



# GraphQL in 2020



Photo by Nicolas Gras

# Who am I

GraphQL in 2020



Charly POLY

# Who am I

GraphQL in 2020



Charly POLY

Lead Sr. Software Engineer at Double



# Who am I

GraphQL in 2020



Charly POLY

Lead Sr. Software Engineer at Double 

Big fan and user of GraphQL since 4 years

## 1. Schema

```
1 type Project {  
2   name: String!  
3   tagline: String!  
4   contributors: [User]  
5 }  
6  
7 type Query {  
8   project(name: String!): Project  
9 }
```

## 1. Schema

```
1 type Project {  
2   name: String!  
3   tagline: String!  
4   contributors: [User]  
5 }  
6  
7 type Query {  
8   project(name: String!): Project  
9 }
```



```
1 const resolvers = {  
2   Query: {  
3     project: (parent, args, context, info) => (  
4       projects.find(p => p.name === args.name)  
5     ),  
6   },  
7 };;
```

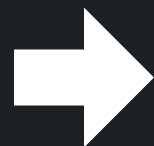
## 2. Resolvers

## 1. Schema

```
1 type Project {  
2   name: String!  
3   tagline: String!  
4   contributors: [User]  
5 }  
6  
7 type Query {  
8   project(name: String!): Project  
9 }
```



```
1 const resolvers = {  
2   Query: {  
3     project: (parent, args, context, info) => (  
4       projects.find(p => p.name === args.name)  
5     ),  
6   },  
7 };;
```



```
1 {  
2   project(name: "GraphQL") {  
3     tagline  
4   }  
5 }
```

## 2. Resolvers

## 3. Query

# GraphQL in 1'

GraphQL in 2020

## 1. Schema

```
1 type Project {  
2   name: String!  
3   tagline: String!  
4   contributors: [User]  
5 }  
6  
7 type Query {  
8   project(name: String!): Project  
9 }
```



```
1 const resolvers = {  
2   Query: {  
3     project: (parent, args, context, info) => (  
4       projects.find(p => p.name === args.name)  
5     ),  
6   },  
7 };;
```

## 2. Resolvers

## 4. JSON data

```
1 {  
2   "project": {  
3     "tagline": "A query language for APIs"  
4   }  
5 }
```



```
1 {  
2   project(name: "GraphQL") {  
3     tagline  
4   }  
5 }
```

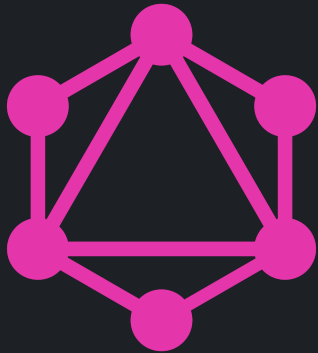
## 3. Query





# GraphQL in 2015

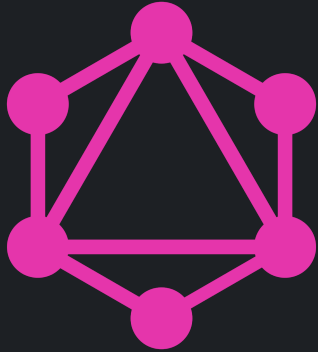
GraphQL in 2020



## initial public release

server: graphql@0.0.2 (+ express)

client: relay@0.1.0



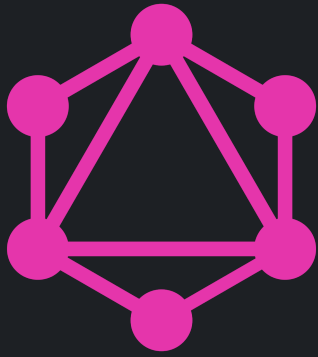
## initial public release

server: graphql@0.0.2 (+ express)

```
1 import {
2   graphql,
3   GraphQLSchema,
4   GraphQLObjectType,
5   GraphQLString,
6 } from 'graphql';
7
8 var schema = new GraphQLSchema({
9   query: new GraphQLObjectType({
10     name: 'RootQueryType',
11     fields: {
12       hello: {
13         type: GraphQLString,
14         resolve() {
15           return 'world';
16         },
17       },
18     },
19   }),
20 });
```

# GraphQL in 2016

GraphQL in 2020



## Ecosystem awakening

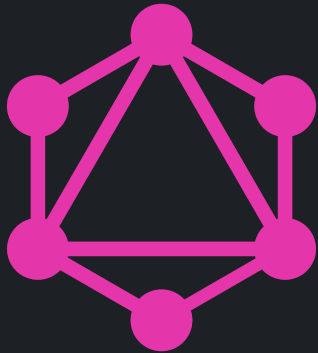
graphql-tools

apollo-server

apollo-client (+ react-apollo)

graphiql

## Libraries to boost development



## Ecosystem awakening

graphql-tools

```
1 import { makeExecutableSchema } from '@graphql-tools'
2
3
4 const resolvers = /*...*/
5
6 const typeDefs = `
7 # ...
8
9 schema {
10   query: Query
11   mutation: Mutation
12 }
13 `;
14
15 const executableSchema = makeExecutableSchema({
16   typeDefs,
17   resolvers,
18 });
```

# GraphQL in 2016 - intermission

GraphQL in 2020

Spotify GraphQL Console

```
1 {
2   me {
3     display_name
4     playlists(throttle: 100, continueOnError: 1, limit: -1) {
5       name
6     }
7   }
8 }
9
```

```
{
  "data": {
    "me": {
      "display_name": "Charly Poly",
      "playlists": [
        {
          "name": "Trajectoire"
        },
        {
          "name": "This Is Népal"
        },
        {
          "name": "2Fingz"
        },
        {
          "name": "This Is Basshunter"
        },
        {
          "name": "Standards de la Chanson Française"
        },
        {
          "name": "This Is Masego"
        },
        {
          "name": "Reggeaton 2019"
        },
        {
          "name": "Max Richter – The Blue Notebooks (15 Years)"
        },
        {
          "name": "Arrival Soundtrack"
        },
        {
          "name": "40"
        },
        {
          "name": "Night walk 🌙"
        },
        {
          "name": "Nights"
        },
        {
          "name": "CLASSIC RAMMERS"
        },
        {
          "name": "Pickup Music Weekly - R&B, Indie Pop & Folk | New Music | Pickup Music 2020"
        }
      ]
    }
  }
}
```

< Schema Query X

Q Search Query...

the schema allows the following query:

FIELDS

me: PrivateUser

user(id: String!): PublicUser

track(id: String, name: String): Track

tracks(ids: String!): [Track]

audio\_features(trackIds: String!): [AudioFeatures]

audio\_feature(trackId: String!): AudioFeatures

artist(id: String, name: String): Artist

artists(ids: String!): [Artist]

album(id: String!): Album

albums(ids: String!): [Album]

playlist(id: String!, userId: String!): Playlist

 @whereischarly

POWERED WITH SPOTIFY | [Open source code](#) | [About us](#)

6



*Feedback written on August 04, 2018*

## **Why use GraphQL, good and bad reasons**

*Feedback written on August 04, 2018*

## **Why use GraphQL, good and bad reasons**

- **Build smooth user-experiences**  
"Ask for what you want", optimistic UIs



*Feedback written on August 04, 2018*

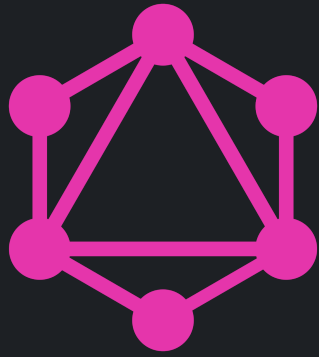
## **Why use GraphQL, good and bad reasons**

- **Build smooth user-experiences**  
"Ask for what you want", optimistic UIs
- **Solve data-complexity issue on front-end side**  
Apollo cache, typed mutations, DDD APIs

*Feedback written on August 04, 2018*

## Why use GraphQL, good and bad reasons

- **Build smooth user-experiences**  
"Ask for what you want", optimistic UIs
- **Solve data-complexity issue on front-end side**  
Apollo cache, typed mutations, DDD APIs
- **Microservices orchestration**  
Apollo schema stitching ➡ Apollo Federation



## Ecosystem on the rise

**Libraries:** Apollo, Relay

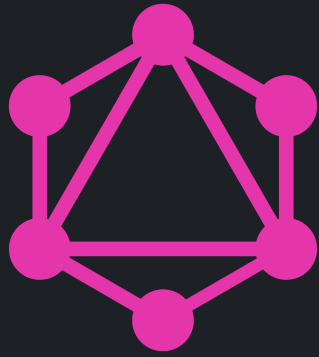
**Concepts:** Schema stitching, client cache, optimistic UI patterns

**Services:** AWS App Sync, Apollo Optics, Prisma

**Faster and more robust development**

# GraphQL in 2018

GraphQL in 2020



First public GraphQL APIs



# GraphQL in 2019

GraphQL in 2020

## State of JS 2019

### Highest Interest

Awarded to the technology developers are most interested in learning.



