

Introduction to GraphQL



Introduction to GraphQL

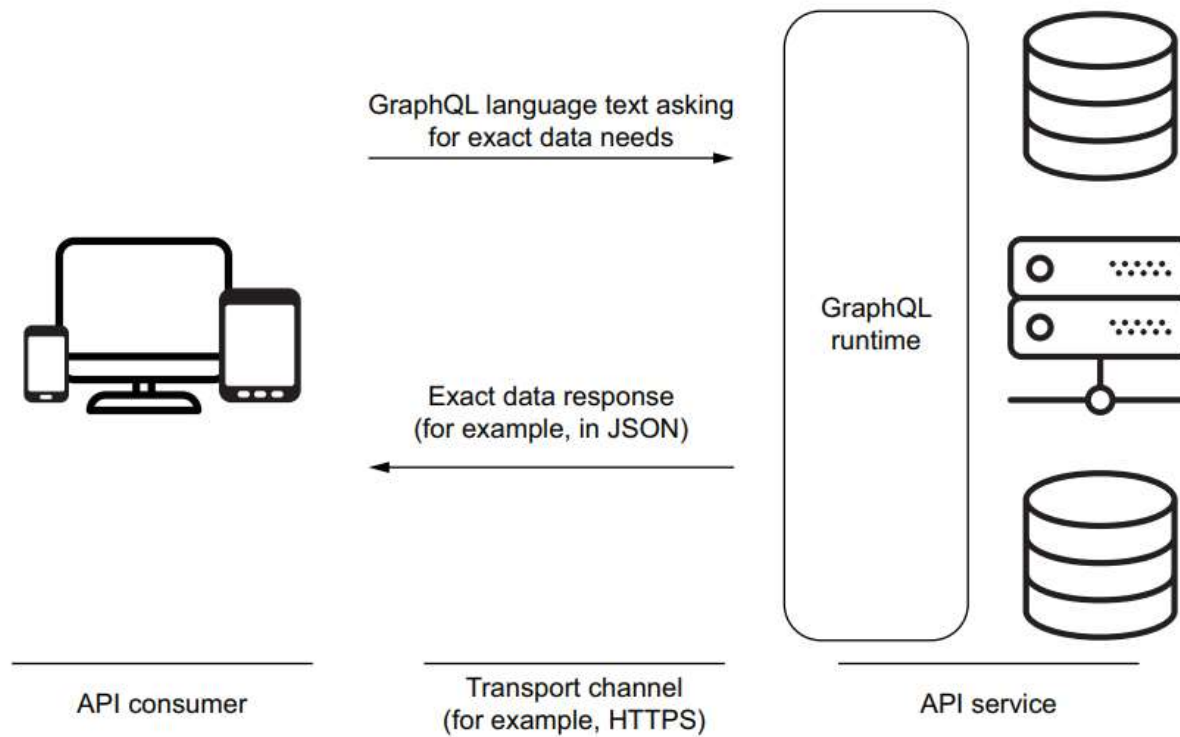
This lesson covers

- Understanding GraphQL and the design concepts behind it
- How GraphQL differs from alternatives like REST APIs
- Understanding the language used by GraphQL clients and services
- Understanding the advantages and disadvantages of GraphQL

What is GraphQL?

