gRPC
is the new WCF

FUTURE TECH 2020

THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES

What’s gRPC

gRPC Stands for Remote Procedure Call

is a modern high-performance RPC framework

is based on HTTP/2, Protocol Buffers and other modern standard-based technologies

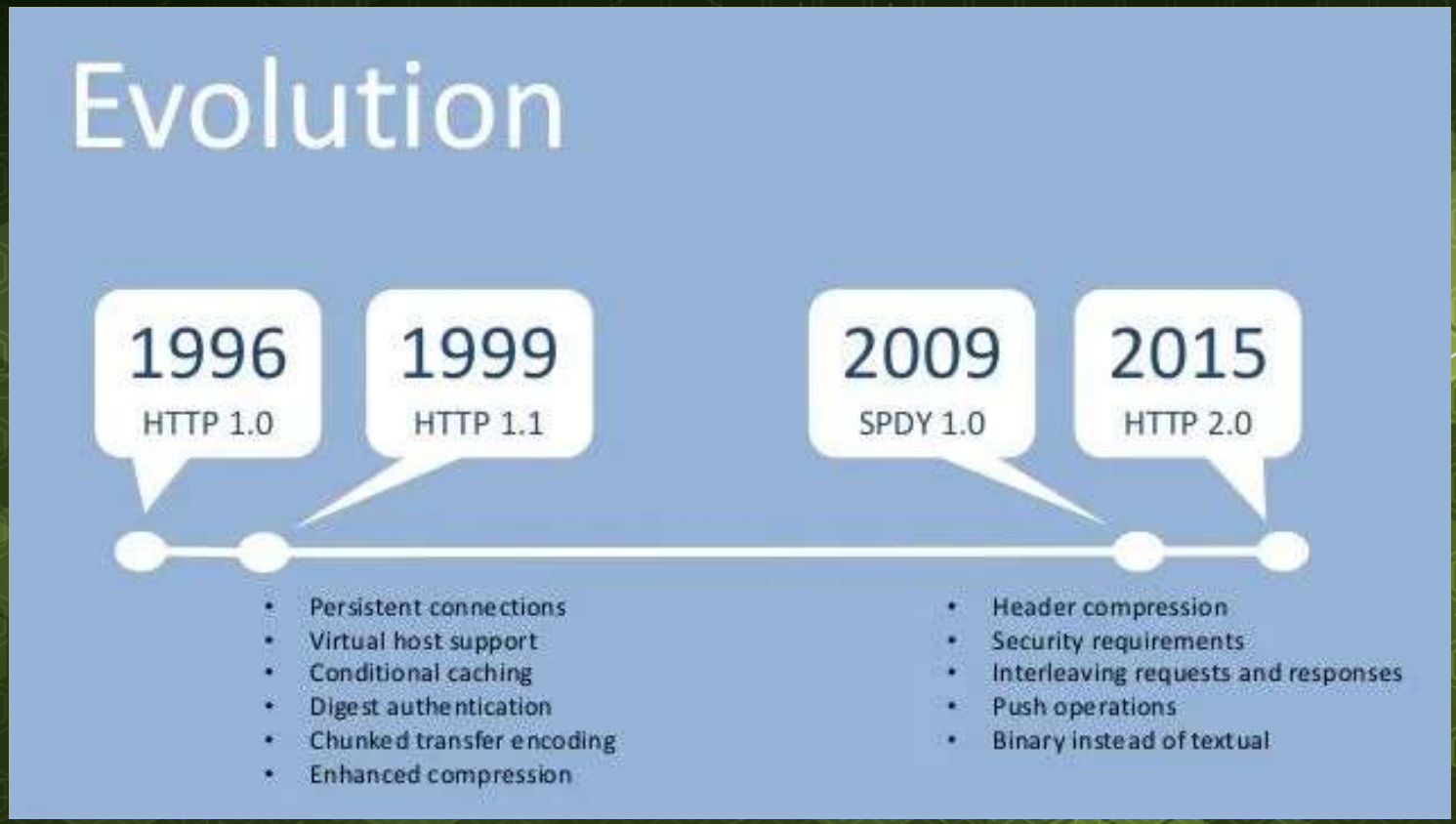
What's gRPC

gRPC is Payload agnostic: is support protobuf but other serialization formats like flatbuffers, json etc are easily configurable

The architecture is extensible, most of the feature are provided by Filters: Auth, tracing, name resolution, LB, message size checks, RPC deadlines

Works with many transports, by default uses HTTP2 (other implementaions: QUIC, Cronet, Inprocess and more)

History of HTTP



Source https://frontnet.eu/

HTTP 1.x: Limited parallelism

New TCP connection per HTTP connection

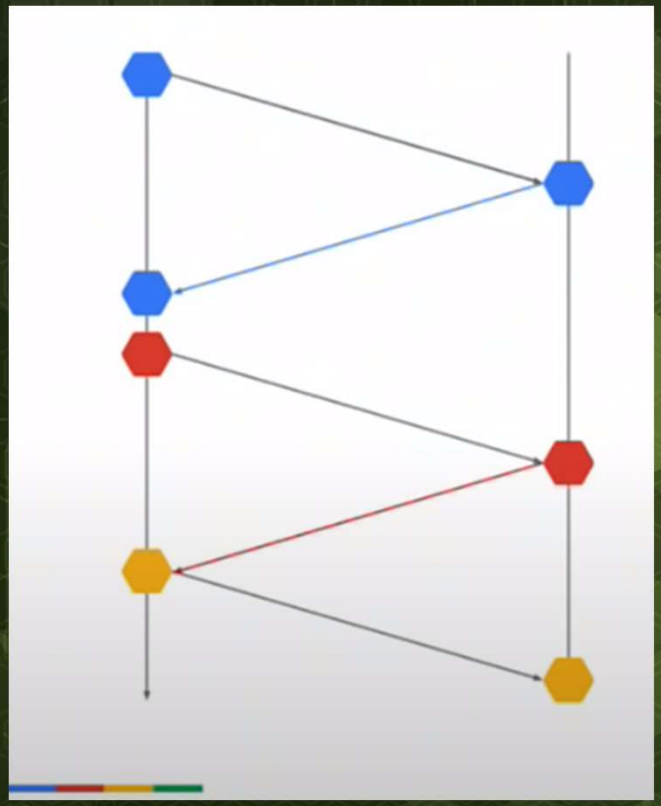
Number of parallel HTTP request
=
Number of TCP Connection

