

ozone  
tech



# Protocol Buffers. gRPC

Иван Стрелков  
Разработчик информационных систем

# О чём сегодня пойдет речь?

1. Protocol Buffers
2. gRPC
3. grpc-gateway



# Protocol Buffers



Главные недостатки JSON  
– большой размер данных  
и медленный парсинг

# Что такое Protocol Buffers

1. (эффективный) бинарный формат
2. для (де)кодирования нужно знать схему
3. парсеры/сериализаторы генерируются **для многих языков**
4. фокус на обратную совместимость

# Пример

```
syntax = "proto3";  
  
package golang.school.demo;  
  
message SearchRequest {  
    string query = 1;  
    int32 page_number = 2;  
    int32 results_per_page = 3;  
}
```

# Типы данных - скалярные

```
message TestMessage {  
    bool some_bool = 1;  
    uint8 some_uint8 = 2;  
    double some_double = 3;  
    string some_string = 4;  
    bytes binary_data = 5;  
}
```

# Сообщение может быть полем

```
message InnerMessage {  
    string name = 1;  
}
```

```
message Message {  
    InnerMessage inner = 1;  
}
```

# Типы данных – repeated

```
message Comment {  
    string text = 1;  
}
```

```
message TestMessage {  
    repeated Comment comments = 1;  
    repeated int64 ids = 2;  
}
```

# Типы данных - map

```
message SomeMessage {  
    string text = 1;  
}
```

```
message MessageWithMap {  
    map<string, SomeMessage> my_map = 1;  
}
```

# enum

```
enum Status {  
    UNKNOWN = 0;  
    ENABLED = 1;  
    DISABLED = 2;  
}
```

```
message Message {  
    Status status = 1;  
}
```

# oneof

```
message NumericValue {
    string name = 1;
    oneof test_oneof {
        int64 int = 2;
        double float = 3;
        FormattedFloat formatted = 4;
    }
}
```

# Вложенность

```
message NestedDemo {  
    message NestedMessage {  
        enum NestedEnum {  
            UNKNOWN = 0;  
        }  
        NestedEnum value = 1;  
    }  
    NestedMessage result = 1  
}
```

# Вложенность

```
message NestedDemo {  
    message NestedMessage {  
        enum NestedEnum {  
            UNKNOWN = 0;  
        }  
        NestedEnum value = 1;  
    }  
    NestedMessage.NestedEnum result = 1  
}
```



Демо



В protobuf невозможно  
отличить значение по  
умолчанию от отсутствия  
поля

# wrappers

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

message ParamValue {
    oneof value_oneof {
        google.protobuf.StringValue str = 1;
        google.protobuf.DoubleValue double = 2;
        google.protobuf.Int64Value int = 3;
        google.protobuf.BoolValue bool = 4;
    }
}
```

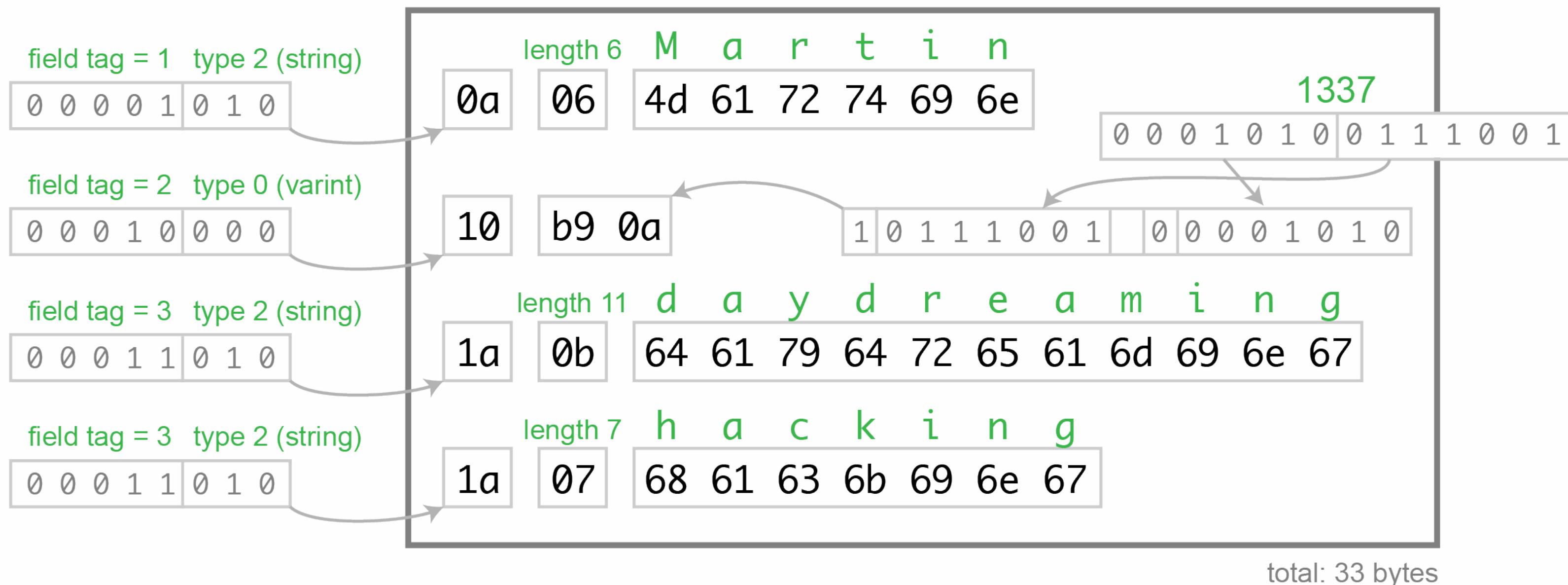
# timestamp

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

message Message {
    string text = 1;
    google.protobuf.Timestamp created_at = 2;
    google.protobuf.Timestamp updated_at = 3;
}
```