**89.6%** of developers who have heard about GraphQL want to learn it. That's some serious interest!

# GraphQL in 2020?

GraphQL in 2020

# GraphQL in 2020?

GraphQL in 2020

GraphQL modules

graphqless

Apollo Federation

Hasura

GraphQL Mesh

AWS AppSync

Prisma

GraphQL Code Generator

Relay

DGraph

Apollo Platform

Hasura

GraphQL Engine

GraphCMS

urql

Gatsby

Apollo State management

TypeGraphQL

# GraphQL in 2020?

GraphQL in 2020

GraphQL modules

graphqless

Apollo Federation

Hasura

GraphQL Mesh

AWS AppSync

Prisma

GraphQL Code Generator

Relay

DGraph

Apollo Platform

Hasura

GraphQL Engine

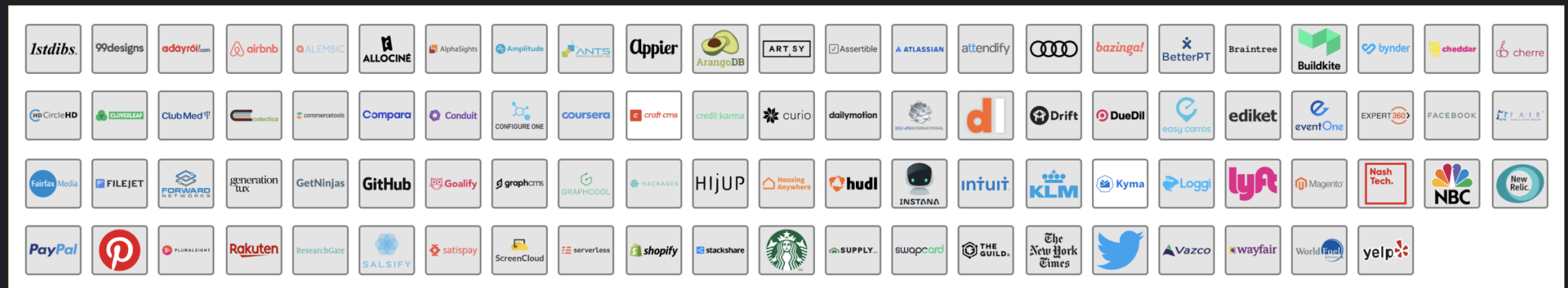
Apollo State management

GraphCMS

urql

TypeGraphQL

Gatsby





*“ GraphQL is much more than an efficient way of fetching data from the client side*

*“ GraphQL is much more than an efficient way of fetching data from the client side*

*“ GraphQL is much more than an efficient way of fetching data from the client side*

## 1. The GraphQL innovations on front-end side

*“ GraphQL is much more than an efficient way of fetching data from the client side*

1. The GraphQL innovations on front-end side
2. GraphQL bridging the gaps on server-side



# The GraphQL innovations on front-end side

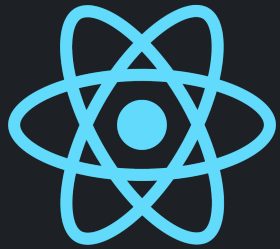
GraphQL as state management

# GraphQL as state management

GraphQL in 2020

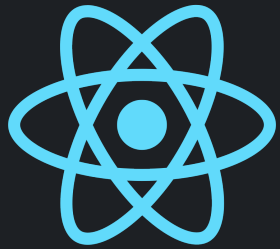
# GraphQL as state management

GraphQL in 2020



# GraphQL as state management

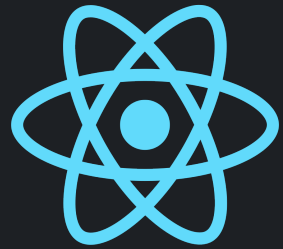
GraphQL in 2020





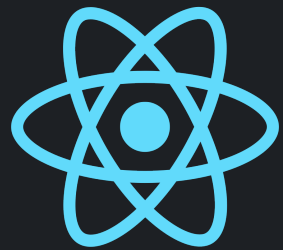
# GraphQL as state management

GraphQL in 2020



# GraphQL as state management

GraphQL in 2020



Recoil

# GraphQL as state management

GraphQL in 2020

*“ GraphQL is a “data query and manipulation language for APIs”*

# GraphQL as state management

GraphQL in 2020

*“ GraphQL is a “data query and manipulation language for ~~APIs~~”*



# GraphQL as state management

GraphQL in 2020

Apollo Client mission

# GraphQL as state management

GraphQL in 2020

## Apollo Client mission

- **Apollo Client 1 & 2:** build the best GraphQL Client

# GraphQL as state management

GraphQL in 2020

## Apollo Client mission

- **Apollo Client 1 & 2:** build the best GraphQL Client
- **Apollo Client 3:** a library for interacting with a client-side data graph

# GraphQL as state management

GraphQL in 2020

## Apollo Client mission

- **Apollo Client 1 & 2:** build the best GraphQL Client
- **Apollo Client 3:** a library for interacting with a client-side data graph

## Local state management with GraphQL



# GraphQL as state management

GraphQL in 2020

## Simple local state example

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

```
1  const { data } = useQuery(`
2    query isDarkMode {
3      darkMode @client
4    }
5  `)
```

# GraphQL as state management

GraphQL in 2020

## Example: darkMode settings

```
1 const { data } = useQuery(`
2   query isDarkMode {
3     darkMode @client
4   }
5 `)
```

```
1 import { TypePolicies, makeVar } from '@apollo/client';
2
3
4 export const DARK_MODE_KEY = 'dark-mode'
5
6 export const darkMode = makeVar<boolean>(localStorage.getItem(DARK_MODE_KEY) || false)
7
8
9 export const resolvers: TypePolicies = {
10   Query: {
11     fields: {
12       darkMode: {
13         read: () => darkMode(),
14       },
15     },
16   },
17 };
18
19
20 export default resolvers
```

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

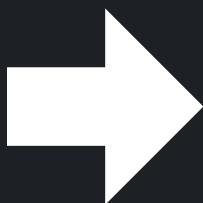
```
1 darkMode(true)
```

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

```
1 darkMode(true)
```



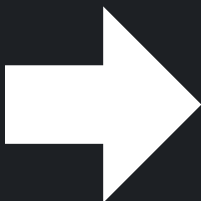
```
1 import { TypePolicies, makeVar } from '@apollo/client';
2
3
4 export const DARK_MODE_KEY = 'dark-mode'
5
6 export const darkMode = makeVar<boolean>(localStorage.get
7
8
9 export const resolvers: TypePolicies = {
10   Query: {
11     fields: {
12       darkMode: {
13         read: () => darkMode(),
14       },
15     },
16   },
17 };
18
19
20 export default resolvers
```

# GraphQL as state management

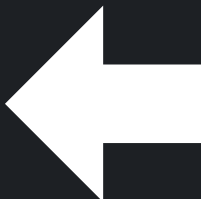
GraphQL in 2020

Example: darkMode settings

```
1 darkMode(true)
```



updates  
useQuery() hook



```
1 import { TypePolicies, makeVar } from '@apollo/client';
2
3
4 export const DARK_MODE_KEY = 'dark-mode'
5
6 export const darkMode = makeVar<boolean>(localStorage.get
7
8
9 export const resolvers: TypePolicies = {
10   Query: {
11     fields: {
12       darkMode: {
13         read: () => darkMode(),
14       },
15     },
16   },
17 };
18
19
20 export default resolvers
```



# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

# GraphQL as state management

GraphQL in 2020

Example: darkMode settings

```
1 import { darkMode, DARK_MODE_KEY } from './resolvers'
2
3 const useUpdateDarkMode = () => useCallback(
4   (enabled: boolean) => {
5     localStorage.setItem(DARK_MODE_KEY, enabled)
6     darkMode(enabled)
7   },
8   [],
9 )
```

More advanced local state example

# GraphQL as state management

GraphQL in 2020

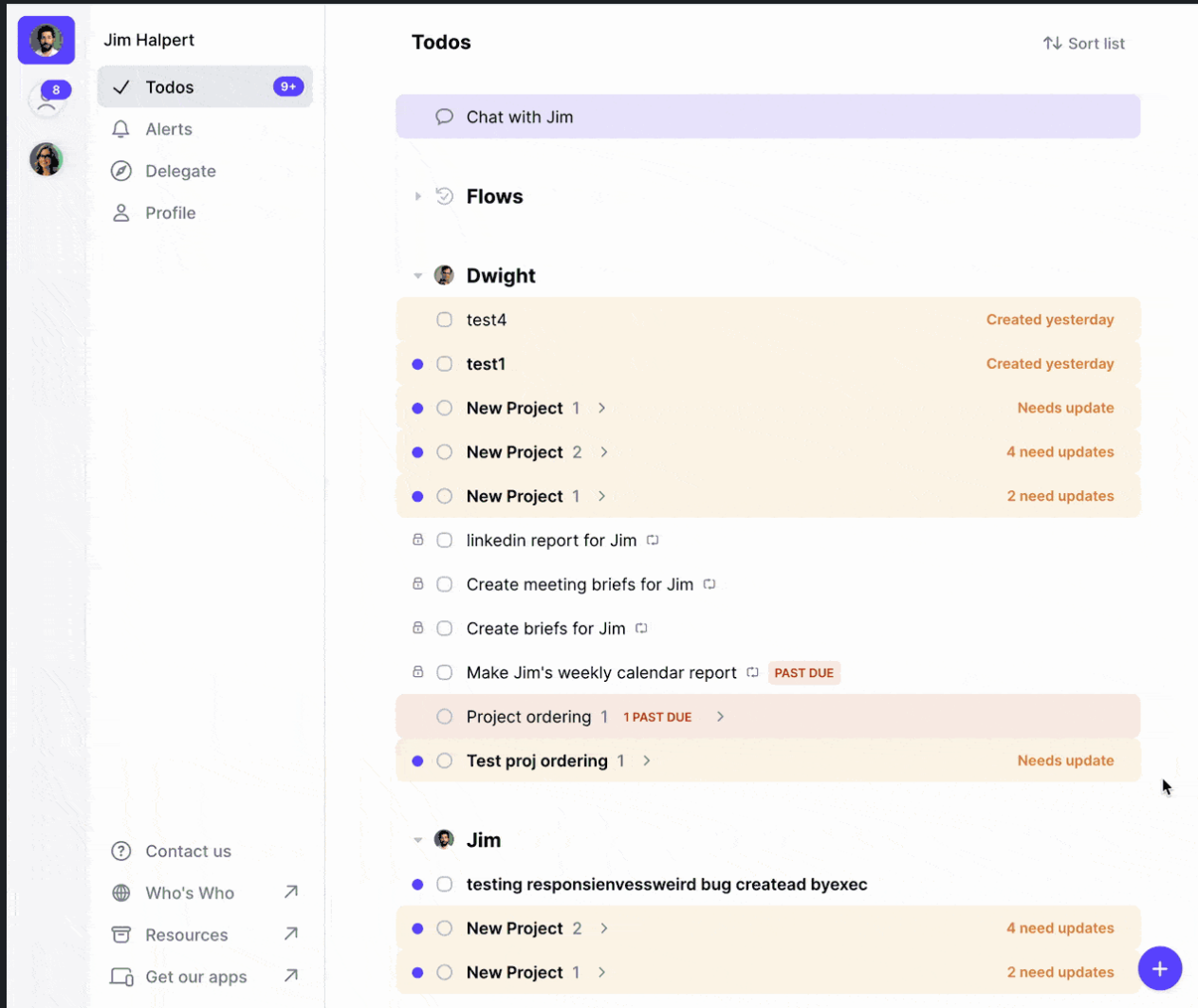
The screenshot displays a web application interface for managing tasks. On the left, a sidebar shows the user 'Jim Halpert' and navigation links: 'Todos' (9+), 'Alerts', 'Delegate', and 'Profile'. The main content area is titled 'Todos' and includes a 'Sort list' button. The tasks are organized into sections for different users:

