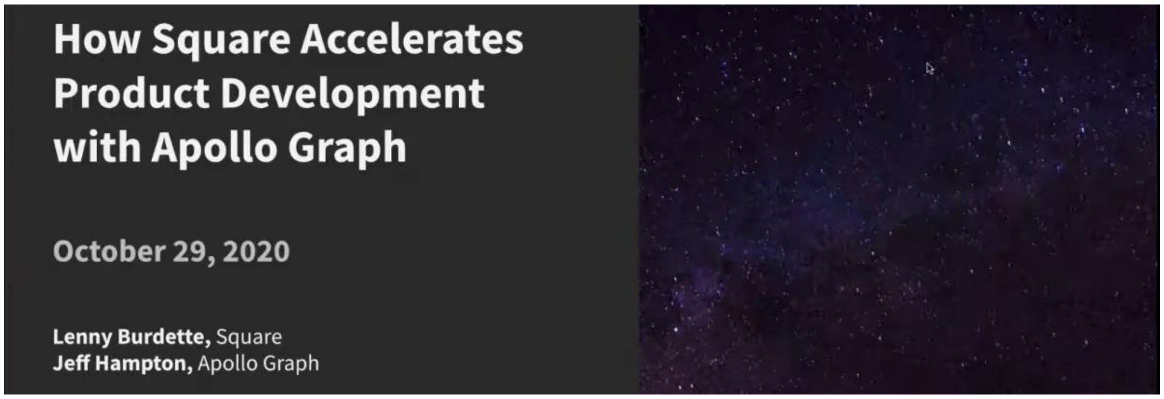
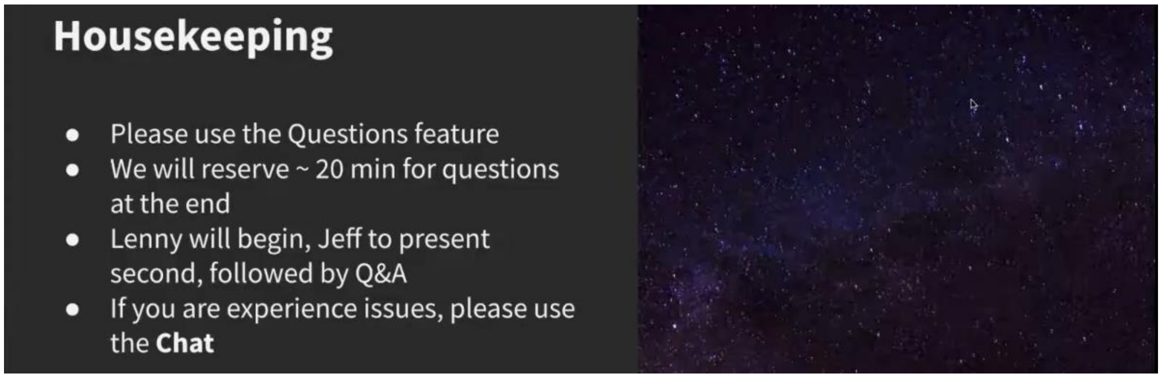


How Square Accelerates Product Development with Apollo Graph

October 29, 2020

Lenny Burdette, Square
Jeff Hampton, Apollo Graph

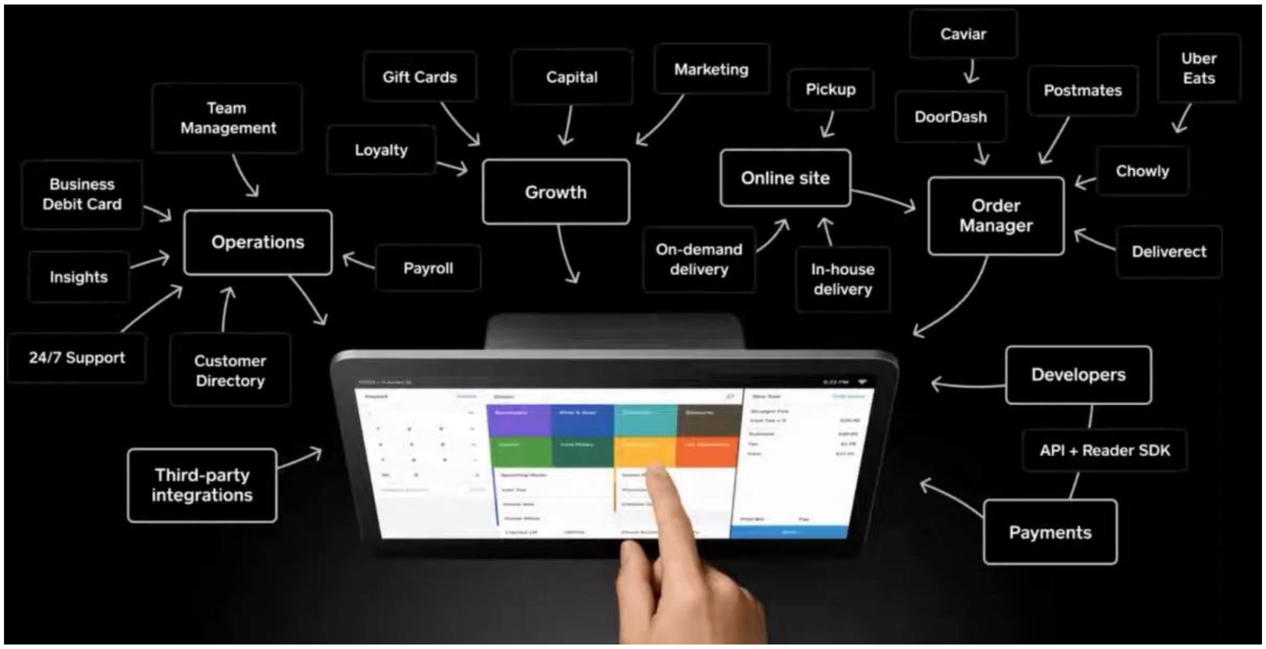


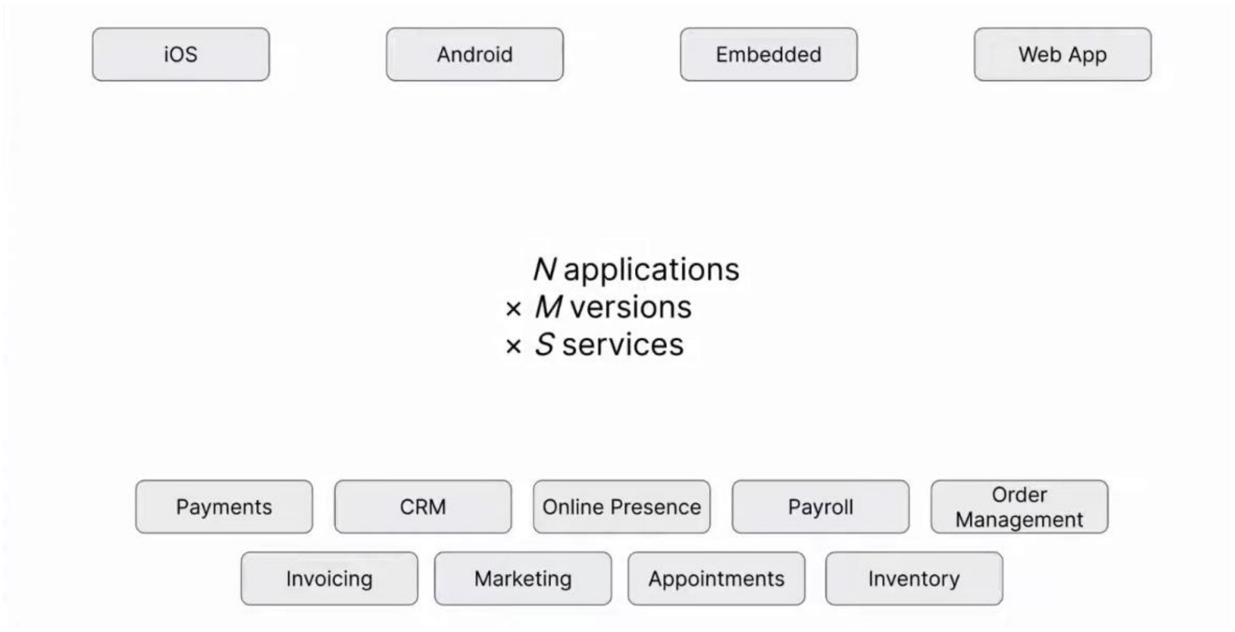
Housekeeping

- Please use the Questions feature
- We will reserve ~ 20 min for questions at the end
- Lenny will begin, Jeff to present second, followed by Q&A
- If you are experience issues, please use the **Chat**

