DEALSYTE

Introduction to

APOLLO

by Clemente Serrano

BeerJS, Santiago 2020

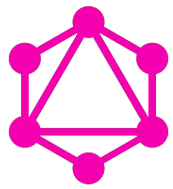
GraphQL, the core of Apollo

DEALSYTE

- GraphQL got hugely popular as an alternative approach to building an API over REST
- Is a **query language for APIs** and a **runtime** for fulfilling those queries with your existing data:
  - Ask for what you need, get exactly that
  - Get many resources in a single request
  - Describe what's possible with a type system

GraphQL, the core of Apollo

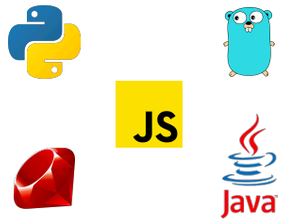
The big picture



Schema

Types and fields on those types

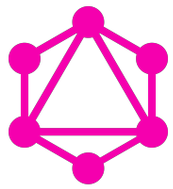
+



Resolvers

Functions for each field on each type

=



Service

```
type Product {
  id: ID!
  createdAt: Date!
  images: [String]
  name: String
  price: Int
 }

type Mutation {
  createProduct(
    images:[String]
    name: String
    price: Int
  ): Product
 }
```

```
type Query {
  getProduct(id:ID!):Product
  getProducts: [Product]
 }

input PaymentInput {
  transactionAmount: Float!
  paymentMethodId: ID!
  products: [ID!]!
 }

enum CardType {
  CREDIT_CARD
  DEBIT_CARD
}
```

```
function createProduct(parent,args,ctx){
  try {
    return Product.create(args);
  } catch (error) {
    throw new Error(error)
  }
};

function getProduct(parent, { _id }) {
  try {
    return Product.findOne({ _id });
  } catch (error) {
    throw new Error(error);
  };
}
```

```
query {
  getProducts {
    id
    name
    images
  }
}

mutation  {
  createProduct(price: 30) {
    id
    name
   }
}
```

GraphQL, the core of Apollo

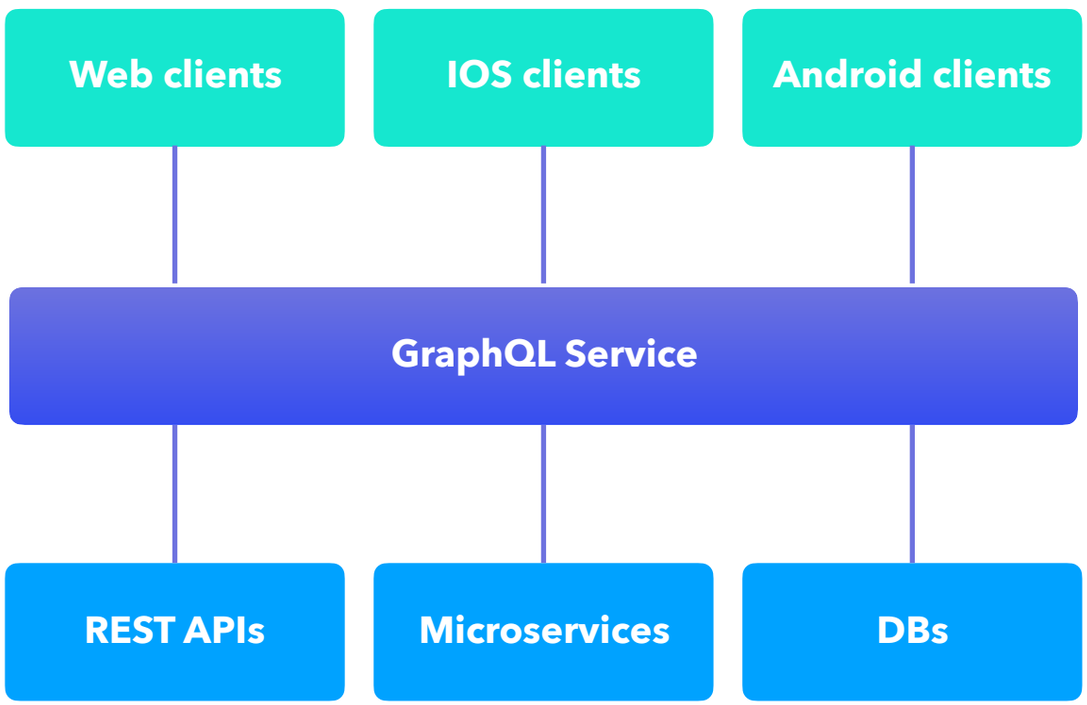
Think in Graph

```
GET api/products/buyers

GET api/products/profits

GET api/products/images
```

```
query {
  getProducts {
    buyers {
      id
      name
    }
    profits {
      amount
      buyer {
        id
        address
      }
    }
    images
  }
}
```



DEALSYTE

GraphQL serves over HTTP

REST uses "resources" as its core concept. GraphQL uses **entity graphs** as concept.

