webMethods GraphQL Guide

> **References**
>
> - **Software AG Tech Forum (GraphQL Demo)**
> - **GraphQL.org (GraphQL Learn)**
> - **Kakao Blog (GraphQL 개념 잡기)**
> - **Vlog (GraphQL Schema)**

- **GraphQL 이란 (gql)**
  - SQL 과 비슷한 일종의 쿼리 언어
  - 웹 클라이언트가 데이터를 서버로부터 효율적으로 가져오도록 하기 위해 개발 된 쿼리 언어
  - gql 문장은 주로 클라이언트 시스템에서 작성되고 호출
  - HTTP POST 메서드와 웹소켓 프로토콜 활용, 필요에 따라서 TCP/UDP 를 활용하거나 L2 이더넷 프레임을 활용할 수 도 있음.

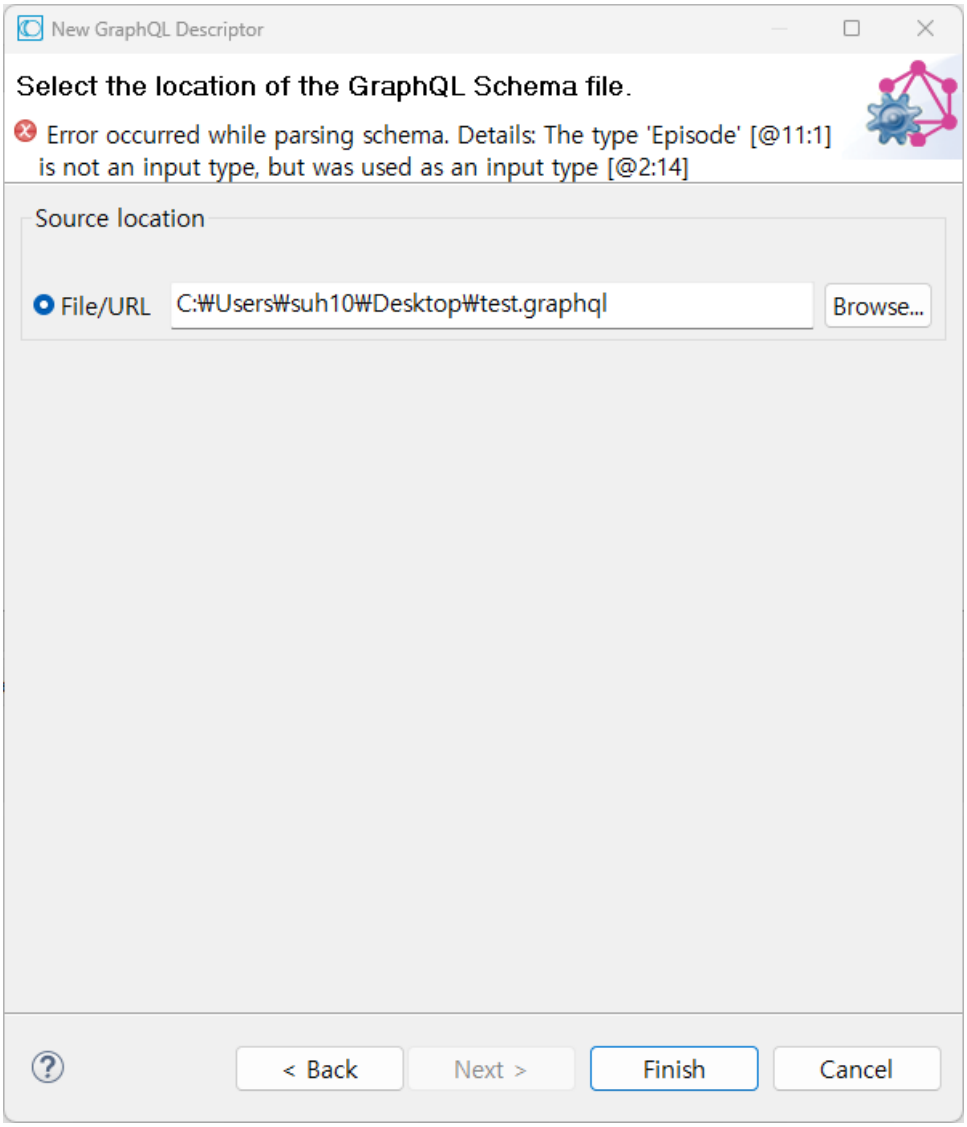
- REST API 와 달리 단 하나의 Endpoint만 존재하며, 쿼리 조합을 통해 응답 값이 결정된다.
  - REST API 는 여러 번 호출 해야 하는 경우에도 gql API는 한번의 호출(쿼리)로 처리 가능