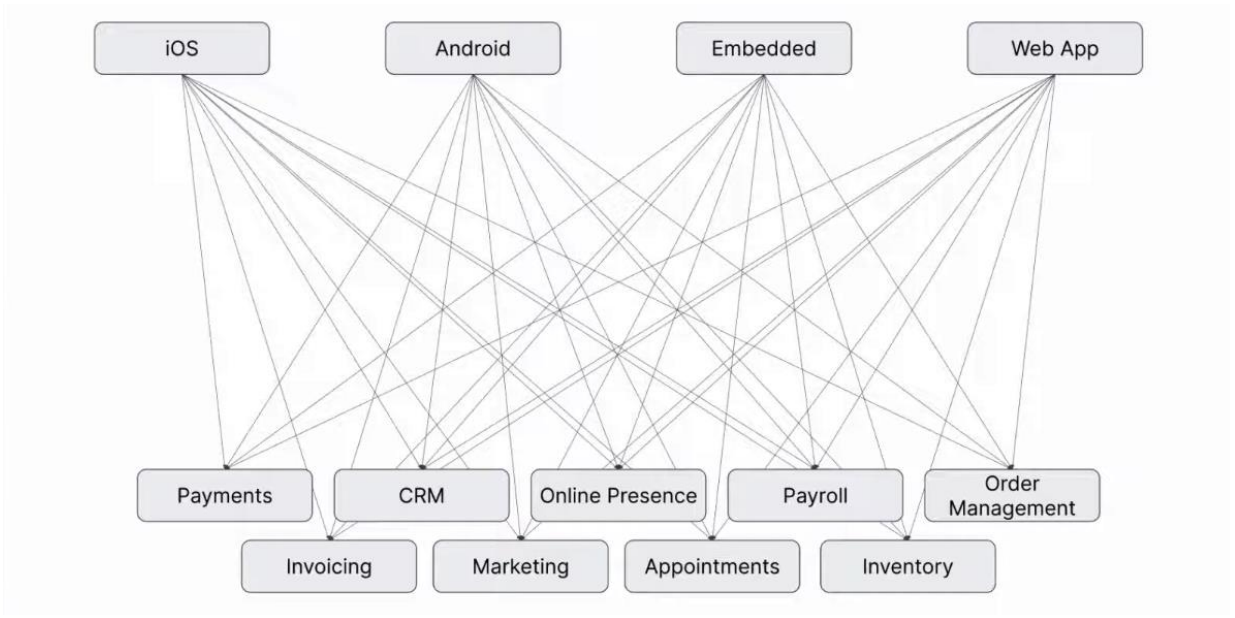
GraphQL at Square

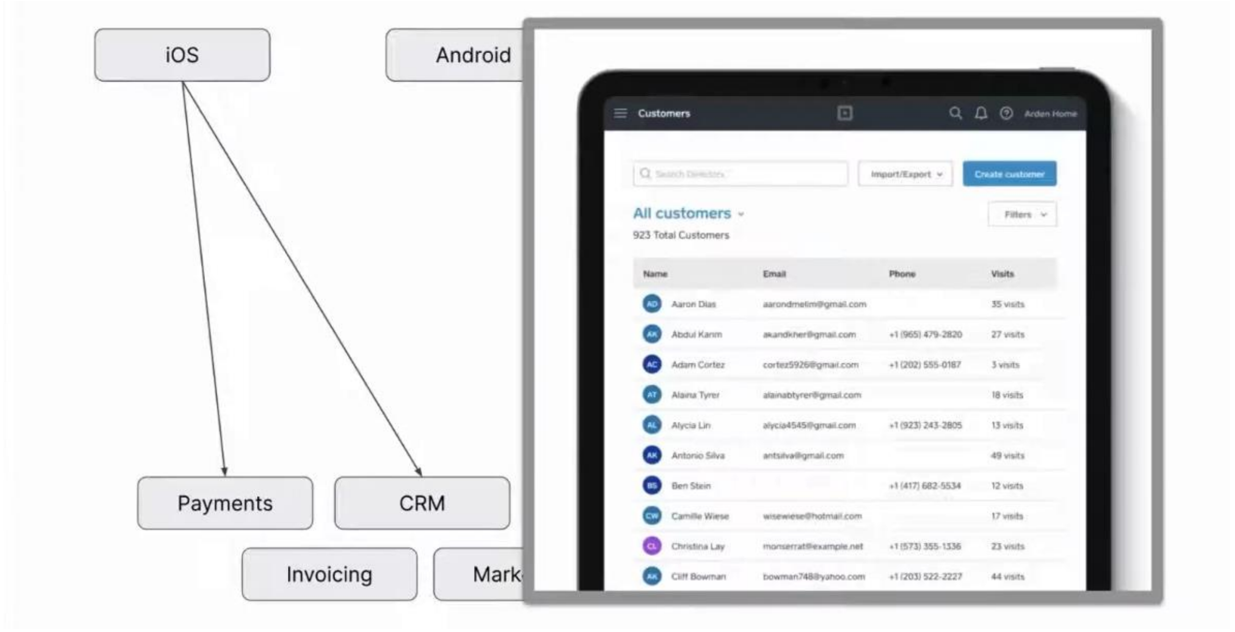
October 29, 2020

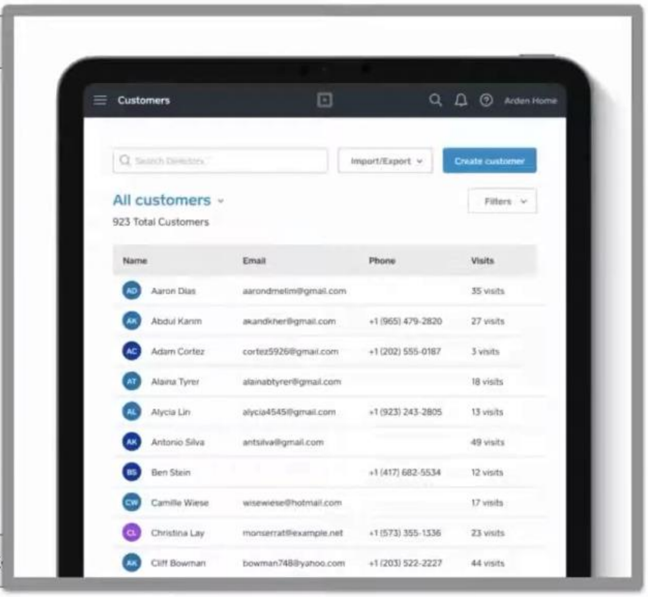




N applications
× M versions
× S services



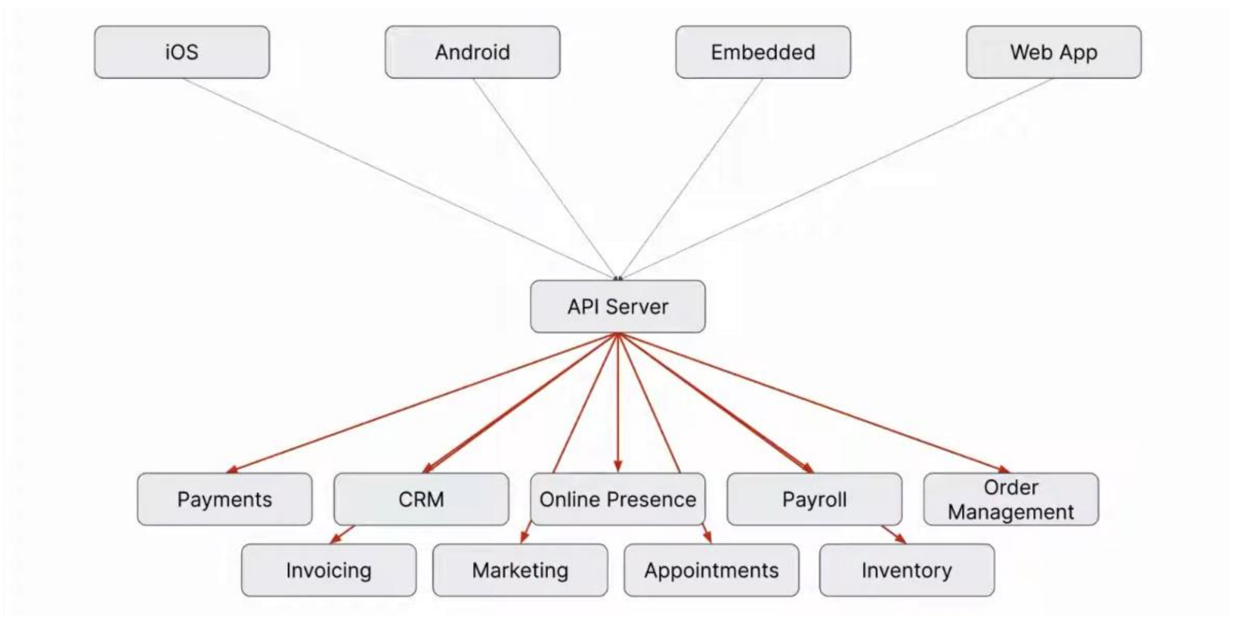


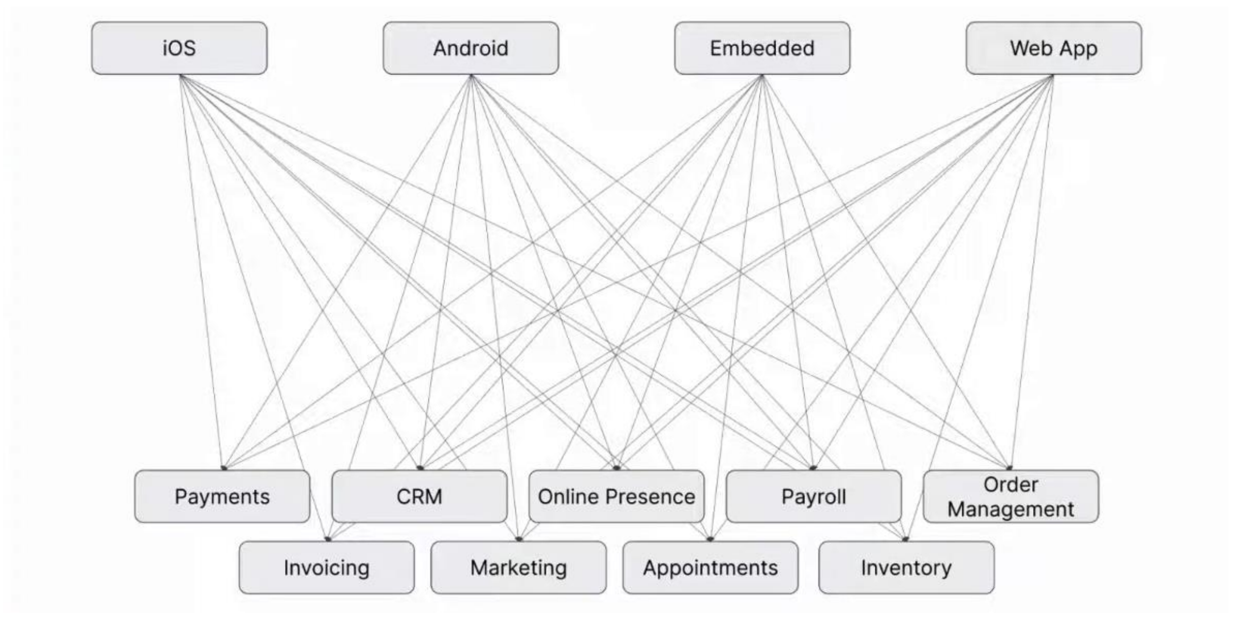


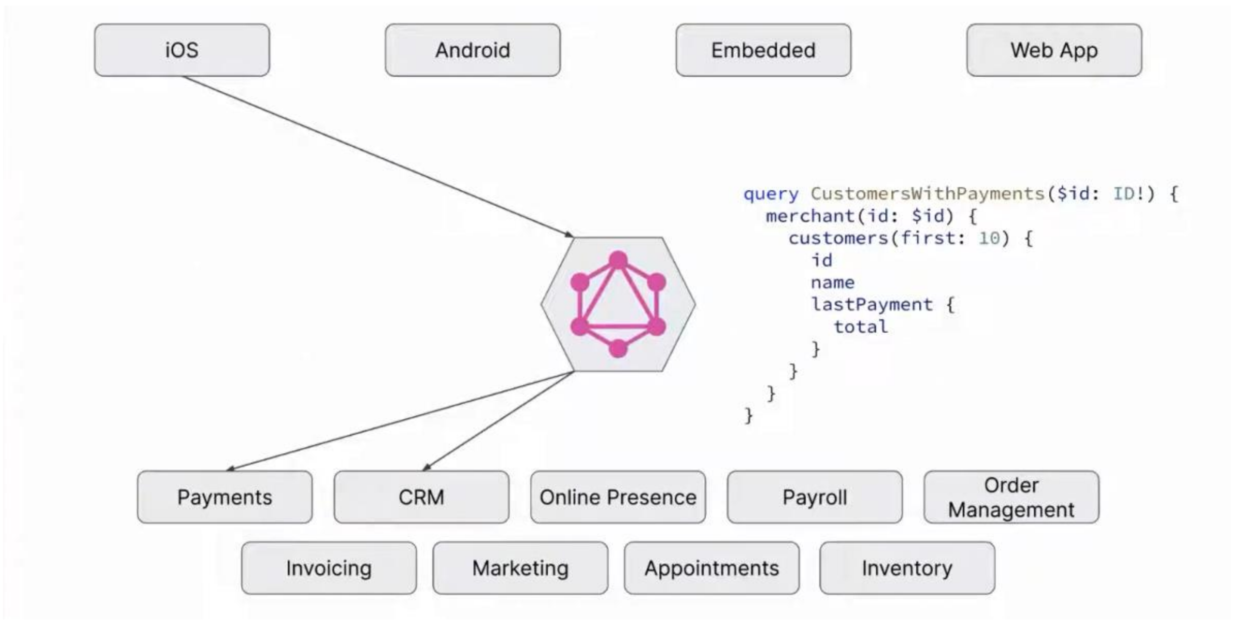
All customers

923 Total Customers

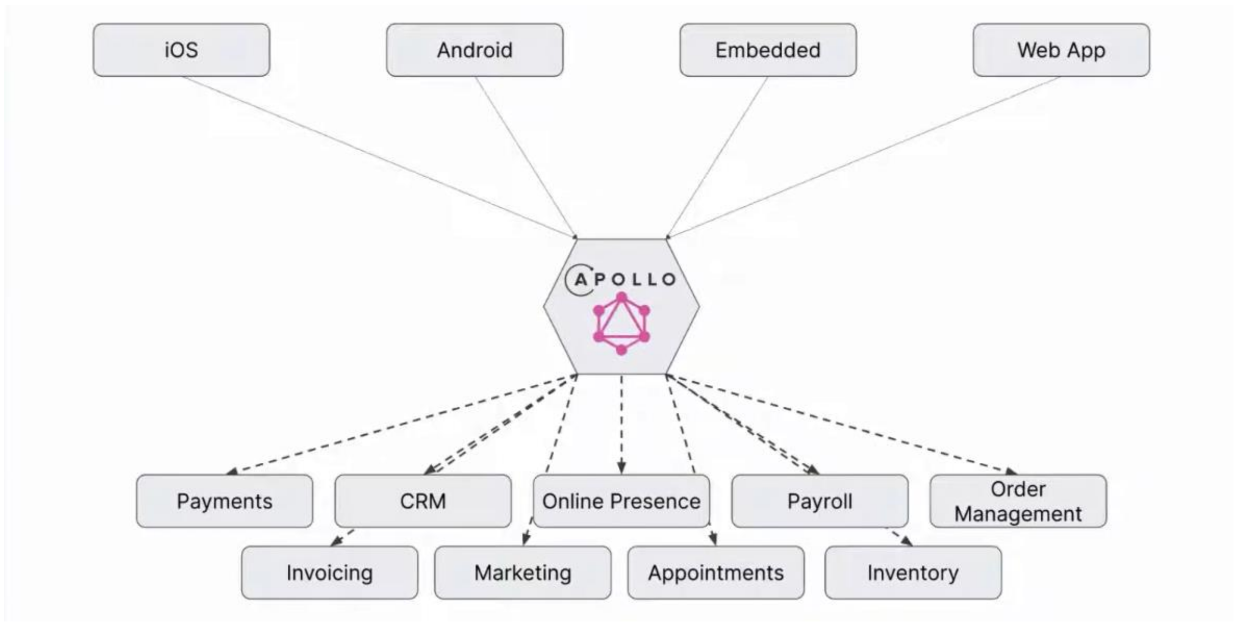
Name	Email	Phone	Visits
Aaron Dias	aarondmelim@gmail.com		35 visits
Abdul Karim	akandkher@gmail.com	+1 (965) 479-2820	27 visits
Adam Cortez	cortez5926@gmail.com	+1 (202) 555-0187	3 visits
Alaina Tyrer	alainabtyrer@gmail.com		18 visits
Alycia Lin	alycia4545@gmail.com	+1 (923) 243-2805	13 visits
Antonio Silva	antsilva@gmail.com		49 visits
Ben Stein		+1 (417) 682-5534	12 visits
Camille Wiese	wisewiese@hotmail.com		17 visits
Christina Lay	monserrat@example.net	+1 (573) 355-1336	23 visits
Cliff Bowman	bowman748@yahoo.com	+1 (203) 522-2227	44 visits

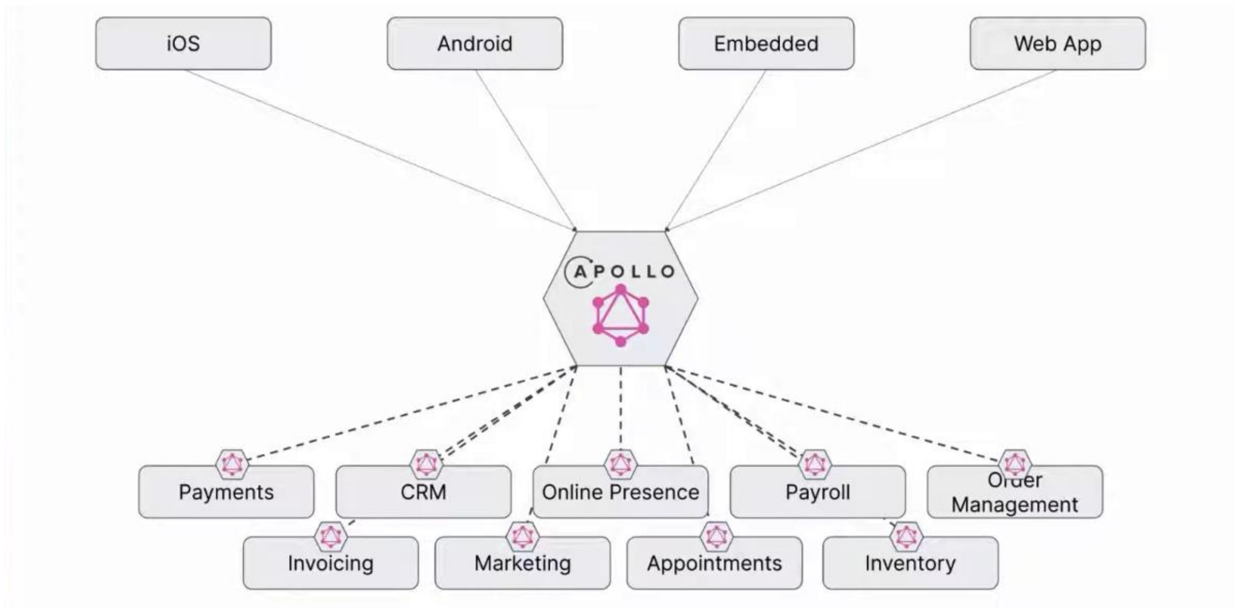


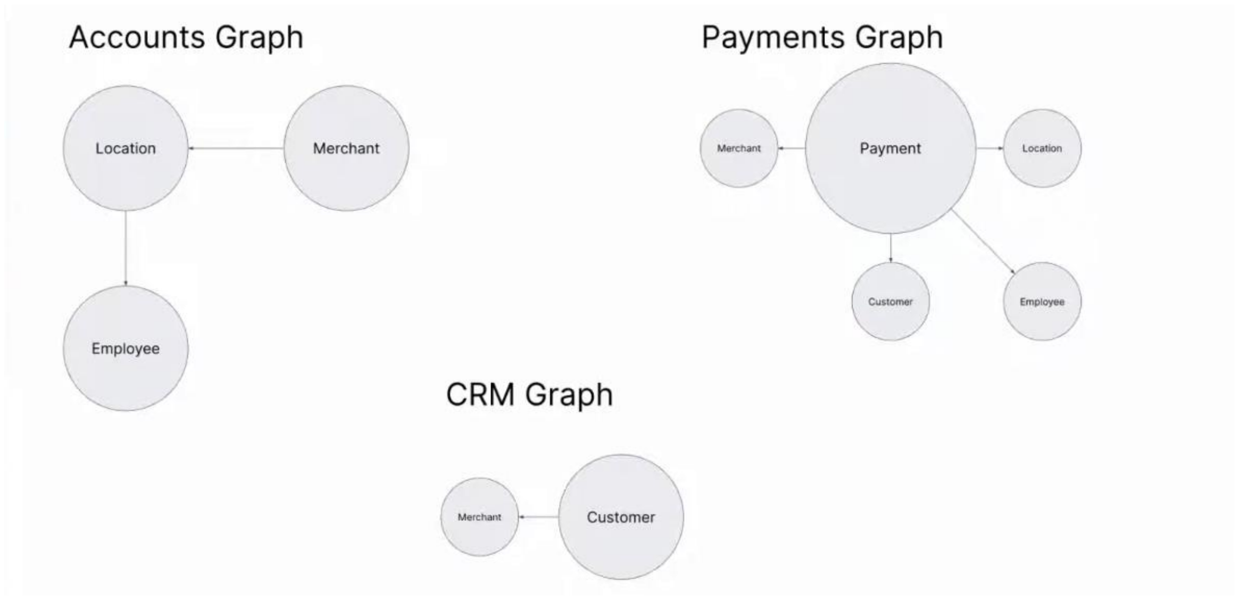




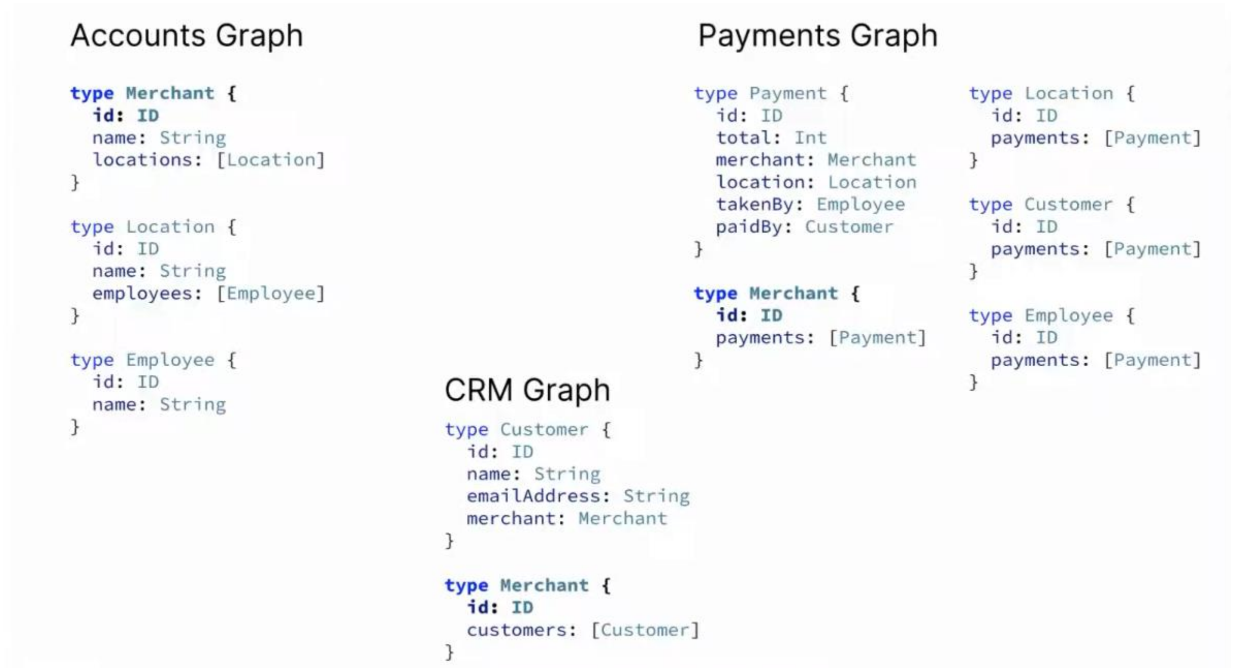
```
query CustomersWithPayments($id: ID!) {
  merchant(id: $id) {
    customers(first: 10) {
      id
      name
      lastPayment {
        total
      }
    }
  }
}
```







Each group develops GQL graph/services for the data relevant to their domain along with the relationships



Accounts Graph

```
type Merchant {
  id: ID
  name: String
  locations: [Location]
}

type Location {
  id: ID
  name: String
  employees: [Employee]
}

type Employee {
  id: ID
  name: String
}
```

