Introduction to GraphQL

Calvin Webster, Technical Architect @ Somo Global

February 27, 2020

What is GraphQL?

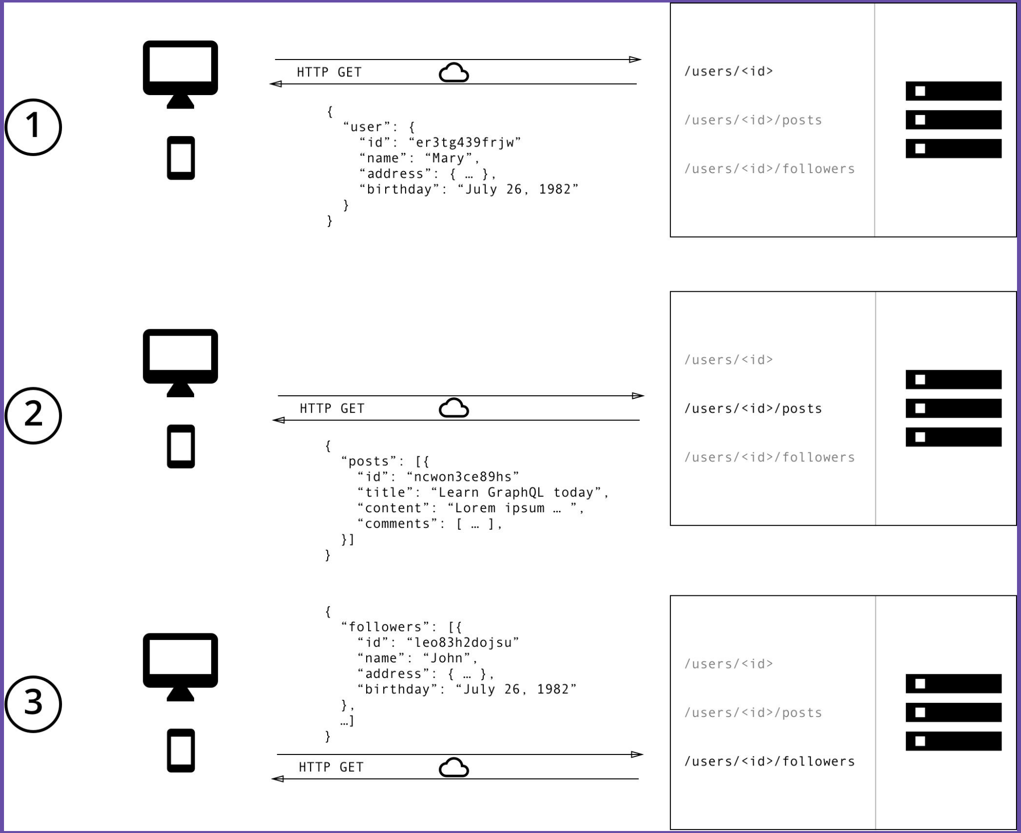
Intro to GraphQL

GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

Sounds good, right? So let’s compare to REST

REST API

- Resources defined by ‘endpoints’
- Client(s) call to many endpoints
- Uses HTTP as transport
- Well established and widely used



```
{
  “user”: {
    “id”: “er3tg439frjw”
    “name”: “Mary”,
    “address”: { … },
    “birthday”: “July 26, 1982”
  }
}
```

```
{
  “posts”: [{
    “id”: “ncwon3ce89hs”
    “title”: “Learn GraphQL today”,
    “content”: “Lorem ipsum … ”,
    “comments”: [ … ],
  }]
}
```

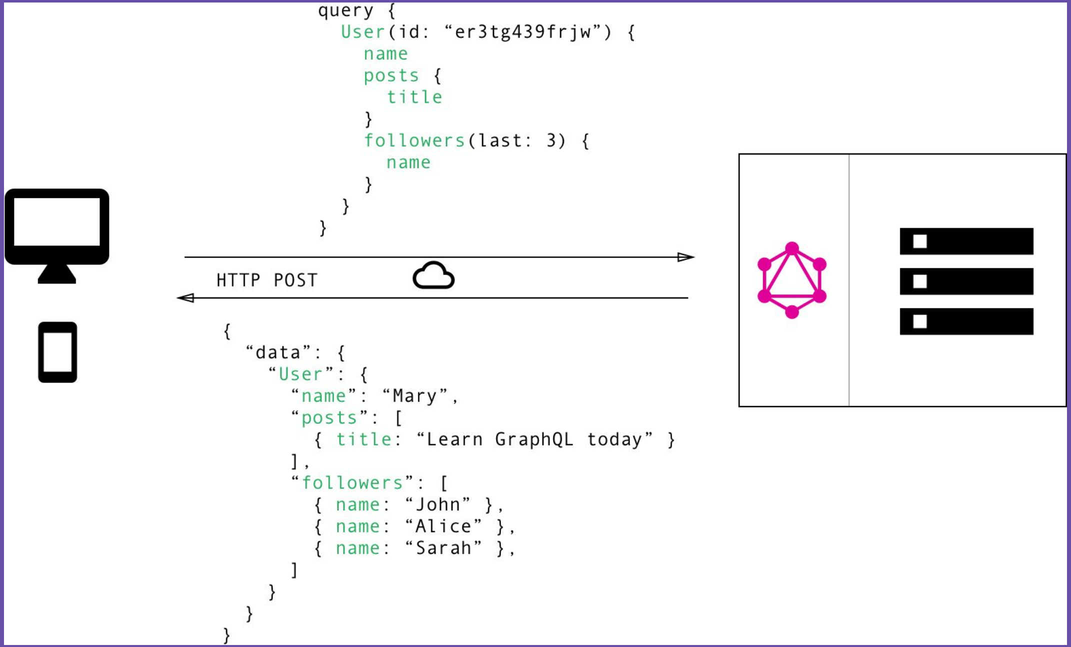
```
{
  “followers”: [{
    “id”: “leo83h2dojsu”
    “name”: “John”,
    “address”: { … },
    “birthday”: “July 26, 1982”
  },
  …]
}
```

Source: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/

GraphQL is a query language for APIs...