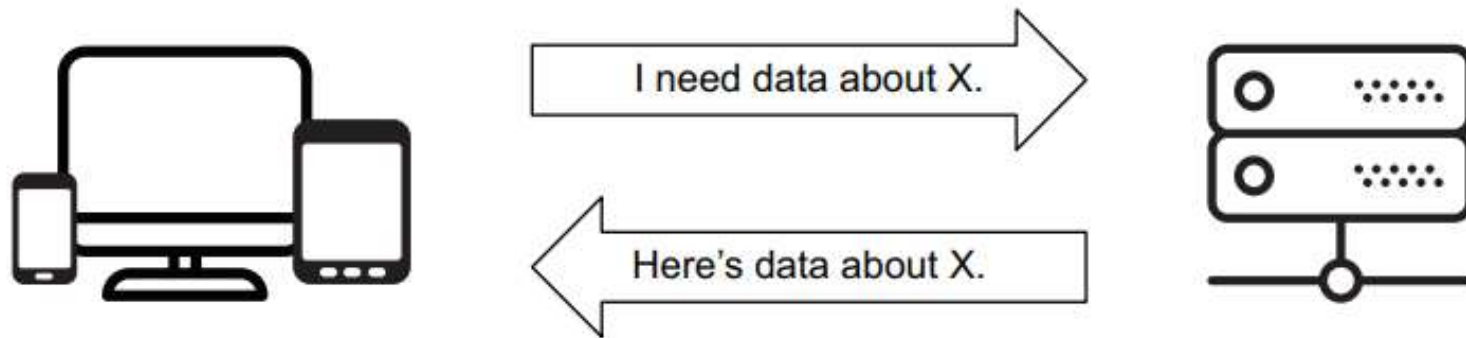
- The word graph in GraphQL comes from the fact that the best way to represent data in the real world is with a graph-like data structure.
- If you analyze any data model, big or small, you'll always find it to be a graph of objects with many relations between them.
- That was the first “Aha!” moment for me when I started learning about GraphQL.

What is GraphQL?



The big picture

- In general, an API is an interface that enables communication between multiple components in an application.



The big picture

- For example, if we have a table of data about a company's employees, the following is an example SQL statement to read data about the employees in one department.

```
SELECT id, first_name, last_name, email, birth_date, hire_date  
FROM employees  
WHERE department = 'ENGINEERING'
```

The big picture

- Here is another example SQL statement that inserts data for a new employee.

```
INSERT INTO employees (first_name, last_name, email, birth_date, hire_date)
VALUES ('Jane', 'Doe', 'jane@doe.name', '01/01/1990', '01/01/2020')
```

The big picture

- JSON is a language that can be used to communicate data.
- Here is a JSON object that can represent Jane's data.

```
{
  "data": {
    "employee": {
      "id": 42,
      "name": "Jane Doe",
      "email": "jane@doe.name",
      "birthDate": "01/01/1990",
      "hireDate": "01/01/2020"
    }
  }
}
```


The big picture

```
{
  "select": {
    "fields": ["name", "email", "birthDate", "hireDate"],
    "from": "employees",
    "where": {
      "id": {
        "equals": 42
      }
    }
  }
}
```

The big picture

- The following is how the previous data requirement can be expressed with a GraphQL query.

```
{  
  employee(id: 42) {  
    name  
    email  
    birthDate  
    hireDate  
  }  
}
```

GraphQL is a specification

- Although Facebook engineers started working on GraphQL in 2012, it was 2015 when they released a public specification document.
- You can see the current version of this document by navigating to az.dev/graphql-spec; it is maintained by a community of companies and individuals on GitHub.
- GraphQL is an evolving language, but the specification document was a genius start for the project because it defined standard rules and practices that all implementers of GraphQL runtimes must adhere to.

GraphQL is a language

- Though the Q (for query) is right there in the name, and querying is associated with reading, GraphQL can be used for both reading and modifying data.
- When you need to read data with GraphQL, you use queries; and when you need to modify data, you use mutations.
- Both queries and mutations are part of the GraphQL language.

GraphQL is a language

- A query language like GraphQL (or SQL) is different from programming languages like JavaScript and Python.
- You cannot use the GraphQL language to create user interfaces or perform complex computations.
- Query languages have more specific use cases, and they often require the use of programming languages to make them work.

GraphQL is a language

- We can use the English language to declaratively communicate data needs and fulfillments.
- For example, imagine that John is the client and Jane is the server, Here is an English data communication session:

John: “Hey Jane, how long does it take sunlight to reach planet Earth?”

Jane: “A bit over 8 minutes.”

John: “How about the light from the moon?”

Jane: “A bit under 2 seconds.”