Initializing a connection is very expensive

Source https://www.dailyrecord.co.uk/

HTTP 1.0: Head of line blocking

Without pipelining

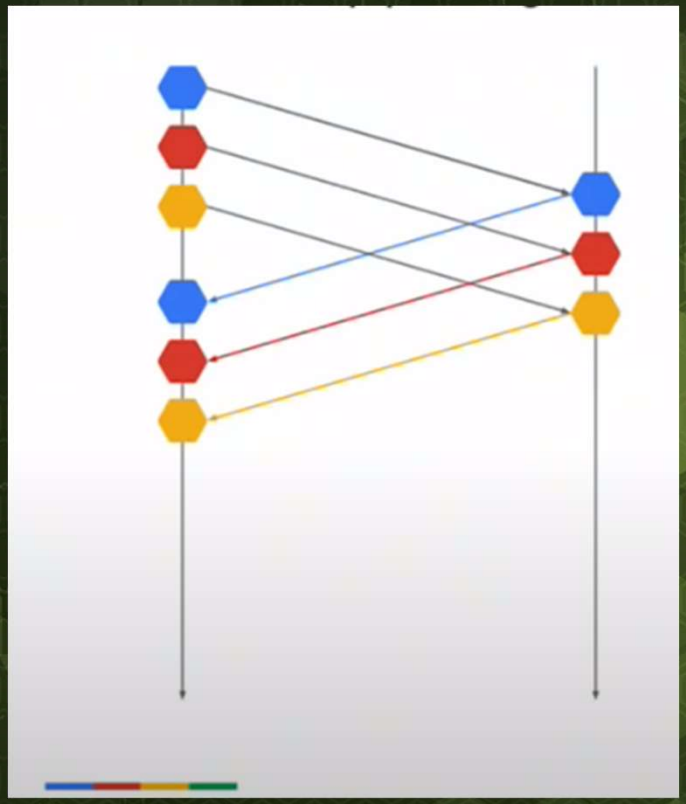


One request wait for the reply
Then a new request make a new reply

Source https://www.dailymail.co.uk/

HTTP 1.1: Head of line blocking

With pipelining

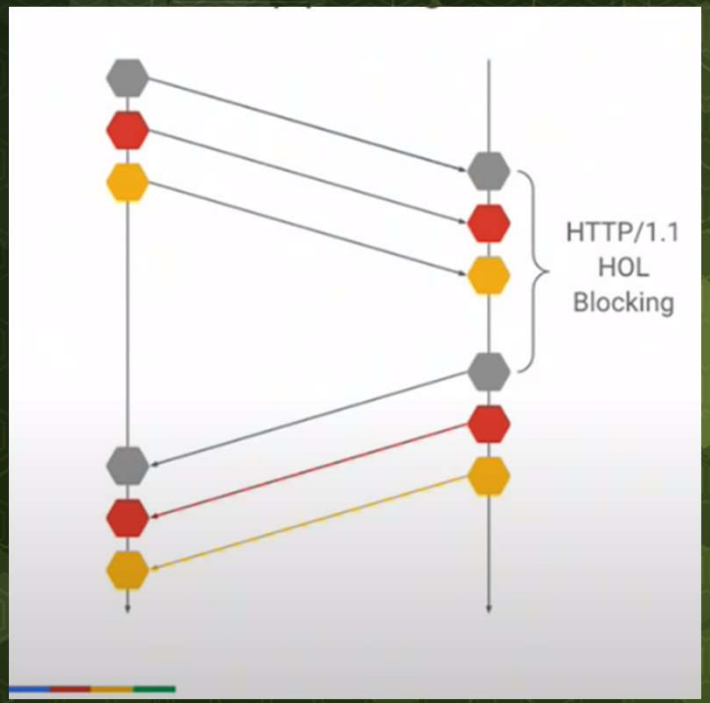


One request wait for the reply
Then a new request make a new reply

Source https://www.dailymail.co.uk/

HTTP 1.1: Head of line blocking

With pipelining



One request wait for the reply
Then a new request make a new reply

Source https://www.dailymail.co.uk/

HTTP 1.x: Protocol Overhead

- HTTP Headers
- Uncompressed plain text headers for each and every HTTP request

HTTP/2

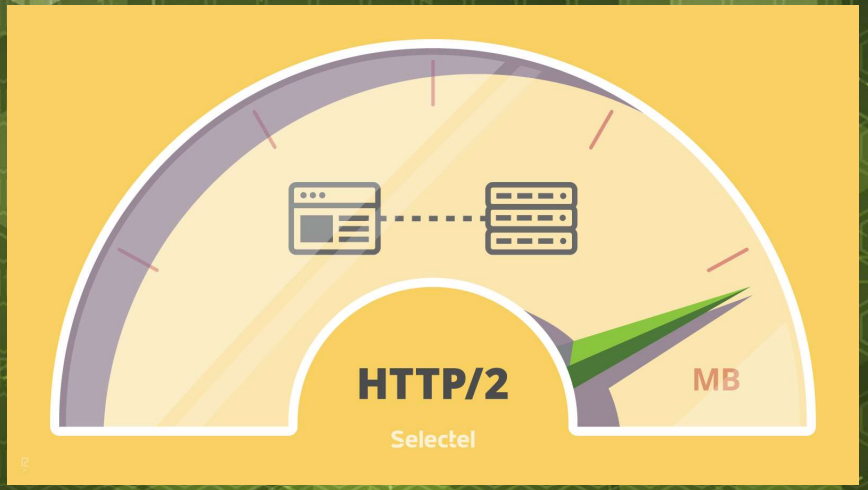
Released in 2015. Extend (not replace) the semantics of HTTP/1.1

Improve end-user perceived latency

Address the “head of line blocking”

Not require multiple connection

Minimize protocol overhead



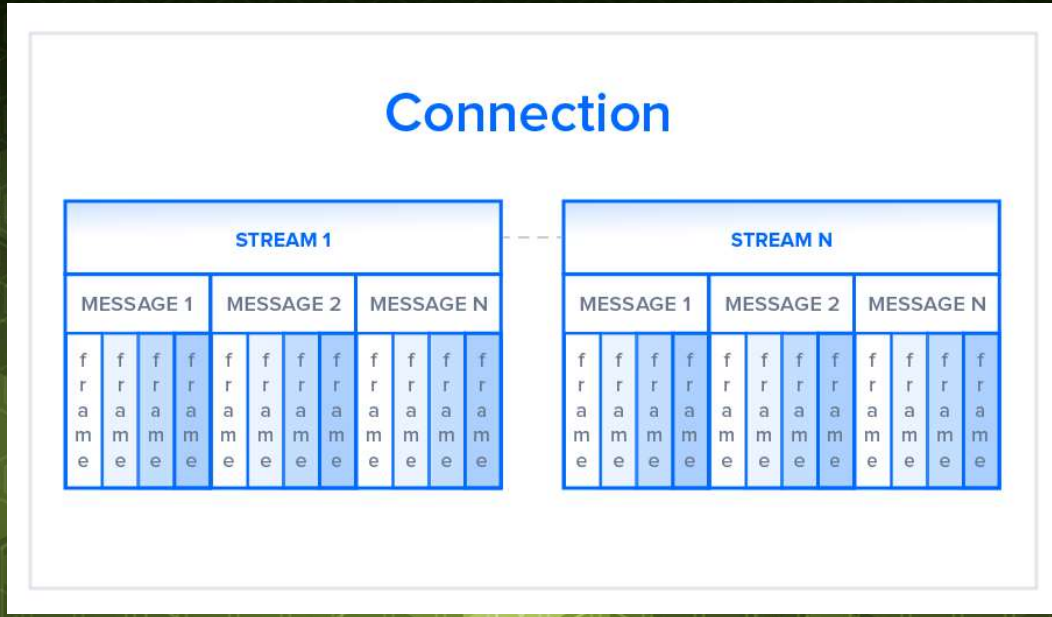
Source https://blog.selectel.com/

HTTP/2 Binary Framing

Tries to address all the previous version problems and extend it

Stream is a bidirectional flow of bytes within an established connection, which may carry one or more messages.