- Chat with Jim** (purple bar)
- Flows** (arrow icon)
- Dwight** (dropdown arrow):
  - test4 (Created yesterday)
  - test1 (Created yesterday)
  - New Project 1 (Needs update)
  - New Project 2 (4 need updates)
  - New Project 1 (2 need updates)
  - linkedin report for Jim
  - Create meeting briefs for Jim
  - Create briefs for Jim
  - Make Jim's weekly calendar report (PAST DUE)
  - Project ordering 1 (1 PAST DUE)
  - Test proj ordering 1 (Needs update)
- Jim** (dropdown arrow):
  - testing responsienvessweird bug createad byexec
  - New Project 2 (4 need updates)
  - New Project 1 (2 need updates)

A blue circular button with a white '+' sign is located at the bottom right of the task list.

# GraphQL as state management

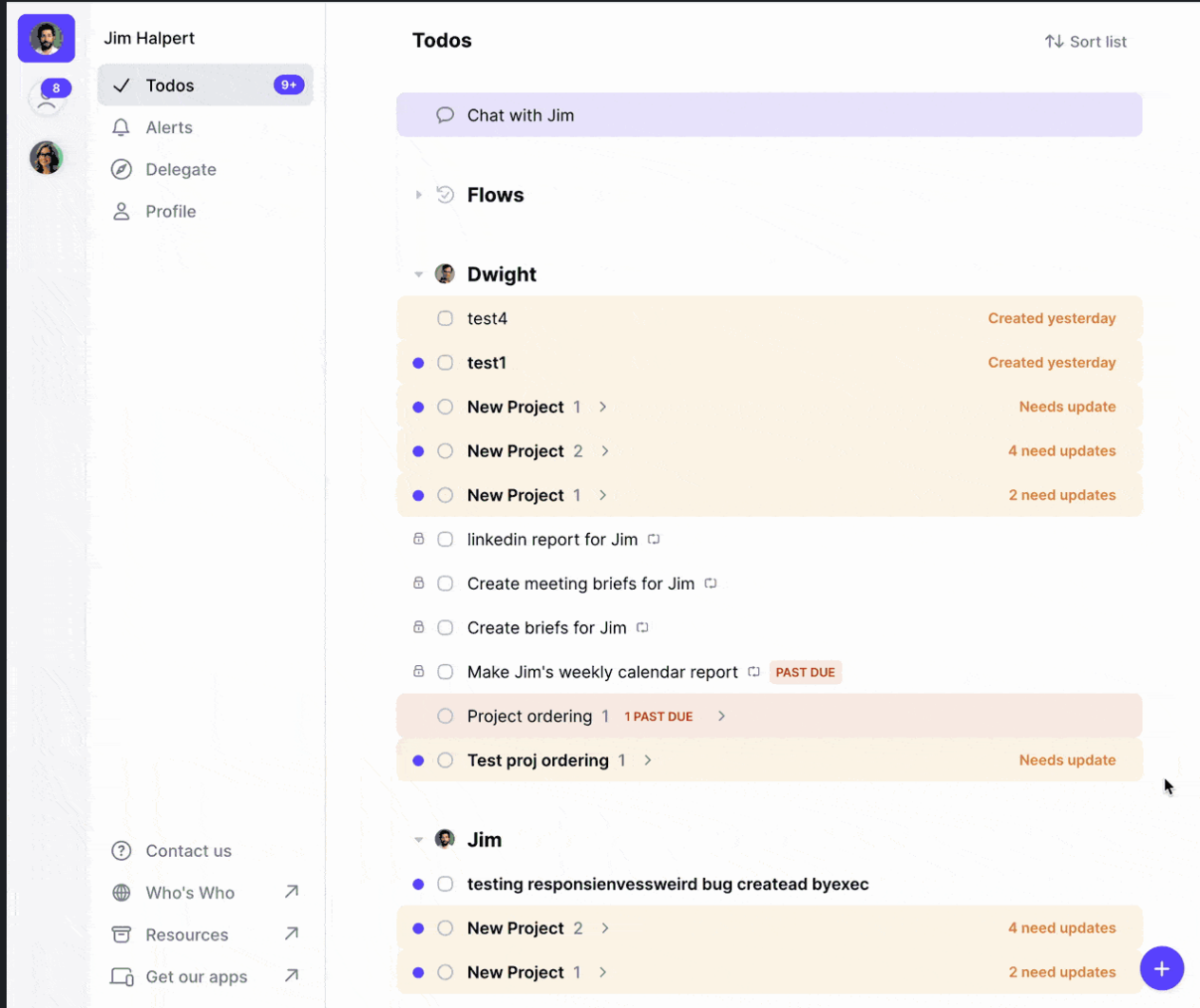
GraphQL in 2020



- TodoList items are remote (GQL API)

# GraphQL as state management

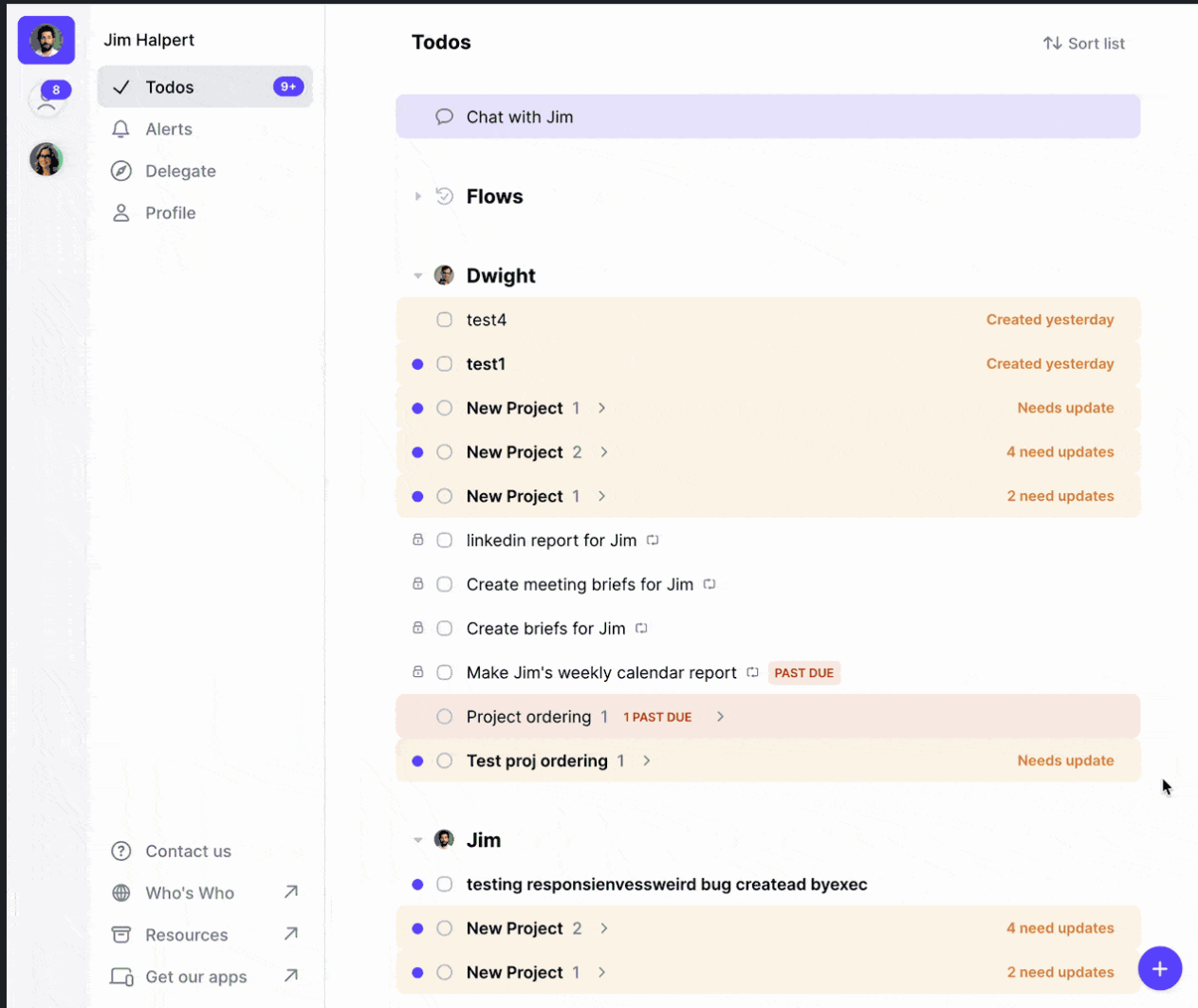
GraphQL in 2020



- TodoList items are remote (GQL API)
- The newly created Todo is local

# GraphQL as state management

GraphQL in 2020



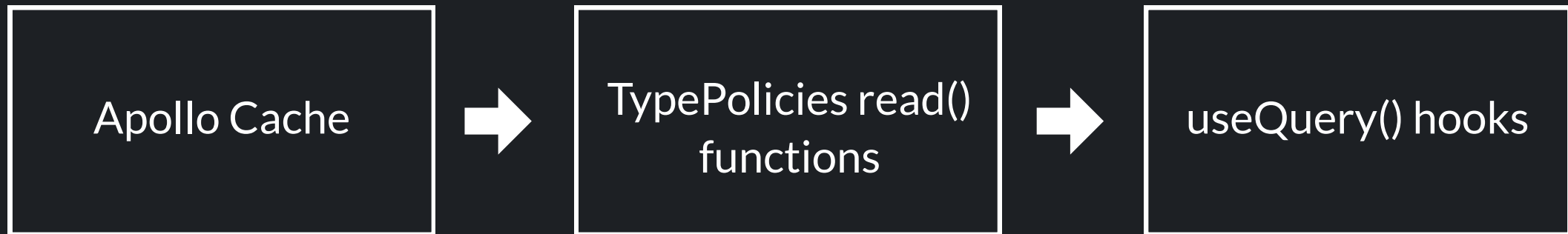
- TodoList items are remote (GQL API)
- The newly created Todo is local
- Totally transparent at component- level

```
1 const { data, error } = useMainTopicListQuery()  
2  
3 // data contains remote + local todos
```