- **GraphQL의 구조**
  - 쿼리 (Query) 타입 : 데이터를 Read 하는데 사용 (필수)
  - 뮤테이션 (Mutation) 타입 : 데이터를 Create, Update, Delete 하는데 사용 (선택)
  - 스키마 (Schema) : 사용할 쿼리나 뮤테이션, Type 등을 정의, *.grqphql 확장자 파일로 저장
  - 리졸버 (Resolver) : 실제 기능 구현 부, 스키마에 정의 된 함수를 구현

- **GraphQL Schema**
  - 모든 GraphQL 서비스는 쿼리가 들어오면 해당 스키마에 대한 유효성 검사 후 실행되는 과정을 거침
  - Schema 의 기본 구성 요소는 객체 타입
    - Character - 객체 타입 명
    - name : String ! - 필드 명 : 필드 타입 (! = non-nullable)
    - [Episode] - Episode 객체의 배열 (Array)

Unsupported:

Subscription root operation.
Custom scalar data type.
Directives.



Note. webMethods 는 Custom Scalar data type 을 지원하지 않음. 위 Episode 같은 구조로 스키마 생성 시 파싱 에러 발생

- Query Type & Mutation Type 예시

```
type Query {
  hero(episode: Episode): Character
  droid(id: ID!): Droid
}
```

  - hero (episode:Episode): Character - hero 라는 쿼리에 Input 파라미터는 episode (Episode 객체), Output은 Character 객체
  - Query 는 Read, Mutation 은 Create,Update,Delete

- Enum Type

```
enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}
```

    - 열거형 타입으로 열거된 특정한 값들로 제한되는 객체

  - 이외 타입 참고 (interface, union, input)

- **인트로스펙션 (Introspection) - 참고**
  - 기존 서버-클라이언트 협업 방식에서 연동 규격서라고 불리우는 API 명세서를 주고받는 절차가 GraphQL에서는 이러한 인트로스펙션용 쿼리가 따로 존재하여 일반 gql 쿼리문을 작성하듯이 작성하여 명세서를 확인
  - 대부분의 gql 라이브러리에서 쿼리용 IDE를 제공하여 IDE 내부에서 인트로스펙션을 제공하며 직접 쿼리, 뮤테이션,필드 스키마를 확인 가능

※ webMethods GraphQL Provider 생성 및 호출 가이드

STEP 1. Schema 파일 작성

테스트 스키마 작성 예시 (Calculator.graphql)

```
type Query{
  add(number1:String,number2:String):Output
  sub(number1:String,number2:String):Output
  div(number1:String,number2:String):Output
  mul(number1:String,number2:String):Output
}

type Mutation{
  savePipeline(input1:String,input2:String):Filename
}

type Output{
  output:String
}

type Filename{
  filename:String
  status:String
}
```

