gRPC

Dekan KILIÇ

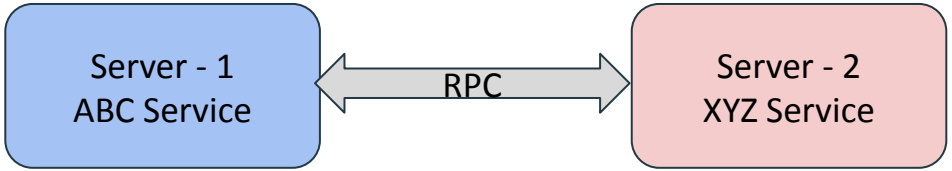
RPC Frameworks

- Java RMI
- gRPC
- Apache Avro
- Apache Thrift

Dekan KILIÇ

What is RPC

Remote Procedure Call is a protocol that one program can use to request a service from a program located on another computer in a network. RPC is used for client-server based applications, where multiple devices need to communicate with each other.



Dekan KILIÇ

Features of RPC

- **Synchronous Communication:** Typically, the client sends a request to the server and waits for a response making RPCs synchronous in nature.

- **Data Serialization:** Data sent over the network needs to be serialized (converted into a stream of bytes) and then deserialized on the receiving end.

What is gRPC

- Open-source framework developed by Google for building high-performance, scalable, and efficient distributed systems
- Modern and Lightweight communication protocol
- Based on Remote Procedure Call model and uses protocol buffers data serialization format to define the structure of messages and service interfaces

Dekan KILIÇ

Why gRPC

Microservices: gRPC is designed for low latency and high throughput communication. gRPC is great for lightweight microservices where efficiency is critical.

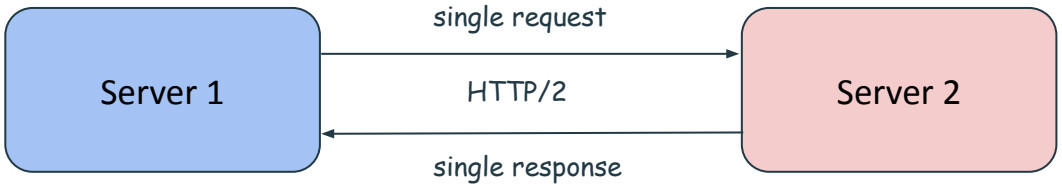
Point-to-point real-time communication: gRPC has excellent support for bi-directional streaming. gRPC services can push messages in real-time without polling.

Network constrained environments: gRPC messages are serialized with Protobuf, a lightweight message format. A gRPC message is always smaller than an equivalent JSON message.

Communication Patterns

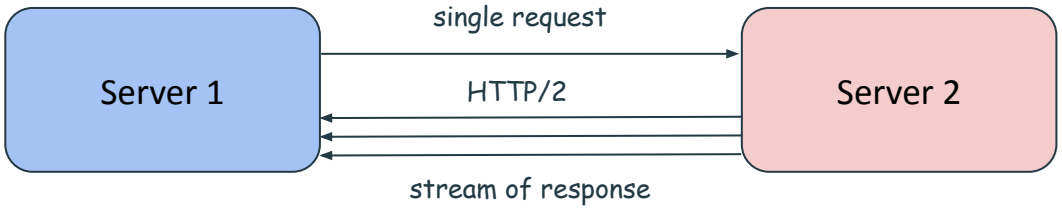
Unary RPC

The most basic pattern used by gRPC is the Unary RPC, which allows a client to send a single request to a server and receive a single response in return.



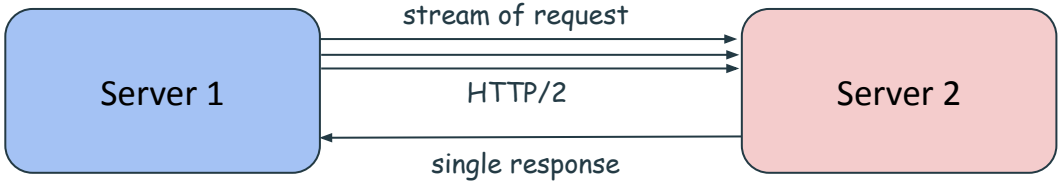
Server Streaming RPC

Server Streaming RPC allows a client to send a request to a server and receive a bunch of streams of responses in return. This is useful for cases where the server needs to send a bunch of data to the client.



