~ Alejandro Gonzalez

“Hack & Learn - Oxidar.org”
~ Hernán Gonzalez

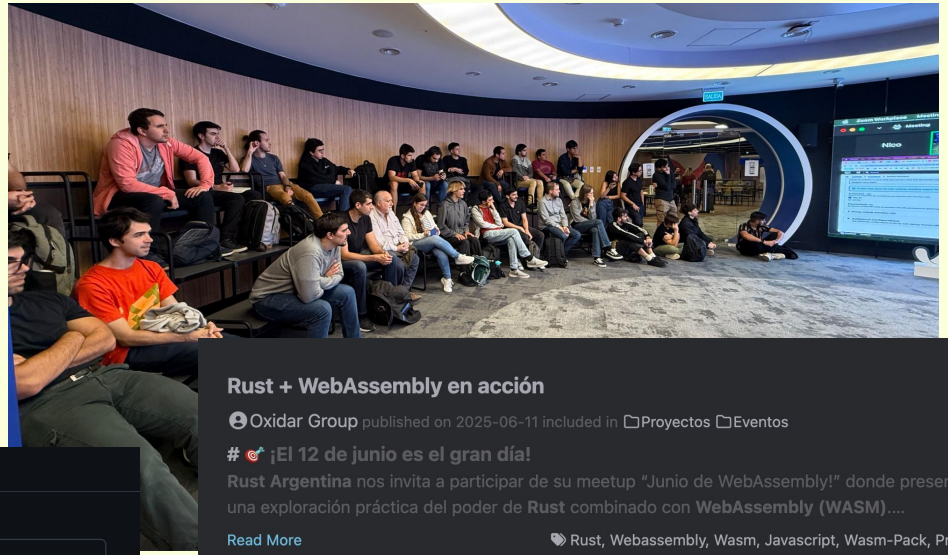
https://oxidar.org - 21/Aug/2025

OXIDAR.org

Espacio Latinoamericano para la divulgación del lenguaje de programación Rust. 🦀

Rustaceans

- Talleres auto-guiados
- Materiales de divulgación
- Repositorios públicos (MIT)
- Grupos de Colaboración



Overview | Repositories 5 | Projects | Packages | People

Pinned

oxidar-wasm Public
Forked from tkenda/rust-wasm
Rust ☆1

oxidar-lambdas Public
Rust

oxidar-url-shortener Public
A URL shortener using Axum and Shuttle.dev
Rust

oxidar-ls Public
Taller Oxidar - ls
Rust 1

Rust + WebAssembly en acción

Oxidar Group published on 2025-06-11 included in Proyectos Eventos

 # 🎯 ¡El 12 de junio es el gran día!

Rust Argentina nos invita a participar de su meetup "Junio de WebAssembly!" donde presentaremos una exploración práctica del poder de **Rust** combinado con **WebAssembly (WASM)**....

Read More — Rust, Webassembly, Wasm, Javascript, Wasm-Pack, Presentacion

Rust en AWS Lambda con Cargo Lambda

Oxidar Group published on 2025-04-03 included in Proyectos Tutoriales

 # 🚀 Una nueva colaboración de la comunidad

En **Oxidar** creemos en el poder del aprendizaje colaborativo. Por eso, nos complace compartir uno de nuestros proyectos más recientes: **oxidar-lambdas**, una exploración práctica sobre cómo desplegar...

Read More — Rust, Aws, Lambda, Cargo-Lambda, Serverless, Colaboracion

OXIDAR.org

Espacio Latinoamericano para la divulgación del lenguaje de programación Rust. 🦀

Sumate a la comunidad!

