

GraphQL



what is graphql

Let's shape the answer in your **Better Questions Checklist** style so it's structured and useful:

Goal

You want to understand **what GraphQL is**, probably in comparison to REST or in the context of APIs.

Context

GraphQL is a **query language for APIs** and a **runtime for executing queries**. It was developed by Facebook in 2012 and open-sourced in 2015.

- REST APIs expose multiple endpoints (e.g., `/users`, `/users/{id}/orders`).
- GraphQL exposes a **single endpoint** (usually `/graphql`) where the client decides what data it needs.

Explanation (Core Concept)

- **GraphQL = Query Language + Runtime**
 - **Query Language** → lets clients specify **exactly what fields** they want (nothing more, nothing less).
 - **Runtime** → resolves those queries using the data you define in your schema.
- **Schema Driven**
 - You define types (**User**, **Product**, etc.) and their relationships.
 - Queries are validated against this schema.
- **Client-controlled queries**
 - Instead of multiple endpoints, clients ask for the data they need in one structured query.

Example

REST request

```
bash

GET /users/1
Response:
{
  "id": 1,
  "name": "Sumit",
  "email": "sumit@example.com",
```

```
"address": {
  "street": "123 Lane",
  "city": "Delhi"
}
```

GraphQL query

```
graphql

{
  user(id: 1) {
    name
    address {
      city
    }
  }
}
```

GraphQL response

```
json

{
  "data": {
    "user": {
      "name": "Sumit",
      "address": {
        "city": "Delhi"
      }
    }
  }
}
```

👉 Only the requested fields (**name**, **address.city**) are returned.

Pros & Cons

Pros

- No over-fetching or under-fetching (client controls fields).
- Single endpoint for all queries.
- Strongly typed schema (good for tooling, validation, IDE autocomplete).
- Easy to evolve APIs without breaking changes.

Cons

- More complex server setup than REST.
- Query performance issues if not optimized (clients can request too much).
- Caching is trickier than REST.

Short Answer

GraphQL is a **query language for APIs** that lets clients ask for exactly the data they need from a single endpoint, unlike REST where the server defines what's returned.

Do you want me to also explain **GraphQL vs REST in interview terms** (like "when would you choose one over the other")? That might be handy for your Selenium/TestNG + API testing background.



sure

Perfect 👍 let's do a **GraphQL vs REST comparison in interview style** — concise, clear, and with pros/cons.

♦ GraphQL vs REST — Interview Explanation

1. Endpoints

- **REST** → Multiple endpoints (`/users`, `/users/{id}/orders`).
- **GraphQL** → Single endpoint (`/graphql`), all queries go here.

2. Data Fetching

- **REST** → Server decides the response structure. You often **over-fetch** (extra unused fields) or **under-fetch** (need multiple calls).
- **GraphQL** → Client specifies exactly what it needs (no extra fields, no missing fields).

3. Schema & Type System

- **REST** → Usually not strictly typed, relies on documentation.
- **GraphQL** → Strongly typed schema (**User**, **Product**, **Order**) with introspection (self-documenting).

4. Versioning

- **REST** → Often needs versioning (`/v1/users`, `/v2/users`).
- **GraphQL** → Versioning usually not required; evolve schema by adding fields while keeping old ones.

5. Performance

- **REST** → May require multiple requests to get related data (e.g., user + orders).
- **GraphQL** → Fetch related data in a **single query**. But risk: complex queries can be expensive.

6. Caching

- **REST** → Easy with HTTP caching (**ETag**, **Cache-Control**).
- **GraphQL** → Trickier since all requests go to one endpoint. Needs custom caching (Apollo, Relay).

7. Error Handling

- **REST** → Uses HTTP status codes (**404**, **500**).
- **GraphQL** → Always returns **200 OK** (even with errors inside the **errors** field).

8. When to Use?

- **Choose REST when:**
 - Simple CRUD APIs.
 - Strong need for built-in HTTP caching.
 - Public APIs that should be lightweight and predictable.
- **Choose GraphQL when:**
 - Clients (mobile/web) need **different data shapes**.
 - Avoiding **over-fetching** / **under-fetching** is critical.
 - You want schema-driven APIs with strong tooling and flexibility.

♦ One-Liner (Interview Ready)

👉 **REST** is **resource-oriented** with multiple endpoints and fixed responses.

👉 **GraphQL** is **query-oriented**, using a single endpoint where the client controls the response shape.

Would you like me to also give you a **mini “API Testing angle”** → i.e., how testing changes in **GraphQL vs REST** (since you’re building RestAssured + TestNG frameworks)? That’ll make it practical for your work + interviews.



can rest assured support graphql

Great question 🙌 — let’s answer it in **Better Questions Checklist style** so it’s practical for you.

✅ Goal

You want to know if **RestAssured (Java API testing library)** can test **GraphQL APIs**.

✅ Context

- RestAssured is commonly used for **REST API testing**.
- GraphQL also runs over **HTTP**, usually with JSON payloads, just structured differently (**query**, **mutation**, **variables**).
- So the real question is: *can we send GraphQL queries as JSON bodies using RestAssured?*

✓ Short Answer

👉 **Yes, RestAssured can test GraphQL APIs.**

Because GraphQL queries are just **POST requests with JSON bodies**, and RestAssured can send those easily.

✓ How (Example)

GraphQL Query

```
graphql

{
  user(id: 1) {
    name
    email
  }
}
```

Request body (GraphQL requires it as JSON)

```
json

{
  "query": "{ user(id: 1) { name email } }"
}
```

RestAssured Test

```
java

import static io.restassured.RestAssured.*;

public class GraphQLTest {
    public static void main(String[] args) {
        String query = "{ user(id: 1) { name email } }";

        given()
            .header("Content-Type", "application/json")
            .body("{\"query\": \"" + query.replace("\"", "\\\"") + "\"}")
        .when()
            .post("http://localhost:8080/graphql")
        .then()
            .statusCode(200)
            .log().all();
    }
}
```

✓ Notes (Best Practices)

- Wrap the query as a string inside a JSON object (`{"query": "...", "variables": {...}}`).
- Escape quotes if you inline the query in Java (or load it from `.graphql/.txt` file for readability).
- Validate responses with **JSONPath** in RestAssured (`.body("data.user.name", equalTo("Sumit"))`).

✓ Pros / Cons

Pros

- Works out-of-the-box (no plugin needed).
- Can reuse your existing TestNG + RestAssured framework.

Cons

- GraphQL has nested responses → JSONPath assertions can get long.
- No native GraphQL query builder in RestAssured (unlike dedicated GraphQL clients like Apollo).

👉 So yes, you can fully test GraphQL APIs with RestAssured by treating them like **POST + JSON body**.

Do you want me to show you a **TestNG test case structure** for GraphQL (with `@Test`, assertions, and reporting) so it plugs into your existing framework?