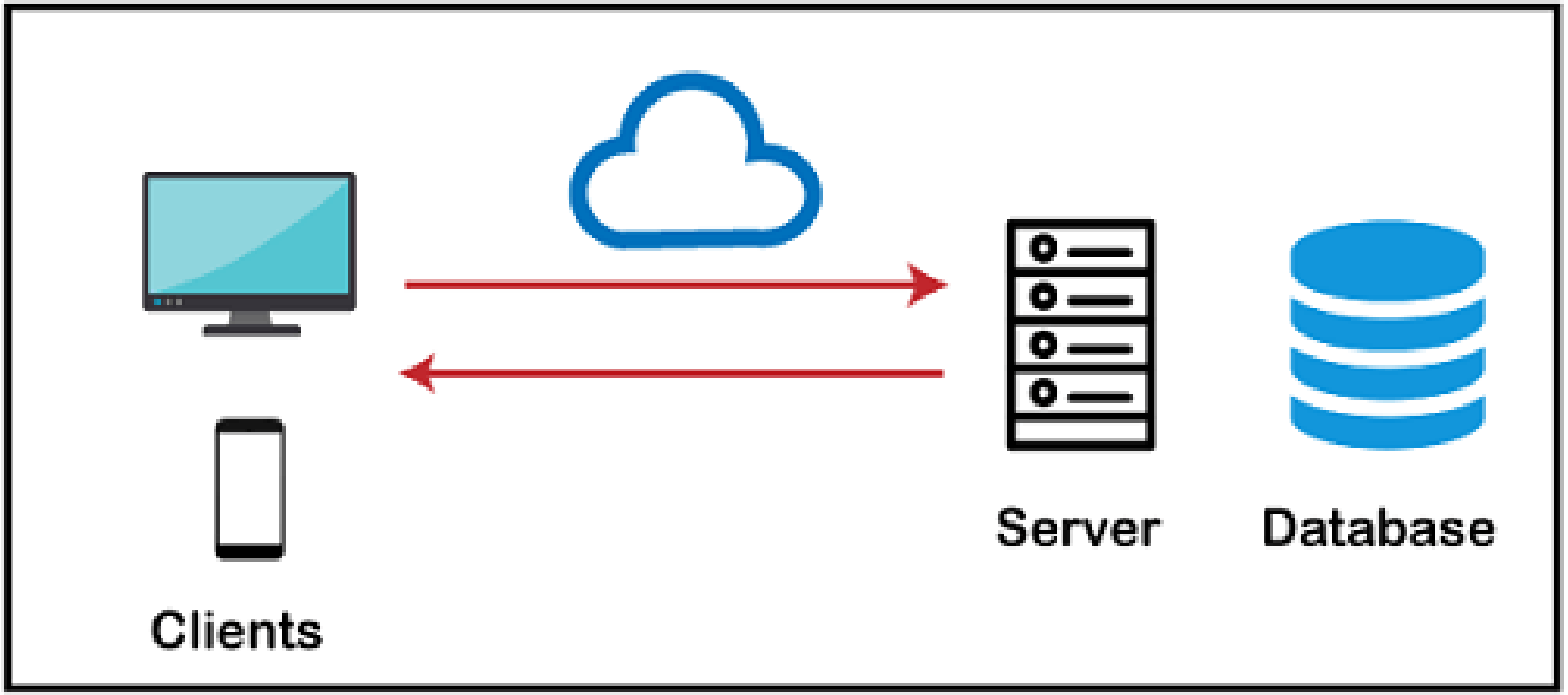
graphql

GraphQL

- GraphQL is a new **API standard invented and developed by Facebook**.
- It is **an open-source** server-side technology, now maintained by a large community of companies and individuals of all over the world.
- It is also an execution engine that works as a data query language and used to fetch declarative data.
- It was developed to optimize RESTful API calls and provides a flexible, robust, and more efficient alternative to REST.
- GraphQL is data query and manipulation language for your API and a server-side runtime for executing queries when you define a type system for your data.
- Unlike the REST APIs, a GraphQL server provides only a single endpoint and responds with the precise data that a client asked for.

A Query Language for APIs



Graphql advantages

- **1.GraphQL is more precise, accurate, and efficient.**

```
{
  employee {
    id
    firstName
  }
}
```

- **2.Retrieve many resources from a single request**
- 3.GraphQL queries are simple and human-readable.
- 4. GraphQL is best suited for microservices and complex systems because of using a simple query.
- 5. It facilitates you to deal with many databases efficiently.
- 6. Data can be fetched using a single API call.
- 7. You don't face over fetching and under fetching issues in GraphQL.
- 8. Using GraphQL, you can discover the schema in the appropriate format.
- 9. GraphQL provides rich and powerful developer tools for testing queries and documentation.