# GraphQL as state management

GraphQL in 2020

Adding the TodoDraft to the list Query





# GraphQL as state management

GraphQL in 2020

## 1. Adding the TodoDraft to the list Query

```
1  const { data } = useQuery(`
2    query getMainTopicList($completedAfter: DateTime!) {
3      activeTopics: topicsList(first: 200, view: ACTIVE) {
4        hasAfter
5        items {
6          ...TopicListItem
7        }
8      }
9    }
10 `)
```

# GraphQL as state management

GraphQL in 2020

## 1. Adding the TodoDraft to the list Query

```
1 const { data } = useQuery(`
2   query getMainTopicList($completedAfter: DateTime!) {
3     activeTopics: topicsList(first: 200, view: ACTIVE) {
4       hasAfter
5       items {
6         ...TopicListItem
7       }
8     }
9   }
10 `)
```

```
1 const typePolicies: TypePolicies = {
2   Query: {
3     fields: {
4       topicsList: {
5         keyArgs: ['view', 'first'],
6         // ...
7         read: (existing, { args }) => {
8           const draft = todoDrafts()
9           return draft ?
10             {
11               ...existing,
12               items: [
13                 draft,
14                 ...(existing ? existing.items : []),
15               ],
16             } :
17             existing
18         },
19       },
20     },
21   },
22 }
```

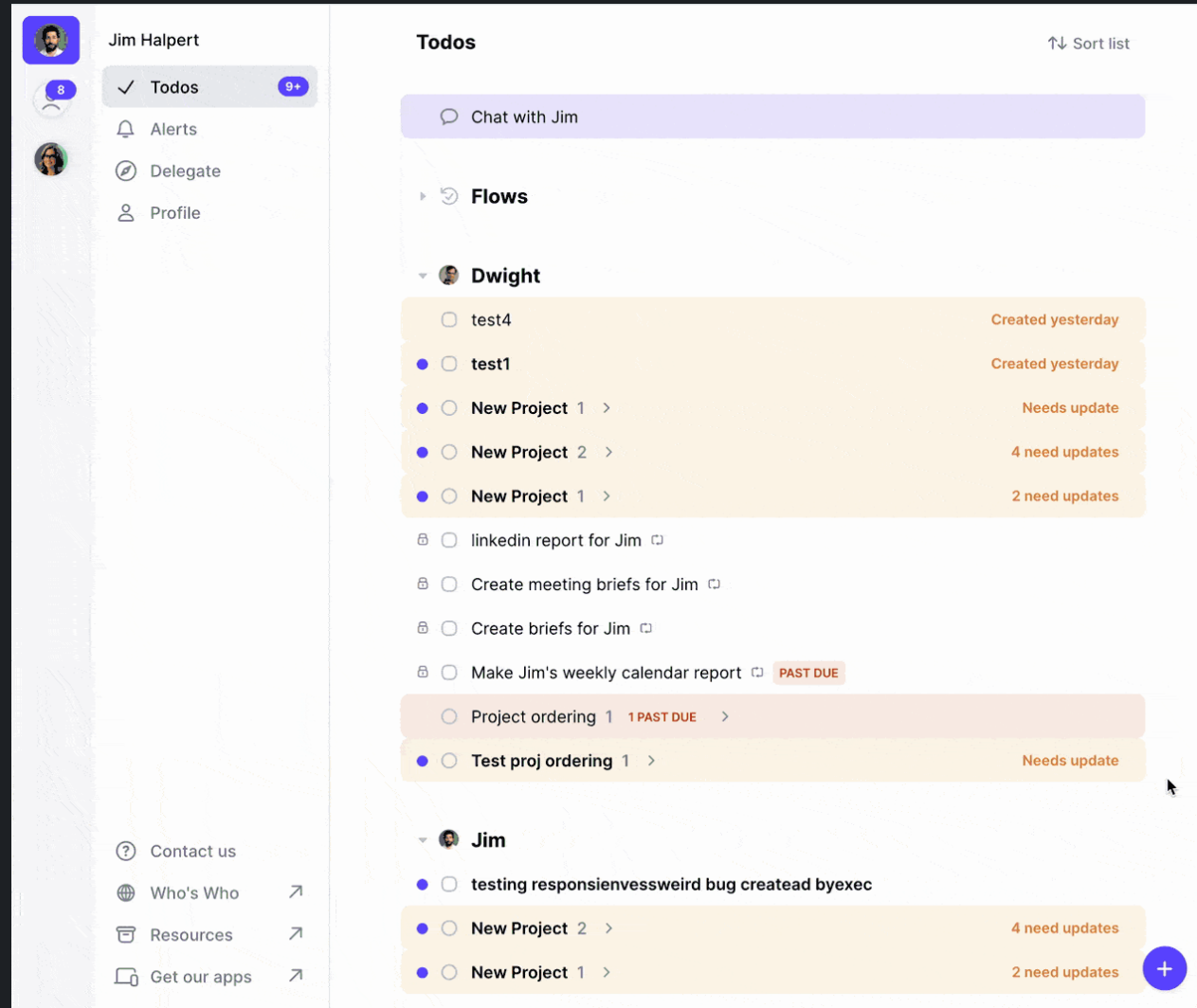
## 2. Todo `isDraft` local property

```
1 fragment TodoListItem on Todo {  
2   id  
3   reference  
4   title  
5   # ...  
6   isDraft @client  
7   # ...  
8 }
```

```
1 export const resolvers: TypePolicies = {  
2   // we augment the `Todo` type  
3   Todo: {  
4     fields: {  
5       isDraft: {  
6         read: (existing = false, { readField }) => isDraft(readField('id')),  
7       },  
8     },  
9   },  
10  };
```

# GraphQL as state management

GraphQL in 2020



# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

- **State:** managed by ApolloCache, along side with APIs data

# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

- **State:** managed by ApolloCache, along side with APIs data
- **Computed values:** TypePolicies

# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

- **State:** managed by ApolloCache, along side with APIs data
- **Computed values:** TypePolicies
- **Actions:** reactive variables or client.writeQuery()



# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

- **State:** managed by ApolloCache, along side with APIs data
- **Computed values:** TypePolicies
- **Actions:** reactive variables or client.writeQuery()
- **Reactions:** Apollo ObservableQuery + reactive variables

# GraphQL as state management

GraphQL in 2020

Full local state management capabilities

- **State:** managed by ApolloCache, along side with APIs data
- **Computed values:** TypePolicies
- **Actions:** reactive variables or client.writeQuery()
- **Reactions:** Apollo ObservableQuery + reactive variables
- **Tools:** Apollo Client Dev tools



# The GraphQL innovations on front-end side

GraphQL Full-stack type safety

*“ GraphQL introspection is a core  
- most underrated - feature of the language*

# GraphQL Full-stack type safety

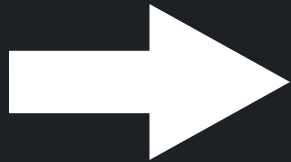
GraphQL in 2020

```
1 {  
2   __schema {  
3     types {  
4       name  
5       kind  
6       fields {  
7         name  
8         type {  
9           name  
10          ofType {  
11            name  
12          }  
13        }  
14      }  
15    }  
16  }  
17 }
```

# GraphQL Full-stack type safety

GraphQL in 2020

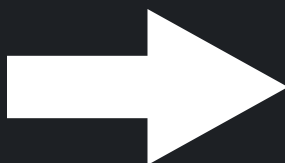
```
1 {  
2   __schema {  
3     types {  
4       name  
5       kind  
6       fields {  
7         name  
8         type {  
9           name  
10          ofType {  
11            name  
12          }  
13        }  
14      }  
15    }  
16  }  
17 }
```



