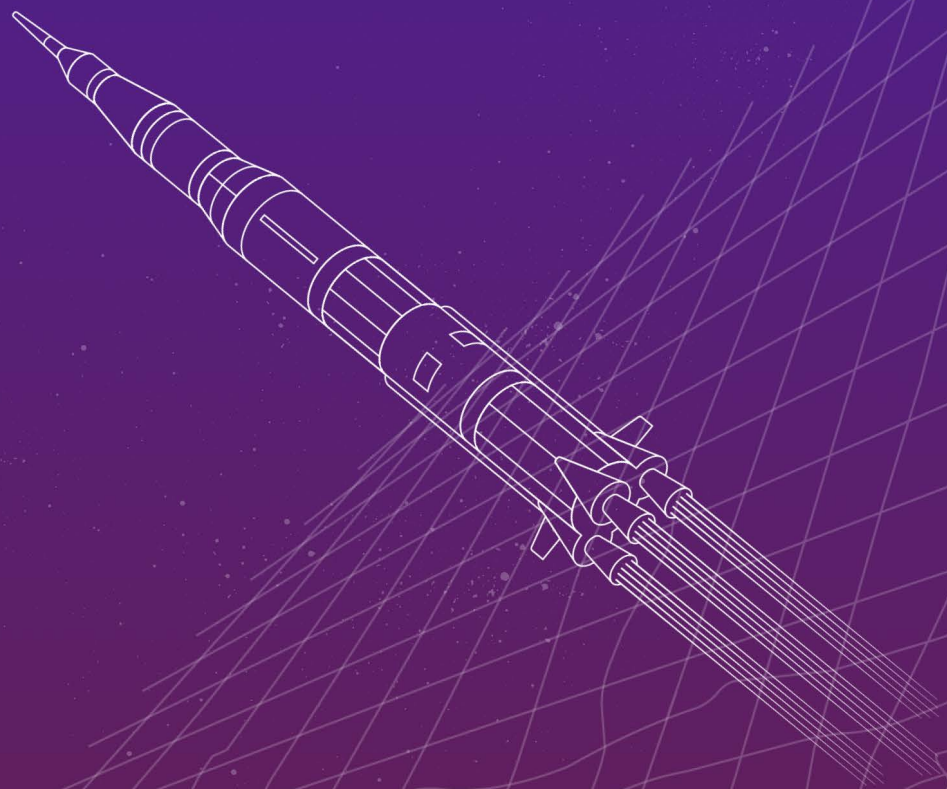
APOLLO

Modernizing Digital Payments with Data Graph at PayPal



PayPal | Case Study

PayPal Key Stats

>300M Users of PayPal's platform

>50 Products powered by a data graph

200 Markets worldwide

PayPal has remained at the forefront of the digital payment revolution for more than 20 years, now with over 300 million consumers and merchants using their payments platform. At the heart of this leadership position is PayPal Engineering, which is constantly looking for ways to iterate faster and make the user experience more responsive, safe, and convenient.

In a series of blog posts, PayPal Engineering gives a peek into their latest modernization effort to accelerate development using a data graph for GraphQL— from designing and validating the graph, to managing its growth.

The transition worked so well that PayPal now has over 50 products powered by a data graph.

PayPal

Paypal's open digital payments platform gives PayPal's account holders the confidence to connect and transact in new and powerful ways, whether they are online, on a mobile device, in an app, or in person.

