Serverless GraphQL

Howdy!

I’m Jared Short

Director of Innovation @ trek10

@shortjared

GraphQL

A query language for your API, and a runtime for executing the queries

Why GraphQL

- One endpoint for your API
- Can provide an easy to understand API layer for your aggregated resources
- Only get the data you ask for, in the same shape
- Streamlined API iteration and extensibility
- Fantastic tooling / community

Technical Details of GraphQL

- Typed API and schema definitions
- Queries == Read, Mutations == Write
- Execution engine parses query to graph, each node runs through resolvers to fetch data
- Elegantly solves 1+n problem

Understanding the Schema

- Explicitly define all types in the graph
- Define lists of types
- Nullable vs non nullable (!)
- Scalars, Unions, Enums, Inputs, etc

TLDR

[GraphQL Schema Cheat Sheet]

Example Schema

```
type Speaker {
  firstName: String!
  lastName: String!
  talks: [Talk]
}

type Talk {
  speakers: [Speaker!]
  startTime: Int!
  endTime: Int
  title: String!
  room: String!
}
```

Understanding Root Resolvers

- Queries and mutations available at the “root”
- Accept arguments for scalars or input types, return types

Query / Mutation Root

```
type QueryRoot {
  speakers(limit: Int = 10): [Speaker]
  talks(sort: ASCENDING): [Talk]
}

input TalkInput {
  title: String!
  Speakers: [ID!]!
}
type MutationRoot {
  createTalk(talk: TalkInput): Talk
}
```

Understanding Root Resolvers

```
type QueryRoot {
  speakers(limit: Int = 10): [Speaker]
  talks(sort: ASCENDING): [Talk]
}
```

```
QueryRoot: {
    speakers(root, args, context){
      return new SpeakerService(context).loadSpeakers(args.limit);
    },
    talks(root, args, context){
      Return new TalkService(context).loadTalks(args.sort);
    }
}
```

Understanding Field Resolvers

```
type Speaker {
  firstName: String!
  lastName: String!
  talks: [Talk]
}
```

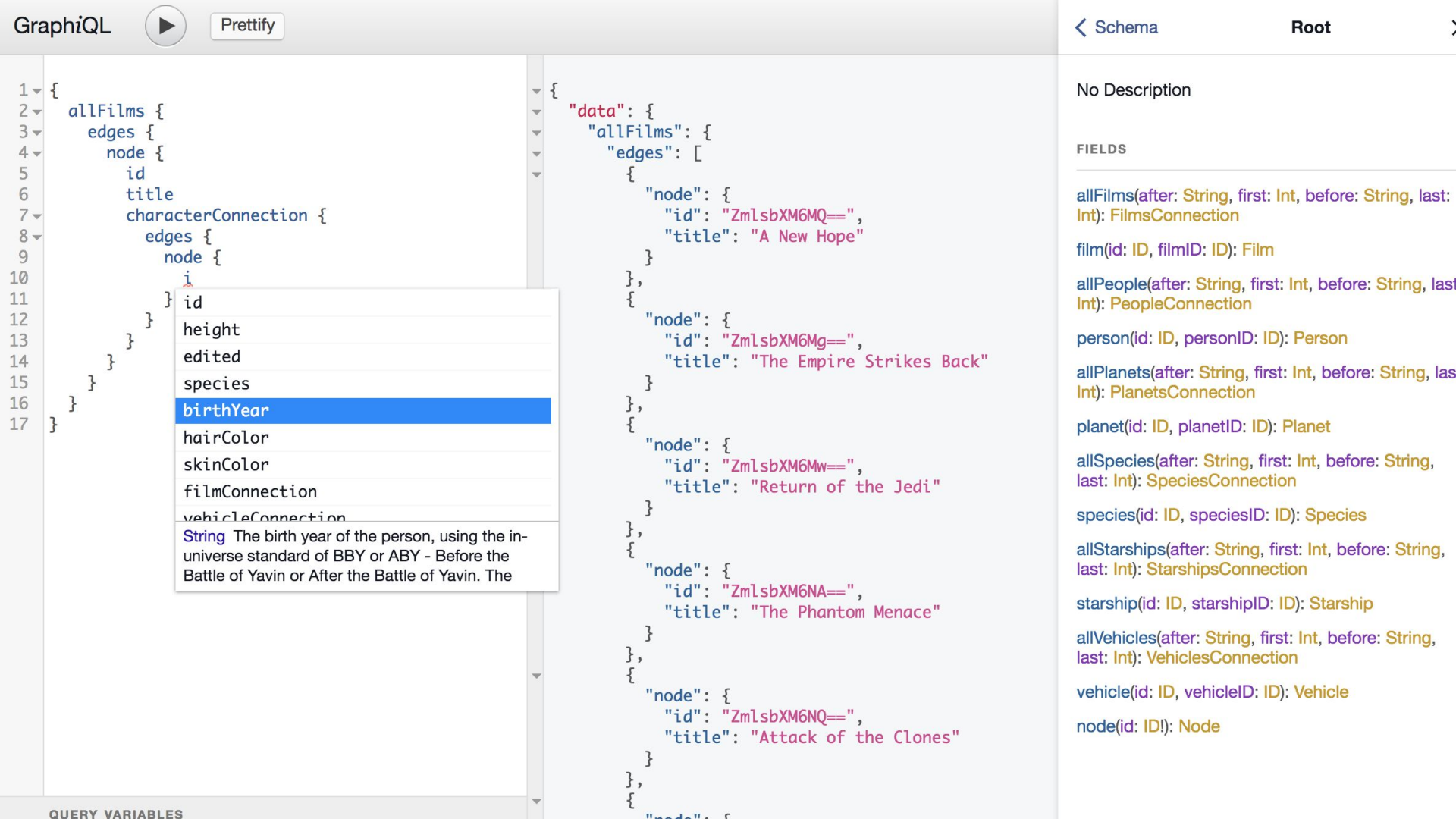
```
Speaker: {
     talks(root, args, context){
       Return new TalkService(context).loadByIds(root.talkIds);
     }
}
```

