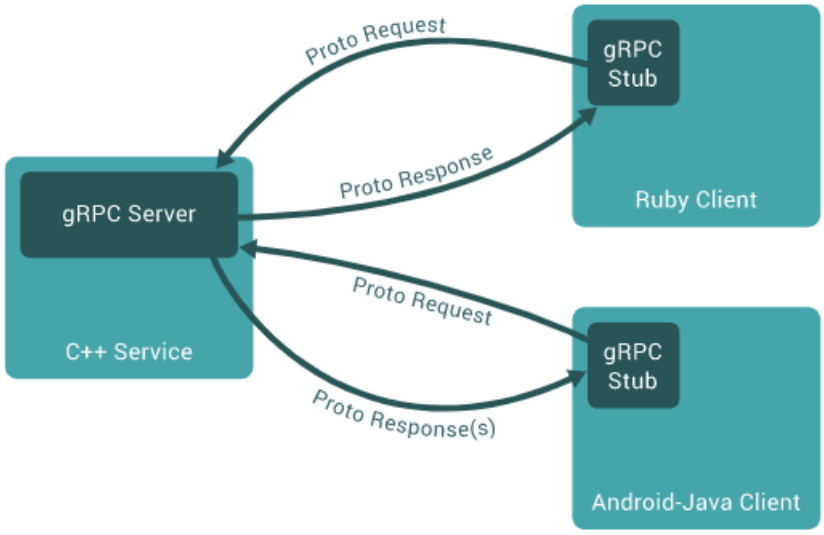
webMethods gRPC Descriptor Guide

> **References**
>
> - **gRPC Document (gRPC 공식 문서)**
> - **Blog (gRPC란)**
> - **Blog (프로토콜 버퍼란)**
> - **Protocol Buffers Documentation (proto3 Guide)**
> - **Blog (Protobuf 문법)**
> - **Youtube (webMethods gRPC Descriptor)**

※ gRPC

- **gRPC 란 (google Remote Procedure Call)**



  - RPC 란 별도의 코딩 없이 다른 주소 공간에서 함수, 프로시저를 실행할 수 있게 하는 프로세스간 통신 기술
  - 서버 측은 정의한 서비스 규격에 따라 인터페이스를 구현 후 gRPC 서버를 실행하여 클라이언트의 호출 처리
  - 클라이언트 측에서는 서버와 동일한 방법을 제공하는 stub 을 사용

- **Stub 이란**
  - RPC의 핵심 개념으로 Parameter 객체를 Message로 Marshalling/Unmarshalling 하는 레이어
  - 즉, gRPC 규격에 맞게 Request Parameter 를 변환하는 레이어

> 💡 **Marshalling / Unmarshalling 이란**
>
> Marshalling 이란 메모리 상에 형상화된 객체 데이터를 적당한 다른 데이터 형태로 변환하는 과정. 컴퓨터간 데이터 전달 또는 프로그램 간 데이터 전달을 할 때 사용.
>
> 반대로 전송된 데이터를 다시 원래의 객체 모양으로 복원하는 작업은 Unmarshalling이라 함

- **gRPC 의 장점 및 특징**
  - 높은 생산성과 다양한 언어 및 플랫폼 지원
  - HTTP/2 기반의 양방향 스트리밍 -> Observer 패턴 사용
  - 성능 상 이점

※ Protocol Buffer

- **Protocol Buffer 와 proto file (protocol buffer schema 파일)**
  - gRPC 는 IDL(Interface Definition Language) 로 Protocol buffer 를 사용
  - proto file 은 프로토콜 버퍼 데이터를 일련의 Key-Value 쌍을 포함하는 작은 논리적 레코드인 message 로 구성한 파일로 Protocol buffer 의 Schema 파일
  - 프로토콜 버퍼로 작업 시 proto file 에서 직렬화 하려는 데이터 구조를 정의 (데이터 구조는 특정 언어에 종속성이 없는 형태로 정의)

> 💡 **IDL 이란 (Interface Definition Language)**
>
> IDL는 서비스를 사용하려는 클라이언트가 서비스가 제공하는 메서드와 속성, 인터페이스, 인터페이스를 알 수 있도록 일부 서비스에 대한 인터페이스를 설명하는 언어로 공급업체 또는 표준 그룹에 따라 변형

※ Protobuf 주요 문법

1. **syntax**

```
syntax = "proto3"; //proto3 버전 문법 사용
```

버전을 정하는 구문으로 proto2, proto3 가 있음.

→ gRPC 사용 시 proto3 사용

2. **Field types**

```
message Foo {
  string name = 1; //문자열 필드.
  int32 age = 2; //정수형 필드.
}
```

double, float : 실수

int32, int64 : 음수에 비효율적

uint32, uint64 : unsigned int 같은 느낌일까요

sint32, sint64 : 부호있는 정수

fixed32, fixed64 : 항상 4, 8바이트. 값이 2^28, 2^56보다 크면 효율적

sfixed32, sfixed64 : 항상 4, 8바이트

bool : 부울

string : UTF-8 혹은 7bit ASCII 이여야 함

bytes : 임의의 바이트 배열 사용가능

default값

numeric : 0

bool : false

string : 빈 문자열

byte : 빈 문자열

3. **service - RPC**

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

실제 rpc 서비스 명 정의

→ rpc {서비스명} ({Request Message}) returns ({Response Message}); 로 정의

※ webMethods gRPC Descriptor 가이드

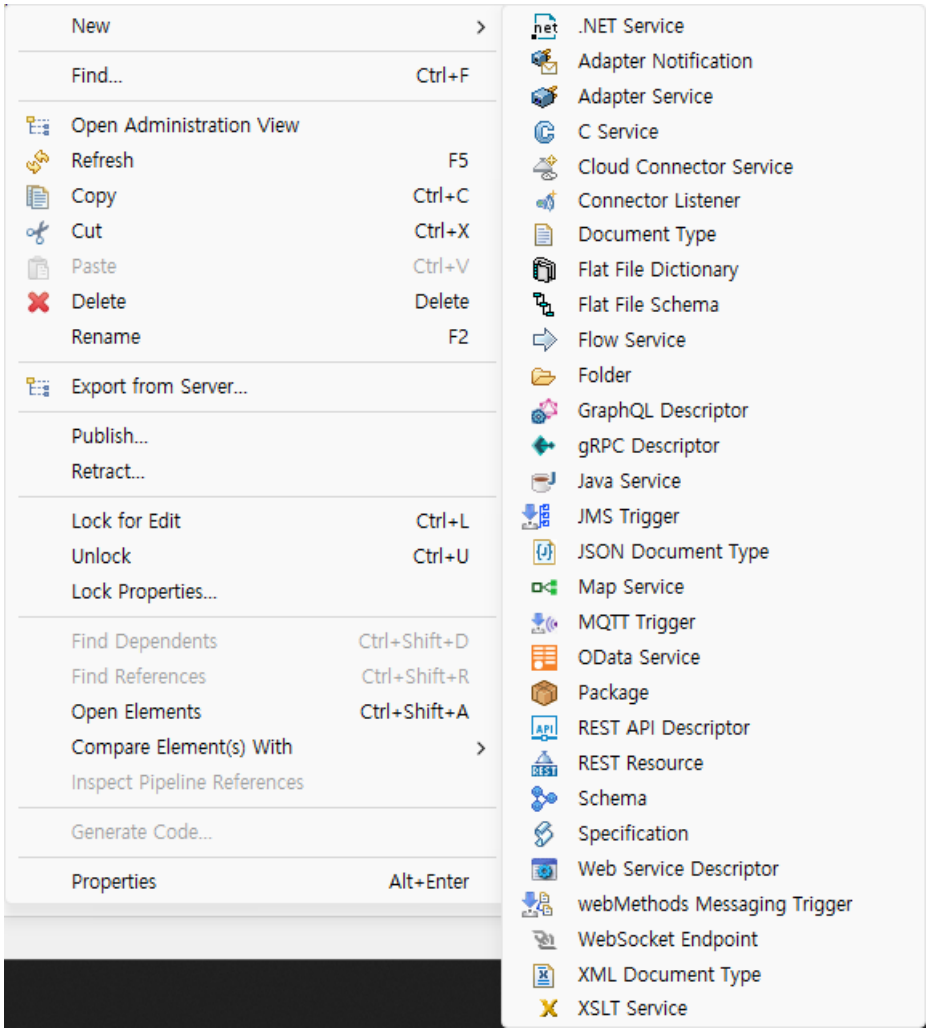
STEP 1. protofile 작성 (.proto 작성)

```
syntax="proto3";

message InputRequest {
  string Paramnum1=1;
  string Paramnum2=2;
}

message OutputResponse{
  string Resultvalue=3;
}

service CalculatorService {
  rpc Add(InputRequest) returns(OutputResponse);
  rpc sub(InputRequest) returns(OutputResponse);
  rpc Mul(InputRequest) returns(OutputResponse);
  rpc Div(InputRequest) returns(OutputResponse);
}
```