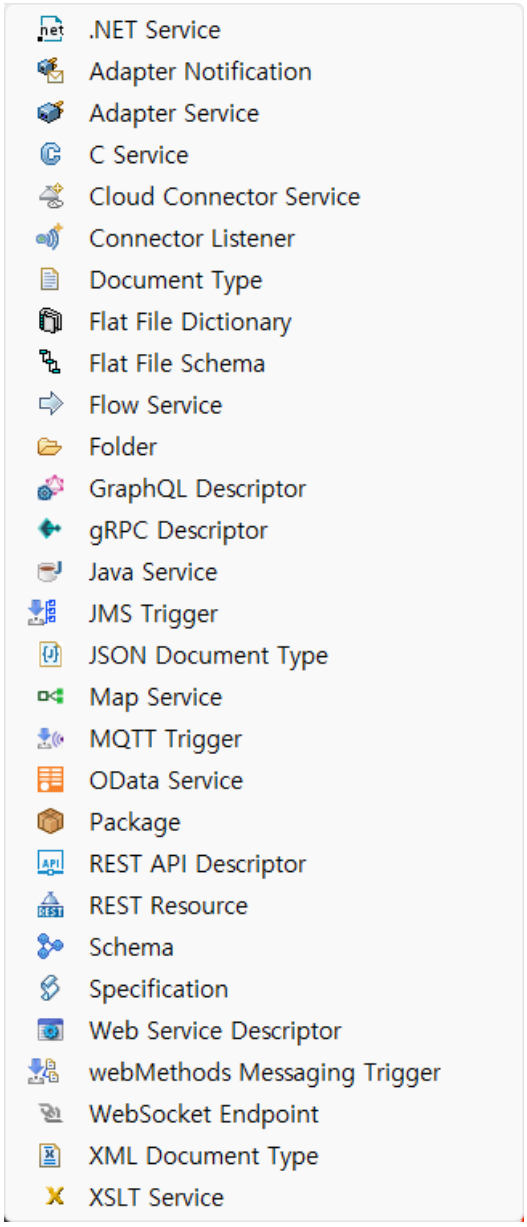
Query, Mutation (= Flowservice 로 변환)

→ Query, Mutation명 + Resolver 라는 이름의 Flowservice 가 생성 됨

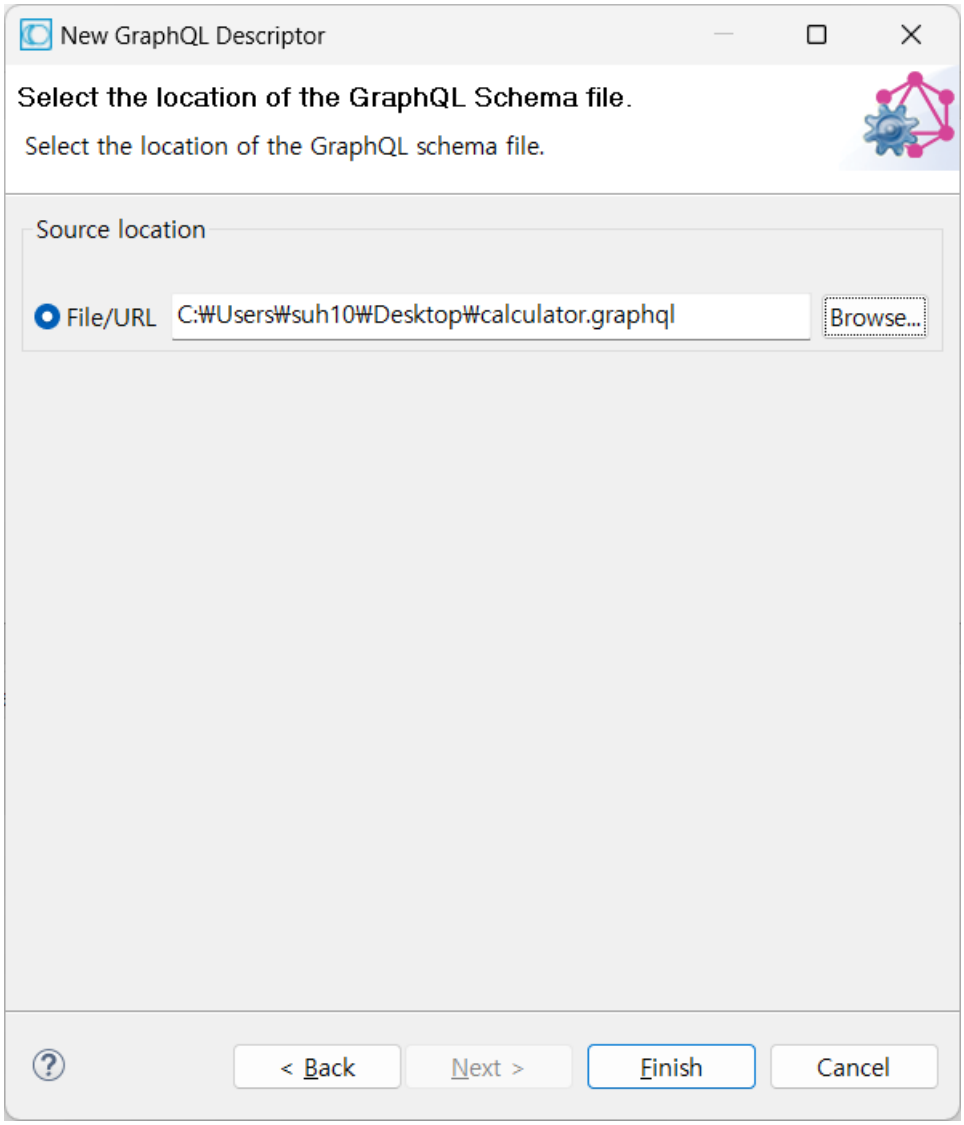
객체 Type (= DocumentType 로 변환)