GraphQL

- Resources defined by a GraphQL Schema
- Client sends query, server orchestrates the data
- Multiple transports (HTTP, AMQP, WebSockets)
- Self Documenting (w/ introspection & tooling)



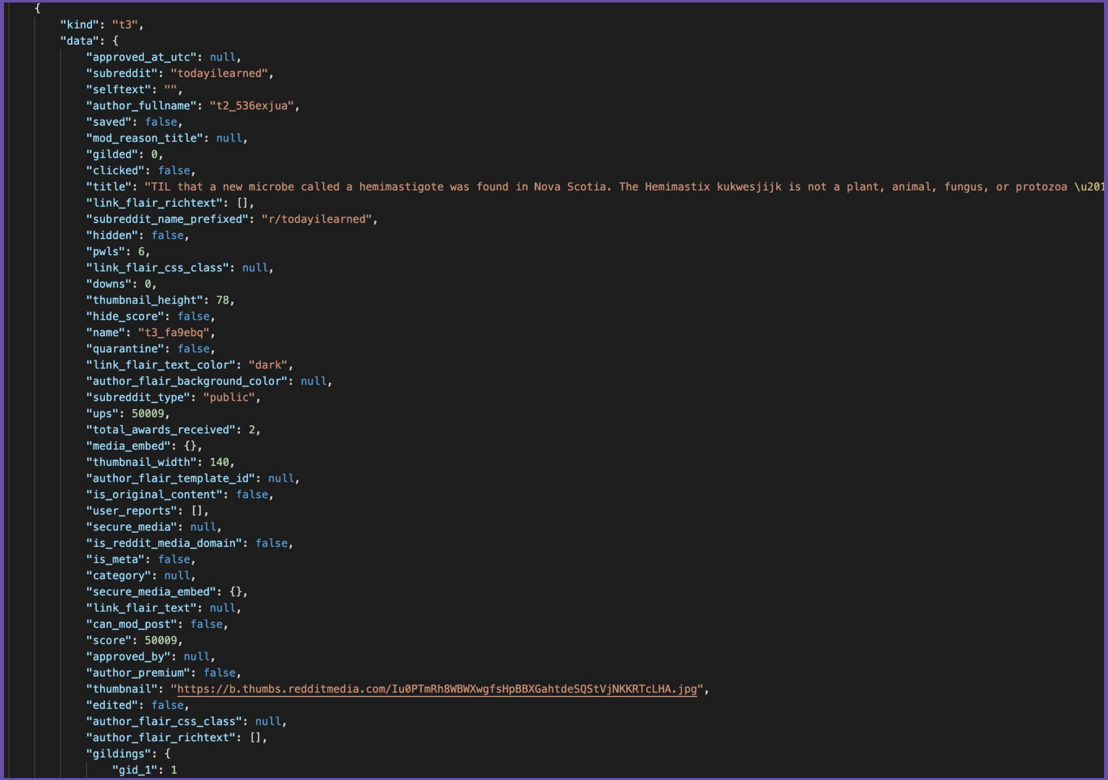
```
query {
  User(id: “er3tg439frjw”) {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

```
{
  “data”: {
    “User”: {
      “name”: “Mary”,
      “posts”: [
        { title: “Learn GraphQL today” }
      ],
      “followers”: [
        { name: “John” },
        { name: “Alice” },
        { name: “Sarah” },
      ]
    }
  }
}
```

But what about REST?

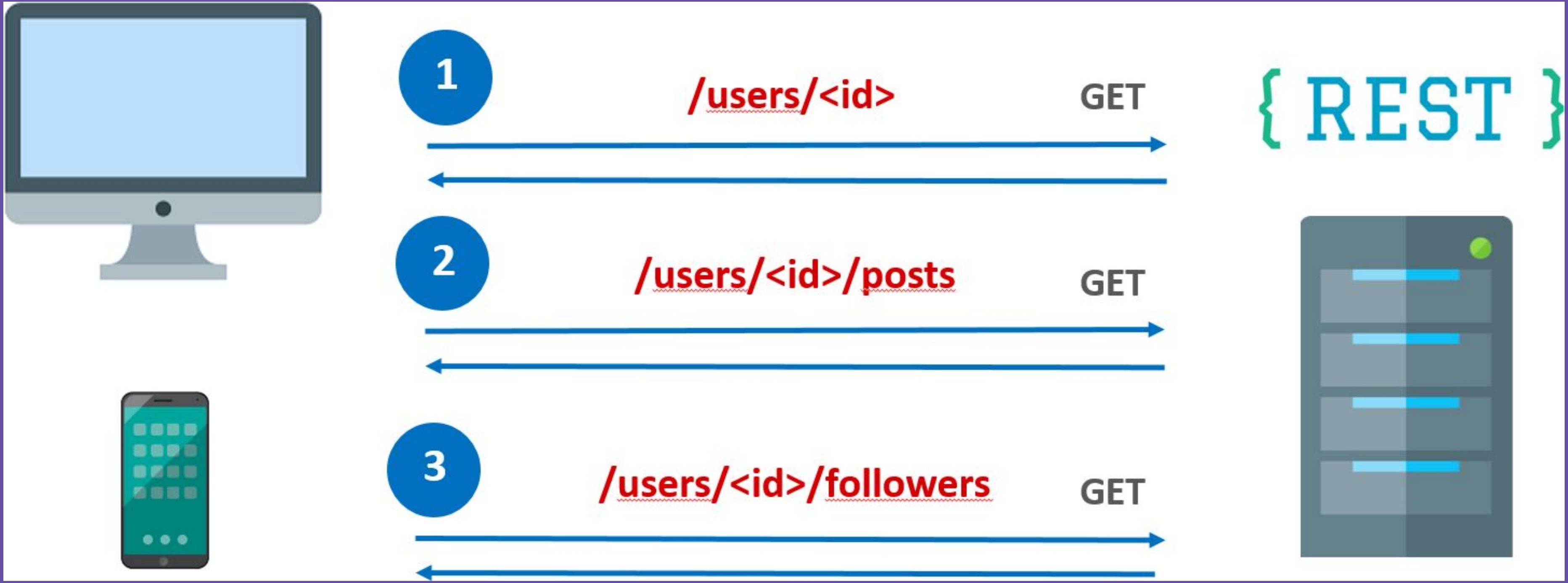
Overfetching

Overfetching



