gRPC Concepts

# 1. Introduction to gRPC

gRPC (Google Remote Procedure Call) is a high-performance, open-source RPC framework developed by Google. It enables communication between distributed applications using HTTP/2 as its transport protocol. gRPC allows strongly-typed, language-agnostic communication between services.

# 2. Key Concepts in gRPC

## 2.1 Protocol Buffers (Protobuf)

Protocol Buffers (Protobuf) is the Interface Definition Language (IDL) used in gRPC. It defines the structure of data (messages) and service methods in `.proto` files.

Example:

syntax = "proto3";  
  
service Greeter {  
 rpc SayHello (HelloRequest) returns (HelloReply);  
}  
  
message HelloRequest {  
 string name = 1;  
}  
  
message HelloReply {  
 string message = 1;  
}

## 2.2 Service Definition

In gRPC, services are defined in `.proto` files. Each service contains RPC methods that can be called by clients. For example, `SayHello` is a remote function exposed by the server.

## 2.3 Communication Patterns

gRPC supports four types of communication patterns:

• Unary RPC – Single request and single response.

• Server Streaming RPC – Single request and multiple responses streamed back.

• Client Streaming RPC – Multiple requests sent from client, single response returned.

• Bidirectional Streaming RPC – Both client and server exchange multiple messages simultaneously.

## 2.4 Generated Code

The `protoc` compiler generates client and server code in multiple programming languages based on `.proto` definitions. Developers implement server logic, while clients call RPCs as local functions.

# 3. Advantages of gRPC

• Language-agnostic – Works across Python, Go, Java, C++, etc.

• Efficient – Uses HTTP/2 for multiplexing, streaming, and low-latency communication.

• Strongly Typed – Protobuf enforces a strict schema, reducing errors.

• Streaming Support – Built-in support for server, client, and bidirectional streaming.

• Modern – Widely adopted for microservices and distributed systems.

# 4. Example Workflow

1. Define service and messages in `.proto` file.

2. Run `protoc` to generate client and server stubs.

3. Implement server logic in chosen language.

4. Client calls RPC methods as if they were local functions.

5. Communication happens over HTTP/2 with Protobuf serialization.