Introduction to GraphQL

Rohan Deshpande (@appwiz)
Principal Engineer, Amazon Web Services

Agenda

```
graphql {
    what
    why
    how
}
```

What is GraphQL

“GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data.”

https://graphql.org

GraphQL type system

- Object types
- Interfaces
- Unions
- Enumerations
- Fields
- Lists

- Scalars
  - String
  - Float
  - Int
  - Boolean
  - ID
  - Custom scalars e.g. Date

GraphQL operations

Queries
read data

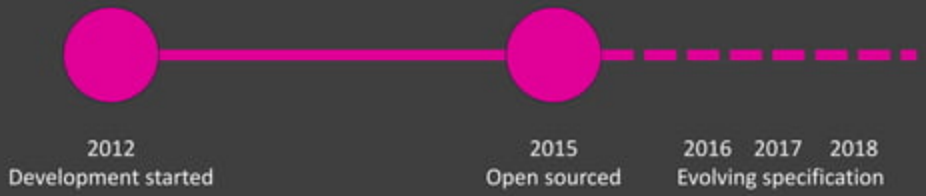
```
query {
  search(q: "name") {
    title
    author
  }
}
```

Mutations
write data

```
mutation {
  create(title: "book") {
    id
  }
}
```

Subscriptions
listen for data

```
subscription {
  onCreate {
    id
    title
  }
}
```



Timeline not to scale

Mobile usage in 2017

- 2+ billion smart phones[1]
- 1.6+ trillion hours[2]

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] https://www.appannie.com/en/insights/market-data/trillion-dollar-app-economy-only-beginning/

logo

book title

book author

★★★★ 55

paperback

10^{15}

only 2 left in stock

hardcover

14^{25}

only 16 left in stock

logo

book title

book author

★★★★ 55

paperback

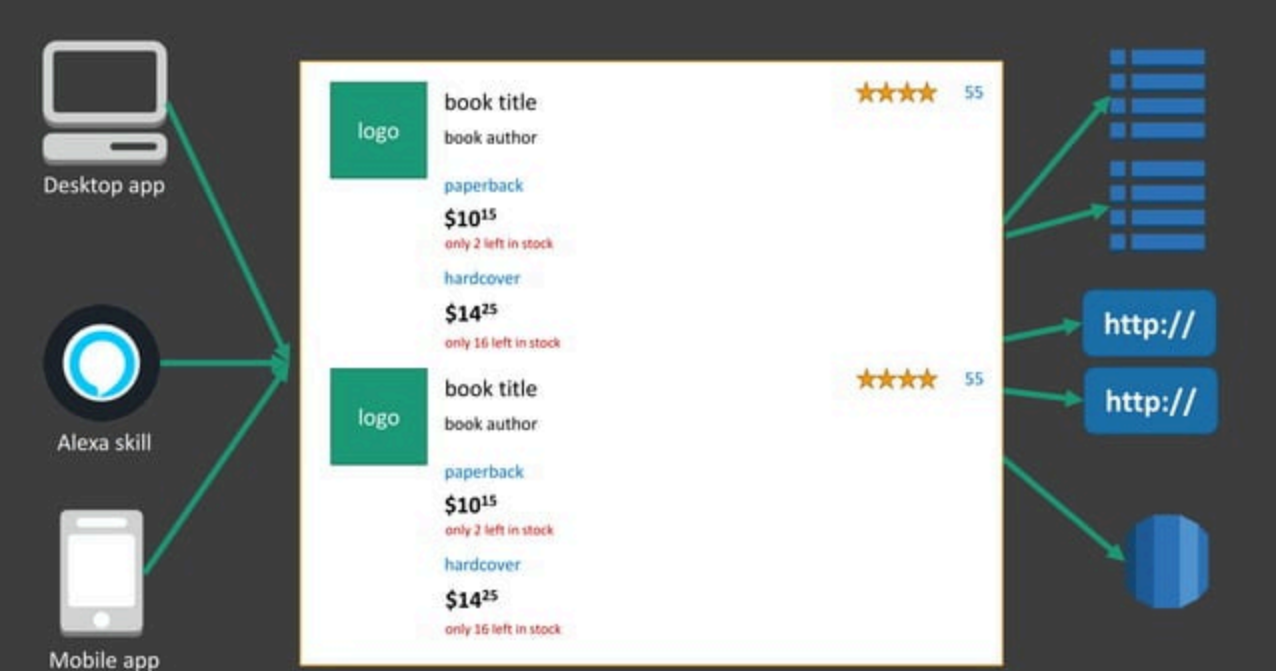
10^{15}

only 2 left in stock

hardcover

14^{25}

only 16 left in stock



A new philosophy for APIs

App data challenges

Data requirements vary across devices and become harder when multiple users share data