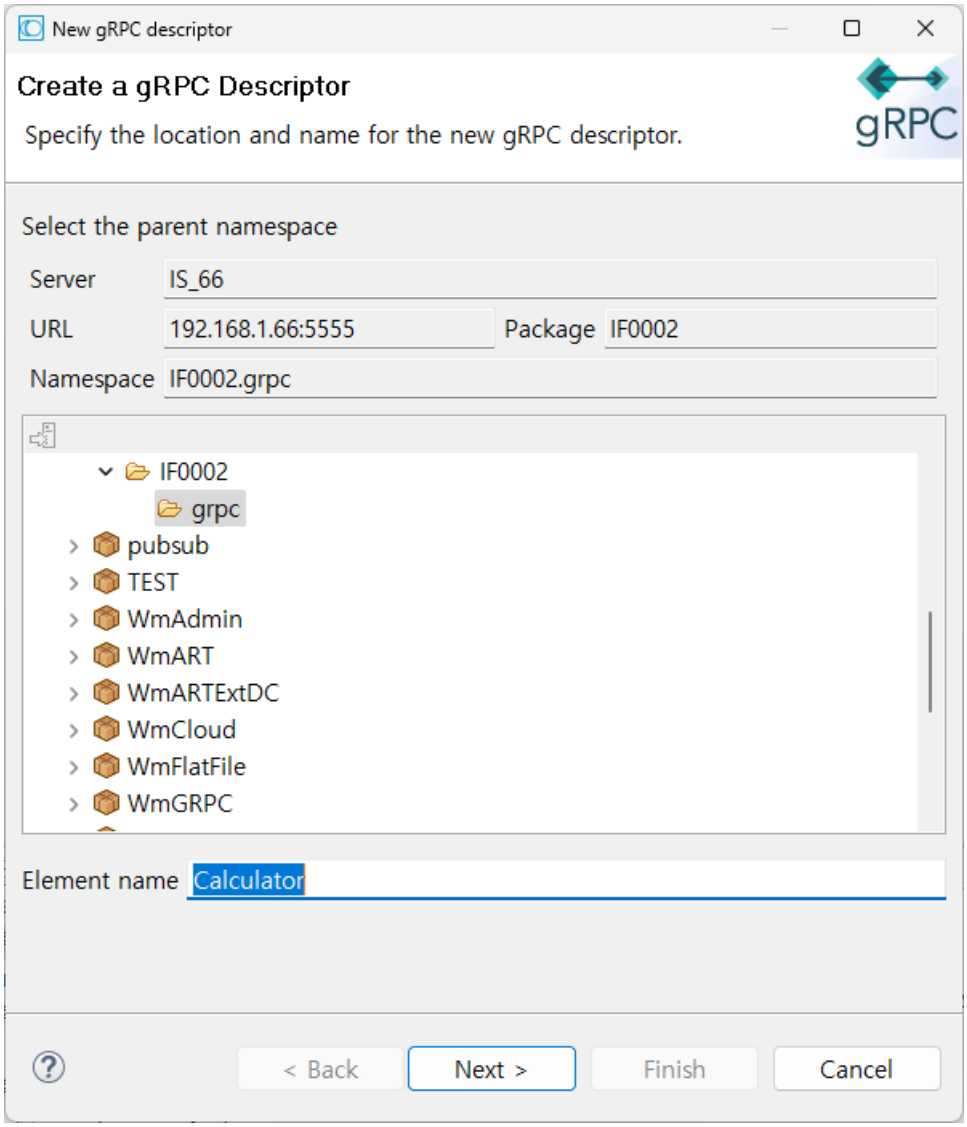
message : 객체 (Input, Output 설정)