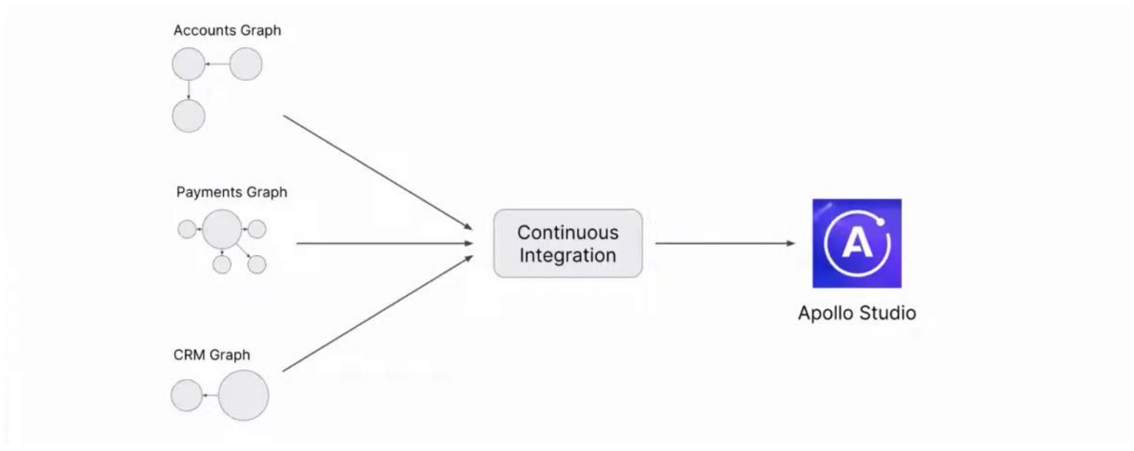
Payments Graph

```
type Payment {
  id: ID
  total: Int
  merchant: Merchant
  location: Location
  takenBy: Employee
  paidBy: Customer
}

type Merchant {
  id: ID
  payments: [Payment]
}

type Location {
  id: ID
  payments: [Payment]
}

type Customer {
  id: ID
  payments: [Payment]
}

type Employee {
  id: ID
  payments: [Payment]
}
```

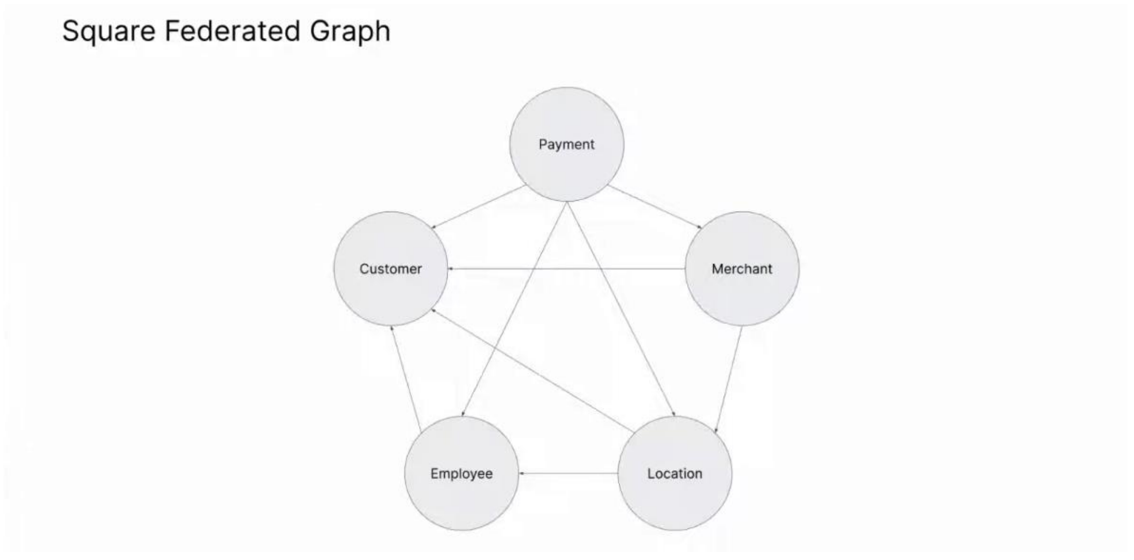
CRM Graph

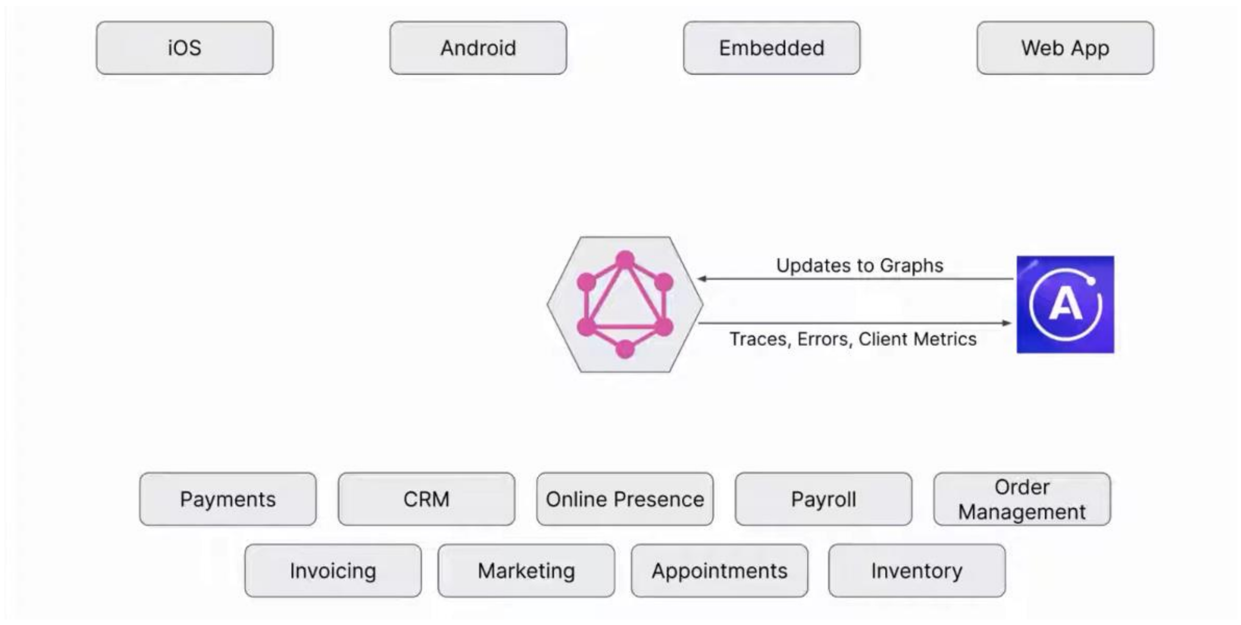
```
type Customer {
  id: ID
  name: String
  emailAddress: String
  merchant: Merchant
}

type Merchant {
  id: ID
  customers: [Customer]
}
```

What we want to do is to combine these individual Merchant types into a single richly-defined type exposed to the FEs via the gateway



We automate this composition in our CI system. Every time a subgraph changes, the change is sent to Apollo Studio for composition and validation. Apollo Studio then creates a unified federated schema/graph.





Frontend engineers want a unified schema

Allowing us to ship features faster, maintain them over time, and create consistency across platforms

Centralized, monolithic API services are difficult to scale

Especially in distributed, polyglot companies

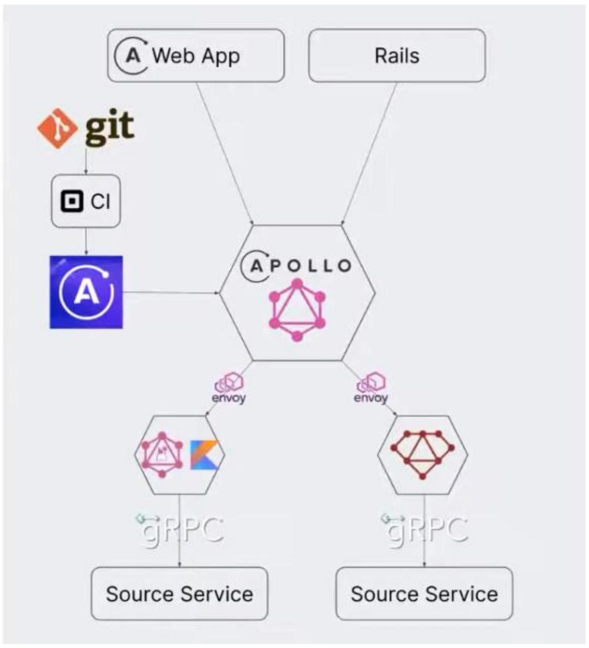
Federation makes it easier to run an automated, centralized API service

Apollo's schema management tools make federation possible

Integration with CI and production systems is key to successfully operating a federated architecture

Technical Details

- **Web app:** Ember / React with Apollo Client
- **Rails app:** graphql gem
- **Gateway:** Node.js app with @apollo/gateway
  - Some custom code to integrate with our Envoy service mech
- **Federated graphs:**
  - graphql-java + graphql-kotlin
  - graphql-ruby
- **Source services:** Java, Ruby, and Go with gRPC APIs
- **Continuous integration:** proprietary system
- **Managed federation:** Apollo Studio



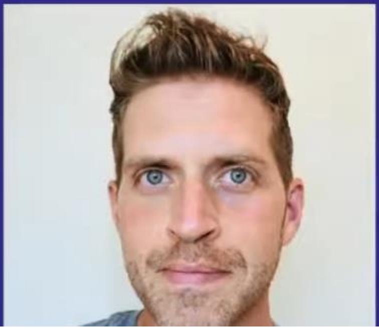
Thank you

Apollo GraphQL

Enterprise Federation

October 2020

Jeff Hampton | Director, Solutions Engineering



- Enterprise market fit, field engineering
- Success engineering
- Enterprise databases, SOLID design
- React Native, Apollo Server contributor
- **Architectural guidance**
- **GraphQL-as-abstraction**
- **Tooling and ergonomics**
- **Federation, Query Planning**
- **One Graph design**

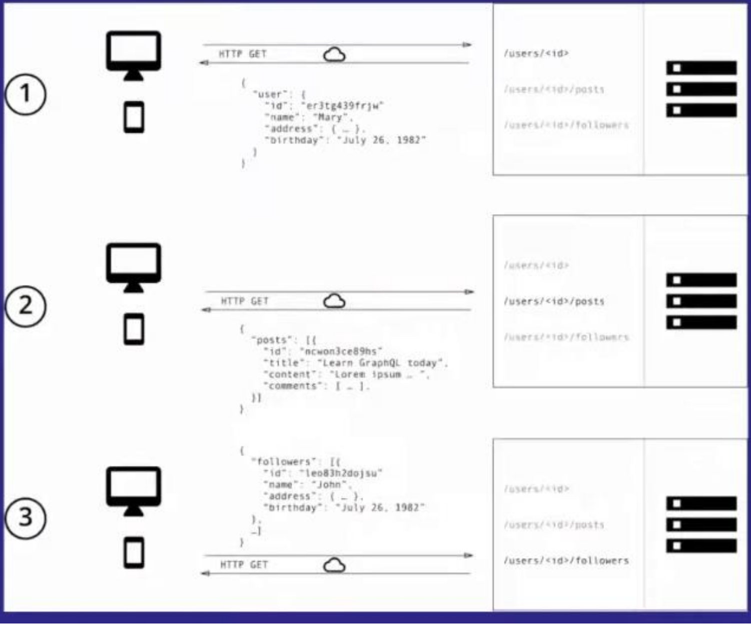
Topics

- GraphQL
- Apollo Platform
- Enterprise-Scale Challenges
- Consolidation using Apollo Federation
- Common Architectures, Workflows, Patterns

Apollo's Perspective

The Apollo Platform

GraphQL vs [REST, RPC, etc]



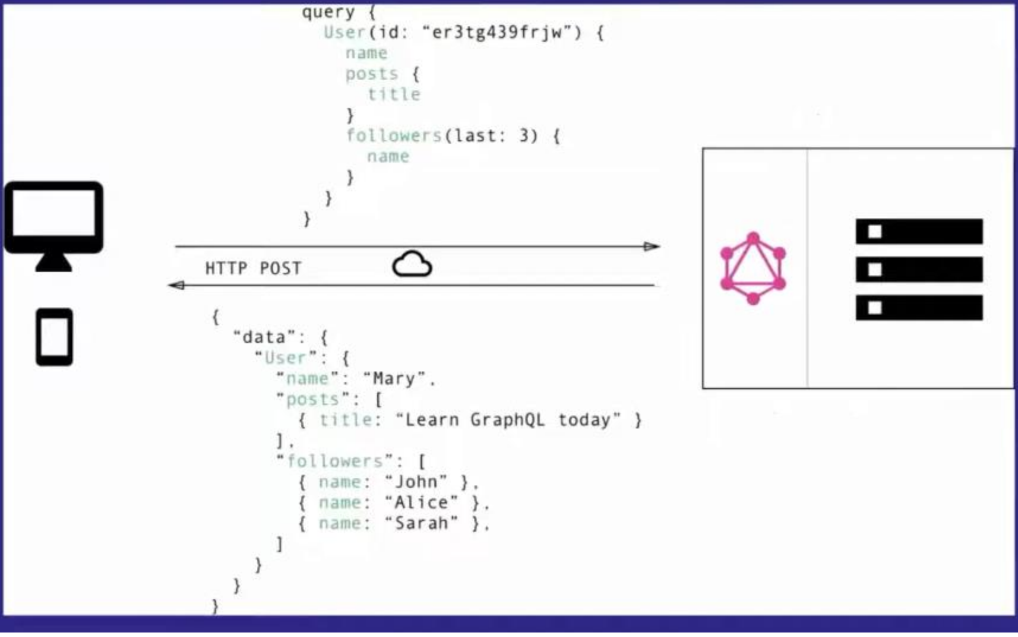
```
{
  "user": {
    "id": "er3tg439frjw"
    "name": "Mary",
    "address": { … },
    "birthday": "July 26, 1982"
  }
}
```

```
{
  "posts": [{
    "id": "ncwon3ce89hs"
    "title": "Learn GraphQL today",
    "content": "Lorem ipsum … ",
    "comments": [ … ],
  }]
}
```

```
{
  "followers": [{
    "id": "leo83h2dojsu"
    "name": "John",
    "address": { … },
    "birthday": "July 26, 1982"
  },
  …]
}
```

APOLLO

GraphQL vs [REST, RPC, etc]



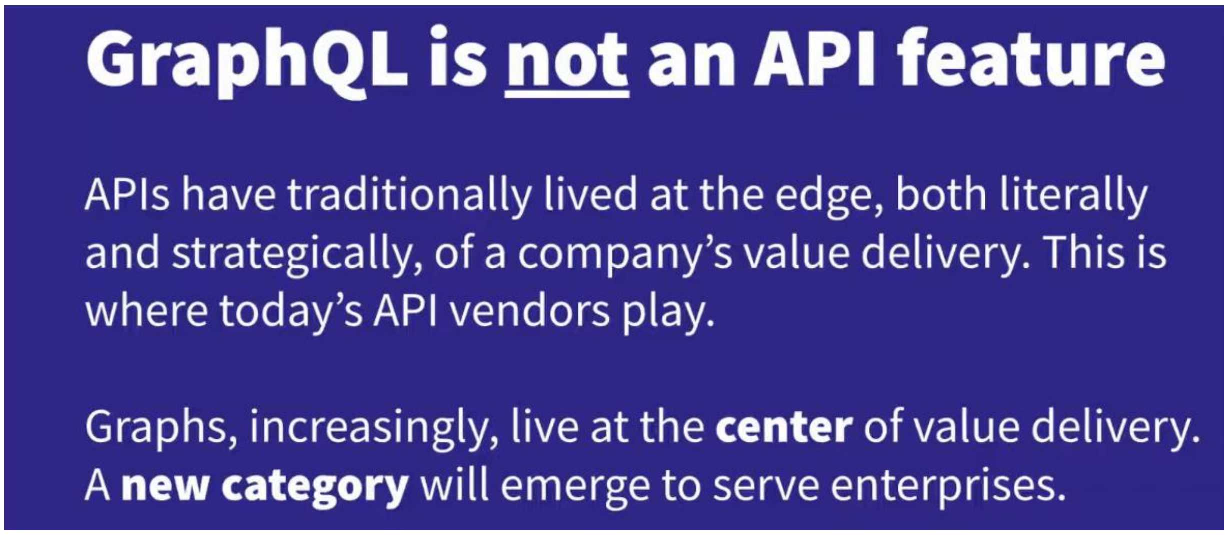
```
query {
  User(id: "er3tg439frjw") {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