As a result, entities in GraphQL are not identified by URLs. Instead, a GraphQL server operates on a single URL/endpoint:

```
curl \
  https://api/graphql?query={getProducts{id,name}}
```

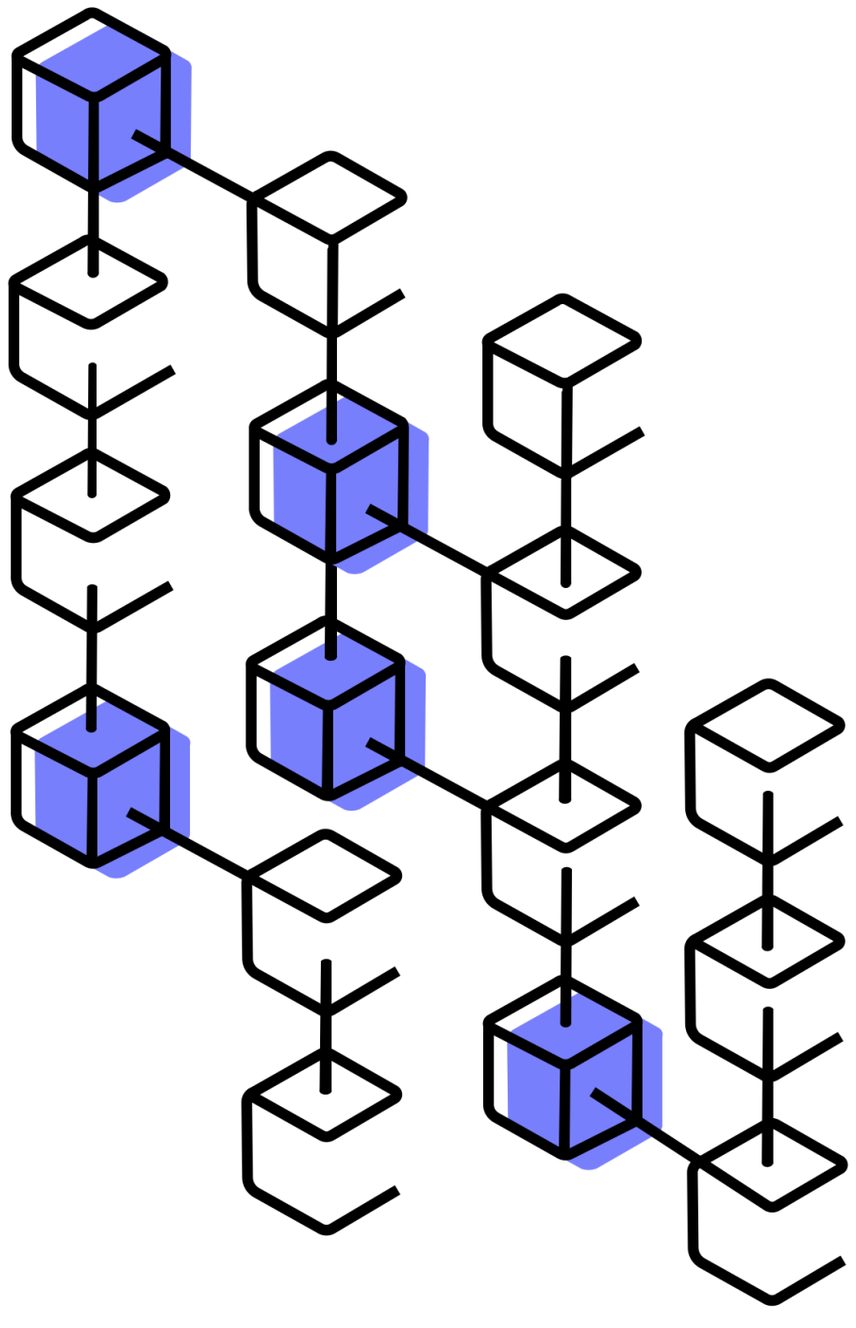
```
curl \
  -X POST \
  -H "Content-Type: application/json" \
  --data '{ "query": "{ createProduct:
{id,name}}", "variables": {"_id":"1"}}' \
  https://api/graphql
```

