gRPC

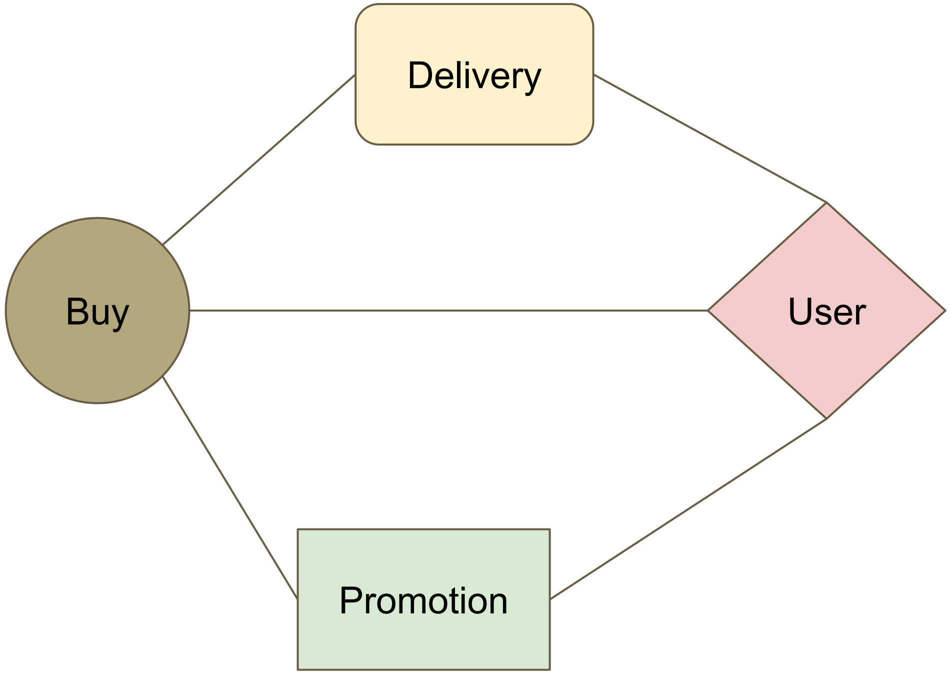
Remote procedure call

Table of content

1. Microservice nowaday

Today’s trend is to build microservice
Why building an API is hard?
What we really need?

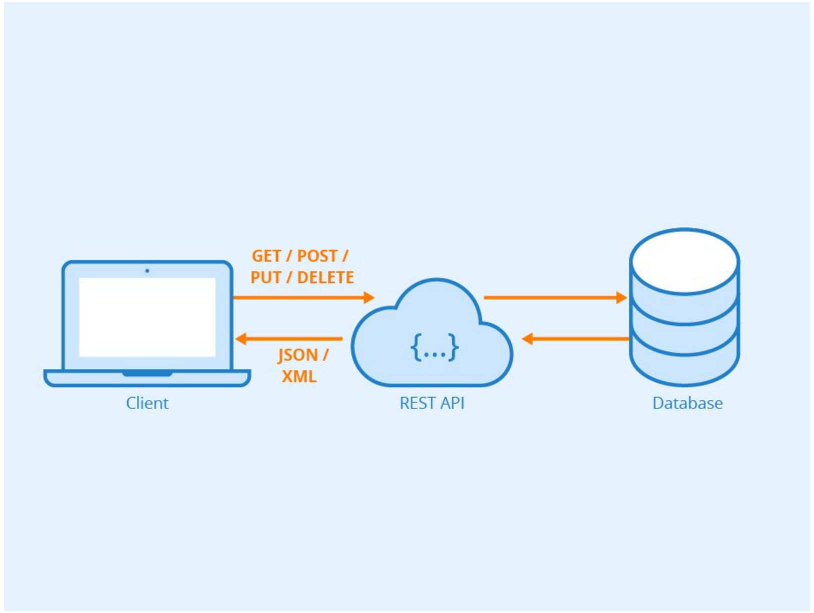
Today's trend is to build microservice



Today's trend is to build microservice

These microservices must exchange information and agree on:

- The API to exchange data
- The data format
- Load balancing
- Many other



Building an API is hard

- Need to think about data model
  - JSON
  - XML

- Need to think about the entry point
  - GET /api/v1/a/b/get/1
  - POST /api/v1/a/b/post/235

- Need to think about how to invoke it and handle errors
  - API
  - Error

Building an API is hard

- How about latency?
- How about scalability to 1000s of clients?
- How about load balancing?
- ...

Hard to build

What we really need?

It’s all about the data

- Send me this REQUEST (Client)
- I will send you this RESPONSE (Server)