A Thin Wrapper on Business Logic

- GraphQL is **NOT** your business logic layer
- As thin as possible on top of your business logic
- You can map to multiple data sources
  - Rest API, Database, Filestore, etc

Developer Client Experience

- Ask for and receive exactly what you want, how you want it
- Limited API versioning concerns
- Easy to write efficient network requests
- API introspection (GraphiQL)


GraphiQL
Prettify
{
allFilms {
edges {
node {
id
title
characterConnection {
edges {
node {
i
}
}
}
}
}
}
}
id
height
edited
species
birthYear
hairColor
skinColor
filmConnection
String The birth year of the person, using the in-universe standard of BBY or ABY - Before the Battle of Yavin or After the Battle of Yavin. The
QUERY VARIABLES
{
"data": {
"allFilms": {
"edges": [
{
"node": {
"id": "ZmlsbXM6MQ==",
"title": "A New Hope"
}
},
{
"node": {
"id": "ZmlsbXM6Mg==",
"title": "The Empire Strikes Back"
}
},
{
"node": {
"id": "ZmlsbXM6Mw==",
"title": "Return of the Jedi"
}
},
{
"node": {
"id": "ZmlsbXM6NA==",
"title": "The Phantom Menace"
}
},
{
"node": {
"id": "ZmlsbXM6NQ==",
"title": "Attack of the Clones"
}
},
{
Schema
Root
No Description
FIELDS
allFilms(after: String, first: Int, before: String, last: Int): FilmsConnection
film(id: ID, filmID: ID): Film
allPeople(after: String, first: Int, before: String, last Int): PeopleConnection
person(id: ID, personID: ID): Person
allPlanets(after: String, first: Int, before: String, las Int): PlanetsConnection
planet(id: ID, planetID: ID): Planet
allSpecies(after: String, first: Int, before: String, last: Int): SpeciesConnection
species(id: ID, speciesID: ID): Species
allStarships(after: String, first: Int, before: String, last: Int): StarshipsConnection
starship(id: ID, starshipID: ID): Starship
allVehicles(after: String, first: Int, before: String, last: Int): VehiclesConnection
vehicle(id: ID, vehicleID: ID): Vehicle
node(id: ID!): Node

Serverless

Serverless?

- Still run on servers, *you just don't care*
- Moves your responsibility higher up the stack
- Billing at ms increments (***micropayments!***)
- Pairs very well with event driven models

Why Serverless

- Reduced complexity of scaling
- High availability
- Economics / Total Cost of Ownership
- Rapid iteration, learn fast

Developer Hints

- Your Functions as a Service (FaaS) provider should be irrelevant
- Rethink traditional requirements and approaches
- Can it be event driven?