```
{
  "data": {
    "User": {
      "name": "Mary",
      "posts": [
        { title: "Learn GraphQL today" }
      ],
      "followers": [
        { name: "John" },
        { name: "Alice" },
        { name: "Sarah" },
      ]
    }
  }
}
```

APOLLO

GraphQL is...

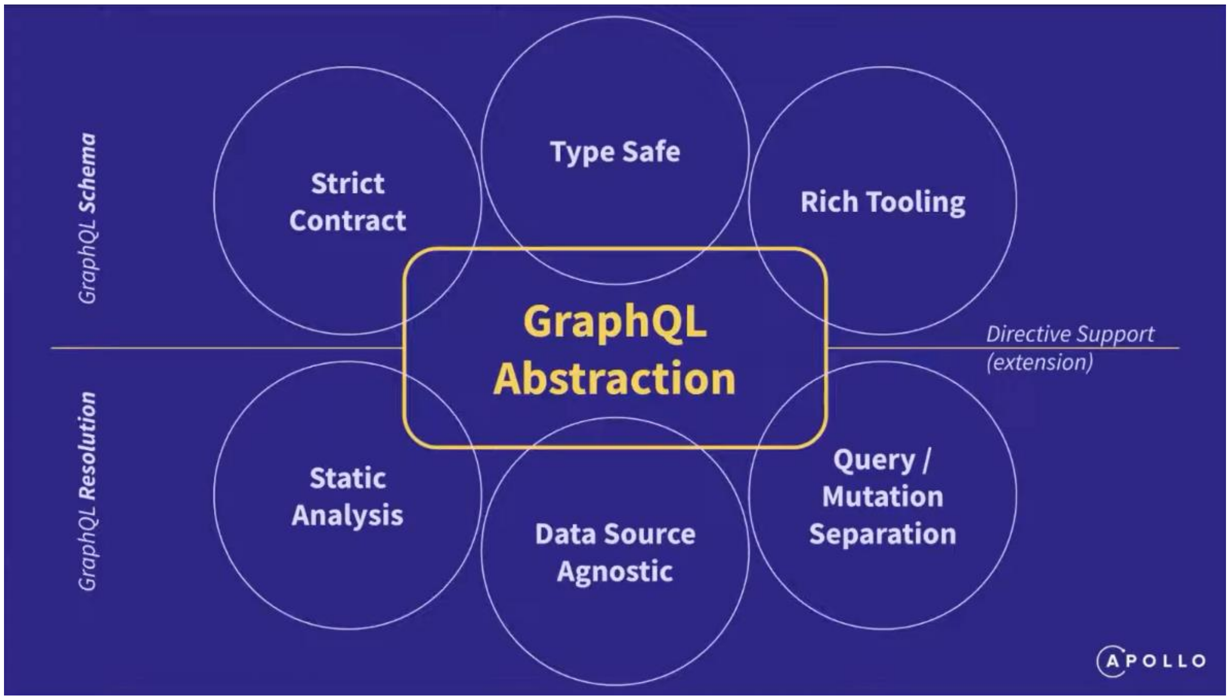
- An **aggregation** layer?
- An **orchestration** layer?
- Front-end **affordance**?

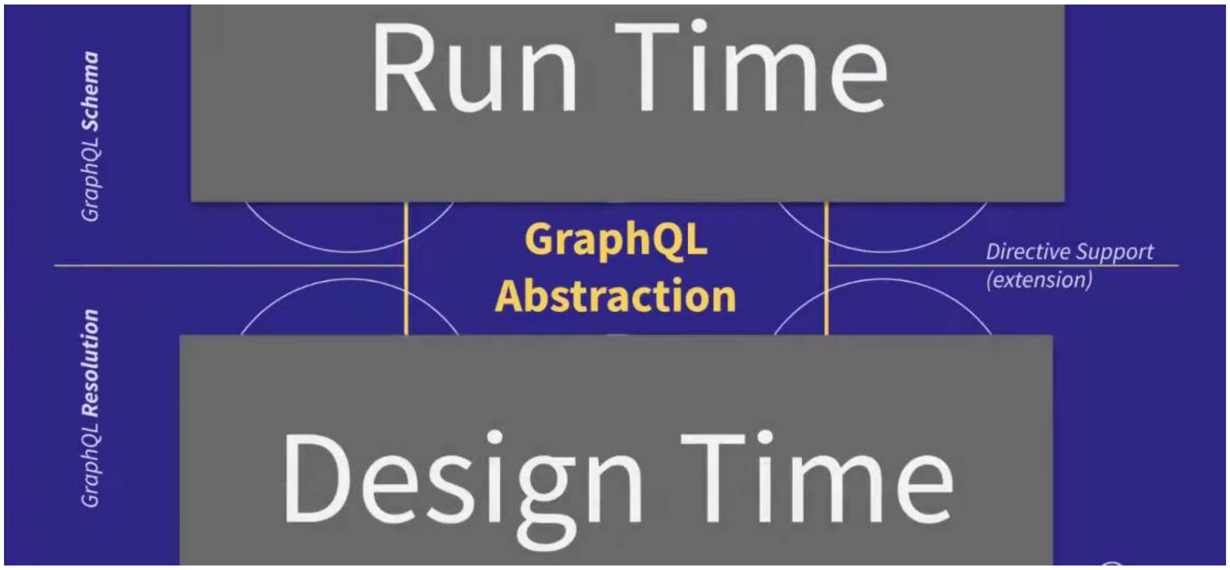


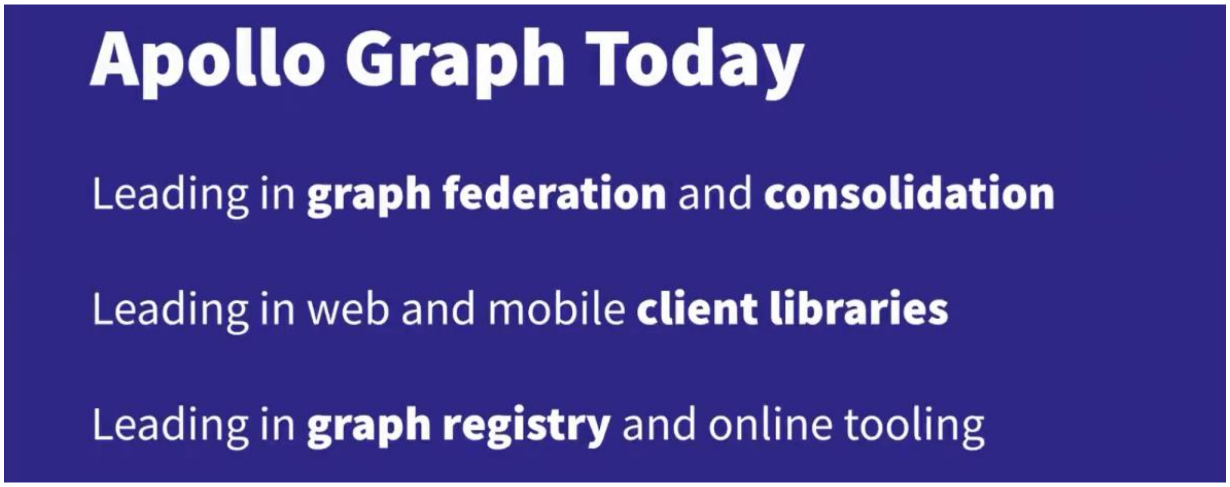
GraphQL is not an API feature

APIs have traditionally lived at the edge, both literally and strategically, of a company's value delivery. This is where today's API vendors play.

Graphs, increasingly, live at the **center** of value delivery. A **new category** will emerge to serve enterprises.







Apollo Graph Today

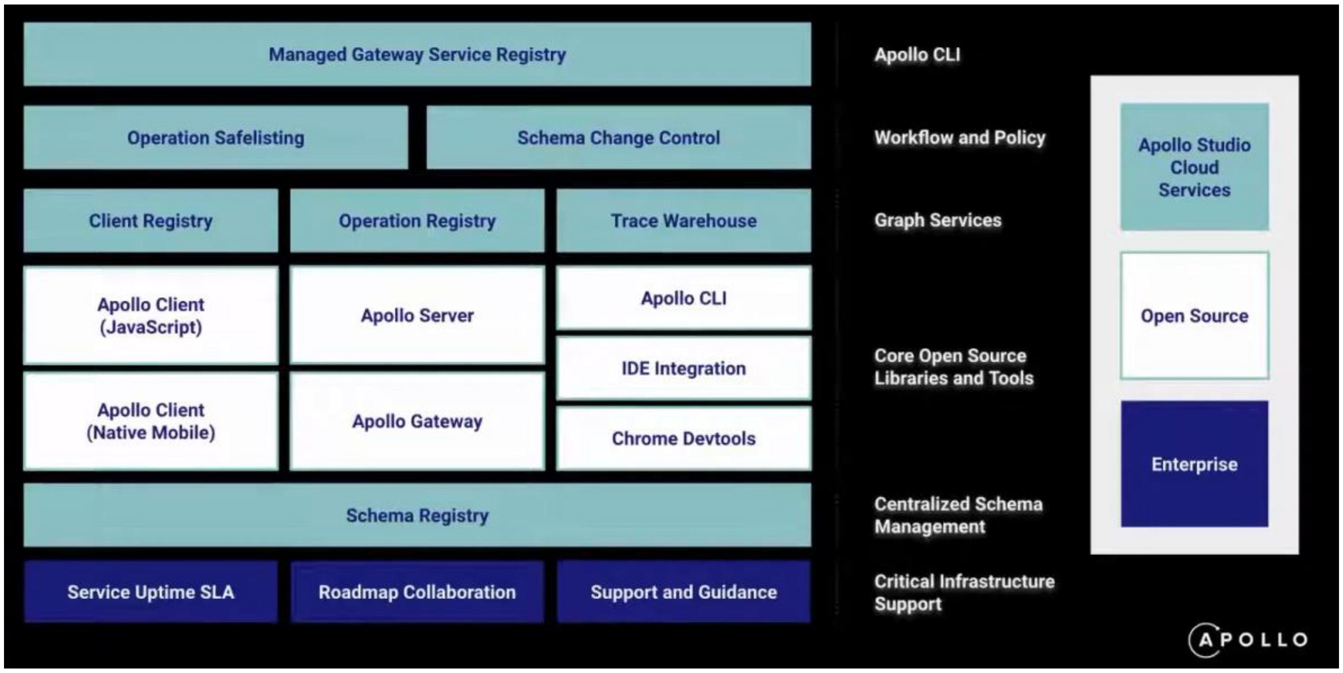
Leading in **graph federation** and **consolidation**

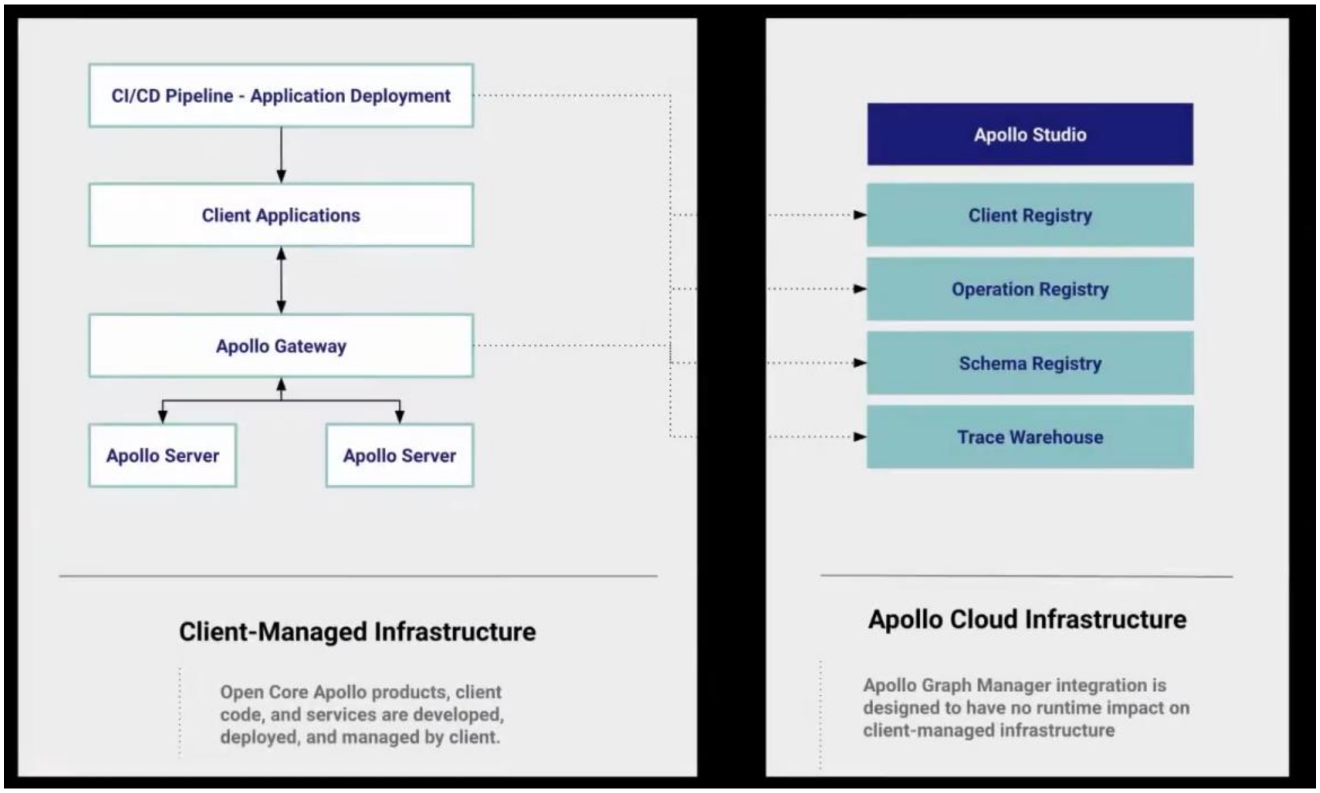
Leading in web and mobile **client libraries**

Leading in **graph registry** and online tooling

Apollo Platform

Enterprise-Grade Data Graph



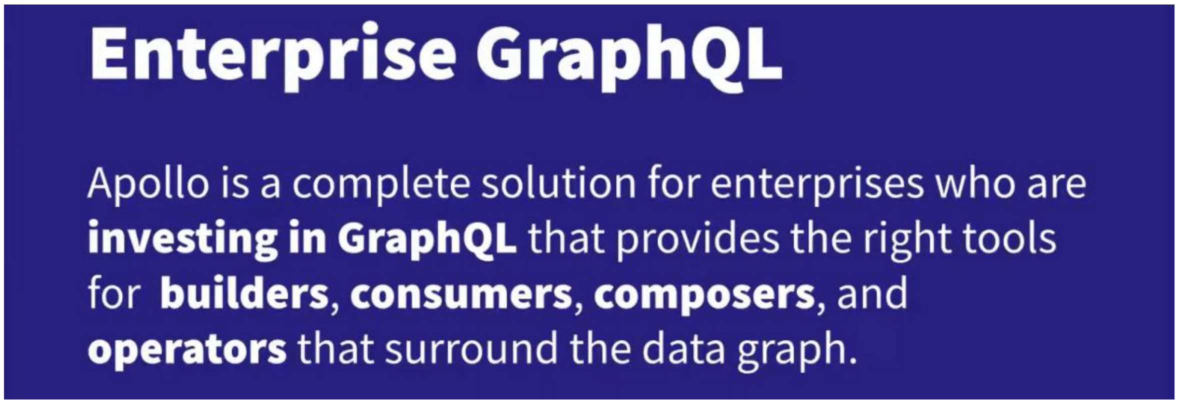


Client-Managed Infrastructure

Open Core Apollo products, client code, and services are developed, deployed, and managed by client.

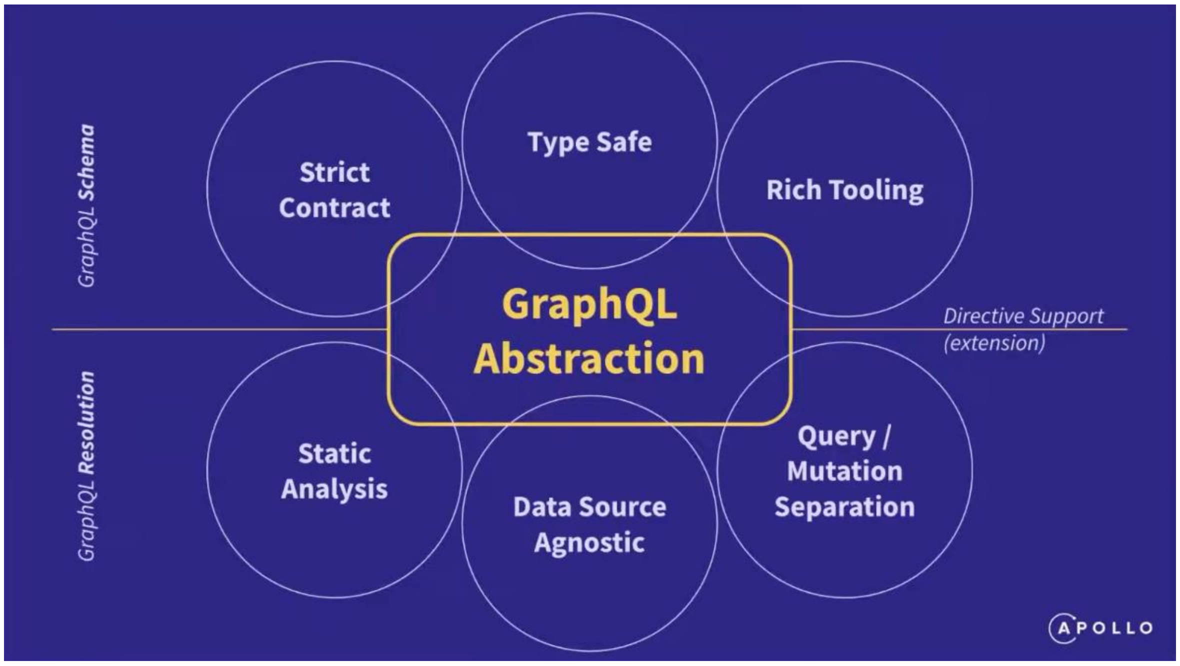
Apollo Cloud Infrastructure

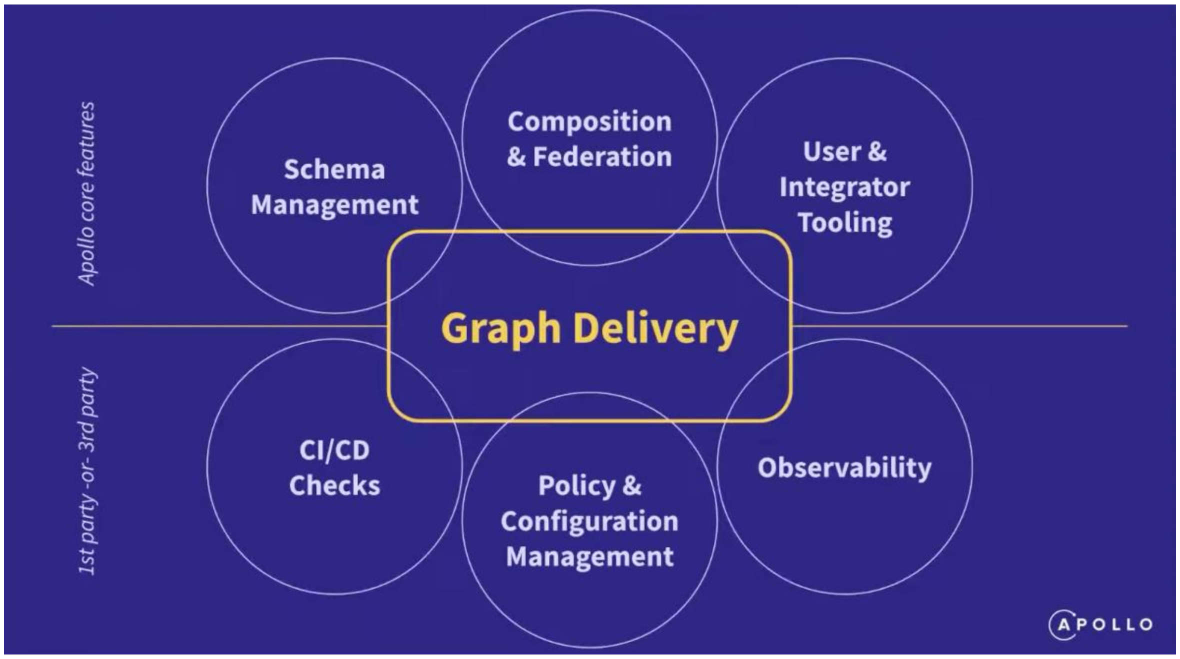
Apollo Graph Manager integration is designed to have no runtime impact on client-managed infrastructure



Enterprise GraphQL

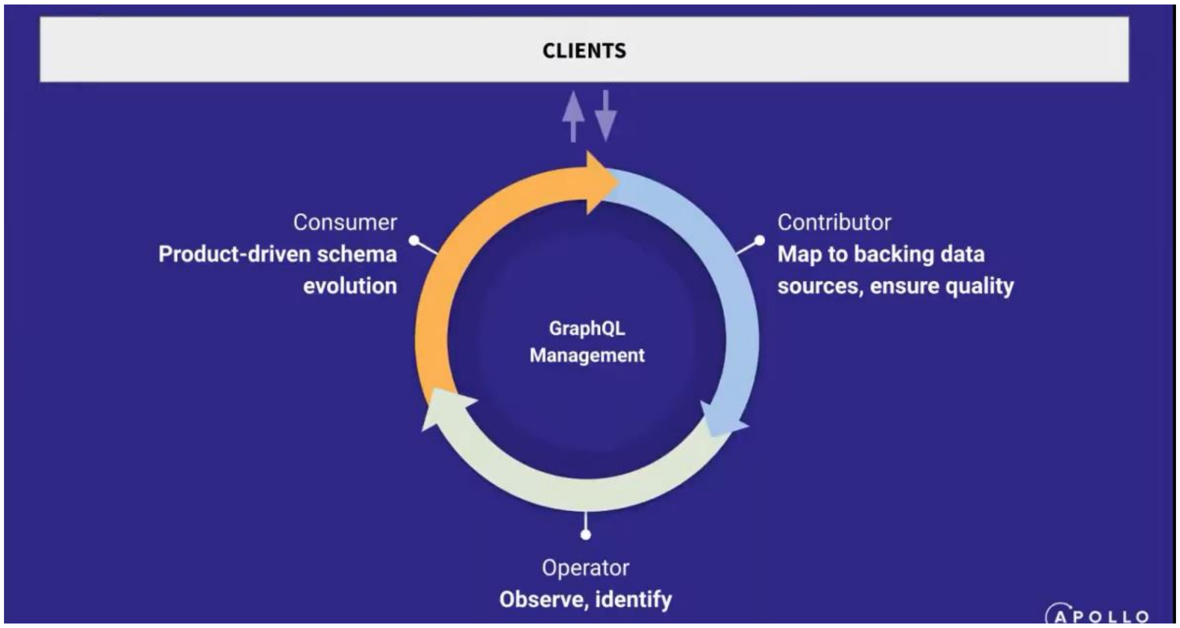