Message is a complete sequence of frames that map to a logical request or response message

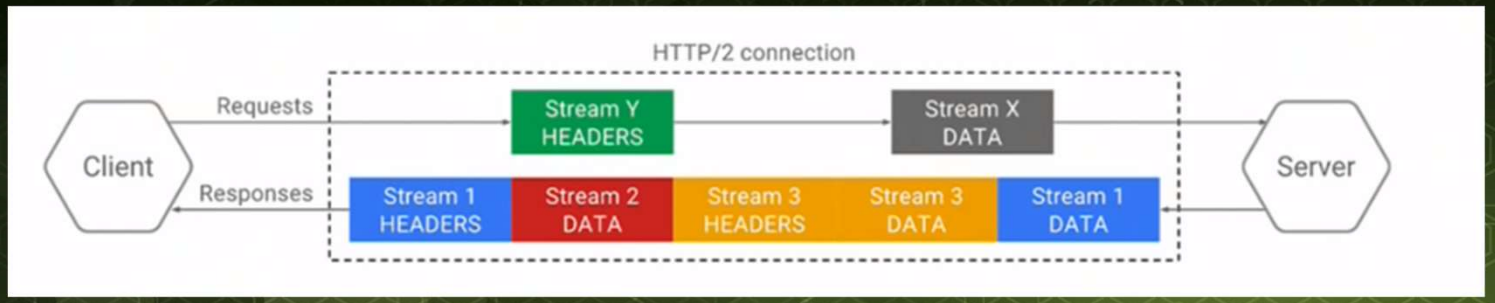


Frame is the smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs: HEADERS for metadata, DATA for payload, RST_STREAM SETTINGS, PUSH_PROMISE, PING, GOAWAY, WINDOW_UPDATE, etc

Source M.Atamel, R.Tsang @Google

FUTURE TECH 2020

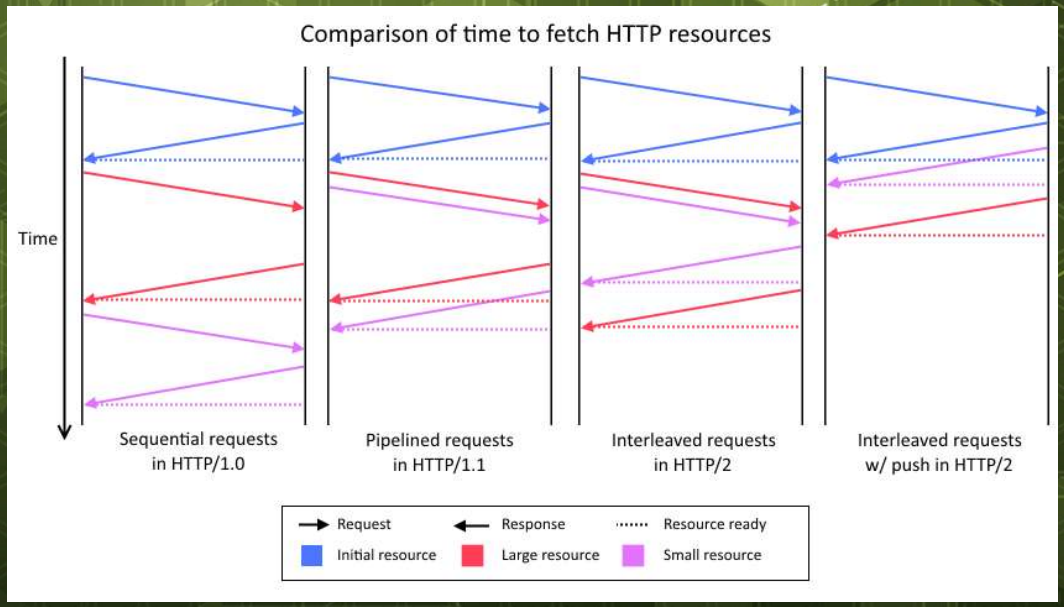
HTTP/2 Request/Response Multiplexing



Interleave multiple requests and responses in parallel without blocking on anyone

Use a **single TCP connection** to deliver multiple requests and responses in parallel

Enable flow-control, server push, etc



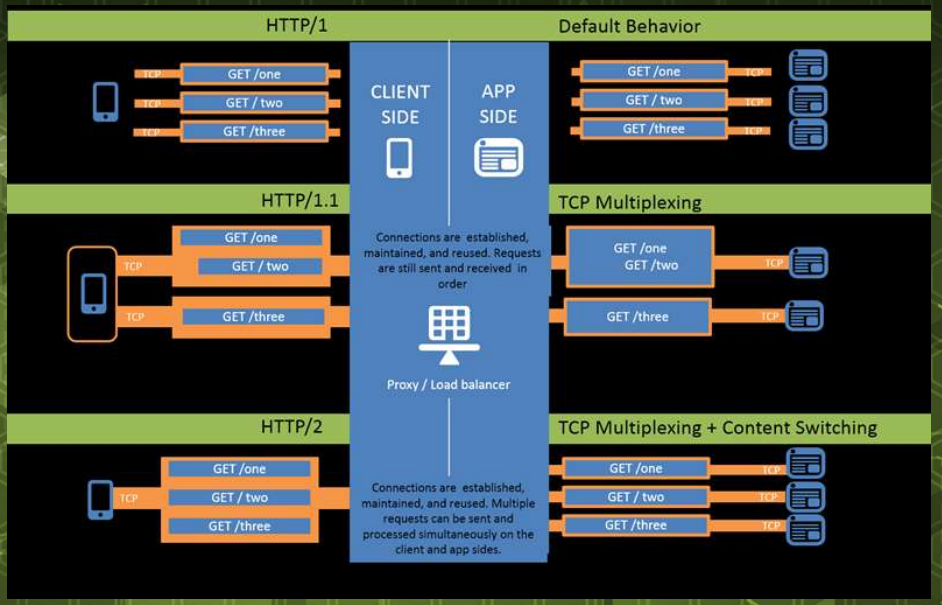
Source M.Atamel, R.Tsang @Google – Comparison of time to fetch HTTP: https://blog.scottlogic.com/

HTTP/2 Request/Response Multiplexing

Interleave multiple requests and responses in parallel without blocking on anyone

Use a **single TCP connection** to deliver multiple requests and responses in parallel