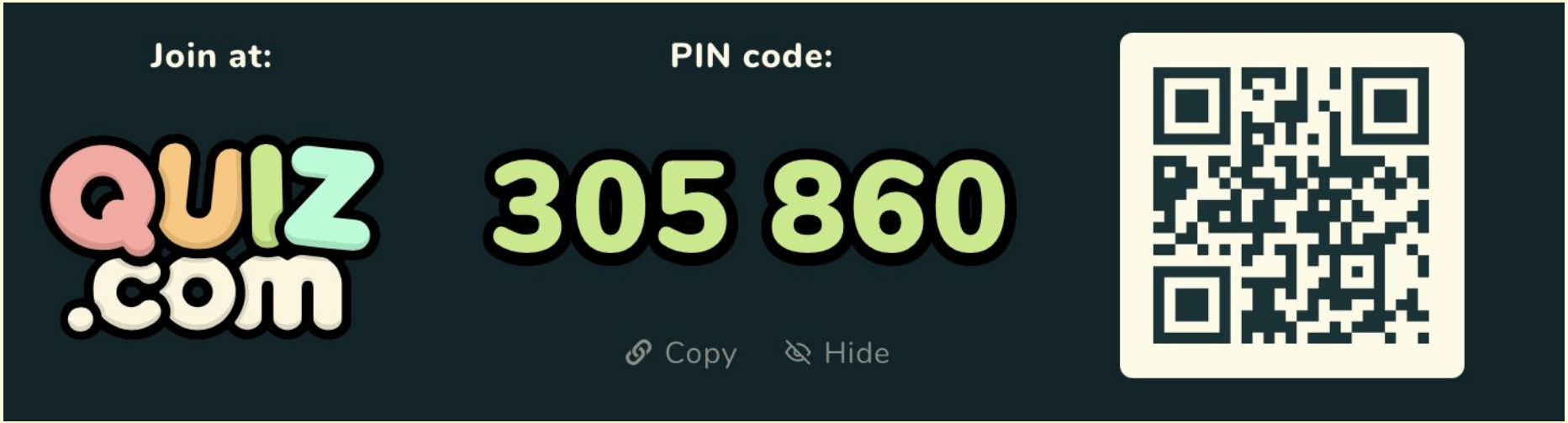
Oxidar.org

Telegram

GitHub

OXIDAR.org

¿Cuánto sabés de Rust?



About me …

Alejandro Gonzalez

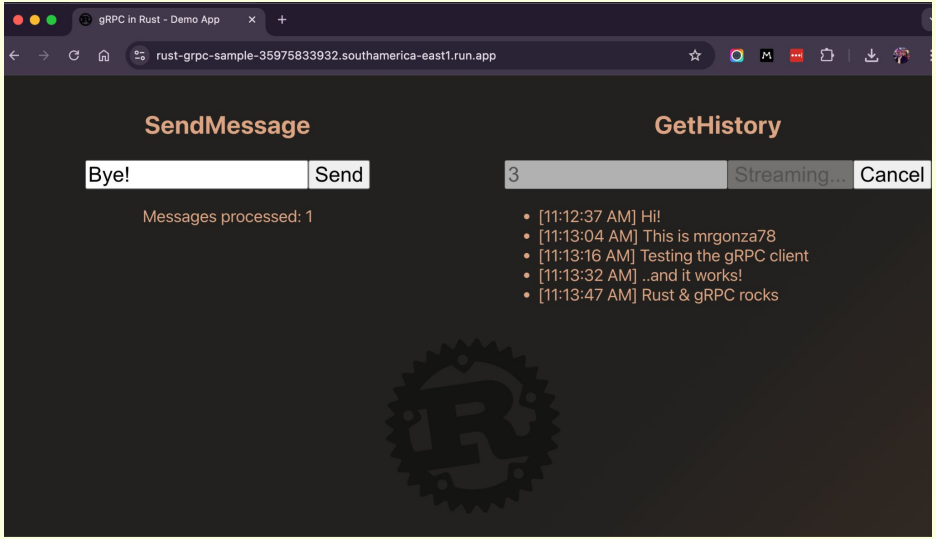
www.linkedin.com/in/aagonzalez

github.com/mrgonza78

Leave your questions here

gRPC Chat demo app

https://rust-grpc-sample-35975833932.southamerica-east1.run.app



Agenda

1. Protocol buffers

- What
- Why / Use cases
- Comparison
- Protoc
- Prost
- Cons / Alternatives / Tools

2. gRPC

- What / Pros / Cons
- Comparison
- Tonic
- Example
- Use cases

Protocol Buffers

https://protobuf.dev

What?