Apollo is a complete solution for enterprises who are **investing in GraphQL** that provides the right tools for **builders**, **consumers**, **composers**, and **operators** that surround the data graph.

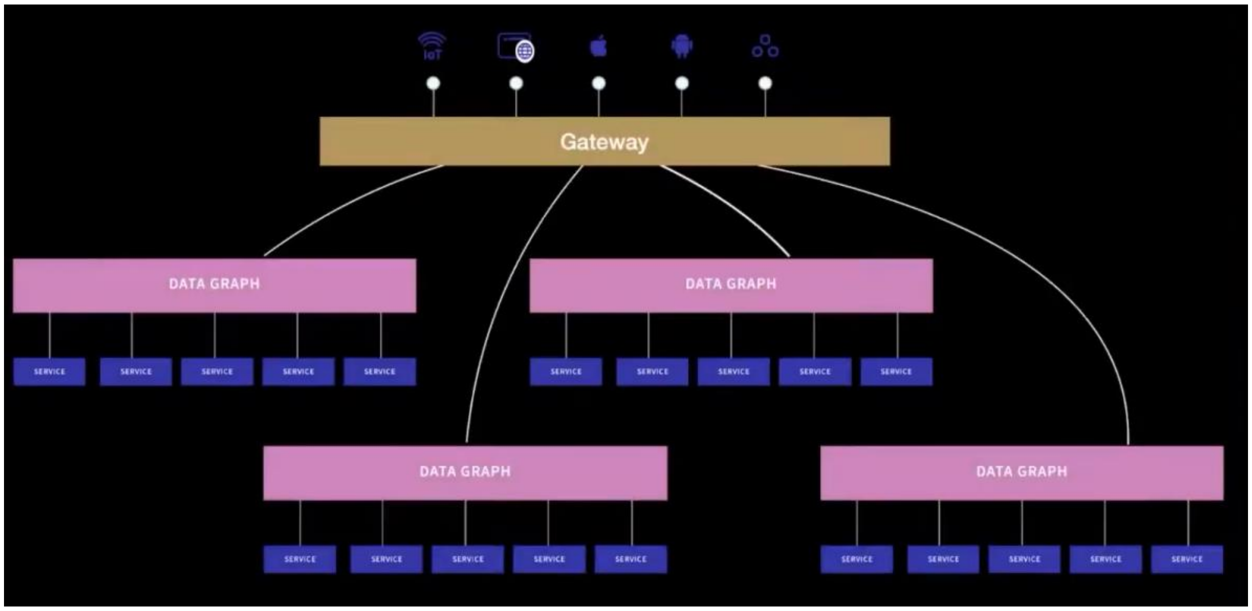


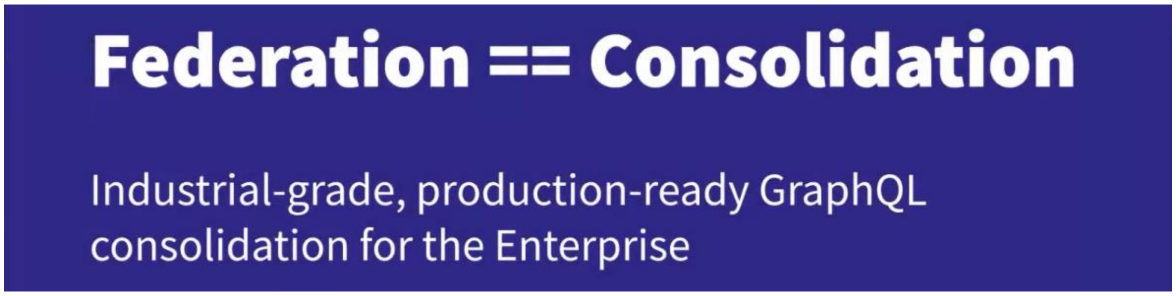


What Apollo Has Learned

Enterprise-Scale Challenges and Consolidation

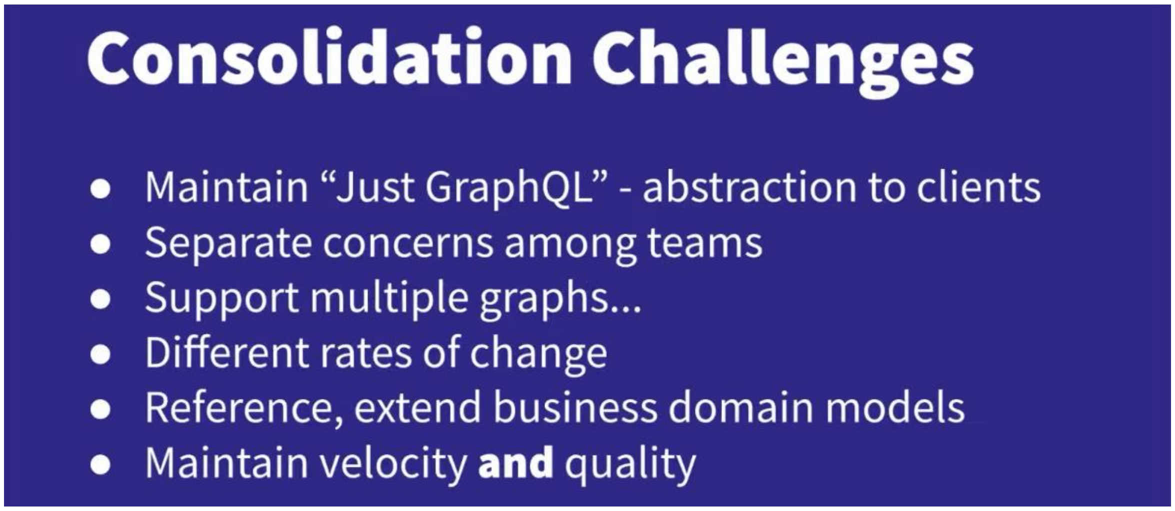






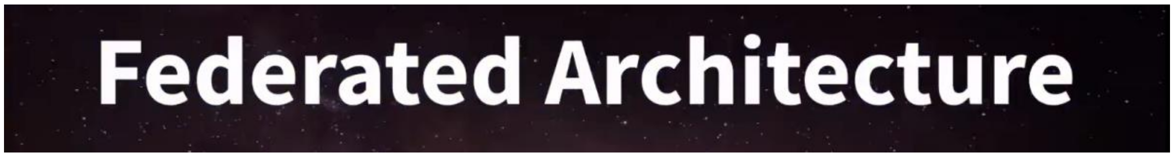
Federation == Consolidation

Industrial-grade, production-ready GraphQL consolidation for the Enterprise

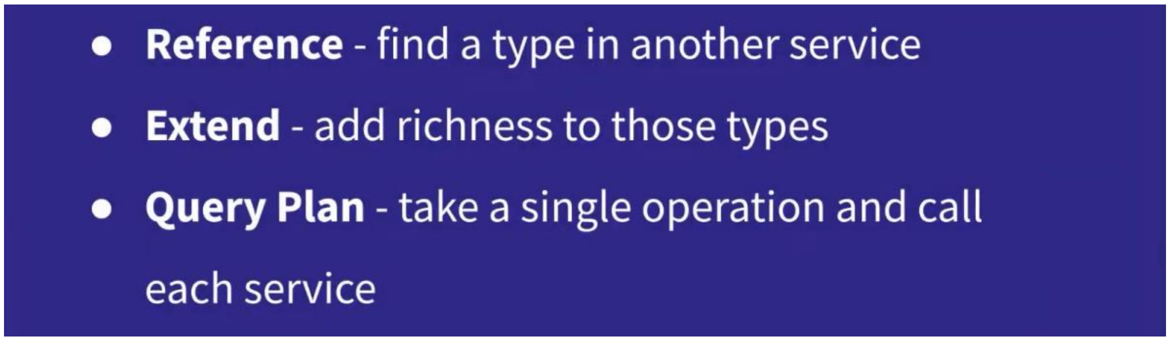


Consolidation Challenges

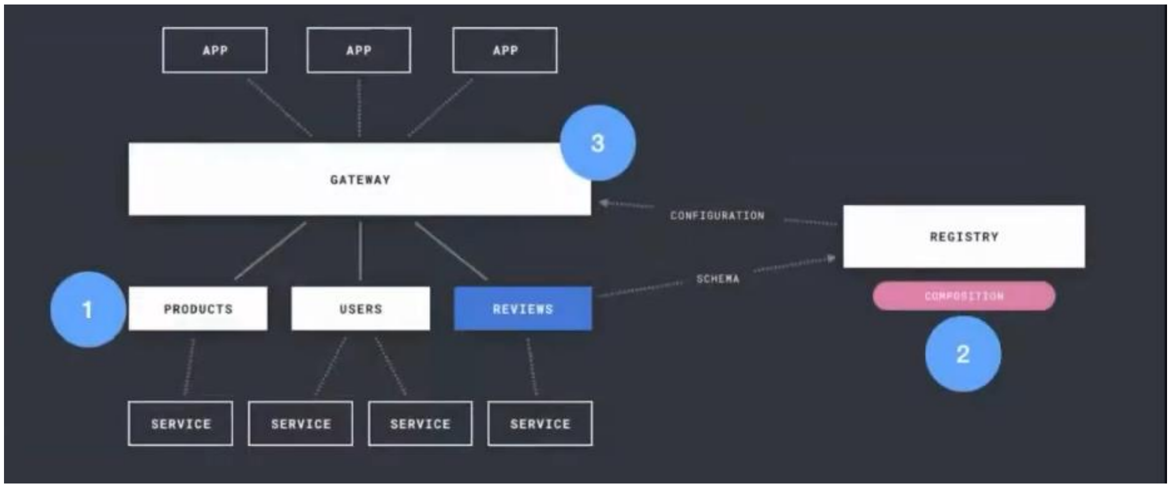
- Maintain "Just GraphQL" - abstraction to clients
- Separate concerns among teams
- Support multiple graphs...
- Different rates of change
- Reference, extend business domain models
- Maintain velocity **and** quality



Federated Architecture



- **Reference** - find a type in another service
- **Extend** - add richness to those types
- **Query Plan** - take a single operation and call each service

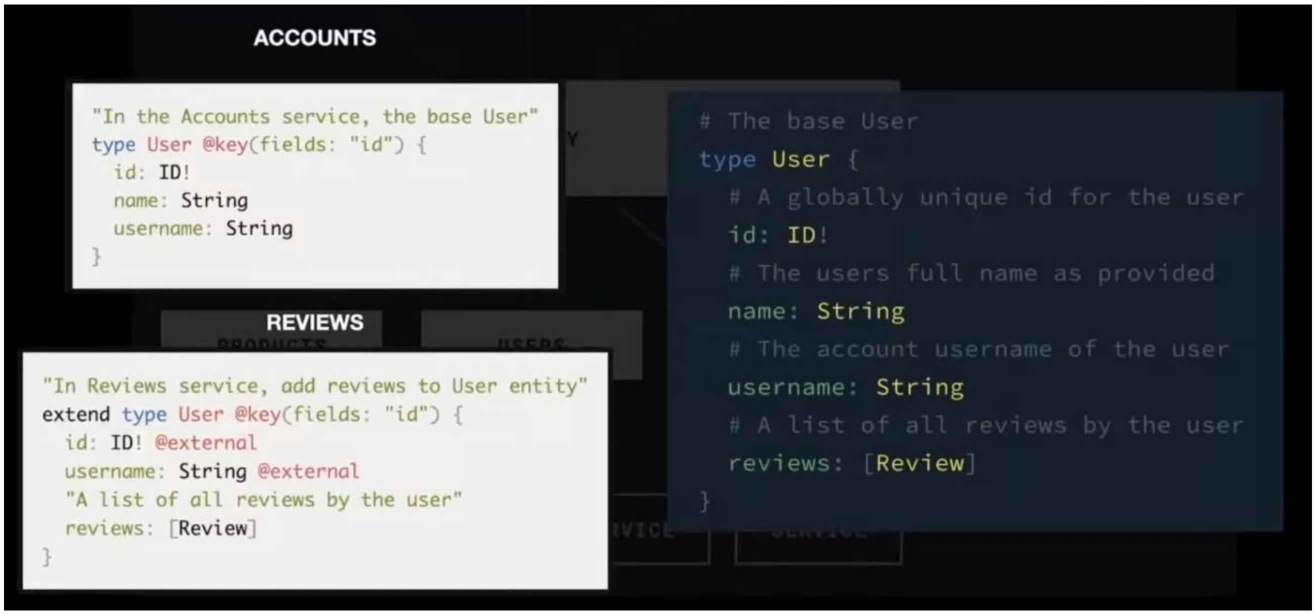


This is a basic flow of what managed federation looks like in the Apollo world. Apollo Studio does new schema validation

Type → Entity

Entity → Extend

We want to be able convert a Type into an Entity and the be able to extend that Entity



ACCOUNTS

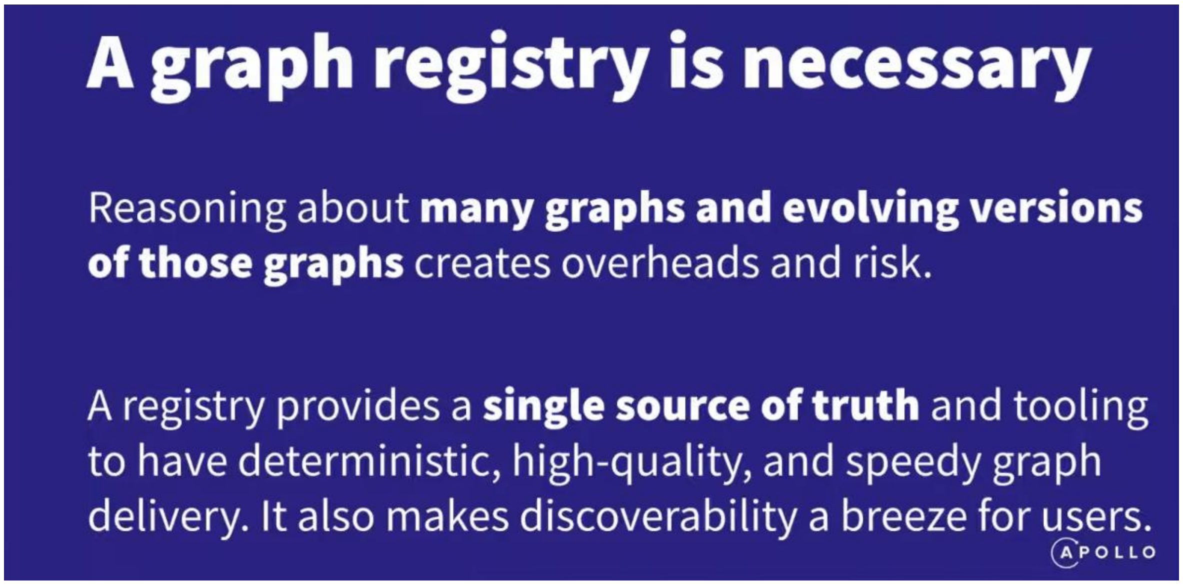
```
"In the Accounts service, the base User"
type User @key(fields: "id") {
  id: ID!
  name: String
  username: String
}
```

REVIEWS

