The Concepts of GRPC

Introduction

- Several organizations, from tech giants to startups, owe their business development to APIs.
- APIs make a range of web products accessible to millions of users across the Internet.
- Choosing the right tech. to provide an API for any app is crucial.

What Is gRPC?

- gRPC is a robust open-source RPC framework used to build scalable and fast APIs.
- It allows the client and server apps to communicate transparently and develop connected systems.
- Many leading tech firms have adopted gRPC, such as Google, Netflix, Square, IBM, Cisco, & Dropbox.
- This framework relies on HTTP/2, protocol buffers, and other modern technology stacks to ensure max. API security, performance, and scalability.

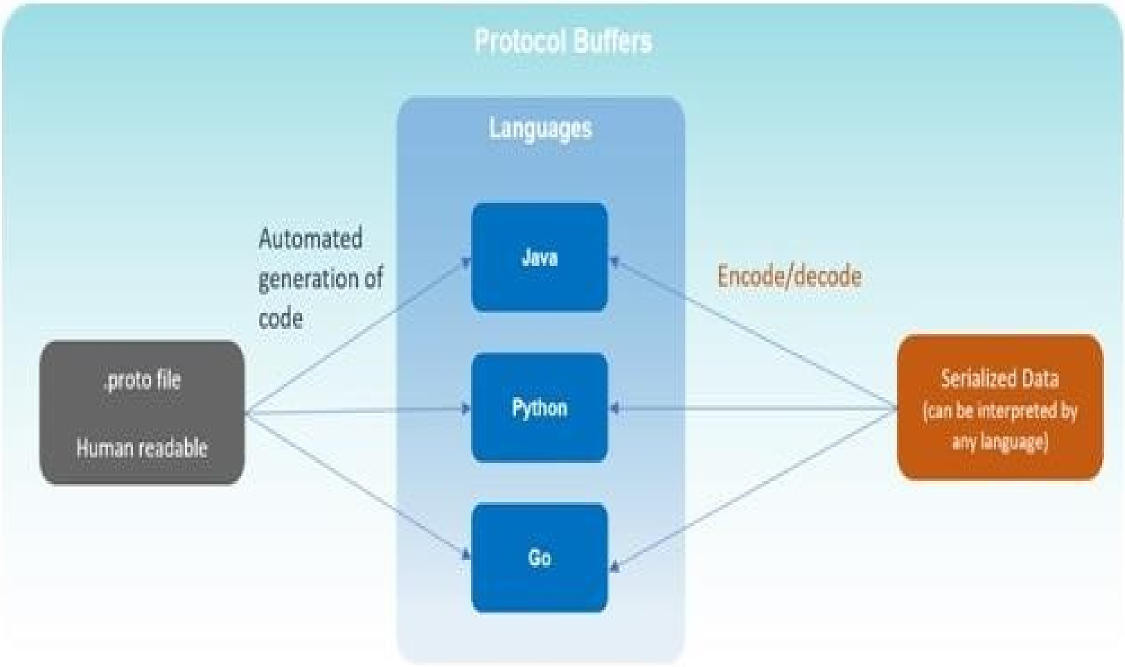
History of the gRPC

- In 2015, Google developed gRPC as an extension of the RPC framework for its internal infrastructure to link many microservices created with different technologies.
- Later -open-source for community use.
- In 2017, it became the Cloud Native Computing Foundation (CNCF) incubation project due to its increasing popularity.

Protocol Buffers

- Most of the gRPC advantages stem from the following concepts:
  - **Protocol buffers**, or Protobuf, is Google’s **serialization / deserialization protocol** that enables the **easy definition of services and auto-generation of client libraries**.
  - **gRPC uses this protocol as their Interface Definition Language (IDL)** and serialization toolset.
  - Current version - proto3,
  - gRPC **services and messages** between clients and servers are defined in **proto files**.

  - The **Protobuf compiler** - protoc, **generates client and server code** that loads the .proto file into the memory at runtime and uses the in-memory schema to serialize / deserialize the binary message.
  - After code generation, message is exchanged between the client and remote service.
- Protobuf offers benefits over JSON and XML.
- **Parsing with Protobuf** requires fewer CPU resources since **data is converted into a binary format**, and encoded messages are lighter in size.
- So, **messages are exchanged faster**, even in machines with a slower CPU, such as mobile devices.


Protocol Buffers
Languages
Automated generation of code
Java
Encode/decode
.proto file
Human readable
Python
Serialized Data (can be interpreted by any language)
Go

Streaming

- Streaming is another key concept of gRPC, where many processes can take place in a single request.
- The multiplexing capability of HTTP/2
  – Sending multiple responses or
  – Receiving multiple requests together over a single TCP connection

Streaming Types

- **Server-streaming RPCs**
  – The client sends a single request to the server and receives back a stream of data sequences.

