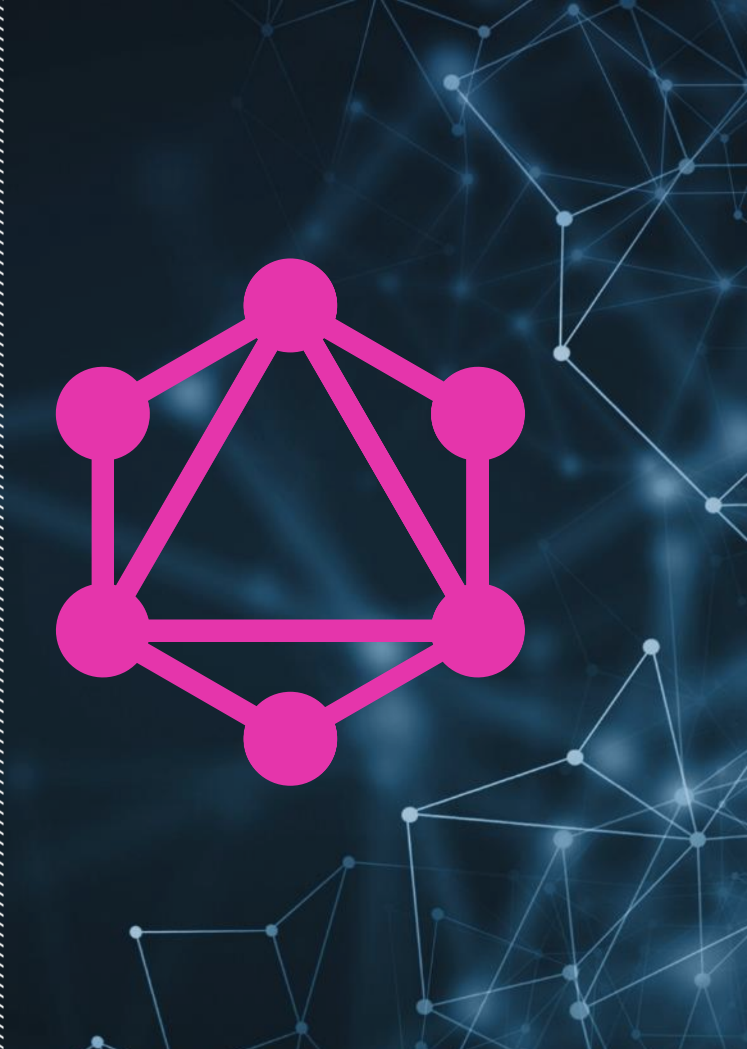
<epam>

GraphQL

A query language for your API

A success story?

Published at *Kornel on Digital on 4/Nov/2019*



By Facebook - https://github.com/facebook/graphql/blob/master/resources/GraphQL%20Logo.svg (BSD)

