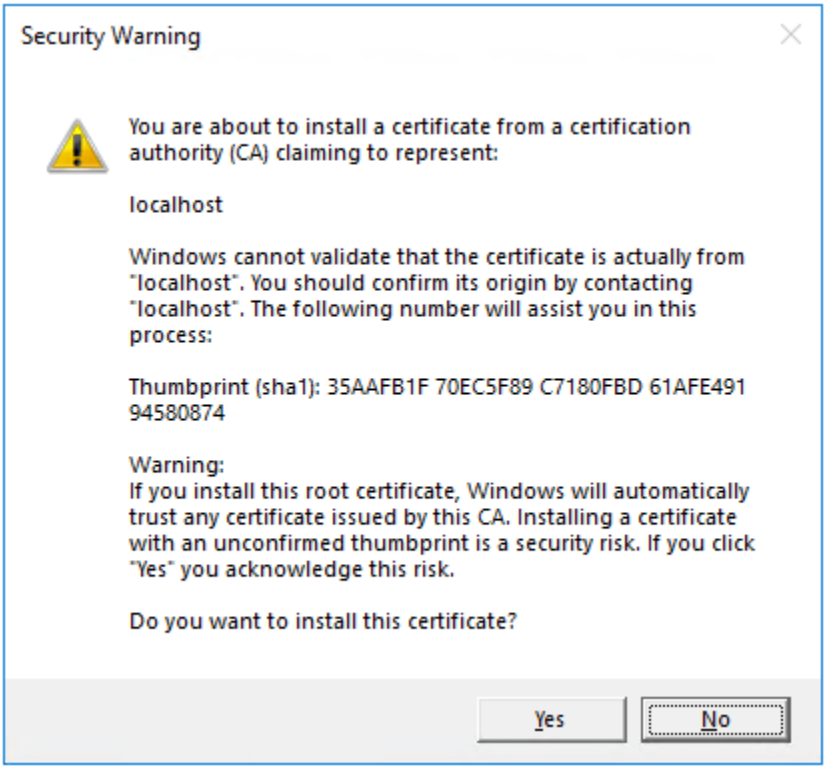
- Press Ctrl+F5 to run without the debugger.

  Visual Studio displays the following dialog:



Select **Yes** if you trust the IIS Express SSL certificate.

The following dialog is displayed:



Select **Yes** if you agree to trust the development certificate.

Visual Studio starts IIS Express and runs the app. The address bar shows `localhost:port#` and not something like `example.com`. That's because `localhost` is the standard hostname for the local computer. Localhost only serves web requests from the local computer. When Visual Studio creates a web project, a random port is used for the web server.

The logs show the service listening on `https://localhost:5001`.

consoleCopiere

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
```

Note

The gRPC template is configured to use **Transport Layer Security (TLS)**. gRPC clients need to use HTTPS to call the server.

macOS doesn't support ASP.NET Core gRPC with TLS. Additional configuration is required to successfully run gRPC services on macOS. For more information, see **Unable to start ASP.NET Core gRPC app on macOS**.

Examine the project files

GrpcGreeter project files:

- *greet.proto* – The *Protos/greet.proto* file defines the `Greeter` gRPC and is used to generate the gRPC server assets. For more information, see Introduction to gRPC.
- *Services* folder: Contains the implementation of the `Greeter` service.
- *appSettings.json* – Contains configuration data, such as protocol used by Kestrel. For more information, see Configuration in ASP.NET Core.
- *Program.cs* – Contains the entry point for the gRPC service. For more information, see .NET Generic Host.
- *Startup.cs* – Contains code that configures app behavior. For more information, see App startup.

Create the gRPC client in a .NET console app

- Open a second instance of Visual Studio and select **Create a new project**.
- In the **Create a new project** dialog, select **Console App (.NET Core)** and select **Next**.
- In the **Name** text box, enter **GrpcGreeterClient** and select **Create**.

Add required packages

The gRPC client project requires the following packages:

- Grpc.Net.Client, which contains the .NET Core client.
- Google.Protobuf, which contains protobuf message APIs for C#.
- Grpc.Tools, which contains C# tooling support for protobuf files. The tooling package isn't required at runtime, so the dependency is marked with `PrivateAssets="All"`.

Install the packages using either the Package Manager Console (PMC) or Manage NuGet Packages.

PMC option to install packages

- From Visual Studio, select **Tools** > **NuGet Package Manager** > **Package Manager Console**
- From the **Package Manager Console** window, run `cd GrpcGreeterClient` to change directories to the folder containing the *GrpcGreeterClient.csproj* files.
- Run the following commands:

  PowerShell

