Nylas

GraphQL vs. REST

IS GRAPHQL THE FUTURE?

HOW CAN YOU IMPLEMENT IT NOW?

OVERVIEW

Developer tools are everywhere, and this makes it difficult to decide where to invest your time; GraphQL is yet another tool on this long list of other potential technologies to explore. Considering that most modern APIs are firmly based upon REST, it might seem reasonable to put it off for later. However, Graph APIs have advantages for many use cases that developers frequently face, such as complex web UIs, and apps that require efficient use of bandwidth such as mobile apps. So, it might be time for you to consider upgrading yourself.

In this article, we'll explore REST APIs and explain how they became the most popular approach, why Graph APIs are rising now, and some important factors to consider if you adopt GraphQL for your APIs.

IN THIS GUIDE YOU'LL LEARN:

1. A Brief History of APIs 3

2. Why GraphQL Makes Sense for APIs 4

3. Who Uses REST APIs? 6

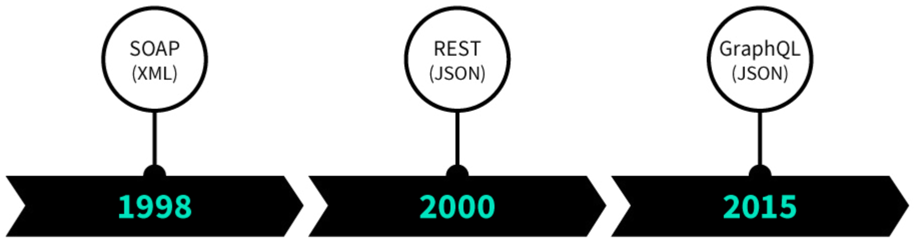
4. Downsides to GraphQL APIs 7

5. Graphing the Future 8

A Brief History of APIs

When developers think of APIs, they likely imagine the modern RESTful interface: make a simple HTTP call and get back some JSON to parse as needed. However, that hasn't always been the most popular convention, and in the future we'll likely see approaches change again.

The first modern APIs appeared in the early 2000s from big companies like Amazon, eBay, and Salesforce. Also known as web services, these APIs primarily relied upon SOAP, an XML-based standard. These APIs were based upon remote procedure calls (a precursor to SOAP was known as XML-RPC), which used developer tooling to expose them as functions within your code. This made SOAP APIs very easy to consume, as long as your tools supported them.



However, SOAP came with some downsides. Not every language had complete support, especially newer languages that were gaining in popularity like Ruby. In addition to being complex to call, the actual requests and responses were heavy with standards-compliant XML. Developers were attracted to the REST convention because it enabled them to move much more quickly by simplifying the way data is provided.

While not a standard, REST is an architectural style first described in Roy Fielding's doctoral dissertation. Typical implementations call resources (URL endpoints) over HTTP, using standard methods (GET, POST, PUT, DELETE.) to communicate actions. Early REST APIs used XML, but JSON has since taken over due to its lighter weight and easy-to-parse format.

Nylas provides REST APIs for email, calendar, and contacts, but we're increasingly looking to GraphQL as a better solution for abstracting away integration complexity.

LEARN MORE »

Though it has seen wide adoption, REST still has its downsides. It was created before the proliferation of smartphones and advanced mobile applications. Even though REST allows developers to work quickly, its responses are still aften heavier than developers want. The desire for a newer, even easier approach to APIs led Facebook to create GraphQL in 2012. The company made the project public in 2015 and others have quickly adopted it as the future for many APIs.

Why GraphQL Makes Sense for APIs

GraphQL is often represented as a REST killer, but in reality, many companies run both interfaces because there are specific situations where each shines. While REST is likely to continue being supported by popular APIs, GraphQL often makes a lot of sense too.

Rather than REST's pre-determined endpoints, GraphQL provides a query language that allows consumers to request specific data from a single URL. For example, A typical REST API might return a list of users that shows all of the data the API has added to their endpoint schema. But, if you want to filter by user attributes or only retrieve select user fields, these features need to be explicitly supported by the REST API.

A TYPICAL REST API REQUEST

```
$ curl -X GET 'https://my.api.com/person/my _ friend'
{
  "name": "My Friend",
  "birthday": "1971-01-01",
  "favorite _ food": "Pizza!",
  "id": "my _ friend",
  "hair _ color": "Black",
  "eye _ color": "Blue",
  "car": "1981 DeLorean,
}
```

Nylas provides REST APIs for email, calendar, and contacts, but we're increasingly looking to GraphQL as a better solution for abstracting away integration complexity.

LEARN MORE »

By contrast, GraphQL has filtering and field selection built into the methods for making API calls. You can request only the data you want and it won’t return anything else.

A TYPICAL GRAPHQL API REQUEST

```
$ curl -X GET 'https://my.api.com' --data '
{
    person(id: "my _ friend") {
        birthday
        favorite _ food
    }
}'
{
  "data": {
    "person": {
      "birthday": "1971-01-01",
      "favorite _ food": "Pizza!"
        }
    }
}
```

REST’s penchant for over-fetching data was a major reason that Facebook invented another solution. Its mobile app and user timeline view required frequent API calls with small and changing data needs, and this could cause issues with app responsiveness if they were relying on a REST API to fetch the data.