Agenda

APIs

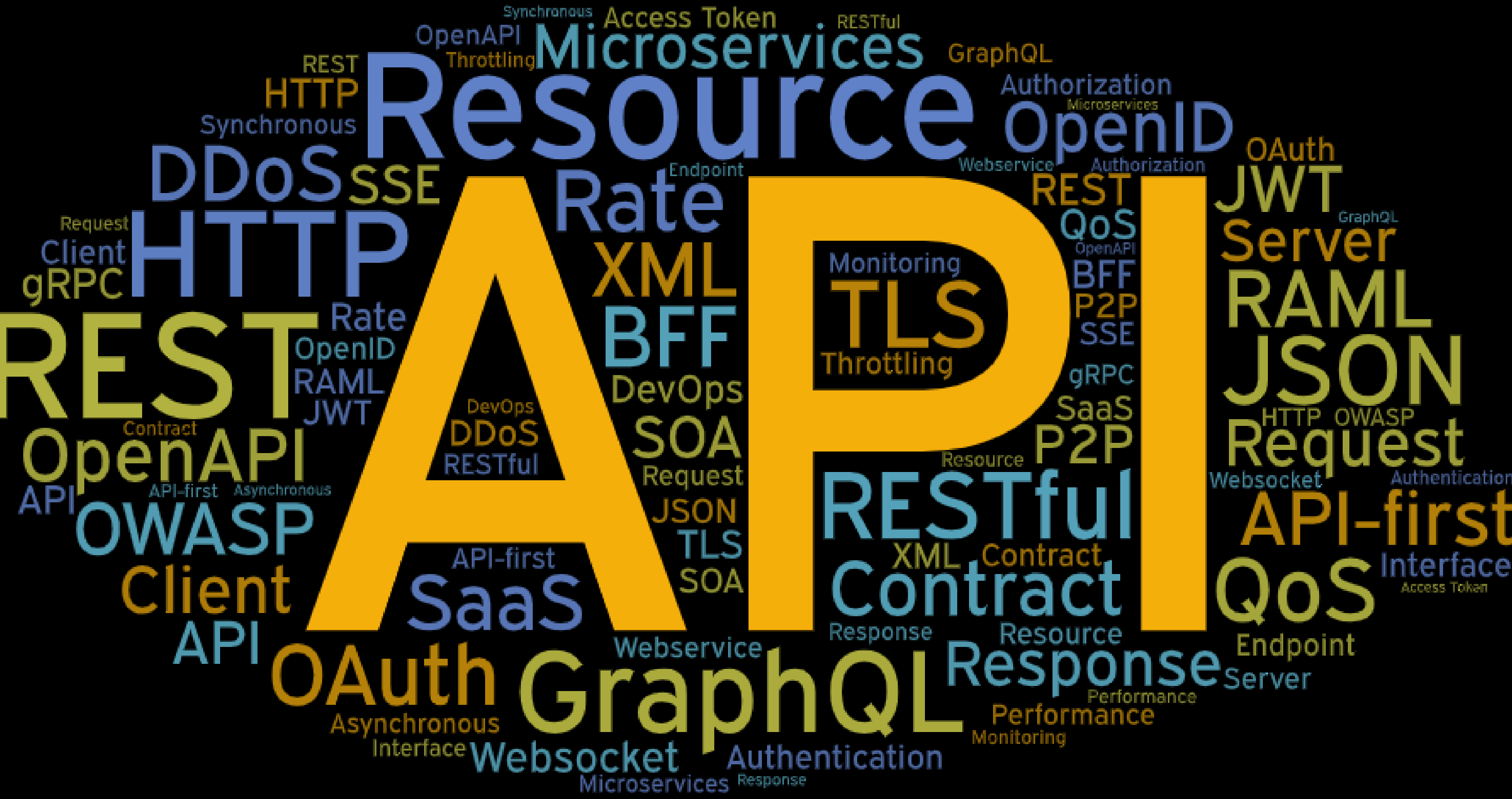
- API Considerations
- RESTful API

GraphQL introduction

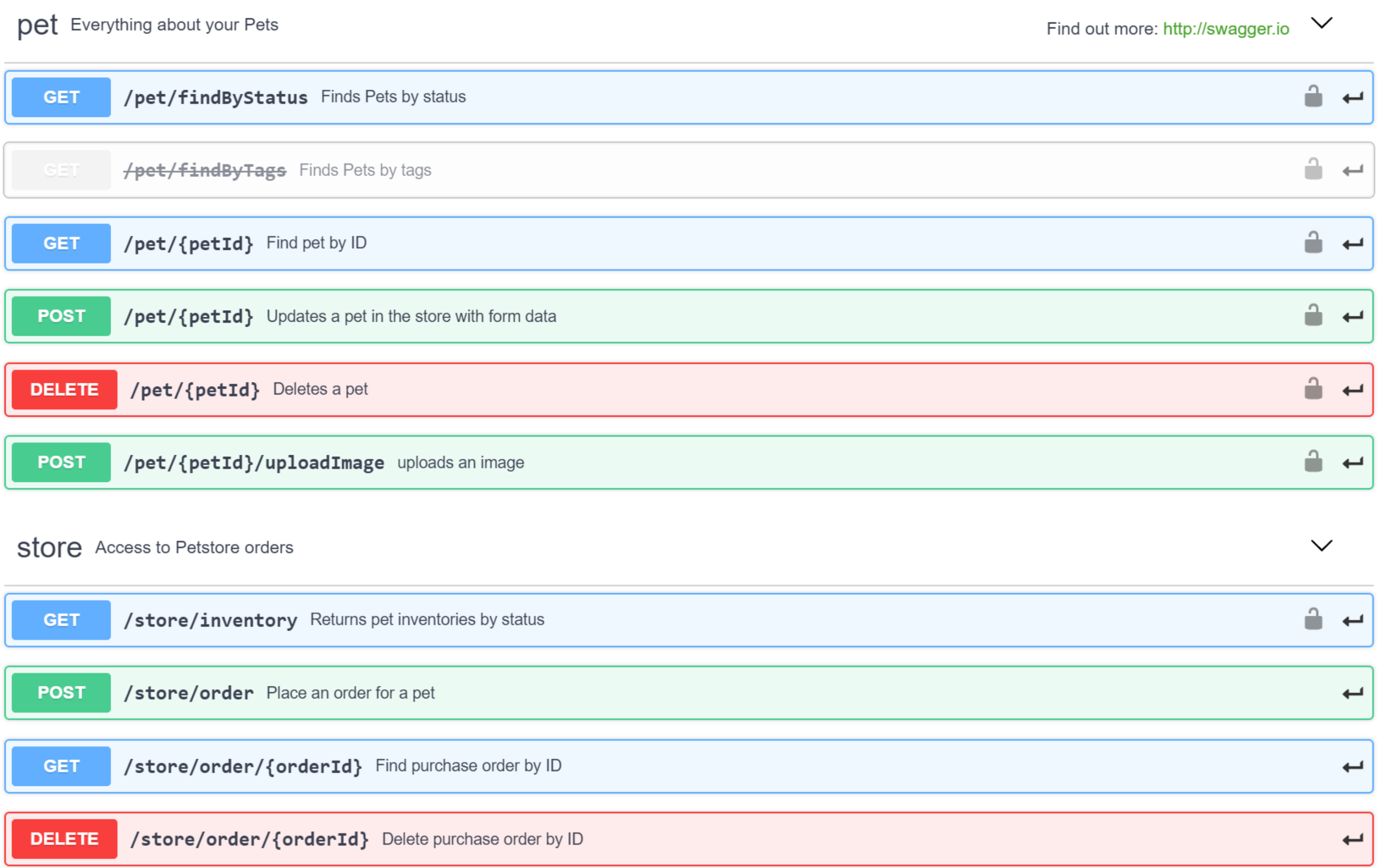
- Design principles
- REST / GraphQL comparision
- GraphQL features
- Non-functional considerations

GraphQL in practice

- Tools
- Demo


Synchronous
Access Token
RESTful
OpenAPI
Microservices
Throttling
GraphQL
REST
HTTP
Resource
Authorization
Microservices
Synchronous
OpenID
OAuth
DDoS
SSE
Endpoint
Webservice
Authorization
Rate
REST
JWT
Request
HTTP
QoS
GraphQL
Client
OpenAPI
Server
XML
Monitoring
BFF
gRPC
TLS
RAML
REST
Rate
BFF
P2P
OpenID
SSE
Throttling
JSON
RAML
DevOps
gRPC
JWT
SaaS
DevOps
Contract
DDoS
SOA
HTTP
OWASP
OpenAPI
P2P
Request
RESTful
Resource
Request
API-first
Asynchronous
Websocket
Authentication
API
RESTful
OWASP
JSON
API-first
TLS
API-first
XML
Contract
Interface
Client
SaaS
SOA
Access Token
Contract
QoS
API
Response
Resource
API
Webservice
Endpoint
OAuth
GraphQL
Response
Server
Performance
Asynchronous
Performance
Interface
Websocket
Authentication
Monitoring
Microservices
Response

From REST to GraphQL



pet Everything about your Pets

Find out more: http://swagger.io

GET /pet/findByStatus Finds Pets by status

GET ~~/pet/findByTags~~ Finds Pets by tags

GET /pet/{petId} Find pet by ID

POST /pet/{petId} Updates a pet in the store with form data

DELETE /pet/{petId} Deletes a pet

POST /pet/{petId}/uploadImage uploads an image

store Access to Petstore orders

GET /store/inventory Returns pet inventories by status

POST /store/order Place an order for a pet

GET /store/order/{orderId} Find purchase order by ID

DELETE /store/order/{orderId} Delete purchase order by ID

GraphQL - A query language for your API

GraphQL is a **query language for APIs** and a **runtime** for fulfilling those queries with your existing data. GraphQL provides a complete and understandable **description** of the **data** in your API, gives clients the power to **ask for exactly what they need** and nothing more, makes it easier to **evolve APIs** over time, and enables powerful developer tools.

https://graphql.org/

Design principles

Hierarchical

A natural way for clients to describe data requirements for their creation and manipulation of view hierarchies

Product centric

Driven by the requirements of views and front-end engineers

Strong typing

Every GraphQL server defines an application specific type system

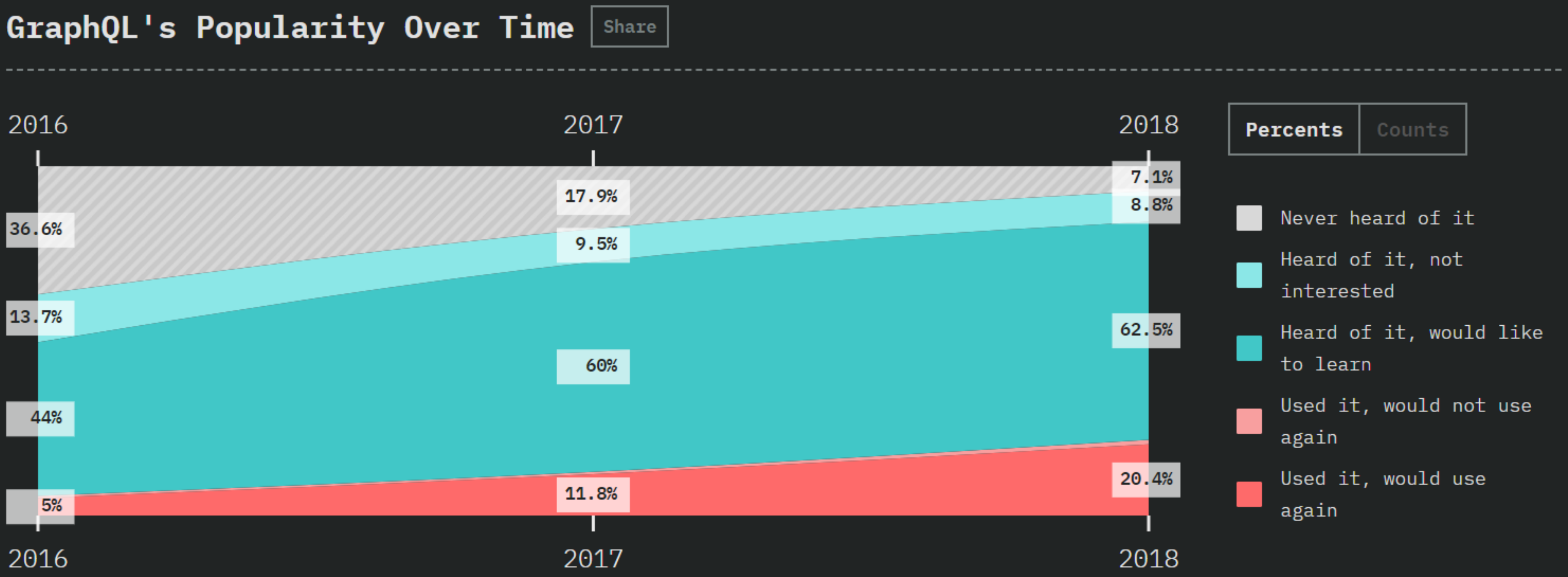
Client-specified queries

Queries with field level granularity. Server returns exactly what a client asks for and no more.

