GRPC (remote procedure calls)

Overview

What is gRPC??

- Grpc is a modern, open source remote procedure call(RPC) framework.
- gRPC is based on the idea of services.

gRPC Motivation

- Google has an internal RPC system called stuby.
- API for 10+ languages.

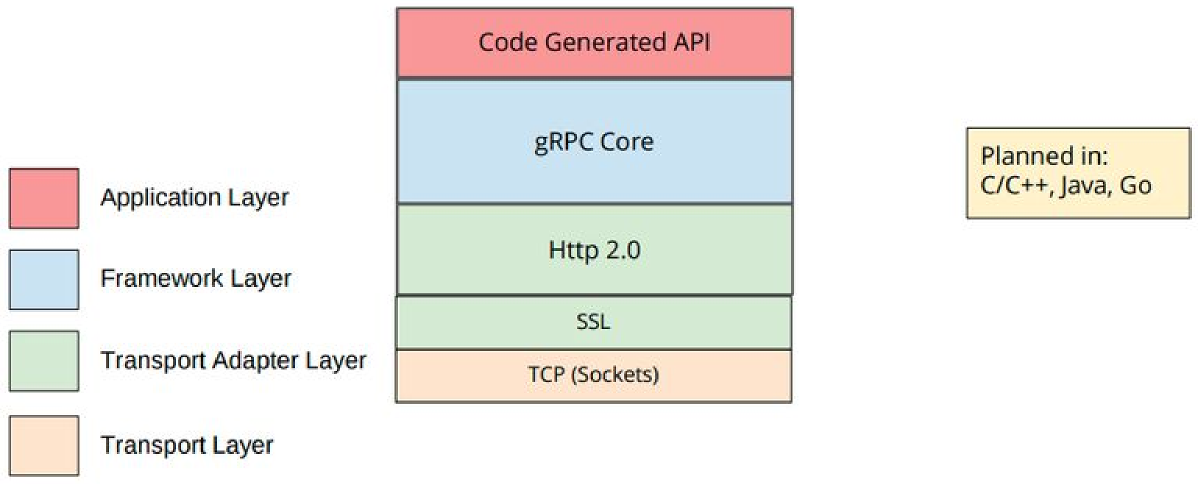
Why gRPC?

- Simple and Multi language, multi platform framework.
- Platform supported: Linux, Android, ios, MacOS, windows.
- Free and open source.
- Transport over HTTT2 + TLS.
- Efficient use of a single TCP connection over concurrent frames.
- Supports bidirectional streaming.
- Supports 10^10 RPC's per seconds.

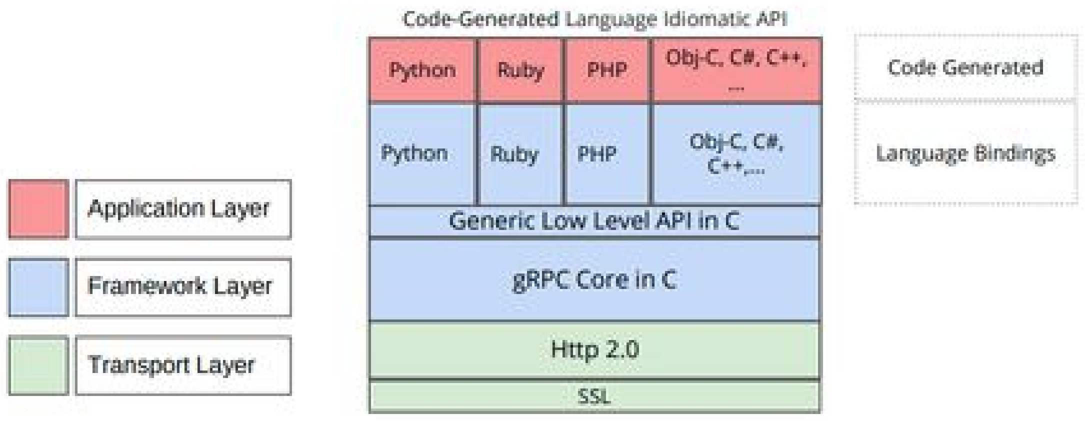
Architecture of GRPC

- Basically Implemented in three stacks : C , JAVA , GO
- Other language implementations warp C runtime libraries.
- Wrapping C runtime gives performance in different languages.