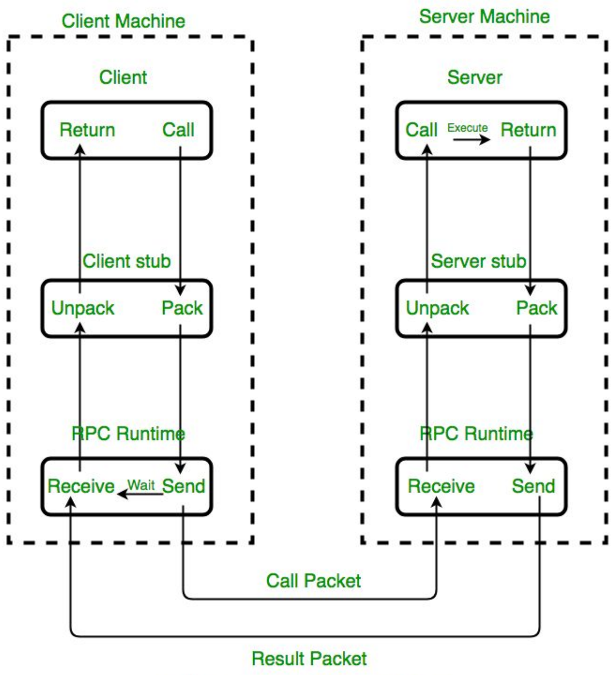
=> **gRPC framework**

2. What is gRPC ?

What is RPC, gRPC ?
Why we use gRPC?
Types of API in gRPC

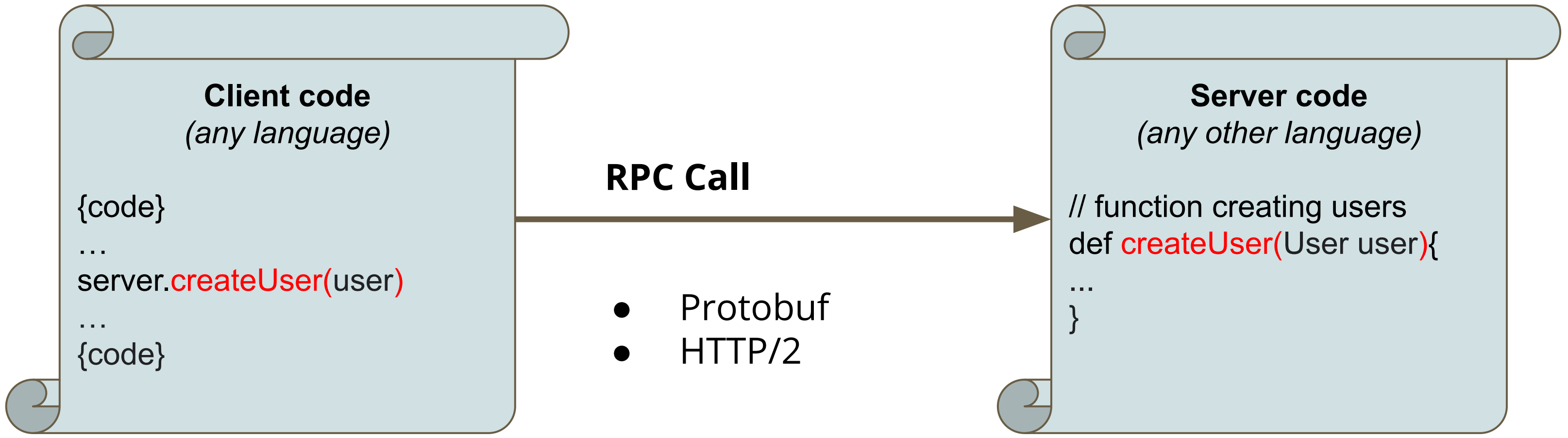
What is RPC?

Remote Procedure Call (RPC) is a method of calling a function from a remote computer to get the result.



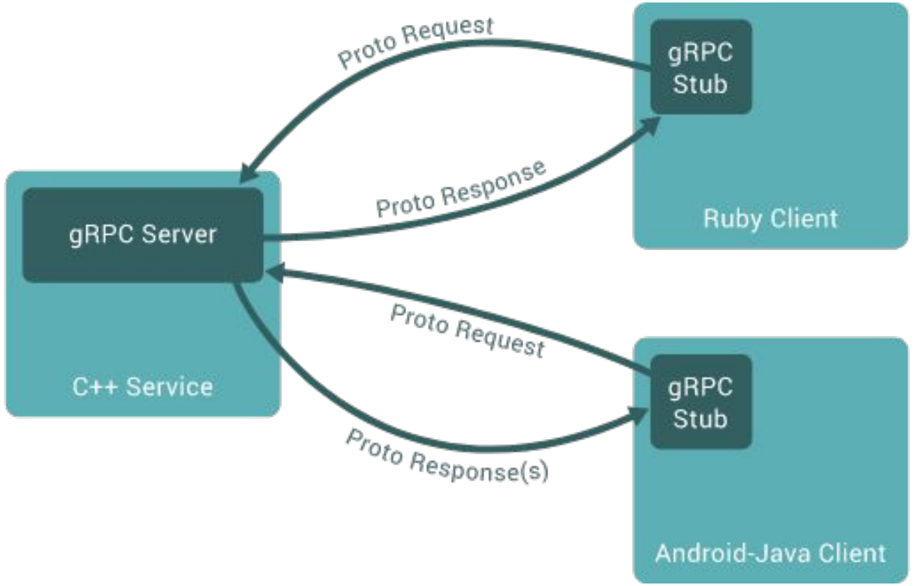
Implementation of RPC mechanism

What is gRPC?



C# C++ Dart Go Java Kotin Node PHP Python Ruby Objective-C

What's gRPC?



- Blocking/synchronous stub
- Non-blocking/asynchronous stub

C# C++ Dart Go Java Kotin Node PHP Python Ruby Objective-C

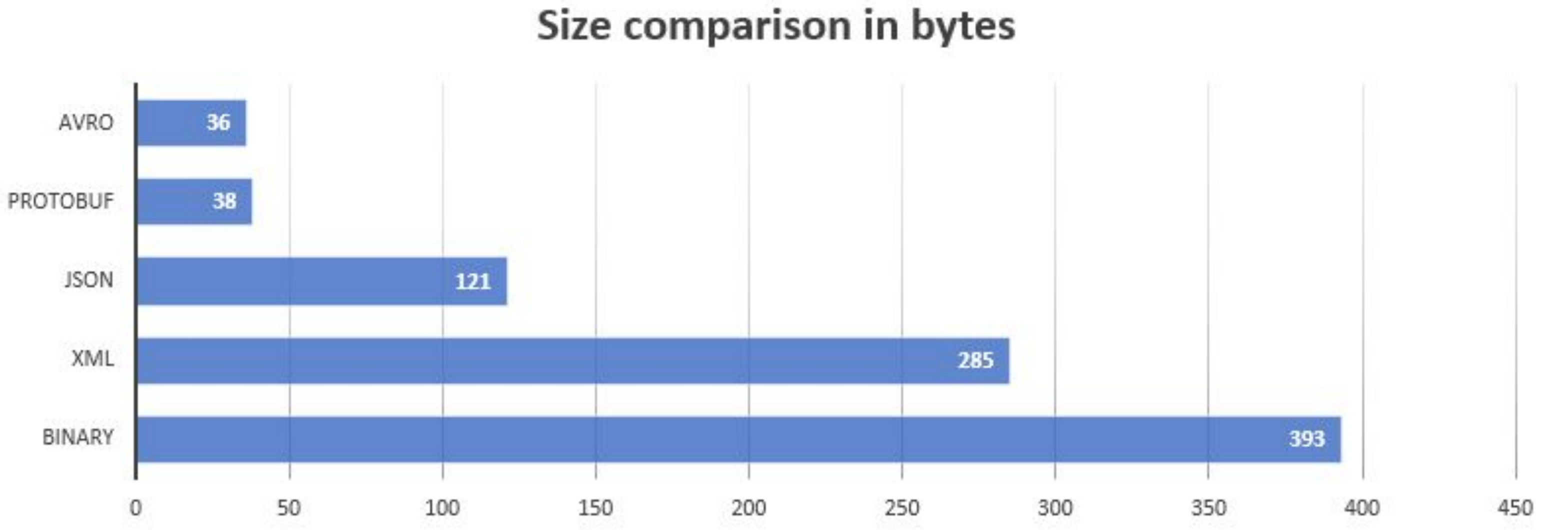
Why we use gRPC?

- Define the messages and services using Protocol Buffers
- Using HTTP/2 for communications (HTTP/2 vs HTTP/1.1 - Performance Comparison (imagekit.io))
- One **.proto** file work for over 12 programming language
- Scalability in gRPC
- Security in gRPC