Introspective

The server's type system is queryable by the language itself

https://graphql.github.io/graphql-spec/



GraphQL's Popularity Over Time

https://2018.stateofjs.com/data-layer/graphql/

GraphQL vs REST

REST & GraphQL

REST

REST is an **architectural style** for designing loosely coupled applications on HTTP

- Documentation per endpoint
- Endpoint per resource
- No formal schema definition
- HTTP verbs (GET, PUT, POST, ...)
- Interact with specific resources
- Client specifies the identity of the resource
- Server defined response
- Additional developer effort on data fetching
- Entire API can be versioned

GraphQL

GraphQL is a **query language** and **a runtime** for fulfilling the queries

- Self-documented via GraphQL introspection
- Endpoint per API
- Strongly typed
- Queries, mutations, subscriptions
- Interact with several resources (one operation)
- No need to specify the identity of the resource(s)
- Client defines the shape of the response
- Think less about data fetching („what” vs. „how”)
- Designed for API evolution

There is **notable impact on (almost) every quality attributes** when someone wants to move away from REST and introduce GraphQL

REST and GraphQL comparision

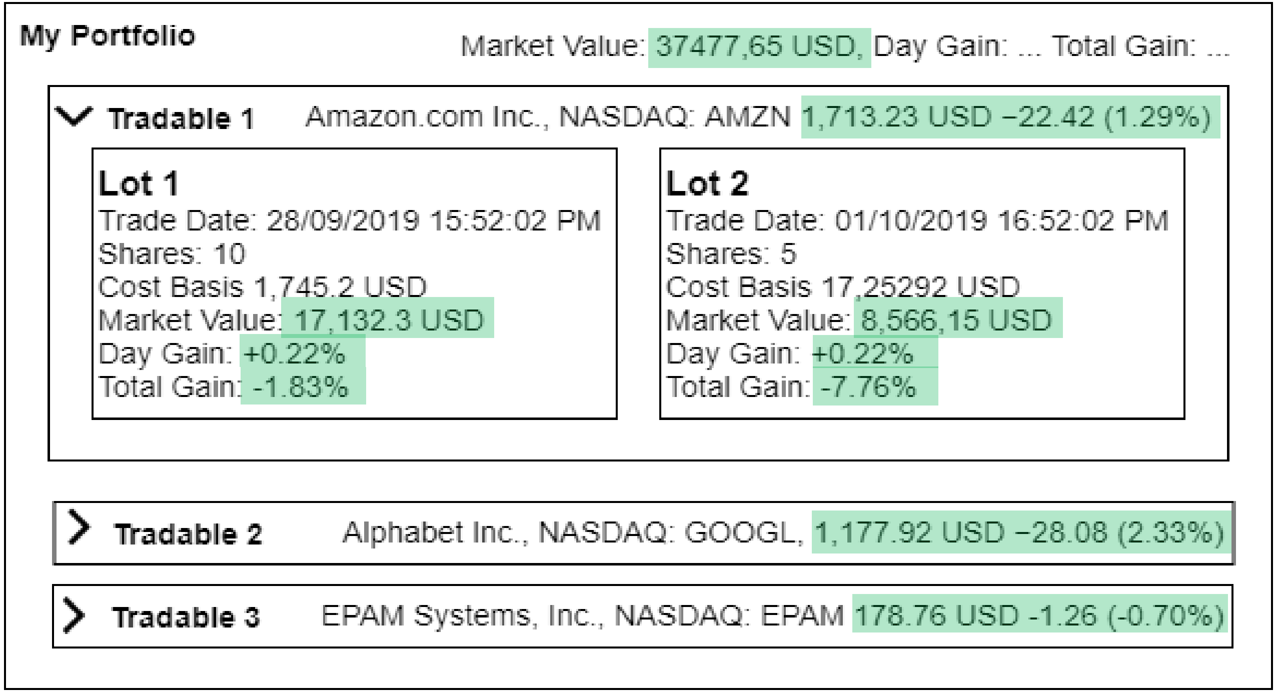
	REST	**GraphQL**
Endpoints	Endpoint per resource	Endpoint per API
Schema, type system	No formal definition by default, but many REST APIs conform to a specification like OpenAPI, JSON schema, ...	Strongly typed by default
Data transfer	HTTP	HTTP is the most common choice, but it is up to the implementor (e.g. MQTT).
Semantics	HTTP verbs (GET, PUT, POST, DELETE)	Queries (data fetching) ,mutations (data manipulations), subscriptions
Payload	Up to the implementor	Typically JSON (compresses well with GZIP)
Errors	HTTP status codes	Response with error fields. No sense of overall, HTTP-level success or failure.
Parameters	Query string parameters, HTTP request body	Query parameters
Response	Server-defined response	Client defines the shape of the response
Subscriptions	Not defined	Supported since June 2018
Pagination	Query string parameters	GraphQL query parameters
Documentation	Documentation per endpoint (e.g. OpenAPI)	Self documented via the GraphQL introspection system
Versioning	The entire API is versioned	More granular versioning possible. Specific fields can be deprecated / rolled in. Designed for API evolution.
Cachability	HTTP caching	Caching on client level, DB, memory. Client libraries like Apollo and Relay provide caching mechanism
Security	Cookies, HTTP basic auth, JWT, ...	Not defined on spec level, but there are many ways (authorization through context, schema/field level authorization, etc.)

Introducing GraphQL by example

GraphQL demo application: Financial portfolio manager

Features

- Watchlist (stocks, etc.)
- Manage my holdings
- Show latest prices
- Display live updates
- Calculate market value
- Day's / total gain



My Portfolio Market Value: 37477,65 USD, Day Gain: ... Total Gain: ...

Tradable 1 Amazon.com Inc., NASDAQ: AMZN 1,713.23 USD −22.42 (1.29%)

Lot 1
Trade Date: 28/09/2019 15:52:02 PM
Shares: 10
Cost Basis 1,745.2 USD
Market Value: 17,132.3 USD
Day Gain: +0.22%
Total Gain: -1.83%

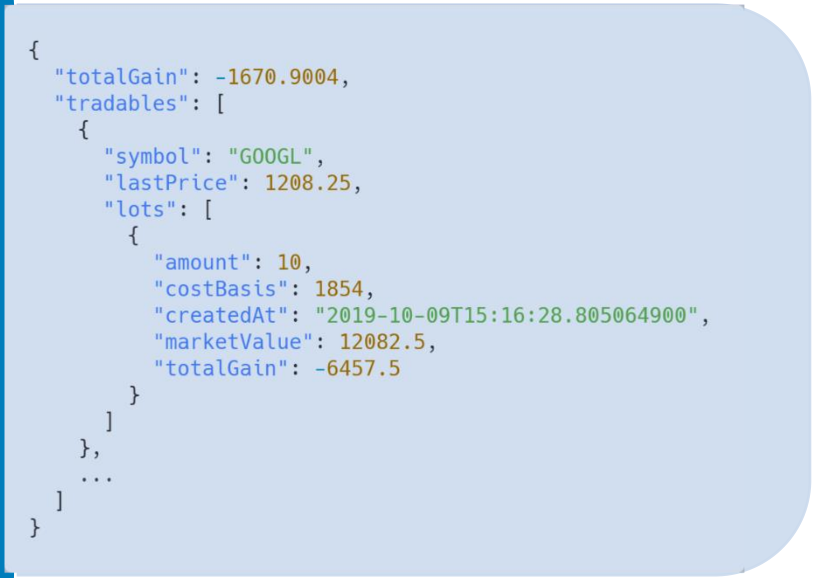
Lot 2
Trade Date: 01/10/2019 16:52:02 PM
Shares: 5
Cost Basis 17,25292 USD
Market Value: 8,566,15 USD
Day Gain: +0.22%
Total Gain: -7.76%

Tradable 2 Alphabet Inc., NASDAQ: GOOGL, 1,177.92 USD −28.08 (2.33%)