Native Implementation

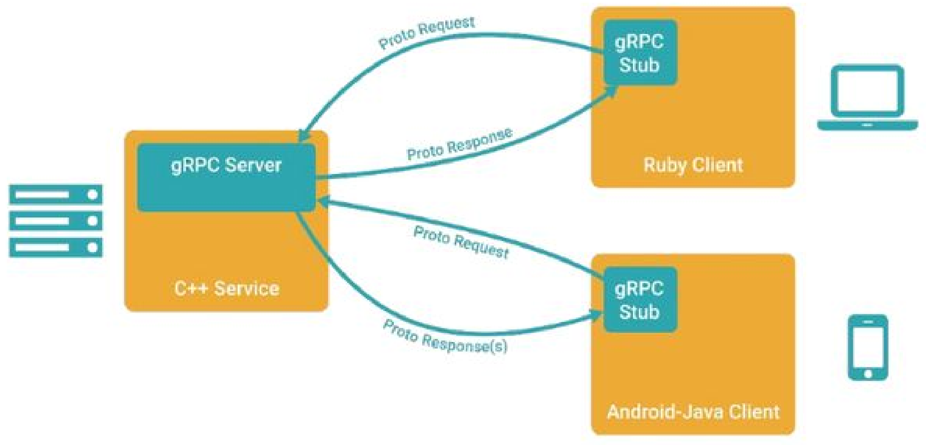


Derived Stack



\

- **Service definition**

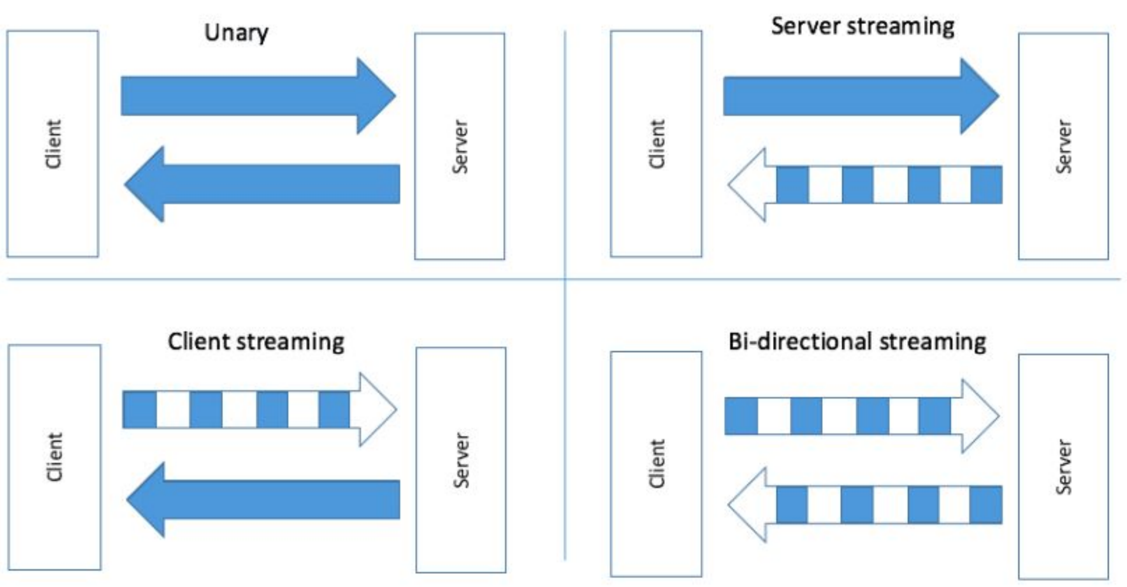


```
service HelloService
{
    rpc SayHello (HelloRequest) returns (HelloResponse);
}

message HelloRequest
{
            string request = 1;
}

message HelloResponse
 {
            string reply = 1;
}
```