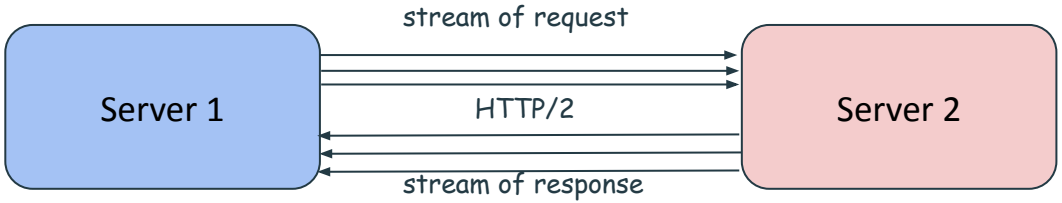
Client Streaming RPC

Client Streaming RPC allows a client to send a stream of requests to a server and receive a single response. This pattern is useful if the client needs to send a stream of data to the server.



Bidirectional Streaming RPC

Bidirectional RPC allows a client to send a stream of requests to a server and receive a stream of responses. This is useful if the client needs to send to a server and receive a bunch of data from server.



Service Definition

```
service DiscountService {
  rpc getDiscount(DiscountRequest) returns (DiscountResponse);
}

message DiscountRequest {
  string code = 1;
  float price = 2;
  int64 externalCategoryId = 3;
}

message DiscountResponse {
  string code = 1;
  float newPrice = 2;
  float oldPrice = 3;
  Response response = 5;
}

message Response {
  bool statusCode = 1
  string message = 2;
}
```

Unary RPC Service Definition

Unary RPCs where the client sends a single request to the server and gets a single response back

```
service DiscountService {
  rpc getDiscount(DiscountRequest) returns (DiscountResponse);
}
```

Server Streaming RPC Service Definition

Server streaming RPCs where the client sends a request to the server and gets a stream to read a sequence of messages back. The client reads from the returned stream until there are no more messages.

```
service DiscountService {
  rpc getDiscount(DiscountRequest) returns (stream DiscountResponse);
}
```

Client Streaming RPC Service Definition

```
service DiscountService {
  rpc getDiscount(stream DiscountRequest) returns (DiscountResponse);
}
```

Bi-directional RPC Service Definition

```
service DiscountService {
  rpc getDiscount(stream DiscountRequest) returns (stream DiscountResponse);
}
```

Protocol Buffers

Protocol Buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

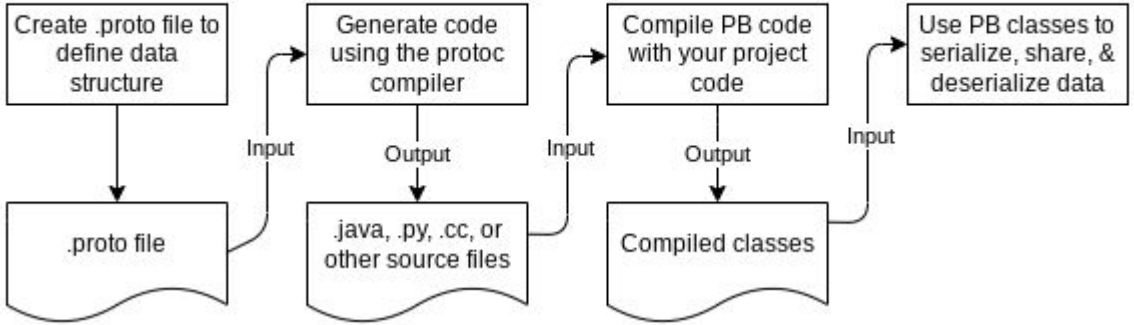
Dekan KILIÇ

Working with Protocol Buffers

- Protocol Buffers is open source mechanism for serializing structured data provided by Google
- Define your data structure in a proto file. This data is structured as messages
- Uses protocol buffer compiler, protoc, to generate data access classes in preferred language
- You can also define your services in proto file to expose them to user