```
"In Reviews service, add reviews to User entity"
extend type User @key(fields: "id") {
  id: ID! @external
  username: String @external
  "A list of all reviews by the user"
  reviews: [Review]
}
```

```
# The base User
type User {
  # A globally unique id for the user
  id: ID!
  # The users full name as provided
  name: String
  # The account username of the user
  username: String
  # A list of all reviews by the user
  reviews: [Review]
}
```

We convert a Type like **User** into an **Entity** by providing a **key** (single value or key or compound key with multiple values), we can then extend the User entity by adding a list of Reviews. The User sees the final composed User graph.

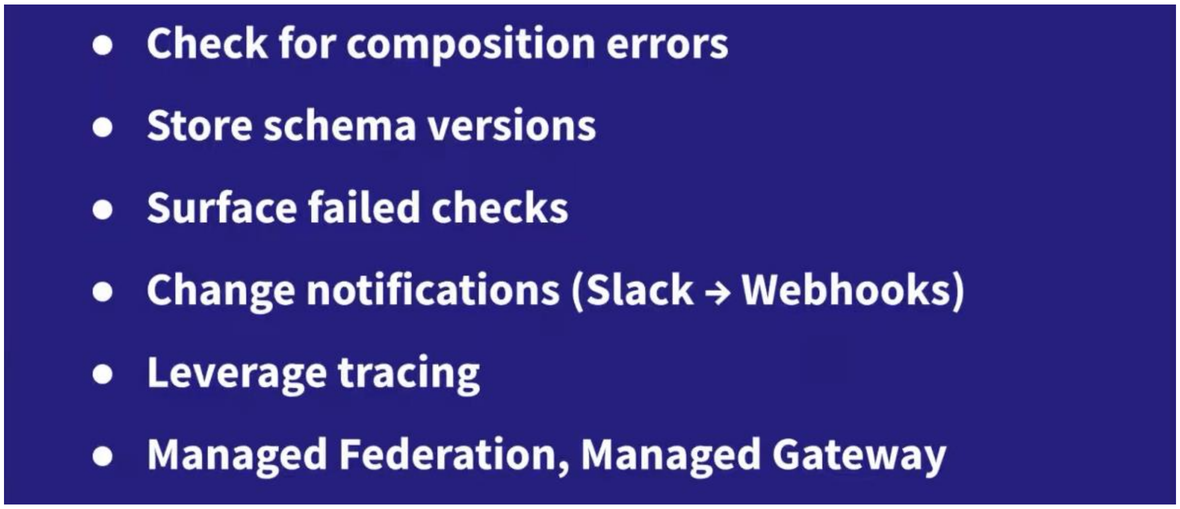


A graph registry is necessary

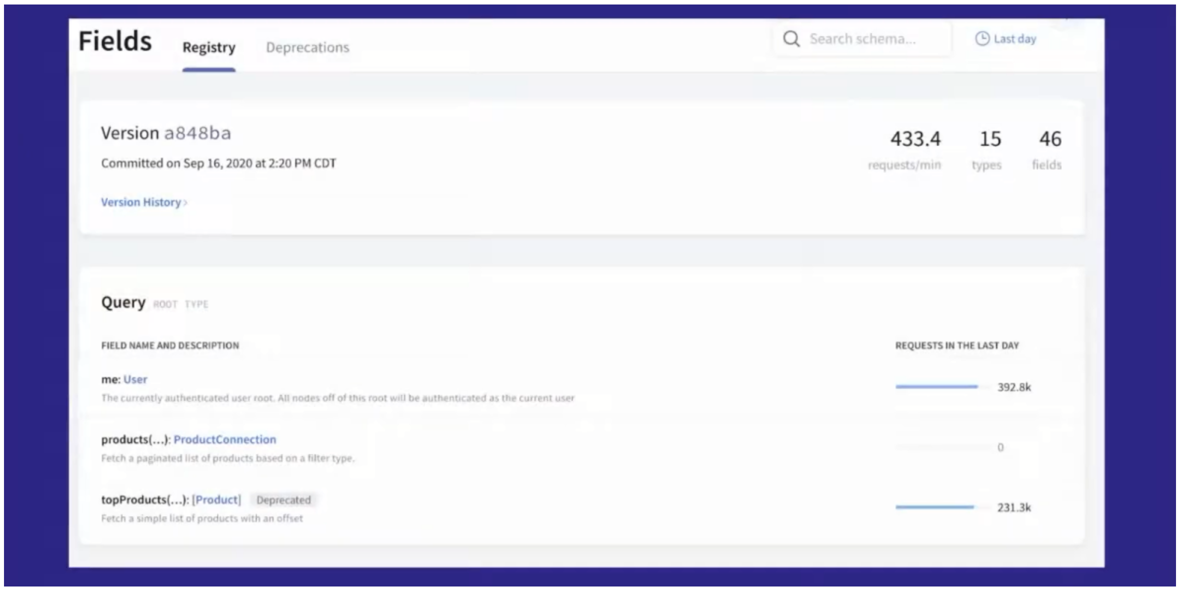
Reasoning about **many graphs and evolving versions of those graphs** creates overheads and risk.

A registry provides a **single source of truth** and tooling to have deterministic, high-quality, and speedy graph delivery. It also makes discoverability a breeze for users.

APOLLO



- **Check for composition errors**
- **Store schema versions**
- **Surface failed checks**
- **Change notifications (Slack → Webhooks)**
- **Leverage tracing**
- **Managed Federation, Managed Gateway**

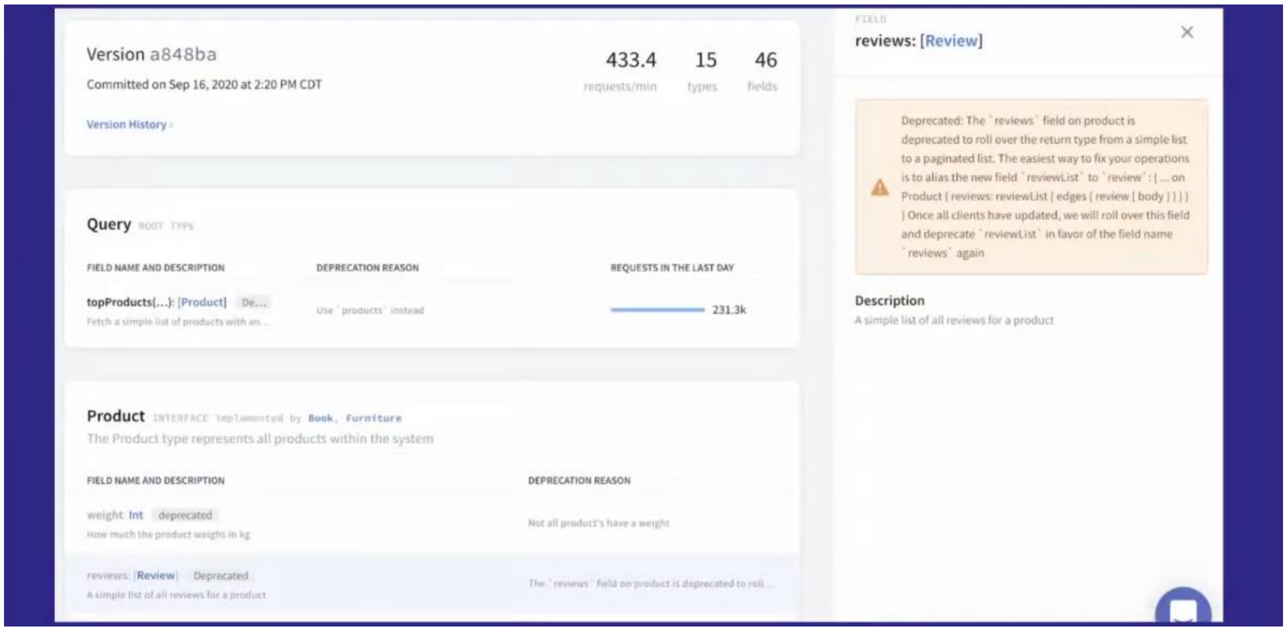


Version a848ba
Committed on Sep 16, 2020 at 2:20 PM CDT

433.4	15	46
requests/min	types	fields

Query ROOT TYPE

FIELD NAME AND DESCRIPTION	REQUESTS IN THE LAST DAY
me: User The currently authenticated user root. All nodes off of this root will be authenticated as the current user	392.8k
products(...): ProductConnection Fetch a paginated list of products based on a filter type.	0
topProducts(...): [Product] Deprecated Fetch a simple list of products with an offset	231.3k



Version a848ba

Committed on Sep 16, 2020 at 2:20 PM CDT

Version History ›

433.4 requests/min | 15 types | 46 fields

Query ROOT TYPE

FIELD NAME AND DESCRIPTION	DEPRECATION REASON	REQUESTS IN THE LAST DAY
topProducts(...): [Product] De... Fetch a simple list of products with an...	Use `products` instead	231.3k

Product INTERFACE implemented by Book, Furniture

The Product type represents all products within the system

FIELD NAME AND DESCRIPTION	DEPRECATION REASON
weight: Int deprecated How much the product weighs in kg	Not all product's have a weight
reviews: [Review] Deprecated A simple list of all reviews for a product	The `reviews` field on product is deprecated to roll ...

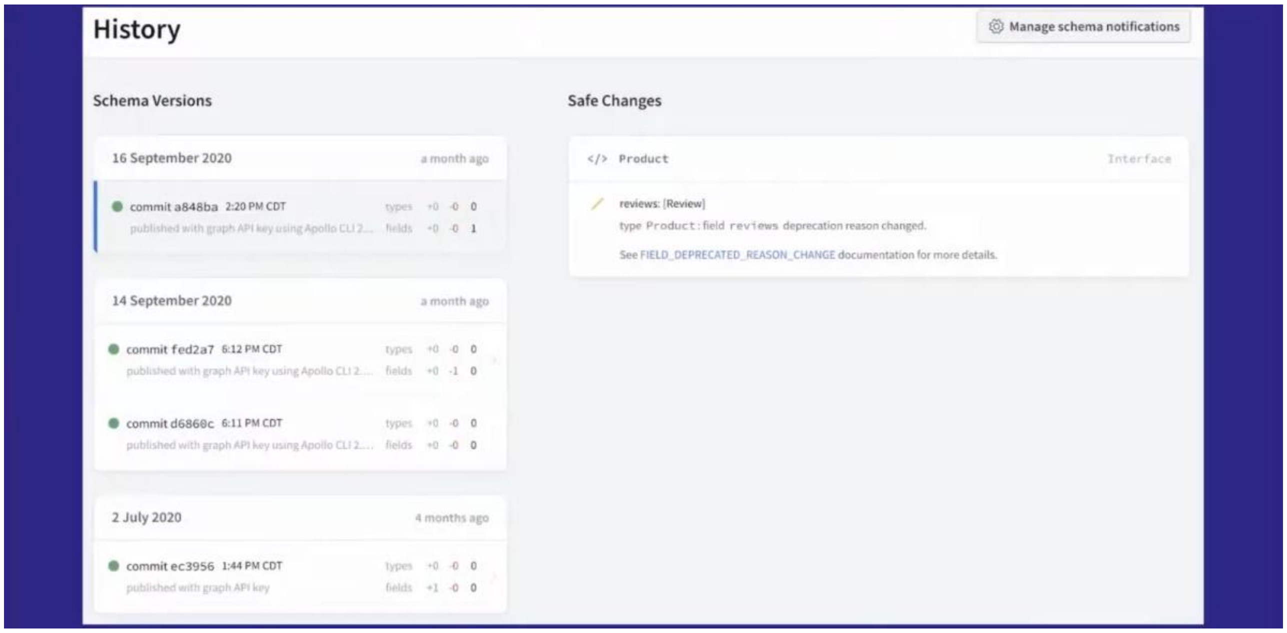
FIELD

reviews: [Review]

Deprecated: The `reviews` field on product is deprecated to roll over the return type from a simple list to a paginated list. The easiest way to fix your operations is to alias the new field `reviewList` to `review` : { ... on Product { reviews: reviewList { edges { review { body } } } } } Once all clients have updated, we will roll over this field and deprecate `reviewList` in favor of the field name `reviews` again

Description

A simple list of all reviews for a product



History

Schema Versions

16 September 2020 a month ago

commit a848ba 2:20 PM CDT	types	+0	-0	0
published with graph API key using Apollo CLI 2....	fields	+0	-0	1

14 September 2020 a month ago

commit fed2a7 6:12 PM CDT	types	+0	-0	0
published with graph API key using Apollo CLI 2....	fields	+0	-1	0
commit d6860c 6:11 PM CDT	types	+0	-0	0
published with graph API key using Apollo CLI 2....	fields	+0	-0	0

2 July 2020 4 months ago

commit ec3956 1:44 PM CDT	types	+0	-0	0
published with graph API key	fields	+1	-0	0

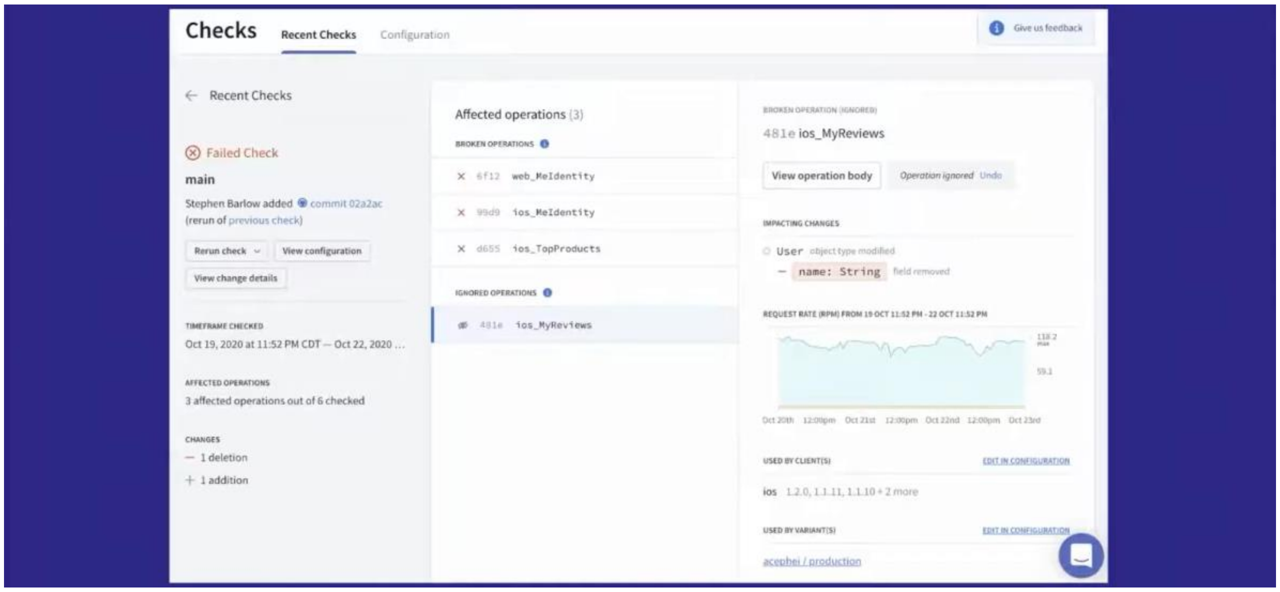
Safe Changes

</> Product Interface

reviews: [Review]

type Product: field reviews deprecation reason changed.

See FIELD_DEPRECATED_REASON_CHANGE documentation for more details.



Checks

← Recent Checks

Failed Check

main

Stephen Barlow added commit 02a2ac (rerun of previous check)

TIMEFRAME CHECKED

Oct 19, 2020 at 11:52 PM CDT — Oct 22, 2020 ...

AFFECTED OPERATIONS

3 affected operations out of 6 checked

CHANGES

— 1 deletion

+ 1 addition

Affected operations (3)

BROKEN OPERATIONS

× 6f12 web_MeIdentity

× 99d9 ios_MeIdentity

× d655 ios_TopProducts

IGNORED OPERATIONS

481e ios_MyReviews

BROKEN OPERATION (IGNORED)

481e ios_MyReviews

IMPACTING CHANGES

User object type modified

— name: String field removed

REQUEST RATE (RPM) FROM 19 OCT 11:52 PM - 22 OCT 11:52 PM

USED BY CLIENT(S) EDIT IN CONFIGURATION

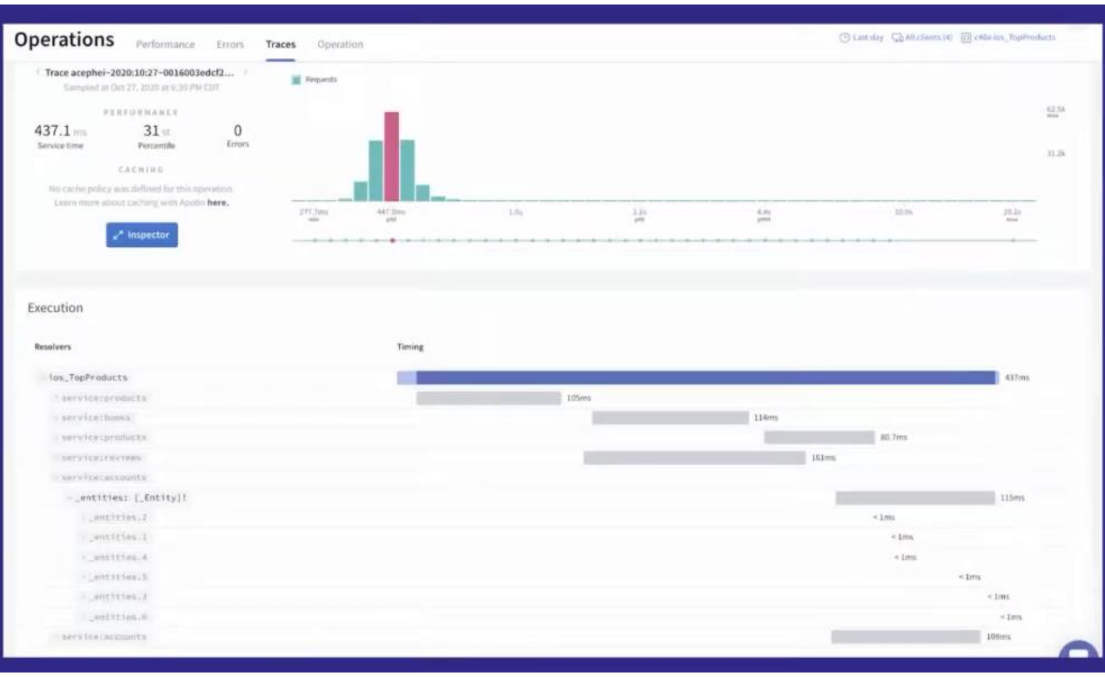
ios 1.2.0, 1.1.11, 1.1.10 + 2 more

USED BY VARIANT(S) EDIT IN CONFIGURATION

acephei / production

Query Planning

- Directives → Metadata
- Traverse document, Parallel and Serial Fetch
  - @key, @requires ← Dependency Groups
  - Flatten, Map ← Pass data among services
  - @provides optimization (reduce hops)
  - RemoteGraphQLDataSource Execution



Trace acephei-2020:10:27-0016003edcf2...

Sampled at Oct 27, 2020 at 9:30 PM CDT

PERFORMANCE

437.1 ms Service time | 31 st Percentile | 0 Errors

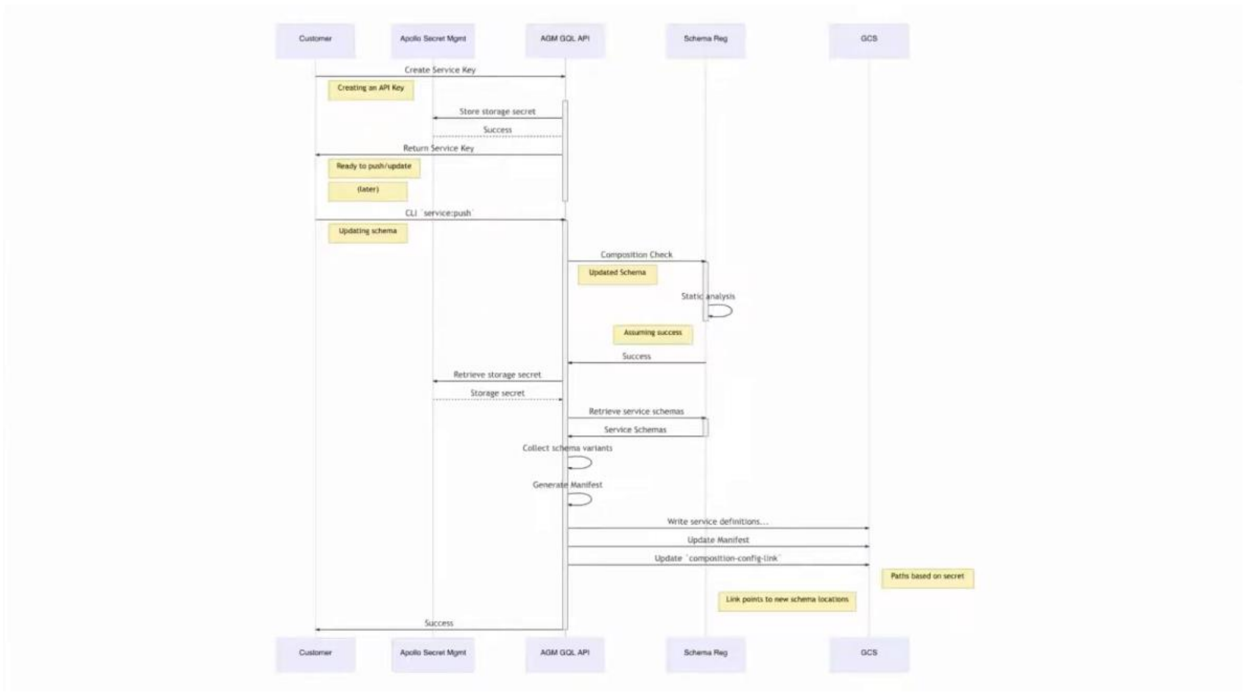
CACHING

No cache policy was defined for this operation. Learn more about caching with Apollo **here.**

Execution

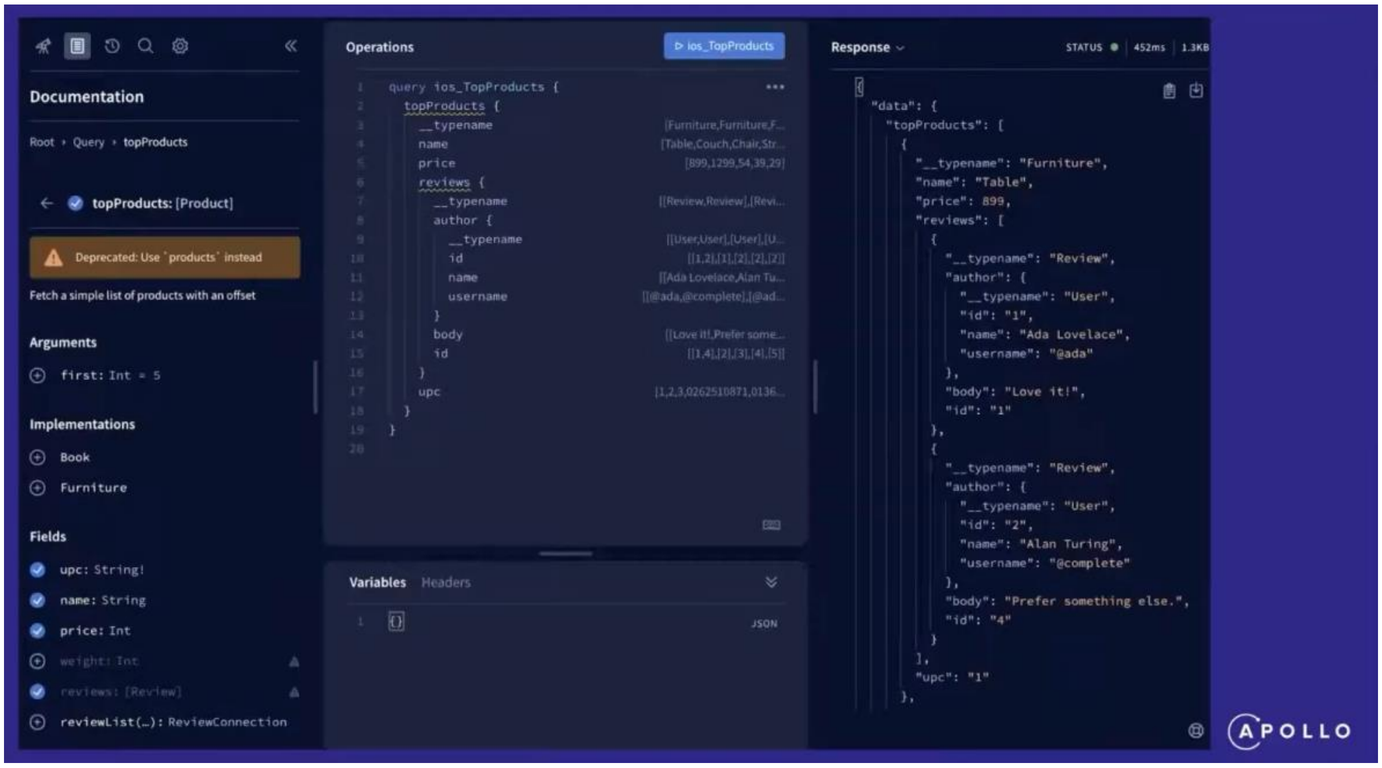
```
Resolvers                          Timing
ios_TopProducts                    437ms
  service:products                 105ms
  service:books                    114ms
  service:products                 80.7ms
  service:reviews                  161ms
  service:accounts
    _entities: [_Entity]!          115ms
      _entities.2                  < 1ms
      _entities.1                  < 1ms
      _entities.4                  < 1ms
      _entities.5                  < 1ms
      _entities.3                  < 1ms
      _entities.6                  < 1ms
  service:accounts                 108ms
```

Managed Gateway



Explore in Apollo Studio

Solving the “blank page” problem of learning and querying a graph



topProducts: [Product]

Deprecated: Use `products` instead

Fetch a simple list of products with an offset

Arguments

first: Int = 5

Implementations

Book

Furniture

Fields

upc: String!

name: String

price: Int

weight: Int

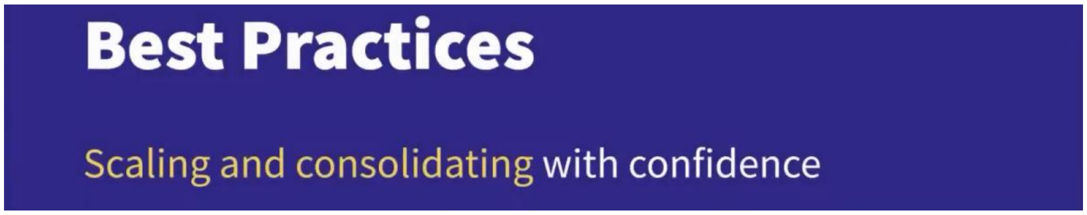
reviews: [Review]

reviewList(…): ReviewConnection