- Formats used for data interchange
- Language & platform agnostic
- Strongly typed schema

```
syntax = "proto3";

package my_package;

enum UserRole {
  GUEST = 0;
  MEMBER = 2;
  MODERATOR = 4;
  ADMIN = 6;
}

message User {
  int32 id = 1;
  string name = 2;
  optional string email = 3;
  bool verified = 4;
  repeated string interest = 5;
  map<string, string> links = 6;
  UserRole role = 7;
}
```

Protocol Buffers

Why ?

- Compact (faster to transmit)
- Fast to serialize/deserialize
- Schema evolution
  - you can add / deprecate fields without breaking existing clients or services
    - Decoder skips unknown fields
    - Missing field are decoded with a default value
    - Reserve tag # of deleted fields
    - Don’t re-use a tag #
    - Don’t change type of a field
    - Check https://protobuf.dev/best-practices/dos-donts

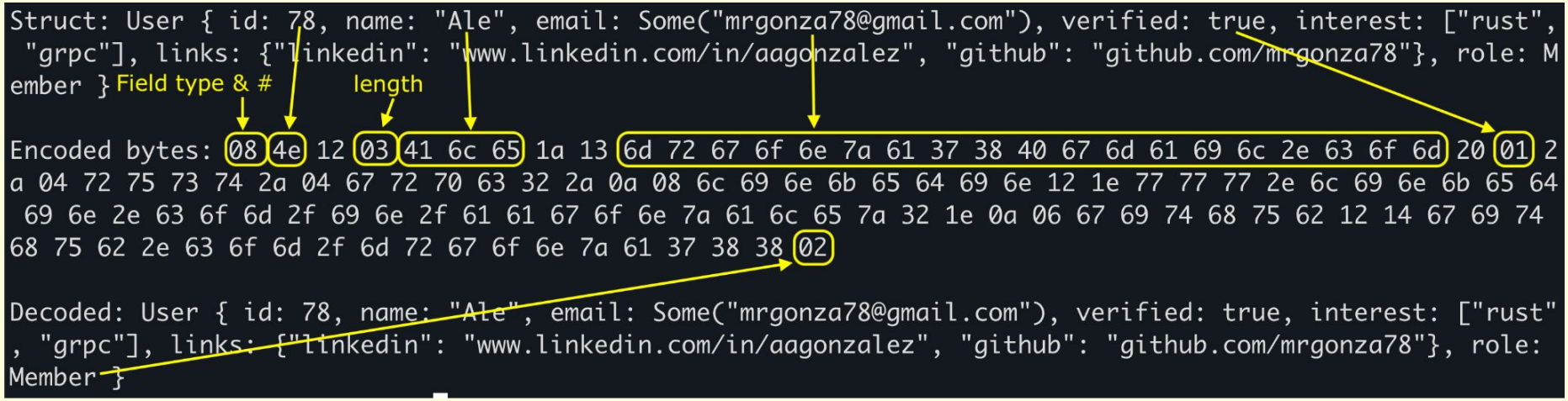
Use Cases

- Efficient data storage & transmission
- gRPC (communication between microservices)

Encoding / decoding

- Field name is not encoded
- Fields type and # is encoded
- Field ordering does not matter
- Variable-Length number encoding

```
syntax = "proto3";

package my_package;

enum UserRole {
  GUEST = 0;
  MEMBER = 2;
  MODERATOR = 4;
  ADMIN = 6;
}

message User {
  int32 id = 1;
  string name = 2;
  optional string email = 3;
  bool verified = 4;
  repeated string interest = 5;
  map<string, string> links = 6;
  UserRole role = 7;
}
```



Comparison

vs JSON

Encoding

- String: ~2.7 times faster
- Numbers: ~34 times faster

Decoding

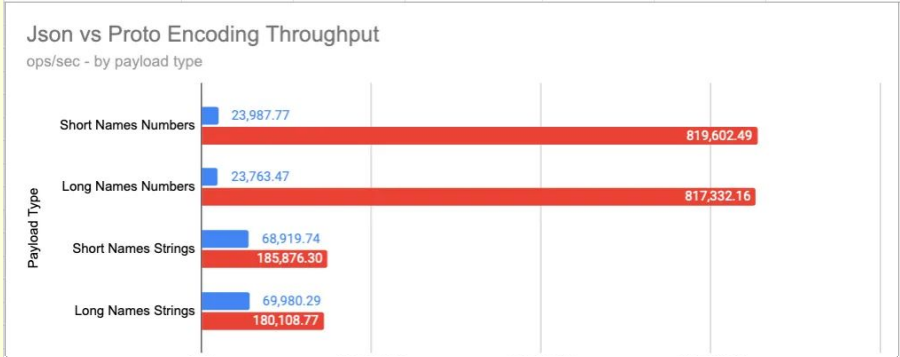
- String: ~23 times faster
- Number: ~38 times faster