Enable flow-control, server push, etc

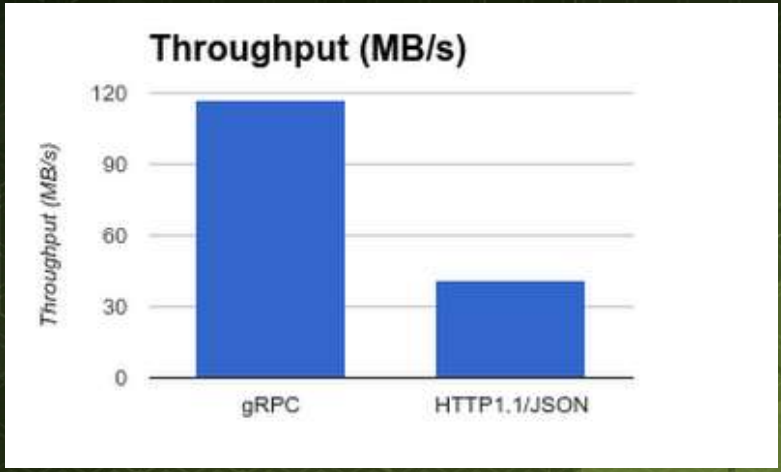


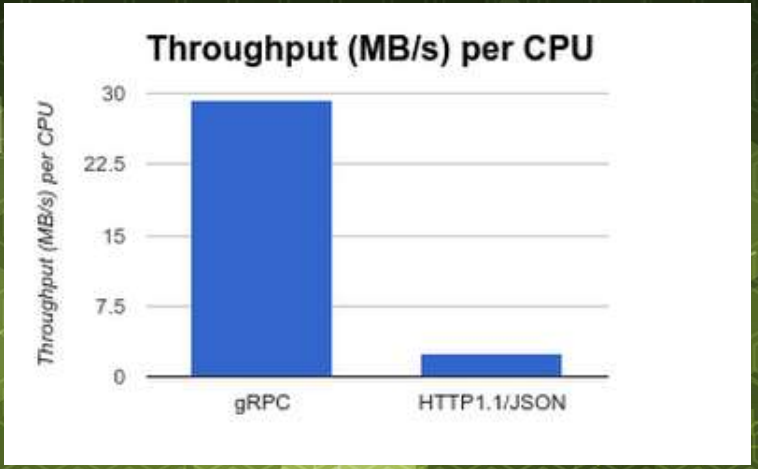
Source https://devcentral.f5.com/

HTTP/2

http://www.http2demo.io/

Protocol Buffers / HTTP2





3x increase throughput
publishing 50KB messages at maximum throughput from a single n1-highcpu-16 Google Compute Engine Virtual Machine instance, using 9 gRPC channels.

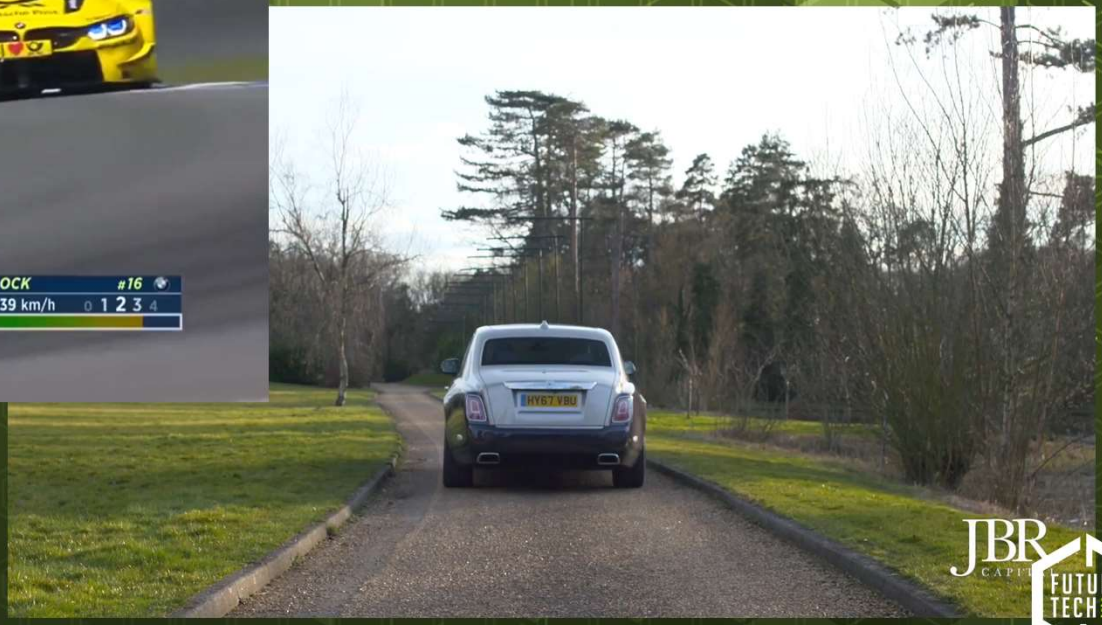
11x difference per CPU
More impressive than the almost 3x increase in throughput, is that it took only 1/4 of the CPU resources

Source M.Atamel, R.Tsang @Google https://cloud.google.com/blog/products/gcp/announcing-grpc-alpha-for-google-cloud-pubsub

Performance vs Confort

Which software are you developing?