GraphQL, the core of Apollo

Graph execution

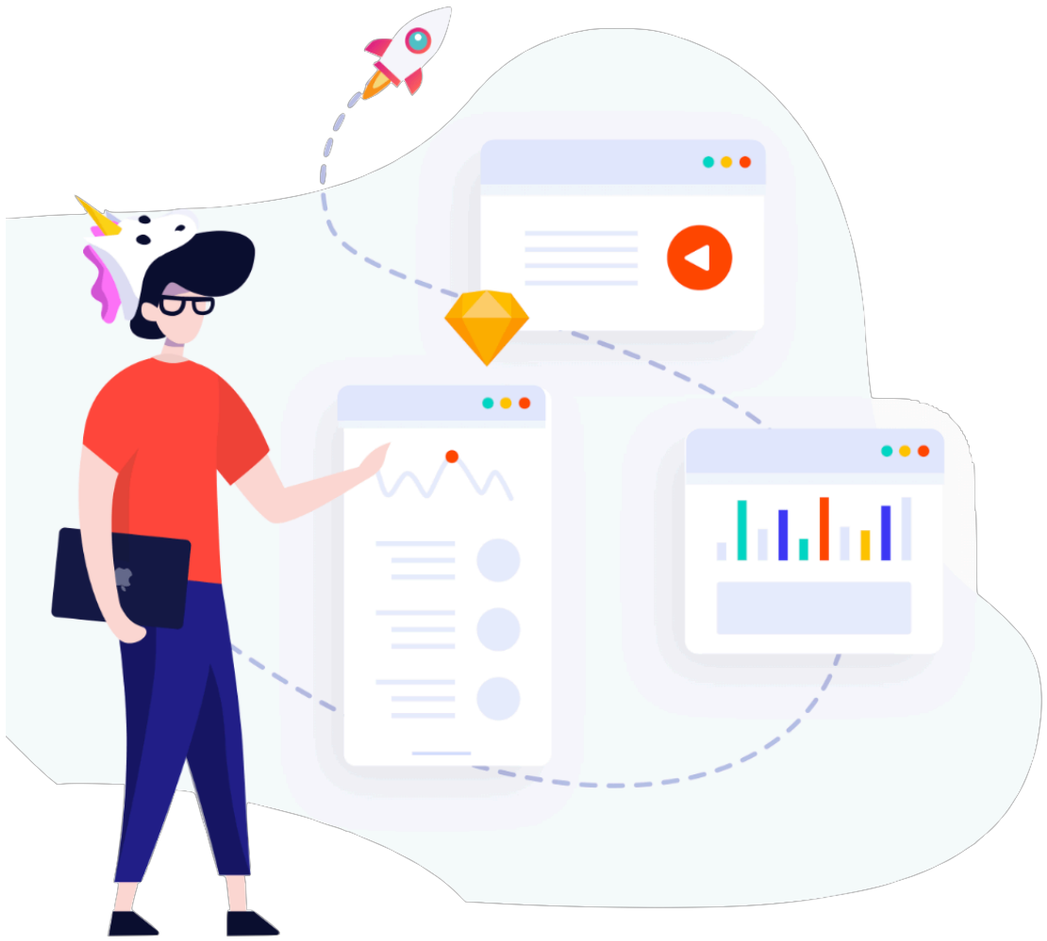
DEALSYTE

- Queries are executed by a GraphQL server which returns a result that mirrors the shape of the requested query
- If a field produces a scalar value, then the execution completes.
- However if a field produces an object value then the query will contain another selection of fields which apply to that object and the field resolver will be executed again



GraphQL, the core of Apollo

Schemas



A GraphQL service uses a **schema** to describe the shape of the data graph (through a **schema definition language**, or **SDL**, that you use to define your schema and store it as a string)

The schema defines a hierarchy of **types** with **fields** that are populated from the back-end data stores