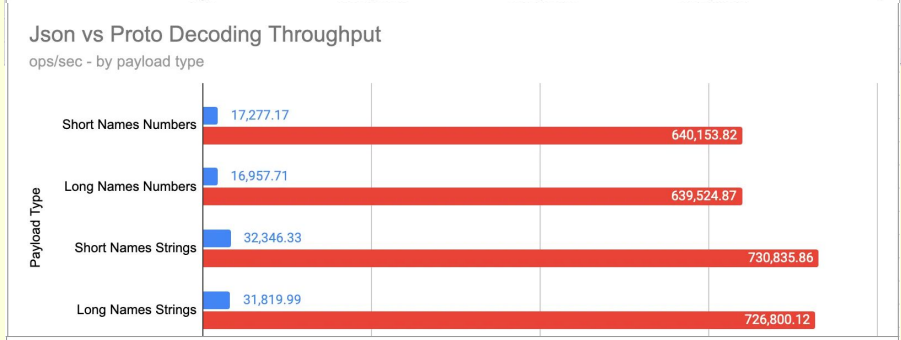
Payload size

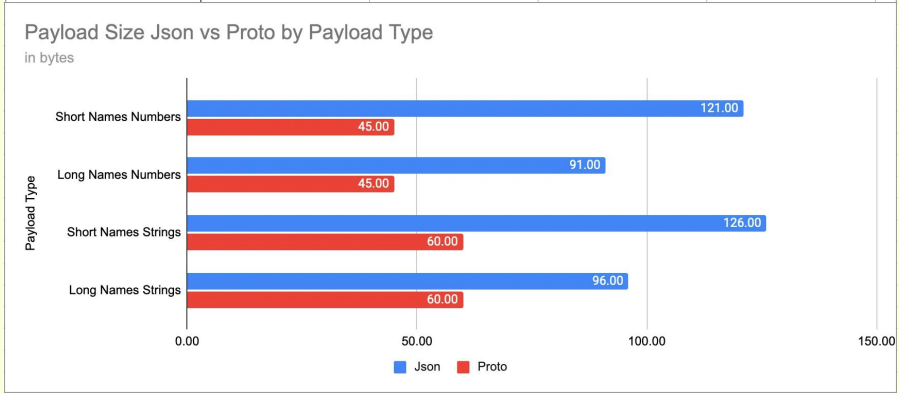
- Between 30% and 70% smaller

Source

- Json vs protocol buffers
- A comparison of serialization formats



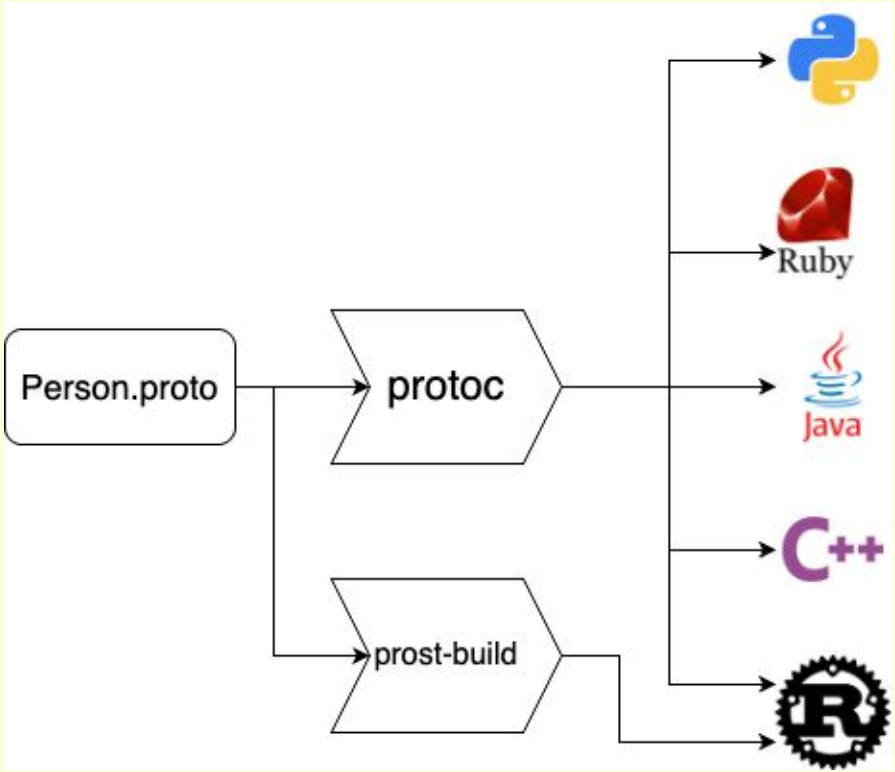




Protoc compiler

the official command-line tool that transforms .proto files into source code

- Supports many languages out of the box
- Can be extended to support new languages via plugins (eg go)



Rust: Prost

https://github.com/tokio-rs/prost

Protocol Buffers implementation for Rust. It generates **simple**, **idiomatic** Rust code from .proto files

```
syntax = "proto3";

package my_package;

enum UserRole {
  GUEST = 0;
  MEMBER = 2;
  MODERATOR = 4;
  ADMIN = 6;
}

message User {
  int32 id = 1;
  string name = 2;
  optional string email = 3;
  bool verified = 4;
  repeated string interest = 5;
  map<string, string> links = 6;
  UserRole role = 7;
}
```

```
// This file is @generated by prost-build.
#[derive(Clone, PartialEq, ::prost::Message)]
6 implementations
pub struct User {
    #[prost(int32, tag = "1")]
    pub id: i32,
    #[prost(string, tag = "2")]
    pub name: ::prost::alloc::string::String,
    #[prost(string, optional, tag = "3")]
    pub email: ::core::option::Option<::prost::alloc::string::String>,
    #[prost(bool, tag = "4")]
    pub verified: bool,
    #[prost(string, repeated, tag = "5")]
    pub interest: ::prost::alloc::vec::Vec<::prost::alloc::string::String>,
    #[prost(map = "string, string", tag = "6")]
    pub links: ::std::collections::HashMap<
        ::prost::alloc::string::String,
        ::prost::alloc::string::String,
    >,
