

GraphQL Makes Your APIs **Better**

The Revolution in Efficiency and Real-time Performance

I'm **Obsessed** with GraphQL



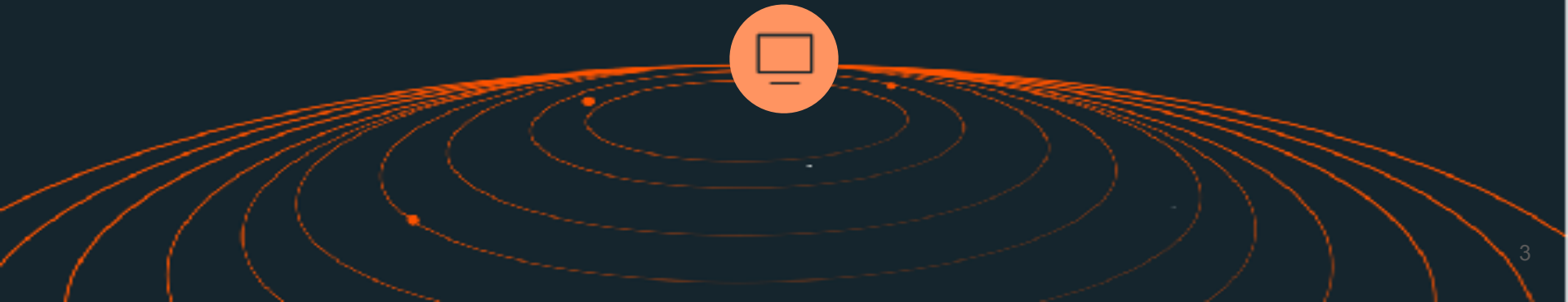
Michael Watson

Developer Relations at Apollo

<https://discord.gg/graphos>

APOLLO

GraphQL Makes Your APIs **Better**





Home

Search

Your Library

Developing with Apollo

This showcase demonstrates some of the capabilities and best practices of developing with Apollo. Log in and we'll show you how developing with Apollo is an amazing experience.

LOG IN

LOG IN

Welcome to the Apollo Showcase demo

This demo app provides a playground to test and learn about various Apollo features to understand how Apollo can be used in a moderately complex app.

Getting started

To use this app, you will need a Spotify account. This will allow the app to make calls to the Spotify API using this app's GraphQL API.

Running this app locally

If you would like to hook this up to your own application, you will first need a Spotify developer token. Once obtained, you will need to add these credentials to the app.

Create a Spotify application

First, visit the [Spotify developer portal](#) and, if necessary, log in. This requires a Spotify user account.

Create and register a new application to generate credentials. Follow the [App settings guide](#) to learn more. We recommend this Spotify app is unique to this demo app.

In the "Edit Settings" dialog, add this app's redirect URI for this app to allow this app to sign in to your Spotify account.

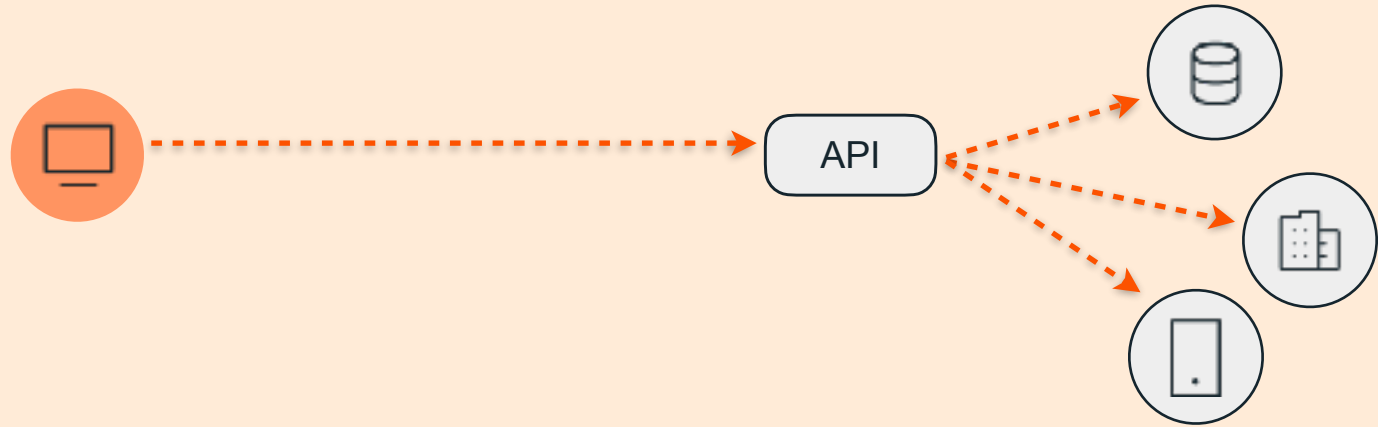
```
http://localhost:3000/oauth/finalize
```

