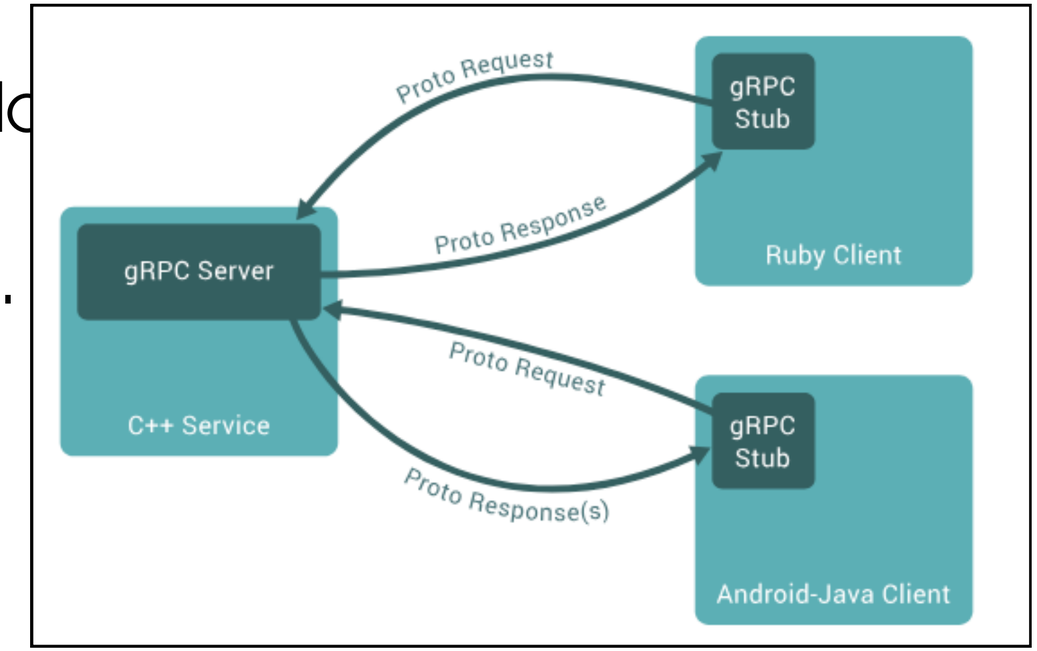
gRPC

Hugo Ramos López
Javier Marcos Tobar

Índice

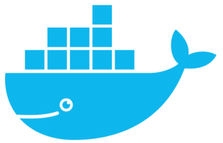
¿Qué es gRPC?

- **Framework open-source** creado por Google.
- **Sistema de comunicación remoto** basado en **RPC** (Remote Procedure Call).
- Usa **HTTP/2** como protocolo de transporte.
- Serializa los datos usando **Protocol Buffers** (protobuf).
- Está diseñado para ser **rápido**, **eficiente** y fuertemente **tipado**.
- Permite **comunicación entre microservicios** escrita en diferentes lenguajes.



Origen y contexto

- **Evolución moderna** del concepto clásico de RPC.
- Nace en 2015 como **reemplazo** eficiente frente a **REST** en sistemas de **alta carga**.
- Adoptado por Google, Netflix, Docker, Kubernetes, etc.
- Diseñado desde el principio para **microservicios y sistemas distribuidos**.


Arquitectura básica

- Se basa en **definir interfaces** mediante un archivo .proto.
- El .proto describe: **servicios**, **métodos** y **tipos** de mensajes.
- De este archivo se generan los **stubs** automáticamente para cada lenguaje.
- Comunicación: **cliente** → **stub** → **servidor**