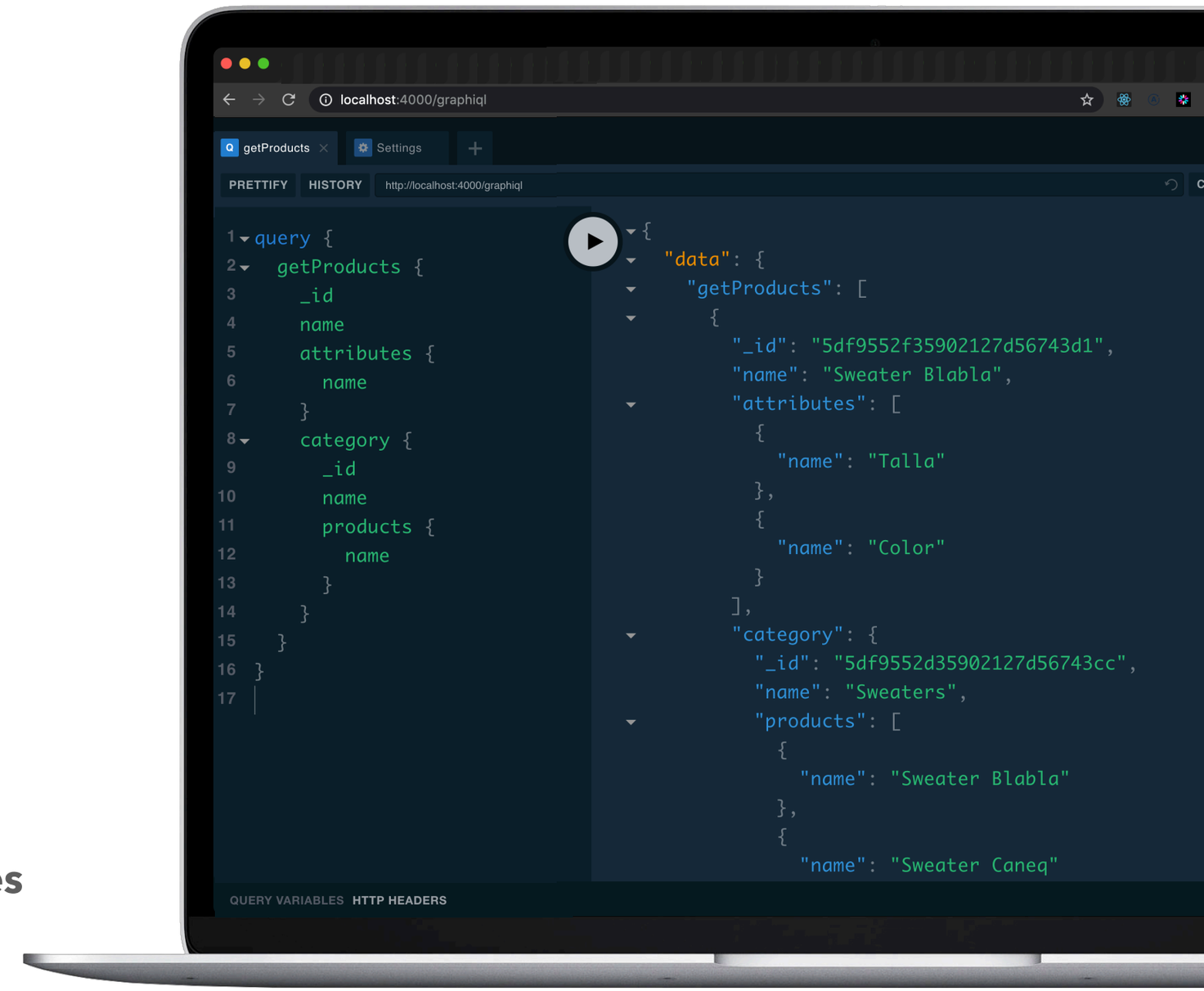
Schema: collection of types and the relationships *between* those types

GraphQL, the core of Apollo

Schemas

DEALSYTE

- **Scalar types**: `Int`, `Float`, `String`, `Boolean`, `ID`, among others
- **Object types**: contains a collection of fields, each of which can be either a scalar type or *another* object type
- **Query and Mutation types**: entry points of the data graph. defines exactly which GraphQL queries clients can execute
- **Input types**: allow to pass objects as arguments to queries and mutations
- **Interfaces, enums and other types**



GraphQL, the core of Apollo

Resolvers

DEALSYTE

- Each resolver represents a single field. Defines the value of that field
- Provide the instructions for turning a GraphQL operation into data
- 4 arguments:
  - **`parent`**: Result returned from the resolver on the parent field
  - **`args`**: Arguments passed into the field in the query
  - **`ctx`**: Object shared by all resolvers in a particular query
  - **`info`**: Info about the execution state of the query