service ≈ flowservice, 실제 구현 서비스 명과 Input, Output 값 설정

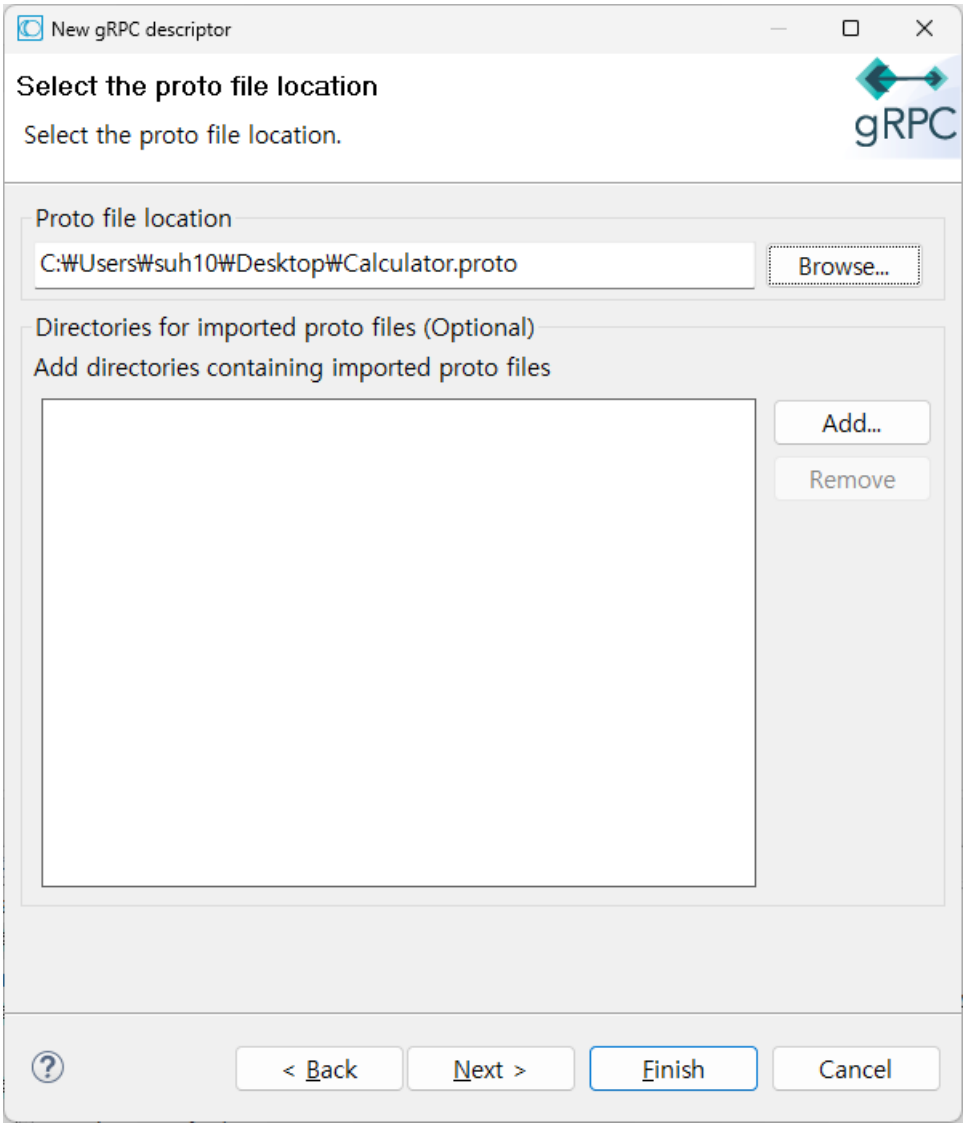
STEP 2. gRPC Descriptor 설정



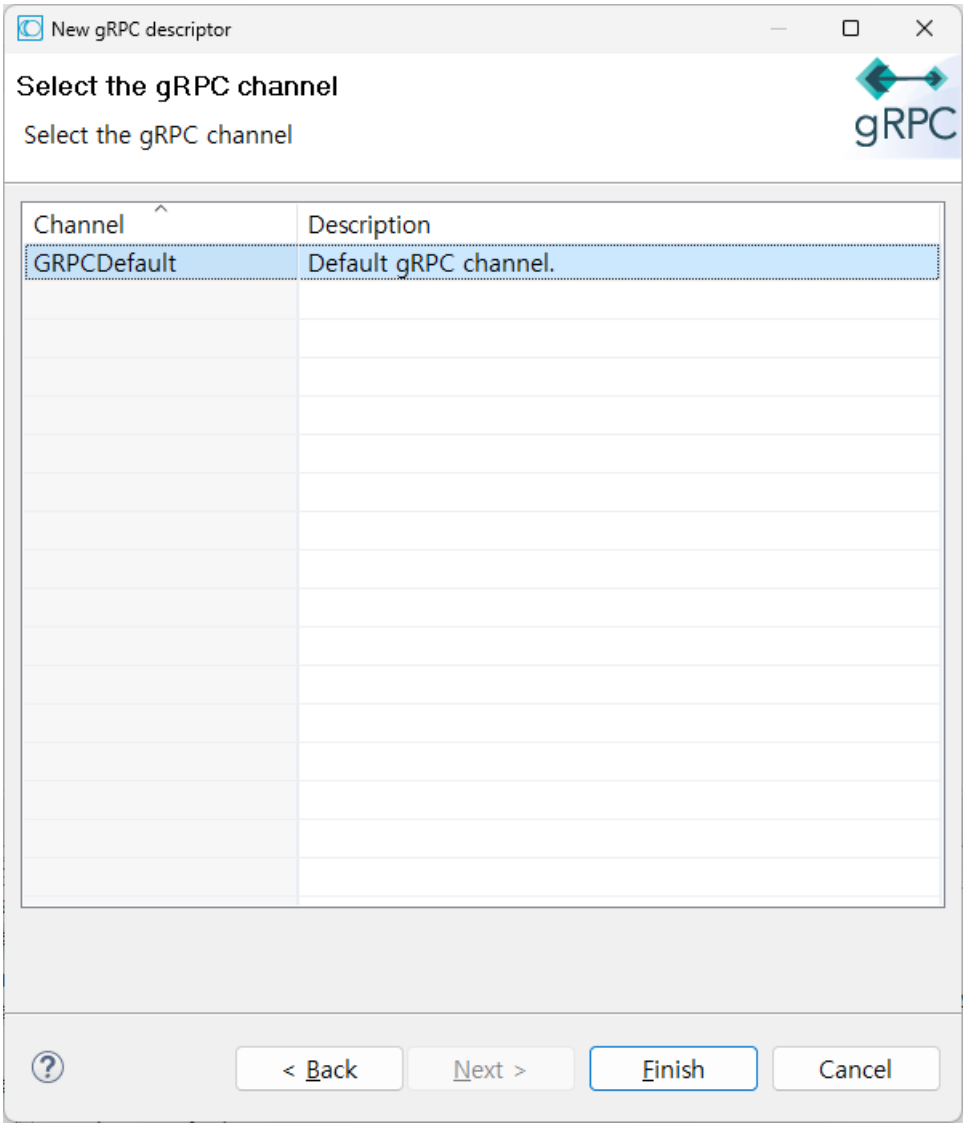
New > gRPC Descriptor 생성



Descriptor 명 입력 후 Next



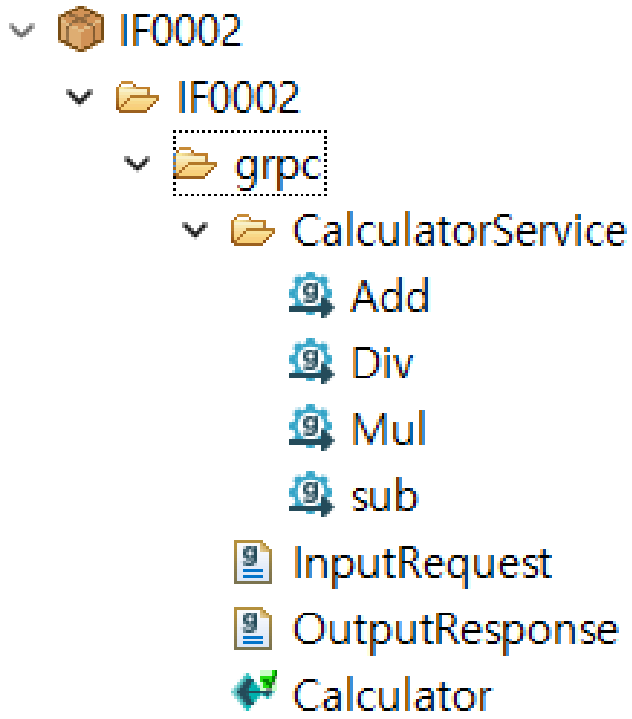
Step1 에서 생성한 예시 .proto 파일 (Calculator.proto) Import 후 Next