  - gRPC provides 4 types of services
    - Unary RPC
    - Server Streaming RPC
    - Client streaming RPC
    - Bidirectional RPC

- **Using API**
  - Starting from service in a .proto file , gRPC provides a protocol buffer compiler plugin that generates client and server code.
  - gRPC users call this API from the client side and implement corresponding API on server side.
    - Server side :

        - Server implements the method declared by service and runs a gRPC server to handle client requests.
        - gRPC infrastructure decodes incoming requests, executes service methods, and encodes service responses.
    - Client side :
        - Client has a local object we called them stub that implements the same method as service.
        - Client call these methods into local object.
- **RPC Life cycle**



1) **Unary RPC**

   Clients send a single request and get a single response.
   - Once the client calls stub method, the server is notified that RPC is invoked with clients metadata, method name and deadline.
   - Once the server has a client request message. It does whatever work is necessary and creates response. The response is returned to the client with status code and optional message.
   - If the status is OK then the client gets a response.

2) **Server streaming RPC**
   - This is the same as unary RPC, except that the server returns a stream of messages in response to client requests.
   - After sending all messages server status details and metadata are sent to the client.
   - Then client complete it’s procedure

3) **Client streaming RPC**

- This is the same as unary RPC, except the client sends a stream of messages to the server.
- Server responds with a single message and status code.
- Then the client completes it's procedure.

4) Bidirectional streaming RPC

- In bidirectional streaming RPC, call is initiated by client invoking method and server receiving client metadata, method name and deadline.
- This streams are independent hence client and server can read and write messages in any order.

5) Deadline / Timeouts

- gRPC allows clients to specify how long they are willing to wait for an RPC.
- After that RPC is terminated with `DEADLINE_EXCEEDED` error.
- On the server side, the server can query to see if a particular RPC is timeout or how much is left.

6) RPC termination

- Both client and server make independent determination of RPC termination.

7) Cancellation of RPC

- Either the client or the server can cancel an RPC at any time.
- A cancellation terminates RPC immediately.

8) Metadata

- Metadata is information about a particular RPC call in the form of list of pair of key-value pairs, where they key and values are typically strings but can be binary.
- Access to metadata is language dependent.

9) Channels

- A gRPC channel provides a connection to a gRPC server on specified host and port. It is used when creating client stub.
- A channel has a state, including *connected, idl.*
- Closing a channel is dependent on the language.

Authentication

- An overview of gRPC authentication, including built-in authentication and plug in our own authentication.
- We can use supported mechanisms, SSL/TLS with or without google token based authentication.
- gRPC also provides a simple authentication API that let us provide all necessary authentication information as credentials when creating a channel.

Supported auth mechanisms

- **SSL/TLS**
    - gRPC has a SSL/TLS integration, promotes the use SSL/TLS to authenticate a server and to encrypt all the data exchanged between client and server.
    - We can used default SSL/TLS credential.

- **ALTS (Application Layer Transport Security)**
    - Application layer transport security (ALTS) is an authentication and encryption system developed by google.
    - It is similar to TLS but has been designed and optimized to meet the need of google environment.
    - It has the following features.
        - Create gRPC server and client with ALTS as the transport security protocol.
        - ALTS connections are end to end protected with privacy and integrity.
        - Client authentication and server authorization support.
        - Minimal code changes to enable ALTS.

- **Token based authentication with google**
    - gRPC provides a generic mechanism to attach metadata based credentials to request and response.
    - By using access tokens (OAuth2 token) only for google services.

Error Handling

- Standard Error Model
    - When RPC call completes server returns an OK status
    - If an error occurs. gRPC returns one of its error codes with optional string error message that provides further details about the error.
    - Error information is available to grpc client.
    - We can create our own status code and Error messages.