# GraphQL Full-stack type safety

GraphQL in 2020

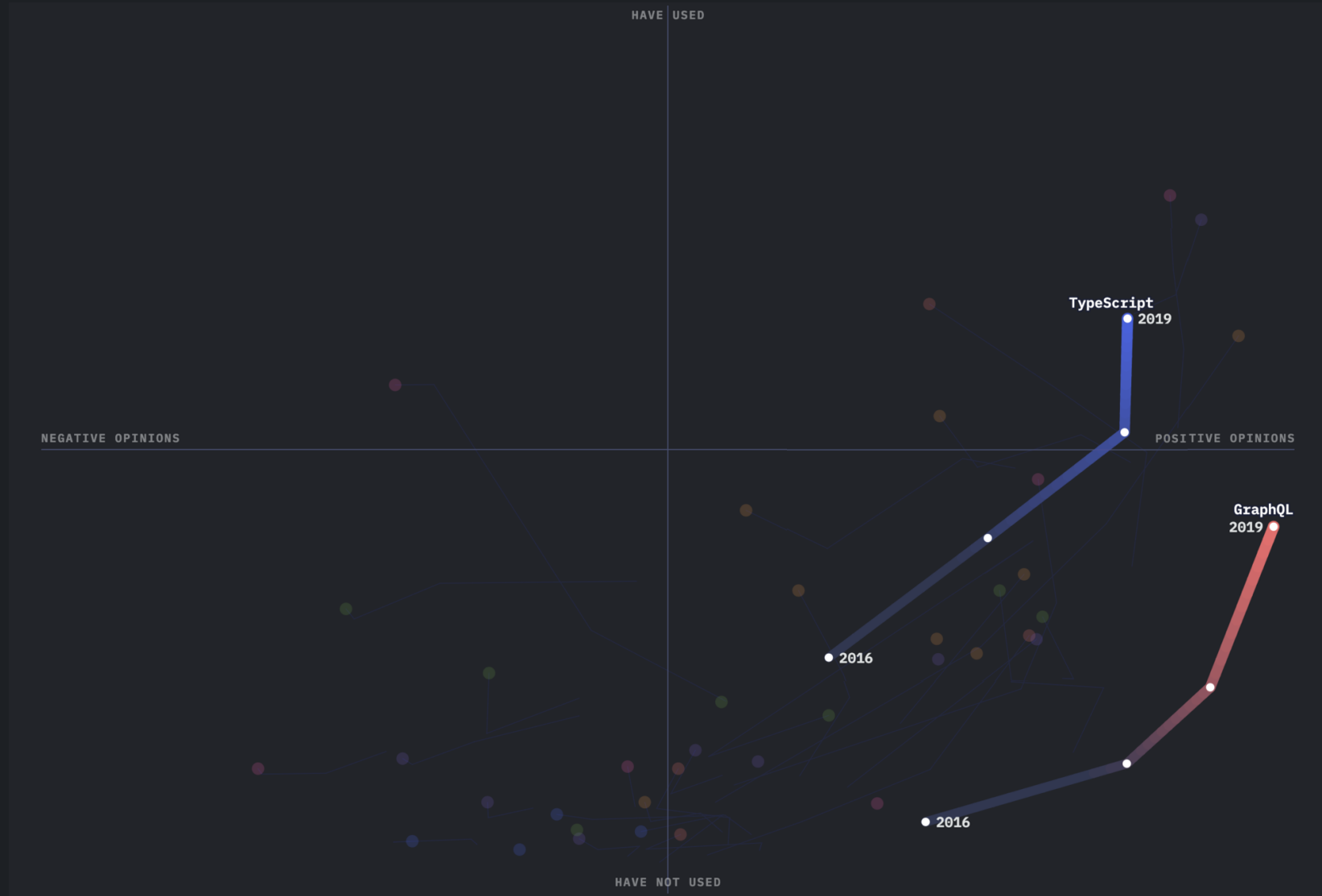
```
1 {
2   __schema {
3     types {
4       name
5       kind
6       fields {
7         name
8         type {
9           name
10          ofType {
11            name
12          }
13        }
14      }
15    }
16  }
17 }
```



```
1 {
2   "data": {
3     "__schema": {
4       "types": [
5         {
6           "name": "Query",
7           /* ... */
8         },
9         {
10          "name": "PrivateUser",
11          "kind": "OBJECT",
12          "fields": [
13            {
14              "name": "id",
15              "type": {
16                "name": "String",
17                "ofType": null
18              }
19            },
20            {
21              "name": "birthday",
22              "type": {
23                "name": "String",
24                "ofType": null
25              }
26            }
27          ]
28        }
29      ]
30    }
31  }
32 }
```

# GraphQL Full-stack type safety

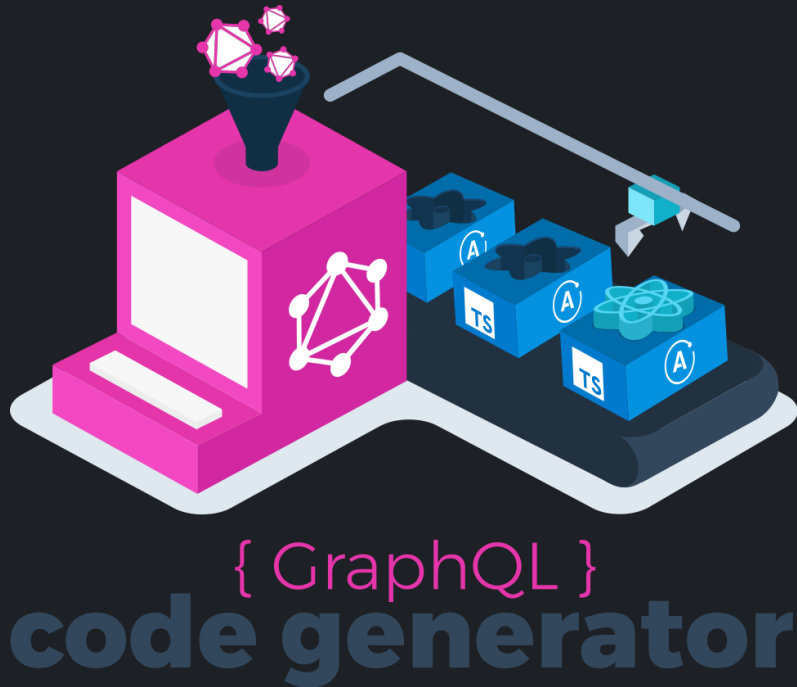
GraphQL in 2020





# GraphQL Full-stack type safety

GraphQL in 2020



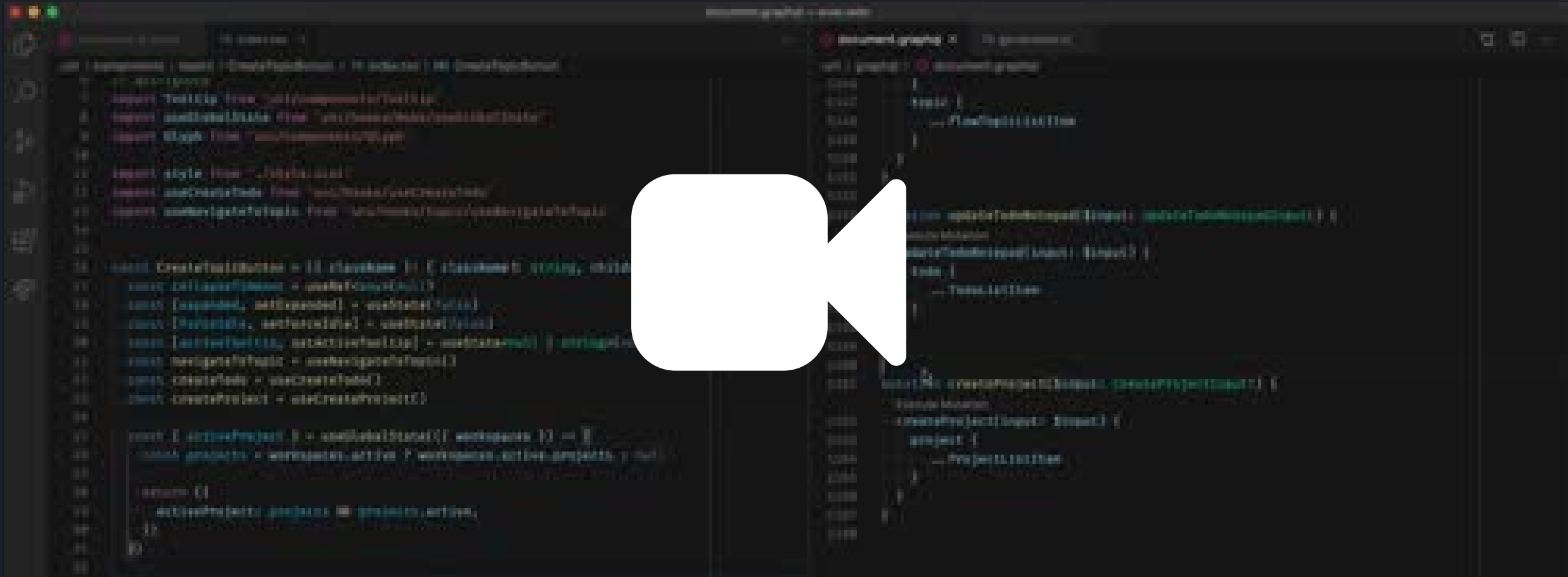
Given a GraphQL Schema, generates:

- TypeScript types definition
- React Apollo hooks definition
- urql components
- Types for Flow, Java, Kotlin

# GraphQL Full-stack type safety

GraphQL in 2020

Typed React hooks generations



# GraphQL Full-stack type safety

GraphQL in 2020

‘TypedDocumentNode’ for simplicity

```
1  /* eslint-disable */
2  import { RatesDocument } from '../typed-document-nodes';
3  import { useQuery } from '@apollo/client';
4
5  // We now have types support and auto complete for the
6  // result type, just by passing 'RatesDocument' as 'query' to apollo client
7  const result = useQuery(RatesDocument, {
8    variables: {
9      currency: "USD",
10    },
11  });
12
13  const rates = result.data.rates;
14  const currency = rates[0].rate;
15
```

*“ We now have 'cheap to maintain'  
best-in-class TypeScript types in our front-end*

# GraphQL (Beyond type safety intermission)

# GraphQL (Beyond type safety intermission)

GraphQL in 2020

Better front-end developer experience

- GraphQL without queries
- GraphQL forms

# GraphQL (Beyond type safety intermission)

GraphQL in 2020

## GraphQL without queries: gqlless

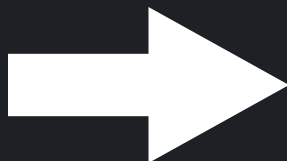
```
1  const User = graphql(({ user }: { user: User }) => (  
2    <div>  
3      <h2>{user.name}</h2>  
4      <img src={user.avatarUrl({ size: 100 })} />  
5    </div>  
6  ))  
7  
8  const App = graphql(() => (  
9    <div>  
10     {query.users.map(user => (  
11       <User key={user.id} user={user} />  
12     ))}  
13   </div>  
14 ))
```