    #[prost(enumeration = "UserRole", tag = "7")]
    pub role: i32,
}
#[derive(Clone, Copy, Debug, PartialEq, Eq, Hash, PartialOrd, Ord, ::prost::E
#[repr(i32)]
12 implementations
pub enum UserRole {
    Guest = 0,
    Member = 2,
    Moderator = 4,
    Admin = 6,
}
```

```
let p = my_package::User {
    id: 78,
    name: "Ale".to_string(),
    email: Some("mrgonza78@gmail.com".to_string()),
    verified: true,
    interest: vec!["rust".to_string(),"grpc".to_string()],
    links: vec![
        ("github".to_string(), "github.com/mrgonza78".to_st
        ("linkedin".to_string(), "www.linkedin.com/in/aagon
    ].into_iter().collect(),
    role: my_package::UserRole::Member.into(),
};
```

☝ github.com/oxidar-org/rust-proto-sample

Protocol Buffers

Cons

- No human-readable
- Less flexible for ad-hoc data
- Limited built-in data types (eg dates/timestamps)

Alternatives

- Thrift. From Facebook, supports code generation, schemas, multiple protocols (binary, compact, JSON), and built-in RPC
- Flatbuffers. Designed by Google; supports zero-copy deserialization (very fast), good for game dev and mobile. Schema-based

Tools

- Protobufpal. online tool used for serialization, deserialization, converting to json, and validation of Protocol Buffers messages

gRPC

https://grpc.io/

What?

high performance, open source RPC framework

- Based on **HTTP/2**
- Uses **Protocol Buffers**
- **Strongly-typed APIs** with automatic client/server stub **code generation** for multiple languages
- Offers **bi-directional streaming** and **asynchronous** communication.
- Supports **authentication**, **load balancing**, **pluggable retries**, etc