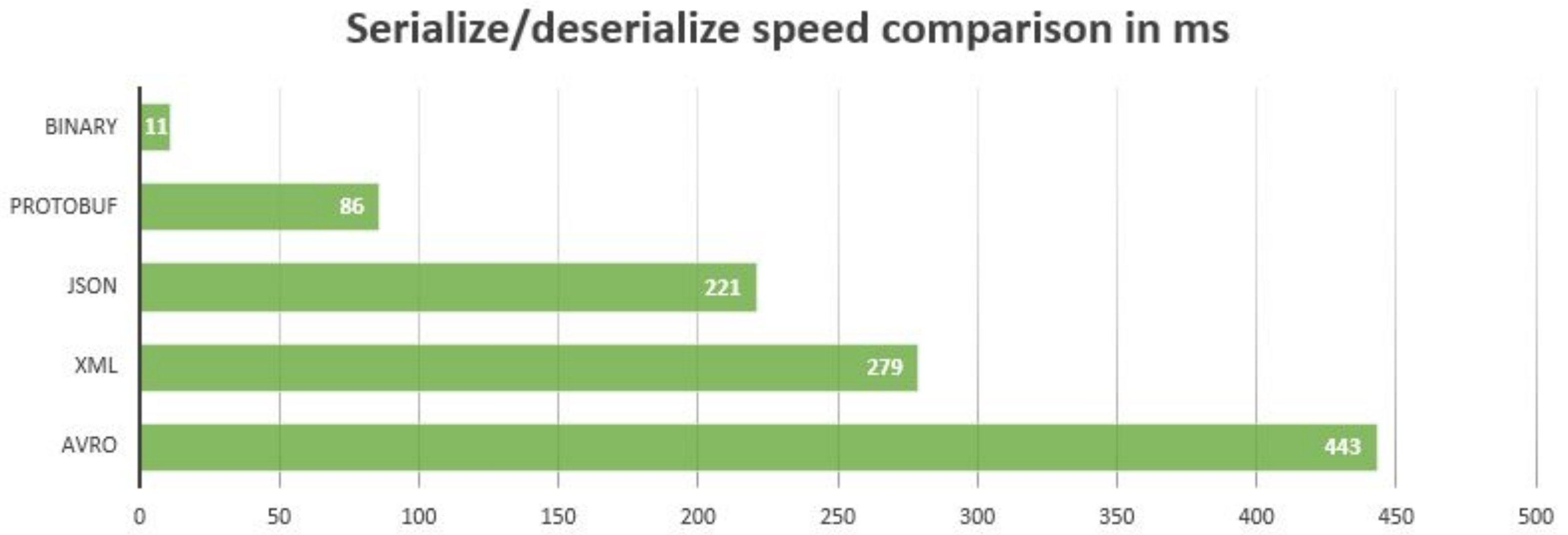
What is ProtoBuf?

- Is a binary format created by Google
- Allow serialization and deserialization of structured data
- Make it simpler, smaller, faster and more maintainable then XML

What is ProtoBuf?



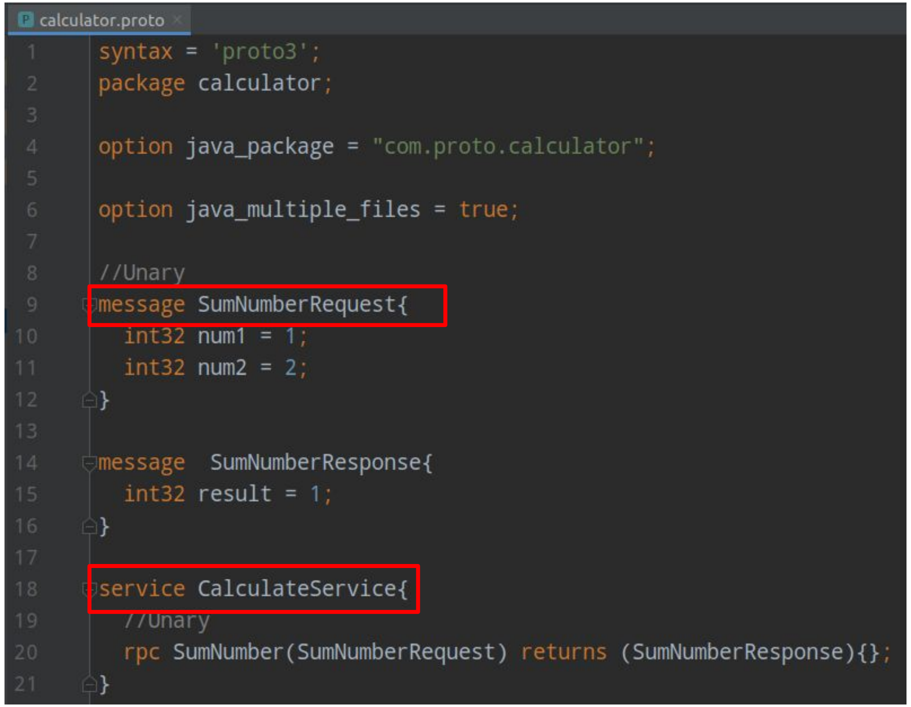
What is ProtoBuf?



Protocol Buffers role in gRPC

Protocol Buffers is used to define the:

- Messages
- Service

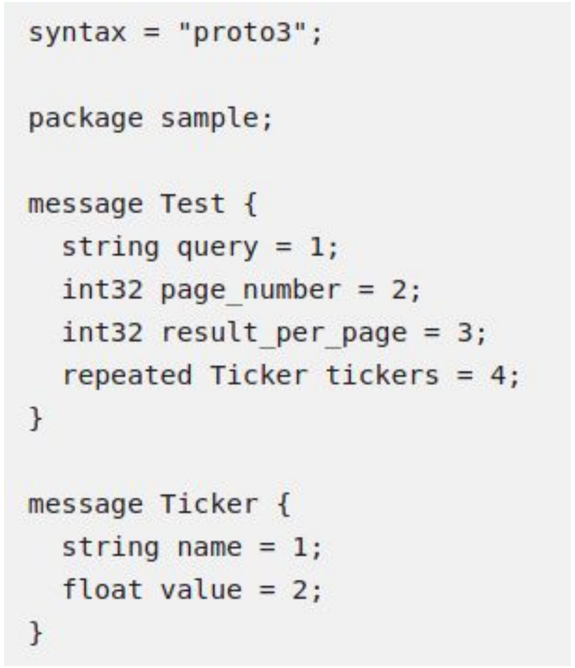


calculator.proto

```
syntax = 'proto3';
package calculator;

option java_package = "com.proto.calculator";

option java_multiple_files = true;

//Unary
message SumNumberRequest{
  int32 num1 = 1;
  int32 num2 = 2;
}

message  SumNumberResponse{
  int32 result = 1;
}

service CalculateService{
  //Unary
  rpc SumNumber(SumNumberRequest) returns (SumNumberResponse){};
}
```

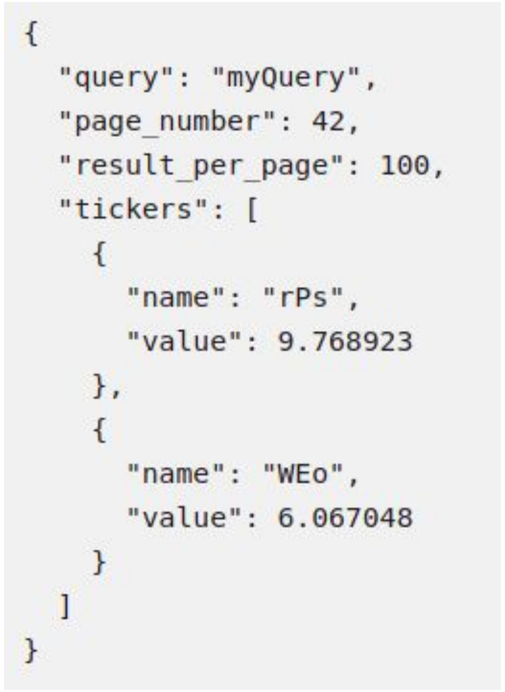
Efficiency of Protocol Buffers over JSON