```
{
    "kind": "t3",
    "data": {
        "approved_at_utc": null,
        "subreddit": "todayilearned",
        "selftext": "",
        "author_fullname": "t2_536exjua",
        "saved": false,
        "mod_reason_title": null,
        "gilded": 0,
        "clicked": false,
        "title": "TIL that a new microbe called a hemimastigote was found in Nova Scotia. The Hemimastix kukwesjijk is not a plant, animal, fungus, or protozoa \u201
        "link_flair_richtext": [],
        "subreddit_name_prefixed": "r/todayilearned",
        "hidden": false,
        "pwls": 6,
        "link_flair_css_class": null,
        "downs": 0,
        "thumbnail_height": 78,
        "hide_score": false,
        "name": "t3_fa9ebq",
        "quarantine": false,
        "link_flair_text_color": "dark",
        "author_flair_background_color": null,
        "subreddit_type": "public",
        "ups": 50009,
        "total_awards_received": 2,
        "media_embed": {},
        "thumbnail_width": 140,
        "author_flair_template_id": null,
        "is_original_content": false,
        "user_reports": [],
        "secure_media": null,
        "is_reddit_media_domain": false,
        "is_meta": false,
        "category": null,
        "secure_media_embed": {},
        "link_flair_text": null,
        "can_mod_post": false,
        "score": 50009,
        "approved_by": null,
        "author_premium": false,
        "thumbnail": "https://b.thumbs.redditmedia.com/Iu0PTmRh8WBWXwgfsHpBBXGahtdeSQStVjNKKRTcLHA.jpg",
        "edited": false,
        "author_flair_css_class": null,
        "author_flair_richtext": [],
        "gildings": {
            "gid_1": 1
```

Underfetching



Credit :https://www.codemotion.com/magazine/dev-hub/web-developer/graphql-testing-with-karate/

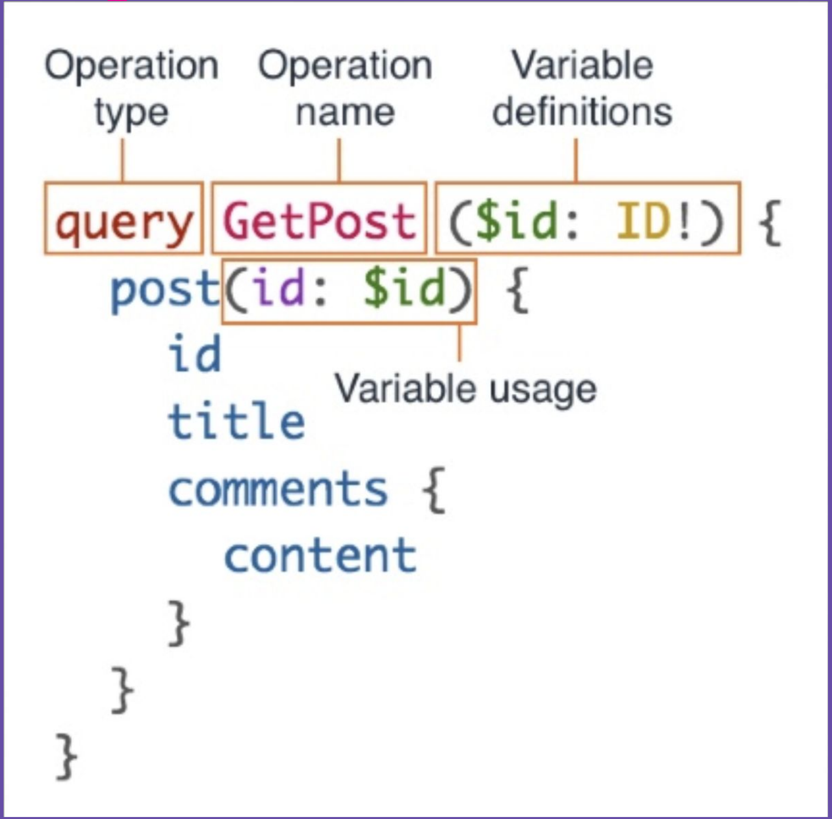
Evolving API's

- Versioning `/api/v2...v5/resource`
- Maintaining
- Deprecation

GraphQL

How GraphQL?

Basics of a Query



```
query GetPost($id: ID!) {
  post(id: $id) {
    id
    title
    comments {
      content
    }
  }
}
```

There are 3 types of operations:

- Query
- Mutation
- Subscription

Credit :https://www.slideshare.net/AmazonWebServices/introduction-to-graphql-95577414

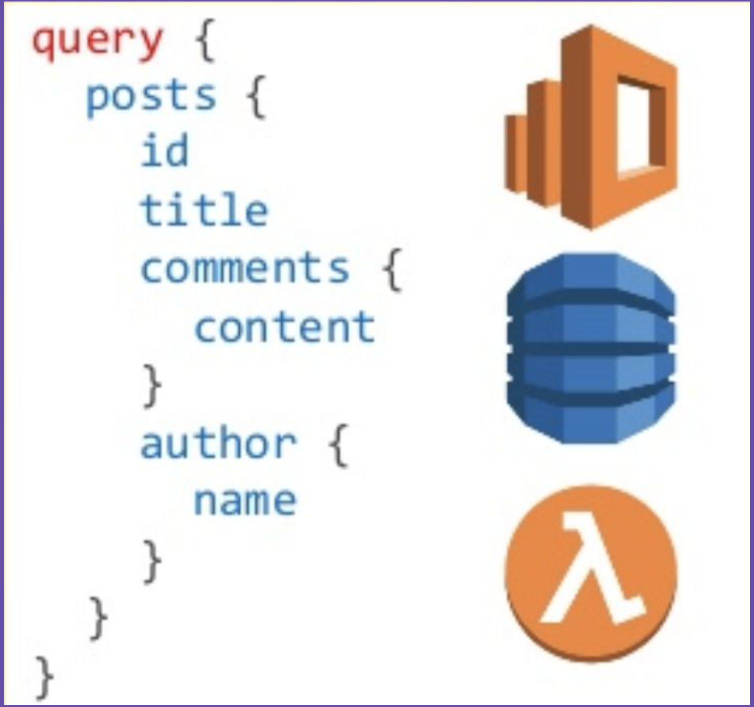
A runtime for fulfilling those queries...

- A GraphQL server responds to a request in the same shape of the the query
- Fields from the query are ‘resolved’ by a function in the GraphQL Server via a ‘resolver’
- GraphQL provides a clear contract via a schema, allowing the clients to control the info it is querying for.

Credit :https://www.slideshare.net/AmazonWebServices/introduction-to-graphql-95577414

A runtime for fulfilling those queries...

Example of field level resolvers



