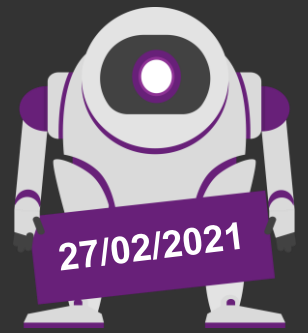
2021

Netcoreconf

gRPC – Introducción a desarrolladores ASP.NET Core



Jorge Serrano Pérez

Software Developer

@J0rgeSerran0

SPONSORS

Microsoft

plain concepts

avanade

dinahosting

everis
an NTT DATA Company

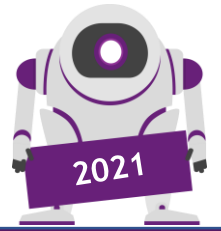
intelequia

iliria

encamina
PIENSA EN COLORES

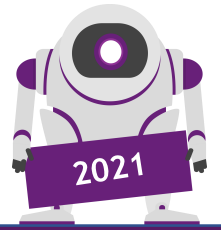
Verne
TECHNOLOGY GROUP

sessionize.com



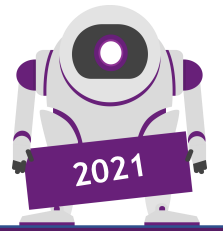
Agenda

1. Historia
2. gRPC
3. Comparación con otras tecnologías
4. Demo gRPC



Agenda

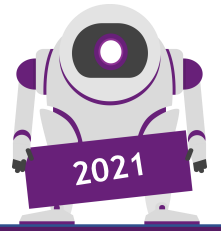
1. Historia



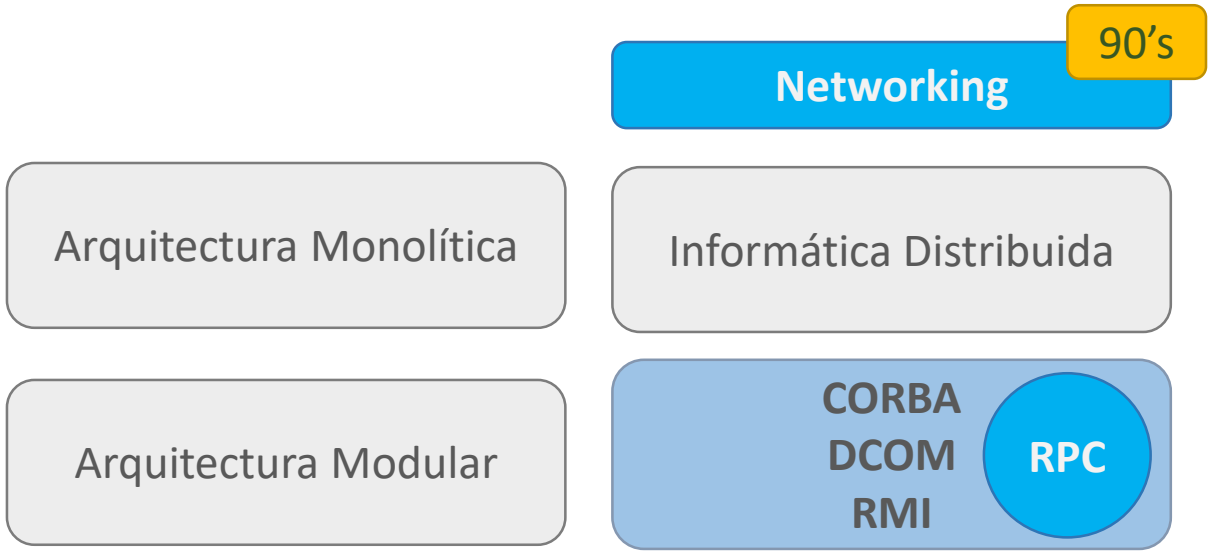
Inicios

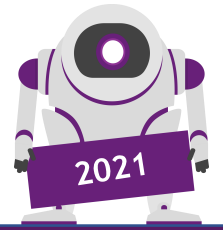
1960s, 1970s, 1981 - RPC (Remote Procedure Call)