Protocol Buffer



```
syntax = "proto3";

package sample;

message Test {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
  repeated Ticker tickers = 4;
}

message Ticker {
  string name = 1;
  float value = 2;
}
```

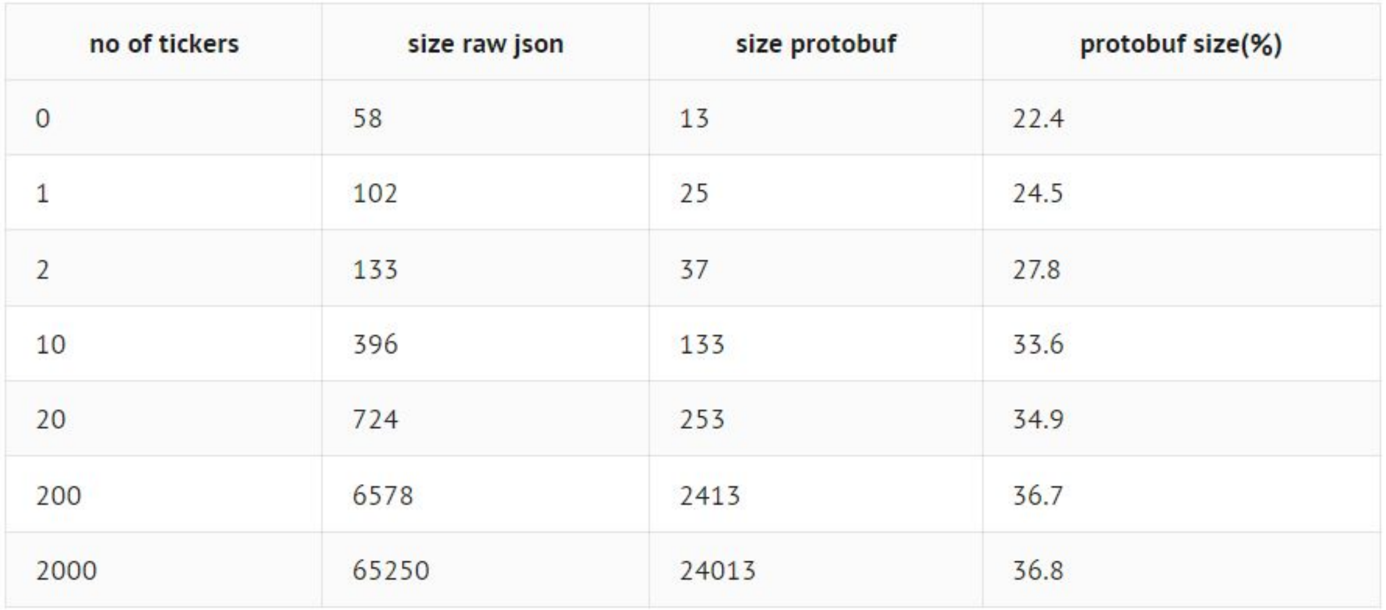
JSON



```
{
  "query": "myQuery",
  "page_number": 42,
  "result_per_page": 100,
  "tickers": [
    {
      "name": "rPs",
      "value": 9.768923
    },
    {
      "name": "WEo",
      "value": 6.067048
    }
  ]
}
```

Efficiency of Protocol Buffers over JSON

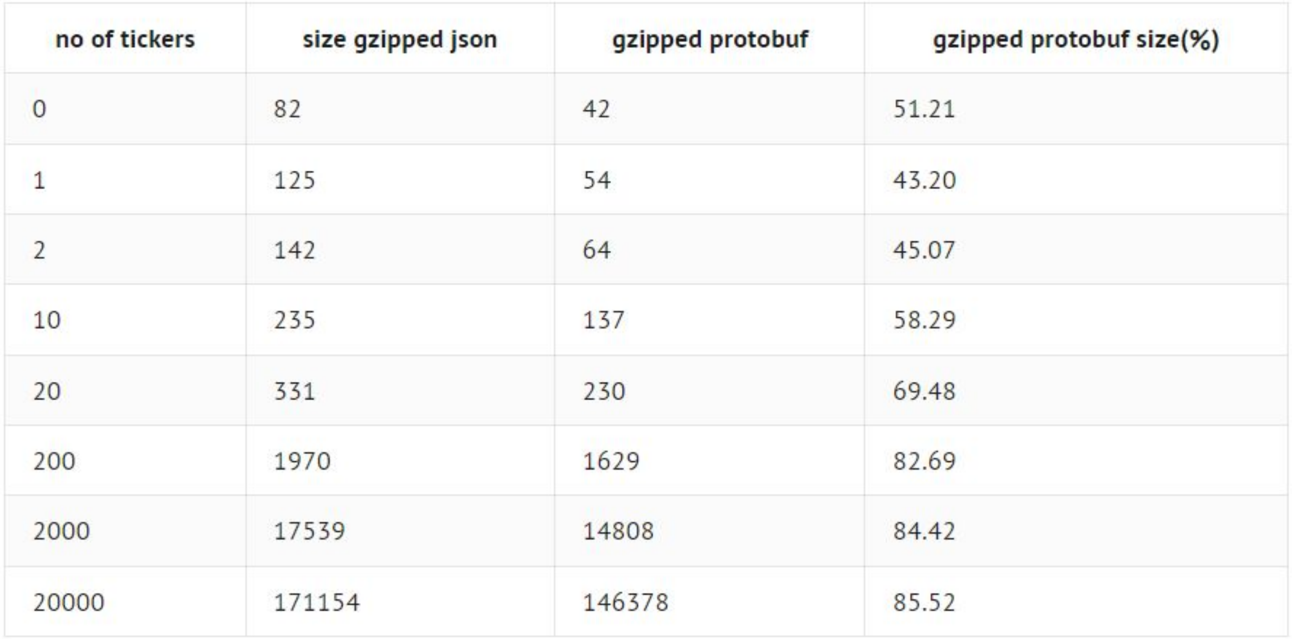
Raw json



no of tickers	size raw json	size protobuf	protobuf size(%)
0	58	13	22.4
1	102	25	24.5
2	133	37	27.8
10	396	133	33.6
20	724	253	34.9
200	6578	2413	36.7
2000	65250	24013	36.8

