Implementation of REST API vs GraphQL in Microservice Architecture
[M21AIE208 and M21AIE245]

Introduction

In this project we are testing and comparing a web application created on python built using a **REST** framework and **GraphQL** framework

REST	GraphQL
REST stands for Representational State Transfer, and it is an architectural style that defines a set of rules and constraints for creating web services. REST uses HTTP methods, such as GET, POST, PUT, and DELETE	GraphQL, on the other hand, is a query language and a specification for creating APIs. GraphQL uses a single endpoint, usually /graphql, to handle all requests. Instead of using HTTP methods
The Request will fetch all the records sent by the server	One of the main advantages of GraphQL over REST is that it allows the client to specify exactly what data it needs
Extra effort is required to create the documentation around the REST api	GraphQL is that it has a self-documenting schema

Components of GraphQL application

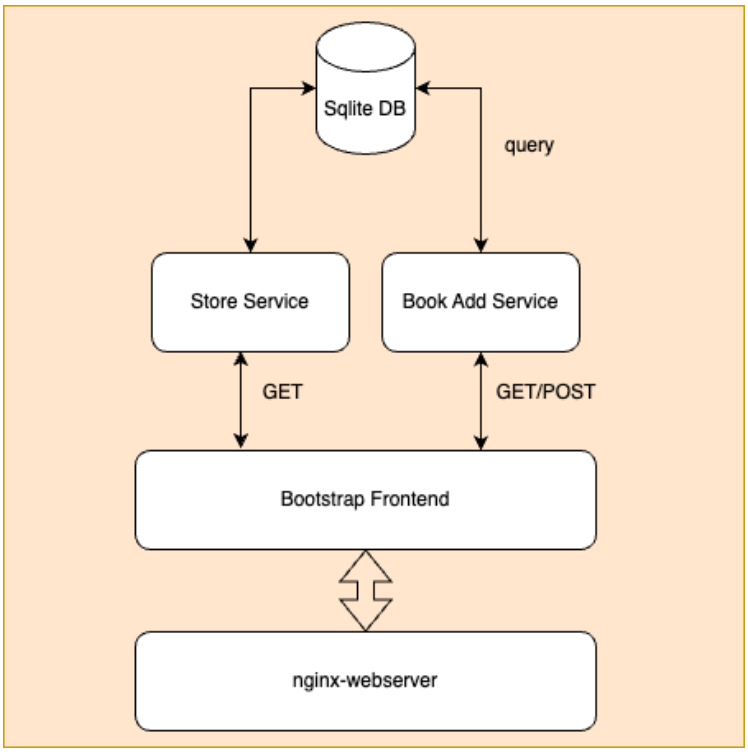
1. **Flask**: A simple framework for building web servers in Python
2. **Ariadne**: A library for using GraphQL applications
3. **Flask-SQLAlchemy**: An extension for Flask that makes it easier to use SQLAlchemy (an ORM) within a Flask application. SQLAlchemy allows us to interact with SQL databases using Python.

Application Overview

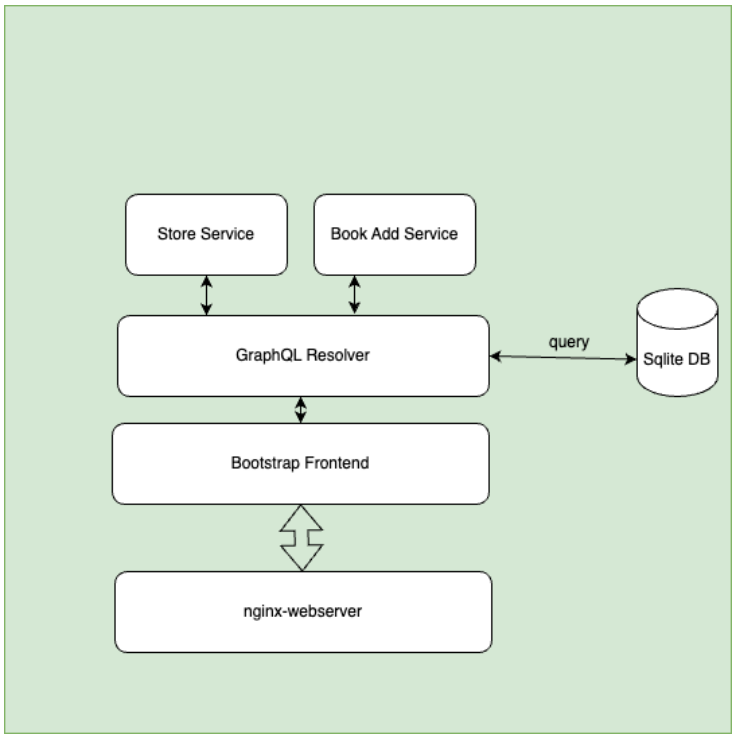
We are trying to create an online Library that has functionality to add the books, their description, author name and the genre type of the book. Then we have a book browser that helps to find the book.