GraphQL, the core of Apollo

Resolvers

DEALSYTE

- Posible results:
  - **`null` | `undefined`**: The object could not be found
  - **An array**: The sub-selection of the query will run once for every item in this array
  - **A promise**: Resolvers often do asynchronous actions like fetching from a database or backend API
  - **A scalar or object value**

Apollo

DEALSYTE

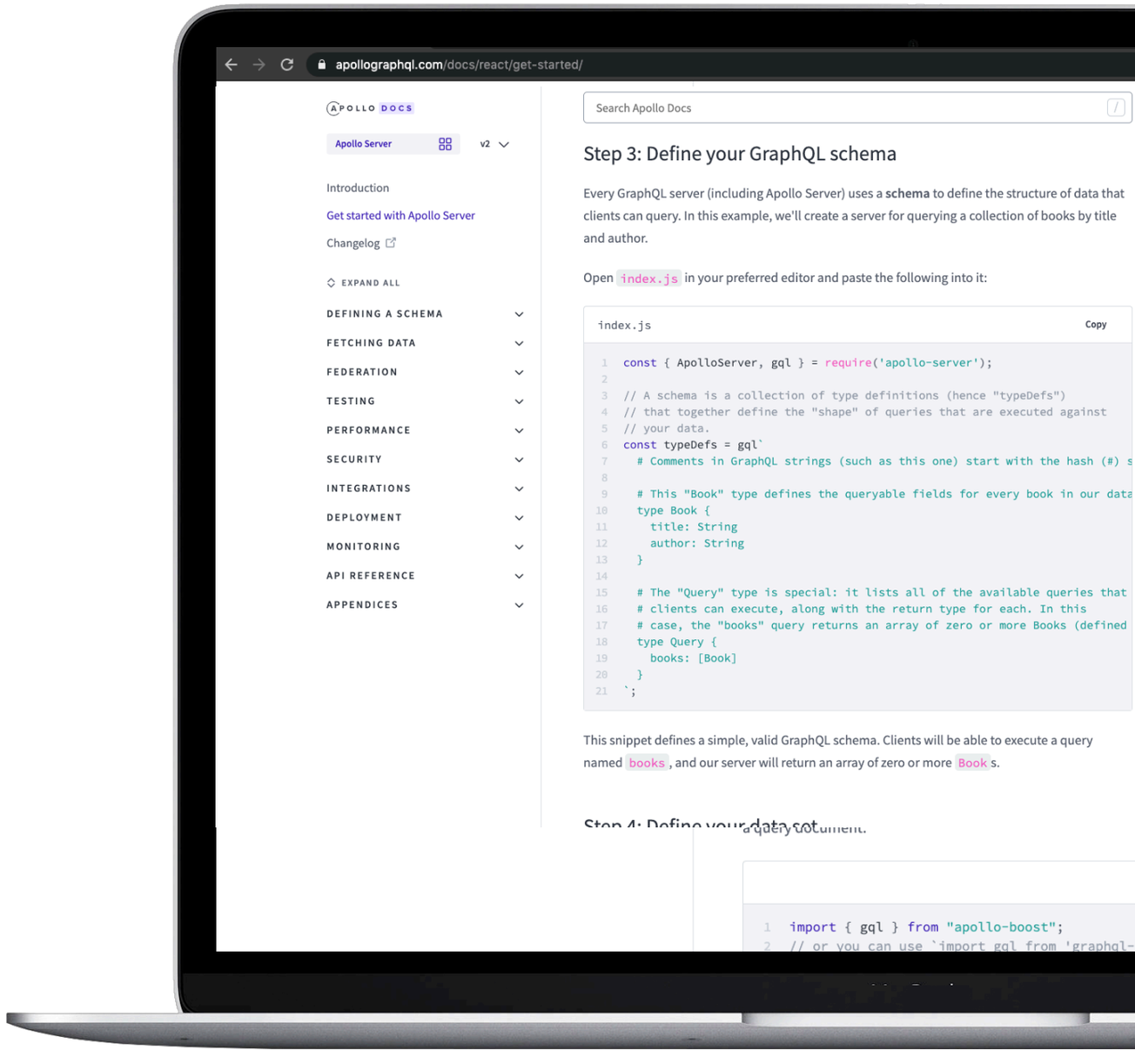
- An industry-standard GraphQL implementation
- Connects your application clients (such as React and iOS apps) to your back-end services:
- **Apollo Engine:** Apollo's cloud service which provides schema versioning, metrics, and enhanced security
- **Apollo Link:** Custom chain of actions that client performs with each GQL op
- **Apollo Server**
- **Apollo Client**