```
Install-Package Grpc.Net.Client
Install-Package Google.Protobuf
Install-Package Grpc.Tools
```

Manage NuGet Packages option to install packages

- Right-click the project in **Solution Explorer** > **Manage NuGet Packages**
- Select the **Browse** tab.
- Enter **Grpc.Net.Client** in the search box.
- Select the **Grpc.Net.Client** package from the **Browse** tab and select **Install**.
- Repeat for `Google.Protobuf` and `Grpc.Tools`.

Add greet.proto

- Create a *Protos* folder in the gRPC client project.
- Copy the *Protos\greet.proto* file from the gRPC Greeter service to the gRPC client project.
- Edit the *GrpcGreeterClient.csproj* project file:

  Right-click the project and select **Edit Project File**.

- Add an item group with a `<Protobuf>` element that refers to the *greet.proto* file:

```
<ItemGroup>
  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />
</ItemGroup>
```

Create the Greeter client

Build the project to create the types in the `GrpcGreeter` namespace.
The `GrpcGreeter` types are generated automatically by the build process.

Update the gRPC client *Program.cs* file with the following code:

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
using GrpcGreeter;
using Grpc.Net.Client;

namespace GrpcGreeterClient
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // The port number(5001) must match the port of the gRPC server.
            using var channel = GrpcChannel.ForAddress("https://localhost:5001");
            var client =  new Greeter.GreeterClient(channel);
            var reply = await client.SayHelloAsync(
                              new HelloRequest { Name = "GreeterClient" });
            Console.WriteLine("Greeting: " + reply.Message);
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Program.cs contains the entry point and logic for the gRPC client.

The Greeter client is created by:

- Instantiating a `GrpcChannel` containing the information for creating the connection to the gRPC service.
- Using the `GrpcChannel` to construct the Greeter client:

```
static async Task Main(string[] args)
{
    // The port number(5001) must match the port of the gRPC server.
    using var channel = GrpcChannel.ForAddress("https://localhost:5001");
    var client =  new Greeter.GreeterClient(channel);
    var reply = await client.SayHelloAsync(
                      new HelloRequest { Name = "GreeterClient" });
    Console.WriteLine("Greeting: " + reply.Message);
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

The Greeter client calls the asynchronous `SayHello` method. The result of the `SayHello` call is displayed:

```
static async Task Main(string[] args)
{
    // The port number(5001) must match the port of the gRPC server.
    using var channel = GrpcChannel.ForAddress("https://localhost:5001");
    var client =  new Greeter.GreeterClient(channel);
    var reply = await client.SayHelloAsync(
                      new HelloRequest { Name = "GreeterClient" });
    Console.WriteLine("Greeting: " + reply.Message);
    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}
```

Test the gRPC client with the gRPC Greeter service

- In the Greeter service, press `Ctrl+F5` to start the server without the debugger.
- In the `GrpcGreeterClient` project, press `Ctrl+F5` to start the client without the debugger.

The client sends a greeting to the service with a message containing its name, *GreeterClient*. The service sends the message "Hello GreeterClient" as a response. The "Hello GreeterClient" response is displayed in the command prompt:

```
Greeting: Hello GreeterClient
Press any key to exit...
```

The gRPC service records the details of the successful call in the logs written to the command prompt:

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\GH\aspnet\docs\4\Docs\aspnetcore\tutorials\grpc\grpc-
start\sample\GrpcGreeter
info: Microsoft.AspNetCore.Hosting.Diagnostics[1]
      Request starting HTTP/2 POST https://localhost:5001/Greet.Greeter/SayHello
application/grpc
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[0]
      Executing endpoint 'gRPC - /Greet.Greeter/SayHello'
info: Microsoft.AspNetCore.Routing.EndpointMiddleware[1]
      Executed endpoint 'gRPC - /Greet.Greeter/SayHello'
info: Microsoft.AspNetCore.Hosting.Diagnostics[2]
      Request finished in 78.32260000000001ms 200 application/grpc
```

Note

The code in this article requires the ASP.NET Core HTTPS development certificate to secure the gRPC service. If the client fails with the message `The remote certificate is invalid according to the validation procedure.`, the development certificate is not trusted. For instructions to fix this issue, see **Trust the ASP.NET Core HTTPS development certificate on Windows and macOS**.

gRPC not supported on Azure App Service

Avertisment

ASP.NET Core gRPC is not currently supported on Azure App Service or IIS. The HTTP/2 implementation of Http.Sys does not support HTTP response trailing headers which gRPC relies on. For more information, see **this GitHub issue**.

Next steps

- Introduction to gRPC on .NET Core
- gRPC services with C#
- Migrating gRPC services from C-core to ASP.NET Core