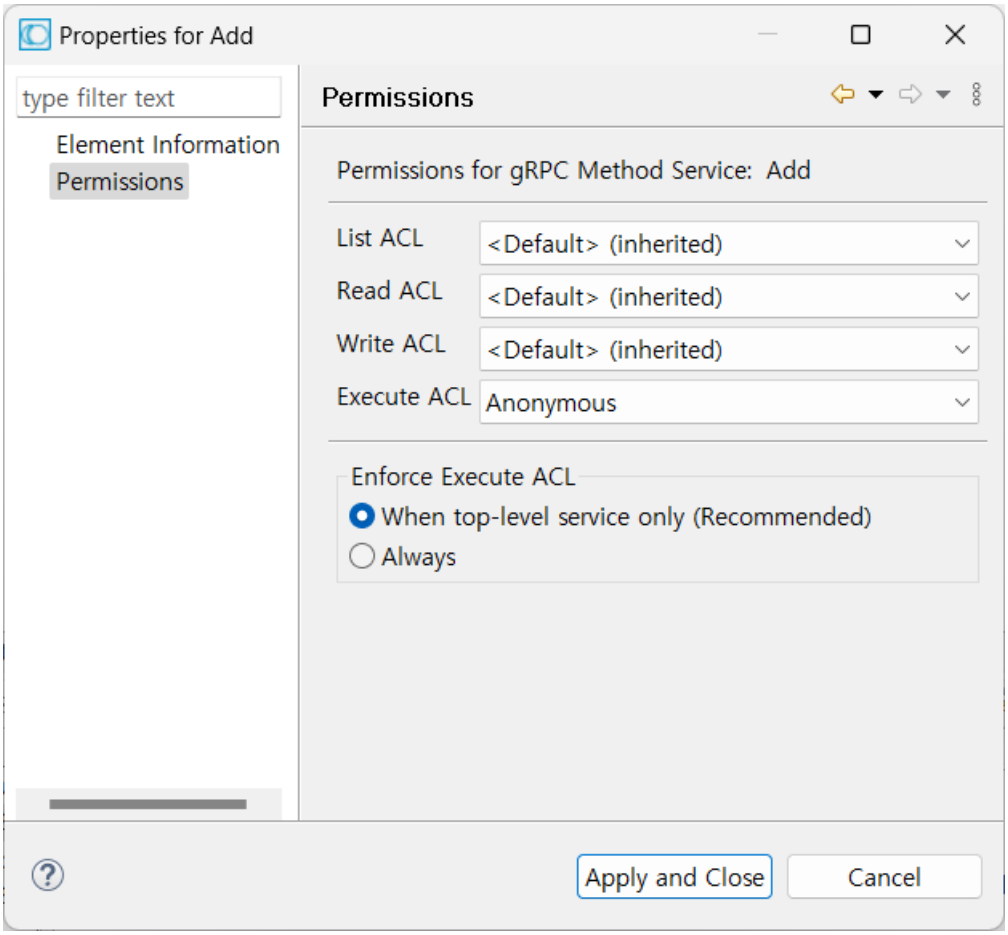
GRPCDefault Channel 설정 (Default 값) > Finish

gRPC Channel 이란?

gRPC 는 HTTP 2.0 기반에 HTTP Streaming을 이용해서 데이터를 보내며, HTTP 2.0의 경우에는 커넥션을 계속 유지하는 메커니즘이기 때문에, gRPC는 보통 하나의 Connection을 맺어 놓고 이 Connection을 통해 계속 메세지를 보내는 방식을 이용하는데, 이 Connection (TCP Connection)을 Channel이라 함.

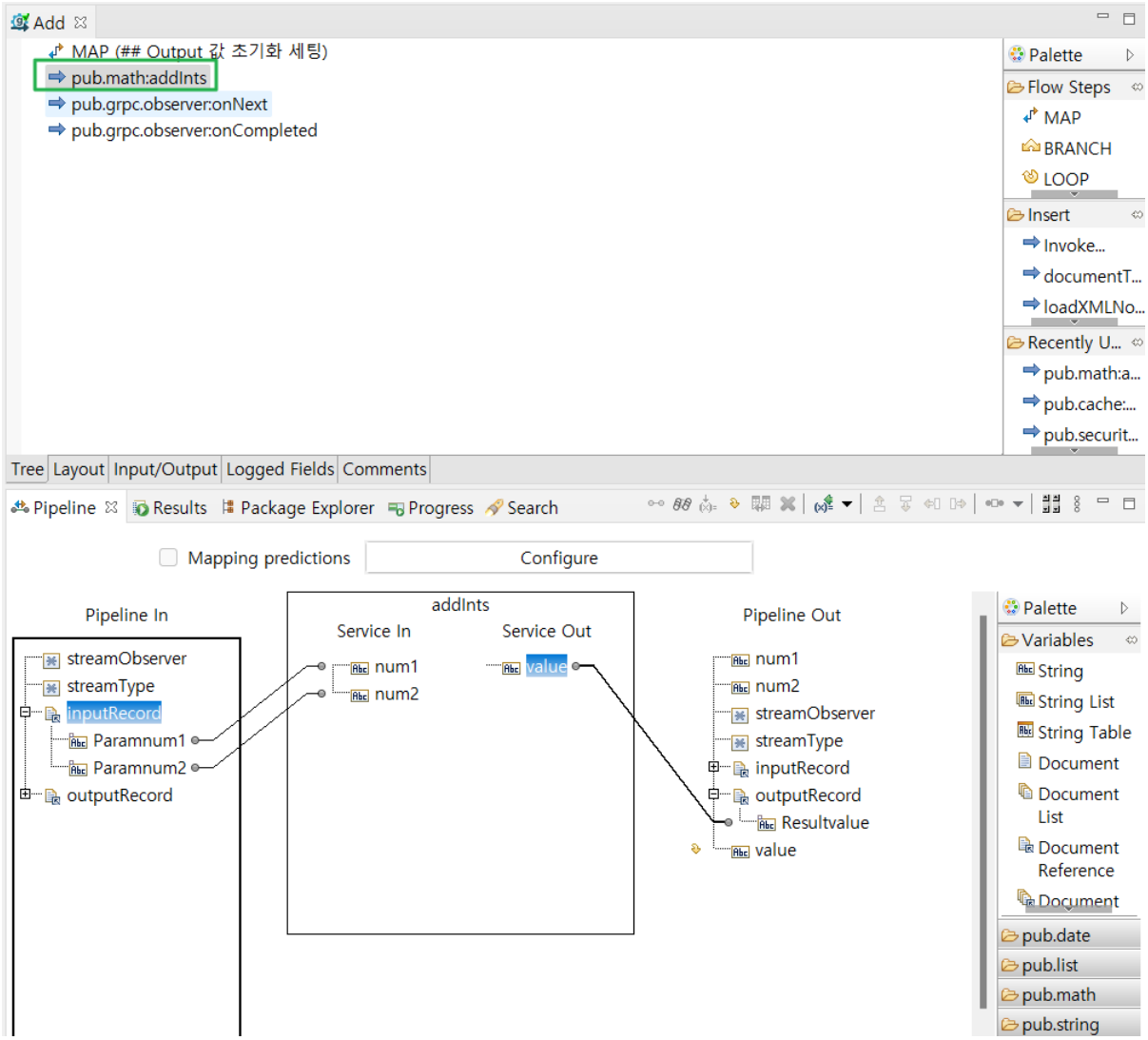


생성 된 gRPC Descriptor (기본 설정으로 Unlock 설정이 되어있어, 수정 시 Lock for Edit 설정 필요)



외부 Client 에서 호출 가능하도록 **Service** 와 **Provider Descriptor** 두 컴포넌트의 Permission을 Anonymous 로 설정 필요

STEP 3. Service 구현



실 구현 소스는 pub.grpc.Observer:onNext 와 pub.grpc.Observer:onCompleted 위에 구현
(위 예시는 addInts 서비스를 구현)

gRPC 관련 Built-in-Service (문서 참고)

pub.grpc.observer:onCompleted : gRPC 서버가 Stream을 성공적으로 수행했음을 알리는 서비스

pub.grpc.observer:onError : gRPC Flowservice의 에러에 대하여 gRPC 클라이언트에 알리는 서비스

(TRY-CATCH 문에서 CATCH 문에 사용)

pub.grpc.observer:onNext : IData (Document) 를 gRPC 응답으로 변환하여 gRPC 클라이언트로 반환 하는 서비스

Observer 패턴 이란?