Users want instant access to data

Users want to continue using their apps even with low or no connectivity

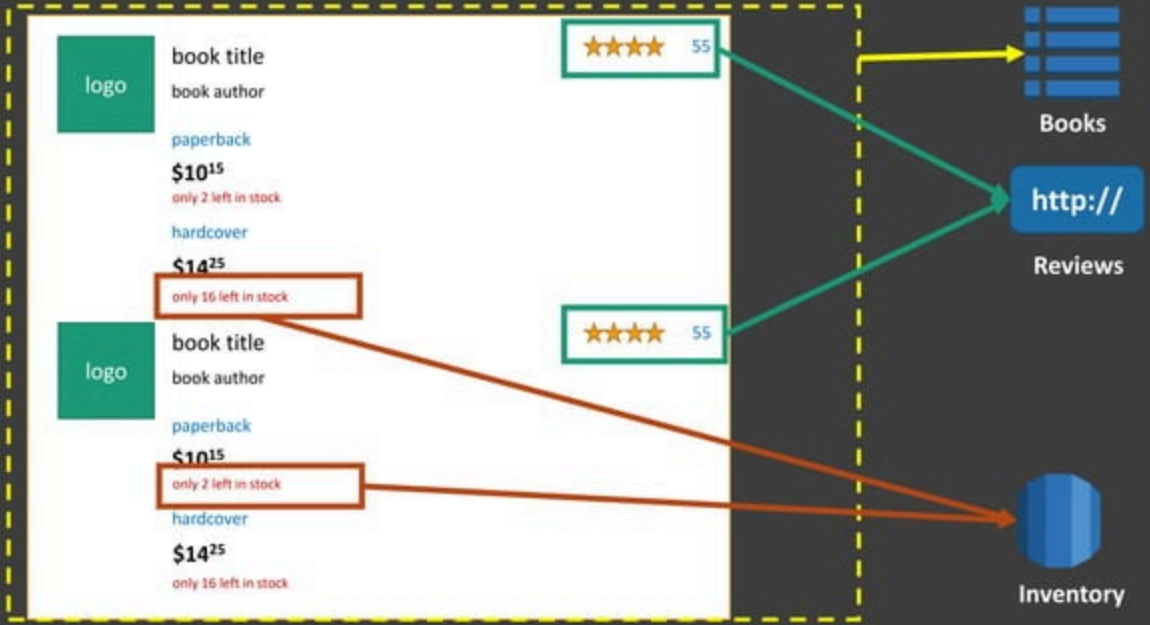
Building scalable data-driven apps without learning distributed systems concepts is hard

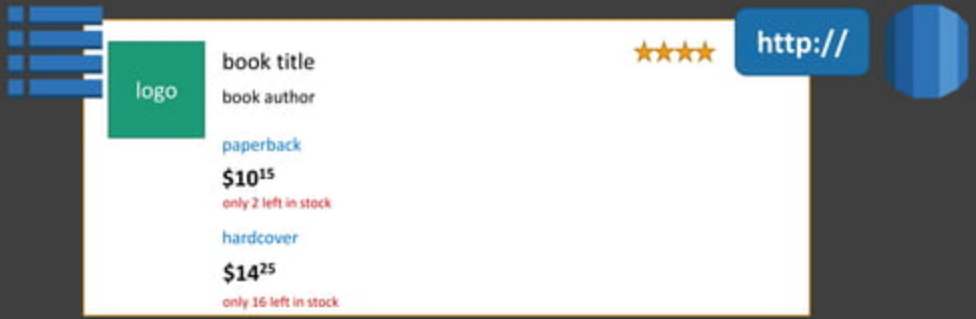
Why is GraphQL compelling?