Apollo

Apollo Server

DEALSYTE

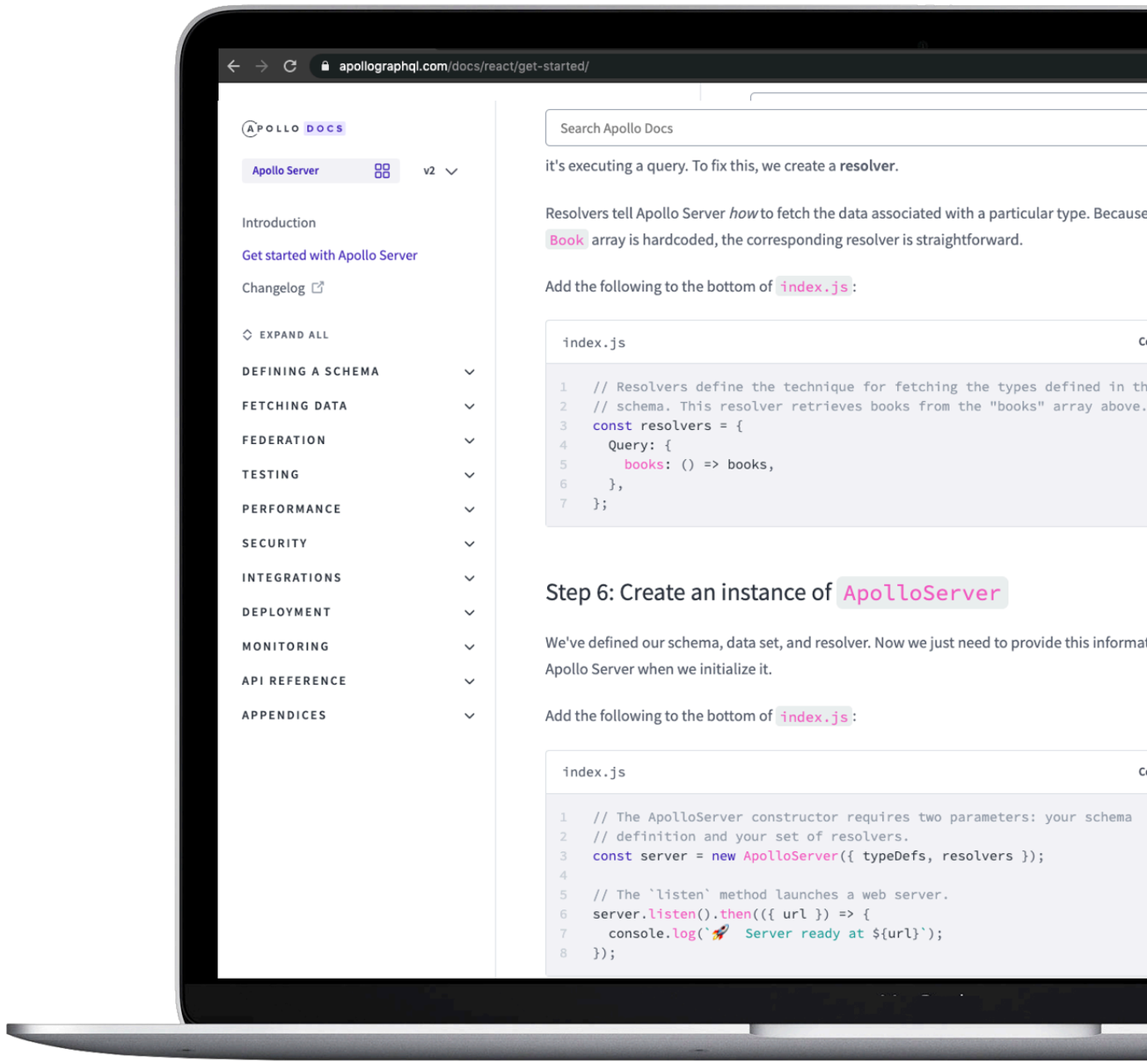
- JavaScript GraphQL server for defining a *schema* and a set of *resolvers* that implement each part of that schema
- 7 simple steps to set it up:
  - Start a node server
  - Install basic dependencies (`apollo-server` and `graphql`)
  - Define a GQL schema (`typedefs` via `gql` tool)
  - Connect to a data source (`mocks` in our case)