Tradable 3 EPAM Systems, Inc., NASDAQ: EPAM 178.76 USD -1.26 (-0.70%)

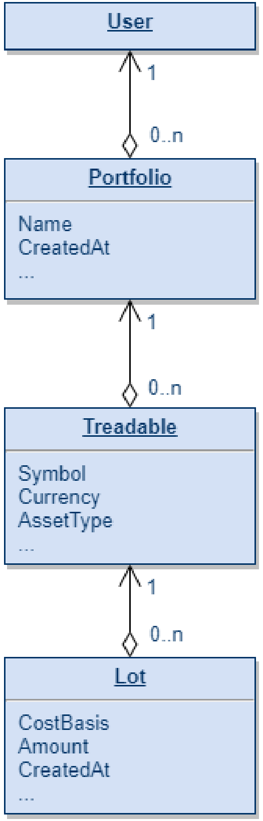
Financial portfolio manager – Data requirements

What are the
data requirements
of the UI?



```
{
  "totalGain": -1670.9004,
  "tradables": [
    {
      "symbol": "GOOGL",
      "lastPrice": 1208.25,
      "lots": [
        {
          "amount": 10,
          "costBasis": 1854,
          "createdAt": "2019-10-09T15:16:28.805064900",
          "marketValue": 12082.5,
          "totalGain": -6457.5
        }
      ]
    },
    ...
  ]
}
```

Do we need the same granularity in the results when the tradables or lots are expanded / collapsed on the UI?



User

1

0..n

Portfolio

Name
CreatedAt
…

1

0..n

Treadable

Symbol
Currency
AssetType
…

1

0..n

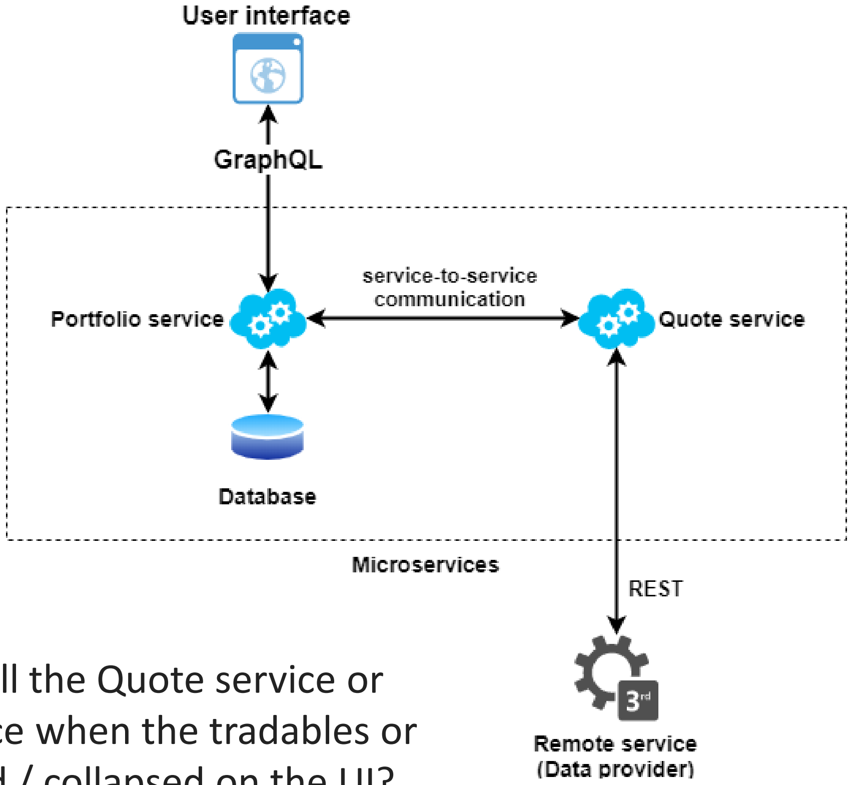
Lot

CostBasis
Amount
CreatedAt
…

Financial portfolio manager - Dependencies

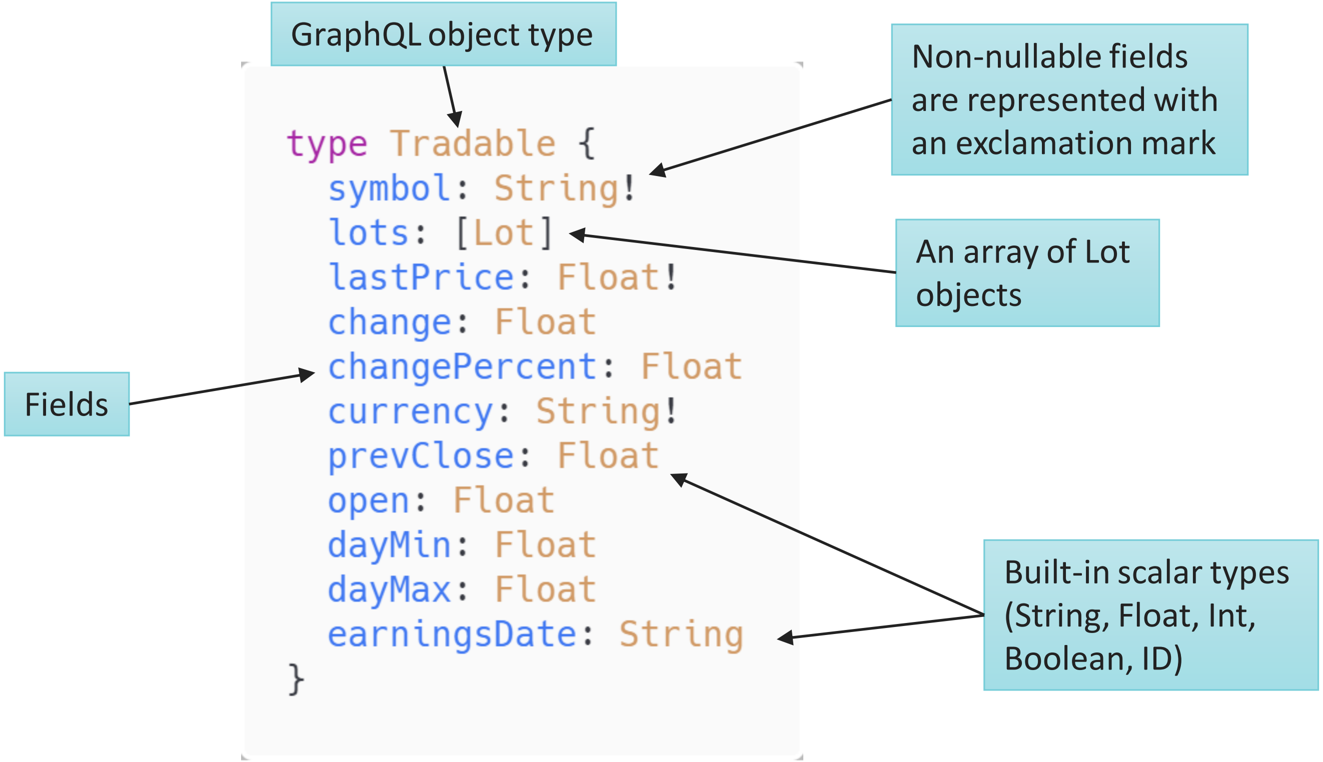
What could be the **dependencies in a potential microservices solution** when resolving the queries and mutations?

- **Portfolio**
  - DB (lots, tradables)
- **Live quotes**
  - Service-to-service call
  - Remote data provider service



Do we need to call the Quote service or the remote service when the tradables or lots are expanded / collapsed on the UI?