Configure this application

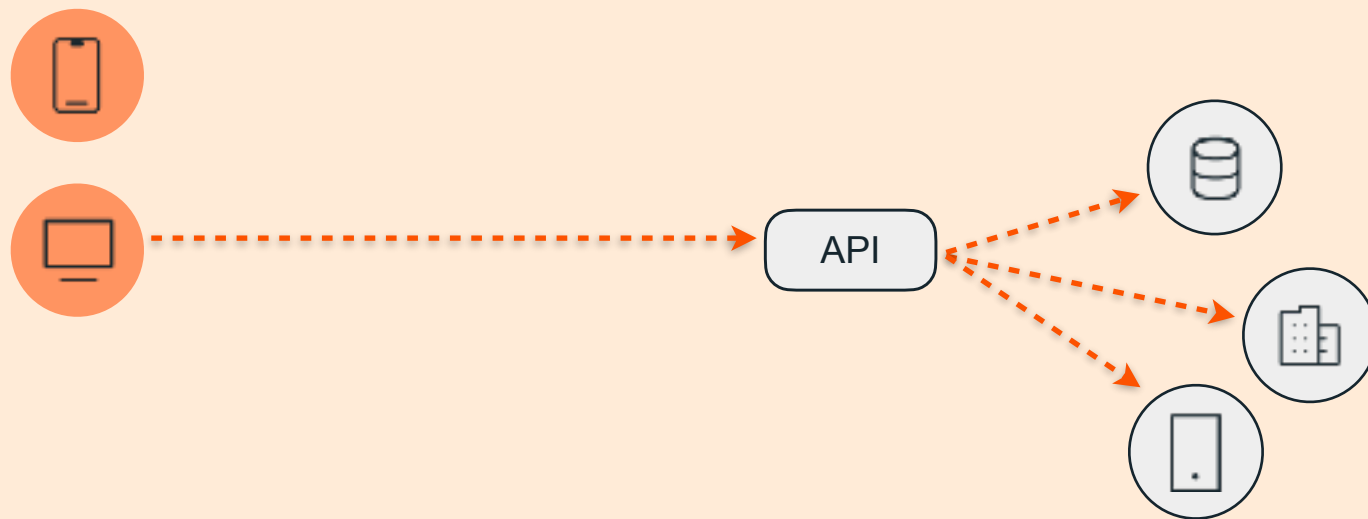
Copy your app's "Client ID" and set it as the `VITE_SPOTIFY_CLIENT_ID` environment variable in `client/.env.development.local` file. This file should look as follows:

```
VITE_SPOTIFY_CLIENT_ID=your-spotify-client-id
```