Source: Tuning Roadshow YouTube Channel



Source: various YouTube Videos

gRPC Implementations

Natively Implemented in three languages

gRPC-C-Core

gRPC C++, C#, Ruby, Node, Python, PHP and Objective-C wrapped around gRPC-C-Core

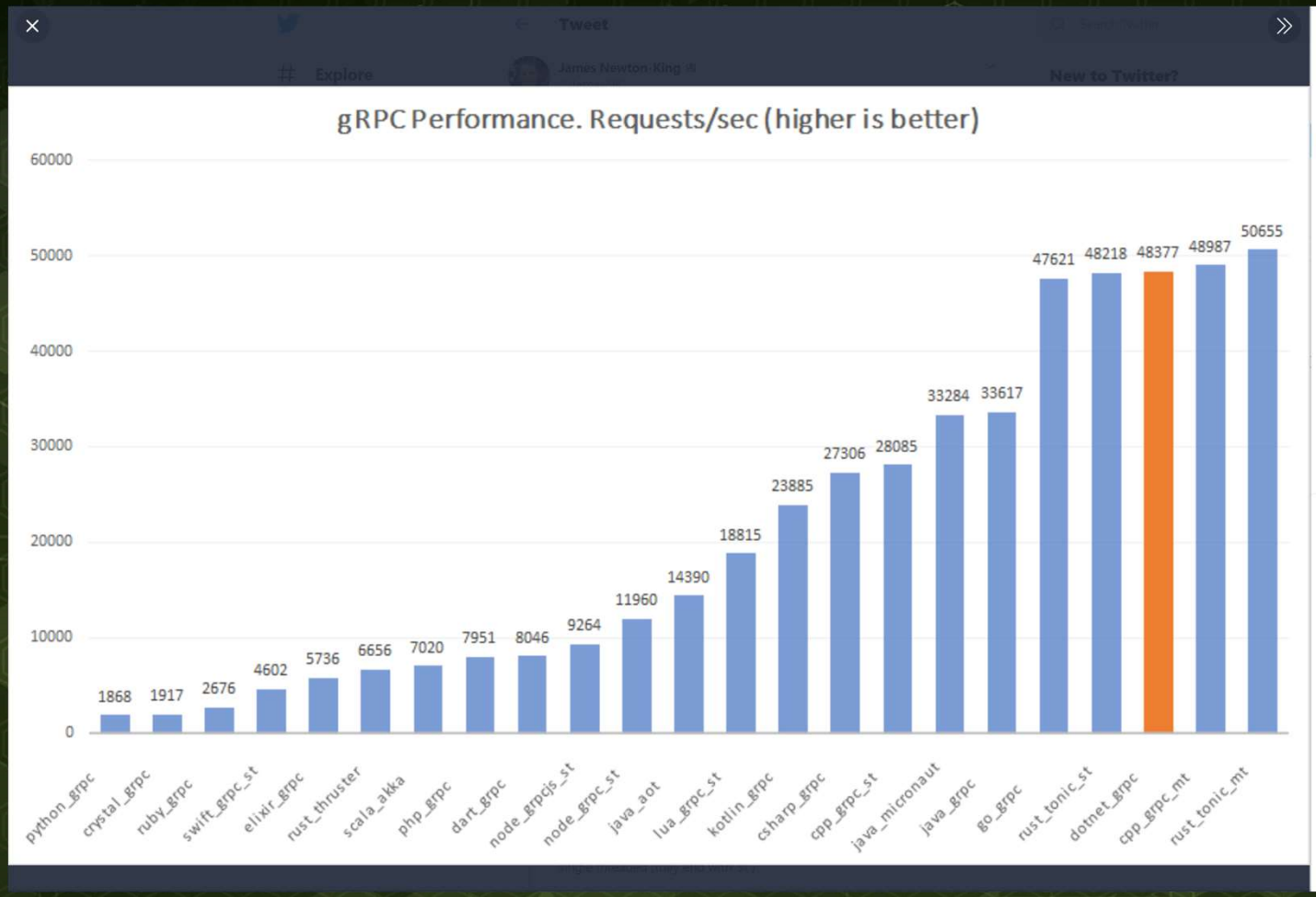
gRPC Java

gRPC GO

gRPC Dart and Swift are coming soon

C++ vs Java…
will gRPC C++ vs gRPC Java be the same?

...and maybe .Net will beat them all ☺



Source https://twitter.com/JamesNK/status/1290411721660592128/photo/1

Protocol Buffers

Google’s Lingua Franca for serializing data: RPC and storage

Binary data representation

Structure can be extended and maintain backward compatibility

Code generation for many languages

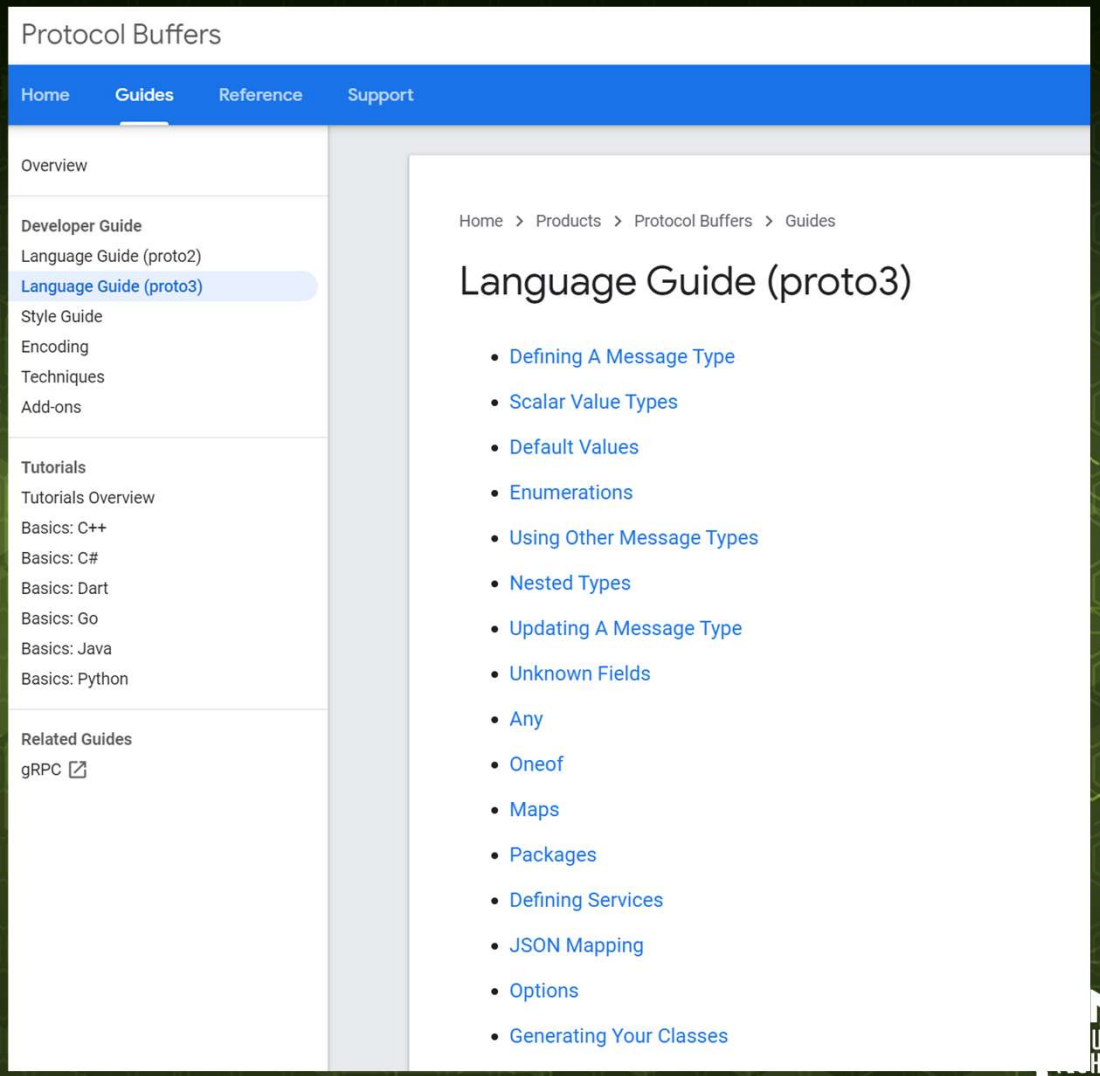
Strongly typed

Not required for gRPC

```
syntax = "proto3";

package greet;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply);
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Protobuf Language guide

https://developers.google.com/protocol-buffers/docs/proto3



gRPC – Unary Type

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