.proto

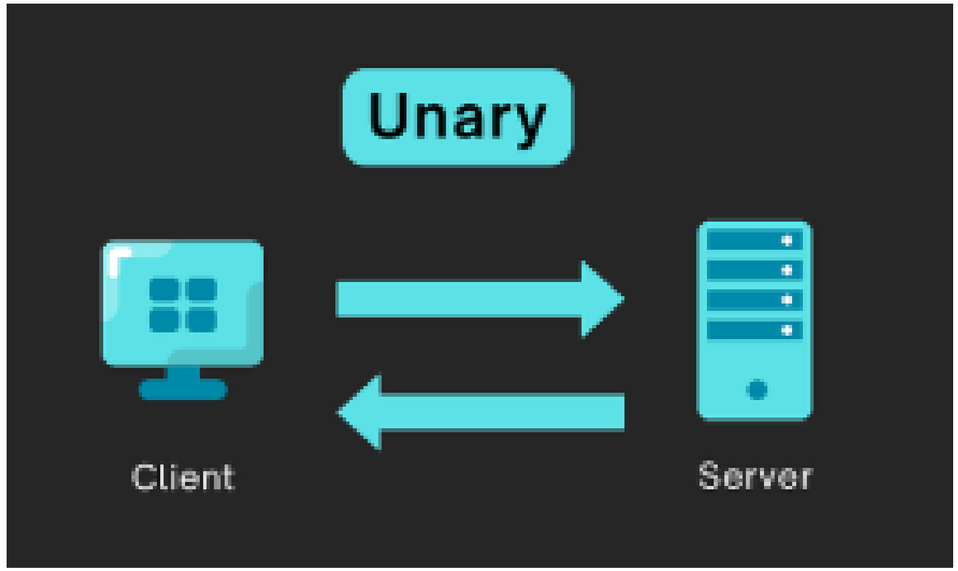
Protocol Buffers

- Es el **formato de serialización** de gRPC.
- Mucho **más eficiente** que JSON o XML.
- **Compacto**, **binario** y de **alto rendimiento**.
- Define estructuras **estrictas** y fuertemente **tipadas**.
- Permite **evolución del esquema** sin romper compatibilidad.

Modos de comunicación

Unary RPC

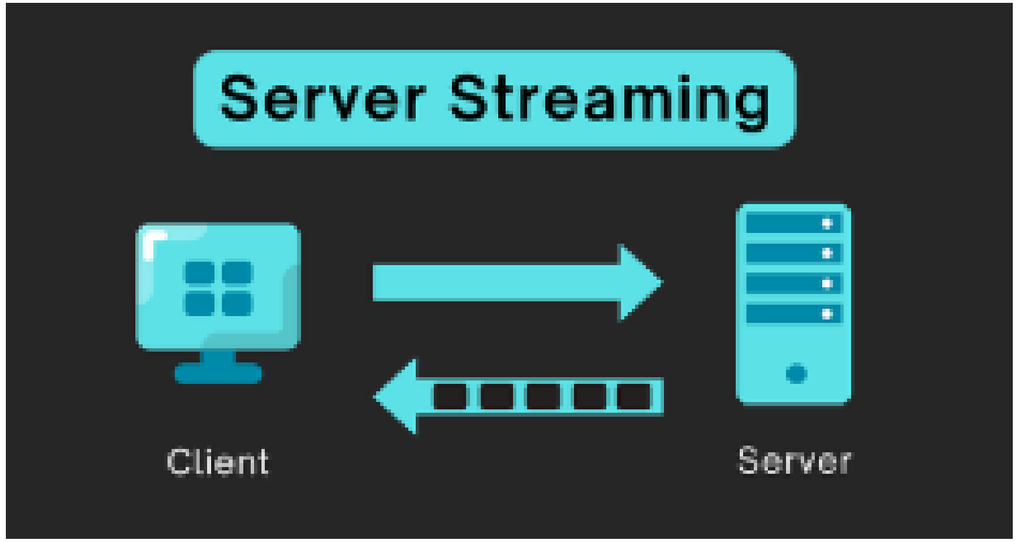
- Un intercambio directo: El cliente envía un mensaje y recibe.
- Equivalente a una petición REST típica



Modos de comunicación

Server-streaming RPC

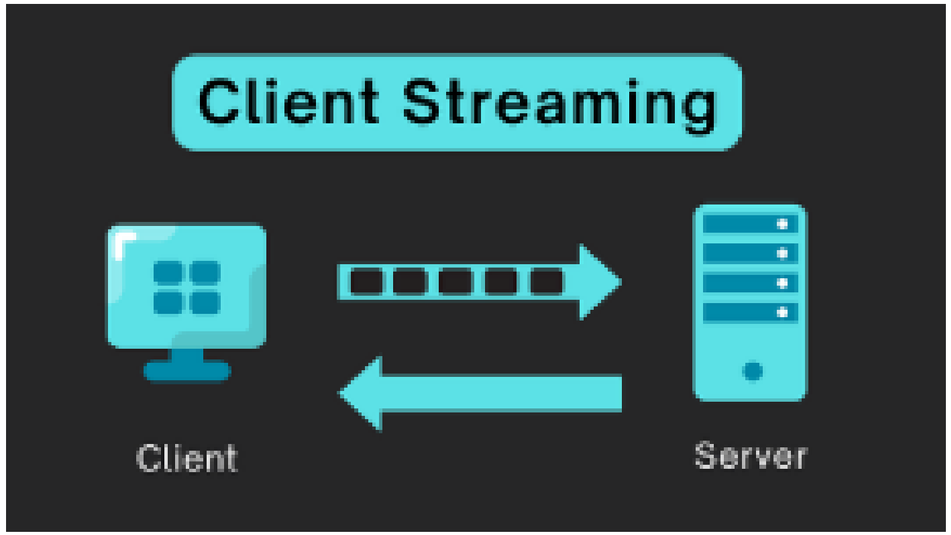
- El servidor envía múltiples mensajes en respuesta.