How Protocol Buffer works

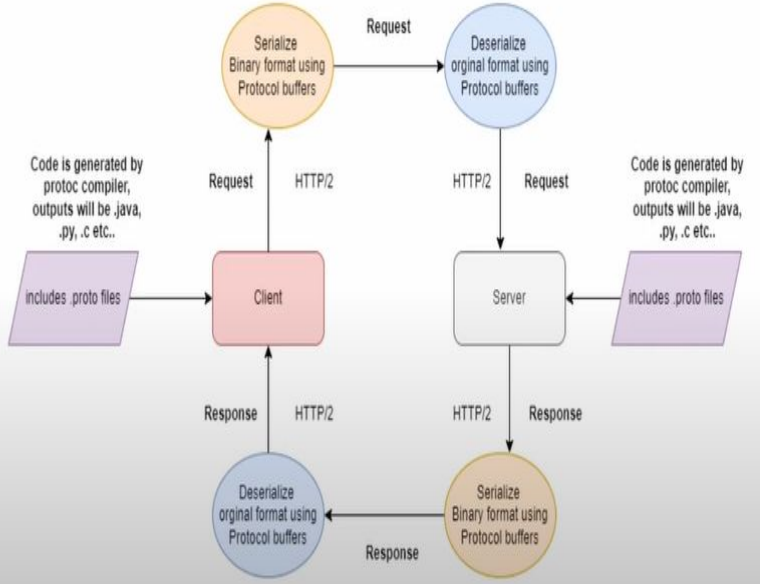
The following diagram shows how you use protocol buffers to work with your data.



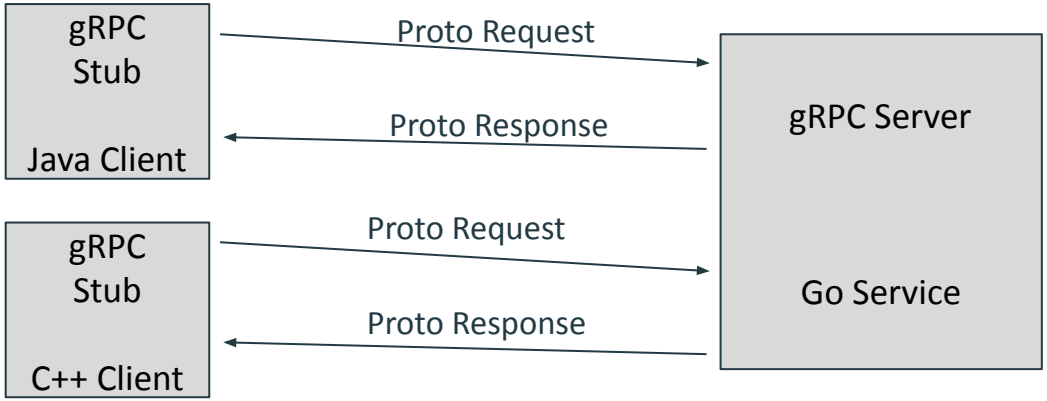
Dekan KILIÇ

How gRPC works

1-) Defining the service interface
2-) Generating code
3-) Client makes a request
4-) Serialization
5-) Network transport
6-) The server receives the request
7-) The server processes the request
8-) The server sends a response
9-) Network transport
10-) The client receives the response
11-) Client processes the response



gRPC Client-Server Example



Dekan KILIÇ

Message

scalar types including various data types like double, int32, etc…

```
message LoginRequest {
    string username = 1;
    string password = 2;
}
```

unique field number used to identify field within message binary format

Dekan KILIÇ

Message Advanced Example

```
message CreateProductRequest{
        reserved 2
        reserved "price";
        required string title = 1;
        repeated Recommendation = 3;     // There can be 0 to n recommendation for this product

        enum Color {
                BLUE = 1;
                RED = 2;
                GREEN = 3;
        }
        optional Color color = 4 [ default = RED];
}

message Recommendation {
        int32 product_id = 1;
        string product_url = 2;
        string title = 3;
        string image_url = 4;
}
```

Dekan KILIÇ