어떤 객체의 상태가 변할 때 그와 연관된 객체 들에게 알림을 보내는 디자인 패턴

즉, gRPC 에서는 HTTP/2 기반으로 통신하며 양방향 스트리밍이 가능하기에 서버와 클라이언트가 서로 동시에 데이터를 스트리밍으로 주고받을 때 서버 클라이언트 간 데이터에 대한 알림을 보내기 위해 사용하는 디자인 패턴

[참고]

서버 측 gRPC 서비스 구현 예시 (JAVA)

```java
@GrpcService
public class HelloServiceImpl extends HelloServiceImplBase {

  // unary
  @Override
  public void hello(HelloRequest request, StreamObserver<HelloResponse> responseObserver) {

    String greeting = request.getFirstName() + "," + request.getLastName();

    HelloResponse response = HelloResponse.newBuilder()
        .setGreeting(greeting)
        .build();

    responseObserver.onNext(response);
    responseObserver.onCompleted();
  }
```

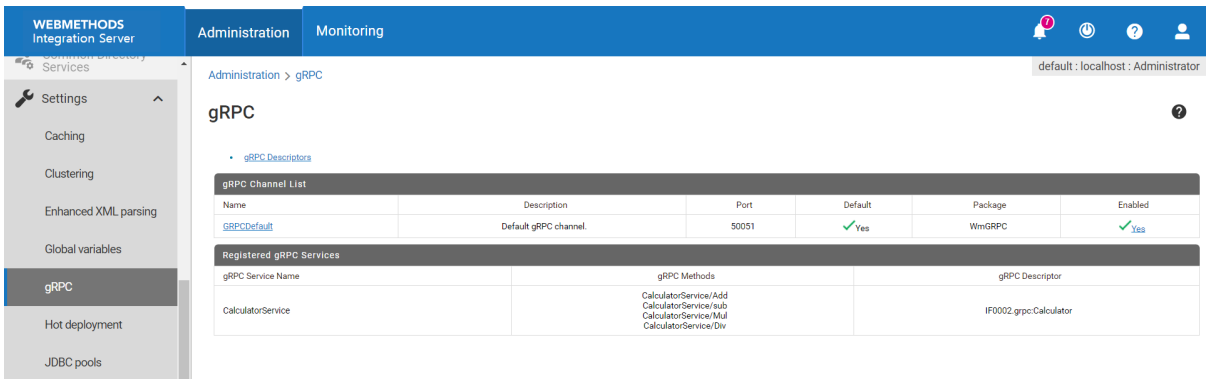
Request 파라미터를 받아 처리 후

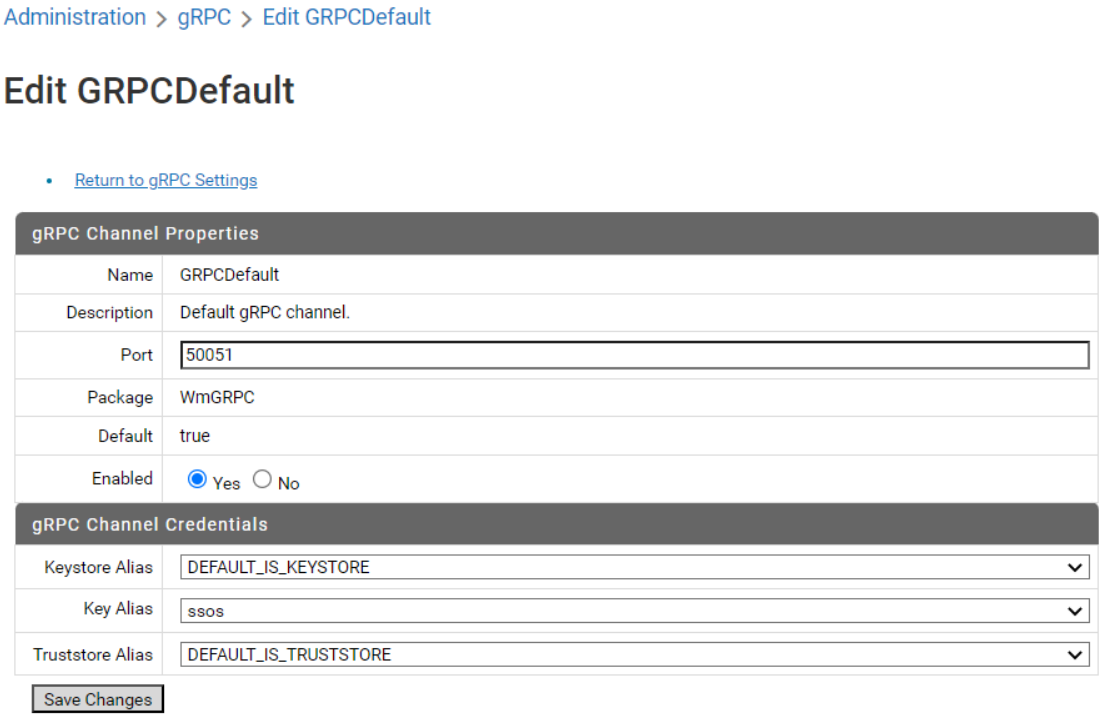
gRPC protofile에 정의 된 Response 방식으로 response 메시지를 만들어