Modos de comunicación

Client-streaming RPC

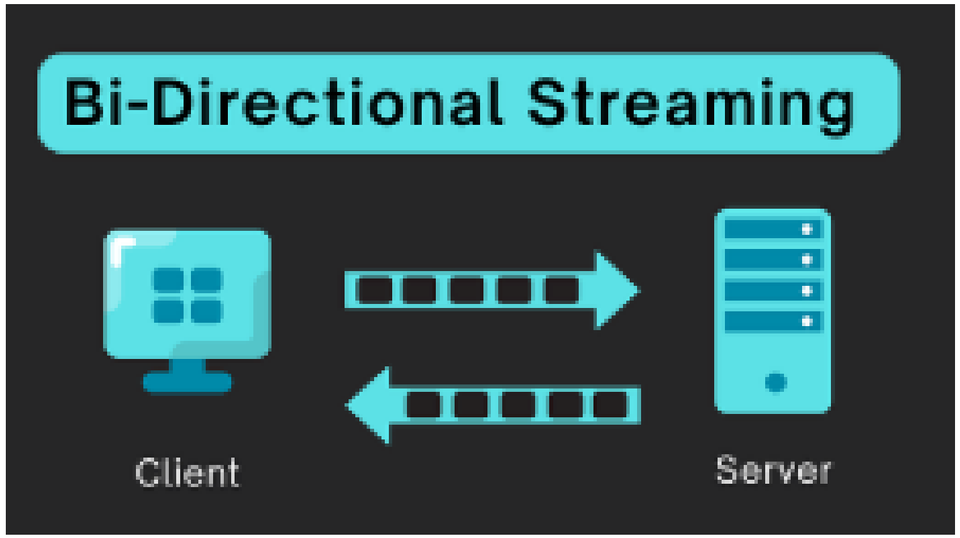
- El cliente envía un flujo de datos.



Modos de comunicación

Bidirectional streaming RPC

- Cliente y servidor intercambian flujos simultáneos.
- Ideal para chat, sensores, telemetría o señales en tiempo real.



