

Introduction to



APOLO

The Apollo logo consists of the word "APOLO" in a bold, white, sans-serif font. The letter "A" is enclosed within a white circle that has a small black dot at the top-right corner, suggesting a celestial body like a planet or moon.

by Clemente Serrano

BeerJS, Santiago 2020

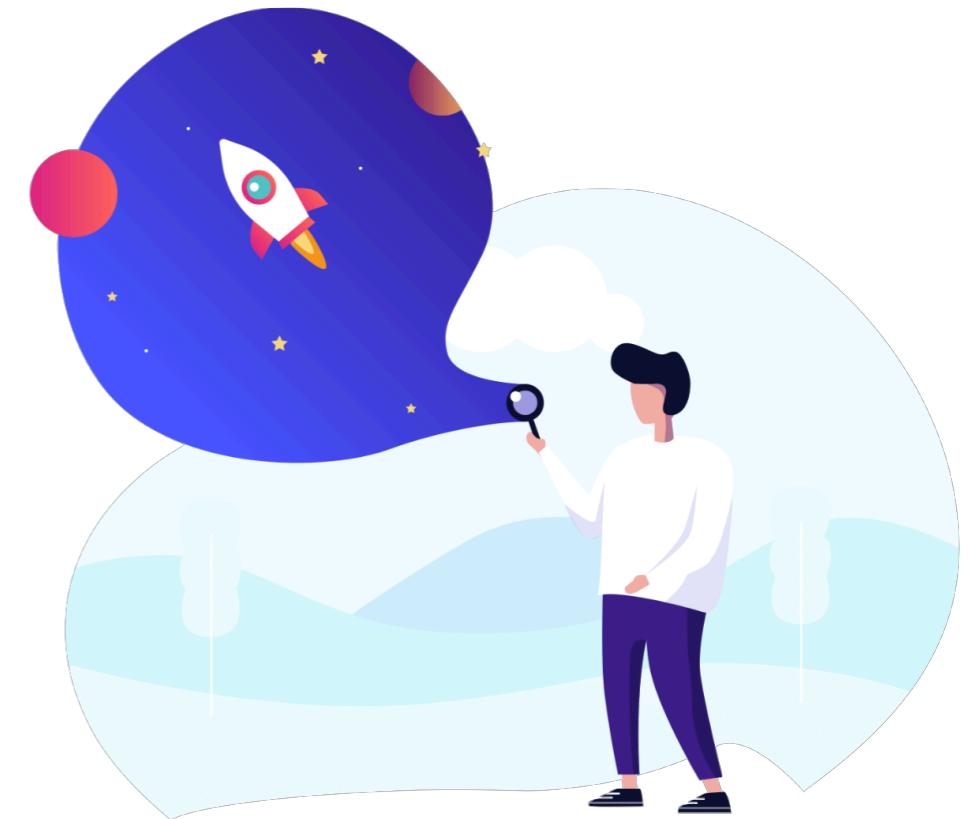
GraphQL, the core of Apollo



- GraphQL got hugely popular as an alternative approach to building an API over REST

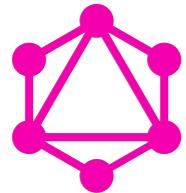
- Is a **query language for APIs** and a **runtime** for fulfilling those queries with your existing data:

- Ask for what you need, get exactly that
- Get many resources in a single request
- Describe what's possible with a type system



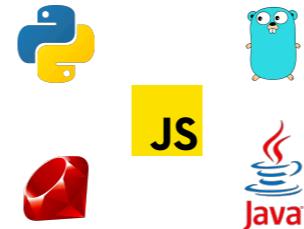
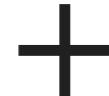
GraphQL, the core of Apollo