```
query {
  posts {
    id
    title
    comments {
      content
    }
    author {
      name
    }
  }
}
```

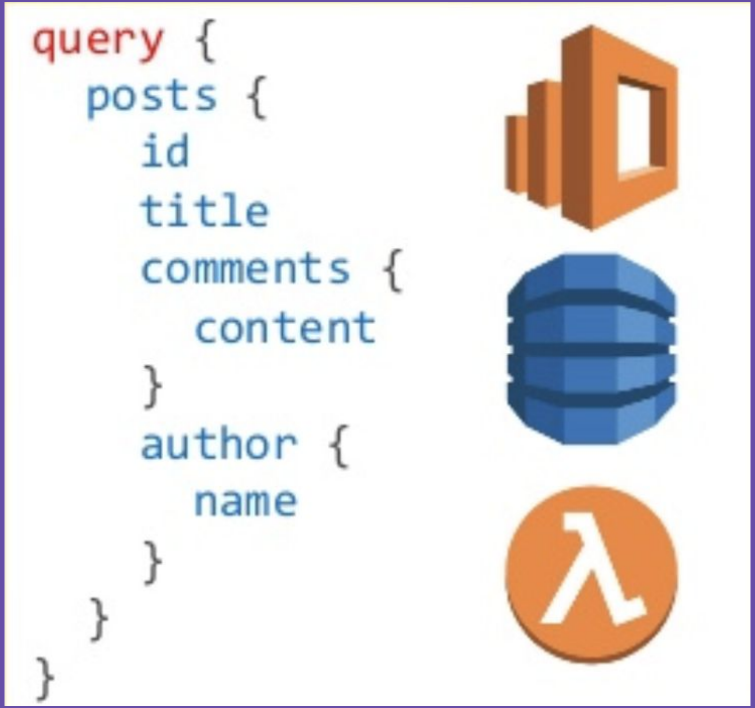
Posts resolved by Elastic Search

Comments resolved by DynamoDB

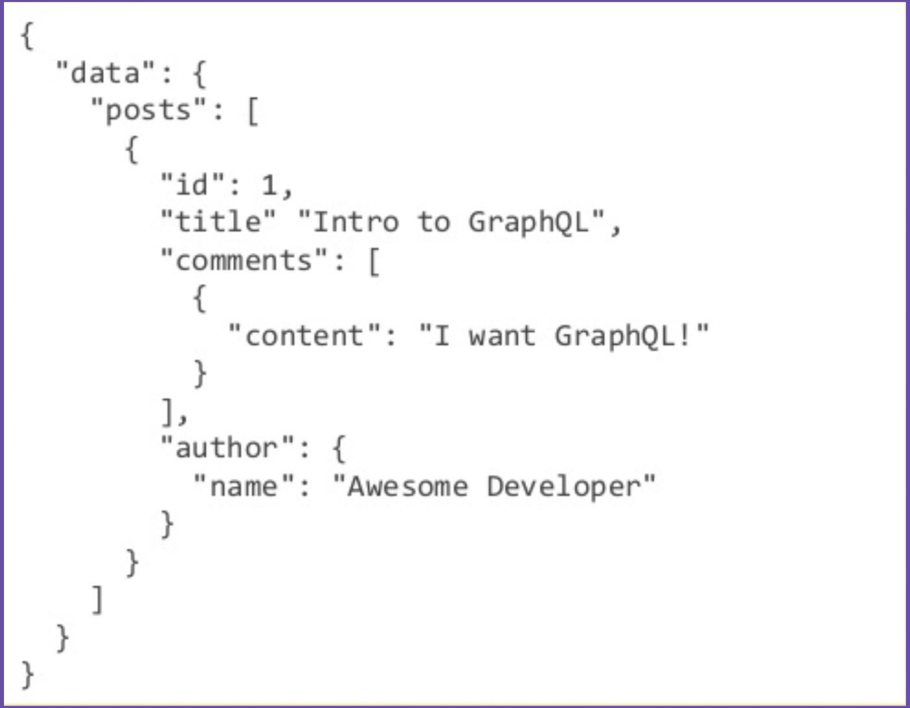
Author resolved by custom lamda

Credit :https://www.slideshare.net/AmazonWebServices/introduction-to-graphql-95577414

A runtime for fulfilling those queries...



```
query {
  posts {
    id
    title
    comments {
      content
    }
    author {
      name
    }
  }
}
```

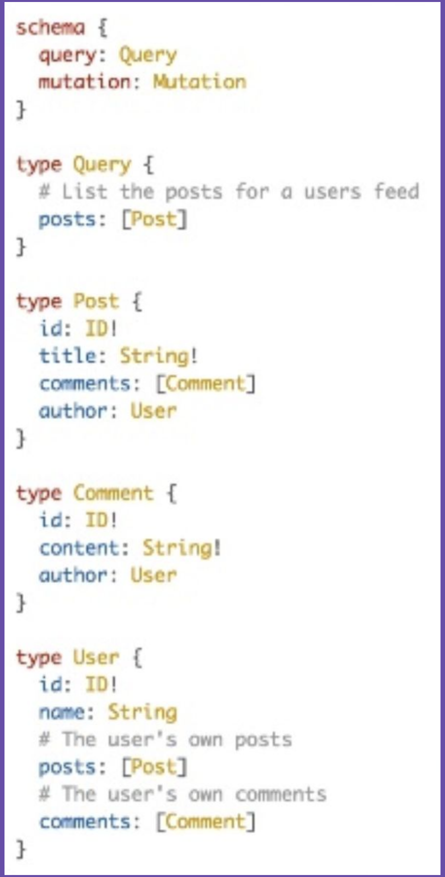