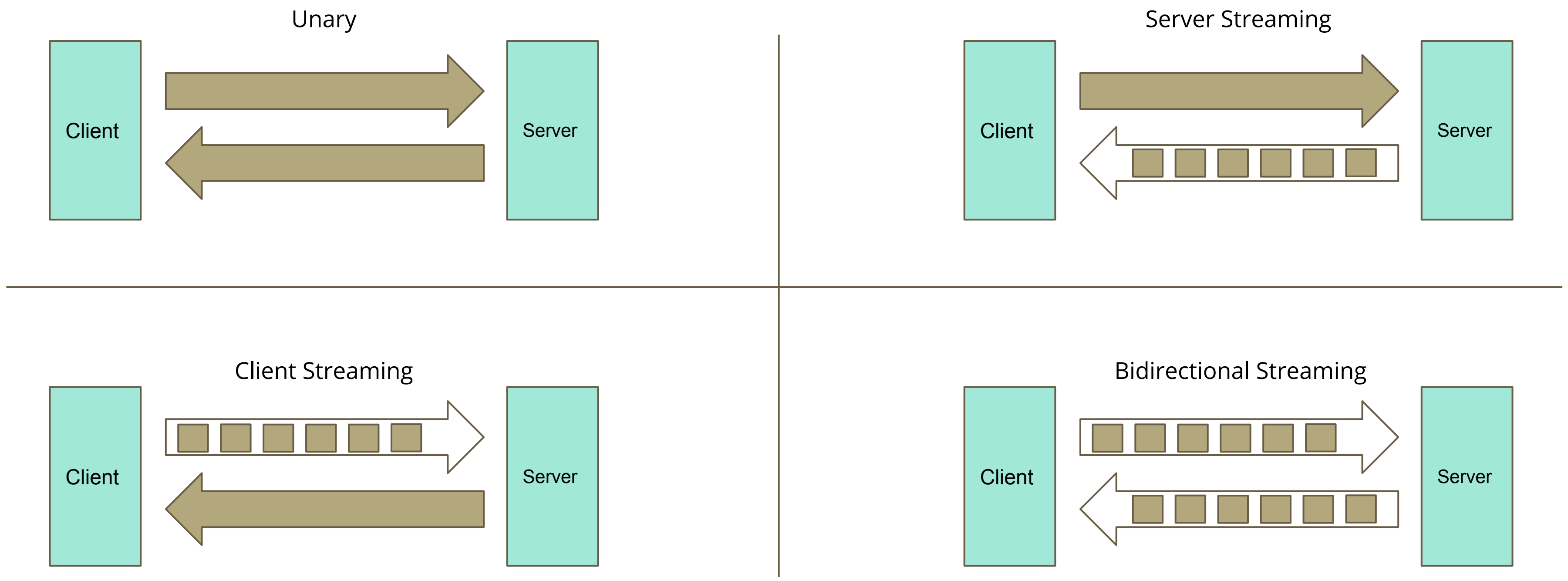
Efficiency of Protocol Buffers over JSON

gzipped json and gzipped protobuf

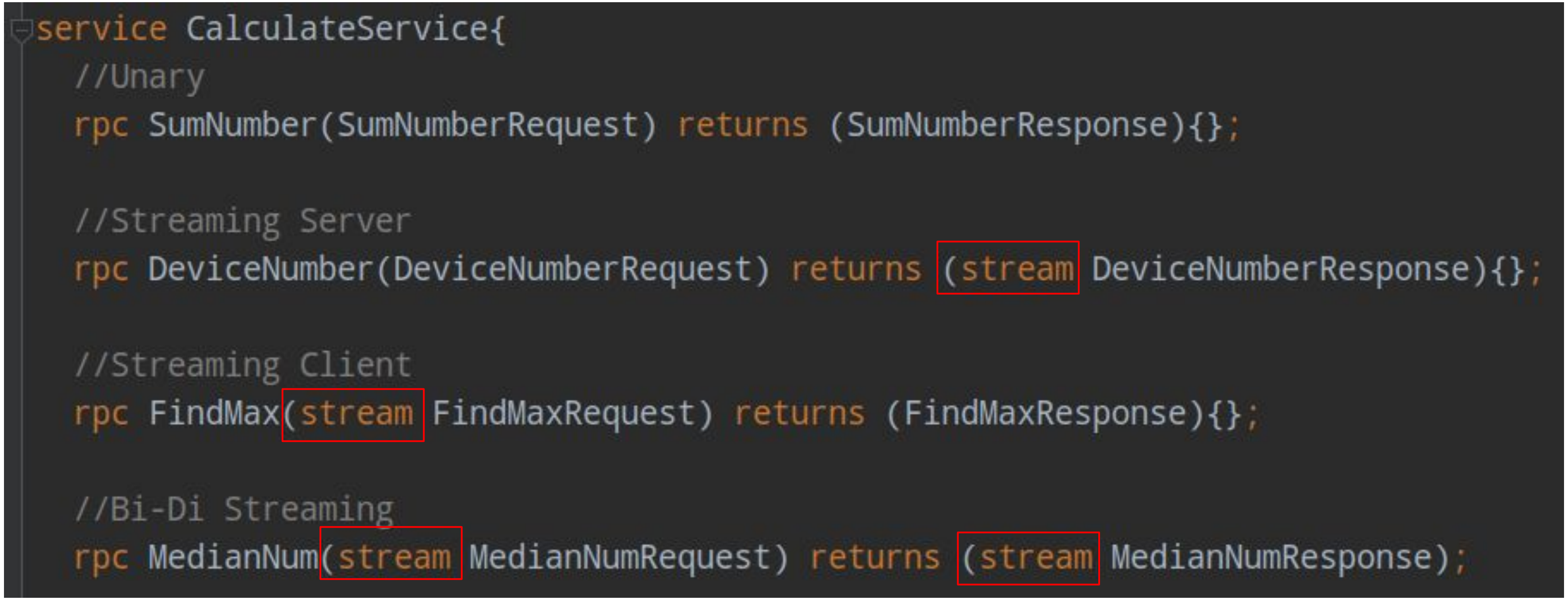


no of tickers	size gzipped json	gzipped protobuf	gzipped protobuf size(%)
0	82	42	51.21
1	125	54	43.20
2	142	64	45.07
10	235	137	58.29
20	331	230	69.48
200	1970	1629	82.69
2000	17539	14808	84.42
20000	171154	146378	85.52

Types of API in gRPC



Types of API in gRPC



```
service CalculateService{
  //Unary
  rpc SumNumber(SumNumberRequest) returns (SumNumberResponse){};

  //Streaming Server
  rpc DeviceNumber(DeviceNumberRequest) returns (stream DeviceNumberResponse){};

  //Streaming Client
  rpc FindMax(stream FindMaxRequest) returns (FindMaxResponse){};

  //Bi-Di Streaming
  rpc MedianNum(stream MedianNumRequest) returns (stream MedianNumResponse);
```