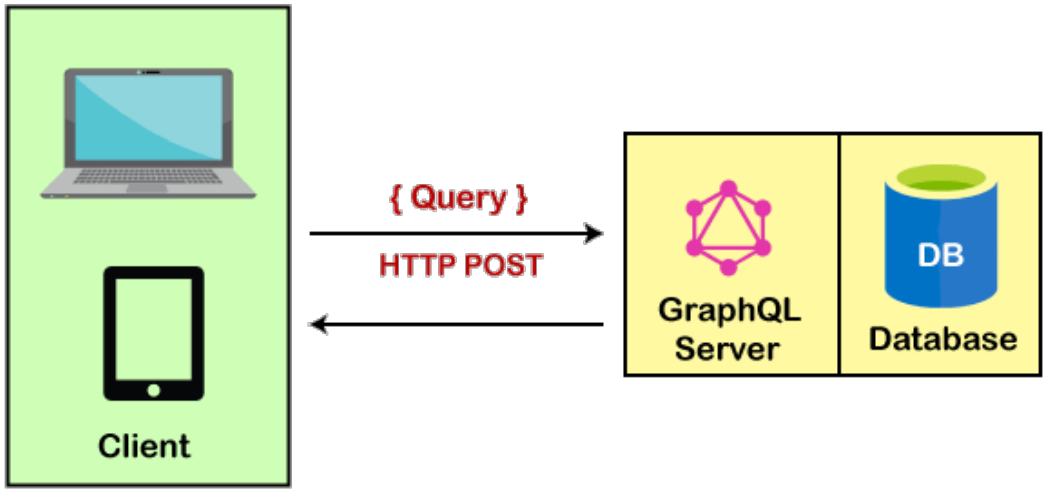
Graphql architecture

- especially in a web/mobile app.
- GraphQL was released as a specification.
- The specification describes the behavior of the GraphQL server. It provides some guidelines to handle requests from the clients and responses from the server, such as supported protocols, the format of the data that the server accepts, the format of the server's response, etc.
- The client makes a request to the GraphQL server. This request is called a query. When a client requests a query to communicate with the server, the transport layer of GraphQL can be connected with any available network protocol such as TCP, WebSocket, or any other transport layer protocol.
- The GraphQL server doesn't care about the database you use.
- It is neutral to databases. You can use a relational or a NoSQL database.

Client-Server flow in GraphQL

- The GraphQL query is not written in JSON. When a client makes a 'POST' request to send a GraphQL query to the server, this query is sent as a string.
- The server receives and extracts the query string. After that, the server processes and validates the GraphQL query according to the GraphQL syntax and the graph data model (GraphQL schema).
- Like the other API servers, the GraphQL API server also makes calls to a database or other services and retrieves the data requested by the client.
- After that, the server takes the data and returns it to the client in a JSON object.

GraphQL Server with a Connected Database

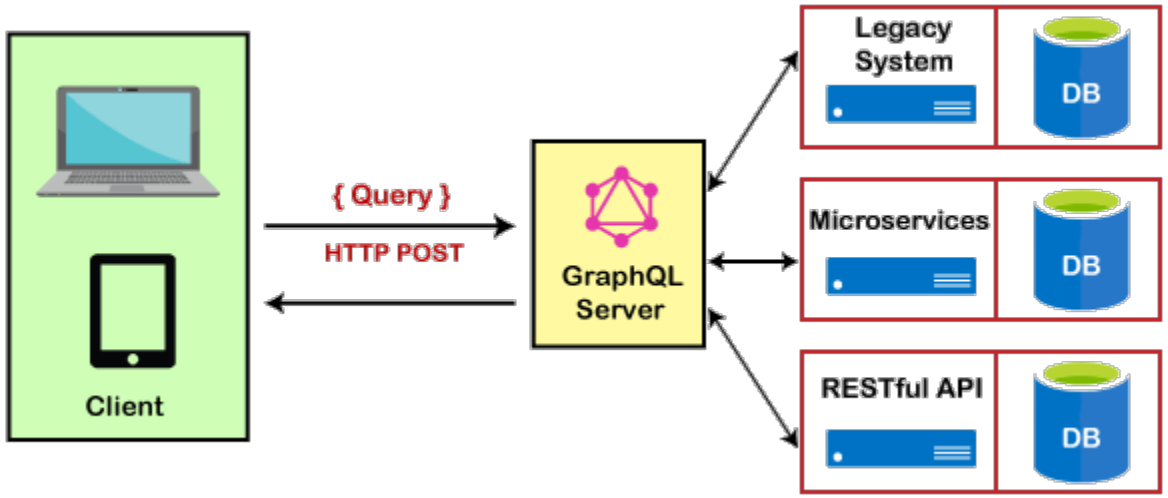


- **Key points of GraphQL Server with a Connected Database**
- This architecture setup is used for simple and new projects.
- It uses a single web server that implements GraphQL.
- This model uses resolving the query.
- In this model, the server resolves the queries and constructs responses with data that it fetches from the database.