```
{
  "data": {
    "posts": [
      {
        "id": 1,
        "title" "Intro to GraphQL",
        "comments": [
          {
            "content": "I want GraphQL!"
          }
        ],
        "author": {
          "name": "Awesome Developer"
        }
      }
    ]
  }
}
```

Credit :https://www.slideshare.net/AmazonWebServices/introduction-to-graphql-95577414

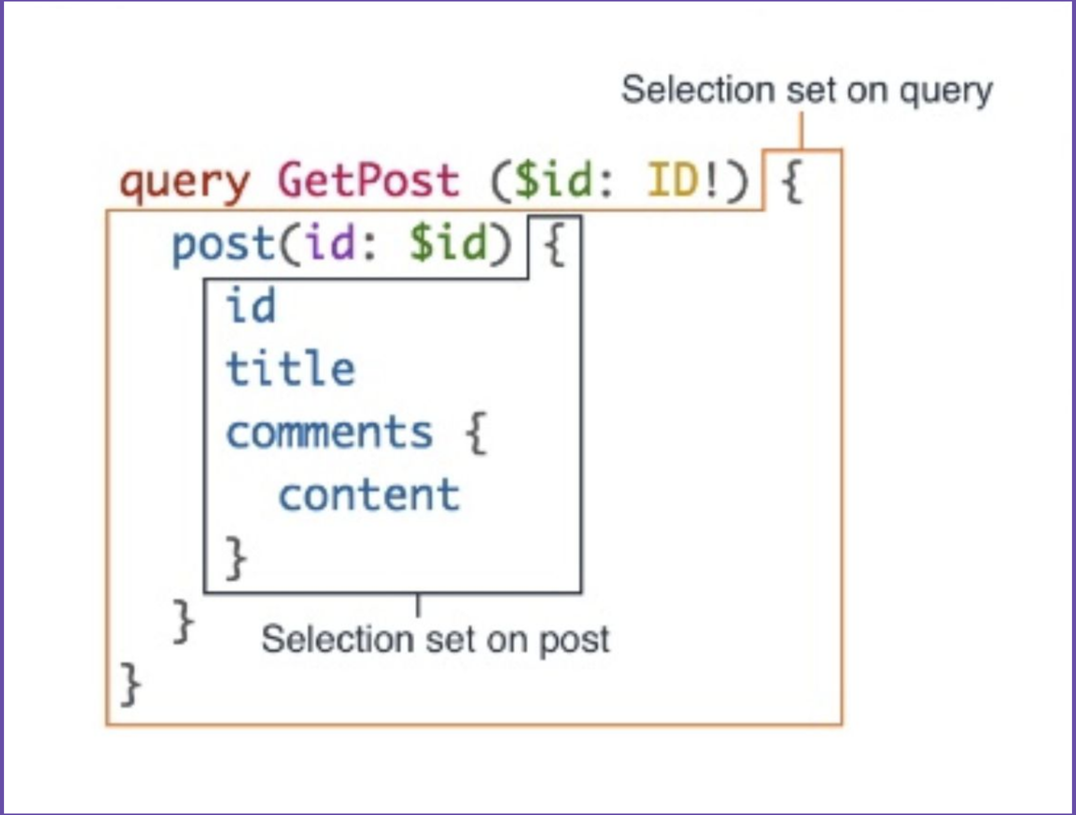
GraphQL provides a description of your data...

- Define Schema with Schema Definition Language (SDL)
- Types (Object, Interface, Union, Enum, Input, Scalar)
- Self Documenting
- Code Generation



```
schema {
  query: Query
  mutation: Mutation
}

type Query {
  # List the posts for a users feed
  posts: [Post]
}

type Post {
  id: ID!
  title: String!
  comments: [Comment]
  author: User
}

type Comment {
  id: ID!
  content: String!
  author: User
}

type User {
  id: ID!
  name: String
  # The user's own posts
  posts: [Post]
  # The user's own comments
  comments: [Comment]
}
```

Selection Sets



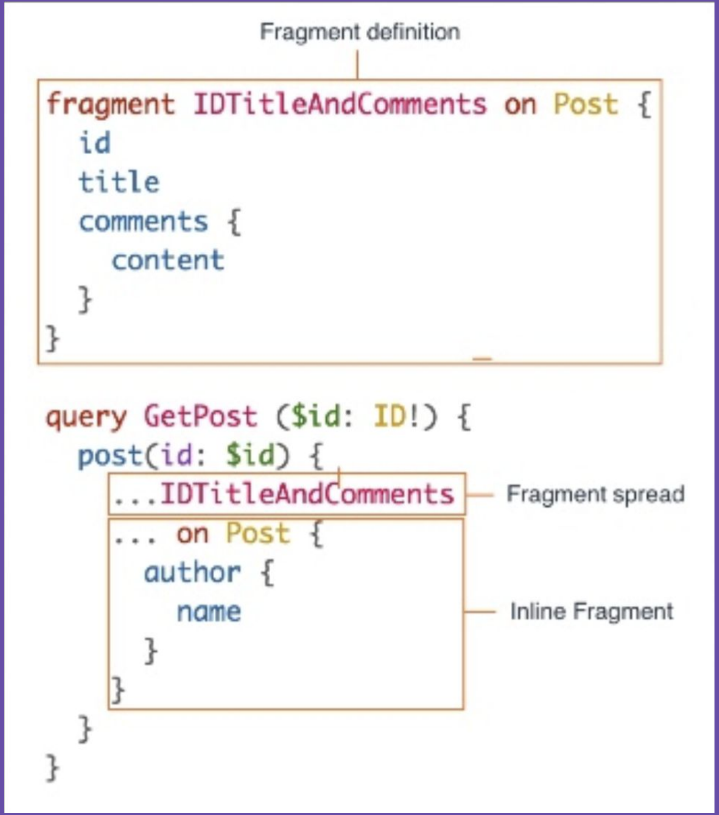
```
query GetPost ($id: ID!) {
  post(id: $id) {
    id
    title
    comments {
      content
    }
  }
}
```

Selection Sets specify what fields to resolve, and consequently, what data to return.

They can be nested and arbitrary.

Fields that return a scalar or enum do no have selection sets.

Fragments

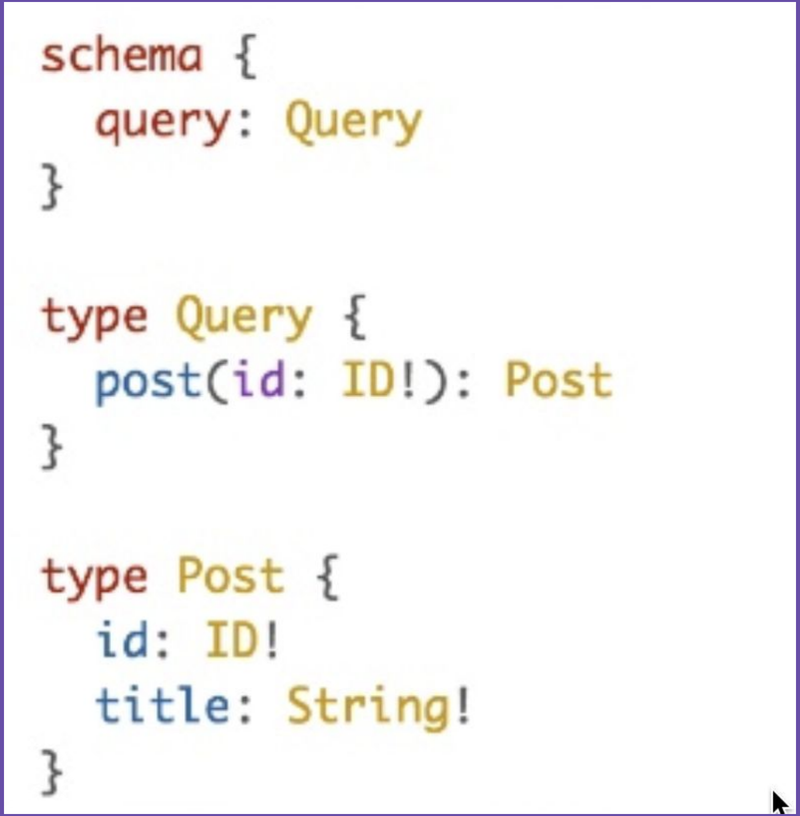