The big picture



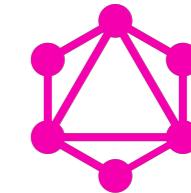
Schema

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  
}
```

```
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  
    }  
}
```

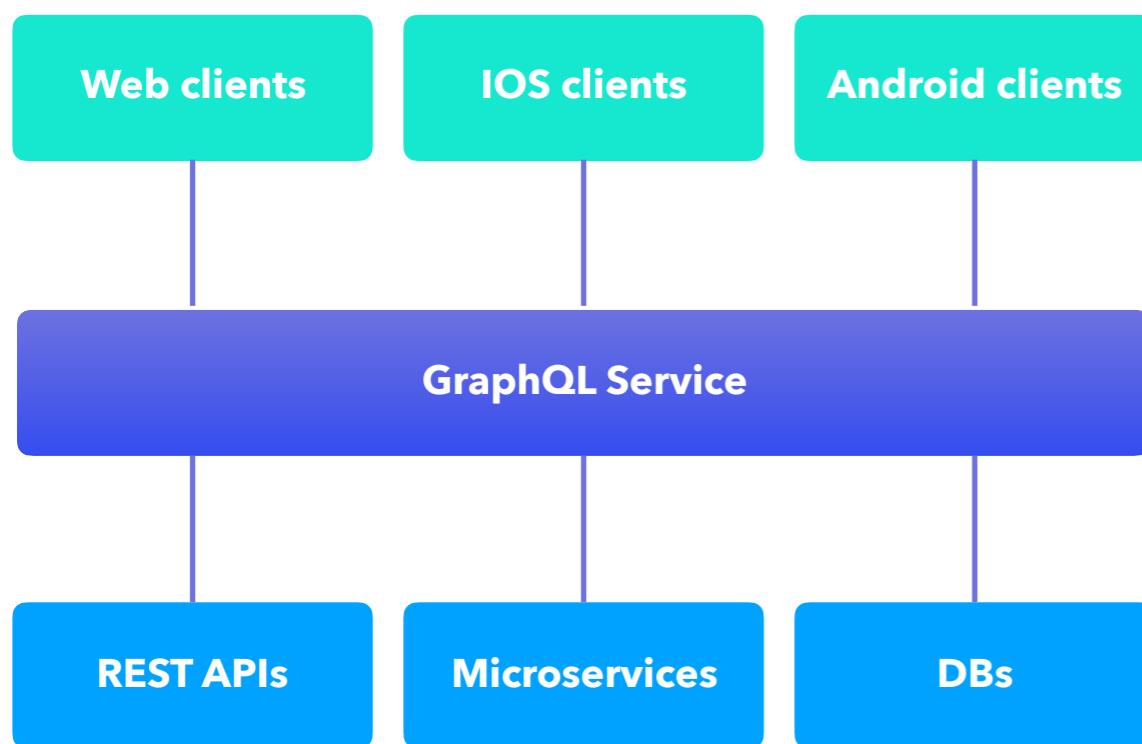
GraphQL, the core of Apollo

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  
  }  
}
```



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:

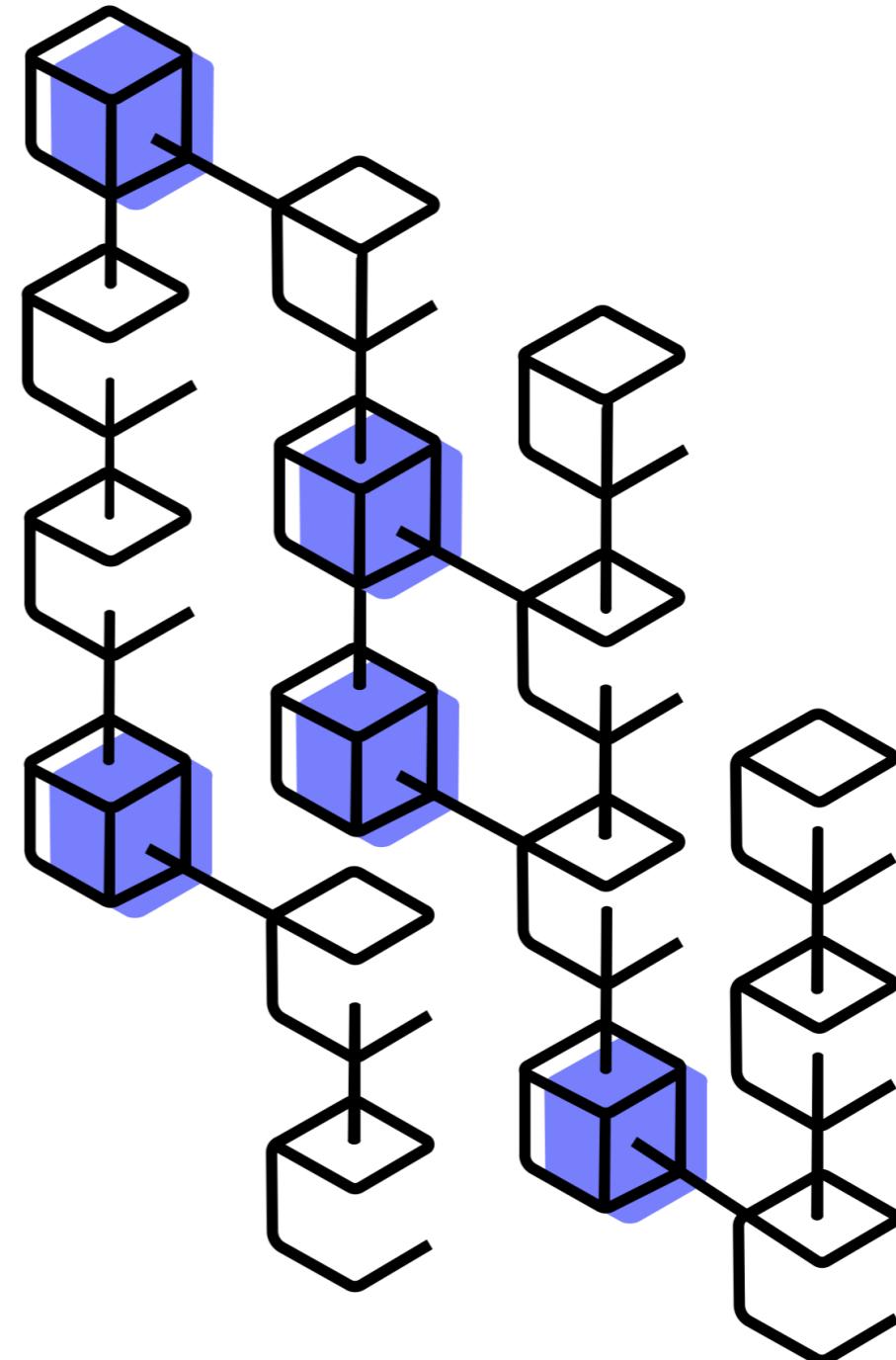
```
curl \  
https://api/graphql?query={getProducts{id,name}}
```