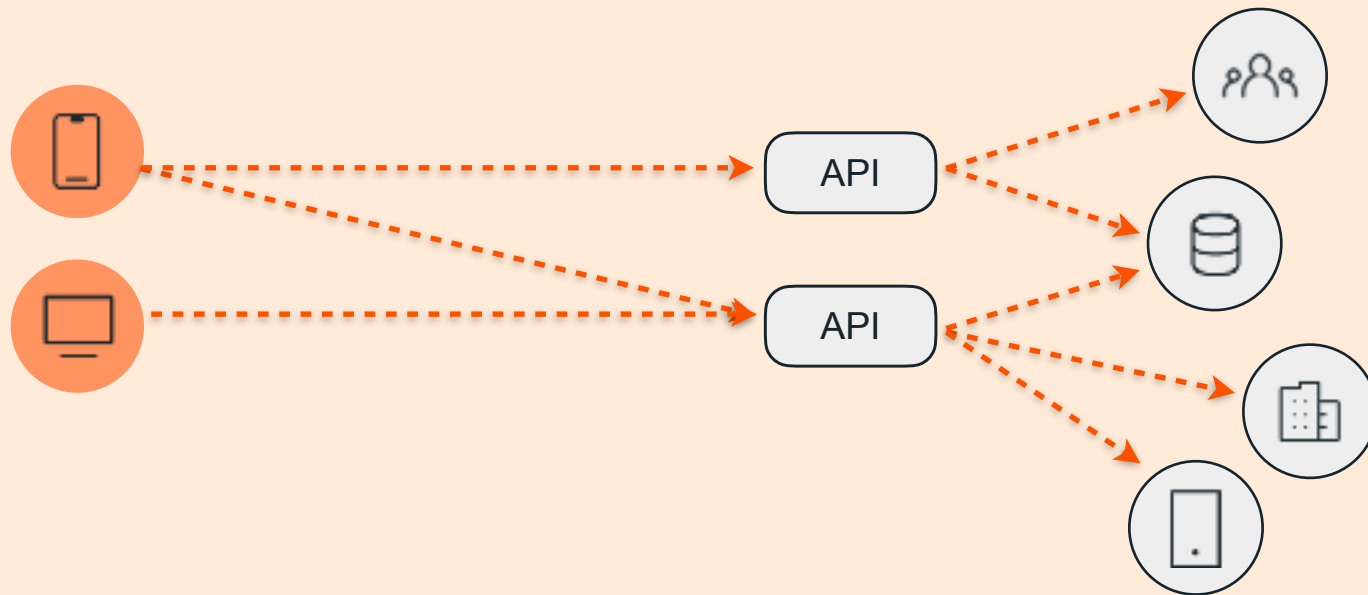
The Initial Dream 🙌



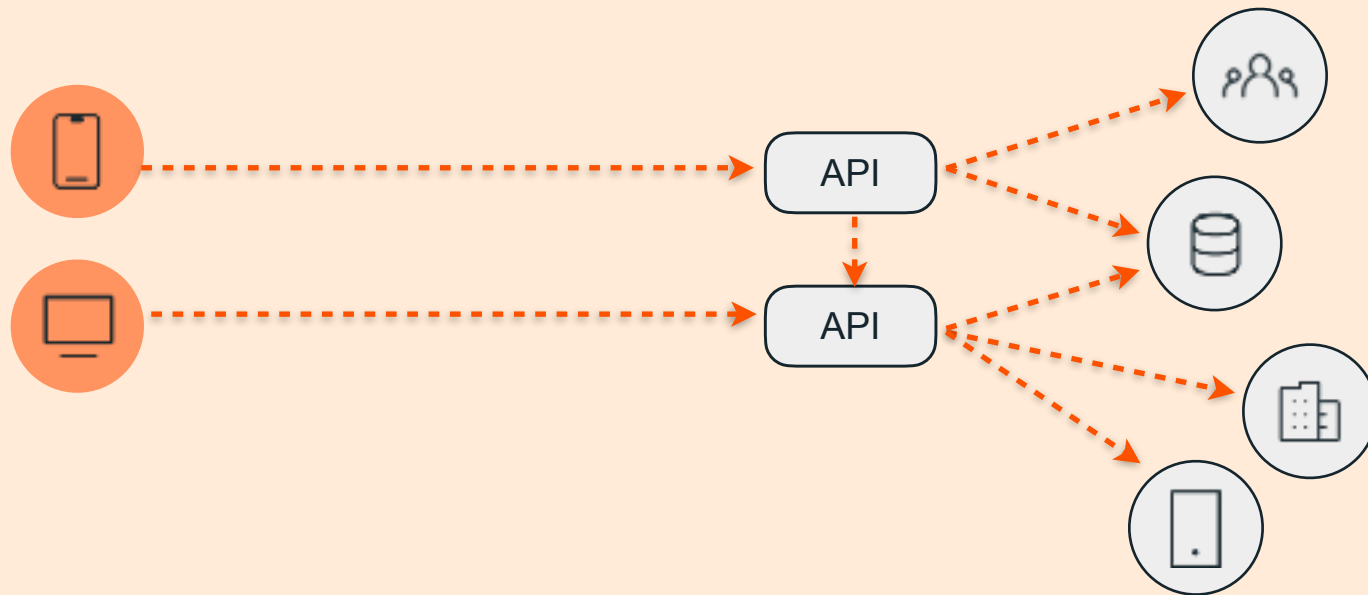
Then came mobile 🚀



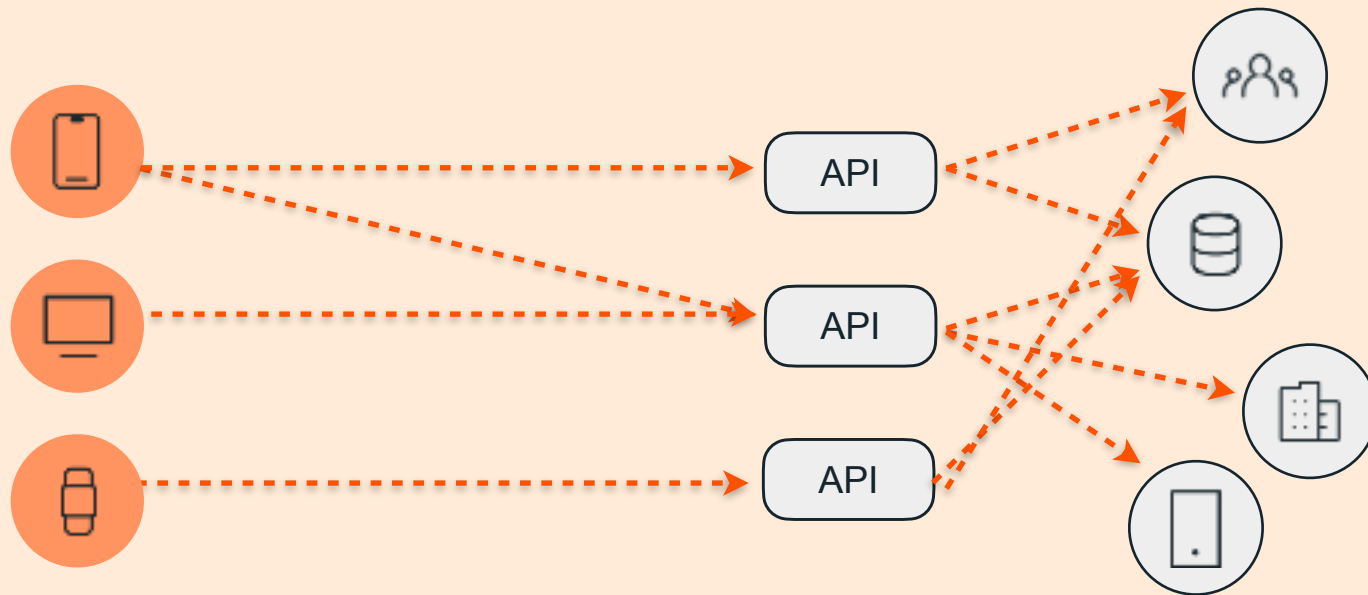
But the mobile team needed something different...