```
fragment IDTitleAndComments on Post {
  id
  title
  comments {
    content
  }
}

query GetPost ($id: ID!) {
  post(id: $id) {
    ...IDTitleAndComments
    ... on Post {
      author {
        name
      }
    }
  }
}
```

Fragments are a basic unit of composition.

Like a named, reusable selection set, but with a few more features.

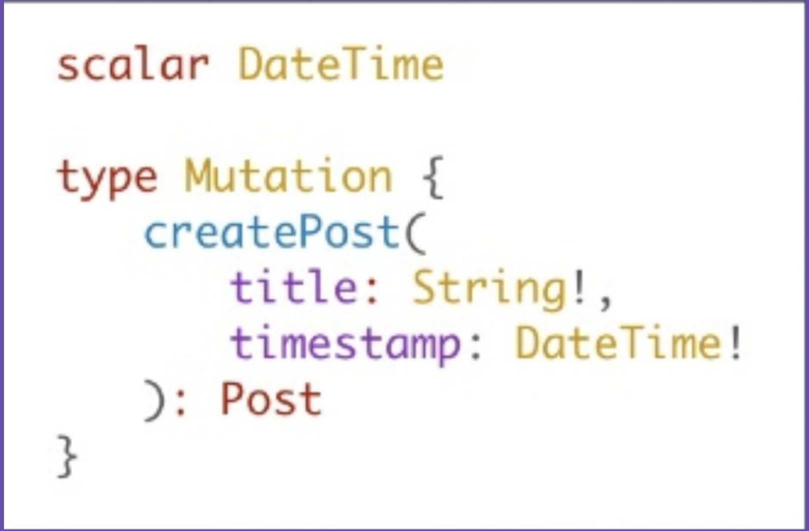
Schema Definition Language (SDL)



```
schema {
  query: Query
}

type Query {
  post(id: ID!): Post
}

type Post {
  id: ID!
  title: String!
}
```

To talk about our schemas in a language agnostic way, and provide a concise way to describe and define our schemas.

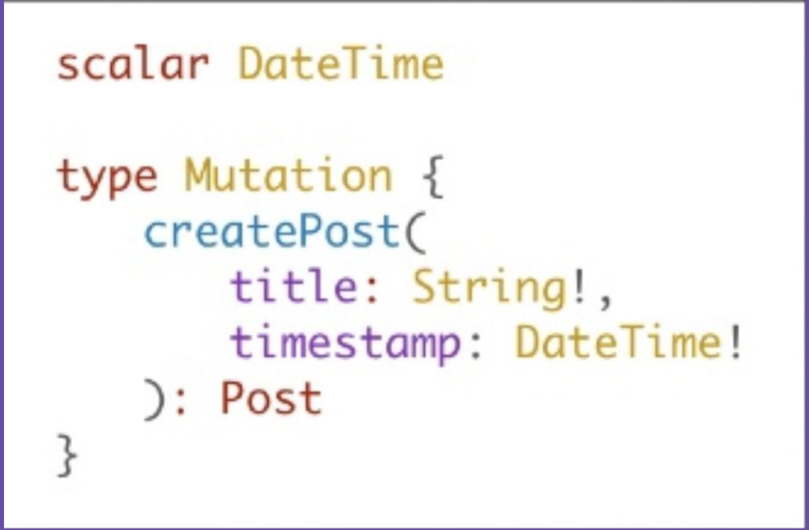
Scalars