→ 위 Output, Filename에 정의 된 필드와 Data Type에 따라 매핑되어 생성

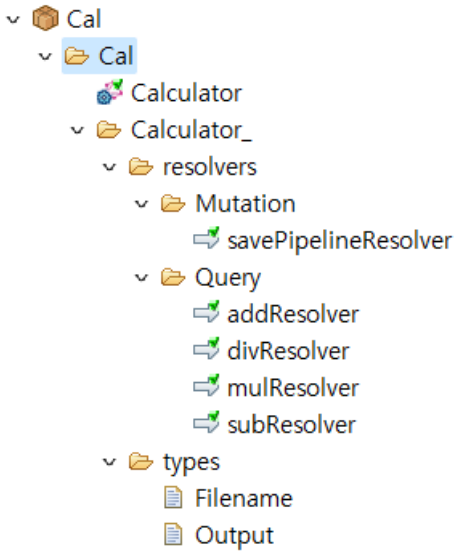
STEP 2. GraphQL Descriptor 생성



생성 할 폴더에 new > GraphQL Descriptor 생성



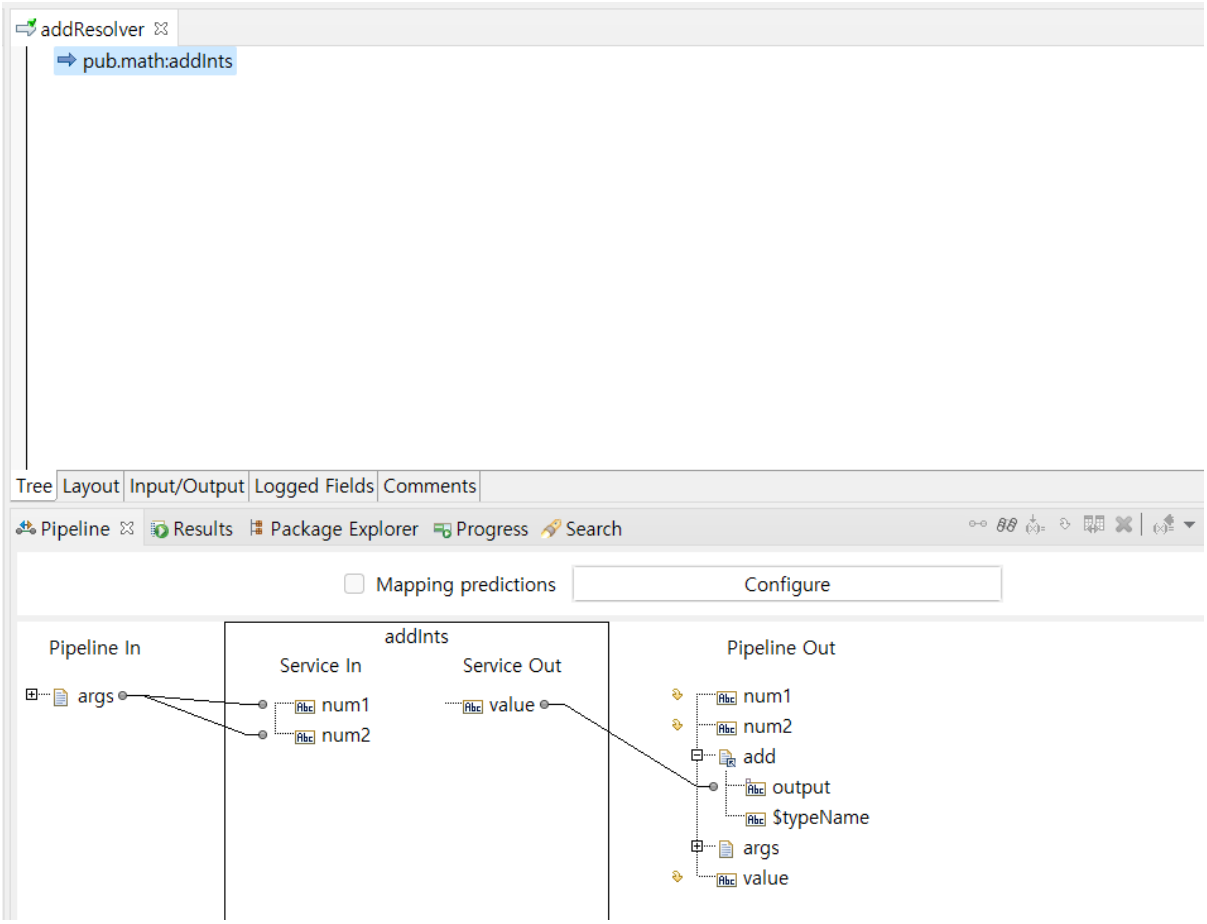
step1 에서 생성 한 graphql 파일을 Import 또는 URL 입력