Pros

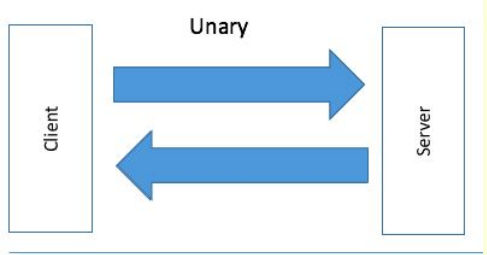
- High performance and low latency.
- Contract-first API design (Protobuf).
- Language and platform agnostic

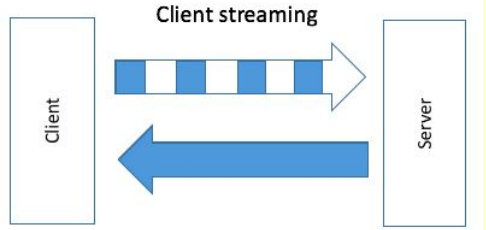
Cons

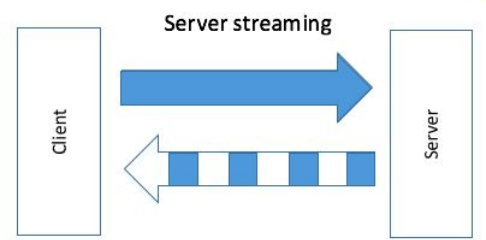
- No human-readable (b/c Protobuf)
- More complex to debug than REST/JSON
- Requires tooling support (for code generation and protocol decoding)

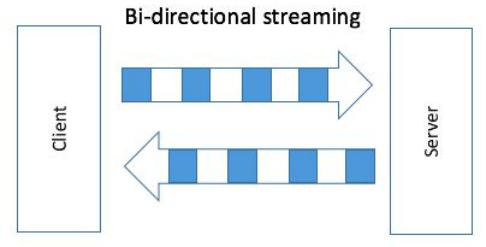
Service Definition

```proto
package chat_package;

service ChatService {
  // Unary function
  rpc SendMessage (Message) returns (SendMessageResponse);
  // Client streaming
  rpc SendBulkMessages (stream Message) returns (SendMessageResponse);
  // Server streaming
  rpc GetHistory(HistoryRequest) returns (stream Message);
  // Bidirectional streaming
  rpc LiveChat(stream Message) returns (stream Message);
}

message Message {
  string content = 1;
}

message SendMessageResponse {
  int32 messages_processed = 1;
}

message HistoryRequest {
  int32 starting_at = 1;
}
```