CHALLENGE

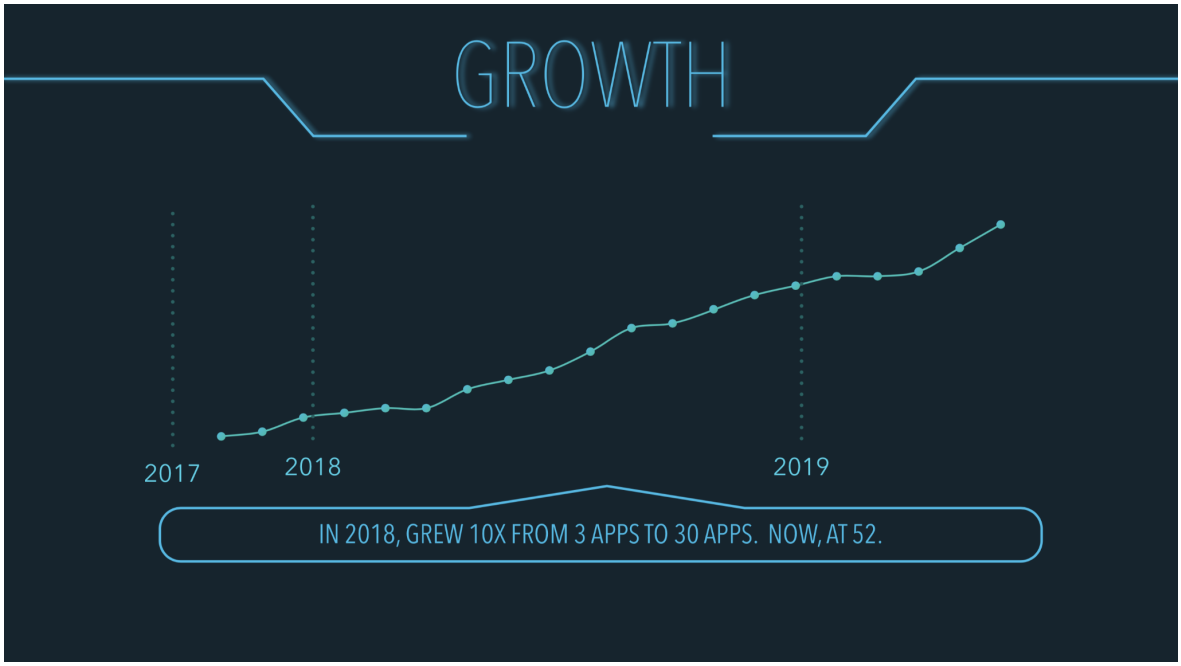
Paypal needed to innovate faster on its payment platform and improve the overall user experience. Legacy REST infrastructure was inadequate to meet their goals.

SOLUTION

A data graph built and managed with the Apollo platform maps PayPal's data and services together in one place, for any developer to draw upon.

RESULTS

- Over 50 PayPal products now powered by a data graph
- Adopted GraphQL in a fraction of the time using Apollo's platform.
- A central team uses Apollo to ensure PayPal's data graph is protected, monitored and managed.

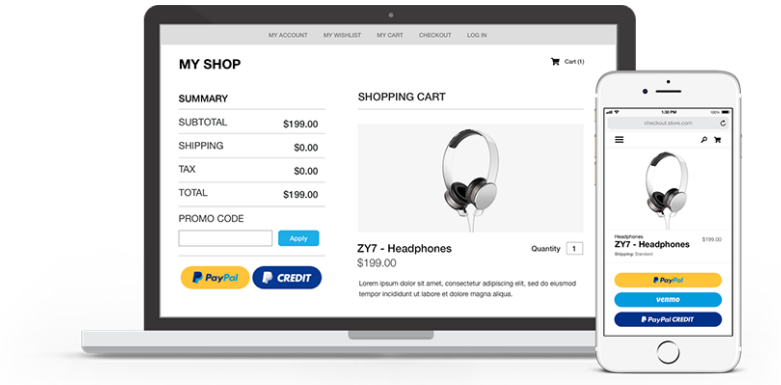


UNLEASHING PRODUCTIVITY GAINS

Mark Stuart, Sr. Manager of Web Platform at PayPal explained how shifting to GraphQL greatly improved developer productivity, flexibility and end-user performance.

With PayPal's Checkout product, they initially went all in on REST but had many developer experience and end-user performance issues. "We found that UI developers were spending less than one-third of their time actually building UI. The remainder of that time was spent figuring out where and how to fetch data, filtering/mapping over that data, and orchestrating API calls," Stuart says.

PAYPAL CHECKOUT



Stuart explains that “every round trip to fetch data costs at least 700ms in network time, not counting the time processing the request on the server. Multiple round trips result in slower performance, more user frustration, and a large impact to our bottom line.” The PayPal Checkout team tried GraphQL with a new product and immediately saw developer productivity and performance gains. They decided to go all in on GraphQL.