GraphQL is a language

- For example, here's a hypothetical single GraphQL query that represents both of John's questions to Jan

```
{  
  timeLightNeedsToTravel(toPlanet: "Earth") {  
    fromTheSun: from(star: "Sun")  
    fromTheMoon: from(moon: "Moon")  
  }  
}
```

GraphQL is a service

- If we teach a client application to speak the GraphQL language, it will be able to communicate any data requirements to a backend data service that also speaks GraphQL.
- To teach a data service to speak GraphQL, you implement a runtime layer and expose that layer to the clients that want to communicate with the service.

GraphQL is a service

- A resolver function represents the instructions on how and where to access raw data.
- For example, a resolver function might issue a SQL statement to a relational database, read a file's data directly from the operating system, or update some cached data in a document database.

AN EXAMPLE OF A SCHEMA AND RESOLVERS

- To understand how resolvers work, let's take the query and assume a client sent it to a GraphQL service.

```
query {  
  employee(id: 42) {  
    name  
    email  
  }  
}
```

GraphQL is a service

- Here's an example to represent the Employee type using GraphQL's schema language

```
type Employee(id: Int!) {  
  name: String!  
  email: String!  
}
```

GraphQL is a service

- The employee field's resolver function for employee #42 might return an object like the following

```
{  
  "id": 42,  
  "first_name": "Jane",  
  "last_name": "Doe",  
  "email": "jane@doe.name",  
  
  "birth_date": "01/01/1990",  
  "hire_date": "01/01/2020"  
}
```

GraphQL is a service

- Let's say we have the following (JavaScript) functions representing the server resolver functions for the name and email fields:

```
// Resolver functions
const name => (source) => `${source.first_name} ${source.last_name}`;
const email => (source) => source.email;
```

GraphQL is a service

- The GraphQL service uses all the responses of these three resolver functions to put together the following single response for the query in listing

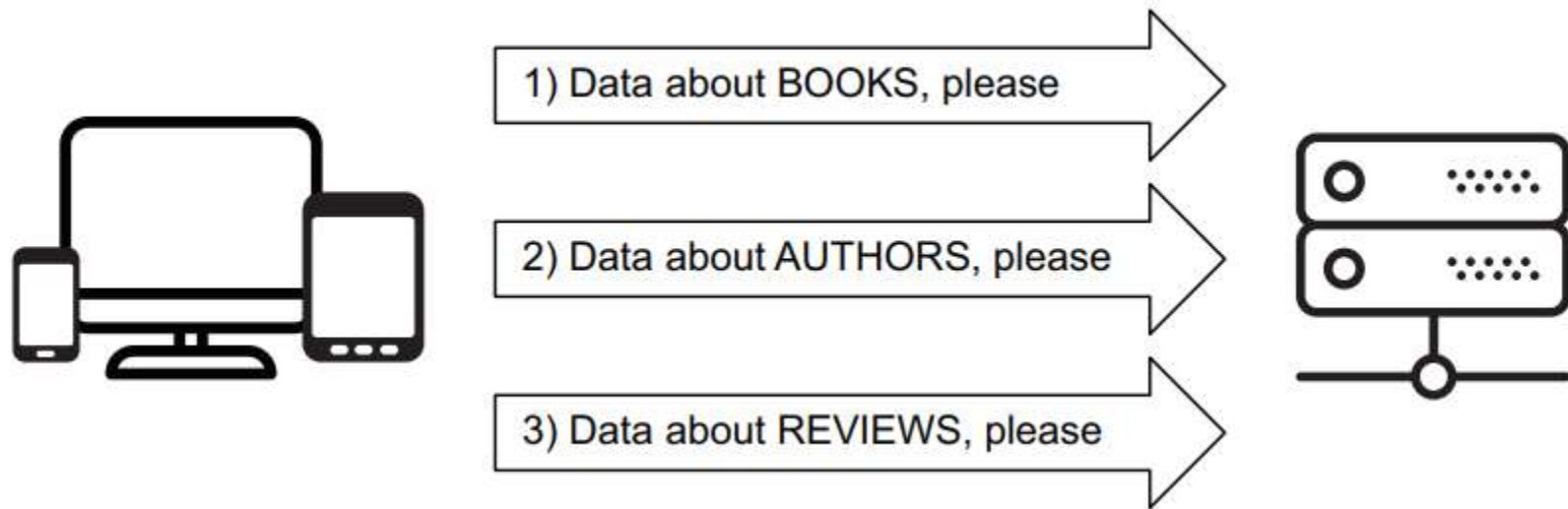
```
{
  data: {
    employee: {
      name: 'Jane Doe',
      email: 'jane@doe.name'
    }
  }
}
```