GraphQL type system: Object types



GraphQL object type

```
type Tradable {
  symbol: String!
  lots: [Lot]
  lastPrice: Float!
  change: Float
  changePercent: Float
  currency: String!
  prevClose: Float
  open: Float
  dayMin: Float
  dayMax: Float
  earningsDate: String
}
```

Non-nullable fields are represented with an exclamation mark

An array of Lot objects

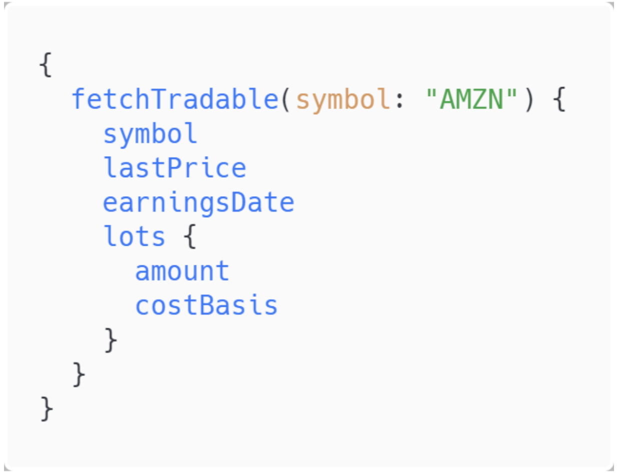
Fields

Built-in scalar types (String, Float, Int, Boolean, ID)

Additional features of the type system: enums, interfaces, unions, input types

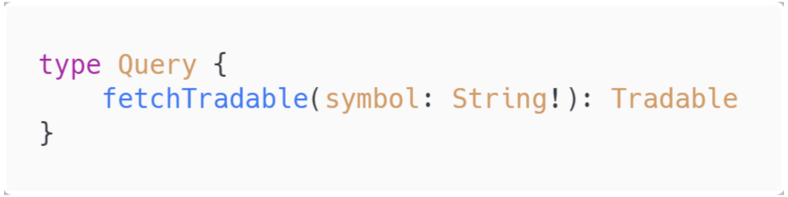
GraphQL type system: Query

To serve a **query operation** like this



```
{
  fetchTradable(symbol: "AMZN") {
    symbol
    lastPrice
    earningsDate
    lots {
      amount
      costBasis
    }
  }
}
```

... the server must have a **query type** like this in the schema:



```
type Query {
    fetchTradable(symbol: String!): Tradable
}
```

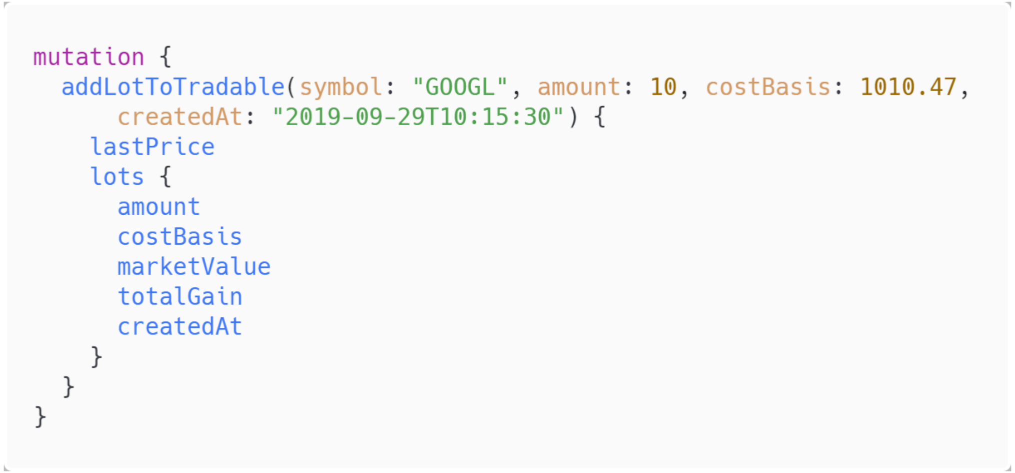
Every GraphQL service has **at least one query type**

A query type specifies an **entry point** and a **way for fetching data**

You can ask for **nested fields** in the return object

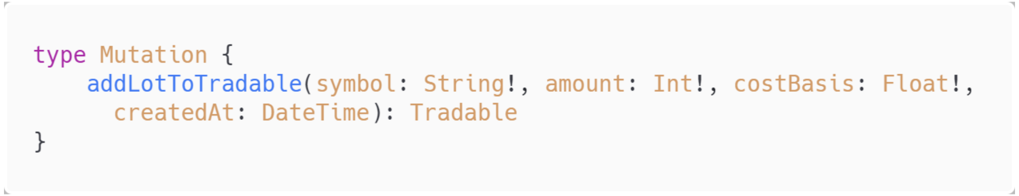
GraphQL type system: Mutation

To serve a **mutation operation** like this



```
mutation {
  addLotToTradable(symbol: "GOOGL", amount: 10, costBasis: 1010.47,
      createdAt: "2019-09-29T10:15:30") {
    lastPrice
    lots {
      amount
      costBasis
      marketValue
      totalGain
      createdAt
    }
  }
}
```

... the server must have a **mutation type** like this in the schema:



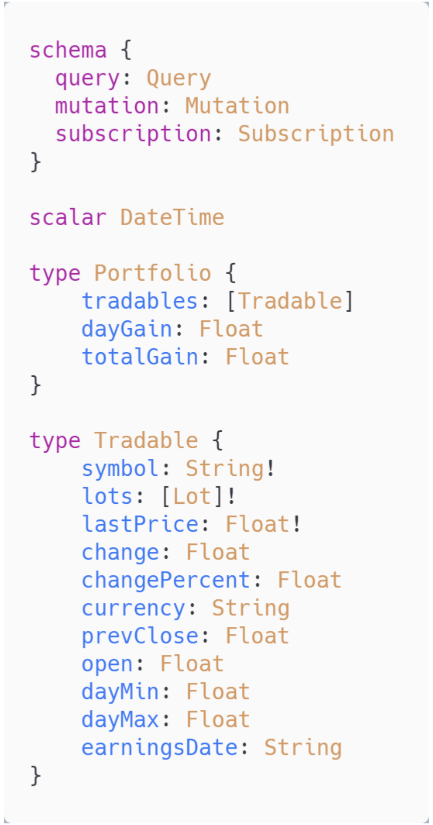
```
type Mutation {
    addLotToTradable(symbol: String!, amount: Int!, costBasis: Float!,
      createdAt: DateTime): Tradable
}
```

A GraphQL schema has at least one query type and **it may or may not have** a **mutation** type

Mutations are used for **modifying server-side data**

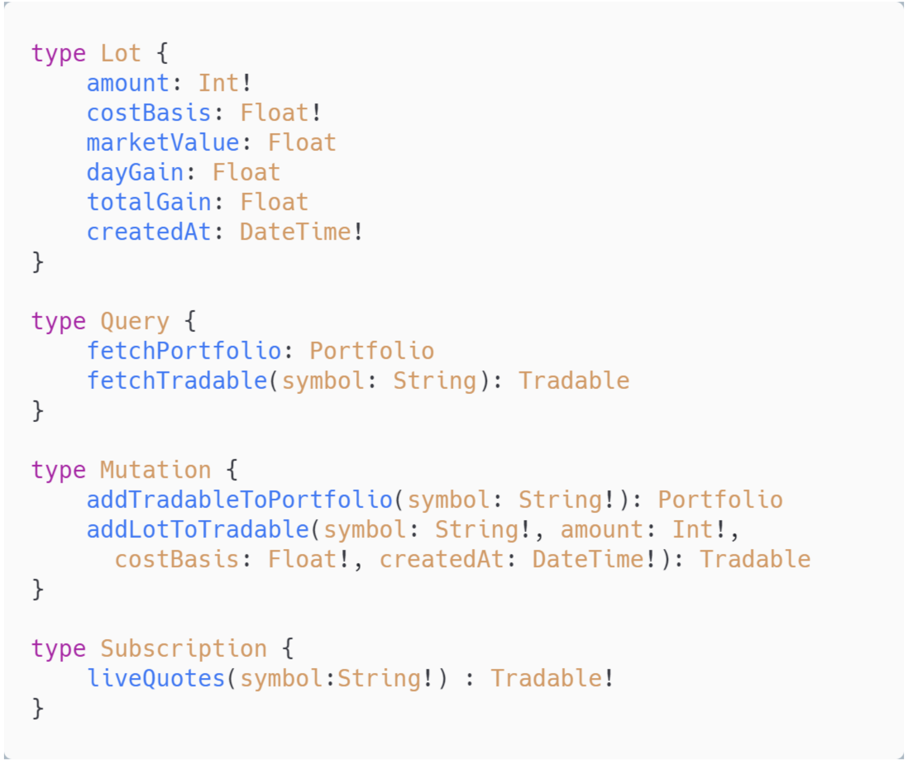
You can ask for **nested fields** in the return object