Serverless GraphQL

Why Serverless GraphQL

- All the normal serverless wins (scalability, availability, etc)
- All the power of GraphQL for clients and systems
- Smooth out some rough edges of GraphQL
  - Resource Exhaustion Attacks
  - Complex Resolvers

Resource Exhaustion Attacks

- Think DoS attack that's easy to do by mistake
- Deeply nested request / pagination abuse
- Useful Optimization & Prevention
  - Enforce pagination & limits
  - Maximum depth checking
  - Request Batching & Caching

Resource Exhaustion Attacks

REQUEST

```
{
  allFilms {
    edges {
      node {
        id
        title
        characterConnection {
          ...CharacterFields
          edges {
            node {
              filmConnection {
                ...FilmFields
                edges {
                  node {
                    characterConnection {
                      ...CharacterFields
                      #
                      # COMMENTED BECAUSE 50/50 THIS BREAKS CHROME
                      #
                      # edges {
                      #   node {
                      #     filmConnection {
                      #       ...FilmFields
                      #     }
                      #   }
                      # }
                    }
```

RESPONSE


```
{
  "data": {
    "allFilms": {
      "edges": [
        {
          "node": {
            "id": "ZmlsbXM6MQ==",
            "title": "A New Hope",
            "characterConnection": {
              "edges": [
                {
                  "node": {
                    "id": "cGVvcGxlOjE=",
                    "name": "Luke Skywalker",
                    "filmConnection": {
                      "edges": [
                        {
                          "node": {
                            "id": "ZmlsbXM6MQ==",
                            "title": "A New Hope",
                            "characterConnection": {
                              "edges": [
                                {
                                  "node": {
                                    "id": "cGVvcGxlOjE=",
                                    "name": "Luke Skywalker"
                                  }
```

REA Mitigation (The Easy Way)

- Each graphql execution is allocated its own resources
- Set a reasonable resource allocation & timeout
- Web Application Firewalls to stop bad actors

Request Batching & Caching

- Fresh cache for each invocation
- Hard part abstracted away from devs
- DataLoader https://github.com/facebook/dataloader

Example Query

```
query {
  speakers {
    firstName
    talks {
      title
      speakers {
        firstName
      }
    }
  }
}
```