```
syntax = "proto3";

package sensorsystem;

// service definition
service SensorService {
  // unary
  rpc GetAvailableSensors (AvailableSensorsRequest) returns (AvailableSensorsResponse);
}

// The request message
message AvailableSensorsRequest {
  string username = 1;
  string message = 2;
}

// The response message
message AvailableSensorsResponse {
  string message = 1;
  string devices = 2;
}

message Device {
  string username = 1;
  string message = 2;
  bytes content = 3;
}
```

```
0 references | Gian Paolo Santopaolo, 8 hours ago | 1 author, 1 change
class Program
{
    0 references | Gian Paolo Santopaolo, 8 hours ago | 1 author, 1 change
    static async Task Main(string[] args)
    {
        var channel = GrpcChannel.ForAddress("https://localhost:5001");
        var client = new SensorService.SensorServiceClient(channel);

        var reply = await client.GetAvailableSensorsAsync(new Sensorsystem.AvailableSensorsRequest
            { Username = "username", Message = "message from client", });
        Console.WriteLine($"Server response: { reply.Message }");
        Console.WriteLine($"Devices from server: { reply.Devices }");

        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }
}
```

gRPC – Server Streaming Type

The client sends a request to the server and gets a stream to read a sequence of message back.
ResponseStream.MoveNext() reads messages streamed from the service.
The client reads from the stream until there are no more messages.
The server streaming call is complete when ResponseStream.MoveNext() returns false.

```
syntax = "proto3";

import "google/protobuf/empty.proto";

package sensorsystem;

// service definition
service SensorService {
  // unary
  rpc GetAvailableSensors (AvailableSensorsRequest) returns (AvailableSensorsResp

  // Server streaming
  rpc ReceiveTemperatureUpdates (TemperatureRequest) returns (stream TemperatureD

}

message TemperatureRequest {
  int32 deviceid = 1;
}

message TemperatureData {
  string devicelocation = 1;
  int32 temperature = 2;
}
```

```
0 references | Gian Paolo Santopaolo, Less than 5 minutes ago | 1 author, 2 changes
class Program
{
    0 references | Gian Paolo Santopaolo, 5 hours ago | 1 author, 1 change
    static async Task Main(string[] args)
    {
        var channel = GrpcChannel.ForAddress("https://localhost:5001");
        var client = new SensorService.SensorServiceClient(channel);

        //Task unarySend = SendUnaryDataAasync(client);
        //Task clientStreaming = SendStremingDataAasync(client);
        Task serverStreaming = ReceiveStreamingDataAsync(client, 1);

        //await Task.WhenAll(unarySend, clientStreaming, serverStreaming);

        await Task.WhenAll(serverStreaming);


        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }

    //server streaming
    1 reference | Gian Paolo Santopaolo, Less than 5 minutes ago | 1 author, 2 changes
    private static async Task ReceiveStreamingDataAsync(SensorService.SensorServiceClient client, int deviceId)
    {
        var cts = new CancellationTokenSource();
        cts.CancelAfter(TimeSpan.FromSeconds(5.0));

        using (var call = client.ReceiveTemperatureUpdates(new TemperatureRequest { Deviceid = deviceId }, cancellationToken: cts.T
        {
            try
            {
                await foreach (var message in call.ResponseStream.ReadAllAsync())
                {
                    Console.WriteLine($"Temperature for device located at: {message.Devicelocation} is {message.Temperature}");
                }
            }
            catch (RpcException ex) when (ex.StatusCode == StatusCode.Cancelled)
            {
                Console.WriteLine("Stream canceled.");
            }
        }
    }
```

gRPC – Client Streaming Type

The client send a sequence of messages to the server using a stream. Once the client has finished writing the messages it waits for the server to read them and return response.

```
syntax = "proto3";

import "google/protobuf/empty.proto";

package sensorsystem;

// service definition
service SensorService {
  // unary
  rpc GetAvailableSensors (AvailableSensorsRequest) returns (AvailableSensorsResponse);

  // Client streaming
  rpc SendSensorData (stream SensorData) returns (SensorDataResponse);
  rpc SendSensorDataNoResponse (stream SensorData) returns (google.protobuf.Empty);
}

// The request message
message AvailableSensorsRequest {
  string username = 1;
  string message = 2;
}

// The response message
message AvailableSensorsResponse {
  string message = 1;
  string devices = 2;
}

message SensorData {
  string data1 = 1;
  int32 data2 = 2;
}

message SensorDataResponse {
  string message = 1;
}

message Device {
  string username = 1;
  string message = 2;
  bytes content = 3;
}
```

```
0 references | Gian Paolo Santopaolo, 7 hours ago | 1 author, 1 change
class Program
{
    0 references | Gian Paolo Santopaolo, 7 hours ago | 1 author, 1 change
    static async Task Main(string[] args)
    {
        var channel = GrpcChannel.ForAddress("https://localhost:5001");
        var client = new SensorService.SensorServiceClient(channel);

        Task unarySend = SendUnaryDataAasync(client);
        Task clientStreaming = SendStremingDataAasync(client);

        await Task.WhenAll(unarySend, clientStreaming);


        Console.WriteLine("Press any key to exit...");
        Console.ReadKey();
    }

    1 reference | Gian Paolo Santopaolo, 7 hours ago | 1 author, 1 change
    private static async Task SendUnaryDataAasync(SensorService.SensorServiceClient client)...

    1 reference | Gian Paolo Santopaolo, 7 hours ago | 1 author, 1 change
    private static async Task SendStremingDataAasync(SensorService.SensorServiceClient client)
    {
        using var call = client.SendSensorData();
        for (var i = 0; i < 100; i++)
        {
            await call.RequestStream.WriteAsync(new SensorData { Data1 = $"Message{i}, Data2 = {i}" });
            await Task.Delay(200);
        }

        await call.RequestStream.CompleteAsync();