Mobile applications and other devices remain excellent use cases for GraphQL because any unused data returned from an API is essentially wasted bandwidth. Since these devices are on networks that can’t always be relied upon for high speed connections, this waste can contribute to a negative user experience. Similarly, many websites today also have more interactive experiences than in the early days of REST. These often rely on internal APIs that are tasked with powering complex user interfaces which can also make excellent use of GraphQL.

Who Uses REST APIs?

Many companies have already adopted GraphQL, with Github being one of the most notable. They have chosen to support only GraphQL with version 4 of their API, and they chose GraphQL because it provides a lot more flexibility for apps that integrate with their API:

GitHub	*GitHub chose GraphQL for our API v4 because it offers significantly more flexibility for our integrators. The ability to define precisely the data you want—and only the data you want—is a powerful advantage over the REST API v3 endpoints. GraphQL lets you replace multiple REST requests with a single call to fetch the data you specify.*

Shopify released a GraphQL API in 2018 to make it easier to build and manage online storefronts, they like it because it helps reduce requests that are made to their API servers:

shopify	*GraphQL gave us big wins by reducing the number of round trips made to the server and giving us a well defined and strongly typed schema to work against. GraphQL has also played a major role in helping us find and fix bugs before impacting our merchant's daily operations.*

Yelp released their first GraphQL API in 2017 and like it for how easy it is to access relational data:

yelp	*GraphQL also makes traversing graphs (and therefore relational data) very easy. Unlike most REST APIs, you don't need to make multiple requests to pull relational data. Based on the schema, you can retrieve data based on the relations they have. For our API users, this means easy access to all the great business information we have available.*

These are just a handful of the major companies that have added GraphQL APIs in recent years. With that said, GraphQL may not be for everybody; the next section of this article will look at some of the downsides to using GraphQL so you can determine if there might be any blockers to deploying it.

Downsides to GraphQL APIs

As we've seen with SOAP and REST before it, there are always trade-offs with any API interface. For certain situations, these may become big enough downsides to seek an alternative. In this section, we'll cover a few scenarios where moving to a GraphQL library might be difficult.

For some, one of these items could be a deal breaker. In other situations, there may be perfectly reasonable ways to eliminate or minimize the downsides. As with most software development, a sensible amount of architectural planning could save you considerable maintenance down the road.

CACHING IS MORE COMPLEX

REST APIs benefit from caching that is provided as part of normal HTTP server functionality. In other words, you can generally rely on modern HTTP servers and clients to properly cache GET requests that are made to a REST API. GraphQL uses a single URL for all requests, which often causes caching to not perform as expected.

Fortunately, there are open source tools that help you resolve this. On the server side, there is DataLoader: a NodeJS utility that can be used for batching and caching. On the client side, you can use Apollo Client: a JavaScript library that includes caching for making requests to GraphQL APIs. Either way, you need to make sure that you've fully evaluated whether your app requires caching and ensure you plan accordingly.

QUERIES REQUIRE OPTIMIZATION

GraphQL relies heavily on relationships between data objects, and as with any other queries on relational databases, it's possible to write inefficient GraphQL queries. As your requests dive further into a data graph, the number of API requests you make can rise very quickly. If you're producing a GraphQL API, you need to make sure you properly use batching where possible to group actions that access the database. You can also implement depth limits that prevent requests from going too far into a data graph. As an API consumer, it's important to consider the complexity of the data requests you make and determine if there are ways to optimize them.

CREATES POTENTIAL TO EXPOSE DATA MODEL

One powerful feature of GraphQL is introspection, which lets you ask a GraphQL schema what queries it supports. This makes development of GraphQL APIs much easier, but can also open you up to exposing your internal data models if this feature is enabled in your production servers. Fortunately, there are ways to restrict what schema are available to users, or even disable the feature entirely. You will need to ensure your API isn't exposing sensitive data with GraphQL's introspection function.

Graphing the Future

Despite some of the downsides to GraphQL APIs, the benefits are considerable for certain use cases. When developers need an efficient way to fetch the right data for things like mobile apps and complex user interfaces, or if they need methods to build applications that allow them to filter and select fields, GraphQL is unrivaled.

To begin exploring GraphQL you'll first want to understand it as an API consumer. Take a look at the official GraphQL introduction to get started. If you're responsible for building and maintaining a REST API, consider setting aside some time to investigate how a GraphQL version of your API might look. As with most development changes, these don't happen overnight. However, you'll want to be armed with informed opinions when the inevitable discussion arises.

SOAP once held the default position in the earliest days of modern APIs. XML was once the future, until JSON became the obvious choice. And while most APIs are currently based on REST, the over-fetching problem is enough of an issue to make a growing number of developers turn to GraphQL for their use case. It’s a good time to learn about GraphQL and look for ways that it can help you build API solutions.

ABOUT NYLAS

The Nylas Communications Platform enables developers to build email, calendar, and contacts integrations that work with 100% of providers via a modern REST API, but we’re increasingly looking to GraphQL as a better solution for abstracting away integration complexity. *Learn More »*

Nylas

Nylas.com

Github.com/Nylas

@Nylas