FROM BFF TO GRAPHQL

Initially, a Backend For Frontend (BFF) architecture was put in place to allow UI developers to iterate quickly with their own custom APIs. Stuart explains what has become a fairly common scenario, “Separate back-end teams developed the core services and then product teams built BFF APIs that orchestrated data from those underlying core services.”

Even though the BFF APIs were specialized and allowed developers to iterate faster, PayPal found them requiring a lot of low-value boilerplate.

According to Stuart, “With BFFs, we have a lot of teams iterating and building the same kinds of things over-and-over. Often, they contain a lot of orchestration logic where you need to grab data from 5, 10, 15 different services, normalize those responses, throw out 95% of it, and map, filter, and sort that data into what you want. It’s monotonous and not a good use of our developer’s time. Rather, UI developers should spend the majority of their time building UI.”

PayPal was looking for a way to get out of the practice of constantly writing and rewriting low-value orchestration code. As GraphQL gained popularity in the industry, they gave it a try and haven’t looked back.

IMPLEMENTING A DATA GRAPH

A data graph offered an elegant extension to their current infrastructure - mapping PayPal’s data and services together in one place, for any developer to draw upon using GraphQL. Building a data graph allowed clients to specify whatever data they need, in whatever combination desired, to support the user experience. And, multiple clients could do it, all backed by the same single graph— they weren’t writing new code for each client.

A data graph was also incredibly straightforward to implement. PayPal built the data graph at the edge of their stack that leverages existing core REST APIs. There was nothing to rip-and-replace, and they could migrate away from legacy infrastructure over time.

Ultimately, a data graph was an ideal solution for bringing new product experiences to market in a reasonable time without diminishing the customer's experience.

As Stuart explains, “Developers can now think in terms of fields — not endpoints, domains, or complex joins. For example, they can traverse the data graph to pick out the user’s first name, a 60x60 profile photo, primary shipping address, and credit cards without having to invoke 6–10 different services.”

> *“A data graph is so much more streamlined and straightforward. It’s just far easier and faster to use.”*
>
> Mark Stuart., Sr. Manager of Web Platform at PayPal

MOVING PAYPAL CHECKOUT (AND MORE) TO THE DATA GRAPH

Validating the concept was easier than expected. They were spinning up a new product, offering a Mobile SDK for developers to integrate PayPal Checkout into their app. The team built it using a data graph in just six weeks with three developers.

This experience was a proof point for the data graph and GraphQL: “We realized we were on to something,” Stuart says. “Our developers were productive, our app was quick, and users were happy.”

The beauty of this approach is it allows PayPal to make the transition to GraphQL in an agile way — adding services over time, maintaining independent service ownership, and allowing product teams to use the data graph whenever they’re ready. It worked out better than expected. Only a year ago, PayPal validated the approach on a few products. Now, their data graph supports more than 50 products across the business.

MANAGING THE DATA GRAPH

As PayPal expanded its use of the data graph — not only supporting more products, but also integrating with internal systems and processes such as authentication and authorization — the team quickly realized that they needed a team and proper tooling to manage the data graph.

Stuart's team became the internal champions of the data graph, evangelizing it internally and rolling out infrastructure that helped teams collaborate on the graph and keep it protected.

The team recognized that building everything themselves would be costly and inhibit the growth of the platform across the organization.

Instead, PayPal began using the Apollo Data Graph Platform, which provided a complete system to manage their data graph. Apollo's Graph Manager, in particular, provided core services that they wouldn't have to build on their own.

Stuart explained, "We could have built these capabilities ourselves, but they wouldn't have been as polished. It would have taken a medium sized team a year or more to build something comparable. In a build vs. buy decision, Apollo Platform was a clear buy. We wanted to deliver the promise of GraphQL to our developers now rather than later."

> *"Graph Manager gives us deep insights at a field-level, confidence against breaking changes, whitelisted queries, and eases client integration by providing linting and in-line SLA timings for every field."*
>
> Mark Stuart., Sr. Manager of Web Platform at PayPal

PayPal Engineering isn't looking back. "GraphQL is taking PayPal by storm," says Stuart.

You can read the full writeup on PayPal's experience and specific recommendations in Stuart's latest blog post.