A complete GraphQL schema



```
schema {
  query: Query
  mutation: Mutation
  subscription: Subscription
}

scalar DateTime

type Portfolio {
    tradables: [Tradable]
    dayGain: Float
    totalGain: Float
}

type Tradable {
    symbol: String!
    lots: [Lot]!
    lastPrice: Float!
    change: Float
    changePercent: Float
    currency: String
    prevClose: Float
    open: Float
    dayMin: Float
    dayMax: Float
    earningsDate: String
}
```

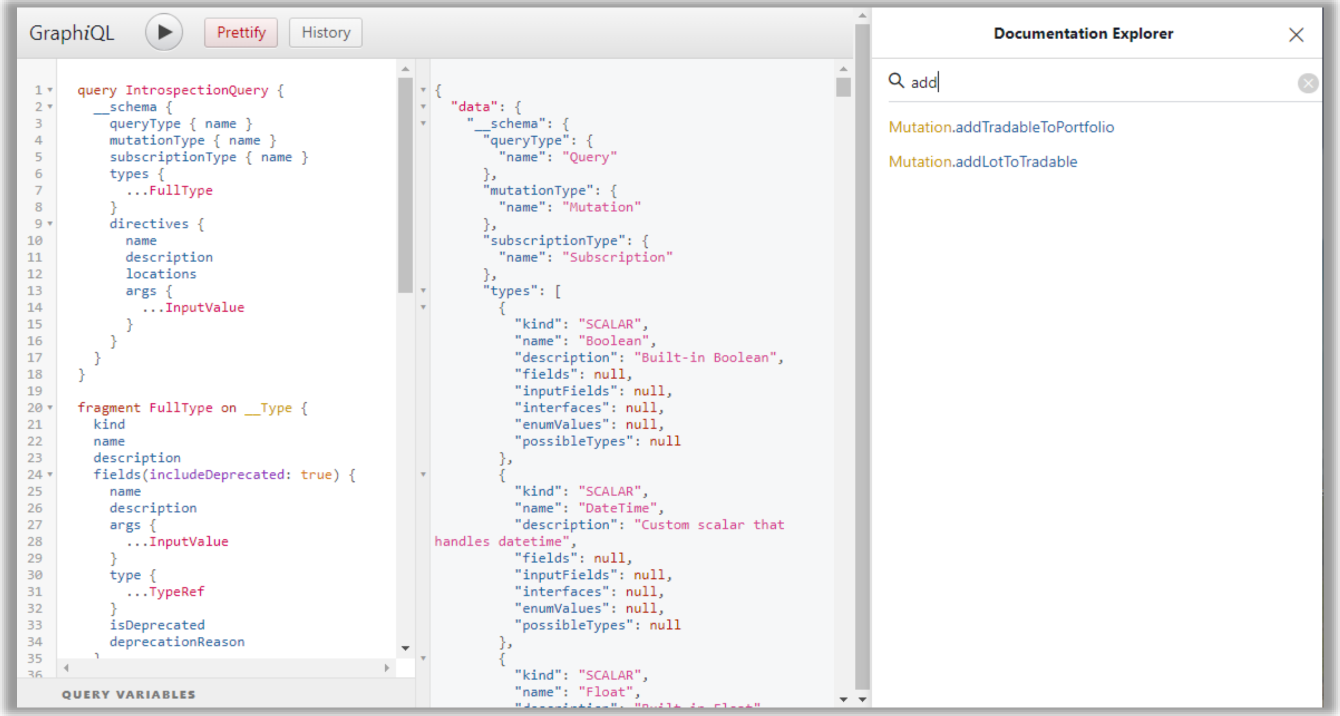
... continued:



```
type Lot {
    amount: Int!
    costBasis: Float!
    marketValue: Float
    dayGain: Float
    totalGain: Float
    createdAt: DateTime!
}

type Query {
    fetchPortfolio: Portfolio
    fetchTradable(symbol: String): Tradable
}

type Mutation {
    addTradableToPortfolio(symbol: String!): Portfolio
    addLotToTradable(symbol: String!, amount: Int!,
      costBasis: Float!, createdAt: DateTime!): Tradable
}

type Subscription {
    liveQuotes(symbol:String!) : Tradable!
}
```

GraphQL schema introspection system

- **Ask a GraphQL schema for information** about what types, queries, mutations and subscriptions it supports
- The **GraphiQL in-browser client** displays documentation by using the introspection system



```
query IntrospectionQuery {
  __schema {
    queryType { name }
    mutationType { name }
    subscriptionType { name }
    types {
      ...FullType
    }
    directives {
      name
      description
      locations
      args {
        ...InputValue
      }
    }
  }
}

fragment FullType on __Type {
  kind
  name
  description
  fields(includeDeprecated: true) {
    name
    description
    args {
      ...InputValue
    }
    type {
      ...TypeRef
    }
    isDeprecated
    deprecationReason
  }
```

```
{
  "data": {
    "__schema": {
      "queryType": {
        "name": "Query"
      },
      "mutationType": {
        "name": "Mutation"
      },
      "subscriptionType": {
        "name": "Subscription"
      },
      "types": [
        {
          "kind": "SCALAR",
          "name": "Boolean",
          "description": "Built-in Boolean",
          "fields": null,
          "inputFields": null,
          "interfaces": null,
          "enumValues": null,
          "possibleTypes": null
        },
        {
          "kind": "SCALAR",
          "name": "DateTime",
          "description": "Custom scalar that
handles datetime",
          "fields": null,
          "inputFields": null,
          "interfaces": null,
          "enumValues": null,
          "possibleTypes": null
        },
        {
          "kind": "SCALAR",
          "name": "Float",
          "description": "Built-in Float"
```

add

Mutation.addTradableToPortfolio

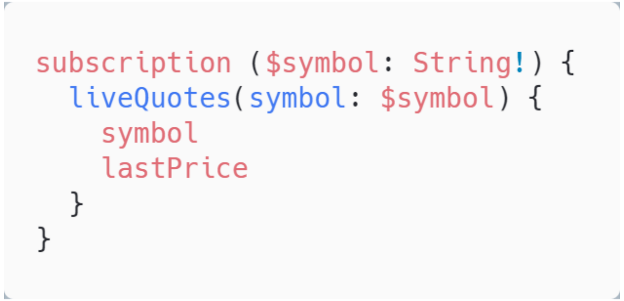
Mutation.addLotToTradable

DEMO

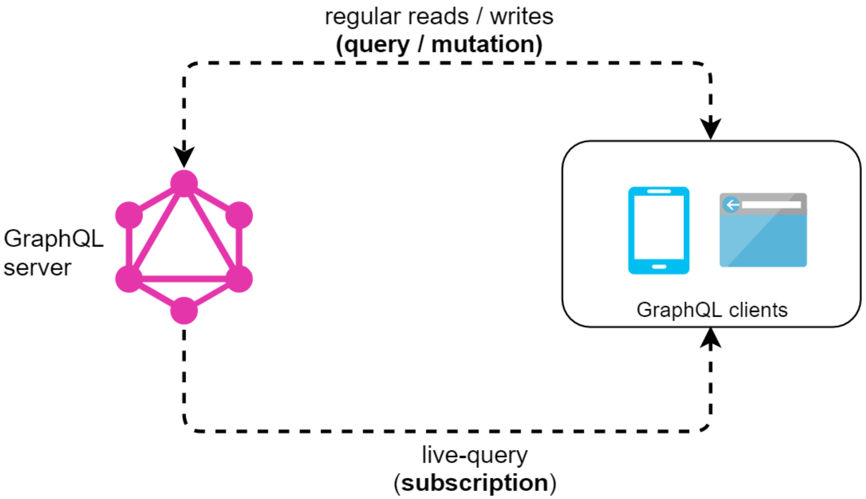
GraphQL subscriptions (since 2018)

- You can use **subscriptions** to provide a **more interactive UI**
  - E.g. chat, messaging, stock price live updates, ...