        var response = await call;
        Console.WriteLine($"Message response: {response.Message}");
    }
}
```

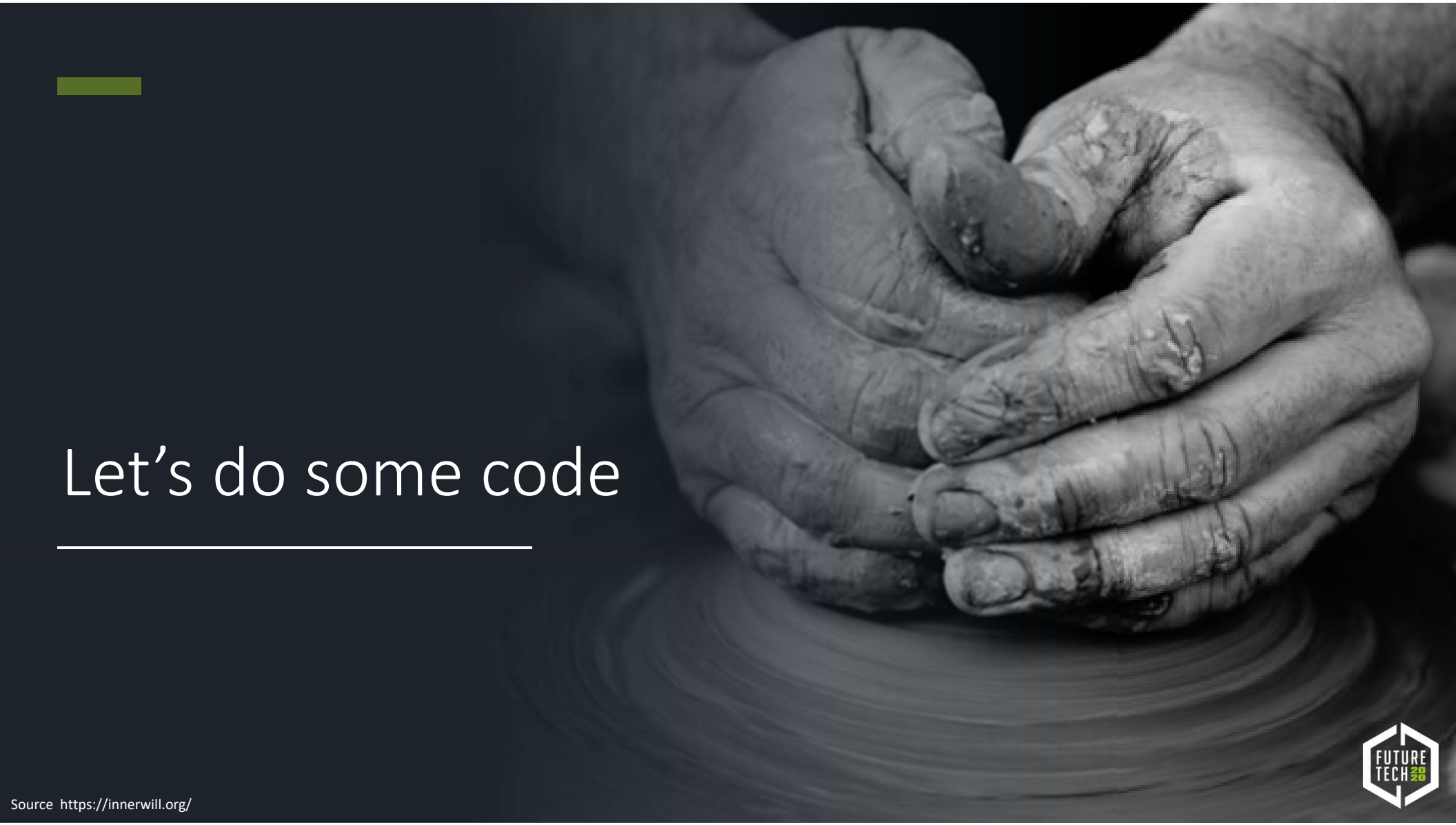
gRPC – Bidi Type

Both sides send a sequence of messages using a read-write stream. The two streams operate independently. The order of messages in each stream is preserved

```
import "google/protobuf/empty.proto";

package sensorsystem;

// service definition
service SensorService {
  // unary
  rpc GetAvailableSensors (AvailableSensorsRequest) returns (AvailableSensorsResponse);

  // Client streaming
  rpc SendSensorData (stream SensorData) returns (SensorDataResponse);
  rpc SendSensorDataNoResponse (stream SensorData) returns (google.protobuf.Empty);

  // Server streaming
  rpc ReceiveTemperatureUpdates (TemperatureRequest) returns (stream TemperatureData);

  // BI-Directional streaming
  rpc StreamData (stream SensorData) returns (stream SensorData);
}

// The request message
message AvailableSensorsRequest {
  string username = 1;
  string message = 2;
}
```

```
0 references | Gian Paolo Santopaolo, 11 hours ago | 1 author, 2 changes
static async Task Main(string[] args)
{
    var randomizer = new Random();
    var channel = GrpcChannel.ForAddress("https://localhost:5001");
    var client = new SensorService.SensorServiceClient(channel);

    Console.WriteLine("Please type the device ID:");
    var line = Console.ReadLine();

    int sensorId = 0;
    int.TryParse(line, out sensorId);


    using (var cli = client.StreamData())
    {
        _ = Task.Run(async () =>
        {
            while (await cli.ResponseStream.MoveNext(cancellationToken: CancellationToken.None))
            {
                var response = cli.ResponseStream.Current;
                Console.WriteLine($"Receiving data from Sensor {response.SensorID} sent data1 {response.Data1}, dat
            }
        });


        Console.WriteLine("Press any key to send random data or Q to stop.");

        while ((line = Console.ReadLine()) != null)
        {
            if (line.ToLower() == "q")
            {
                break;
            }