```
scalar DateTime

type Mutation {
    createPost(
        title: String!,
        timestamp: DateTime!
    ): Post
}
```

String, Int, Float, Boolean, and ID

Objects

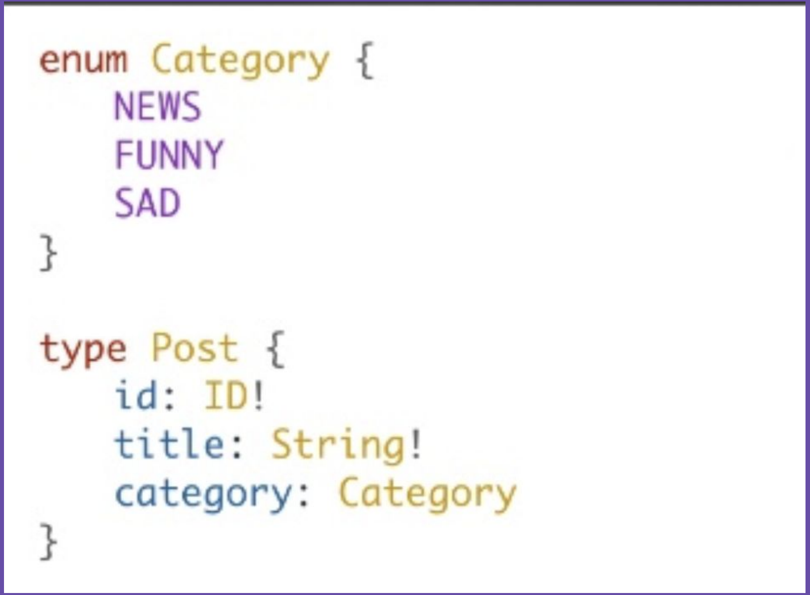


```
scalar DateTime

type Mutation {
    createPost(
        title: String!,
        timestamp: DateTime!
    ): Post
}
```

Contains a set of fields, each of which is a specific type.

Root types, like Query, Mutation, and Subscription are also Object types

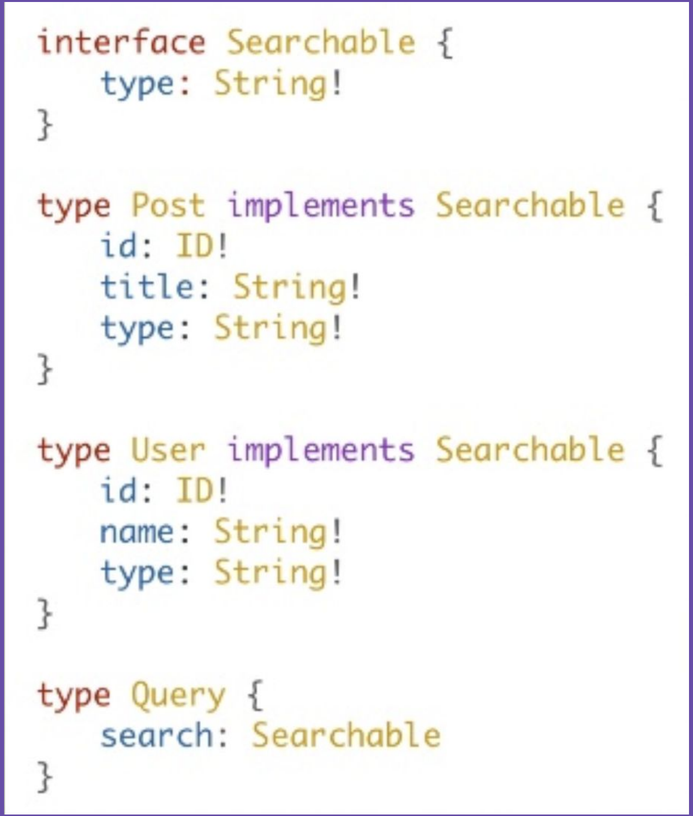
Enums



```
enum Category {
    NEWS
    FUNNY
    SAD
}

type Post {
    id: ID!
    title: String!
    category: Category
}
```

Denotes a Scalar that is a predetermined set of values.

Interfaces

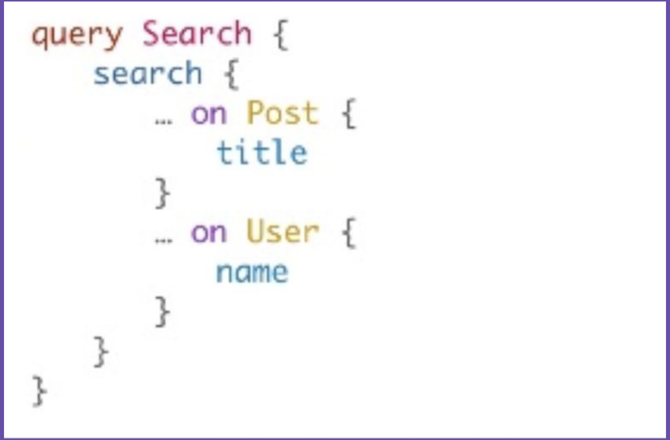


```
interface Searchable {
    type: String!
}

type Post implements Searchable {
    id: ID!
    title: String!
    type: String!
}

type User implements Searchable {
    id: ID!
    name: String!
    type: String!
}