```
query ios_TopProducts {
  topProducts {
    __typename                         [Furniture,Furniture,F...
    name                               [Table,Couch,Chair,Str...
    price                              [899,1299,54,39,29]
    reviews {
      __typename                       [[Review,Review],[Revi...
      author {
        __typename                     [[User,User],[User],[U...
        id                             [[1,2],[1],[2],[2],[2]]
        name                           [[Ada Lovelace,Alan Tu...
        username                       [[@ada,@complete],[@ad...
      }
      body                             [[Love it!,Prefer some...
      id                               [[1,4],[2],[3],[4],[5]]
    }
    upc                                [1,2,3,0262510871,0136...
  }
}
```

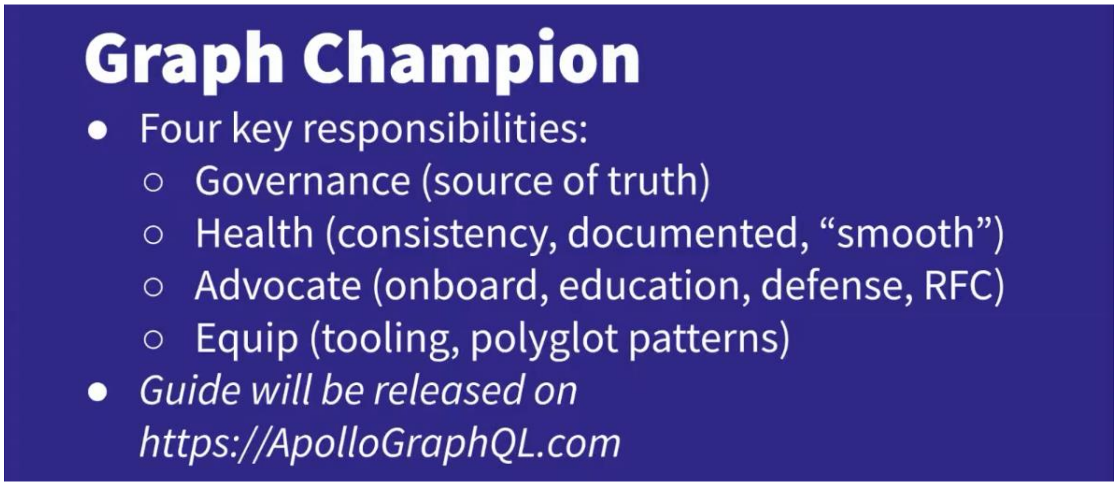
```
{}
```