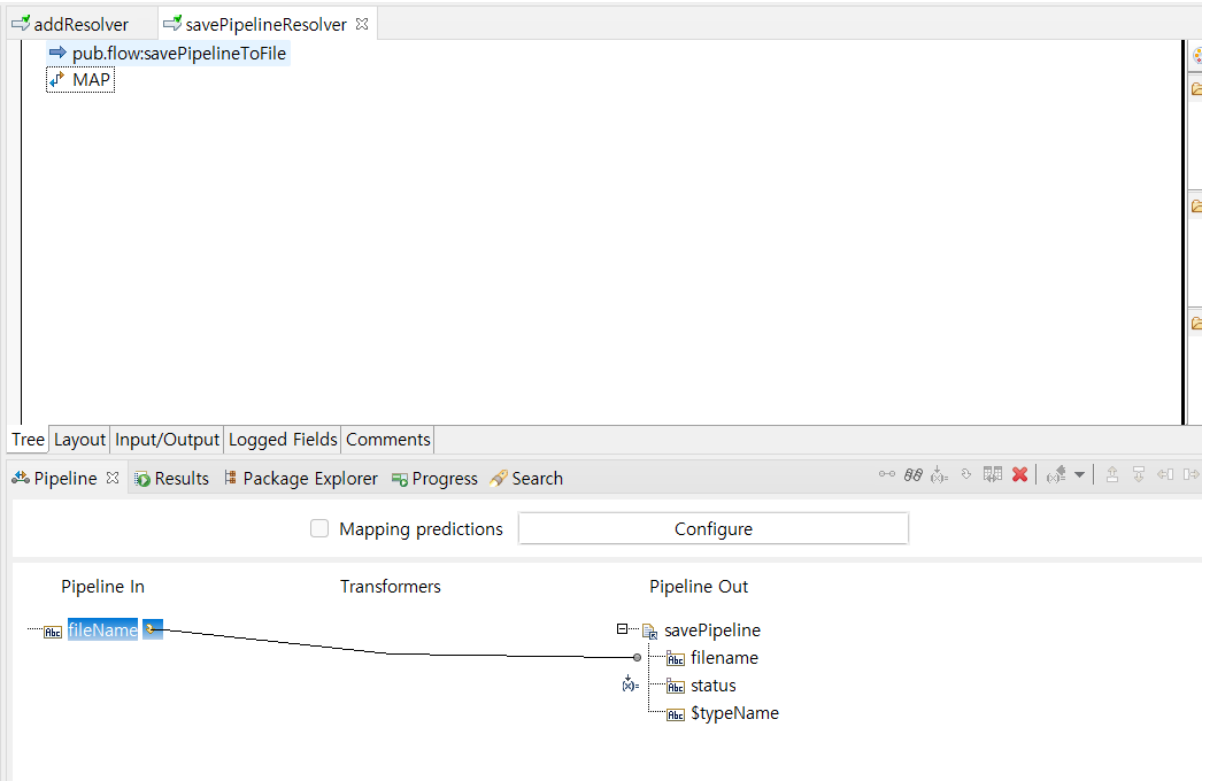
생성 완료 된 GraphQL Descriptor 예시

STEP 3. Resolver (FlowService) 개발 (addResolver, savePipelineResolver)