# GraphQL (Beyond type safety intermission)

GraphQL in 2020

## GraphQL without queries: gqlless

```
1 const User = graphql(({ user }: { user: User }) => {
2   <div>
3     <h2>{user.name}</h2>
4     <img src={user.avatarUrl({ size: 100 })} />
5   </div>
6 })
7
8 const App = graphql(() => {
9   <div>
10     {query.users.map(user => (
11       <User key={user.id} user={user} />
12     ))}
13   </div>
14 })
```



```
1 query App {
2   users {
3     id
4     name
5     avatarUrl(size: 100)
6   }
7 }
```

+ TypeScript support



# GraphQL (Beyond type safety intermission)

GraphQL in 2020

## GraphQL forms: Frontier

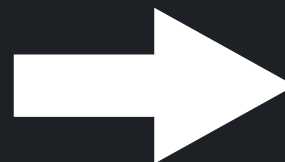
```
1 import gql from "graphql-tag";
2 import { Frontier } from "frontier-forms";
3 import { myApplicationKit } from "../uiKit";
4 import { client } from "../apollo-client";
5
6 const mutation = gql`
7   mutation($user: User!) {
8     createUser(user: $user) { id }
9   }
10 `;
11
12 <Frontier
13   client={client}
14   mutation={mutation}
15   uiKit={myApplicationKit}
16 />
```

# GraphQL (Beyond type safety intermission)

GraphQL in 2020

## GraphQL forms: Frontier

```
1 import gql from "graphql-tag";
2 import { Frontier } from "frontier-forms";
3 import { myApplicationKit } from "../uiKit";
4 import { client } from "../apollo-client";
5
6 const mutation = gql`
7   mutation($user: User!) {
8     createUser(user: $user) { id }
9   }
10 `;
11
12 <Frontier
13   client={client}
14   mutation={mutation}
15   uiKit={myApplicationKit}
16 />
```



### Create a user

Company name \*

E-mail \*

First name \*

Last name \*

Save



GraphQL bridging the gaps on server-side

# GraphQL bridging the gaps on server-side

GraphQL in 2020

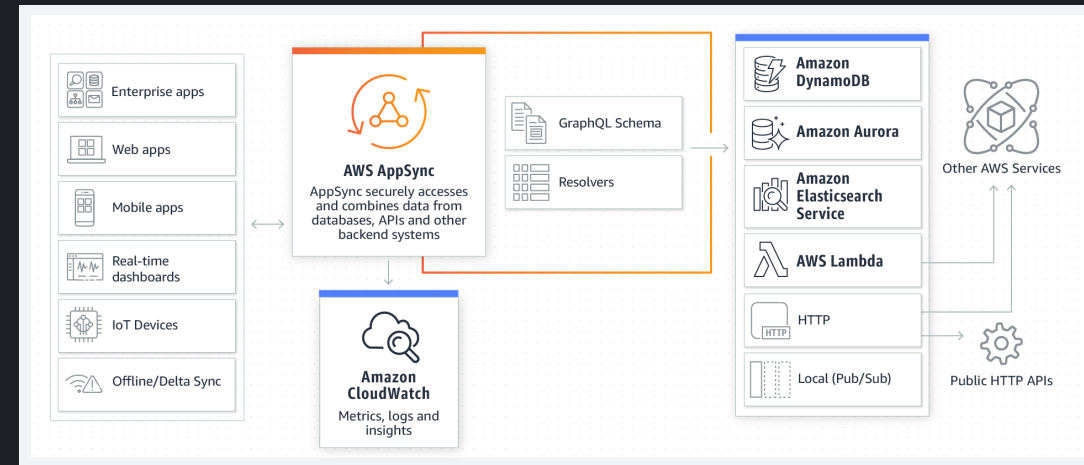
## Apollo



## Prisma



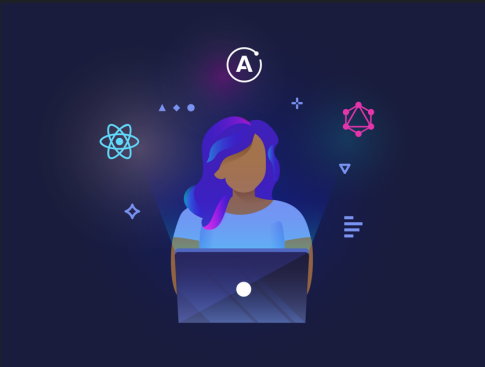
## AWS AppSync



# GraphQL bridging the gaps on server-side

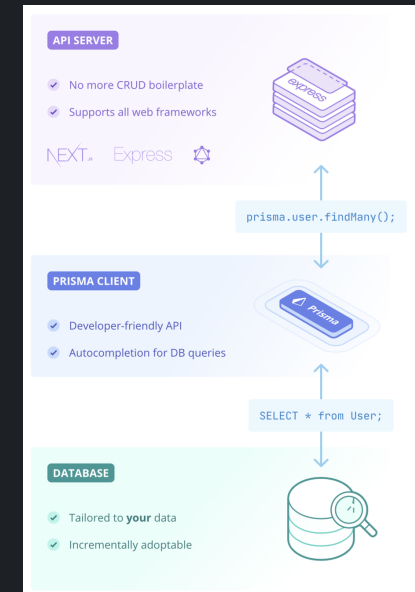
# GraphQL in 2020

# Apollo



- What about more complex GraphQL APIs without a from-scratch development?

# Prisma



# GraphQL bridging the gaps on server-side

GraphQL in 2020

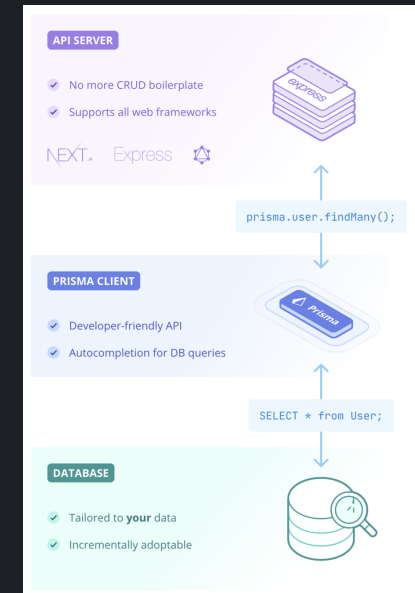
## Apollo



## Hasura



## Prisma



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## Hasura

Hasura translate GraphQL AST to SQL AST,  
providing blazing fast execution with  
minimum configuration

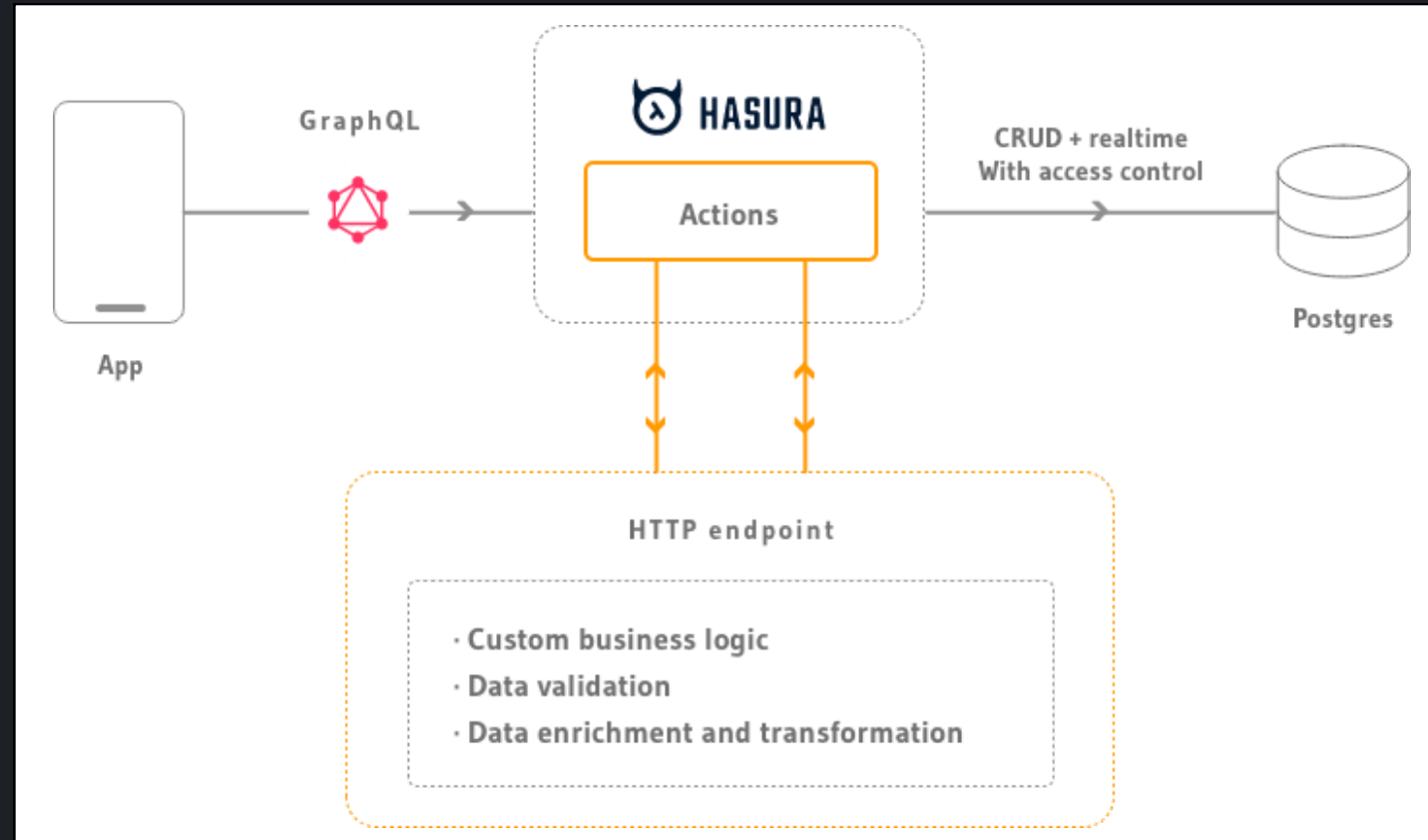


# GraphQL bridging the gaps on server-side

GraphQL in 2020

Hasura

Custom logic support via Actions





# GraphQL bridging the gaps on server-side

GraphQL in 2020

Hasura

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## Hasura

- ACL support

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## Hasura

- ACL support
- Authentication / Authorization (JWT)