responseObserver.onNext(response) - Response 처리

responseObserver.onCompleted(); - 성공적으로 처리 됨을 클라이언트 측에 알림

STEP 4. gRPC Admin Page 확인



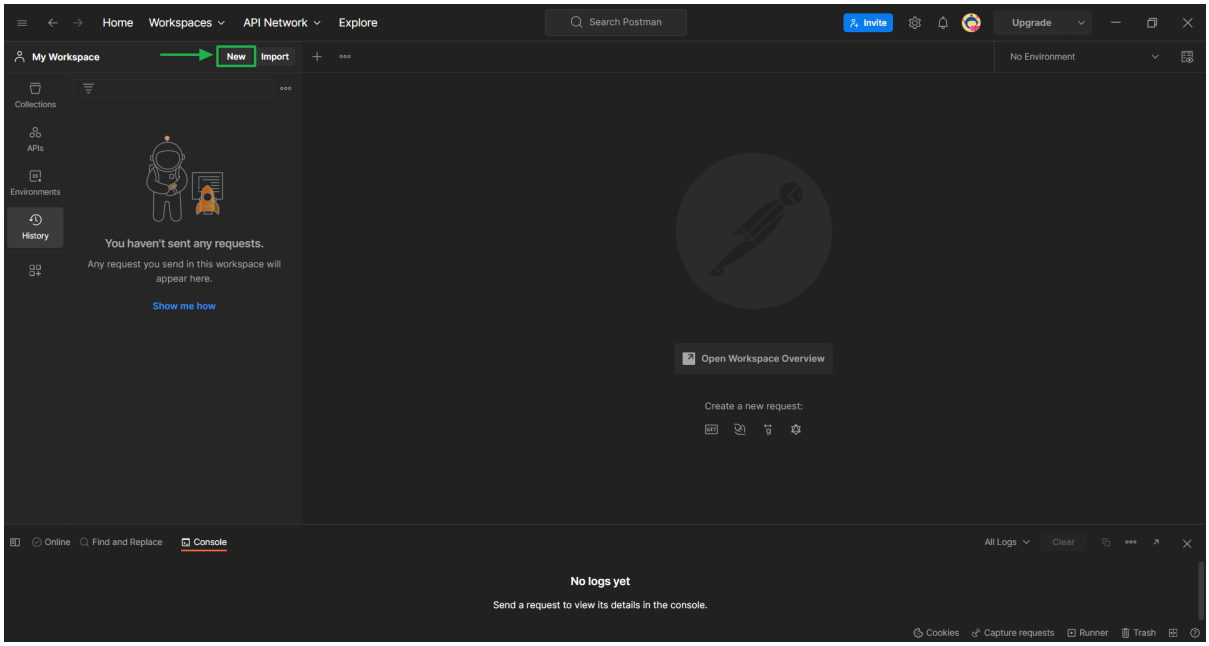


Settings > gRPC 에서 등록 된 Descriptor 와 gRPC Channel 의 Port 를 확인

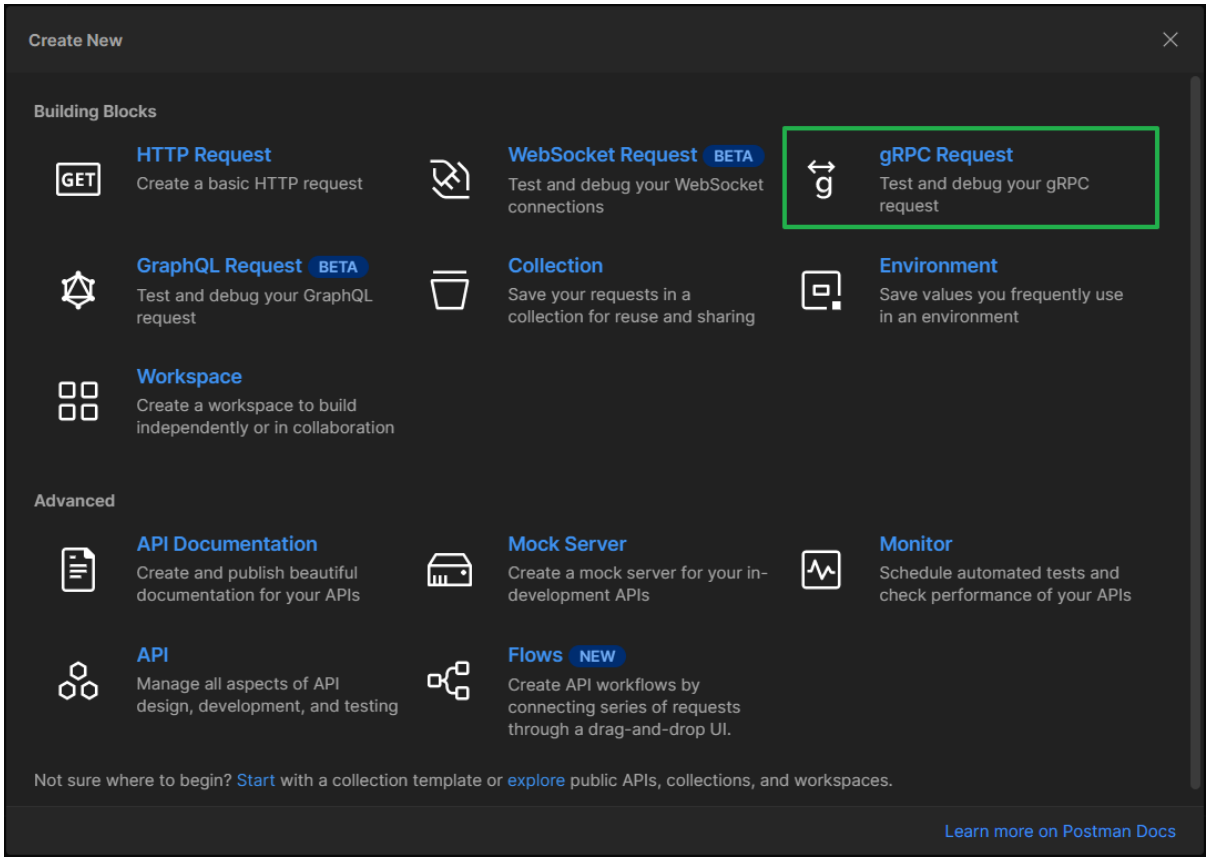
Channel 클릭 후 Properties 수정 가능 (Port, TCP Connection 시 사용 되는 인증서 설정 가능)

STEP 5. Postman (Client)을 사용하여 gRPC 호출

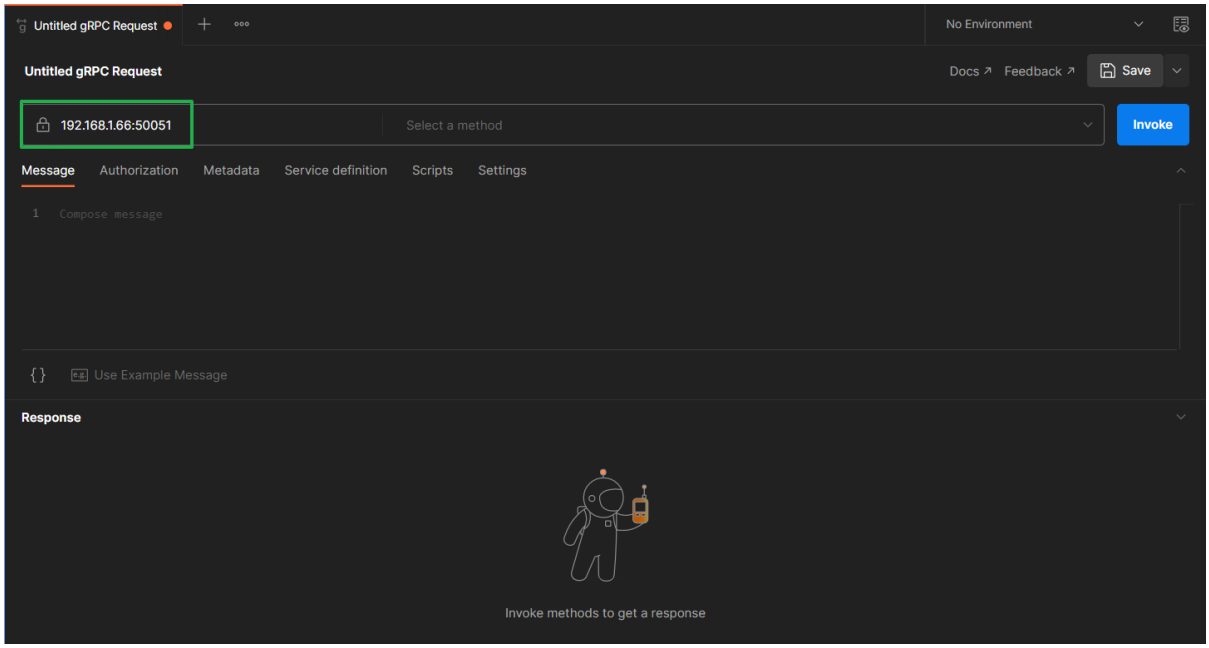
gRPC Endpoint : {webMethods IP} : 50051 (Default)