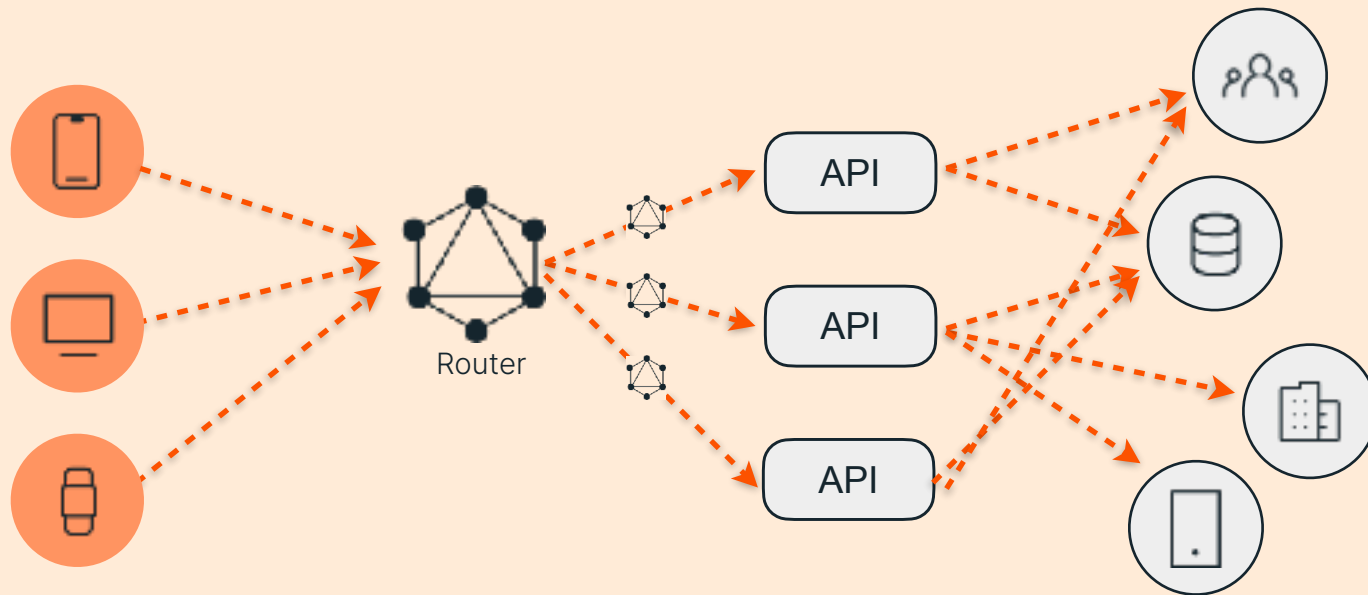
But the mobile team needed something different...



What about vNext?



Add GraphQL to existing APIs



The Router



- Observability
- GraphQL for all
 - anyone can write a query
- Real-time
- More efficient query execution
 - Caching, query plans and @defer
- Security built in



Observability

Spotify/poc -- FEDERATED V1.4

Operations **Fields** <

Search for a field name or parent type

Used 125 Total Requests

- image.url 7.2k
- Artist.name 7.1k
- Artist.id 7.1k
- Album.images 7.1k
- Album.id 7.1k
- Track.album 7.1k
- Track.artists 7.1k
- Album.name 7.1k**
- Track.name 7.1k
- Device.type 7.1k
- Device.name 7.1k

Album.name

Last day ~ Reset

The name of the album. In case of an album takedown, the value may be an empty string.

DATE FIRST SEEN TRACK: Over 3 months ago DATE LAST SEEN TRACK: an hour ago TAGS: No tags found. TAGS: specify

Subscribers: specify

Field Definition

As defined in the API schema. [View SDL](#)

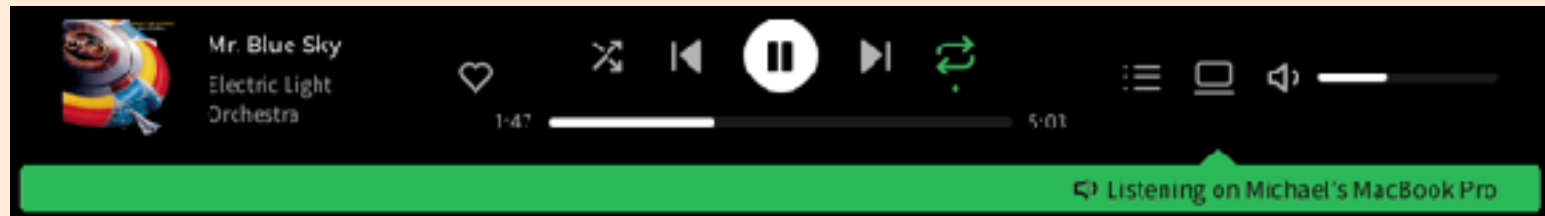
```
object Album {
  name: String!
}
```

Clients & Operations

Used by 2 clients and 15 operations in the last day.

Specify Showcase Website 6 operations	6.4k requests	>
Unidentified clients 9 operations	764 requests	>

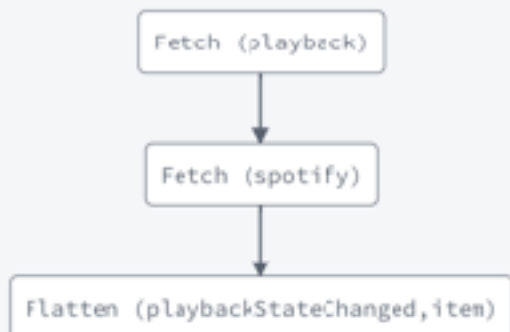
Real-Time Data



```
subscription PlaybackStateSubscriberSubscription {
  playbackStateChanged {
    ... PlaybackStateFragment
    __typename
  }
}
```

```
fragment PlaybackStateFragment on PlaybackState {
  progressMs
  isPlaying
  repeatState
  shuffleState
}
```