Query Metrics

179 Speakers & 130 Talks

No Cache or Batching

Requests: 359

Cache Only / Batch Only

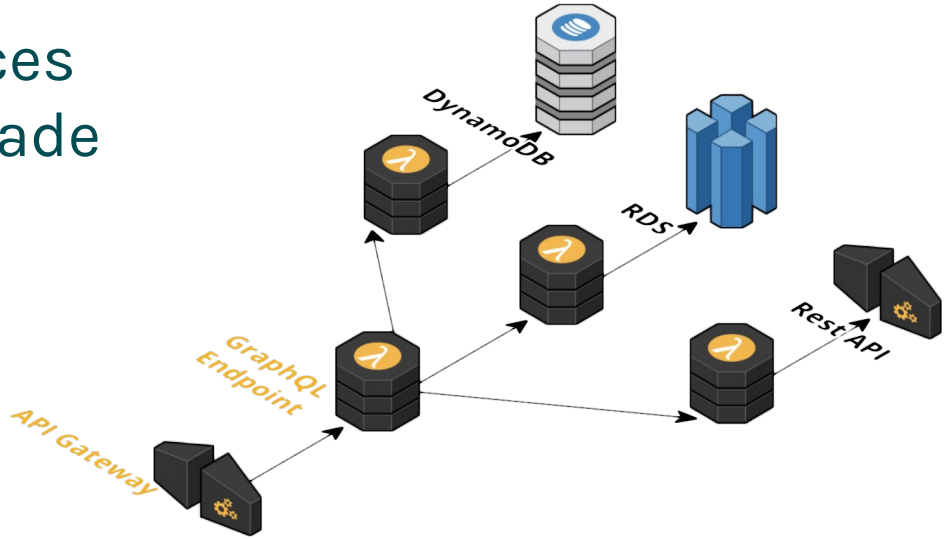
Requests: 132 / 3

Cache & Batching

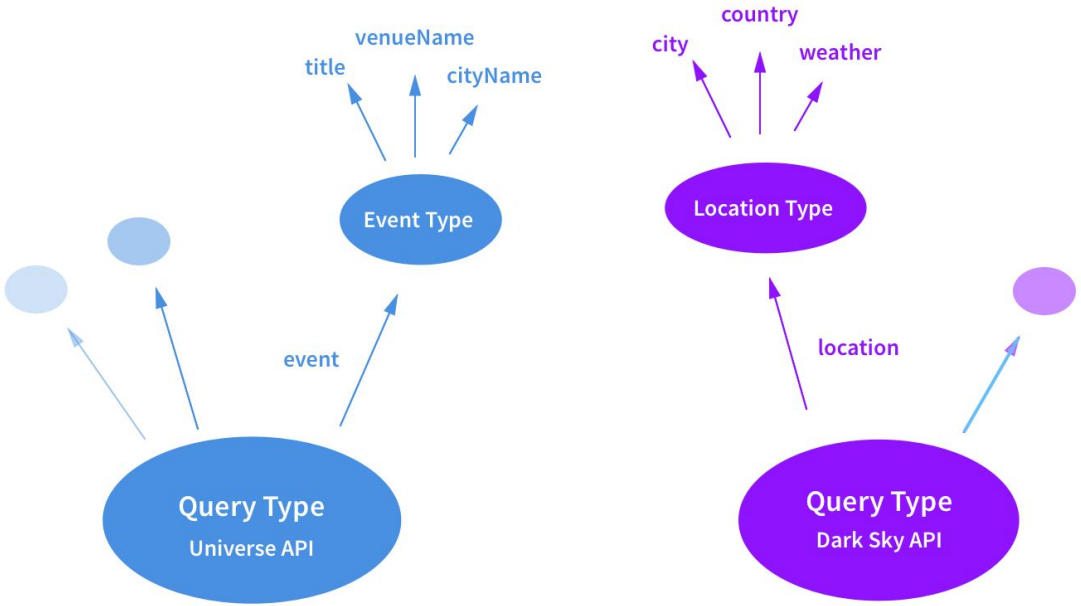
Requests: 2

Complex Resolvers & Multi-Data Sources

- Multiple data sources and aggregation made easy
- More expensive queries, split to different lambda



Schema Stitching



Souce: https://dev-blog.apollodata.com/graphql-schema-stitching-8af23354ac37

Schema Stitching

REQUEST

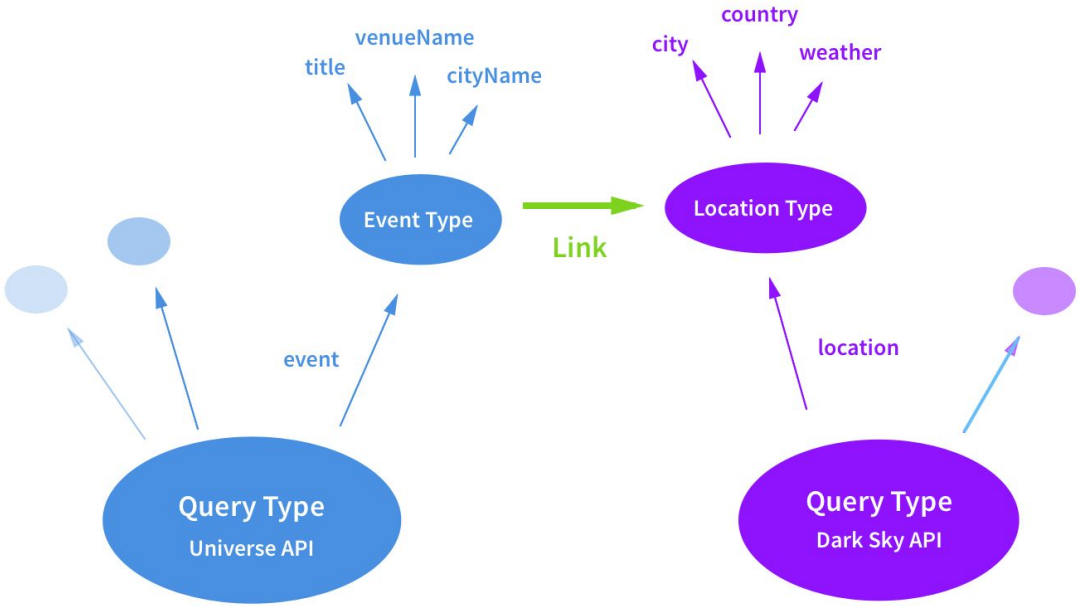
```
query {
  event(id: "5983706debf3140039d1e8b4") {
    title
    venueName
    cityName
  }

  location(place: "San Francisco") {
    city
    country
    weather {
      summary
      temperature
    }
  }
}
```