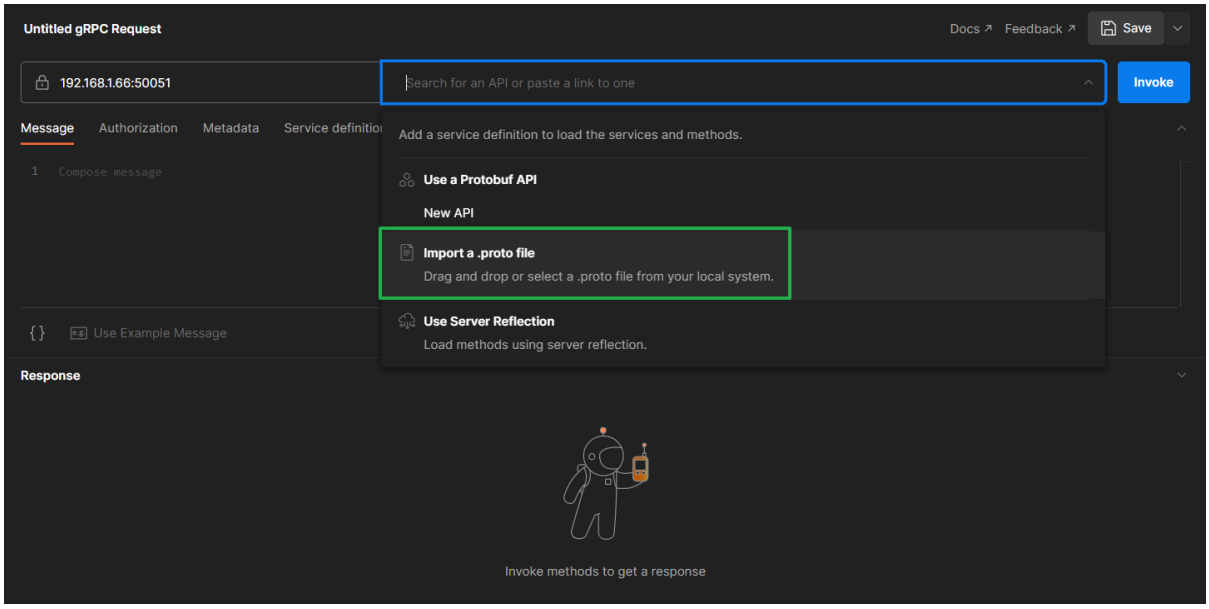
좌측 상단 New 클릭

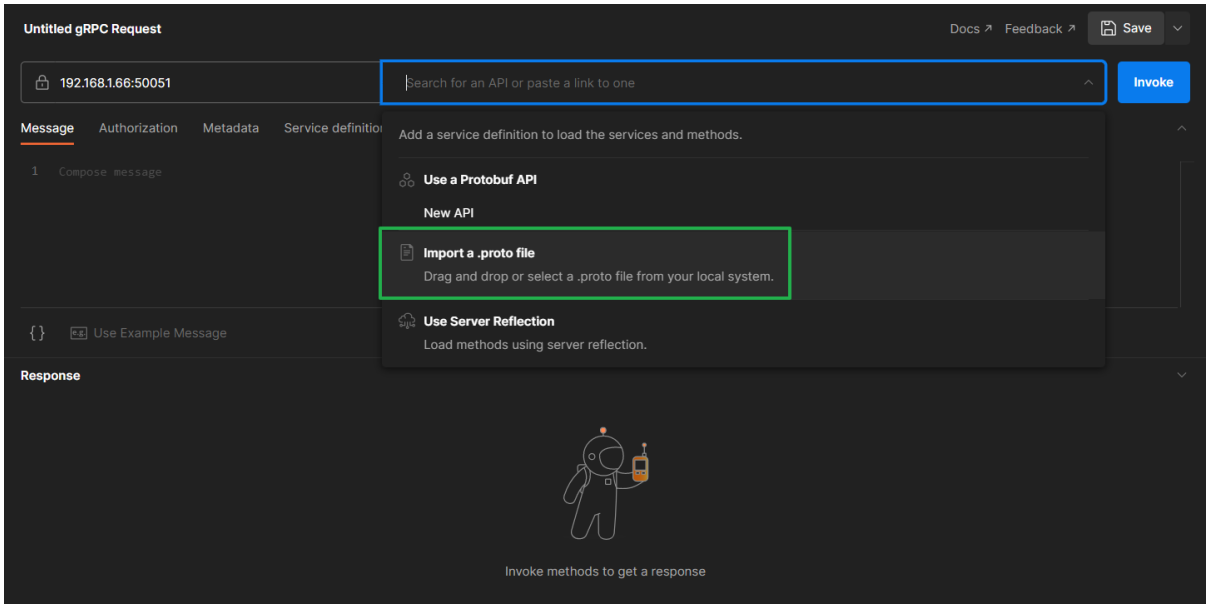


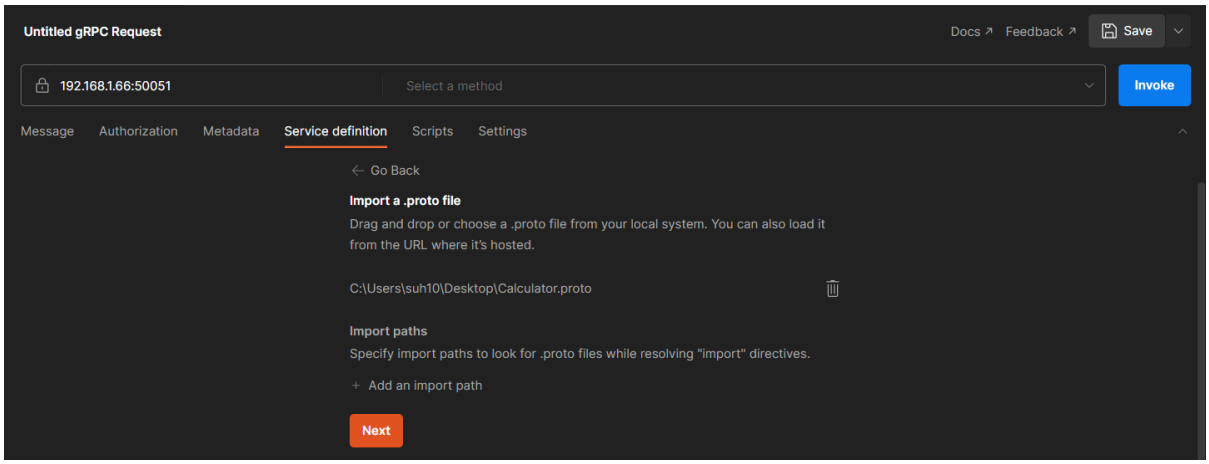
gRPC Request 클릭

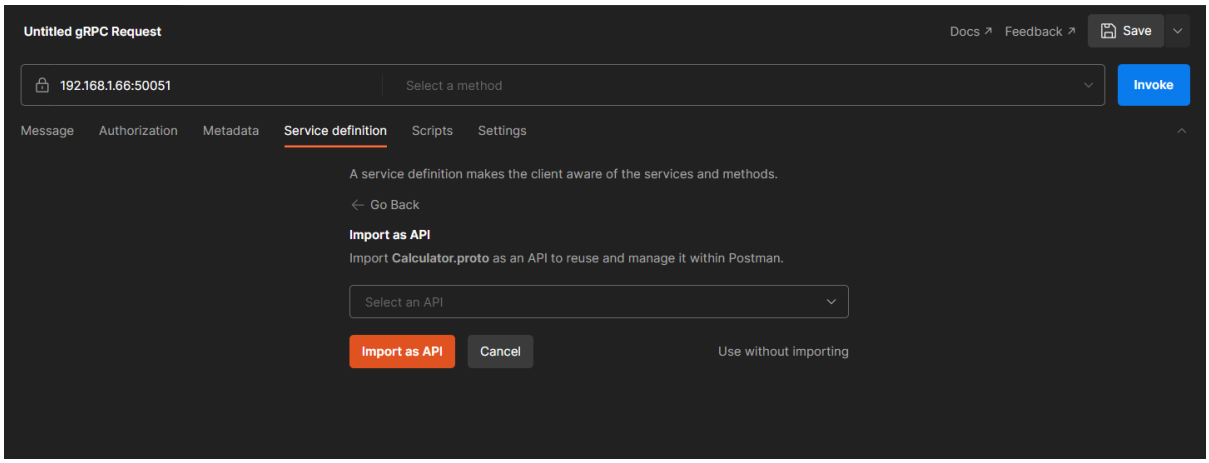


gRPC Endpoint 입력, Enable TLS (좌물쇠 아이콘 클릭)