Scalability in gRPC

gRPC Servers are asynchronous by default

- Do not block threads on request
- Can serve millions of requests in parallel

gRPC Clients can be asynchronous or synchronous

Google has 10 BILLION gRPC requests being made per second internally

3. How to implement gRPC

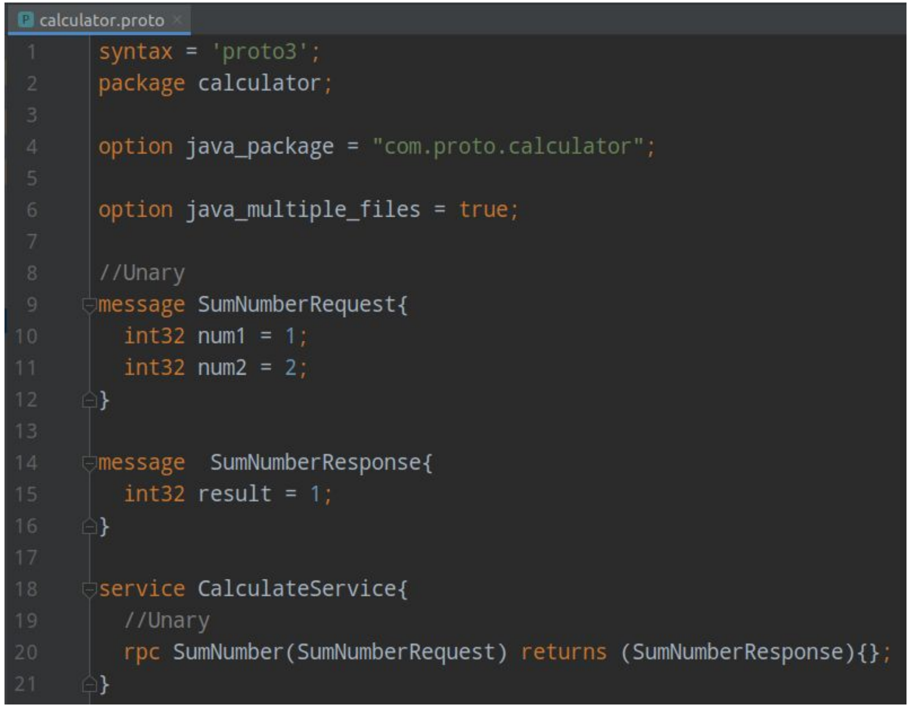
Directory structure

Using java for define client and server

- calculate.proto
- server.java
- serverImpl.java
- client.java

calculate.proto

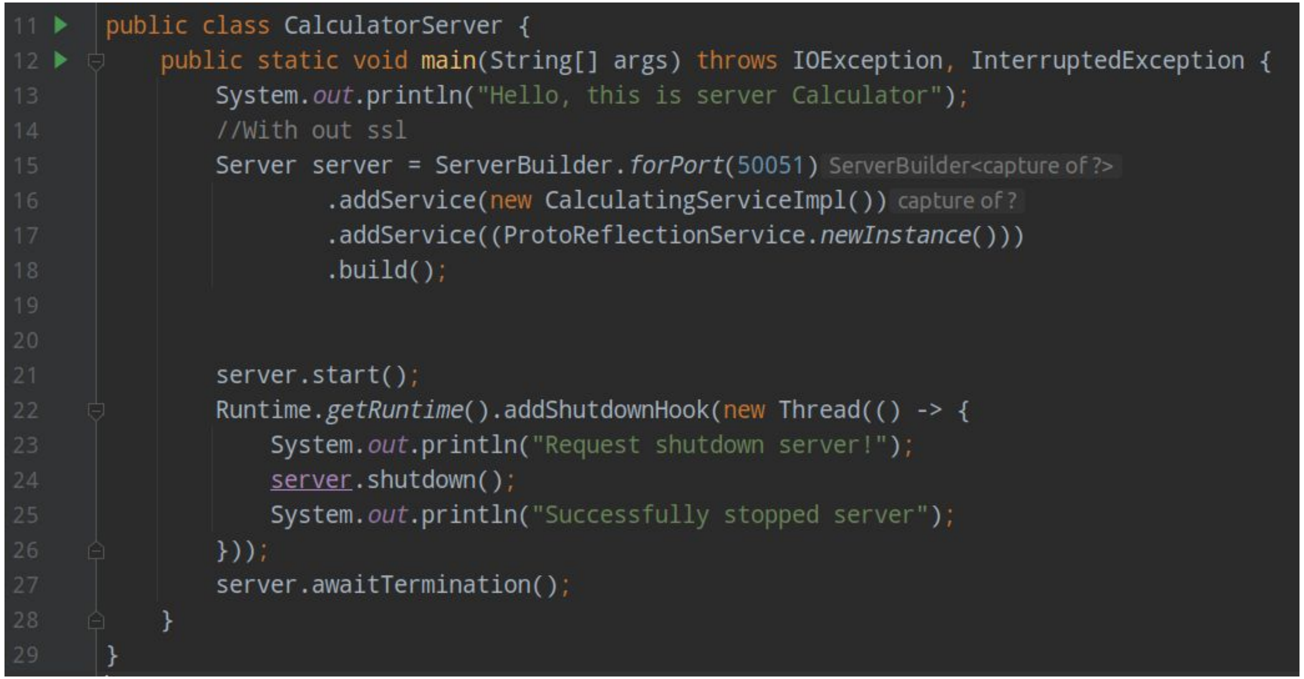
- syntax
- package
- 2 option
- message
- service



```
syntax = 'proto3';
package calculator;

option java_package = "com.proto.calculator";

option java_multiple_files = true;

//Unary
message SumNumberRequest{
  int32 num1 = 1;
  int32 num2 = 2;
}

message  SumNumberResponse{
  int32 result = 1;
}

service CalculateService{
  //Unary
  rpc SumNumber(SumNumberRequest) returns (SumNumberResponse){};
}
```

server.java

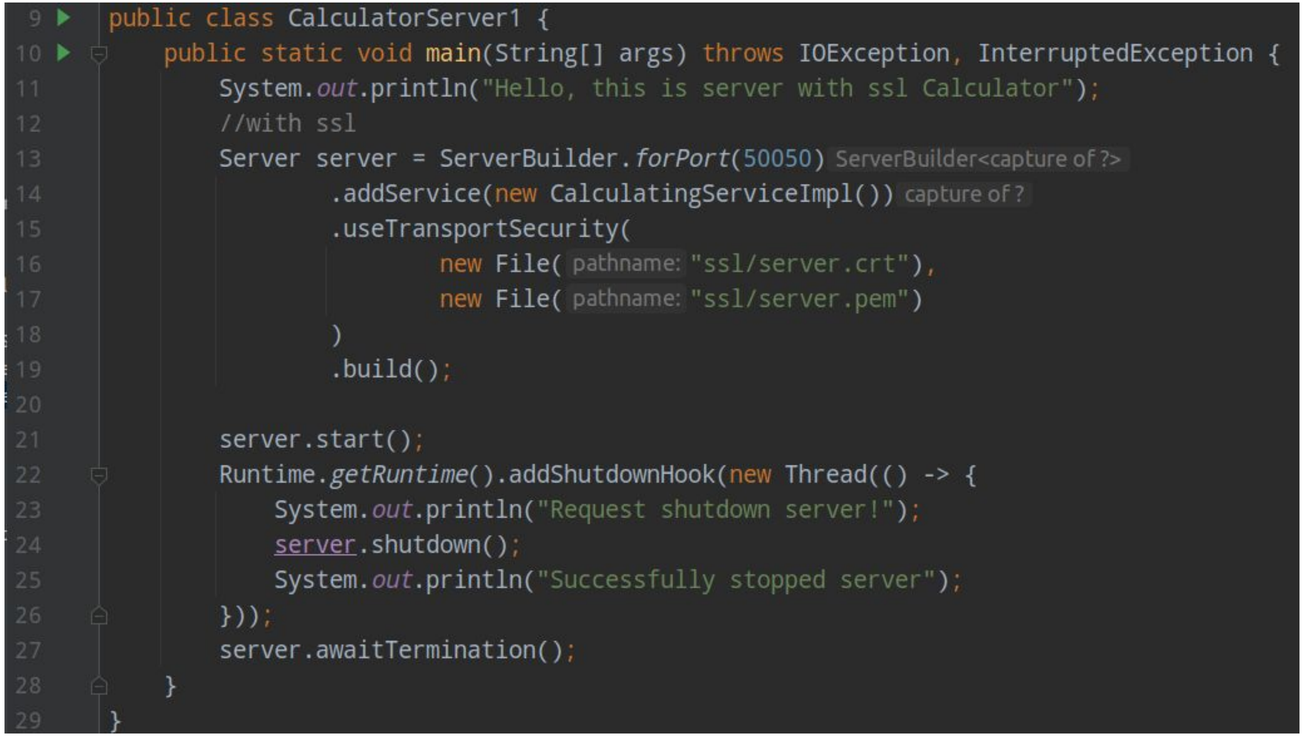
Without SSL



```
public class CalculatorServer {
    public static void main(String[] args) throws IOException, InterruptedException {
        System.out.println("Hello, this is server Calculator");
        //With out ssl
        Server server = ServerBuilder.forPort(50051) ServerBuilder<capture of ?>
                .addService(new CalculatingServiceImpl()) capture of ?
                .addService((ProtoReflectionService.newInstance()))
                .build();


        server.start();
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("Request shutdown server!");
            server.shutdown();
            System.out.println("Successfully stopped server");
        }));
        server.awaitTermination();
    }
}
```