We will be building the application using **Python Flask** framework for backend, **Ariadne for GraphQL** framework, **HTML Bootstrap** for frontend and a light weight database like **Sqlite**

High Level Architecture of REST based application



High Level Architecture of GraphQL based application



GraphQL has its own language, the GraphQL Schema Definition Language (SDL), which is used to write GraphQL schemas

```
schema {
    query: Query
}

type Post {
    id: ID!
    book_title: String!
    description: String!
    created_at: String!
    author: String!
    genre: String!
}

type PostResult {
    success: Boolean!
    errors: [String]
    post: Post
}

type PostsResult {
    success: Boolean!
    errors: [String]
    post: [Post]
}

type Query {
    listPosts: PostsResult!
    getPost(id: ID!): PostResult!
}
```

Fig 1 GraphQL schema for Library

Fetching Data in GraphQL

When working with REST, we usually fetch data by making HTTP GET requests to various endpoints. GraphQL works a little differently. We have a single endpoint, from where the client can request all the data that it needs. The client does this by posting a *query.* We also right a resolver function to fetch all the Post items

```
from .models import Post
2 usages
def listPosts_resolver(obj, info):
    try:
        posts = [post.to_dict() for post in Post.query.all()]
        # print(posts)
        payload = {
            "success": True,
            "post": posts
        }
    except Exception as error:
        payload = {
            "success": False,
            "errors": [str(error)]
        }
    return payload
```

Fig 2 Resolver function

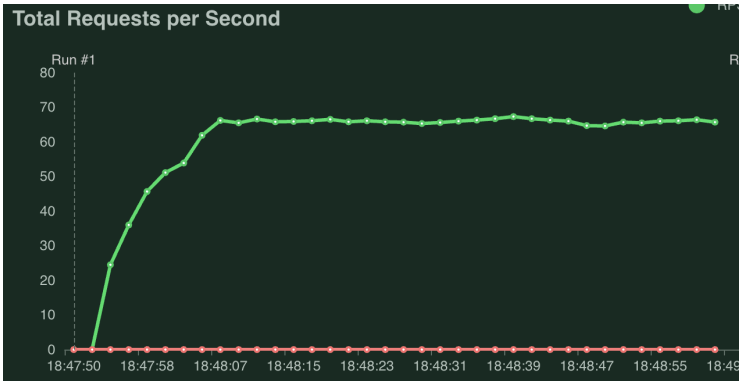
Binding the Resolver

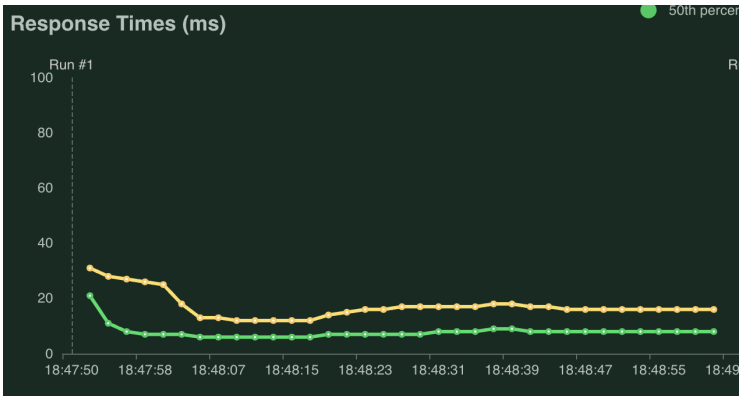
Will use Ariadne, which is a Python library for implementing GraphQL servers. Schema-first: Ariadne enables Python developers to use schema-first approach to the API implementation. Once we write a resolver, we need to tell Ariadne which field it corresponds to from the schema, so we need to bind the `listPosts_resolver` function to the field `Post` in our GraphQL schema