- Permite la comunicación entre cliente y servidor
- Utilizado en muchos sistemas distribuidos
- Resuelve diferentes problemas como la seguridad, sincronización, el manejo de flujos de datos, etc.
- Permite ejecutar código en otra máquina remota sin preocuparnos de las comunicaciones
- Utilizado por (*NFS [Network File System], D-Bus, CORBA [Common Object Request Broker Architecture], RMI [Java Remote Method Invocation], SOAP [Simple Object Access Protocol] basado en XML-RPC, JSON-RPC, Apache Thrift, Apache Avro, gRPC*, etc)



Arquitecturas





Arquitecturas

Web (2000's)

HTTP
XML
SOAP
WSDL
RPC

Web 2.0 y Cloud

HTTP/XML/SOAP
HTTP/JSON
REST APIs

Microservices

Interfaces definidas a través de APIs

Microservices

Web
Mobile Devices
IoT

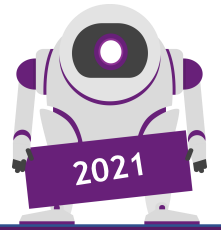
Problemas

Capacidad de proceso
Ancho de banda

HTTP/JSON con REST

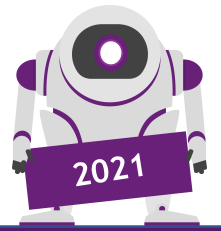
2016

GRPC



Agenda

2. gRPC



Orígen de gRPC

Jul 2008
Versión inicial de Protocol Buffers
Versión 1 de Protocol Buffers interna para Google
Mejoras y Versión 2 de Protocol Buffers

Dec 2014
Versión 3 alpha de Protocol Buffers
Desarrollo en paralelo con gRPC

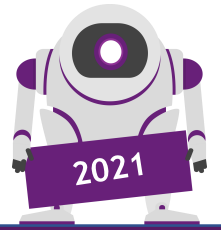
Mar 2015
Google publica gRPC como open source
gRPC utiliza HTTP/2
gRPC está influenciado por Stubby (RPC Framework de Google)

May 2015
HTTP/2 es publicado como RFC 7540

Jul 2016
Versión 3 de Protocol Buffers

Aug 2016
Google publica la versión 1.0 de gRPC
Considerada estable y óptima para entornos de producción

Mar 2017
CNCF (Cloud Native Computing Foundation) añade
gRPC a su porfolio



gRPC – Qué

gRPC (2015)

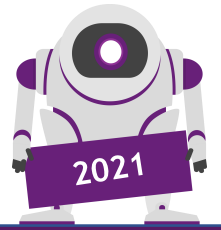
- Proyecto Open Source de Google (https://github.com/grpc/)
- Es un Framework
- Agnóstico al lenguaje de programación
- Es una versión actual de Stubby RPC utilizada por Google
- Está basado en RPC (Remote Procedure Call)
- RPC + Streaming
- Nos permite exponer servicios para que sean consumidos por aplicaciones externas

ASP.NET Core

- La implementación en C# está disponible en el sitio oficial gRPC
  Implementación basada en la librería nativa escrita en C (gRPC C-core)

ASP.NET Core 3 y superior

- Implementación basada en Kestrel HTTP Server y ASP.NET Core (manejado ASP.NET Core-based gRPC)



gRPC – Características

Multilenguaje

- Java, Go, C, C++, Node.js, Python, Ruby, Objective-C, PHP, C#,…

Multiplataforma

- Windows, Linux, macOS and Mobile Devices, Webs, IoT

Como IDL (Interface Definition Language) usa protocol buffers (Contract-First API Development)

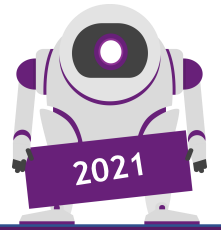
- Se usa para configurar las comunicaciones fuertemente tipadas entre clientes y servidores en RPC
- Describe la interfaz del servicio, los métodos y la estructura de los mensajes *payload* de ida y vuelta

Protocol Buffer (protobuf) => .proto

- Lenguaje independiente de Google y agnóstico de plataforma
- Con mecanismos extensibles para la serialización binaria de datos estructurados
- Rápido, de menor tamaño (menos uso de red) y más simple

Para el transporte, se apoya en el stándard HTTP/2 (RCF 7540)

- Streaming bidireccional y compresión de cabeceras
- Multiplexing y control de flujo (congestión) en una conexión TCP (una conexión TCP soporta muchos streams que pueden ser ejecutados en paralelo sin tener que esperar a otra conexión como sí pasa en HTTP 1.1)
- Alto rendimiento



gRPC – Características

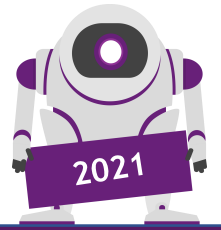
Asíncrono => por naturaleza, las redes son inherentemente asíncronas => no bloquear hilos