# Как хранятся сообщения

## Protocol Buffers



<https://martin.kleppmann.com/2012/12/05/schema-evolution-in-avro-protocol-buffers-thrift.html>

# Как хранятся сообщения

Type	Meaning	Used For
0	Varint	int32, int64, uint32, uint64, sint32, sint64, bool, enum
1	64-bit	fixed64, sfixed64, double
2	Length-delimited	string, bytes, embedded messages, packed repeated fields
3	Start group	groups (deprecated)
4	End group	groups (deprecated)
5	32-bit	fixed32, sfixed32, float

<https://developers.google.com/protocol-buffers/docs/encoding#structure>

# Обратная совместимость

1. нельзя менять номер существующего поля
2. можно добавлять/убирать поля
3. нельзя добавлять новое поле на место старого номера! (reserved)

# reserved

```
syntax = "proto3";

message SearchRequest {
    string query = 1;
    int32 page_number = 2;
    int32 result_per_page = 3;
}
```

# reserved

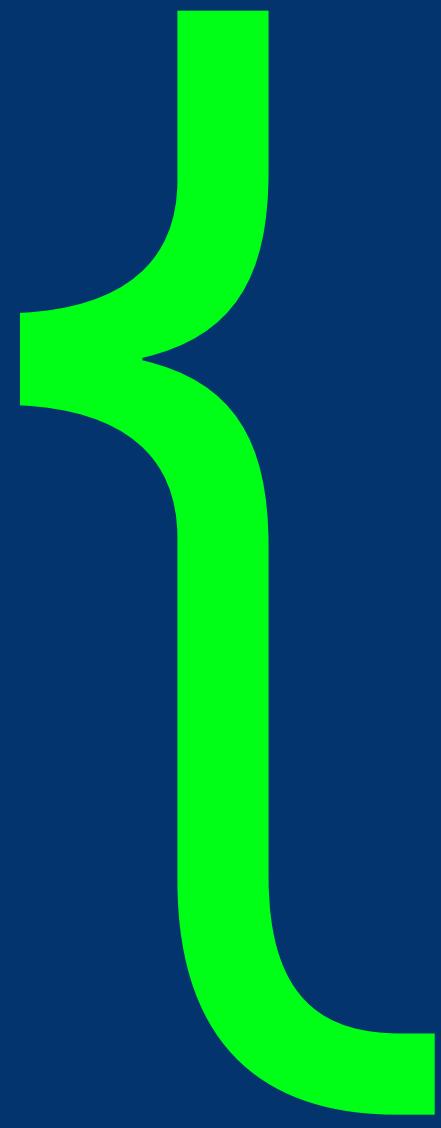
```
syntax = "proto3";  
  
message SearchRequest {  
    string query = 1;  
    int32 page_number = 2;  
    reserved 3;  
}
```

# Обратная совместимость

4. `int32`, `uint32`, `int64`, `uint64` и `bool` – взаимозаменяемы (преобразование по правилам C++)
5. `string` и `bytes` взаимозаменяемы (если передается валидная UTF-8 строка)
6. сообщение можно заменить `bytes` с сериализованным представлением