- Technical considerations
  - Transport via **websocket** is the most popular
  - GraphQL **types and resolvers can be reused**
  - **Limitations** on file descriptors, memory, CPU
  - Choose from polling, websocket, SSE, ... carefully
  - The server needs to **persist and re-evaluate** the query for potentially many subscribers



```
subscription ($symbol: String!) {
  liveQuotes(symbol: $symbol) {
    symbol
    lastPrice
  }
}
```



https://hackernoon.com/from-zero-to-graphql-subscriptions-416b9e0284f3

What about the Non-Functional Requirements (NFR)?

Recoverability
Availability
Maintainability
Regulatory
Integrity
Reliability
Recoverability
Scalability
Scalability
Usability
Manageability
Availability
Performance
Security
Interoperability
Serviceability
Serviceability
Regulatory
Maintainability
Regulatory
Data
Capacity
Security
Serviceability
Scalability
Capacity
Reliability
Interoperability
Environmental
Maintainability
Maintainability
Interoperability
Manageability
Reliability
Regulatory
Interoperability
Data
Environmental
Data
Manageability
Security
Data
Security
Performance
Performance
Reliability
Integrity
Integrity
Data
Usability
Availability
Manageability
Capacity
Capacity
Regulatory
Capacity
Availability
Performance
Availability
Reliability
Interoperability
Manageability
Serviceability
Scalability
Environmental
Serviceability
Security
Integrity
Recoverability
Maintainability
Security
Scalability
Data
Performance
Integrity
Regulatory
Environmental
Environmental
Performance
Scalability
Usability
Environmental
Usability
Availability
Scalability
Capacity
Manageability
Interoperability
Reliability
Recoverability
Maintainability
Performance
Recoverability
Usability
Integrity
Recoverability
Serviceability
Usability
Capacity

A few challenges related to NFRs...

Security

- **Common security vulnerabilities** (e.g. DDoS)
- **GraphQL specific** vulnerabilities
  - Query cost/complexity -> query analysis and rate limiting
  - Whitelist query patterns, use persisted queries
- **Authentication & authorization** (AAA)
  - Authentication via **non-GraphQL endpoints** → Frontend has to speak both GraphQL and non GraphQL.
  - Use **pure GraphQL**. Authentication via GraphQL itself. Use the context.
  - Build on **top of understood technologies** like JWT or OAuth

Caching

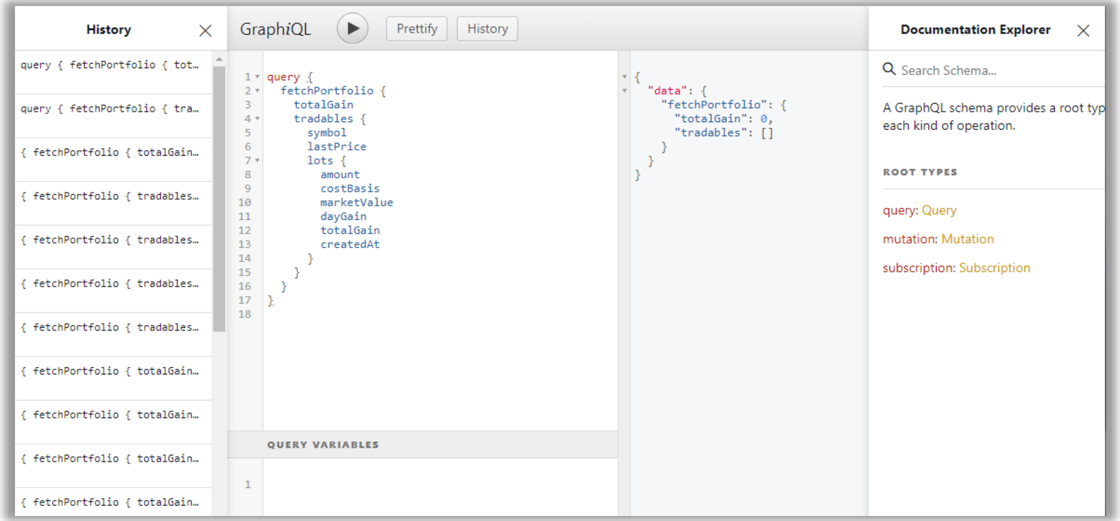
- While REST uses **transport level caching** (web servers), it is not appliable for GraphQL
- **Application level caching, query caching, client-side caching**. Apollo and Relay provide client side caching OOB.

Tools, libraries

Tools, Libraries

```
curl -X POST -H "Content-Type: application/json" --data '{ "query": "{ fetchPortfolio { tradables { symbol lastPrice
} } }" }' http://localhost:8080/graphql
```

- Any tool can be used that can set the following:
    - HTTP verb (POST)
    - URL of the GraphQL endpoint
    - Content type header (application/json)
    - Data sent (JSON)
- **Javascript** with fetch
- Application **libraries** e.g. Apollo
- **GraphiQL**: In-browser IDE



```
query { fetchPortfolio { tot…
query { fetchPortfolio { tra…
{ fetchPortfolio { totalGain…
{ fetchPortfolio { tradables…
{ fetchPortfolio { tradables…
{ fetchPortfolio { tradables…
{ fetchPortfolio { tradables…
{ fetchPortfolio { totalGain…
{ fetchPortfolio { totalGain…
{ fetchPortfolio { totalGain…
{ fetchPortfolio { totalGain…
```

```
query {
  fetchPortfolio {
    totalGain
    tradables {
      symbol
      lastPrice
      lots {
        amount
        costBasis
        marketValue
        dayGain
        totalGain
        createdAt
      }
    }
  }
}
```

```
{
  "data": {
    "fetchPortfolio": {
      "totalGain": 0,
      "tradables": []
    }
  }
}
```

A GraphQL schema provides a root typ
each kind of operation.

ROOT TYPES

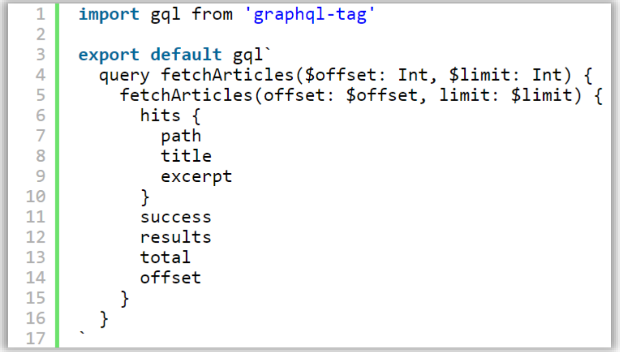
query: Query

mutation: Mutation

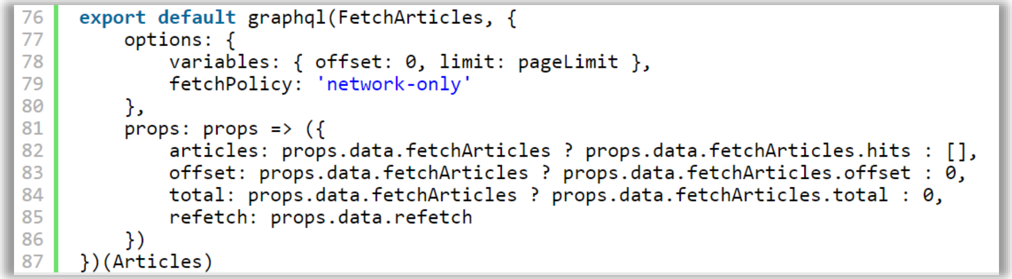
subscription: Subscription

Apollo

- Apollo GraphQL **client** and **view layer integration** for popular frontend frameworks: https://github.com/apollographql/apollo-client
- Apollo GraphQL **server**, that works with most Node.JS HTTP servers + serverless cloud frameworks: https://github.com/apollographql/apollo-server
- **Tools, utilities**: schema generator, mocking, schema stitching, etc.: https://github.com/apollographql/graphql-tools
- Enterprise grade **data graph platform** built on open source Apollo: https://www.apollographql.com
- Probably the biggest GraphQL **community** with events, blogs, forums
  - https://blog.apollographql.com/



```
import gql from 'graphql-tag'

export default gql`
  query fetchArticles($offset: Int, $limit: Int) {
    fetchArticles(offset: $offset, limit: $limit) {
      hits {
        path
        title
        excerpt
      }
      success
      results
      total
      offset
    }
  }
`
```



