GRAPHQL SUMMIT

EVERYTHING GRAPHQL UNDER THE SUN

OCTOBER 3-5, 2022

SHERATON SAN DIEGO HOTEL & MARINA

GRAPHQL SUMMIT

Building Federated GraphQL APIs in Python

Adarsh Divakaran
Product Engineer
Strollby - UST

GraphQL Server Libraries in Python

Strollby

Code first

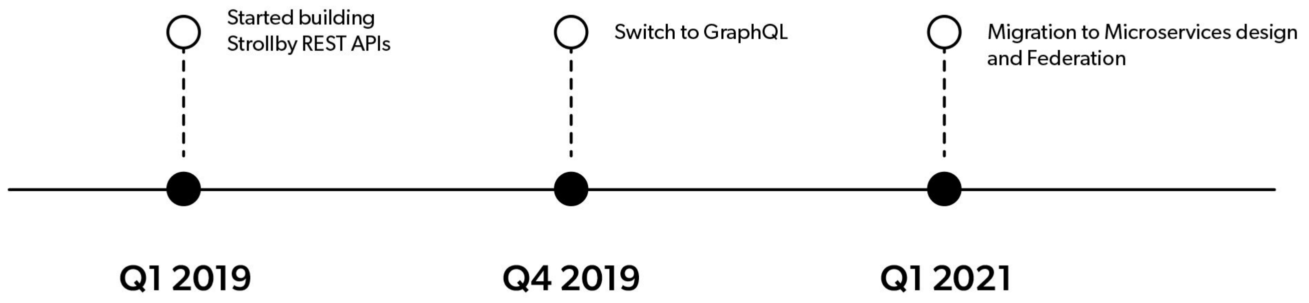
Graphene	★ 7.4k
Strawberry	★ 2.5k

Schema first

Ariadne	★ 1.8k
Tartiflette	★ 827

Strollby

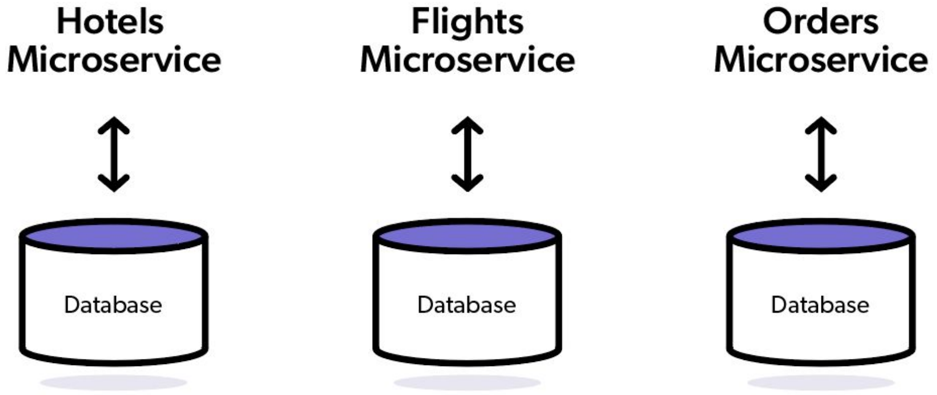
GraphQL Adoption at Strollby


Started building Strollby REST APIs
Switch to GraphQL
Migration to Microservices design and Federation
Q1 2019
Q4 2019
Q1 2021

- Using Graphene to build the Backend APIs for our web and mobile applications
- Started out with a monolithic server
- As the number of functionalities started to grow, schema became complex

Switch to Microservices

Strollby



Switch to Microservices

Strollby

Advantages

- No single point of failure
- Independently scalable
- Microservice schema ownership

Challenges

- Sharing types/data between microservices
- Non-breaking changes to the frontend

Apollo Federation

Strollby

- In a federated architecture, multiple GraphQL APIs are composed into a single federated graph. The individual APIs are called **subgraphs**, and they're composed into a **supergraph**
- Each backend microservice exposes their own subgraph and defines the relations to other services
- Federated schema abstracts the microservices design and exposes a single federated API for the frontend