Apollo

Apollo Server

- Define resolvers for the specified schema
- Create an `ApolloServer` instance
- Run and query de server (GraphiQL)

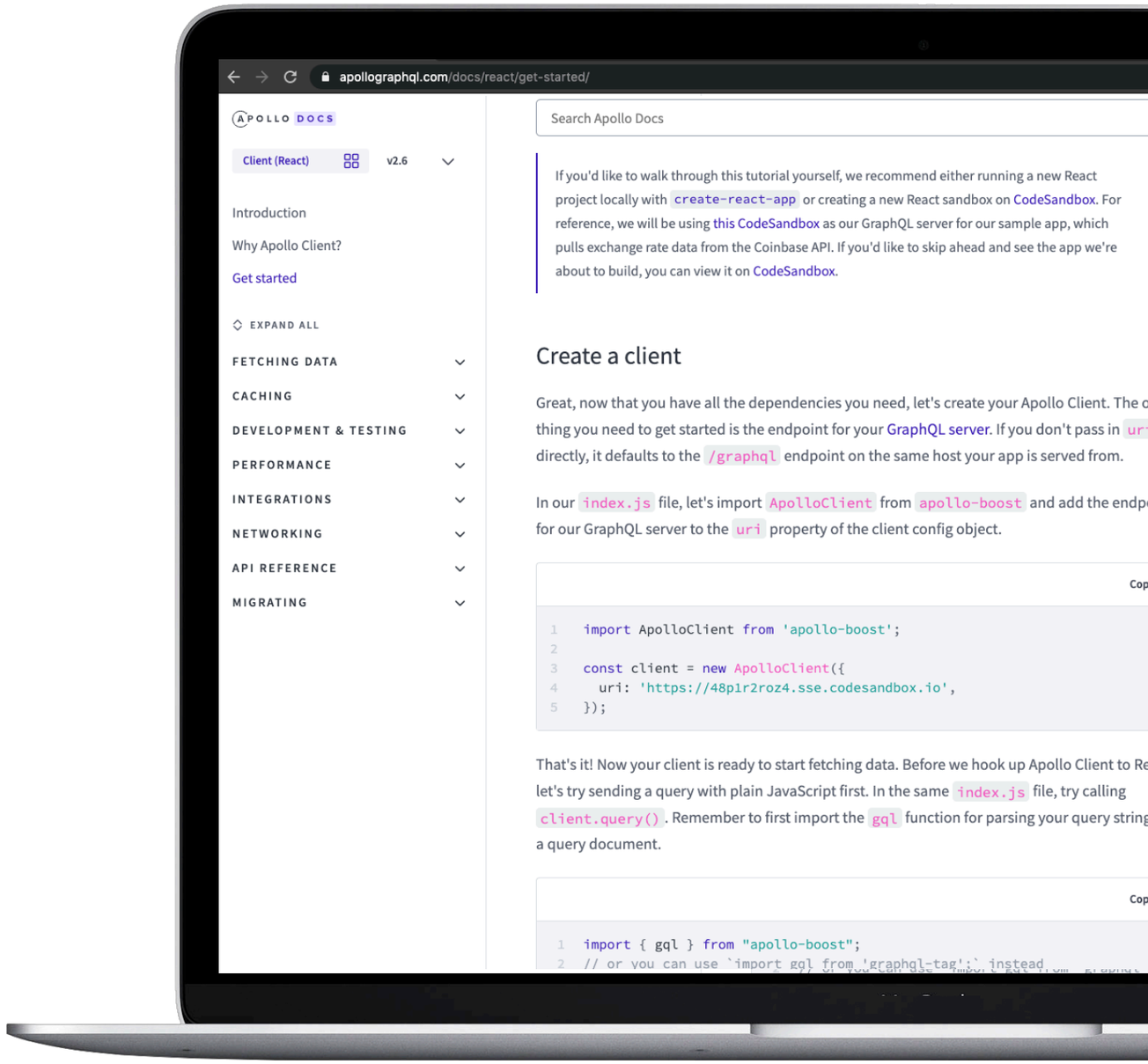


Apollo

Apollo Client (React)