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

GraphQL, the core of Apollo

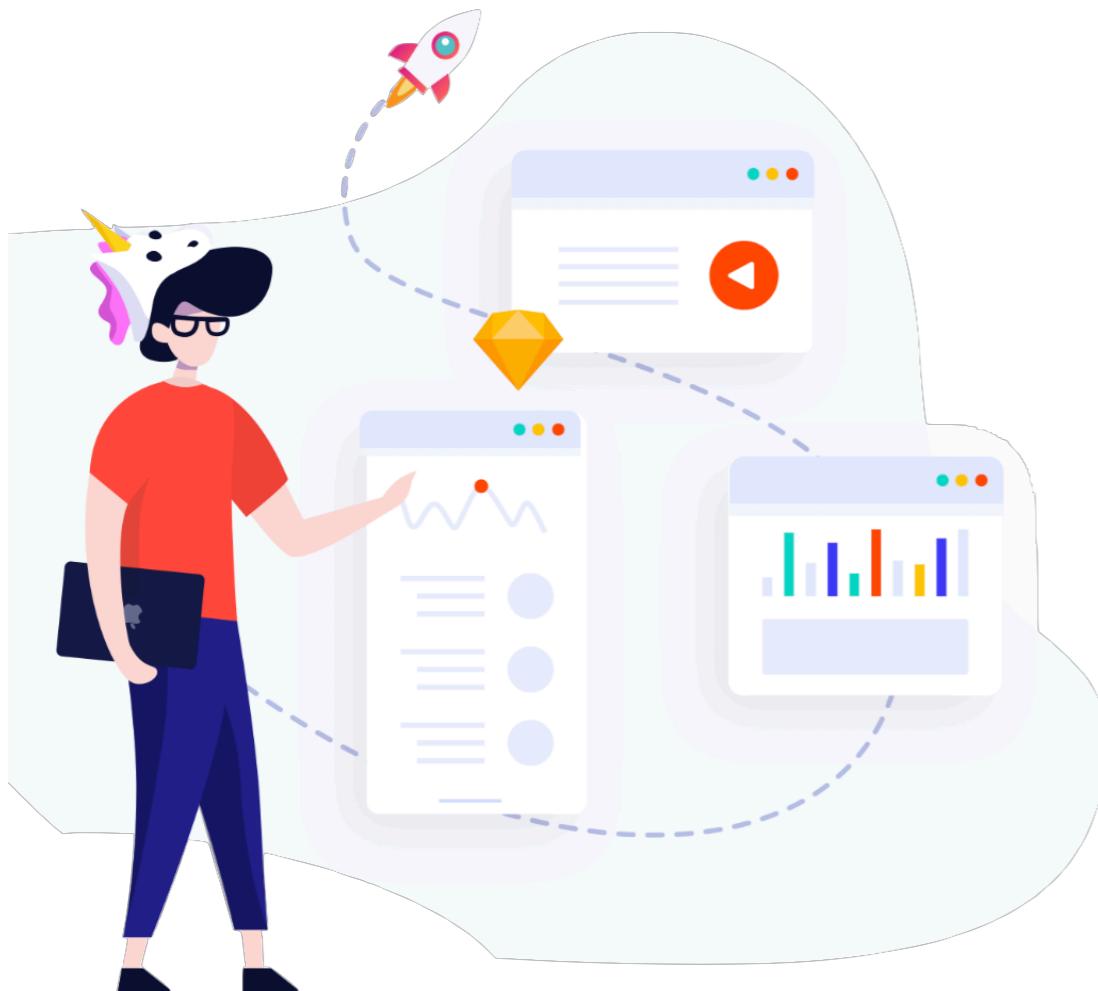
Graph execution

- 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



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**

A screenshot of the GraphQL playground interface on a Mac OS X desktop. The window title is 'localhost:4000/graphiql'. The left pane shows a code editor with a query:

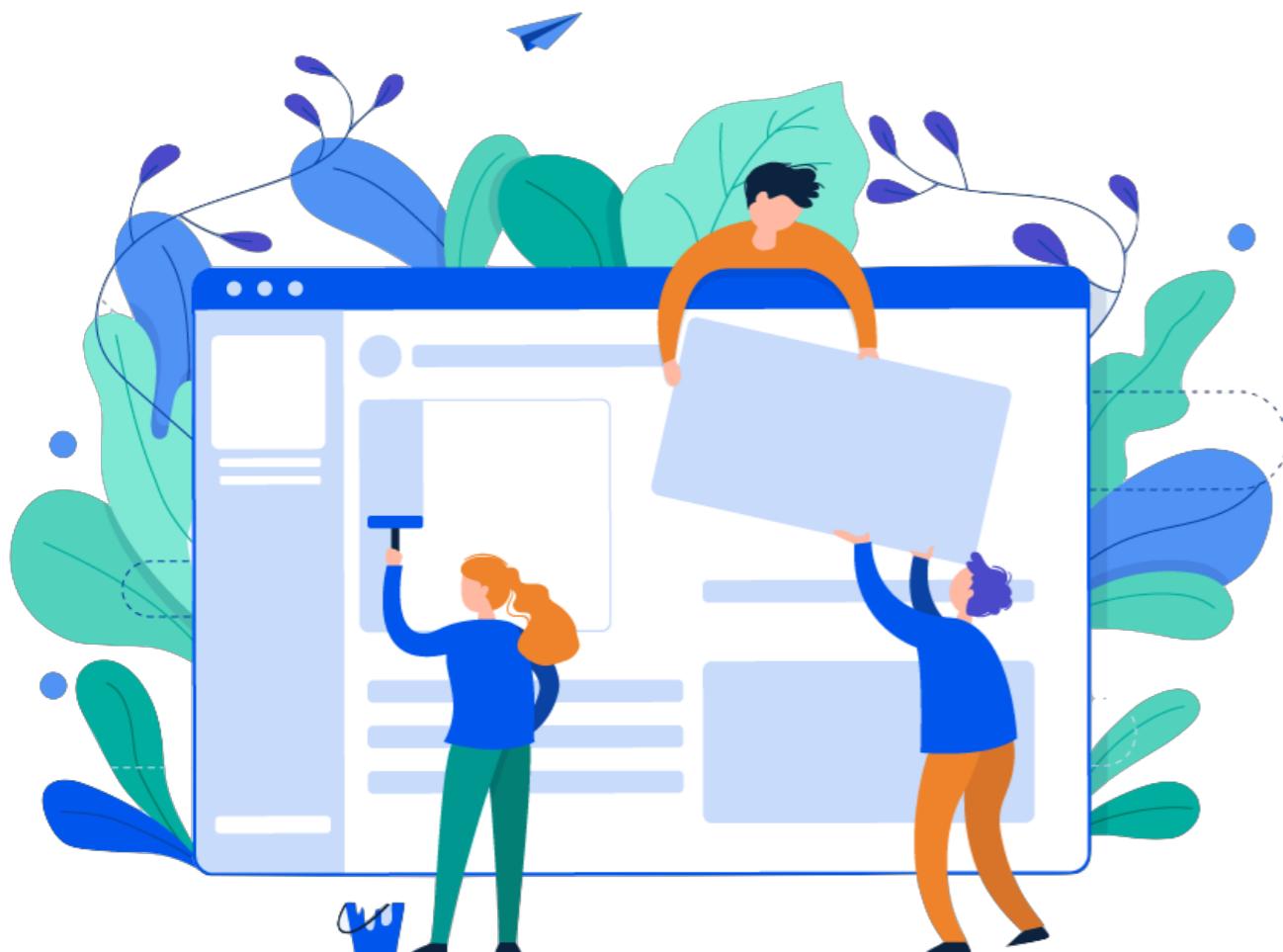
```
query { getProducts { _id name attributes { name } category { _id name products { name } } } }
```

. The right pane shows the resulting JSON data. A circular arrow icon is overlaid on the JSON tree view.

```
data: { getProducts: [ { _id: "5df9552f35902127d56743d1", name: "Sweater Blabla", attributes: [ { name: "Talla" }, { name: "Color" } ], category: { _id: "5df9552d35902127d56743cc", name: "Sweaters", products: [ { name: "Sweater Blabla" }, { name: "Sweater Caneq" } ] } } ] }
```

GraphQL, the core of Apollo

Resolvers



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

- Possible 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**





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



- 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)

The screenshot shows a laptop displaying the Apollo Docs website at `apollographql.com/docs/react/get-started/`. The page title is "Step 3: Define your GraphQL schema". It explains that every GraphQL server uses a schema to define the structure of data that clients can query. A code snippet for `index.js` is provided:

```
const { ApolloServer, gql } = require('apollo-server');

// A schema is a collection of type definitions (hence "typeDefs")
// that together define the "shape" of queries that are executed against
// your data.
const typeDefs = gql`
  # Comments in GraphQL strings (such as this one) start with the hash (#)
  # This "Book" type defines the queryable fields for every book in our data
  type Book {
    title: String
    author: String
  }

  # The "Query" type is special: it lists all of the available queries that
  # clients can execute, along with the return type for each. In this
  # case, the "books" query returns an array of zero or more Books (defined
  type Query {
    books: [Book]
  }
`;
```

This snippet defines a simple, valid GraphQL schema. Clients will be able to execute a query named `books`, and our server will return an array of zero or more `Book`s.