1. addResolver



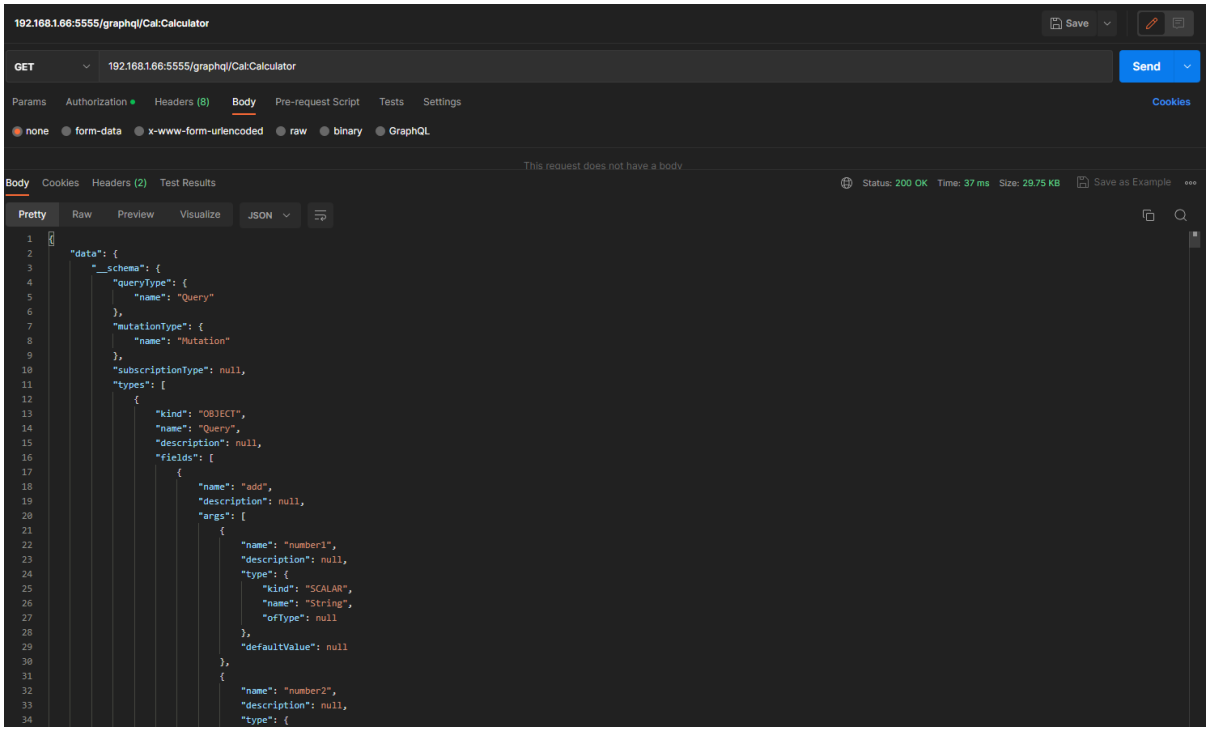
2. savePipelineResolver



STEP 4. POSTMAN 호출 (v7.2 이상)