            int data1 = randomizer.Next(1, 100), data2 = randomizer.Next(1, 100);
            await cli.RequestStream.WriteAsync(new SensorData { SensorID = sensorId, Data1 = data1.ToString(), Data
            Console.WriteLine($"Sending data Sensor {sensorId} data1 {data1}, data2 {data2}");
        }
    }
```

Let’s do some code

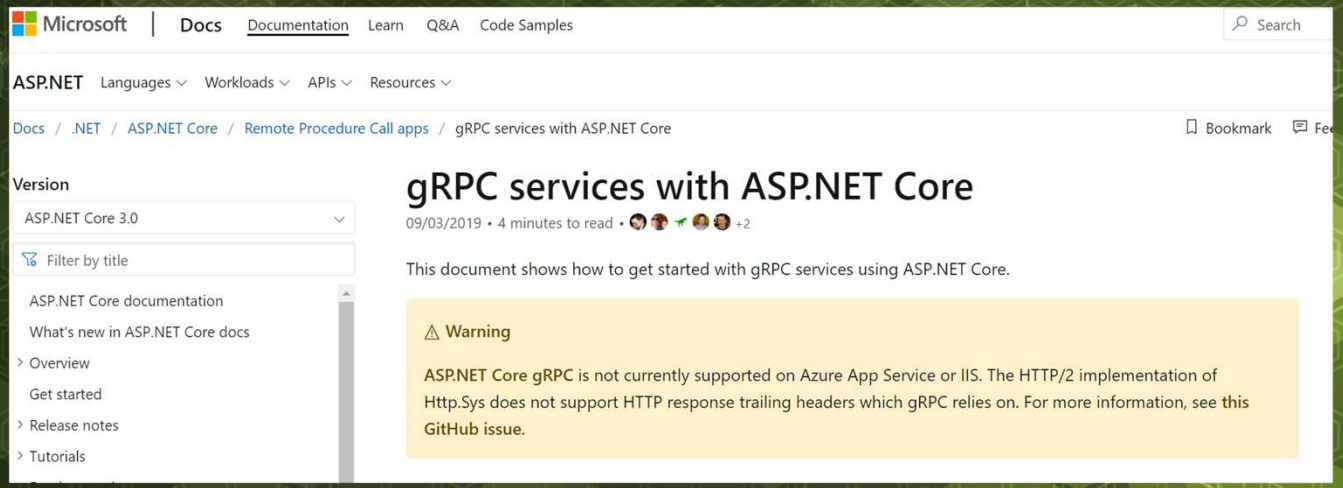


Source https://innerwill.org/

gRPC – and HTTP2

gRPC not supported in App Service and IIS

https://docs.microsoft.com/en-us/aspnet/core/grpc/aspnetcore?view=aspnetcore-3.0&tabs=visual-studio

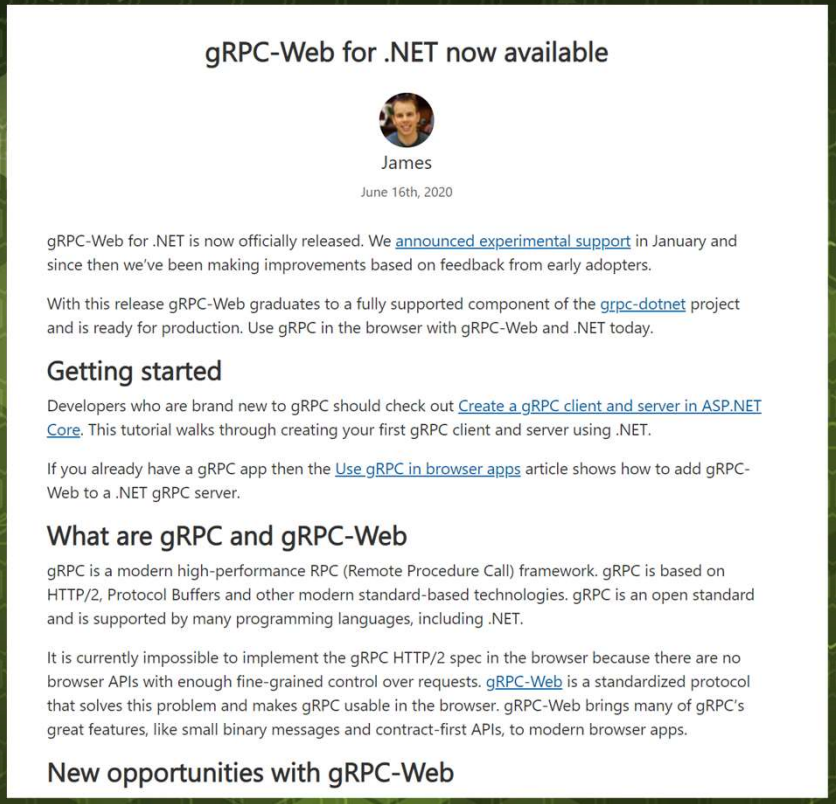


gRPC – and HTTP2

Initially only server to server

Today
Web gRPC (no bidi)

gRPC-Web for .NET now available
https://devblogs.microsoft.com/aspnet/grpc-web-for-net-now-available/



gRPC-Web for .NET now available

James

June 16th, 2020

gRPC-Web for .NET is now officially released. We announced experimental support in January and since then we've been making improvements based on feedback from early adopters.

With this release gRPC-Web graduates to a fully supported component of the grpc-dotnet project and is ready for production. Use gRPC in the browser with gRPC-Web and .NET today.

Getting started

Developers who are brand new to gRPC should check out Create a gRPC client and server in ASP.NET Core. This tutorial walks through creating your first gRPC client and server using .NET.

If you already have a gRPC app then the Use gRPC in browser apps article shows how to add gRPC-Web to a .NET gRPC server.

What are gRPC and gRPC-Web

gRPC is a modern high-performance RPC (Remote Procedure Call) framework. gRPC is based on HTTP/2, Protocol Buffers and other modern standard-based technologies. gRPC is an open standard and is supported by many programming languages, including .NET.

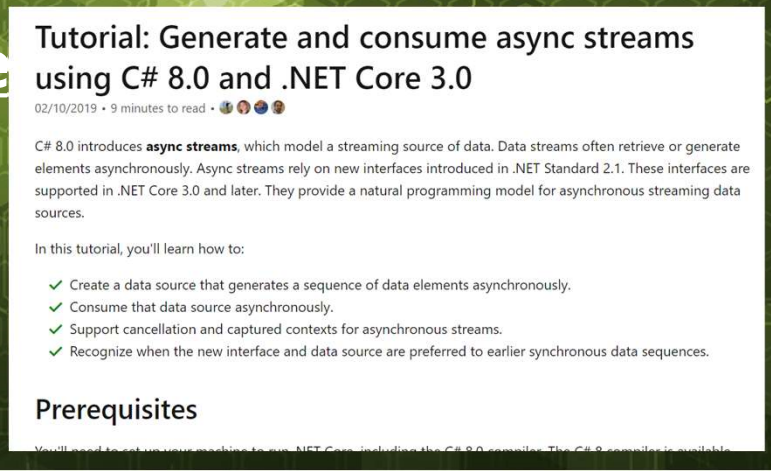
It is currently impossible to implement the gRPC HTTP/2 spec in the browser because there are no browser APIs with enough fine-grained control over requests. gRPC-Web is a standardized protocol that solves this problem and makes gRPC usable in the browser. gRPC-Web brings many of gRPC's great features, like small binary messages and contract-first APIs, to modern browser apps.

New opportunities with gRPC-Web

Async Streaming and gRPC with async enumerables
IAsyncEnumerable<T> and Span<T>

Generate and consume async streams using C# 8.0 and .NET Core 3.0

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/generate-asynchronous-stream



Tutorial: Generate and consume async streams using C# 8.0 and .NET Core 3.0

02/10/2019 • 9 minutes to read •

C# 8.0 introduces **async streams**, which model a streaming source of data. Data streams often retrieve or generate elements asynchronously. Async streams rely on new interfaces introduced in .NET Standard 2.1. These interfaces are supported in .NET Core 3.0 and later. They provide a natural programming model for asynchronous streaming data sources.

In this tutorial, you'll learn how to:

- ✓ Create a data source that generates a sequence of data elements asynchronously.
- ✓ Consume that data source asynchronously.
- ✓ Support cancellation and captured contexts for asynchronous streams.
- ✓ Recognize when the new interface and data source are preferred to earlier synchronous data sequences.

Prerequisites

Got questions ?

Sassion repo: https://github.com/gsantopaolo/grpc-streaming

gRPC not supported in App Service and IIS: https://docs.microsoft.com/en-us/aspnet/core/grpc/aspnetcore?view=aspnetcore-3.0&tabs=visual-studio

gRPC-Web for .NET now available: https://devblogs.microsoft.com/aspnet/grpc-web-for-net-now-available/

Async stream: https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/generate-consume-asynchronous-stream

Protobuf Language guide https://developers.google.com/protocol-buffers/docs/proto3

Thank You!

@gsantopaolo

FUTURE TECH 2020

THE IT CONFERENCE FOR MICROSOFT TECHNOLOGIES