RESPONSE

```
{
  "data": {
    "event": {
      "title": "GraphQL Summit 2017",
      "venueName": "Bespoke",
      "cityName": "San Francisco"
    },
    "location": {
      "city": "San Francisco",
      "country": "United States",
      "weather": {
        "summary": "Mostly Cloudy",
        "temperature": 64.06
      }
    }
  }
}
```

Souce: https://dev-blog.apollodata.com/graphql-schema-stitching-8af23354ac37

Schema Stitching



Souce: https://dev-blog.apollodata.com/graphql-schema-stitching-8af23354ac37

Schema Stitching with Links

REQUEST

```
query {
  event(id: "5983706debf3140039d1e8b4") {
    title
    venueName
    cityName
    location {
      city
      country
      weather {
        summary
        temperature
      }
    }
  }
}
```

RESPONSE


```
{
  "data": {
    "event": {
      "title": "GraphQL Summit 2017",
      "venueName": "Bespoke",
      "cityName": "San Francisco",
      "location": {
        "city": "San Francisco",
        "country": "United States",
        "weather": {
          "summary": "Mostly Cloudy",
          "temperature": 63.68
        }
      }
    }
  }
}
```

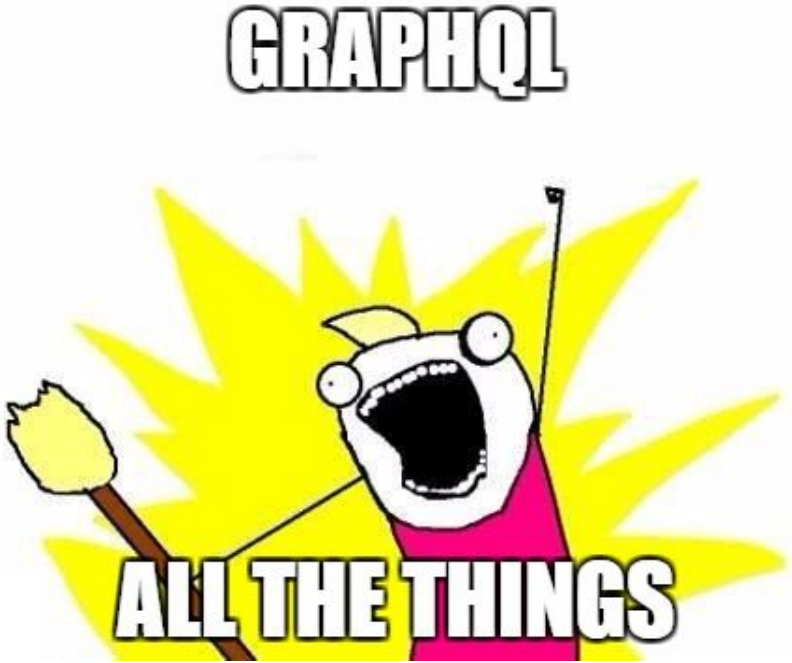

Souce: https://dev-blog.apollodata.com/graphql-schema-stitching-8af23354ac37

Schema Stitching So What

- Still a newer concept and area in GraphQL
- Keep thinking in microservices
- Easily extend and add functionality to your services
- Composable services and products
  - Encourage reuse of services
  - Compose services into whole new offerings

A word of warning....

Don’t be that person...



When not to use (force) it

- Non-problem rarely used legacy system
- Big data exports (async returns is fine)
- Internal buy in is key, if inertia is strongly against you, don’t force it, start small
- You can sneakily integrate downstream without forcing them to migrate

Protect Downstream Systems

- You are only as strong as your weakest resources
- Serverless scales, your dependencies don’t
- Queues, caching, play nice

Back to Basics

- Authorization
- Pagination (relay spec)
- Good documentation

Getting Started

GraphQL

Dan Schafer - GraphQL at Facebook Talk (Pagination & Auth discussed) (via YouTube)

Apollo Stack - http://dev.apollodata.com/

GraphQL Schema Cheat Sheet

Serverless GraphQL

Boilerplate Starter: https://github.com/serverless/serverless-graphql

Graphcool

Hybrid Faas Framework & GraphQL engine: https://www.graph.cool/

- Open Source with SaaS Offerings

Thanks!

Questions?

@shortjared