Testing performance with Locust

Load testing the application with 100 user and Spawn rate of 20
GraphQL application test statistics

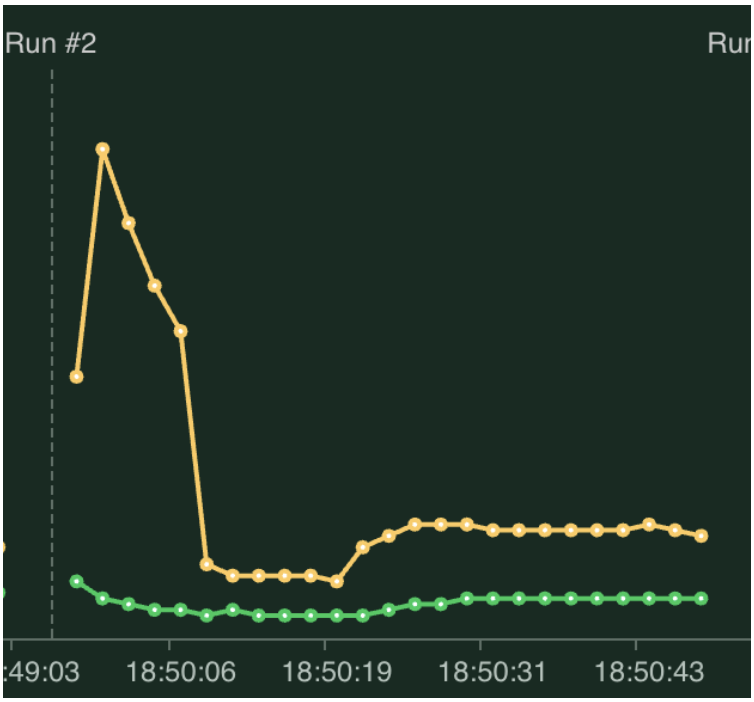
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	843	0	5	10	35	7	1	47	1735	33.9	0
GET	/store	789	0	10	19	46	12	2	49	2647	33.1	0
	Aggregated	**1632**	**0**	**7**	**16**	**44**	**9**	**1**	**49**	**2176**	**67**	**0**

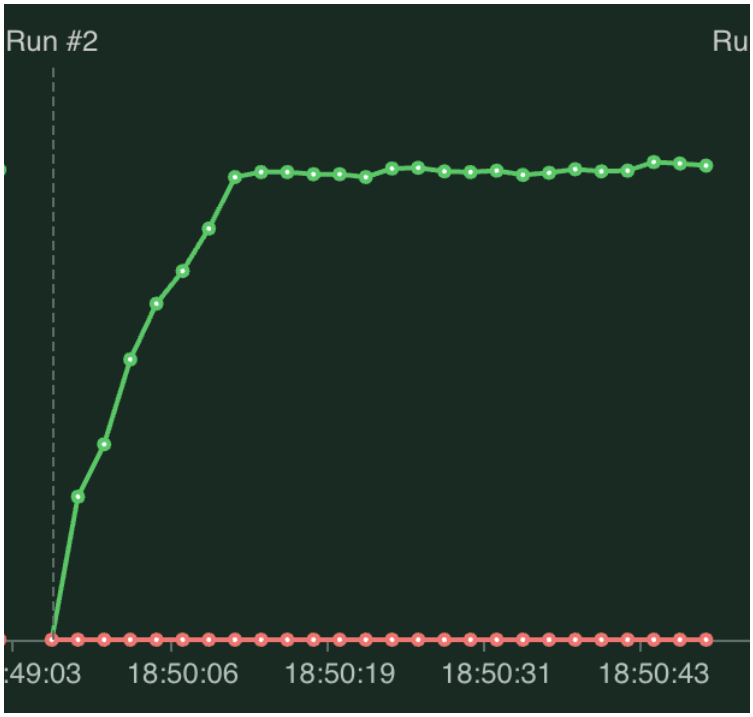




REST application test statistics

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/	867	0	5	15	83	8	1	96	1735	34.4	0
GET	/store	820	0	13	25	94	16	4	101	2149	32.1	0
	Aggregated	**1687**	**0**	**8**	**22**	**84**	**12**	**1**	**101**	**1936**	**66.5**	**0**





Conclusion

A significant difference can be observed between the response time of both the technologies when processing queries of type GET through GraphQL, particularly the ones that retrieve data from two services.