Query Plan Preview



Subscriptions

```
// Response received at 04:10:00
{
  "payload": {
    "playbackStateChanged": {
      "progressMs": 109400,
      "isPlaying": true,
      "repeatState": "CORRECT",
      "shuffleState": false,
      "actions": {
        "isAllowed": {
          "isAllowed": true
        },
        "__typename": "ActionSet"
      }
    },
    "spotify": {
      "uri": "spotify:track:15170560200256041968",

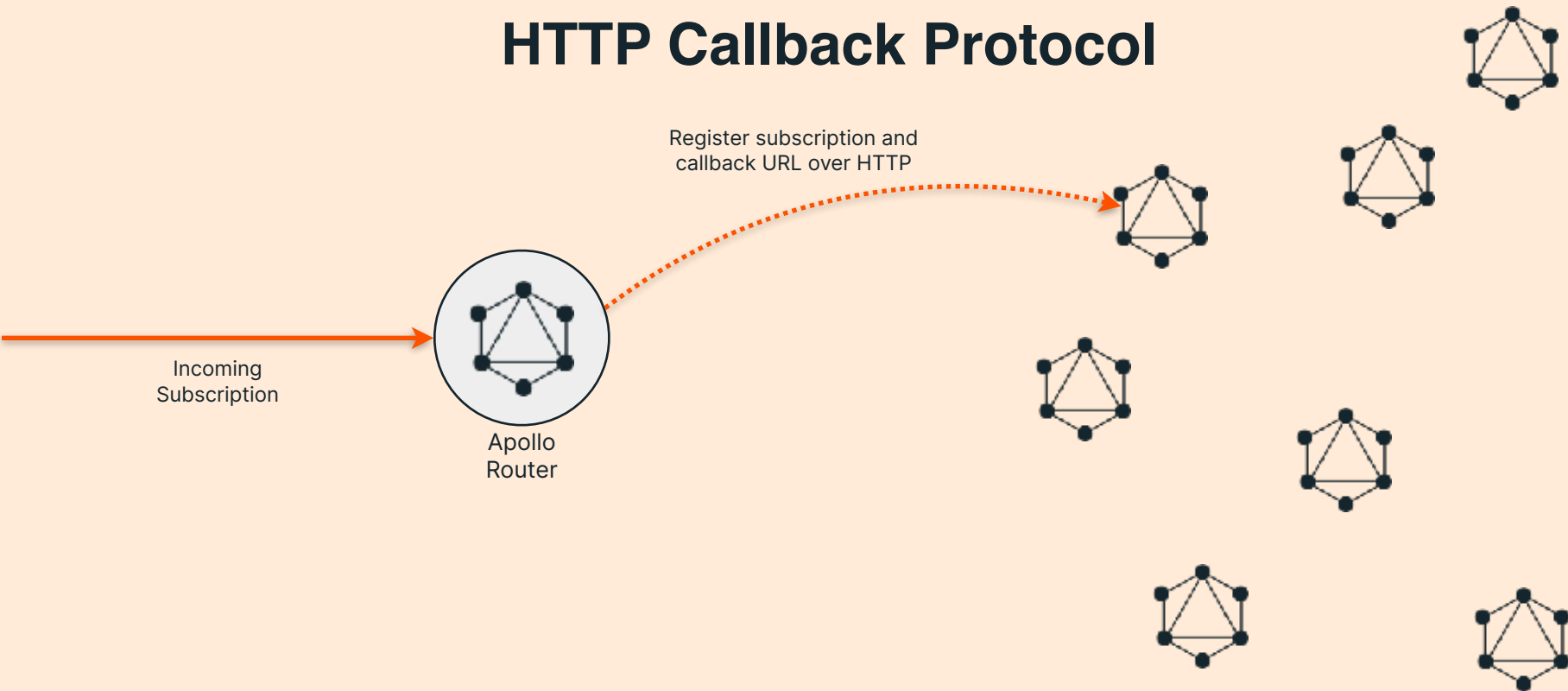
```