Why GraphQL?

- GraphQL is not the only—or even the first—technology to encourage creating efficient data APIs.
- You can use a JSON-based API with a custom query language or implement the Open Data Protocol (OData) on top of a REST API.
- Experienced backend developers have been creating efficient technologies for data APIs since long before GraphQL.

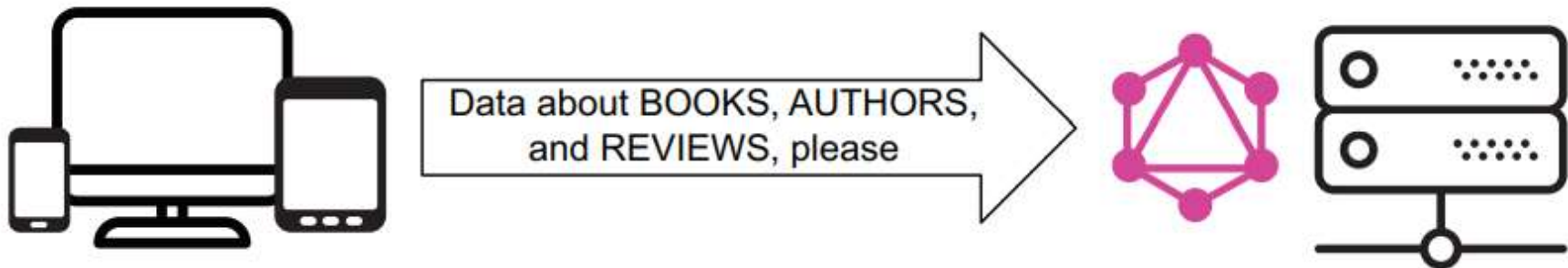
Why GraphQL?

- The client ends up having to communicate with the server multiple times to gather all the data it needs



Why GraphQL?

- The client asks the GraphQL service a single question and gets a single response with precisely what the client needs



Why GraphQL?

This section contains the following lessons:

- lesson 1, Getting Started with Ansible
- lesson 2, Understanding the Fundamentals of Ansible
- lesson 3, Defining Your Inventory
- lesson 4, Playbooks and Roles

What about REST APIs?

- GraphQL APIs are often compared to REST APIs because the latter have been the most popular choice for data APIs demanded by web and mobile applications.
- GraphQL provides a technological alternative to REST APIs, But why do we need an alternative? What is wrong with REST APIs? The biggest relevant problem with REST APIs is the client's need to communicate with multiple data API endpoints.

The GraphQL way

To see the GraphQL way of solving the REST API problems we have talked about, you need to understand the concepts and design decisions behind GraphQL. Let's review the major ones.

- THE TYPED GRAPH SCHEMA
- THE DECLARATIVE LANGUAGE
- THE SINGLE ENDPOINT AND CLIENT LANGUAGE
- THE SIMPLE VERSIONING

REST APIs and GraphQL APIs in action

- Let's go over a one-to-one comparison example between a REST API and a GraphQL API. Imagine that you are building an application to represent the Star Wars films and characters.
- The first UI you tackle is a view to show information about a single Star Wars character.
- This view should display the character's name, birth year, the name of their planet, and the titles of all the films in which they appeared.

```
{
  "data": {
    "person": {
      "name": "Darth Vader",
      "birthYear": "41.9BBY",
      "planet": {
        "name": "Tatooine"
      },
      "films": [
        { "title": "A New Hope" },
        { "title": "The Empire Strikes Back" },
        { "title": "Return of the Jedi" },
        { "title": "Revenge of the Sith" }
      ]
    }
  }
}
```