Traditional data fetching

- ✓ Trivial to set up
- ✓ Standard HTTP Calls

- ✗ Relationships
- ✗ Lists with reduced information
- ✗ Query support
- ✗ Ordering and pagination
- ✗ Notifications





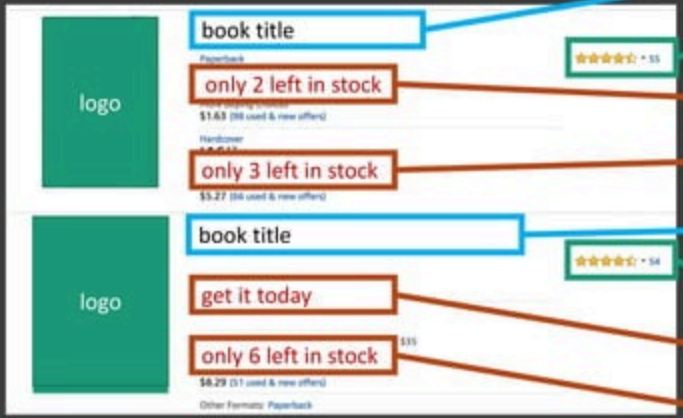

```
type Book {
  uuid: ID
  title: String
  in_stock: Int
  price: Float
  ...
}
```

```
getBook(uuid: "123")
```

✓ Type system

search(q="book")

return ("123", "456", "789"...)



getBook("123")
return ("123", "book title", "author", ["123a", "123b"])

getReviews("123")
return ("4.4", 55)

getInventory("123a")
return ("123", 2)

getInventory("123b")
return ("123", 16)

getBook("456")
return ("456", "book title", "author", ["456a" ,"456b"])

getReviews("456")
return ("456", "4.3", 54)

getInventory("456a")
return ("456", -1)

getInventory("456b")
return ("456", 11)

underfetching

search(q="book")

```
results: [
  book: {
    uuid: "123",
    title: "book title",
    author: "author",
    review: { rating: 4.4, count: 55 },
    items: [
      { uuid: "123a", remaining: 2 },
      { uuid: "123b", remaining: 16 }
    ]},
  book: {
    uuid: "456",
    title: "book title",
    author: "author",
    review: { rating: 4.3, count: 54 },
    items: [
      { uuid: "456a", remaining: -1 },
      { uuid: "456b", remaining: 11 }
    ]}
]
```

search(q="book")

```
results: [
  book: {
    uuid: "123",
    title: "book title",
    author: "author",
    review: { rating: 4.4, count: 55 },
    items: [
      { uuid: "123a", remaining: 2 },
      { uuid: "123b", remaining: 16 }
    ]},
  book: {
    uuid: "456",
    title: "book title",
    author: "author",
    review: { rating: 4.3, count: 54 },
    items: [
      { uuid: "456a", remaining: -1 },
      { uuid: "456b", remaining: 11 }
    ]}
]
```

mobile_search(q="book")

```
results: [
  book: {
    uuid: "123",
    title: "book title",
    author: "author",
    review: { rating: 4.4, count: 55 },
    uuids: [ "123a", "123b" ]
  },
  book: {
    uuid: "456",
    title: "book title",
    author: "author",
    review: { rating: 4.3, count: 54 },
    uuids: [ "456a", "456b" ]
  }
]
```

overfetching

What we want

- Single endpoint to access data
- One request
- Only the fields that are asked

"exactfetching"

- ✓ Type system
- ✓ **GraphQL Queries**

- ✓ Type system
- ✓ GraphQL Queries
- ✓ **GraphQL Mutations**

- ✓ Type system
- ✓ GraphQL Queries
- ✓ GraphQL Mutations
- ✓ **GraphQL Subscriptions**