Extensible

- Tracing
- Monitoring
- Load Balancing
- Health Checking
- Service Discovery
- Auth & Security (TLC, OAuth, etc)
- Proxies

Otros

- Instalación rápida y sencilla
- IDL fácilmente transportable
- Reconexión es automática en conexiones con problemas
- Propagación de errores
- Propagación de cancelación
- QoS (Quality of Service)
- Soporte limitado en navegadores Web (gRPC-Web, aunque no soporta todo gRPC)



gRPC – Beneficios

Compatibilidad

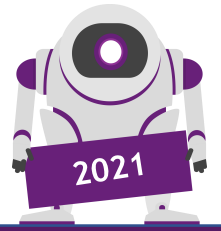
- Multiplataforma
- Multilenguaje
- Evita los *breaking changes*

Rendimiento

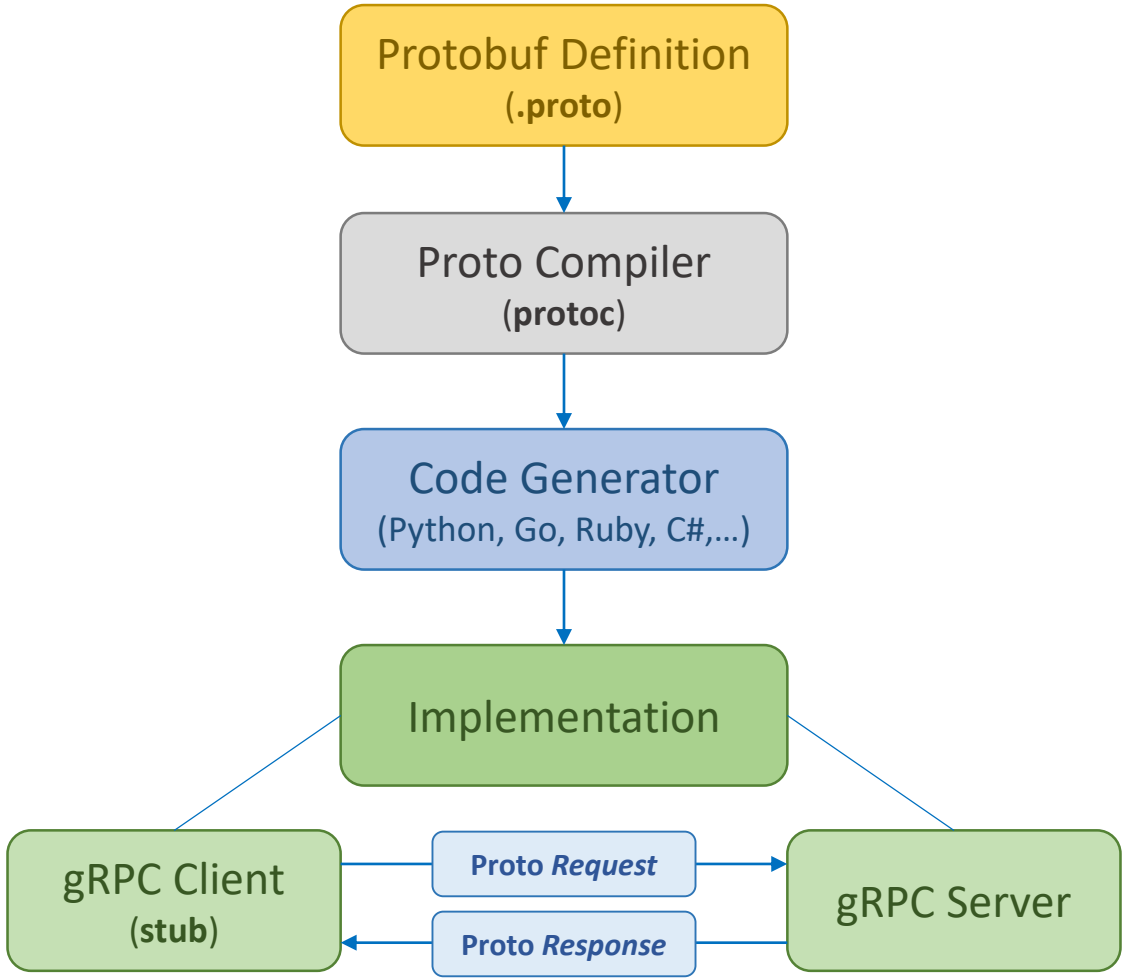
- Manejo de conexiones de red (permite múltiples llamadas por conexión)
- Transmisión rápida de datos (carácter binario)
- Mejora el uso/consumo de recursos (CPU)

