



GraphQL Best Practices

The GraphQL specification is intentionally silent on a handful of important issues facing APIs such as dealing with the network, authorization, and pagination. This doesn't mean that there aren't solutions for these issues when using GraphQL, just that they're outside the description about what GraphQL *is* and instead just common practice.

The articles in this section should not be taken as gospel, and in some cases may rightfully be ignored in favor of some other approach. Some articles introduce some of the philosophy developed within Facebook around designing and deploying GraphQL services, while others are more tactical suggestions for solving common problems like serving over HTTP and performing authorization.

Following are brief descriptions of some of the more common best practices and opinionated stances held by GraphQL services, however each article in this section will go into more depth on these and other topics.

HTTP

GraphQL is typically served over HTTP via a single endpoint which expresses the full set of capabilities of the service. This is in contrast to REST APIs which expose a suite of URLs each of which expose a single resource. While GraphQL could be used alongside a suite of resource URLs, this can make it harder to use with tools like [GrapQL](#).

Read more about this in [Serving over HTTP](#).

LEARN

[Introduction](#)

[Queries and Mutations](#)

[Fields](#)[Arguments](#)[Aliases](#)[Fragments](#)[Operation Name](#)[Variables](#)[Directives](#)[Mutations](#)[Inline Fragments](#)

[Schemas and Types](#)

[Type System](#)[Type Language](#)[Object Types and Fields](#)[Arguments](#)[The Query and Mutation Types](#)[Scalar Types](#)[Enumeration Types](#)[Lists and Non-Null](#)[Interfaces](#)[Union Types](#)[Input Types](#)

[Validation](#)

[Execution](#)

[Introspection](#)

BEST PRACTICES

[Introduction](#)

[Thinking in Graphs](#)

[Serving over HTTP](#)

JSON (with GZIP)

[Authorization](#)

[Pagination](#)

[Global Object Identification](#)

[Caching](#)

GraphQL services typically respond using JSON, however the GraphQL spec **does not require it**. JSON may seem like an odd choice for an API layer promising better network performance, however because it is mostly text, it compresses exceptionally well with GZIP.

It's encouraged that any production GraphQL services enable GZIP and encourage their clients to send the header:

```
Accept-Encoding: gzip
```

JSON is also very familiar to client and API developers, and is easy to read and debug. In fact, the GraphQL syntax is partly inspired by the JSON syntax.

Versioning

While there's nothing that prevents a GraphQL service from being versioned just like any other REST API, GraphQL takes a strong opinion on avoiding versioning by providing the tools for the continuous evolution of a GraphQL schema.

Why do most APIs version? When there's limited control over the data that's returned from an API endpoint, *any change* can be considered a breaking change, and breaking changes require a new version. If adding new features to an API requires a new version, then a tradeoff emerges between releasing often and having many incremental versions versus the understandability and maintainability of the API.

In contrast, GraphQL only returns the data that's explicitly requested, so new capabilities can be added via new types and new fields on those types without creating a breaking change. This has led to a common practice of always avoiding breaking changes and serving a versionless API.

Nullability

Most type systems which recognise "null" provide both the common type and the *nullable* version of that type, whereby default types do not include "null" unless explicitly declared. However, in a GraphQL type system, every field is *nullable* by default. This is because there are many things that can go awry in a networked service backed by databases and other services. A database could go down, an asynchronous action could fail, an exception could be thrown. Beyond simply system failures, authorization can often be granular, where individual fields within a request can have different authorization rules.

By defaulting every field to *nullable*, any of these reasons may result in just that field returned "null" rather than having a complete failure for the request. Instead, GraphQL provides **non-null** variants of types which make a guarantee to clients that if requested, the field will never return "null". Instead, if an error occurs, the previous parent field will be "null" instead.

When designing a GraphQL schema, it's important to keep in mind all the problems that could go wrong and if "null" is an appropriate value for a failed field. Typically it is, but occasionally, it's not. In those cases, use non-null types to make that guarantee.

Pagination

The GraphQL type system allows for some fields to return **lists of values**, but leaves the pagination of longer lists of values up to the API designer. There are a wide range of possible API designs for pagination, each of which has pros and cons.

Typically fields that could return long lists accept arguments "first" and "after" to allow for specifying a specific region of a

list, where "after" is a unique identifier of each of the values in the list.

Ultimately designing APIs with feature-rich pagination led to a best practice pattern called "Connections". Some client tools for GraphQL, such as [Relay](#), know about the Connections pattern and can automatically provide automatic support for client-side pagination when a GraphQL API employs this pattern.

Read more about this in the article on [Pagination](#).

Server-side Batching & Caching

GraphQL is designed in a way that allows you to write clean code on the server, where every field on every type has a focused single-purpose function for resolving that value. However without additional consideration, a naive GraphQL service could be very "chatty" or repeatedly load data from your databases.

This is commonly solved by a batching technique, where multiple requests for data from a backend are collected over a short period of time and then dispatched in a single request to an underlying database or microservice by using a tool like Facebook's [DataLoader](#).

[Continue Reading →](#)


Thinking in Graphs






Learn

[Introduction to GraphQL](#)
[Best Practices](#)
[Frequently Asked Questions](#)
[Training Courses](#)


Code

 [GitHub](#)
[GraphQL Specification](#)
[Libraries & Tools](#)
[Services & Vendors](#)

Community

 [@graphql](#)
 [Discord](#)
 [Stack Overflow](#)
[Resources](#)
[Events](#)
[Landscape](#)

& More

[News Blog](#)
[GraphQL Foundation](#)
[Logo and Brand Guidelines](#)
[Code of Conduct](#)
[Contact Us](#)
 [Edit this page](#)

Copyright © 2021 The GraphQL Foundation. All rights reserved. The Linux Foundation has registered trademarks and uses trademarks. For a list of trademarks of The Linux Foundation, please see our [Trademark Usage](#) page. Linux is a registered trademark of Linus Torvalds. [Privacy Policy](#) and [Terms of Use](#).