REST APIs and GraphQL APIs in action

- Assuming that a data service can give us this exact structure, here is one possible way to represent its view with a frontend component library like React.js.

```
// The Container Component:  
<PersonProfile person={data.person}></PersonProfile>  
// The PersonProfile Component:  
Name: {data.person.name}  
  Birth Year: {data.person.birthYear}  
  Planet: {data.person.planet.name}  
  Films: {data.person.films.map(film => film.title)}
```

REST APIs and GraphQL APIs in action

- Assuming that you know that character's ID, a REST API is expected to expose that information with an endpoint like this:


GET - /people/{id}

REST APIs and GraphQL APIs in action

- The JSON response for this request could be something like the following:

```
{
  "name": "Darth Vader",
  "birthYear": "41.9BBY",
  "planetId": 1
  "filmIds": [1, 2, 3, 6],
  . . . . .
}
```

**Other information
that is not needed
for this view**



REST APIs and GraphQL APIs in action

Then, to read the planet's name, you ask

```
GET - /planets/1
```

And to read the film titles, you ask

```
GET - /films/1
```

```
GET - /films/2
```

```
GET - /films/3
```

```
GET - /films/6
```

REST APIs and GraphQL APIs in action

- For example, if the API server implemented nested resources and understood the relationship between a person and a film, you could read the film data (along with the character data) with something like this:

GET - /people/{id}/films

REST APIs and GraphQL APIs in action

- Since you want to ask for data in a single network round trip, there has to be a way for you to express the complete data needs for the server to parse.
- You do this with a GraphQL query:

GET or POST - /graphql?query={·-·-·}

In English	In GraphQL
<p>The view needs:</p> <p>a person's name,</p> <p>birth year,</p> <p>planet's name,</p> <p>and the titles of all their films.</p>	<pre>{ person(ID:) { name birthYear planet { name } films { title } } }</pre>

GraphQL query (question)	Needed JSON (answer)
<pre> { person(ID:) { name birthYear planet { name } films { title } } } </pre>	<pre> { "data": { "person": { "name": "Darth Vader", "birthYear": "41.9BBY", "planet": { "name": "Tatooine" }, "films": [{ "title": "A New Hope" }, { "title": "The Empire Strikes Back" }, { "title": "Return of the Jedi" }, { "title": "Revenge of the Sith" }] } } } </pre>

REST APIs and GraphQL APIs in action

If the answer statement is

The name of the Star Wars character who has the ID 4 is Darth Vader.

A good representation of the question is the same statement without the answer part:

(What is) the name of the Star Wars character who has the ID 4?

REST APIs and GraphQL APIs in action

```
{  
  person(personID: 4) {  
    name  
    birthYear  
    homeworld {  
      name  
    }  
    filmConnection {  
      films {  
        title  
      }  
    }  
  }  
}
```