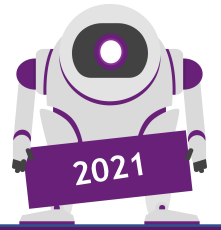
Mantenimiento (a través de plugins)

- Tracing/Logging
- Monitoring



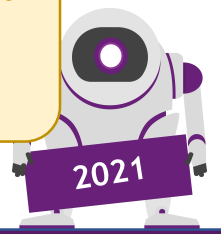
gRPC – Flujo de Trabajo





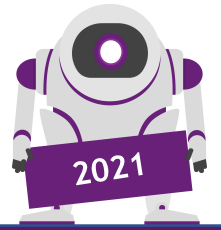
gRPC – Tipos de Servicios

Unary RPCs	El cliente envía al servidor una petición simple El cliente recibe del servidor una respuesta simple
Server Streaming RPCs	El cliente envía al servidor una petición El servidor envía streams El cliente lee los streams devueltos por el servidor hasta que no tiene más mensajes *gRPC garantiza el orden de los mensajes dentro de la petición*
Client Streaming RPCs	El cliente escribe una secuencia de mensajes que envía al servidor por stream El servidor leerá toda la secuencia de mensajes según le van llegando Una vez que el cliente envía todos los mensajes, espera la respuesta del servidor *gRPC garantiza el orden de los mensajes dentro de la petición*
Bidirectional Streaming RPCs	Los extremos envían una secuencia de mensajes con un stream de lectura/escritura Los streams operan de forma independiente Cliente y Servidor pueden leer y escribir en cualquier orden



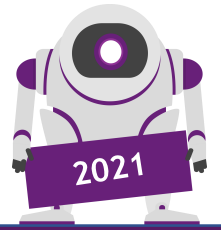
gRPC – Tipos de Servicios (Ejemplos)

Unary RPCs	rpc Unary(MyRequest) returns (MyResponse);
Server Streaming RPCs	rpc ServerStreaming(MyRequest) returns (stream MyResponse);
Client Streaming RPCs	rpc ClientStreaming(stream MyRequest) returns (MyResponse);
Bidirectional Streaming RPCs	rpc BidirectionalStreaming(stream MyRequest) returns (stream MyResponse);



gRPC – C# Data Types vs Proto Data Types

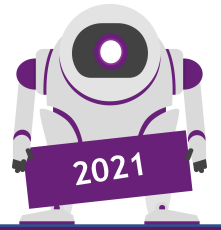
C# Data Type	.proto Data Type
double	double
float	float
int	int32 sint32 sfixed32
long	int64 sint64 sfixed64
uint	uint32 fixed32
ulong	uint64 fixed64
bool	bool
string	string
ByteString	bytes



gRPC – Status Codes

https://godoc.org/google.golang.org/grpc/codes/

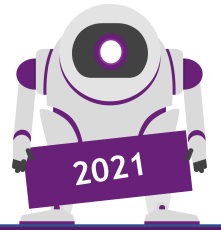
OK	=> 200 OK
CANCELED	=> 499 Client Closed Request
UNKNOWN	=> 500 Internet Server Error
INVALID_ARGUMENT	=> 400 Bad Request
DEADLINE_EXCEEDED	=> 504 Gateway Timeout
NOT_FOUND	=> 404 Not Found
ALREADY_EXISTS	=> 409 Conflict
PERMISSION_DENIED	=> 403 Forbidden
RESOURCE_EXHAUSTED	=> 429 Too Many Requests
FAILED_PRECONDITION	=> 400 Bad Request
ABORTED	=> 409 Conflict
OUT_OF_RANGE	=> 400 Bad Request
UNIMPLEMENTED	=> 501 Not Implemented
INTERNAL	=> 500 Internal Server Error
UNAVAILABLE	=> 503 Service Unavailable
DATA_LOSS	=> 500 Internet Server Error
UNAUTHENTICATED	=> 401 Unauthorized