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support

## Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support
- Actions for custom logic
- Trigger webhooks on database events

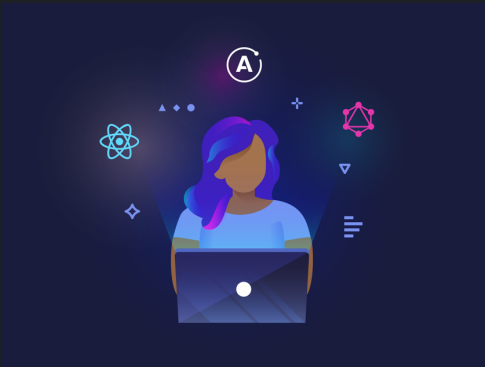
## Hasura

- ACL support
- Authentication / Authorization (JWT)
- Remote schemas support (schema stitching)
- Subscriptions support
- Actions for custom logic
- Trigger webhooks on database events
- Bonus: one-click install on most cloud providers

# GraphQL bridging the gaps on server-side

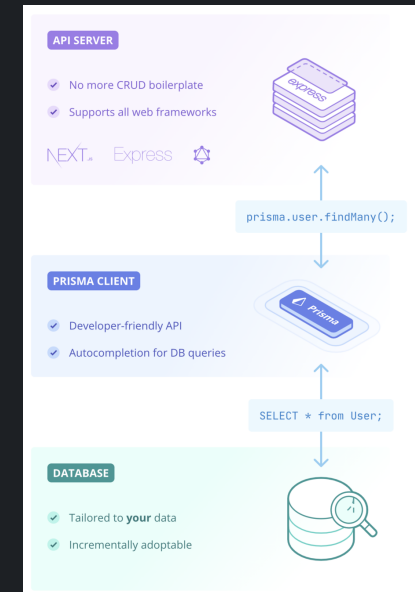
GraphQL in 2020

## Apollo



- What about old REST APIs and external services?

## Prisma

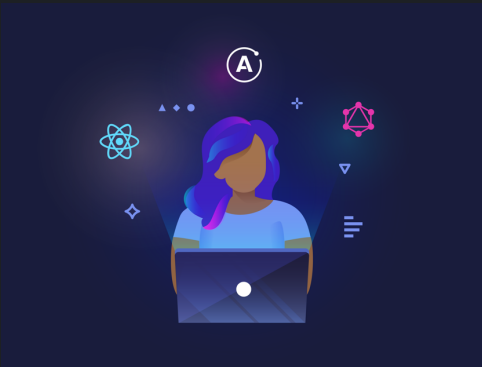




# GraphQL bridging the gaps on server-side

GraphQL in 2020

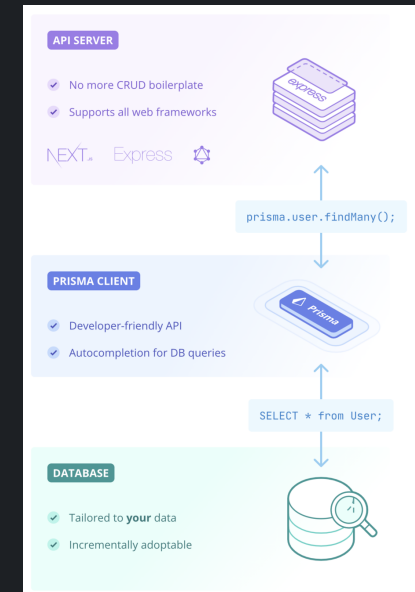
## Apollo



## GraphQL Mesh



## Prisma



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

*“ GraphQL Mesh leverages API definition standards to transform existing API to GraphQL APIs*



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

## Spotify GraphQL in 5'

The screenshot displays the Spotify GraphQL Console interface. On the left, a query is entered in a dark-themed editor:

```
1 {
2   me {
3     display_name
4     playlists(throttle: 100, continueOnError: 1, limit: -1) {
5       name
6     }
7   }
8 }
9
```

On the right, the JSON response is shown:

```
{
  "data": {
    "me": {
      "display_name": "Charly Poly",
      "playlists": [
        {
          "name": "Trajectoire"
        },
        {
          "name": "This Is N  pal"
        },
        {
          "name": "2Fingz"
        },
        {
          "name": "This Is Basshunter"
        },
        {
          "name": "Standards de la Chanson Fran  aise"
        },
        {
          "name": "This Is Masego"
        },
        {
          "name": "Reggeaton 2019"
        },
        {
          "name": "Max Richter - The Blue Notebooks (15 Years)"
        },
        {
          "name": "Arrival Soundtrack"
        },
        {
          "name": "40"
        },
        {
          "name": "Night walk   "
        },
        {
          "name": "Nights"
        },
        {
          "name": "CLASSIC RAMMERS"
        },
        {
          "name": "Pickup Music Weekly - R&B, Indie Pop & Folk | New Music | Pickup Music 2020"
        }
      ]
    }
  }
}
```

At the bottom right, a sidebar titled "Schema" lists the fields available in the schema:

- me: PrivateUser
- user(id: String!): PublicUser
- track(id: String, name: String): Track
- tracks(ids: String!): [Track]
- audio\_features(trackId: String!): [AudioFeatures]
- audio\_feature(trackId: String!): AudioFeatures
- artist(id: String, name: String): Artist
- artists(ids: String!): [Artist]
- album(id: String!): Album
- albums(ids: String!): [Album]
- playlist(id: String!, userId: String!): Playlist

At the bottom of the console, there is a footer with the Spotify logo and the text: "Powered with Spotify | [Open source code](#) | [About us](#)".

# GraphQL bridging the gaps on server-side

GraphQL in 2020

GraphQL Mesh

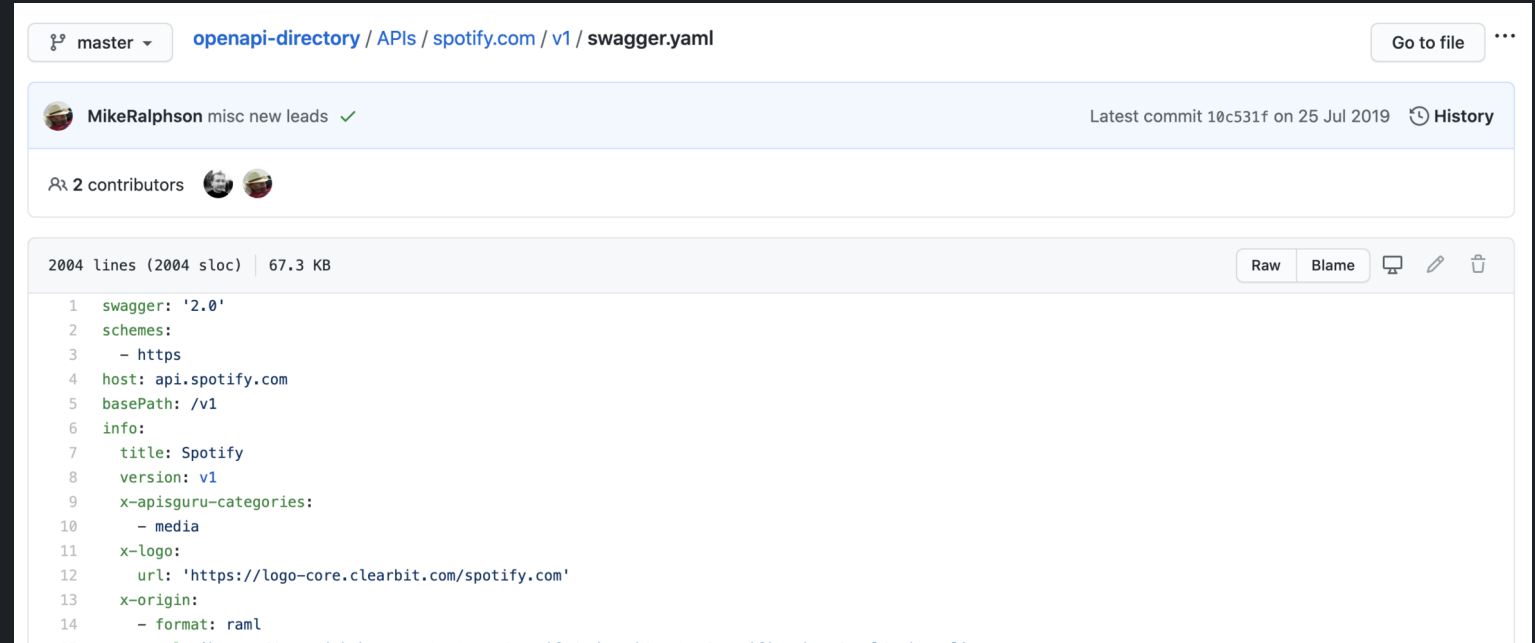
Spotify GraphQL in 5'

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

## Spotify GraphQL in 5'



The screenshot shows a GitHub repository page for 'openapi-directory' with the file path 'APIs / spotify.com / v1 / swagger.yaml'. The file is owned by 'MikeRalphson' and has a latest commit '10c531f' on '25 Jul 2019'. It has 2 contributors and 2004 lines of code (2004 sloc) with a size of 67.3 KB. The code is a Swagger 2.0 specification for the Spotify API.