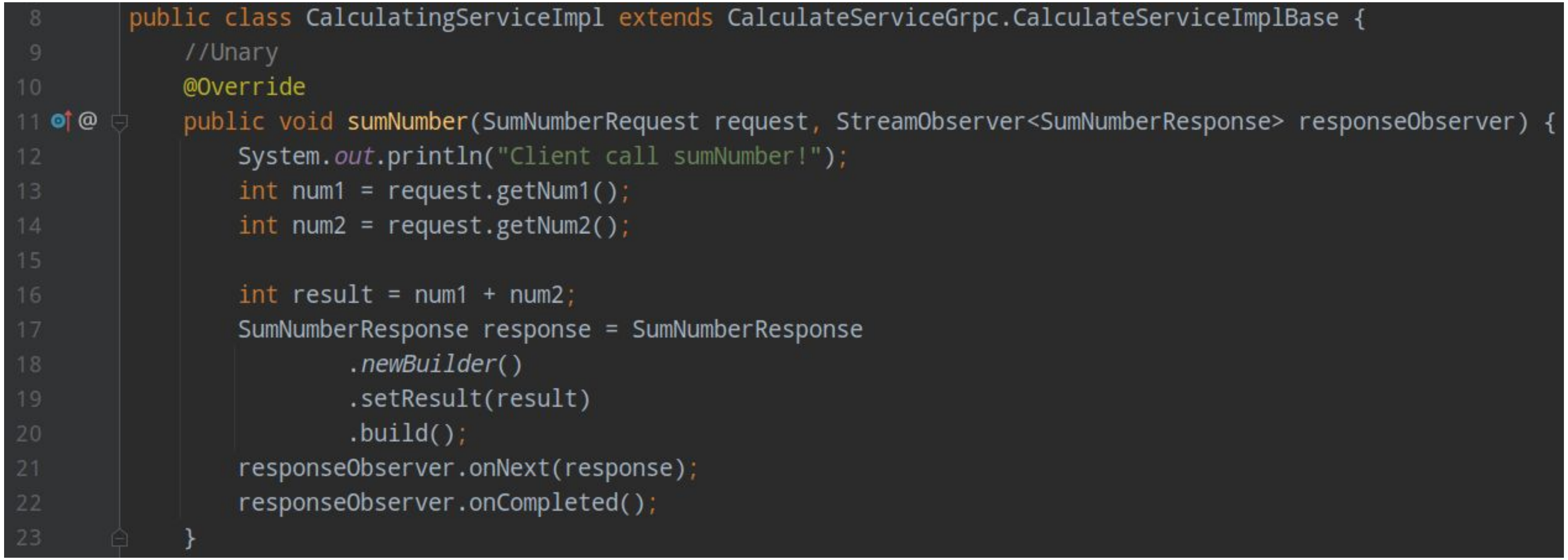
server.java

With SSL



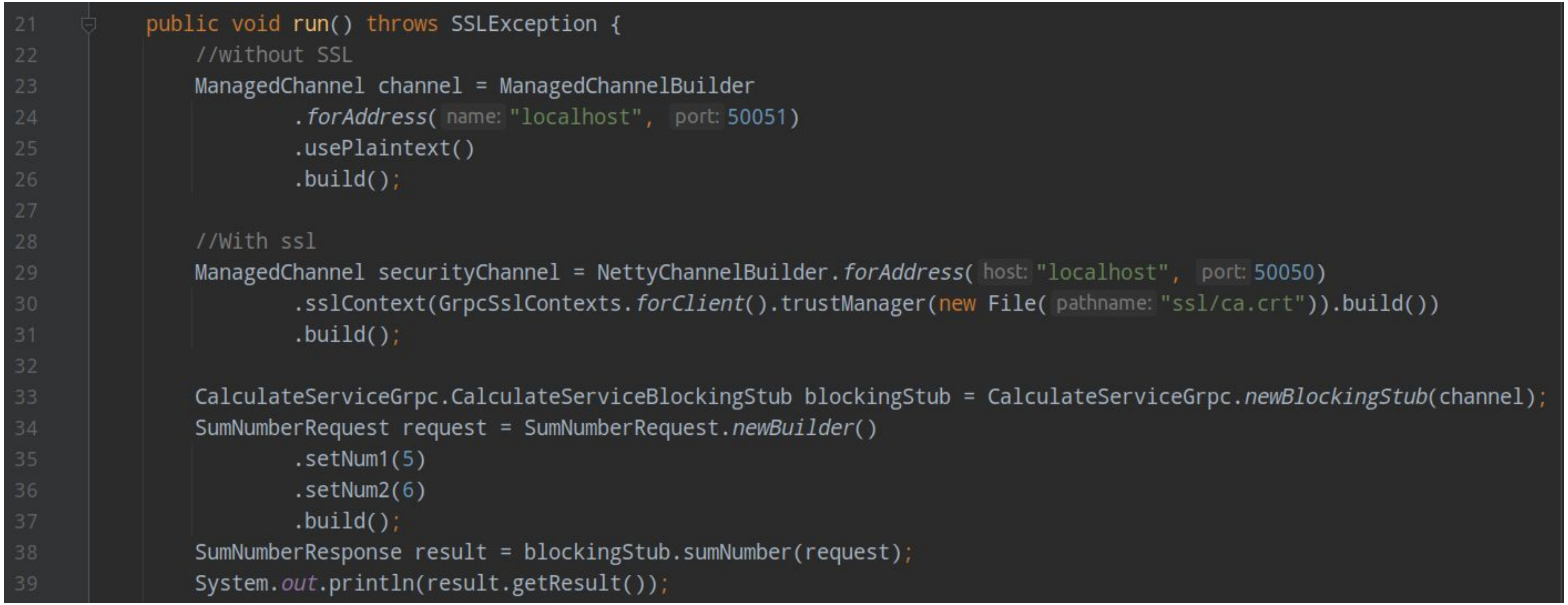
```java
public class CalculatorServer1 {
    public static void main(String[] args) throws IOException, InterruptedException {
        System.out.println("Hello, this is server with ssl Calculator");
        //with ssl
        Server server = ServerBuilder.forPort(50050) ServerBuilder<capture of ?>
                .addService(new CalculatingServiceImpl()) capture of ?
                .useTransportSecurity(
                        new File( pathname: "ssl/server.crt"),
                        new File( pathname: "ssl/server.pem")
                )
                .build();

        server.start();
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("Request shutdown server!");
            server.shutdown();
            System.out.println("Successfully stopped server");
        }));
        server.awaitTermination();
    }
}
```