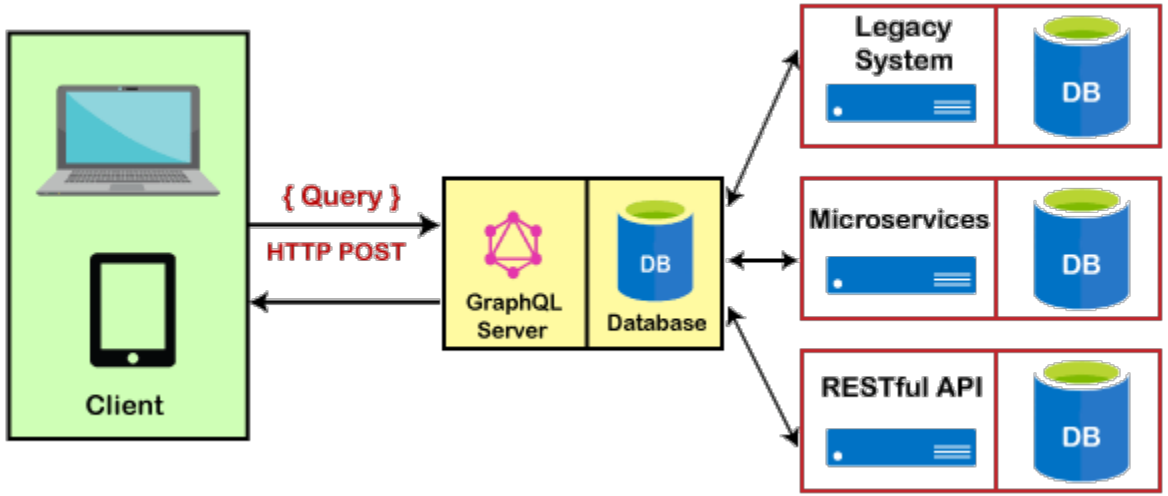
How to build a GraphQL server?

- To use GraphQL in our project, we have to build the GraphQL server. You can do it in any programming language of your choice.
- 3 common architectural models for a GraphQL server.
- GraphQL server with a connected database
- GraphQL server integrated with the existing system.
- A hybrid approach with a connected database and integration of the existing system

GraphQL Server integrated with the Existing System.



A Hybrid approach with a Connected Database and Integrated Systems



GraphQL First look

- **Node.js:** As we have explained in the previous chapter, Node.js is an open-source, cross-platform, server runtime environment that is used to execute JavaScript code outside of a web browser.
- **Express:** Express is a popular web application framework for Node.js. It is used to build websites and web applications along with Node.js. In this example, we shall set up Express with Node.js and build a GraphQL API on top of the Express framework.
- **Apollo Server:** Apollo Server is an open-source, spec-compliant GraphQL server which can be used easily with any GraphQL client, such as Apollo Client. Apollo Server is the most preferred server to build a production-ready, self-documenting GraphQL API that can able to retrieve data from any source.
- **GraphiQL:** GraphiQL is the integrated development environment (IDE) for GraphQL. It is one of the best and complete tools to make Gatsby websites (A website that takes static data sources like Markdown and turns them into a dynamic site using React JS, called Gatsby website).

GraphQL Application Components

- GraphQL components are very important to make communication happen. Let's see what the important GraphQL key components and the way they communicate with each other are.
- There are 2 types of GraphQL application components:
- GraphQL Server-side Components
- GraphQL Client-side Components
- GraphQL Server-side Components
- The GraphQL API uses three main components that reside on the GraphQL server. These components are **query, resolver, and schema**. Server-side components allow parsing the queries coming from GraphQL client applications.

query

A query is a client application request made by the GraphQL client to make communication with the GraphQL server. It is used to read or retrieve values. A query field can support arguments and points to arrays.

There are 2 important parts of a query: field and arguments.

Field: In a GraphQL query, a field specifies that we are asking for particular information from the server.

An example of a field in GraphQL query.

```
query {
   student {
      id firstName
   }
```

```
"data": {
   "student":[ {
      "id": 101,
      "firstName": "Albert"
   }
   ,
   …
]
}
}
```

Here in the above query example, we ask the server for the field called the student, and its subfields like id and firstName and the GraphQL server sends the asked data.