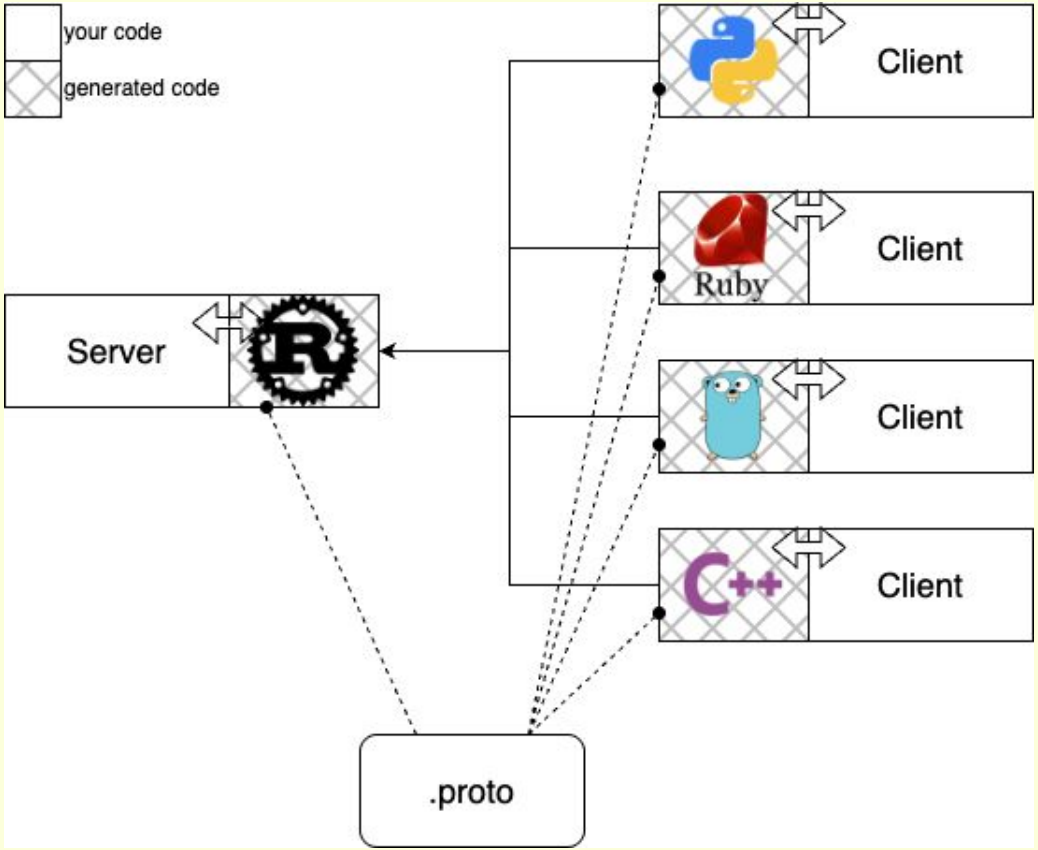








gRPC


your code
generated code
Client
Client
Ruby
Server
Client
Client
.proto

Rust: TONIC

https://github.com/hyperium/tonic

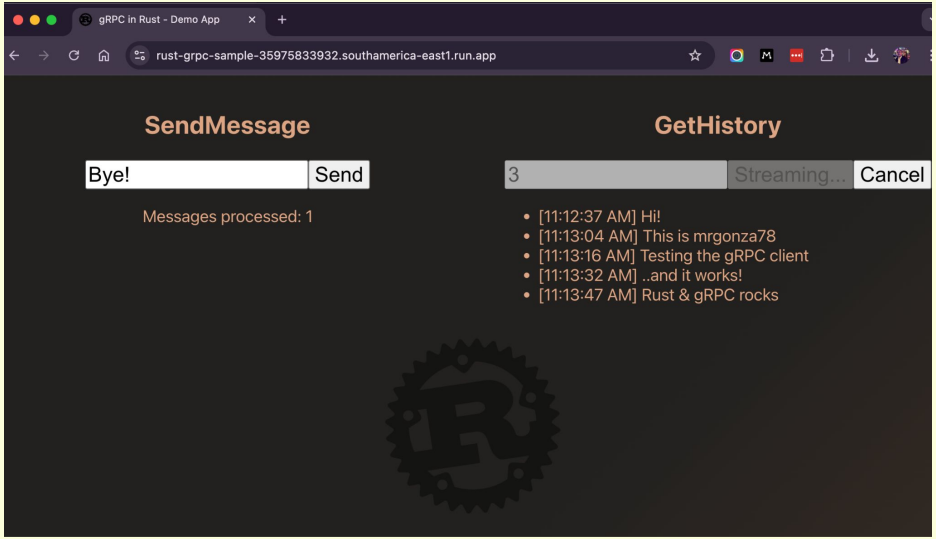
- generic gRPC implementation
- HTTP/2 based on hyper
- codegen powered by prost

TONIC Generated code

👇see in github.com/oxidar-org/rust-grpc-sample

TONIC Chat Demo App

https://rust-grpc-sample-35975833932.southamerica-east1.run.app



Use cases

Microservices

Backend composed of multiple microservices needs to talk to each other

IoT

IoT device communicates with a backend that requires low latency and low bandwidth

Envoy

HTTP/network proxy. Extensible/customizable via gRPC services.
Eg: ext_authz gRPC API for external authorization.