  - The sequence is preserved, and server messages continuously stream until there are no messages left.

- **Client-streaming RPCs**
  - The client sends a stream of data sequences to the server, which then processes and returns a single response to the client.
  - Once again, gRPC guarantees message sequencing within an independent RPC call.
- **Bidirectional-streaming RPCs**
  - It is two-way streaming where both client and server sends a sequence of messages to each other.
- Both streams operate independently; thus, they can transmit messages in any sequence.
- The sequence of messages in each stream is preserved.

Http 2.0 Advanced Capabilities

- Compatible with HTTP/1.1, HTTP/2
- Binary Framing Layer

- Unlike HTTP/1.1, HTTP/2 **request/response is divided into small messages and framed in binary format**, making message transmission efficient.
- With binary framing, the HTTP/2 protocol has made request/response multiplexing possible without blocking network resources.

- **Streaming**
  - **Bidirectional full-duplex** streaming in which the client can request and the server can respond simultaneously.
- **Flow Control**
  - Flow control mechanism is used in HTTP/2, enabling detailed control of memory used to buffer in-flight messages.
- **Header Compression**
  - Everything in HTTP/2, including headers, is **encoded** before sending, significantly improving overall performance.

- Using the **HPACK compression** method, HTTP/2 only shares the value different from the previous HTTP header packets.
- Processing
  - **With HTTP/2, gRPC supports both Synchronous and Asynchronous processing**,
  - This can be used to perform different types of interaction and streaming RPCs.

Benefits Derived

- All these features of HTTP/2 enable gRPC to:
  - use fewer resources,
  - resulting in **reduced response times** between apps and services running in the cloud
  - **longer battery life** for a client running mobile devices.

Channels

- Channels are a core concept in gRPC.
- The HTTP/2 streams allow many simultaneous streams on one connection.

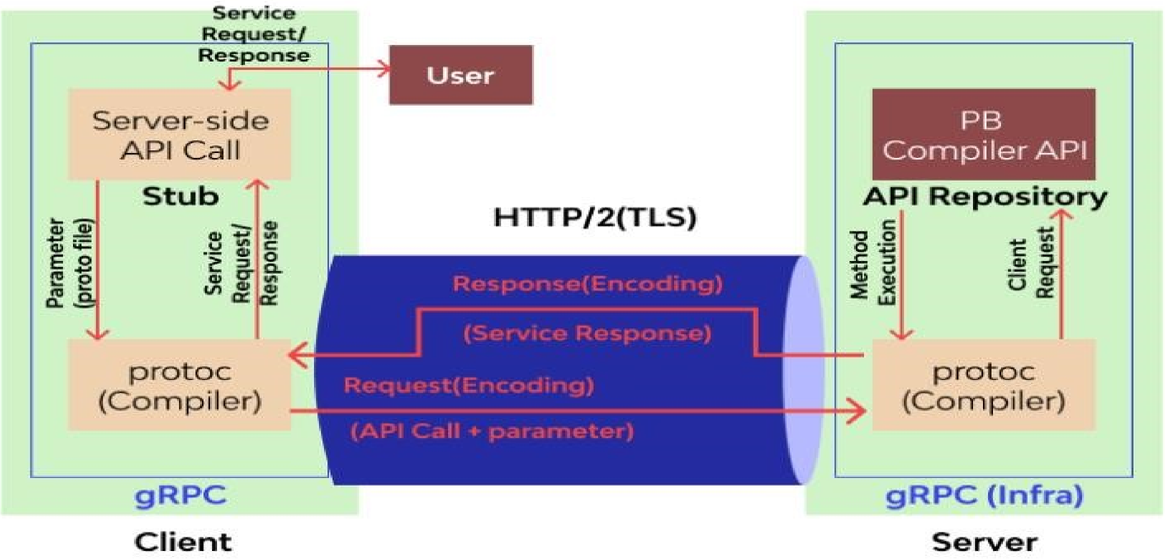
- Channels extend this concept by supporting multiple streams over multiple concurrent connections.
- **Channels** provide a way to connect to the gRPC server on a specified address and port and are **used in creating a client stub.**

Code Generation

- The prime feature of gRPC methodology is the **native code generation** for client/server applications.
- **gRPC frameworks use protoc compiler to generate code from the .proto file.**
- **It can produce server-side skeletons and client-side network stubs**, which saves significant development time in applications with various services.

gRPC Architecture

- In gRPC, **every client service includes a stub** (autogenerated files), similar to an interface containing the current remote procedures.



- The gRPC client makes the local procedure call to the stub with parameters to be sent to the server.
- The client stub then serializes the parameters with the marshaling process using Protobuf and forwards the request to the local client-time library in the local machine.
- **The OS makes a call to the remote server machine via HTTP/2 protocol**.