Implementing Federation

Strollby

We will be using:

- Federation Supported Python GraphQL Servers:

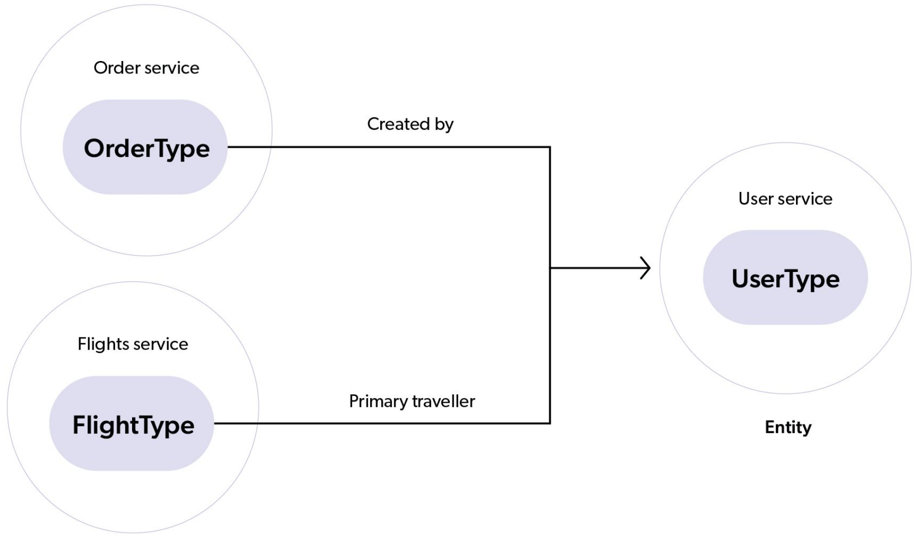
  Graphene with Graphene Federation for Subgraphs
- Apollo Gateway - NodeJS - For generating Federated schema from Subgraphs

Apollo Federation
Key Concepts

Entity

Strollby

In Apollo Federation, an entity is an object type that you define canonically in *one* subgraph and can then reference and extend in *other* subgraphs.



@key and @extends

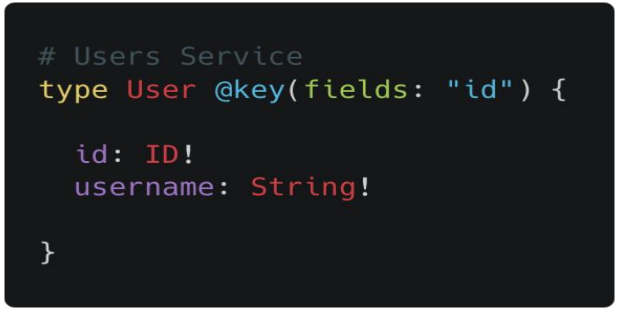
Strollby

A directive decorates part of a GraphQL schema or operation with additional configuration. Denoted using ‘@’ - similar to decorators in Python

@key

Designates an object type as an entity.

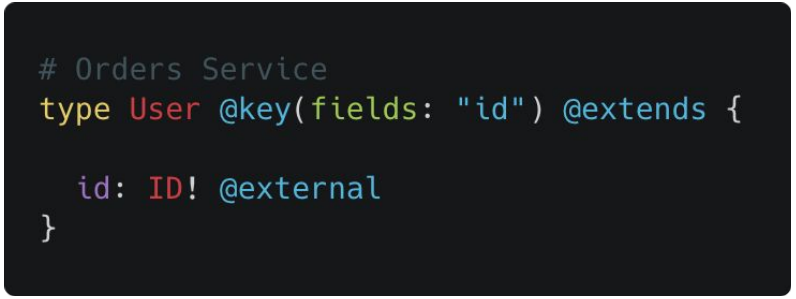
The @key directive is used to indicate fields that can be used to uniquely identify and fetch an object or interface.



```
# Users Service
type User @key(fields: "id") {

  id: ID!
  username: String!

}
```

@extends

The extends keyword indicates that the ‘decorated’ object type is an entity that's defined in another subgraph.