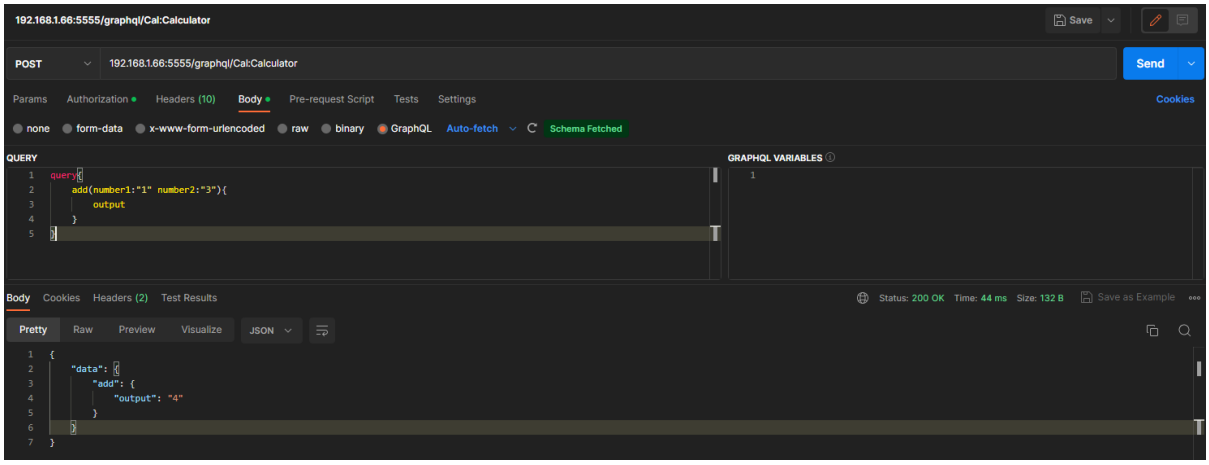
호출 Endpoint: {webMethods IP : Port}/graphql/{Folder Name}/{GraphQL Descriptor Name}

[GET] 호출 - Introspection 응답



[POST 호출] - Query, Mutation 호출

1. Query 호출

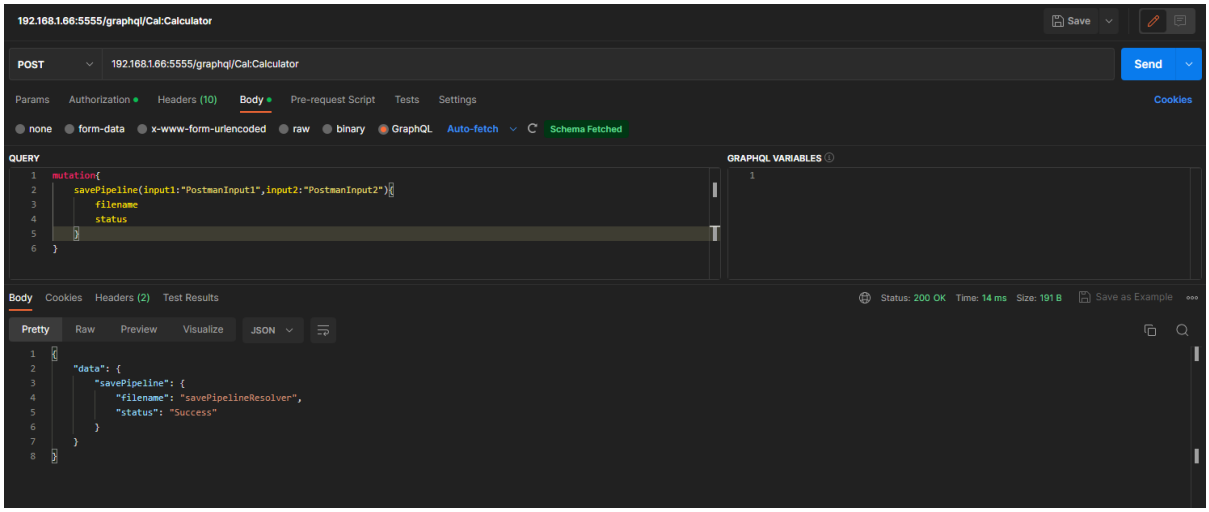



```
query{
    add(number1:"1" number2:"3"){  -> 호출 할 Resolver 명과 Request 파라미터 입력
        output    -> output 값 입력
    }
}
```

Response

```
{
    "data": {
        "add": {
            "output": "4"
        }
    }
}
```

2. Mutation 호출




```
mutation{
    savePipeline(input1:"PostmanInput1",input2:"PostmanInput2"){ -> 호출 할 Mutation 명과 Request 파라미터 값 입력
        filename -> output 값 입력
        status
    }
}
```

Response

```
{
    "data": {
        "savePipeline": {
            "filename": "savePipelineResolver",
            "status": "Success"
        }
    }
}
```