Argument: Every field on a GraphQL object type can have zero or more arguments.

Resolver and Schema

- Resolver Resolvers are used to provide directions for converting GraphQL operation into data. They provide instructions for turning a GraphQL operation into data. They define resolver functions to resolve the query to data.
- The resolvers also separate database schema and API schema, which makes it easy to modify the content obtained from the database.
- Schema The GraphQL schema is the center of any GraphQL server implementation. It describes its available functionality to the connected clients. The core building block within schemas is called a type.

GraphQL Servers

- GraphQL is simply a specification. We have to use some GraphQL server implementations to make communication.

- Types of GraphQL Servers Implementation
- GraphQL-JS: GraphQL-JS is used with Express. It is the original reference implementation of GraphQL.

- GraphQL-Server: GraphQL-Server is the all-in-one GraphQL server implementation of Apollo. It can be accessed from any GraphQL client.

- GraphQL-Serverless: GraphQL-Serverless is the Back4App instant GraphQL API fully integrated with MongoDB database and Cloud Functions.

- GraphQL Yoga: GraphQL Yoga is Prisma's server implementation built on Express and Appolo servers.

GraphQL Client-side Components

- The client-side components reside on GraphQL clients. The GraphQL client is a code or a JavaScript library that makes POST requests to the GraphQL server. It may be a CMS like Drupal, a single page application, a mobile application, etc.

- See the below client-side components:

- GraphiQL: It is a browser-based interface used for editing and testing GraphQL queries and mutations.

- ApolloClient: It is one of the best tools to build GraphQL client applications. It can be easily integrated with all JavaScript front-end. It saves network traffic by caching requests and normalizing data. It supports pagination, prefetching data, and connection between the data layer to the view layer.

- GraphQL Clients
- We can query our GraphQL API directly, but it is always a good practice to use a dedicated client library. It also makes things easier.

Graphql schema

- A GraphQL schema is at the core of any GraphQL server implementation.
- It describes the functionality available to the client applications that connect to it.
- We can use any programming language to create a GraphQL schema and build an interface around it.
- The GraphQL runtime defines a generic graph-based schema to publish the capabilities of the data service it represents.
- Client applications can query the schema within its capabilities. This approach decouples clients from servers and allows both to evolve and scale independently.
- The **makeExecutableSchema** function in graphql-tools helps you to bind schema and resolvers.

```
import { makeExecutableSchema } from 'graphql-tools';

const jsSchema = makeExecutableSchema({
  typeDefs,
  resolvers, // optional
  logger, // optional
  allowUndefinedInResolve = false, // optional
  resolverValidationOptions = {}, // optional
  directiveResolvers = null, // optional
  schemaDirectives = null,  // optional
  parseOptions = {},  // optional
  inheritResolversFromInterfaces = false  // optional
});
```

schema.graphql

```
type Query {
  greeting:String
  students:[Student]
}

type Student {
  id:ID!
  firstName:String
  lastName:String
  password:String
  collegeId:String
}
```

resolver.js

```
const db = require('./db')
const Query = {
  greeting:() => {
    return "hello from  Mithibites !!!"
  },
  students:() => db.students.list()
}

module.exports = {Query}
```

Resolver

- Resolver is a collection of functions that generate response for a GraphQL query.
- In simple terms, a resolver acts as a GraphQL query handler. Every resolver function in a GraphQL schema accepts four positional arguments as given below −

- fieldName:(root, args, context, info) => { result }

//resolver function with no parameters and returning string

- greeting:() => {
- 	return "hello from TutorialsPoint !!!"
- }

- //resolver function with no parameters and returning list
- students:() => db.students.list()

- //resolver function with arguments and returning object
- studentById:(root,args,context,info) => {
- 	return db.students.get(args.id);
- }

Schema.graphql

```
type Query {
  greeting:String
  students:[Student]
  studentById(id:ID!):Student
}

type Student {
  id:ID!
  firstName:String
  lastName:String
  password:String
  collegeId:String
}
```

Resolver.js

```
const db = require('./db')
const Query = {
  //resolver function for greeting
  greeting:() => {
    return "hello from  TutorialsPoint !!!"
  },

  //resolver function for students returns list
  students:() => db.students.list(),

  //resolver function for studentbyId
  studentById:(root,args,context,info) => {
    //args will contain parameter passed in query
    return db.students.get(args.id);
  }
}
module.exports = {Query}
```

Mutation

- Mutation queries modify data in the data store and returns a value. It can be used to insert, update, or delete data. Mutations are defined as a part of the schema.

- The syntax of a mutation query is given below –

```
mutation{
  someEditOperation(dataField:"valueOfField"):returnType
}
```