# Обратная совместимость

7. repeated можно добавить/убрать у string, bytes и сообщений
8. int32, uint32, int64, uint64 взаимозаменяемы с enum
9. можно превратить поле/ряд полей новым oneof



gRPC

# gRPC

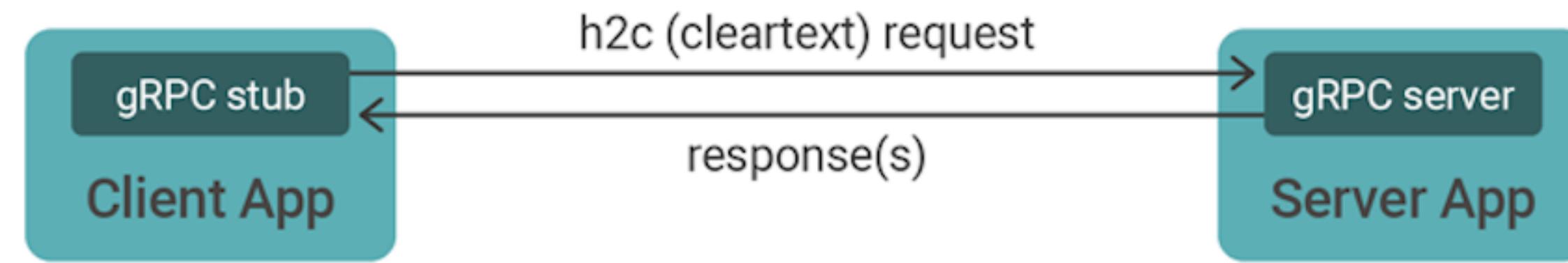
## 1. RPC на базе Protocol Buffers

# Remote Procedure Call

*From Wikipedia, the free encyclopedia*

In distributed computing, a **remote procedure call (RPC)** is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

# gRPC



# gRPC

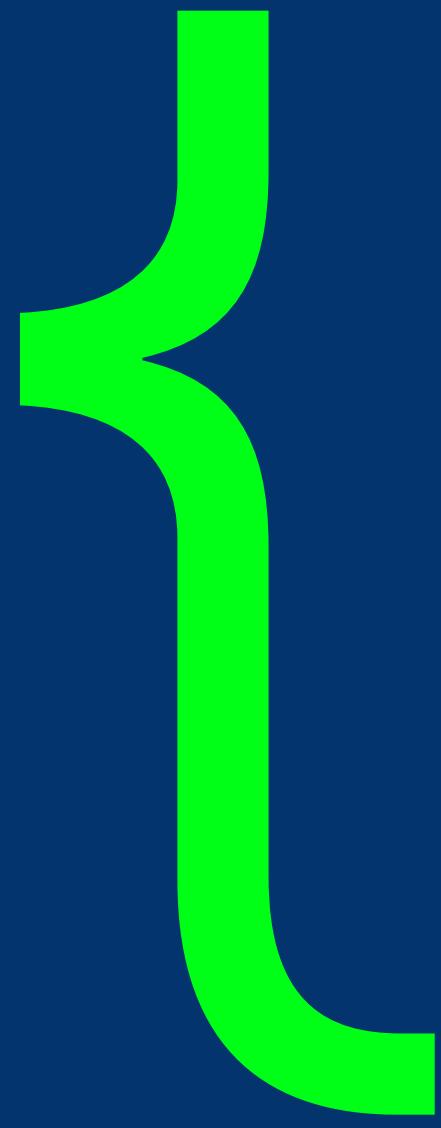
1. RPC на базе Protocol Buffers
2. сообщения пересыпаются по HTTP/2

# gRPC – определение схемы

```
service EntityService {  
    rpc CreateEntity (CreateEntityRequest) returns (CreateEntityResponse);  
}
```

# gRPC – определение метода

```
func (i *Implementation) CreateEntity(  
    ctx context.Context,  
    req *desc.CreateEntityRequest,  
) (*desc.CreateEntityResponse, error) {  
    return &desc.CreateEntityResponse{  
        // ...  
    }, nil  
}
```



Демо

# gRPC – возможные варианты

```
service HelloService {  
    rpc SayHello (HelloRequest) returns (HelloResponse);  
  
    rpc LotsOfReplies (HelloRequest) returns (stream HelloResponse);  
  
    rpc LotsOfGreetings (stream HelloRequest) returns (HelloResponse);  
  
    rpc BidiHello (stream HelloRequest) returns (stream HelloResponse);  
}
```