Below the code snippet, there is a section titled "Step 4: Define your data source".

```
import { gql } from "apollo-boost";
// or you can use `import gql from 'graphql-`
```

Apollo

Apollo Server



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

The screenshot shows a laptop screen with two main sections. On the left is the Apollo Docs sidebar, which includes links for Apollo Server, Introduction, Get started with Apollo Server, Changelog, EXPAND ALL, DEFINING A SCHEMA, FETCHING DATA, FEDERATION, TESTING, PERFORMANCE, SECURITY, INTEGRATIONS, DEPLOYMENT, MONITORING, API REFERENCE, and APPENDICES. The right section displays a web page from apollographql.com/docs/react/get-started/. The page has a search bar and a message: "it's executing a query. To fix this, we create a resolver." It explains that resolvers tell Apollo Server how to fetch data associated with a type. Below this, a code snippet for `index.js` is shown:

```
1 // Resolvers define the technique for fetching the types defined in the
2 // schema. This resolver retrieves books from the "books" array above.
3 const resolvers = {
4   Query: {
5     books: () => books,
6   },
7};
```

Below the code, a section titled "Step 6: Create an instance of `ApolloServer`" is shown. It says: "We've defined our schema, data set, and resolver. Now we just need to provide this information to the `ApolloServer` when we initialize it." Another code snippet for `index.js` is provided:

```
1 // The ApolloServer constructor requires two parameters: your schema
2 // definition and your set of resolvers.
3 const server = new ApolloServer({ typeDefs, resolvers });
4
5 // The `listen` method launches a web server.
6 server.listen().then(({ url }) => {
7   console.log(`🚀 Server ready at ${url}`);
8});
```

Apollo

Apollo Client (React)



- 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)

The screenshot shows a laptop displaying the Apollo GraphQL documentation website at apollographql.com/docs/react/get-started/. The page is titled 'Client (React)' and shows the 'Get started' section. The main content area contains a 'Create a client' tutorial. It includes a code snippet for initializing an ApolloClient:

```
1 import ApolloClient from 'apollo-boost';
2
3 const client = new ApolloClient({
4   uri: 'https://48p1r2roz4.sse.codesandbox.io',
5 });
```

The tutorial explains that this client is ready to fetch data and provides instructions to send a query using the client's query() method.

Apollo

Apollo Client (React)



Connect Apollo Client to React

- (ApolloProvider from apollo/react-hooks)
- Request data (with render-props or hooks!)
- Verify everything on your browser ;)

The screenshot shows a laptop displaying the Apollo GraphQL documentation website at apollographql.com/docs/react/get-started/. The page is titled 'Client (React)' and shows the 'v2.6' version. The left sidebar contains navigation links for 'Introduction', 'Why Apollo Client?', 'Get started', 'EXPAND ALL', 'FETCHING DATA', 'CACHING', 'DEVELOPMENT & TESTING', 'PERFORMANCE', 'INTEGRATIONS', 'NETWORKING', 'API REFERENCE', and 'MIGRATING'. The main content area features a heading 'Connect your client to React' with a sub-section about using ApolloProvider. Below this is a code snippet for 'index.js':

```
1 import React from 'react';
2 import { render } from 'react-dom';
3
4 import { ApolloProvider } from '@apollo/react-hooks';
5
6 const App = () => (
7   <ApolloProvider client={client}>
8     <div>
9       <h2>My first Apollo app 🚀</h2>
10      </div>
11    </ApolloProvider>
12  );
13
14 render(<App />, document.getElementById('root'))
```

At the bottom right, there is a section titled 'Request data' with the text: 'Once your ApolloProvider is hooked up, you're ready to ...'.



Thanks for coming!