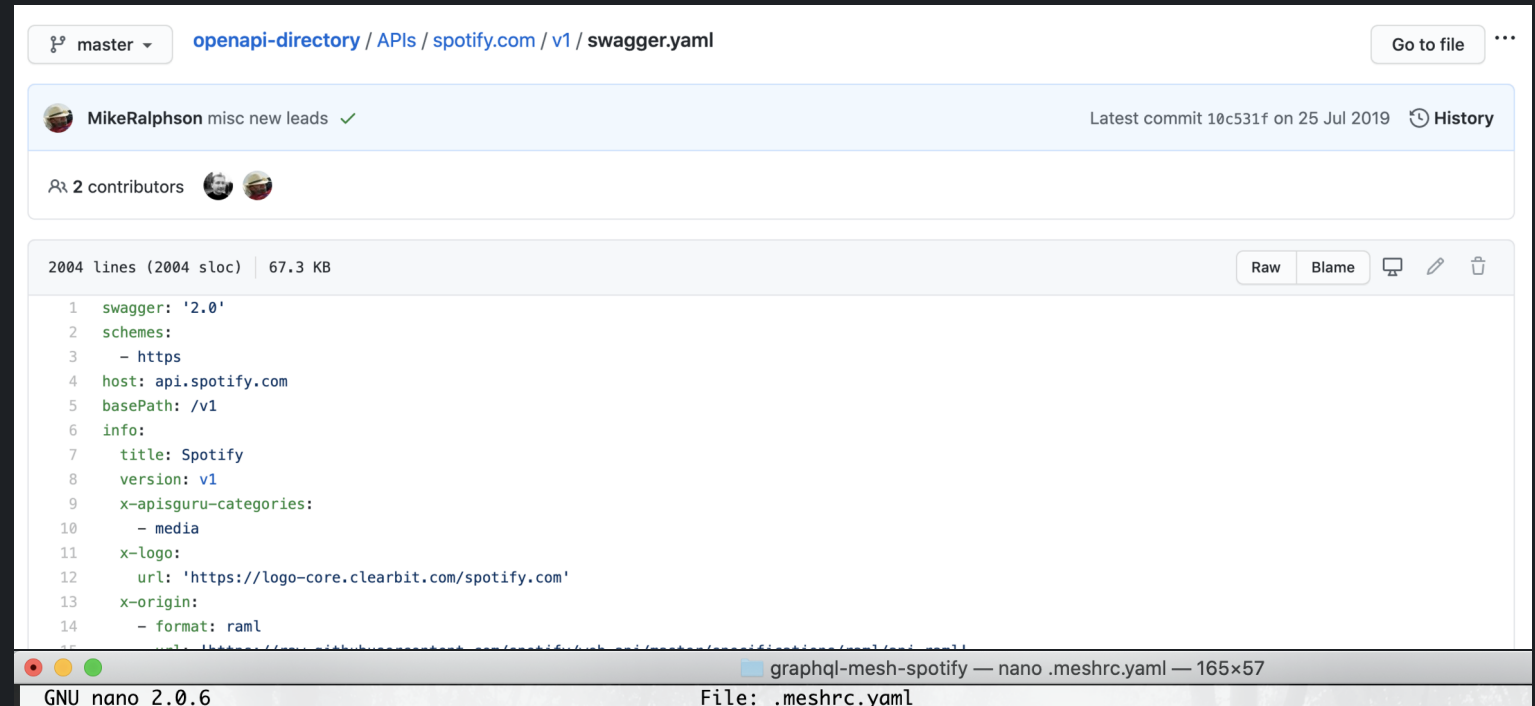
```
1  swagger: '2.0'
2  schemes:
3    - https
4  host: api.spotify.com
5  basePath: /v1
6  info:
7    title: Spotify
8    version: v1
9  x-apisguru-categories:
10    - media
11  x-logo:
12    url: 'https://logo-core.clearbit.com/spotify.com'
13  x-origin:
14    - format: raml
```

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

## Spotify GraphQL in 5'



The screenshot shows a GitHub repository for `openapi-directory` at the path `APIs / spotify.com / v1 / swagger.yaml`. The repository is owned by `MikeRalphson` and has 2 contributors. The latest commit is `10c531f` on 25 Jul 2019. The file `swagger.yaml` is 2004 lines (2004 sloc) and 67.3 KB. The file content is as follows:

```
1  swagger: '2.0'
2  schemes:
3    - https
4  host: api.spotify.com
5  basePath: /v1
6  info:
7    title: Spotify
8    version: v1
9  x-apisguru-categories:
10    - media
11  x-logo:
12    url: 'https://logo-core.clearbit.com/spotify.com'
13  x-origin:
14    - format: raml
```

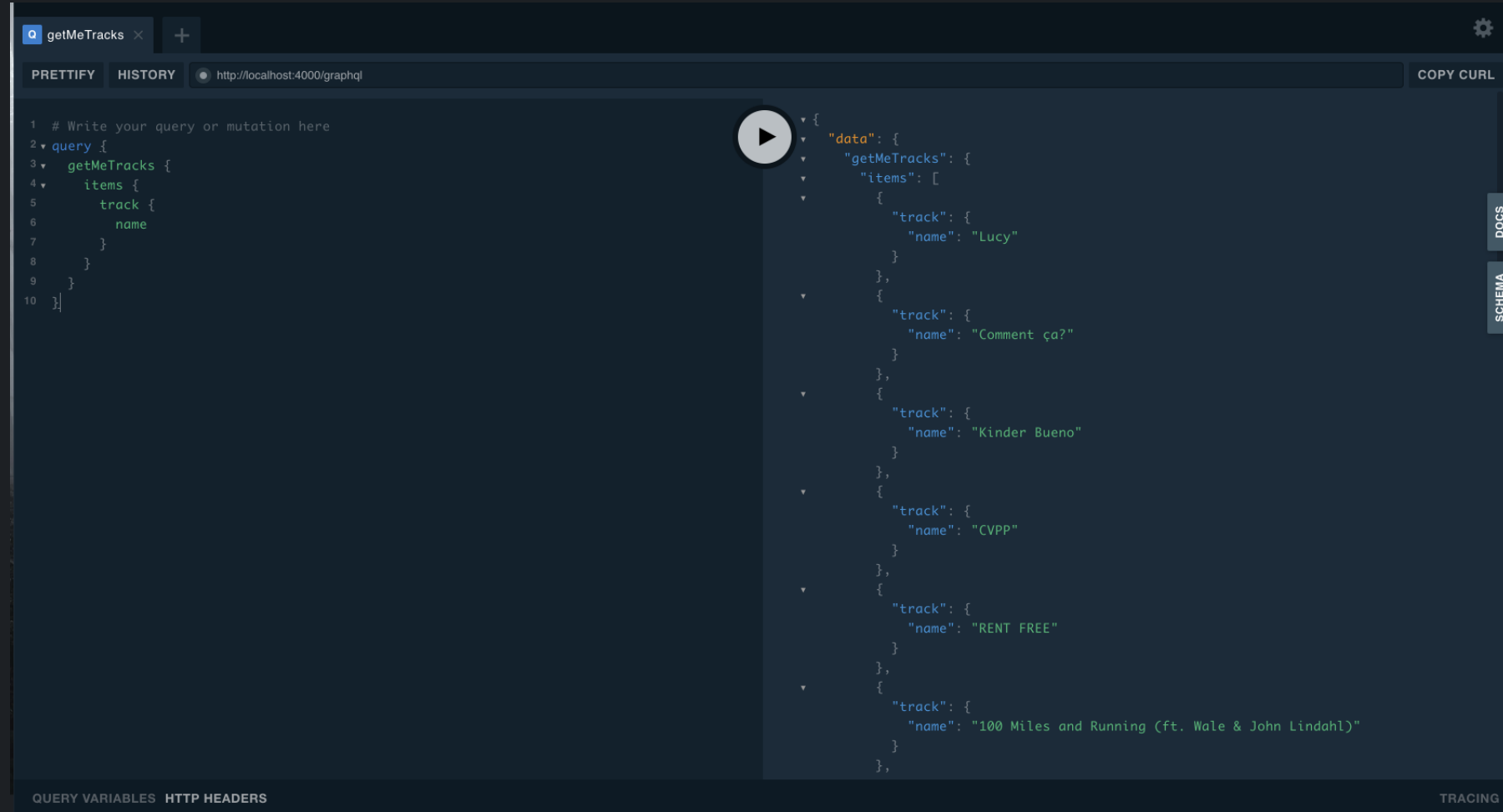
```
sources:
- name: Spotify
  handler:
    openapi:
      source: https://raw.githubusercontent.com/APIs-guru/openapi-directory/master/APIs/spotify.com/v1/swagger.yaml
      operationHeaders:
        Authorization: █
```

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

## Spotify GraphQL in 5'



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

- GraphQL proxy over your data

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

## GraphQL Mesh

- GraphQL proxy over your data
- Schema transformation
- Resolvers composition
- Caching
- SDK generation

*“ GraphQL Mesh doesn’t aim to create [...] public GraphQL schema [...], you should consider implementing another layer that exposes your public data [...].*

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL API building in 2020



# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions
- Hasura helps you build performant and powerful API

# GraphQL bridging the gaps on server-side

GraphQL in 2020

## GraphQL API building in 2020

- Apollo, Prisma, and AWS AppSync are not unique solutions
- Hasura helps you build performant and powerful API
- GraphQL Mesh brings you universal GraphQL

# GraphQL in 2020 ⚡

Wrap-up

# GraphQL in 2020 ⚡

GraphQL in 2020

Beyond a "Facebook & Apollo technology"

- The Guild
- Hasura
- Dgraph
- and many more individual contributors!

## Beyond fetching data from the client-side

- Better front-end developer experience (Full-stack type safety, ...)
- Easier complex GraphQL API building (Hasura)
- Fast GraphQL Database (Dgraph, Hasura)
- GraphQL as a universal data language (state, GraphQL Mesh)

# GraphQL: Looking forward

GraphQL in 2020

## GraphQL in 2021?

- Subscriptions / Async evolutions
  - Apollo 3, Hasura
  - @defer, @live
- Smarter client-side cache?
- Better mobile clients ecosystem?
- more public APIs?

# Thank you!

**n** slides on [noti.st/charlypoly](https://noti.st/charlypoly)

 @whereischarly

 @wittydeveloper