Pushing the boundaries with Real-time

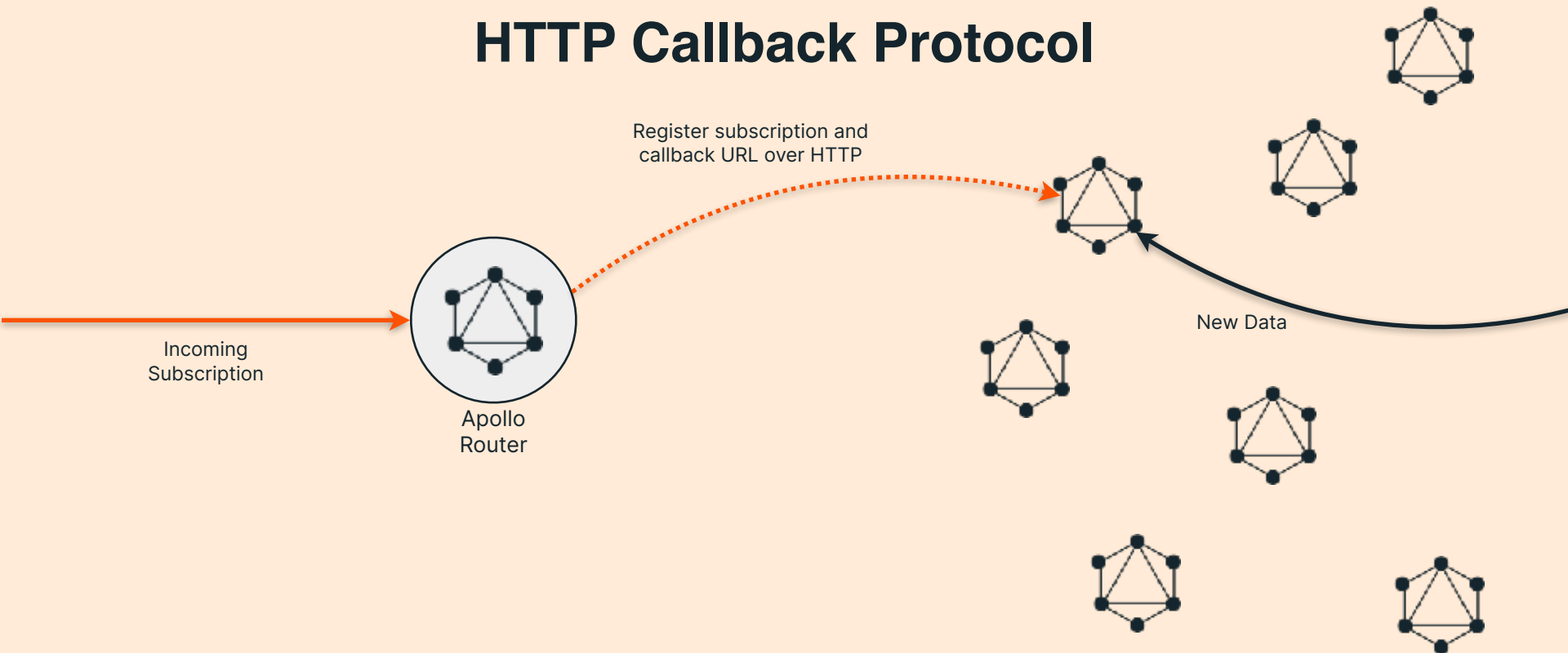
WebSockets

HTTP Callback

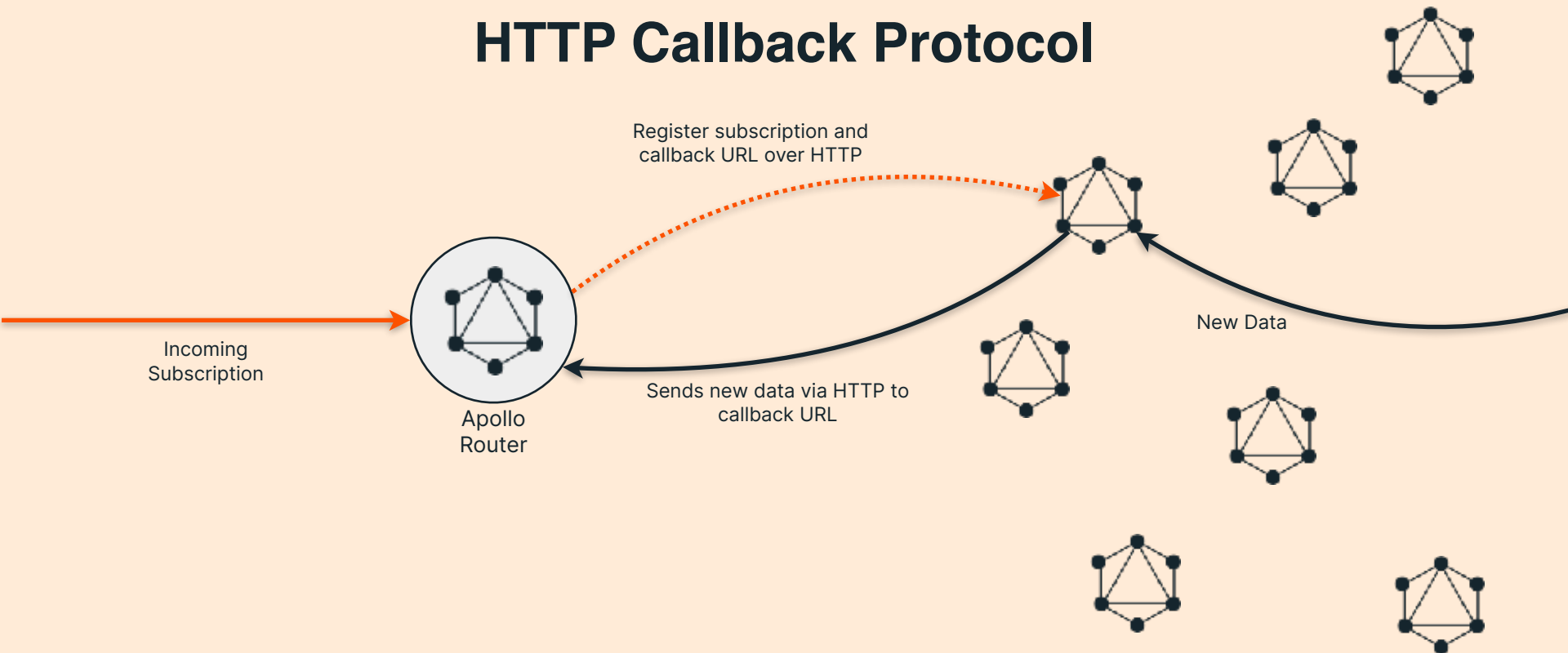
HTTP Callback Protocol



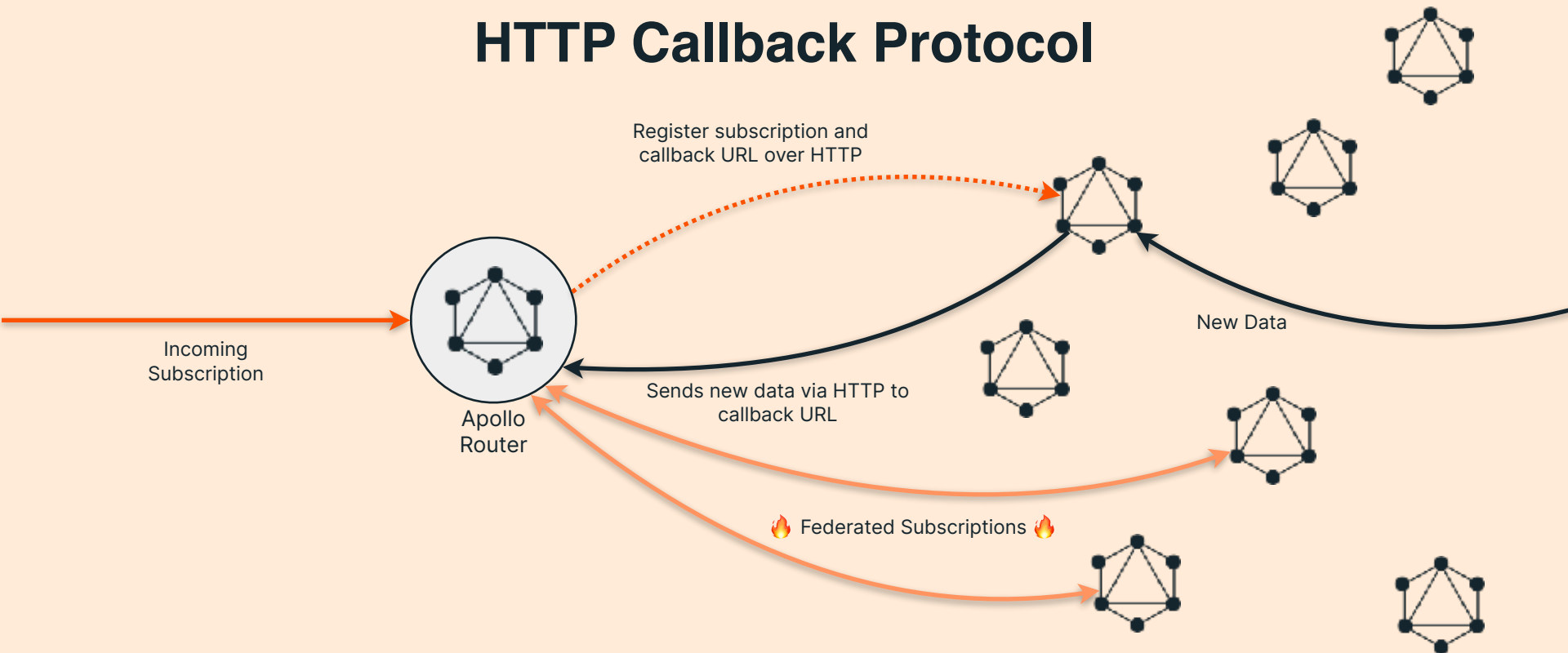
HTTP Callback Protocol



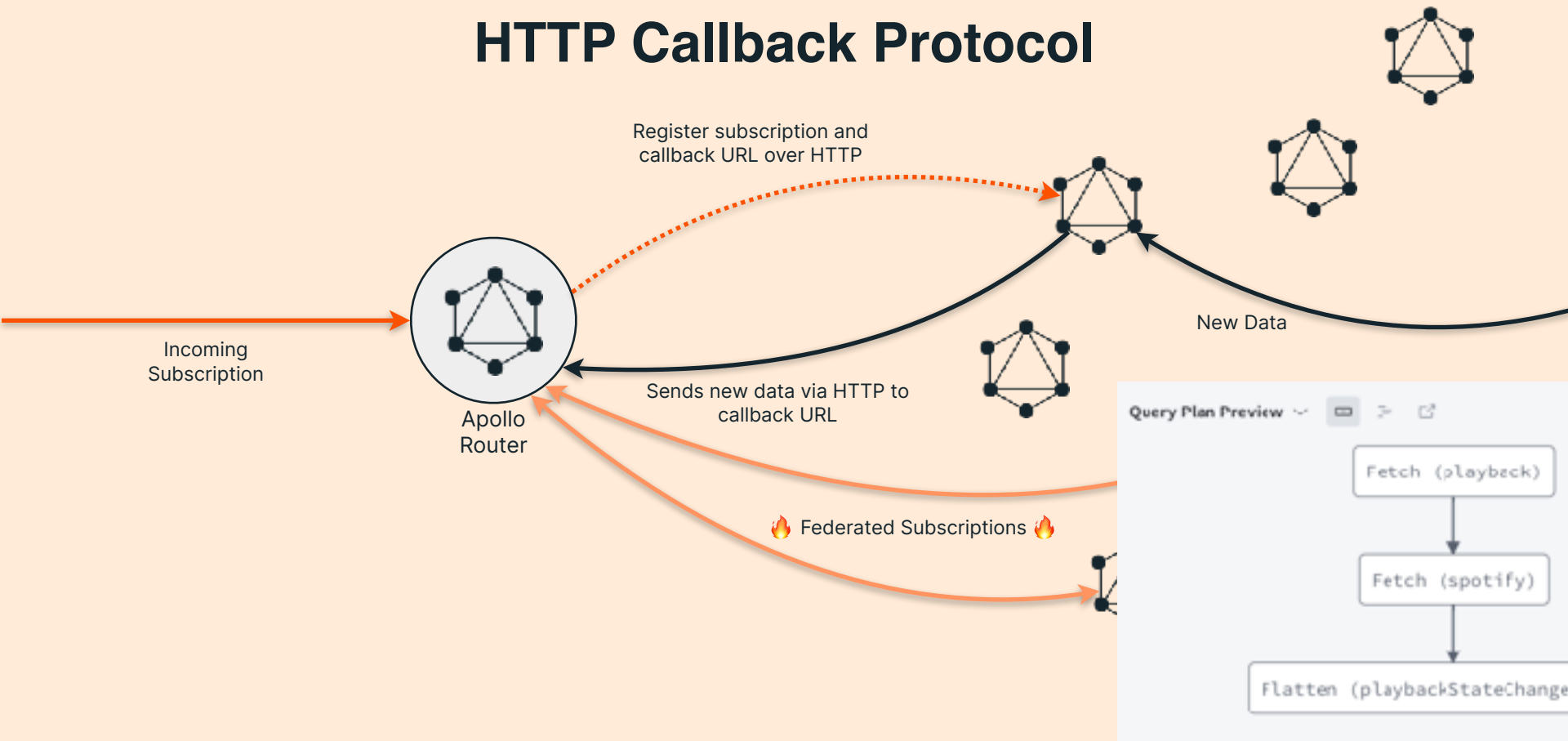
HTTP Callback Protocol



HTTP Callback Protocol



HTTP Callback Protocol



Subscriptions in your flavor 🍦

```
subscription:
  enabled: true
  mode:
    passthrough:
      subgraphs:
        reviews:
          path: /ws
          protocol: graphql_transport_ws
    preview_callback:
      public_url: http://public_url_of_my_router_instance:4000
      listen: 0.0.0.0:4000
      path: /callback
      subgraphs:
        - accounts
```