type Query {
    search: Searchable
}
```

Defines a common set of fields.

Object types implement Interfaces

Query with Fragments

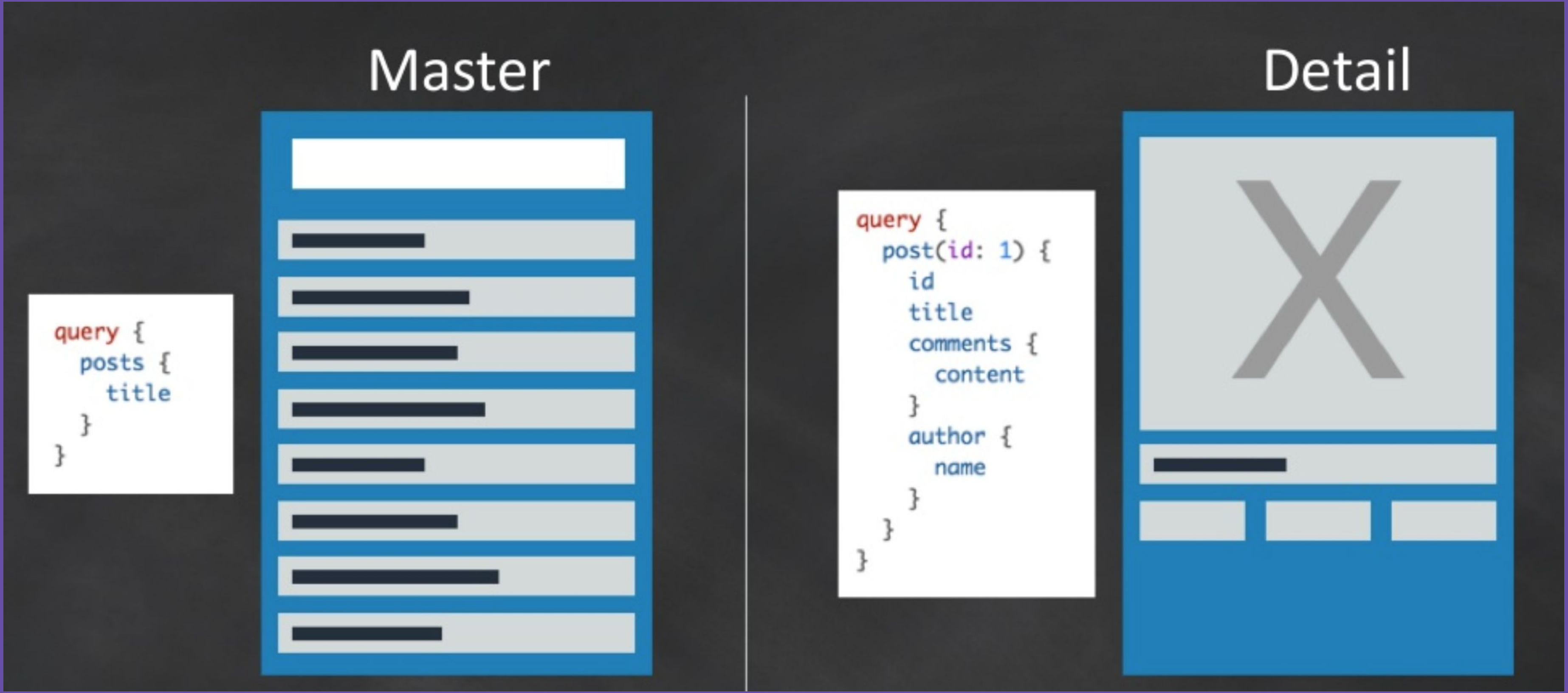


```
query Search {
    search {
        … on Post {
            title
        }
        … on User {
            name
        }
    }
}
```

Give clients the power to ask for what they need...

- Each UI component has its own data requirements
- The same backend model can be viewed in different ways
- The network is costly
- The security boundary should be defined by the backend, not ehe UI
- A queryable API is a better solution than many different endpoints per view.

Use Case: Master => Detail views



```
query {
  posts {
    title
  }
}
```

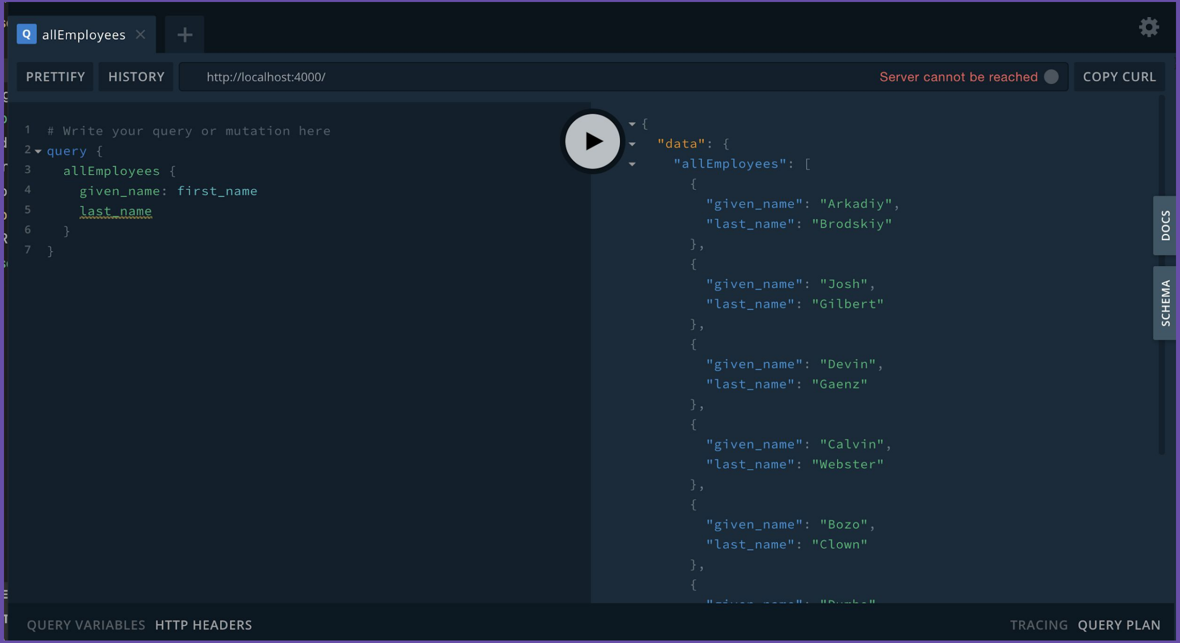
```
query {
  post(id: 1) {
    id
    title
    comments {
      content
    }
    author {
      name
    }
  }
}
```

Makes it easier to evolve APIs over time...

- GraphQL requires that every request specify every field
- We understand what is being used, so we can deprecate with confidence
- Schema allows for marking fields as deprecated.

Powerful developer tooling

- Optimized, caching and offline client libraries
- Introspection
- Code generation
- IDE integrations



```
# Write your query or mutation here
query {
  allEmployees {
    given_name: first_name
    last_name
  }
}
```

```
{
  "data": {
    "allEmployees": [
      {
        "given_name": "Arkadiy",
        "last_name": "Brodskiy"
      },
      {
        "given_name": "Josh",
        "last_name": "Gilbert"
      },
      {
        "given_name": "Devin",
        "last_name": "Gaenz"
      },
      {
        "given_name": "Calvin",
        "last_name": "Webster"
      },
      {
        "given_name": "Bozo",
        "last_name": "Clown"
      },
      {
```

Let’s try to make something!

Demo Time

Thank you!

- GraphQL: https://graphql.org
- How To GraphQL: https://www.howtographql.com/
- Github: https://github.com/calweb
- Twitter: https://twitter.com/calweb
- Heavily borrowed from: https://www.slideshare.net/AmazonWebServices/introduction-to-graphql-95577414