serverImpl.java



```java
public class CalculatingServiceImpl extends CalculateServiceGrpc.CalculateServiceImplBase {
    //Unary
    @Override
    public void sumNumber(SumNumberRequest request, StreamObserver<SumNumberResponse> responseObserver) {
        System.out.println("Client call sumNumber!");
        int num1 = request.getNum1();
        int num2 = request.getNum2();

        int result = num1 + num2;
        SumNumberResponse response = SumNumberResponse
                .newBuilder()
                .setResult(result)
                .build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
```

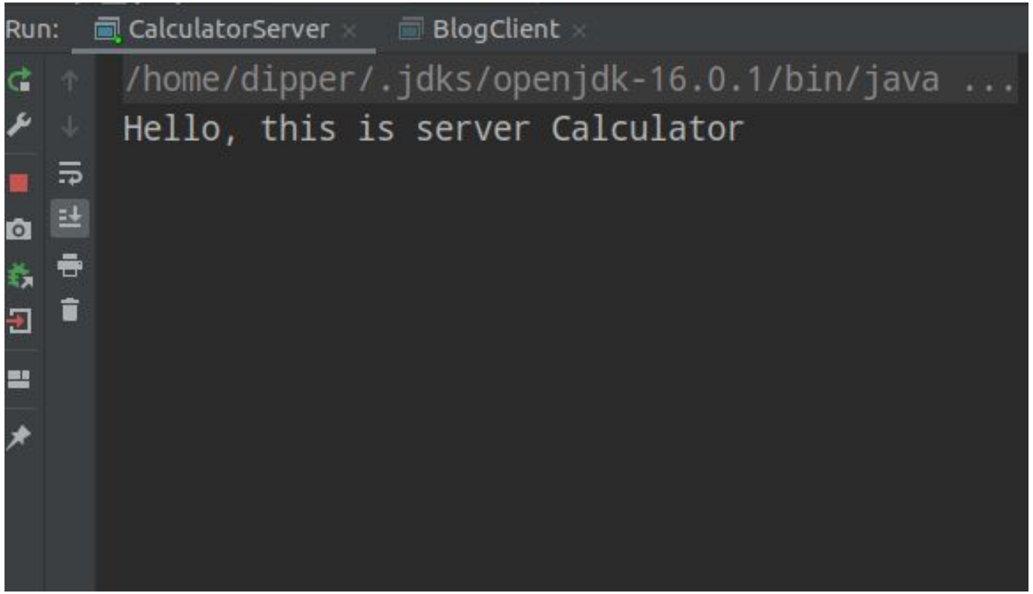
client.java



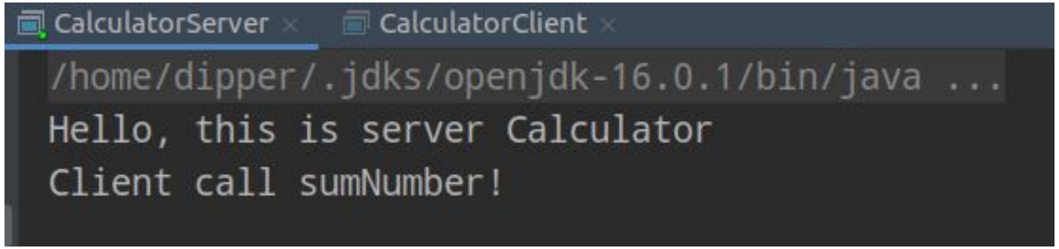
```java
    public void run() throws SSLException {
        //without SSL
        ManagedChannel channel = ManagedChannelBuilder
                .forAddress( name: "localhost",  port: 50051)
                .usePlaintext()
                .build();

        //With ssl
        ManagedChannel securityChannel = NettyChannelBuilder.forAddress( host: "localhost",  port: 50050)
                .sslContext(GrpcSslContexts.forClient().trustManager(new File( pathname: "ssl/ca.crt")).build())
                .build();

        CalculateServiceGrpc.CalculateServiceBlockingStub blockingStub = CalculateServiceGrpc.newBlockingStub(channel);
        SumNumberRequest request = SumNumberRequest.newBuilder()
                .setNum1(5)
                .setNum2(6)
                .build();
        SumNumberResponse result = blockingStub.sumNumber(request);
        System.out.println(result.getResult());
```

Tính tổng 2 số a và b

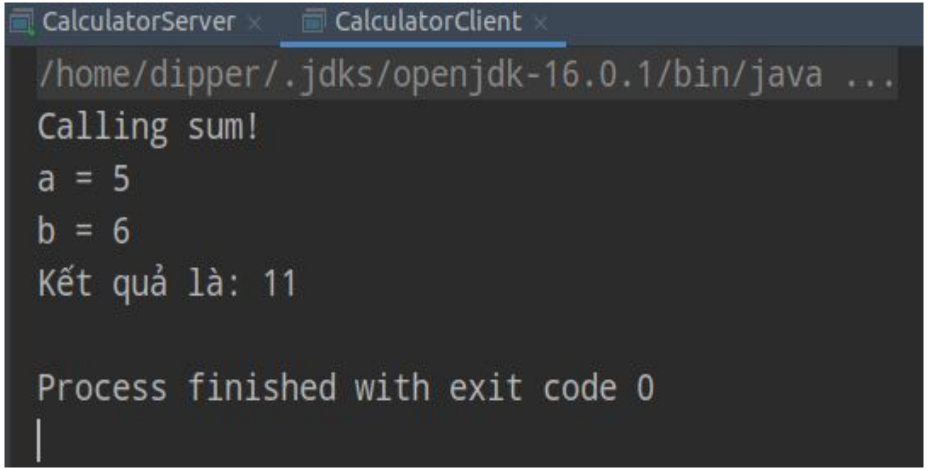
CalculatorServer.java



```
/home/dipper/.jdks/openjdk-16.0.1/bin/java ...
Hello, this is server Calculator
```



```
/home/dipper/.jdks/openjdk-16.0.1/bin/java ...
Hello, this is server Calculator
Client call sumNumber!
```



```
/home/dipper/.jdks/openjdk-16.0.1/bin/java ...
Calling sum!
a = 5
b = 6
Kết quả là: 11

Process finished with exit code 0
```