```
{
  "data": {
    "topProducts": [
      {
        "__typename": "Furniture",
        "name": "Table",
        "price": 899,
        "reviews": [
          {
            "__typename": "Review",
            "author": {
              "__typename": "User",
              "id": "1",
              "name": "Ada Lovelace",
              "username": "@ada"
            },
            "body": "Love it!",
            "id": "1"
          },
          {
            "__typename": "Review",
            "author": {
              "__typename": "User",
              "id": "2",
              "name": "Alan Turing",
              "username": "@complete"
            },
            "body": "Prefer something else.",
            "id": "4"
          }
        ],
        "upc": "1"
      },
```



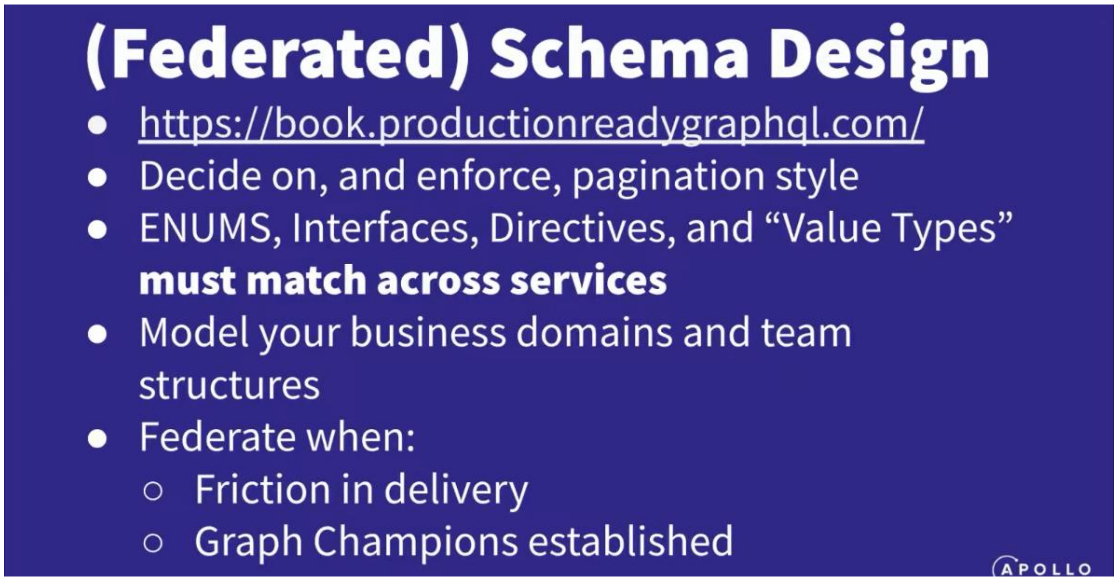
Best Practices

Scaling and consolidating with confidence



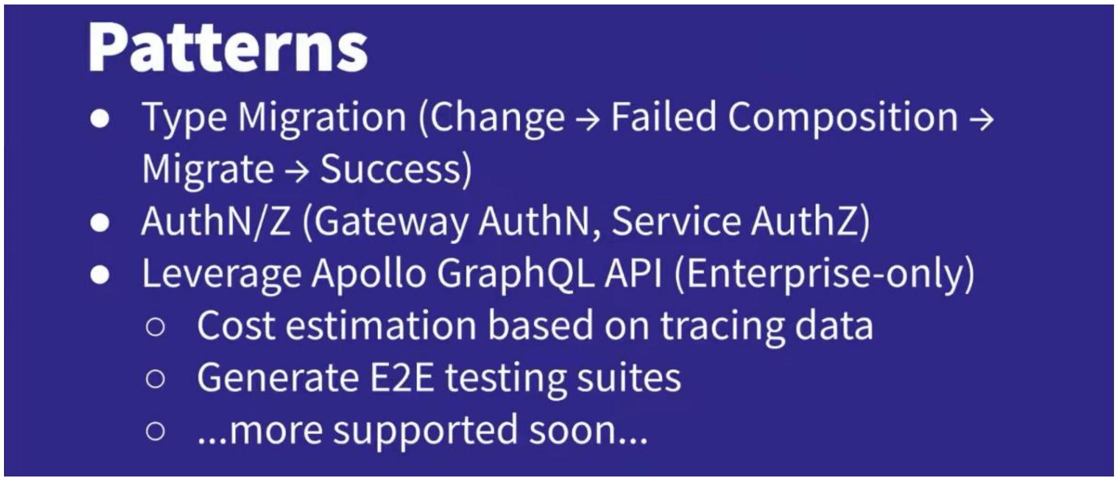
Graph Champion

- Four key responsibilities:
  - Governance (source of truth)
  - Health (consistency, documented, “smooth”)
  - Advocate (onboard, education, defense, RFC)
  - Equip (tooling, polyglot patterns)
- *Guide will be released on https://ApolloGraphQL.com*



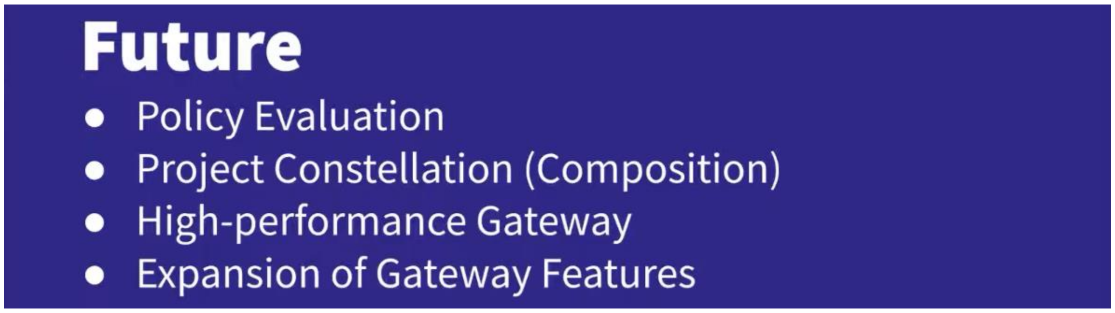
(Federated) Schema Design

- https://book.productionreadygraphql.com/
- Decide on, and enforce, pagination style
- ENUMS, Interfaces, Directives, and “Value Types” **must match across services**
- Model your business domains and team structures
- Federate when:
  - Friction in delivery
  - Graph Champions established



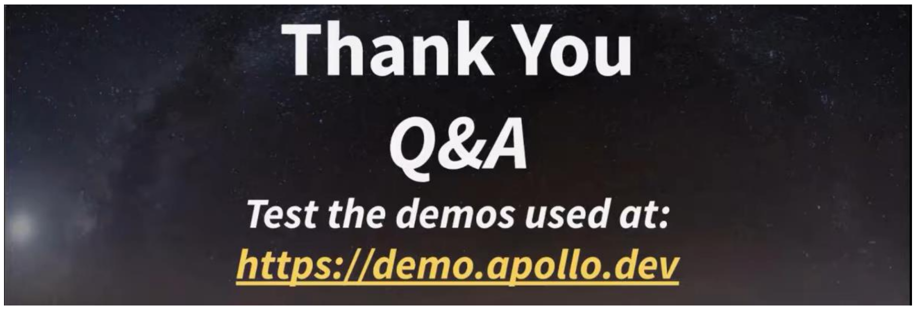
Patterns

- Type Migration (Change → Failed Composition → Migrate → Success)
- AuthN/Z (Gateway AuthN, Service AuthZ)
- Leverage Apollo GraphQL API (Enterprise-only)
  - Cost estimation based on tracing data
  - Generate E2E testing suites
  - ...more supported soon...



Future

- Policy Evaluation
- Project Constellation (Composition)
- High-performance Gateway
- Expansion of Gateway Features



Thank You

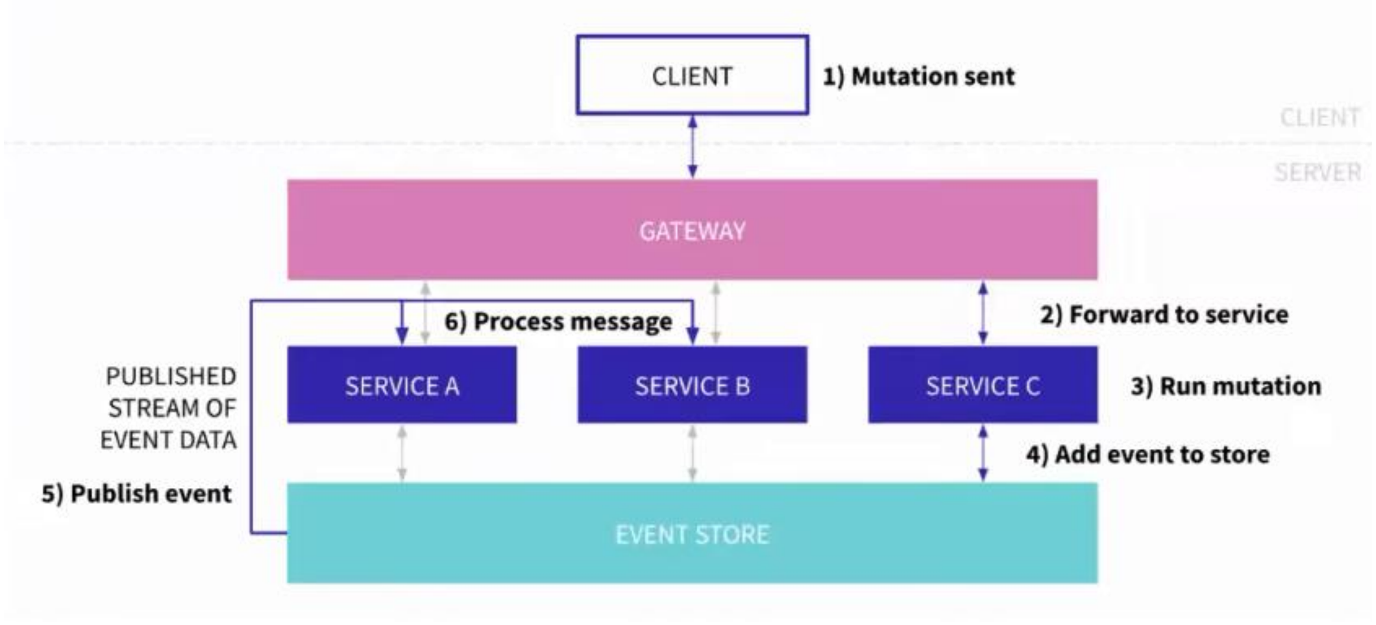
Q&A

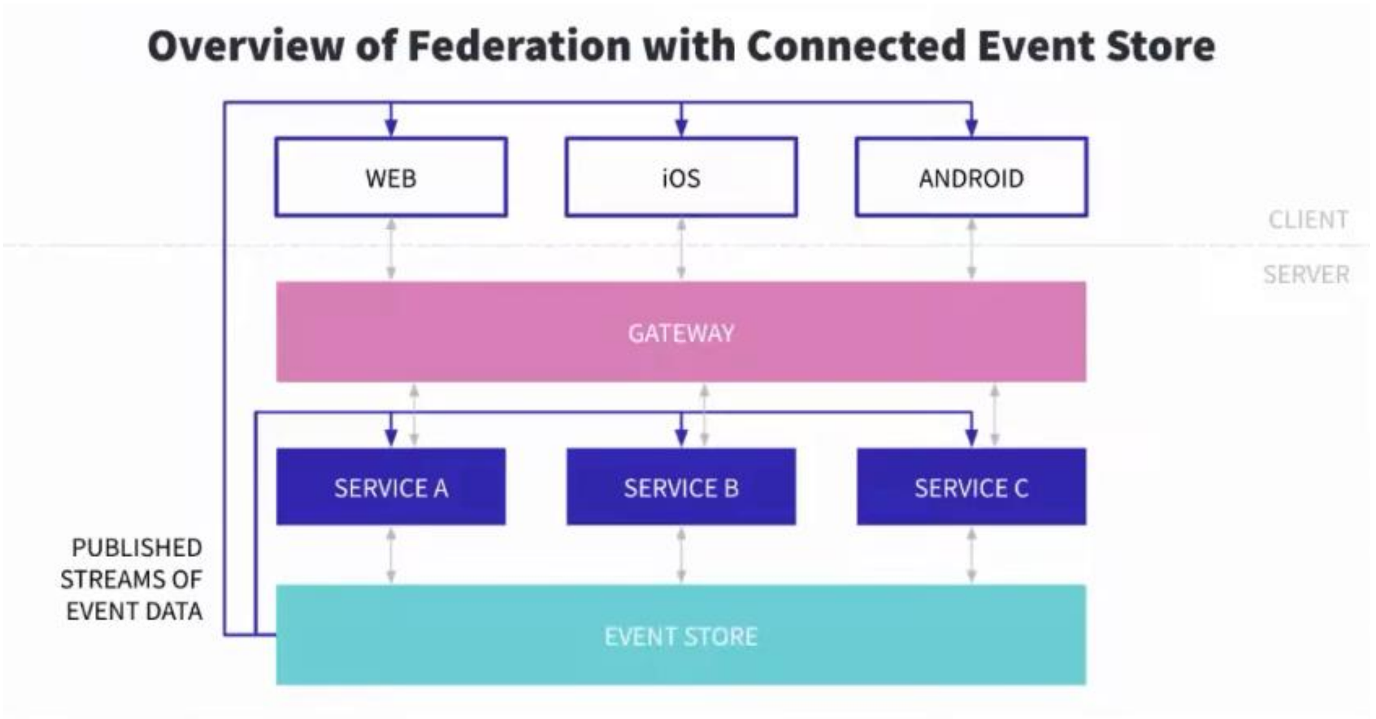
Test the demos used at:

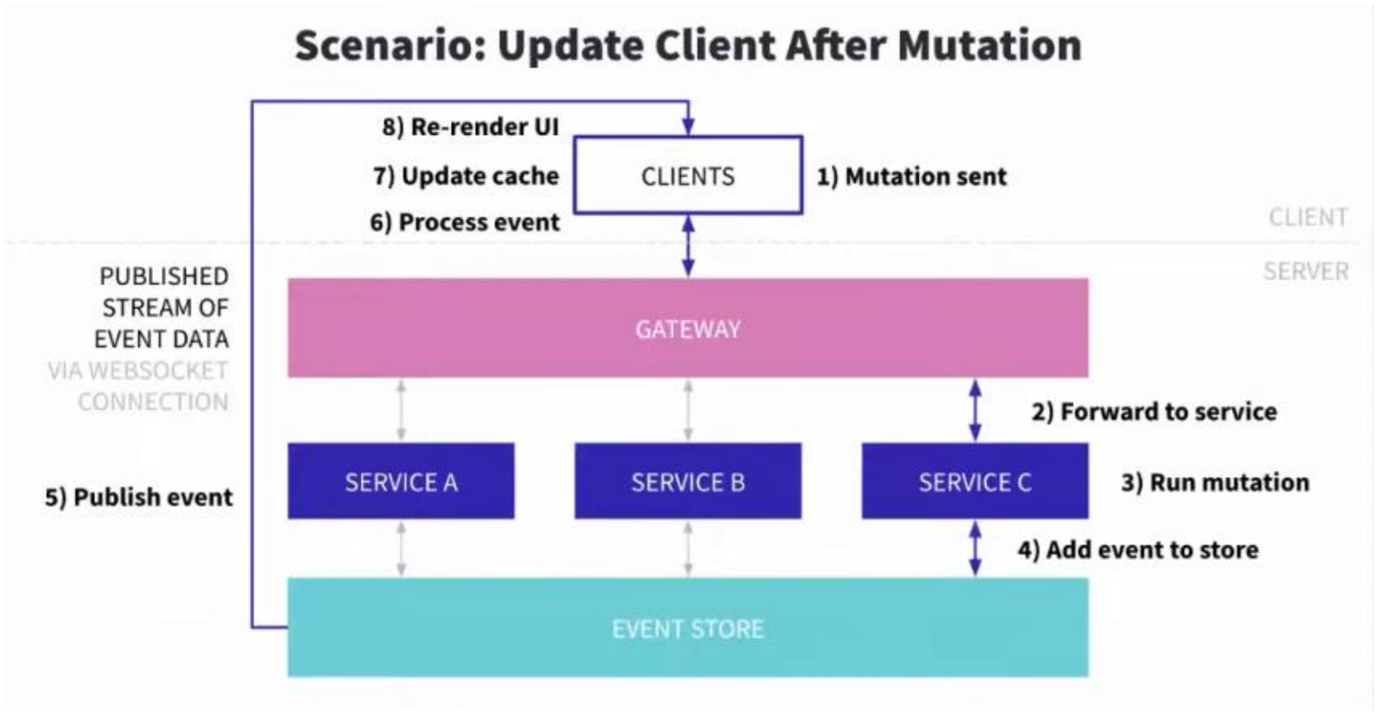
https://demo.apollo.dev

An Architecture for Real-time Queries with Apollo Federation

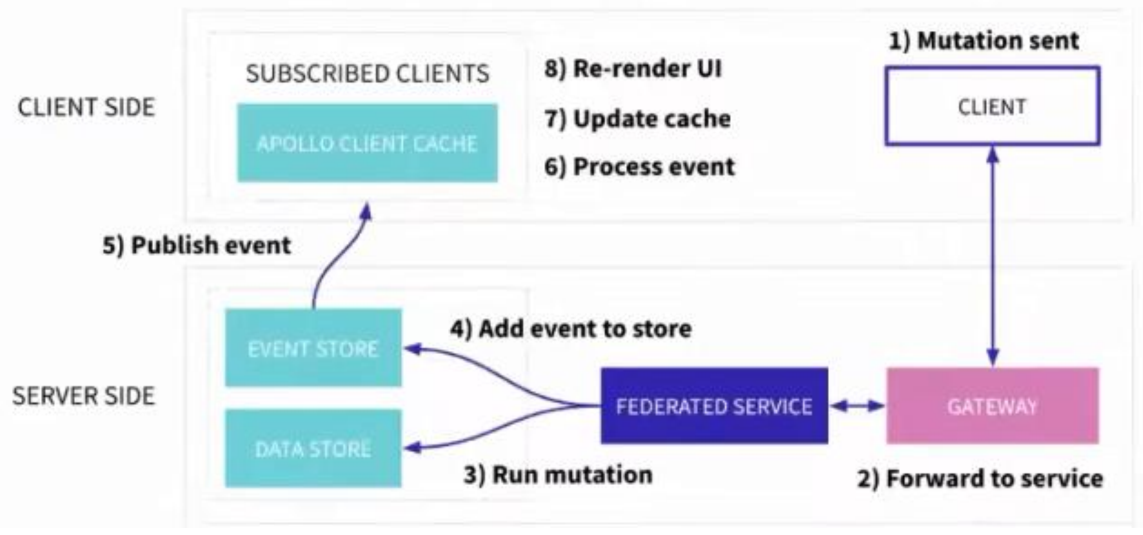
Scenario: Interservice Communication




Overview of Federation with Connected Event Store
WEB
iOS
ANDROID
CLIENT
SERVER
GATEWAY
SERVICE A
SERVICE B
SERVICE C
PUBLISHED STREAMS OF EVENT DATA
EVENT STORE


Scenario: Update Client After Mutation
8) Re-render UI
7) Update cache
6) Process event
CLIENTS
1) Mutation sent
CLIENT
SERVER
PUBLISHED STREAM OF EVENT DATA
VIA WEBSOCKET CONNECTION
GATEWAY
2) Forward to service
5) Publish event
SERVICE A
SERVICE B
SERVICE C
3) Run mutation
4) Add event to store
EVENT STORE

Detailed View: Update Client After Mutation



Implementing Service Schema

```
directive @_live(events: [PublishableEvent!]!) on QUERY

directive @_publish(
  payload: String,
  event: PublishableEvent!
) on FIELD_DEFINITION

enum PublishableEvent {
  AUTHOR_REMOVED
  POST_ADDED
}
```

Implementing Service Schema

```
extend type Mutation {
  addPost(authorID: ID!, content: String, title: String): Post
  @_publish(
    payload: "authorID content id publishedAt title"
    event: POST_ADDED
  )
}
```