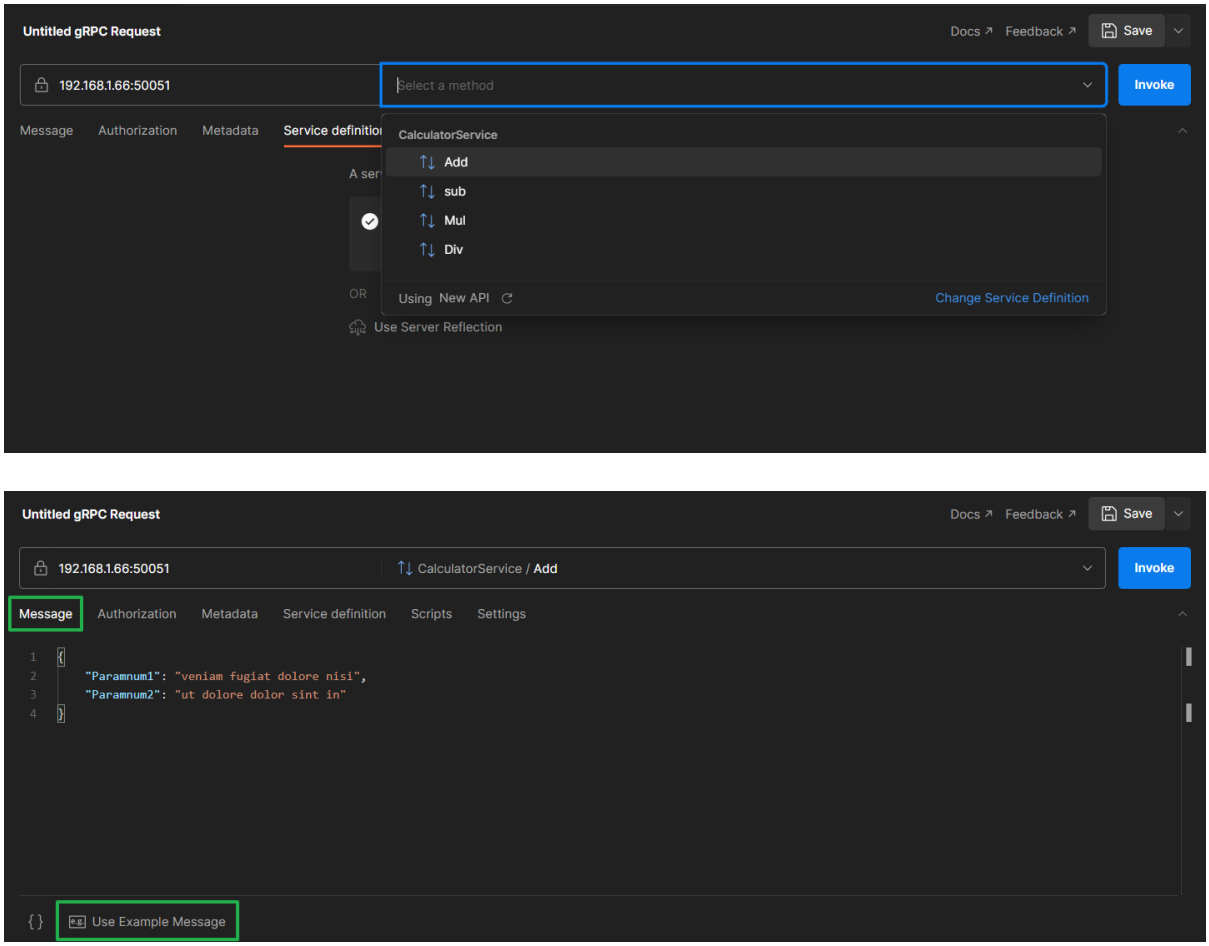




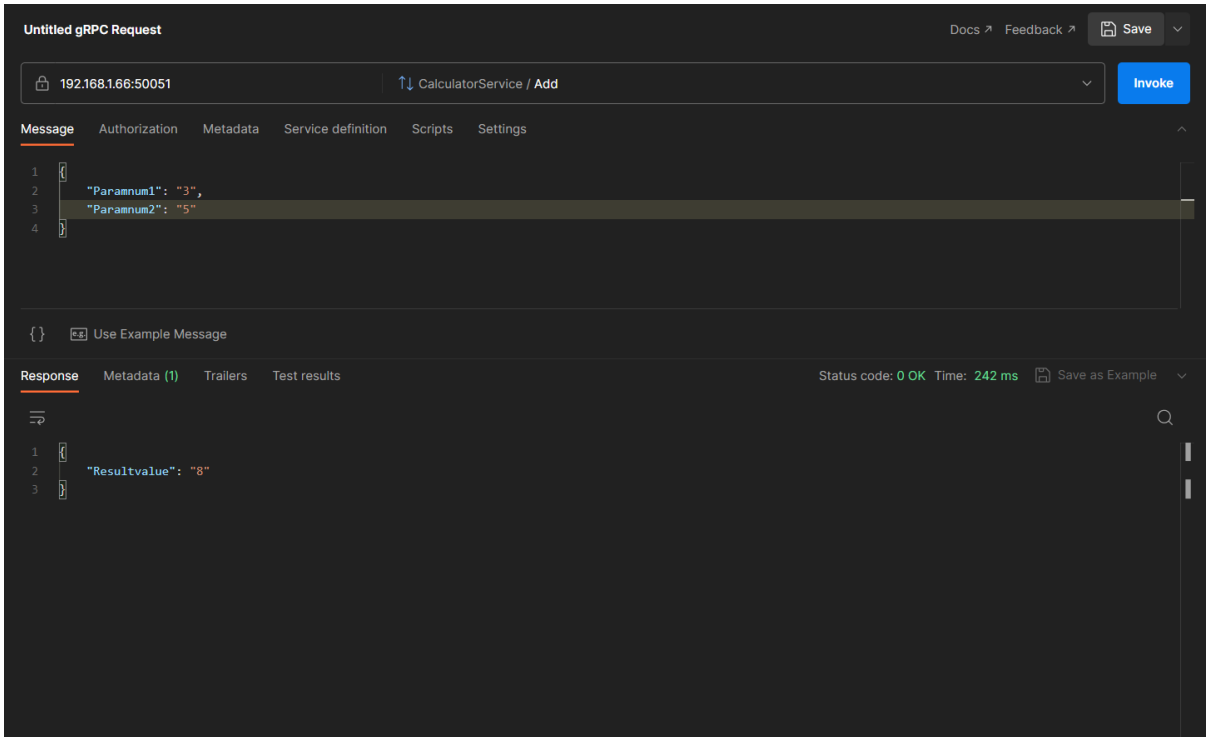




Import a .proto file 클릭 -> step 1 에서 작성한 .proto 파일 import 후 Next -> Import as API



Add Service 선택 후 Message 탭에서 Use Example Message 를 클릭하여 예시 Input Message 를 생성



Request Message 값 수정 후 Invoke 하여 Response 확인