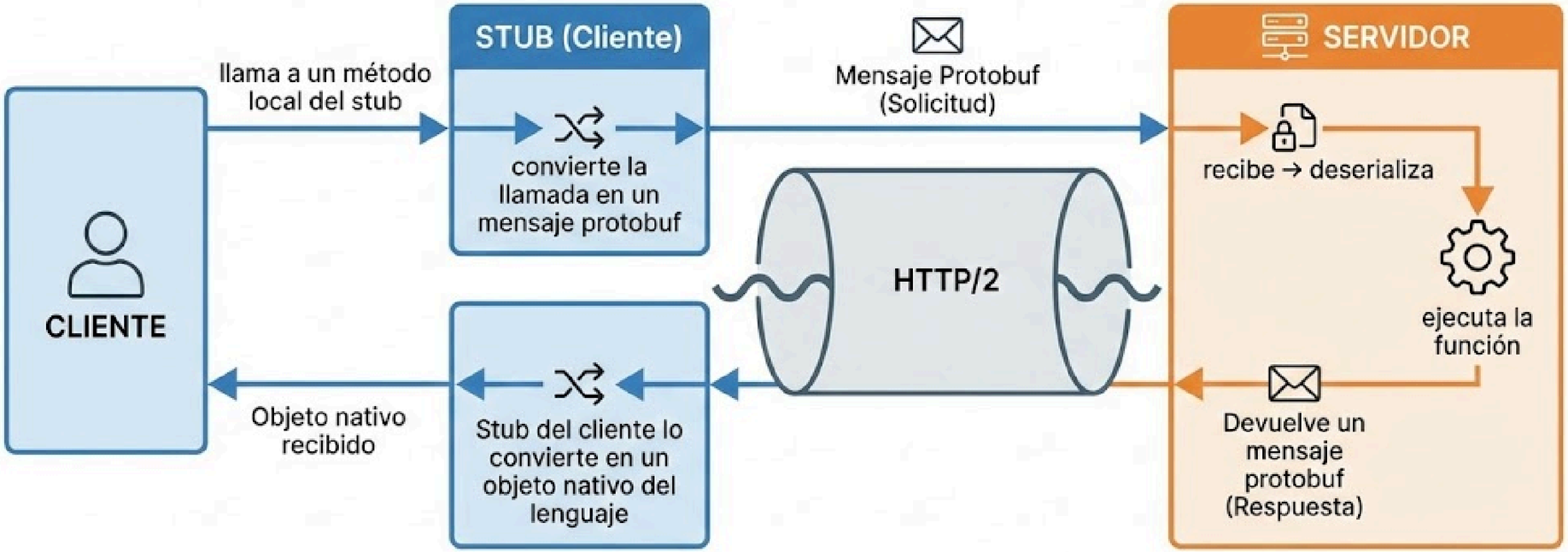
Ciclo de vida de una llamada

1. **Cliente** llama a un **método local** del stub.
2. **Stub** convierte la llamada en un **mensaje protobuf.**
3. **Mensaje** se envía sobre **HTTP/2**.

Ciclo de vida de una llamada

4. **El servidor** procesa la **petición.**
5. Envía la **respuesta** en formato **protobuf.**
6. **El cliente** la **transforma** en un **objeto del lenguaje.**

Ciclo de vida de una llamada



¿Cómo funciona internamente?

HTTP/2:

- Conexión **persistente**,
- **Multiplexación**,
- **Compresión** de cabeceras.

Protobuf:

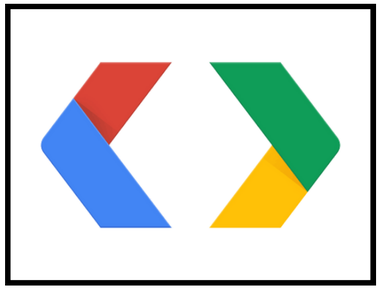
- Formato **binario** ultra-compacto.

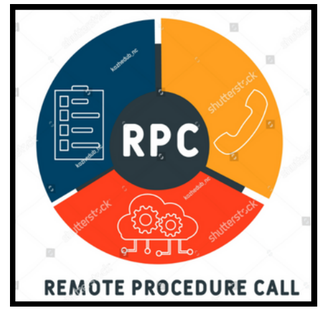
RPC:

- El **cliente invoca métodos** como si fueran locales.

Resultado:

- **velocidad** + **eficiencia** + **bajo consumo** de ancho de banda



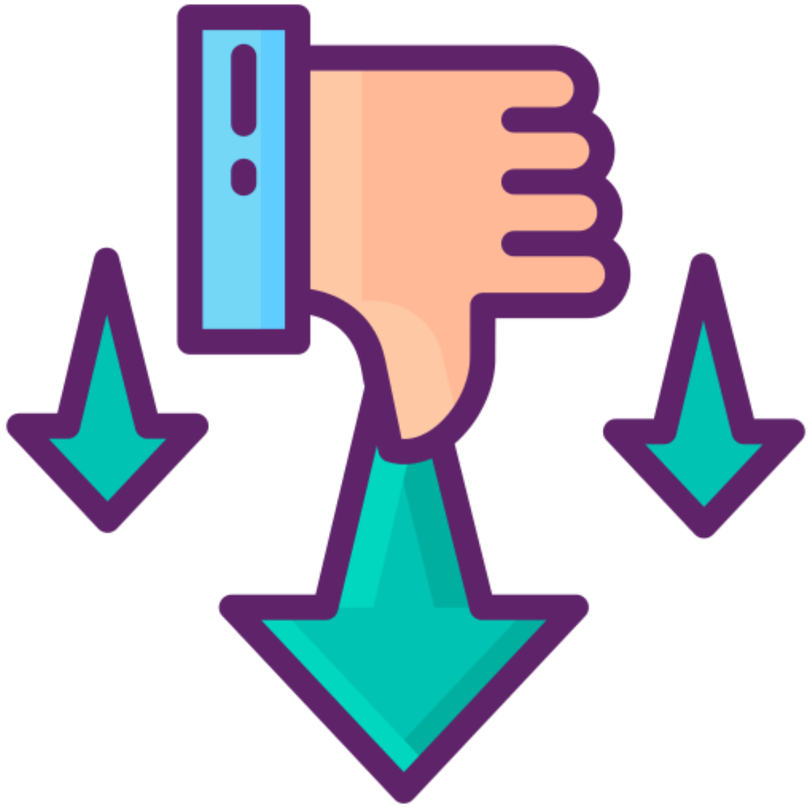


Ventajas de gRPC

- Alto rendimiento.
- Streaming **bidireccional** real.
- Tipado **estricto** y API autodocumentada.
- Mejor **eficiencia** que REST/JSON.
- **Interoperabilidad** multilenguaje.
- **Integración** natural con Kubernetes y microservicios.
- Ideal para sistemas internos de **backend**.

Inconvenientes de gRPC

- **No** está pensado para ser consumido **desde un navegador** directamente.
- Requiere herramientas específicas (generadores de código).
- Depuración **menos intuitiva** que REST.
- **No** siempre compatible con firewalls o proxies antiguos.
- **No** es la mejor opción para **APIs públicas**.



Casos de uso reales

- Microservicios de **alto** rendimiento.
- Sistemas internos de **grandes plataformas** (Google, Netflix).
- Comunicación entre **contenedores** en Kubernetes.
- Sistemas **IoT** y **telemetría**.
- Servicios en **tiempo real**: chat, audio, streaming.
- **Machine Learning** distribuido.

Lenguajes

- Java
- C++
- Python
- Go
- C#
- Node.js
- Ruby
- PHP
- Kotlin
- Swift

Ecosistema

- **Integración directa** con Docker, Kubernetes, Envoy.
- **Observabilidad**: OpenTelemetry, Prometheus.

Seguridad en gRPC

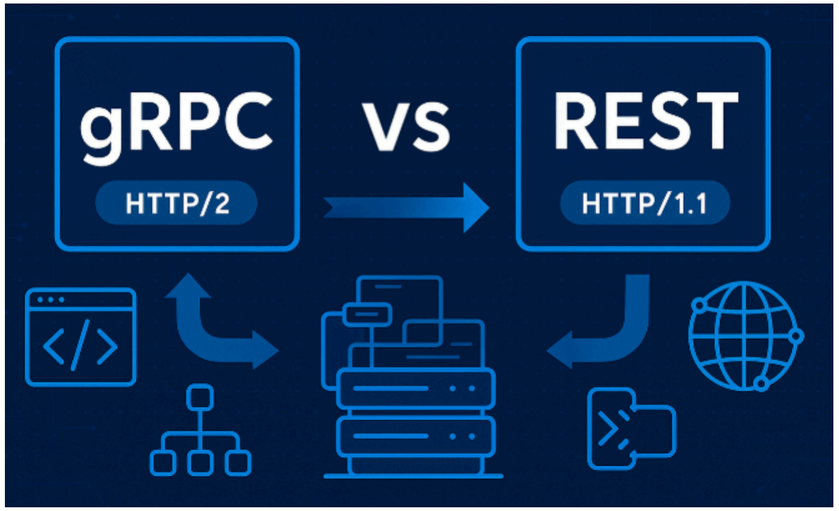
- **HTTP/2 + TLS 1.2/1.3** obligatorio en producción.
- **Certificados** X.509.
- Perfect Forward Secrecy.
- **Autenticación** basada en tokens, OAuth2, JWT.
- Control de acceso por **roles**.

gRPC vs REST

Característica	gRPC	REST
Protocolo	HTTP/2	HTTP/1.1
Formato	Protobuf (binario)	JSON (texto)
Rendimiento	Muy alto	Medio
Streaming	Bidireccional	Limitado (SSE /
Tipado	Estricto	Flexible
Facilidad	Menos intuitivo	Muy fácil
APIs públicas	No ideal	Perfecto
Comunicaciones internas	Excelente	Aceptable

Comparativa

- **gRPC** es superior para **microservicios, alto rendimiento y backend interno**.
- **REST** sigue siendo mejor para **APIs públicas, navegadores, compatibilidad universal**.



Conclusiones sobre gRPC

- gRPC es la **evolución moderna** para sistemas distribuidos.
- **Rápido**, **eficiente** y diseñado para microservicios.
- **Protobuf** y **HTTP/2** lo hacen extremadamente óptimo.
- **No** reemplaza a REST: ambos tienen su lugar.
- La combinación ideal en arquitecturas reales:

  REST externo + gRPC interno.

Gracias por la atención

¿Preguntas o comentarios?