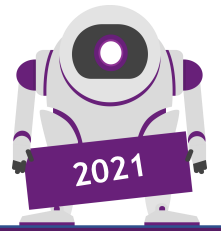
gRPC – C#

gRPC C#
https://github.com/grpc/grpc/tree/master/src/csharp

gRPC for .NET
https://github.com/grpc/grpc-dotnet

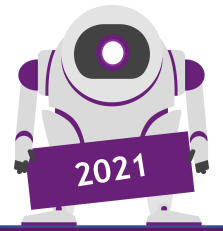


3. Comparación con otras tecnologías



gRPC – vs WCF, REST, SignalR, GraphQL

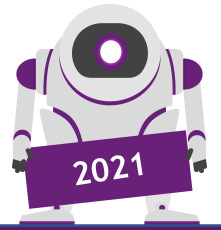
No hay tecnología perfecta para todos los escenarios



gRPC – vs WCF

WCF (Windows Communication Foundation)

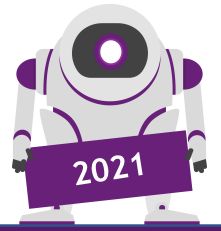
- El uso de WCF en .NET Core no es directo
- La configuración de WCF es tediosa y poco legible
- Se requiere crear, probar, distribuir y descubrir los servicios
- Dependiendo de la solución adoptada, los *payloads* entre cliente y servidor, pueden llegar a ser largos, con un consumo de red elevado, y un rendimiento peor que otras soluciones
- Pasar WCF a .NET Core no es una opción válida debido a su complejidad



gRPC – vs REST

REST

- Multiplataforma
- Utilizable para Microservicios, *pero no sólo para Microservicios*
- Posibilidad de utilizarlo desde cualquier lenguaje moderno
- Define un conjunto de operaciones fijas (GET, POST, PUT, DELETE)
- Usa HTTP 1.x (normalmente HTTP 1.1) mientras que gRPC usa HTTP/2 (más simple y compacto)
- HTTP 1.1 requiere TCP *handshake* por cada petición, HTTP/2 guarda la conexión abierta
- Posibilidad de utilizarlo en todos los navegadores Web (Blazor, Angular, React)
- Sencillo de ser probado (gRPC no es fácil de ser probado)
- Serialización/deserialización de JSON o XML. Trabajar con ellos consume tiempo y recursos
- Menos eficiente que gRPC que serializa binariamente
- gRPC representa un paradigma diferente a la hora de desarrollar servicios



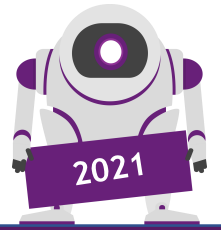
gRPC – vs SignalR & GraphQL

SignalR

- Multicasting o multidifusión (gRPC no es bueno en este aspecto)
- Permite una comunicación bidireccional en tiempo real. Además de las peticiones de cliente a servidor, el servidor puede enviar también, notificaciones a todos los clientes
- Comunicación en tiempo real con un servidor (chat)

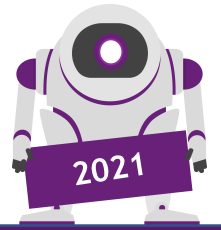
GraphQL

- Permite obtener datos específicos de todo el conjunto o conjuntos de datos (en gRPC todo se realiza a través de contratos y no ofrece esta flexibilidad)
- Crearemos un solo endpoint muy flexible y que abarque diferentes tipos de peticiones, recuperando solamente, los datos que necesitamos



gRPC – Dónde

- Microservicios
- Dispositivos móviles, IoT (Internet of Things) o sistemas restrictivos
- Comunicación entre tecnologías que requieren streaming bidireccional
- En ambientes Web, gRPC-Web ayuda, pero cuidado con el soporte



Agenda

4. Demo gRPC

SPONSORS

Microsoft

plain concepts

avanade

dinahosting

everis
an NTT DATA Company

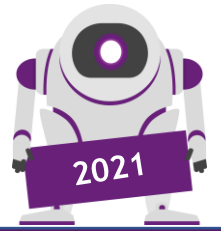
intelequia

iliria

encamina
PIENSA EN COLORES

Verne
TECHNOLOGY GROUP

sessionize.com



More information:
info@netcoreconf.com
@Netcoreconf

Visit on:
netcoreconf.com