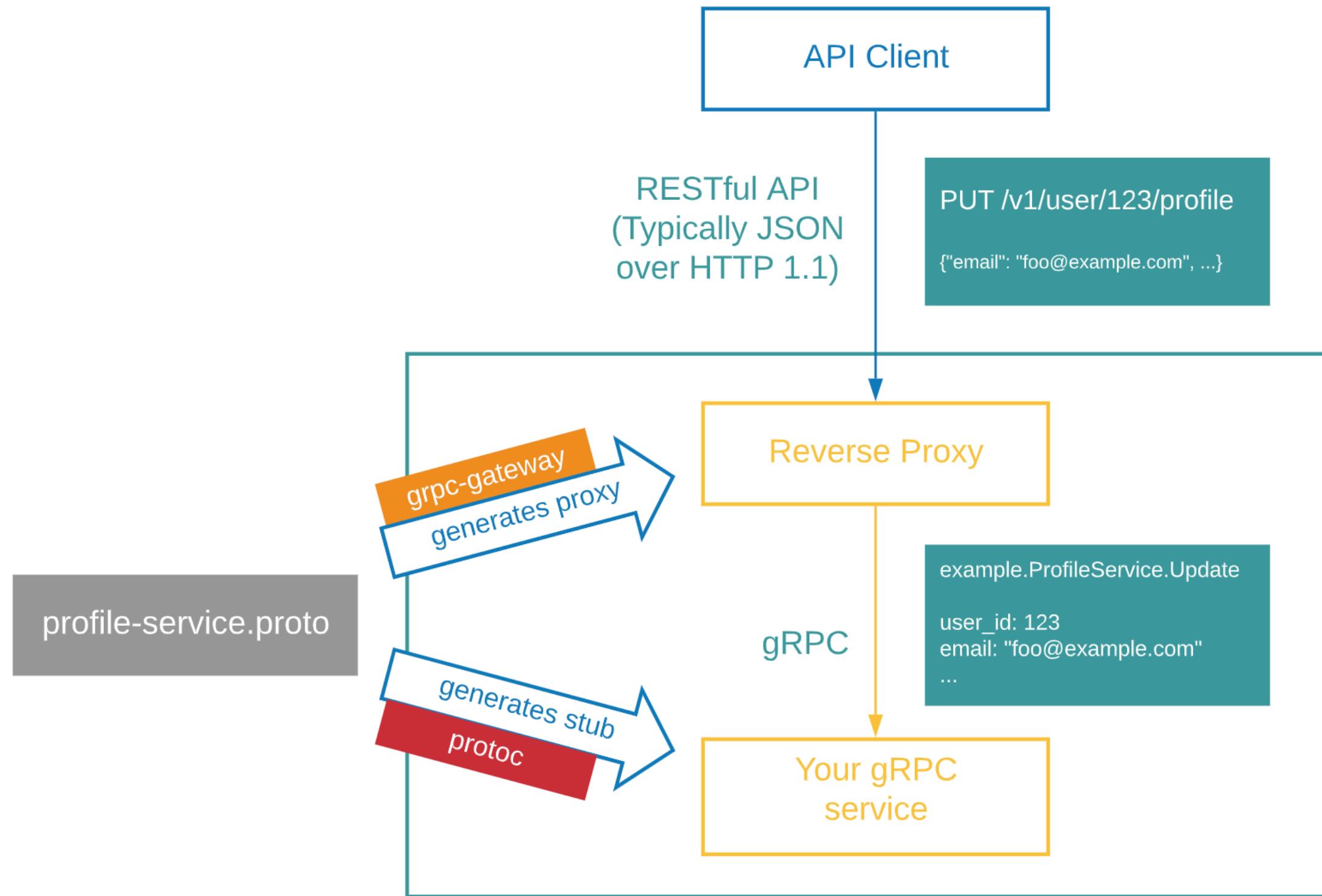


grpc-gateway

# grpc-gateway

1. Генерирует HTTP-ручки для gRPC методов
2. Может генерировать swagger

# grpc-gateway



# Что почитать

1. Language Guide -

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

2. Encoding - <https://developers.google.com/protocol-buffers/docs/encoding>



Спасибо за внимание!

Иван Стрелков



ivstrelkov@ozon.ru