DEALSYTE

- Complete state mgmt library for React. Just write GQL queries, and Apollo Client will take care of requesting and caching data, as well as updating the app UI
- 6 simple steps to set it up:
  - Start a React project (CRA preferred)
  - Install basic dependencies (`apollo-boost @apollo/react-hooks graphql`)
  - Create a client (`ApolloClient` via `apollo-boost` library)

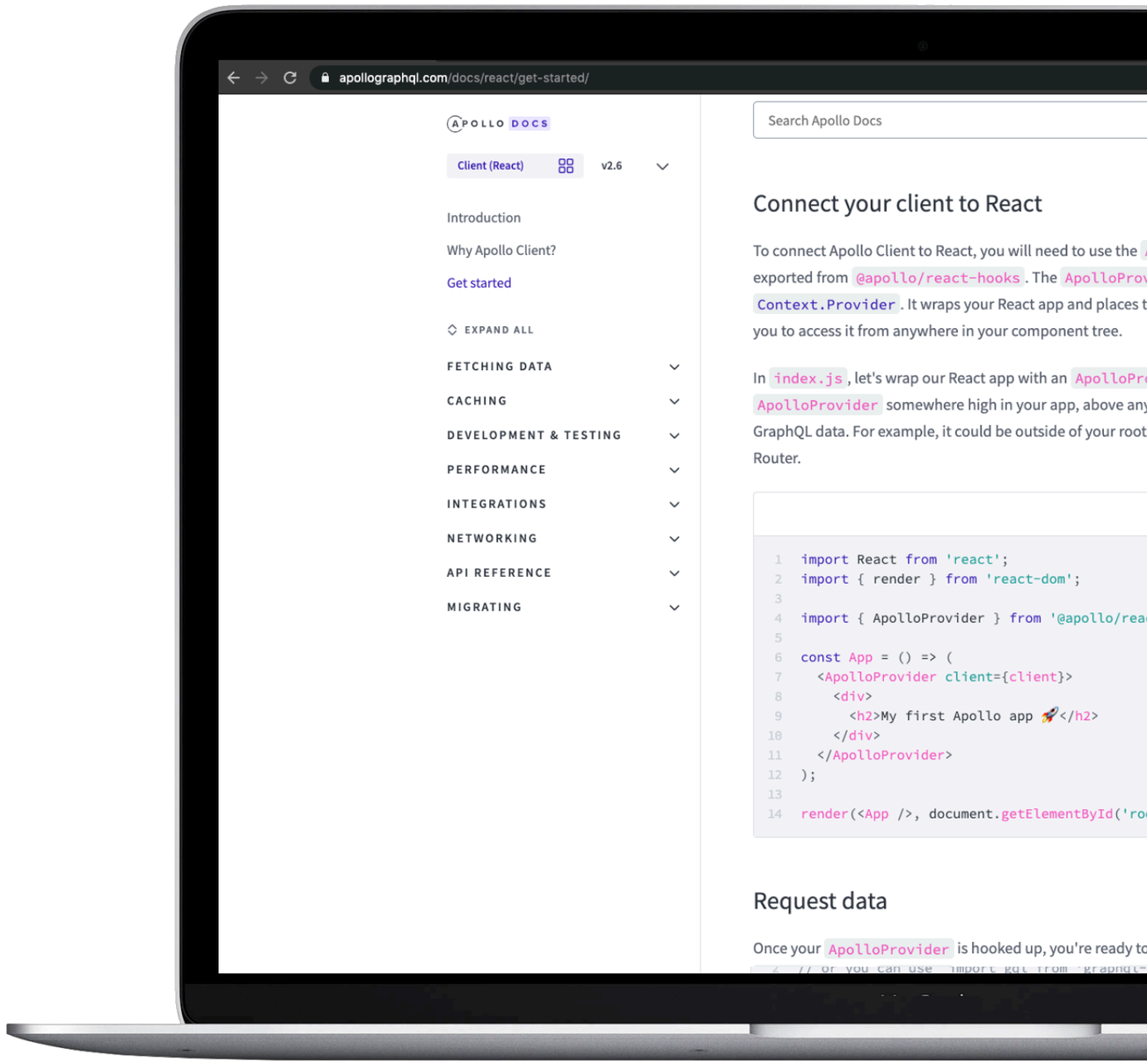


Apollo

Apollo Client (React)

DEALSYTE

- Connect Apollo Client to React (`ApolloProvider` from `apollo/react-hooks`)
- Request data (with render-props or hooks!)
- Verify everything on your browser ;)



DEALSYTE

Thanks for coming!