It's just config for the router

Feature request:

On the Tracks page, we should progressively load the data. You don't need the album or artist details to start interacting with the page.

```
query TrackRouteQuery($trackId: ID!) {
```

```
  track(id: $trackId) {
```

FRONTEND

BACKEND



```
  topTracks {
```

```
    id
```

```
    name
```

The Power of @defer

The screenshot displays the GraphQL Playground interface. At the top, the URL is `http://localhost:4000/`. The main area shows a query editor with the following code:

```
1 query TrackRouteQuery($trackId: ID!) {  
2   track(id: $trackId) {  
3     id  
4     durationMs  
5     name  
6     ...TrackFragment  
7   }  
8 }  
9  
10 fragment TrackFragment on Track {  
11   album {  
12     id  
13     albumType  
14     name  
15     images {
```

The query is executed, and the status is `STATUS 200` with a response time of `103ms`. The query plan is visible, showing a `Fetch (spotify)` operation. The variables section shows the following JSON:

```
1 {  
2   "trackId": "06LzBjkKZpAM1EuB61Vr1"  
3 }
```

The interface also includes a sidebar with navigation icons and a top bar with `Refetch Schema` and `Publish` buttons.

The Power of @defer

The screenshot displays the GraphQL Playground interface. On the left, the 'Operation' tab shows a query for 'TrackRouteQuery' with a deferred fragment. The query is as follows:

```
1 query TrackRouteQuery($trackId: ID!) {  
2   track(id: $trackId) {  
3     id  
4     durationMs  
5     name  
6     ...TrackFragment @defer  
7   }  
8 }  
9  
10 fragment TrackFragment on Track {  
11   album {  
12     id  
13     albumType  
14     name  
15     images {
```

Below the query, the 'Variables' tab shows the following JSON:

```
1 {  
2   "trackId": "06Lz8UkkZpAMfEub8iVri"  
3 }
```

On the right, the 'Query Plan' tab shows a tree diagram illustrating the execution flow:

- The root node is 'Defer', which branches into 'Primary' and 'Deferred'.
- 'Primary' leads to a 'Fetch (spotify)' node.
- 'Deferred' leads to a 'Fetch (spotify)' node, which then leads to a 'Flatten (track)' node.

At the top right of the interface, the status is 'STATUS 200', the execution time is '103ms', and the response size is '4.0KB'. The 'Login' button is visible in the top right corner.

The Power of @defer

The screenshot shows the GraphQL Playground interface. The URL is `http://localhost:4000/`. The operation is named `TrackRouteQuery`. The query is:

```
1 query TrackRouteQuery($trackId: ID!) {
2   track(id: $trackId) {
3     id
4     durationMs
5     name
6     ...TrackFragment @defer
7   }
8 }
9
10 fragment TrackFragment on Track {
11   album {
12     id
13     albumType
14     name
15     images {
```

The response is:

```
{
  "data": {
    "track": {
      "id": "0ELzBJkKZpPAM1EwB61Vr1",
      "durationMs": 239993,
      "name": "To Your Honor"
    }
  }
}
```

The response status is `STATUS 200`, with a response time of `103ms` and a size of `4.0KB`. The response timeline shows a duration of `25.0ms`.

Variables:

```
1 {}
2 "trackId": "0ELzBJkKZpPAM1EwB61Vr1"
3 {}
```

Buttons: `Refetch Schema`, `Publish`, `Log in`, `+ Add files`.

The Power of @defer

The screenshot displays the GraphQL Playground interface. The top navigation bar includes a 'Sandbox' tab, the URL 'http://localhost:4300/', and buttons for 'Refresh Schema' and 'Publish'. The main area is divided into three sections: 'Operation', 'Response', and 'Variables/Script'.

Operation: The query is as follows:

```
1 query TrackRouteQuery($trackId: ID!) {
2   track(id: $trackId) {
3     id
4     durationMs
5     name
6     ...TrackFragment @defer
7   }
8 }
9
10 fragment TrackFragment on Track {
11   album {
12     id
13     albumType
14     name
15     images {
```

Response: The JSON response is:

```
{
  "data": {
    "track": {
      "id": "0ELzBJkKZpPAM1EwB61Vr1",
      "durationMs": 230000,
      "name": "To Your Honor",
      "album": [
        {
          "id": "2cprpJCYbCbP2RKYG1EJkZ",
          "albumType": "ALBUM",
          "name": "To Your Honor",
          "images": [
            {
              "url": "https://i.scdn.co/image/ab67616d0000b1e8738c44879c75e2685881b35d72"
            },
            {
              "url": "https://i.scdn.co/image/ab67616d0000b1e8738c44879c75e2685881b35d72"
            }
          ]
        }
      ]
    }
  }
}
```

Variables: The 'Variables' section shows a single variable:

```
1 {
2   "trackId": "0ELzBJkKZpPAM1EwB61Vr1"
3 }
```

The 'Response Timeline' at the bottom indicates a total response time of 32.0ms.



GraphQL
makes
your APIs
better

You can add
GraphQL to your
existing API
today

Thanks!



Michael Watson

Developer Relations at Apollo

<https://discord.gg/graphos>

APOLLO