Google Cloud

gRPC is the default protocol for many GCP APIs.
Eg: BigQuery for streaming query results

Etcd

The backing store for Kubernetes uses gRPC for its API.
Eg: KV.Put, KV.Get .. see docs

CockroachDB

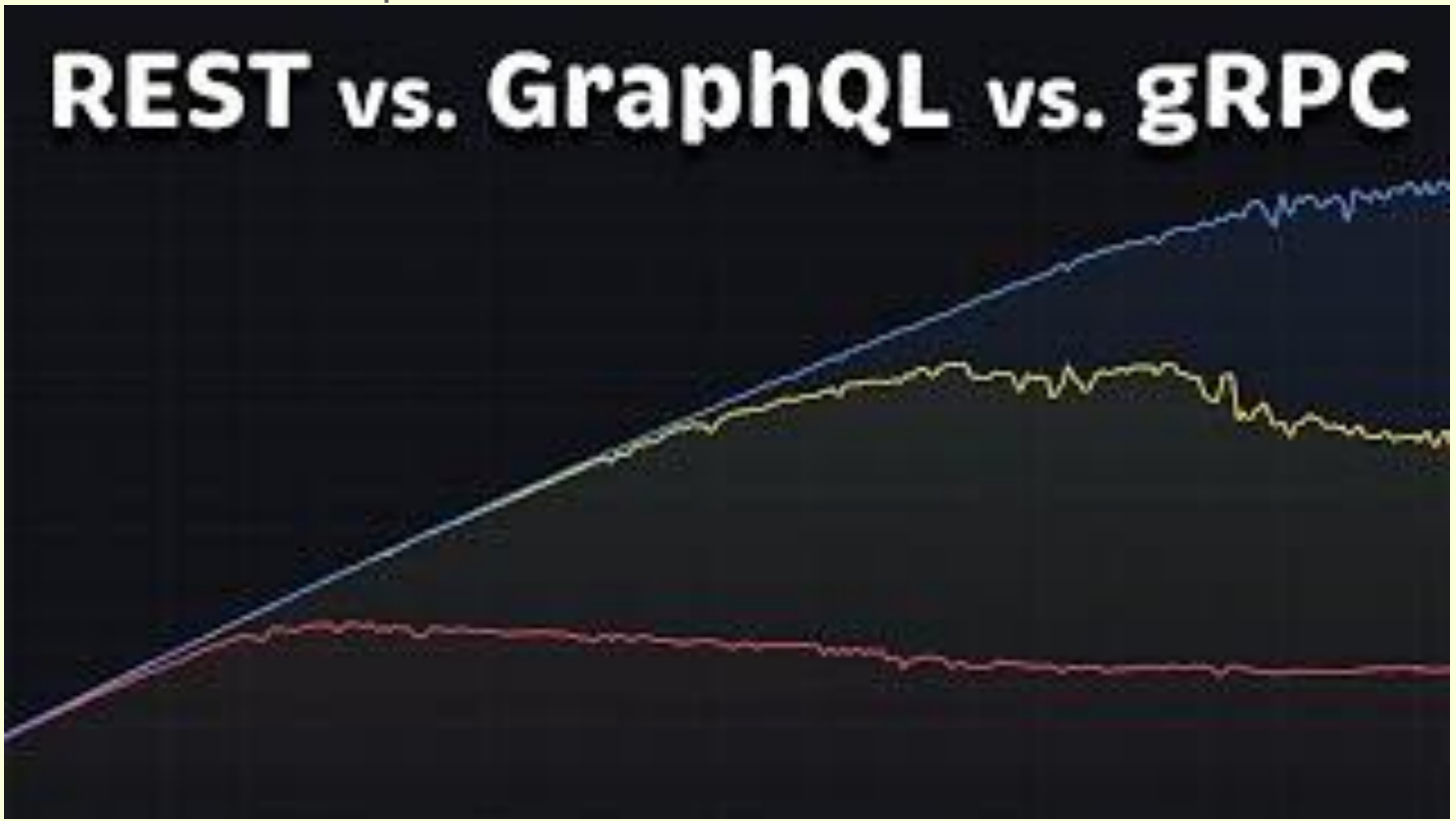
SQL and admin over gRPC

Tools

grpcurl: Like cURL, but for gRPC
postman: platform/tool for testing APIs (supports gRPC)
Awesome-grpc: A curated list of useful resources for gRPC

Comparison

vs REST & GraphQL



MUCHAS GRACIAS!

Déjanos tus PRs y comentarios.

github.com/oxidar-org