- The server’s OS receives the packets and **calls the server stub procedure**, which decodes the received parameters and executes the respective procedure invocation using Protobuf.
- The server stub then sends back the encoded response to the client transport layer.
- The client stub gets back the result message and unpacks the returned parameters, and the execution returns to the caller.

Performance

- gRPC offers up to **10x faster performance** and API-security than REST+JSON communication as it uses Protobuf and HTTP/2.
- Protobuf serializes messages on server and client quickly, resulting in small and compact message payloads.
- **HTTP/2 scales up the performance** ranking via:
  – server push,
  – multiplexing, &
  – header compression.

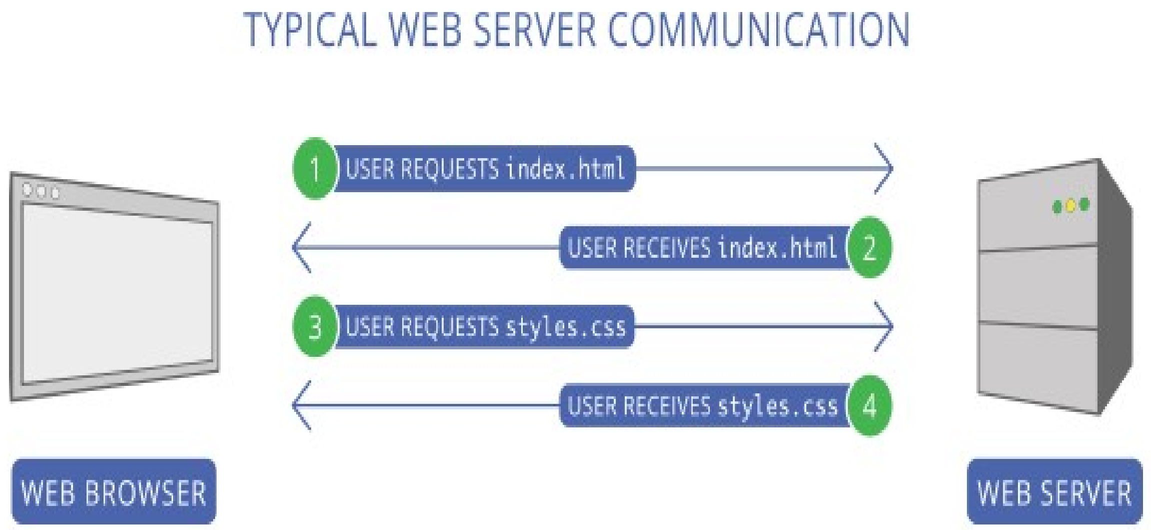
- **Server push** enables HTTP/2 to push content from server to client before getting requested, while multiplexing eliminates head-of-line blocking.
- HTTP/2 uses a more **advanced compression method** to make the messages smaller, resulting in faster loading.

HTTP/2 Server Push

- Server push allows to send site assets to the user before they've even asked for them.
- Accessing websites has always followed a request and response pattern.
- The user sends a request to a remote server, and with some delay, the server responds with the requested content.
- **The initial request to a web server is commonly for an HTML document**.
- **Server replies with the requested HTML resource**.
- The HTML is then parsed by the browser, where references to other assets are discovered, such as **style sheets, scripts and images**.

- Upon their discovery, the **browser makes separate requests for those assets**, which are then responded to in kind.

Typical Web Access Pattern


TYPICAL WEB SERVER COMMUNICATION
1 USER REQUESTS index.html
USER RECEIVES index.html 2
3 USER REQUESTS styles.css
USER RECEIVES styles.css 4
WEB BROWSER
WEB SERVER

Drawbacks

- Problem is that it forces the user to wait for the browser to discover and retrieve critical assets until *after* an HTML document has been downloaded.
- This delays rendering and increases load times.

- **Server push lets the server preemptively "push" website assets** to the client without the user having explicitly asked for them.
- When used with care, one can send what we *know* the user is going to need for the page they're requesting.

Interoperability

- gRPC tools and libraries are designed to work with multiple platforms and programming languages, including Java, JavaScript, Ruby, Python, Go, Dart, Objective-C, C#, and more.

Should I Use gRPC instead of REST?

- Basically, gRPC is another alternative that could be useful in certain circumstances:
  - large-scale microservices connections
  - real-time communication
  - low-power, low-bandwidth systems
  - multi-language environments

- While HTTP supports mediators for edge caching, **gRPC calls use the POST method, which is a threat to API-security**.
- **The responses can’t be cached through intermediaries.**
- Moreover, the gRPC specification doesn’t make any provisions and even indicates the wish for cache semantics between server and client.