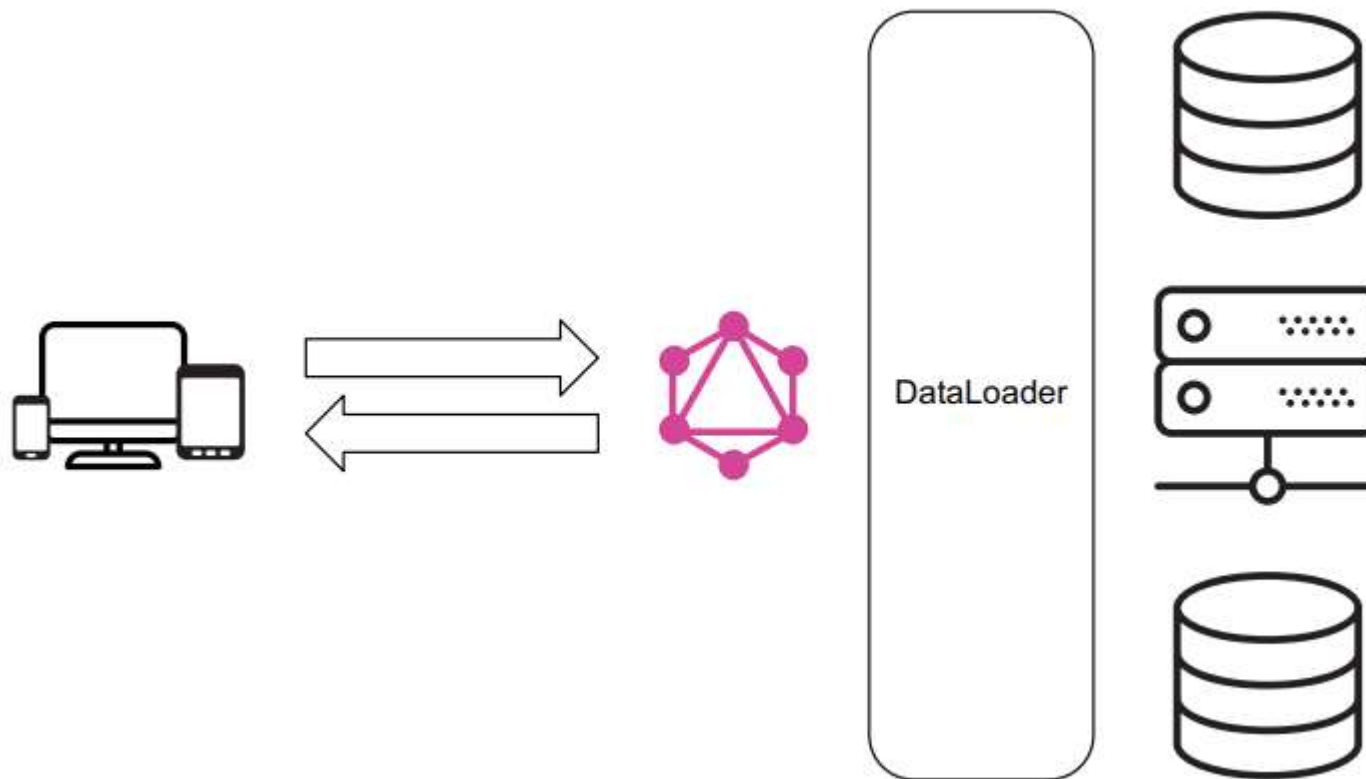

GraphQL problems

Security

- A critical threat for GraphQL APIs is resource-exhaustion attacks (aka denial-of-service attacks).
- A GraphQL server can be attacked with overly complex queries that consume all the server resources.
- It is very simple to query for deeply nested relationships (user → friends → friends → friends ...) or use field aliases to ask for the same field many times.

Caching and optimizing

- One task that GraphQL makes a bit more challenging is clients' caching of data. Responses from REST APIs are a lot easier to cache because of their dictionary nature.
- A specific URL gives certain data, so you can use the URL itself as the cache key. With GraphQL, you can adopt a similar basic approach and use the query text as a key to cache its response.
- But this approach is limited, is not very efficient, and can cause problems with data consistency.



Learning curve

- Working with GraphQL requires a bigger learning curve than the alternatives.
- A developer writing a GraphQL-based frontend application has to learn the syntax of the GraphQL language.
- A developer implementing a GraphQL backend service has to learn a lot more than just the language: they have to learn the API syntax of a GraphQL implementation.

Summary

- The best way to represent data in the real world is with a graph data structure.
- A data model is a graph of related objects. GraphQL embraces this fact.
- A GraphQL system has two primary components: the query language, which can be used by consumers of data APIs to request their exact data needs; and the runtime layer on the backend, which publishes a public schema describing the capabilities and requirements of data models

"Complete Lab"