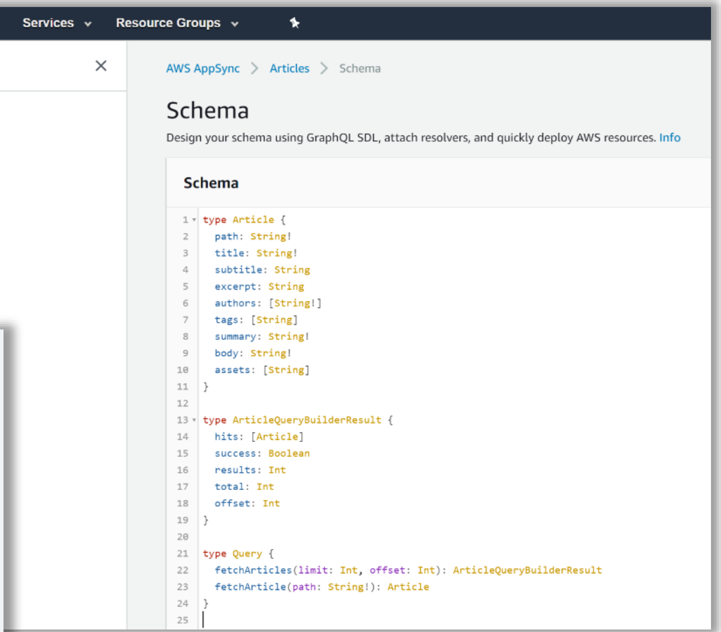
```
export default graphql(FetchArticles, {
    options: {
        variables: { offset: 0, limit: pageLimit },
        fetchPolicy: 'network-only'
    },
    props: props => ({
        articles: props.data.fetchArticles ? props.data.fetchArticles.hits : [],
        offset: props.data.fetchArticles ? props.data.fetchArticles.offset : 0,
        total: props.data.fetchArticles ? props.data.fetchArticles.total : 0,
        refetch: props.data.refetch
    })
})(Articles)
```

https://www.apollographql.com/

AWS AppSync

For AWS users and those who prefer managed service

- **Managed service** on AWS for building flexible GraphQL APIs
- **Data sources** and **resolvers** are used by AWS AppSync to translate GraphQL requests and fetch information from the underlying resources

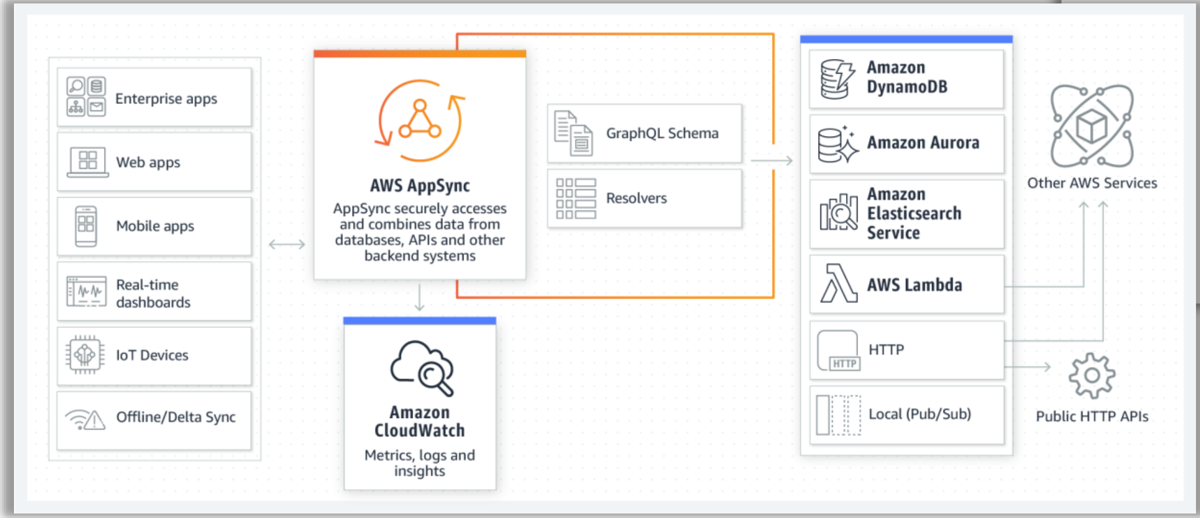


Schema

Design your schema using GraphQL SDL, attach resolvers, and quickly deploy AWS resources. Info

Schema

```
type Article {
  path: String!
  title: String!
  subtitle: String
  excerpt: String
  authors: [String!]
  tags: [String]
  summary: String!
  body: String!
  assets: [String]
}

type ArticleQueryBuilderResult {
  hits: [Article]
  success: Boolean
  results: Int
  total: Int
  offset: Int
}

type Query {
  fetchArticles(limit: Int, offset: Int): ArticleQueryBuilderResult
  fetchArticle(path: String!): Article
}
```

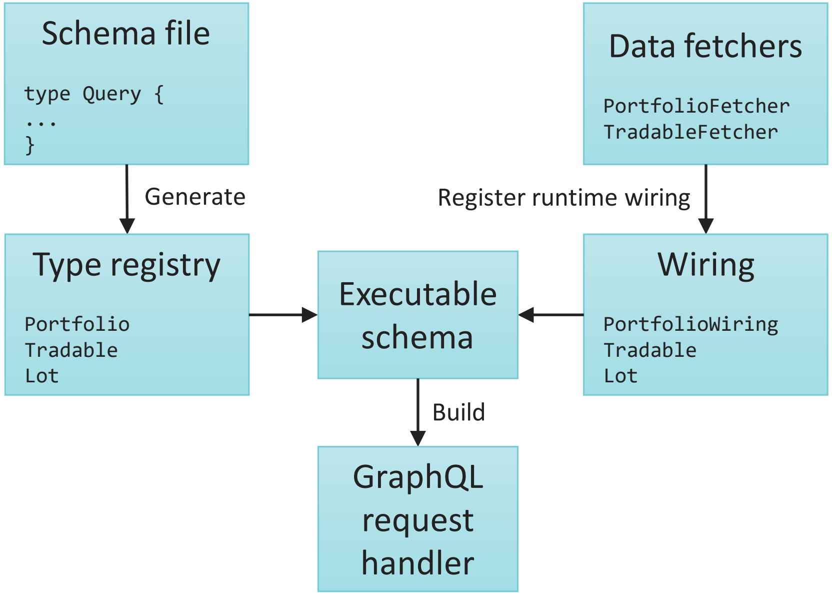


https://aws.amazon.com/appsync/

graphql-java

- **Java** (server) implementation for GraphQL
- It just deals with **executing queries**. No HTTP server, etc.
- **Exposing the API** needs to be done with another tool
- A **schema first** tool for graphql-java: GraphQL Java Tools

- Hint
  - **Get started** building a GraphQL Java microservice using a contract first approach:
    - **GraphQL Java Spring Boot**, or
    - **GraphQL light-4j**



https://www.graphql-java.com/tutorials/getting-started-with-spring-boot/

QUESTIONS