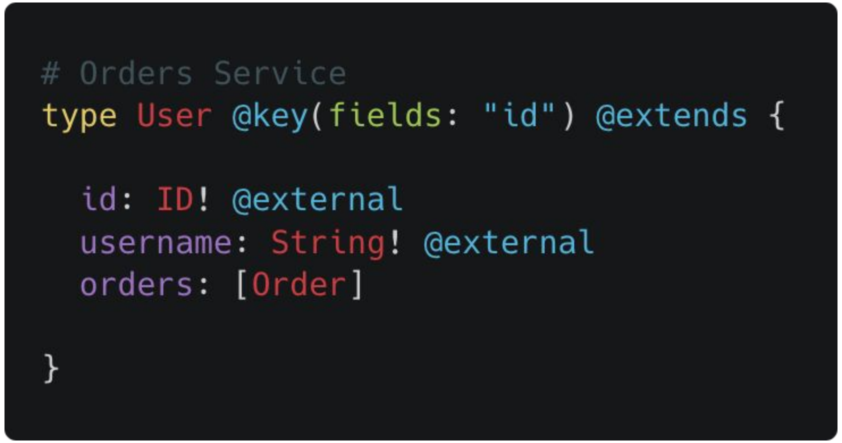
```
# Orders Service
type User @key(fields: "id") @extends {

  id: ID! @external
}
```

@external

Strollby

The @external directive is used to mark a **field** as owned by another service.

This allows service A to use fields from service B while also knowing at runtime the types of that field.



```
# Orders Service
type User @key(fields: "id") @extends {

  id: ID! @external
  username: String! @external
  orders: [Order]

}
```

Reference Resolver Function

The reference resolver function enables the gateway's query planner to resolve a particular entity by its @key fields.

```graphql
# Schema

type Flight @key(fields: "flightNumber") {
  flightNumber: ID!
  carrier: String
  totalSeats: Int
}
```

```python
@key('flight_number')
class Flight(graphene.ObjectType):
    ... # field definitions

    def __resolve_reference(reference, info):

        return FlightModel.\
            fetch_by_flight_number(reference.flight_number)
```

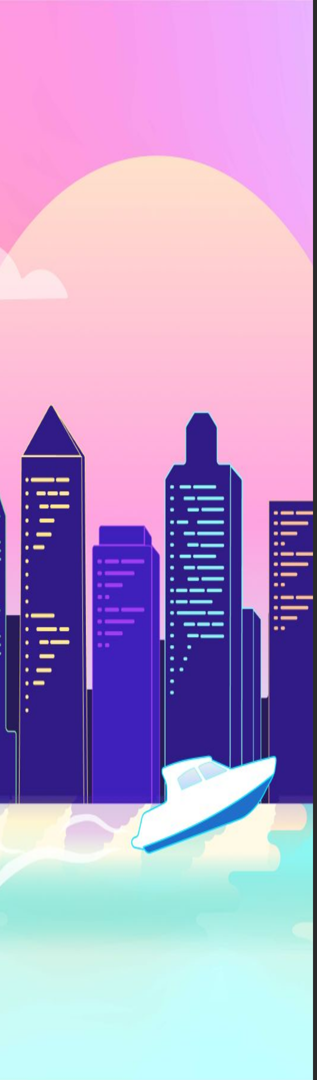
Demo - Federated Flight Booking

Strollby

Using Federation Specification V1

1. Define *flights* subgraph for managing Flight bookings
2. Define *users* subgraph, add an entity
3. Extend the entity in flights subgraph
4. Run the federated query

Demo



Any questions ?

Contact me at @adarshd905

Stop by my Topic Table
Thursday, October 5 at 12:30pm, Table 7
Eventides Lawn (lunch area)