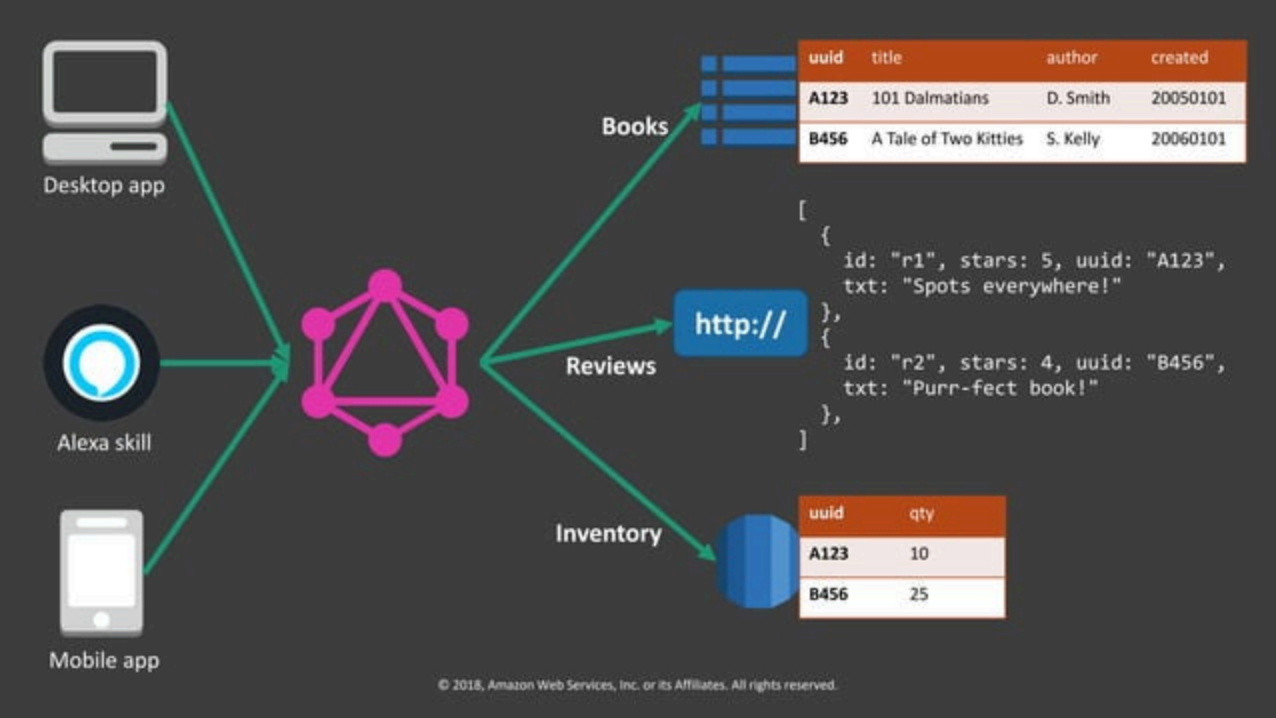
Open specification

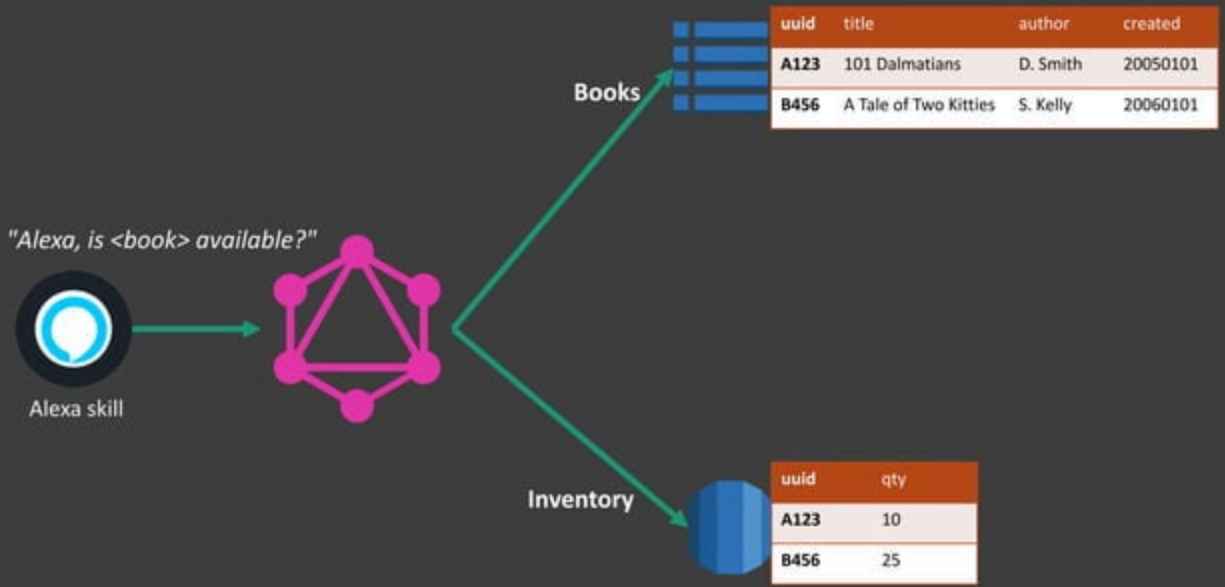
http://facebook.github.io/graphql/draft/

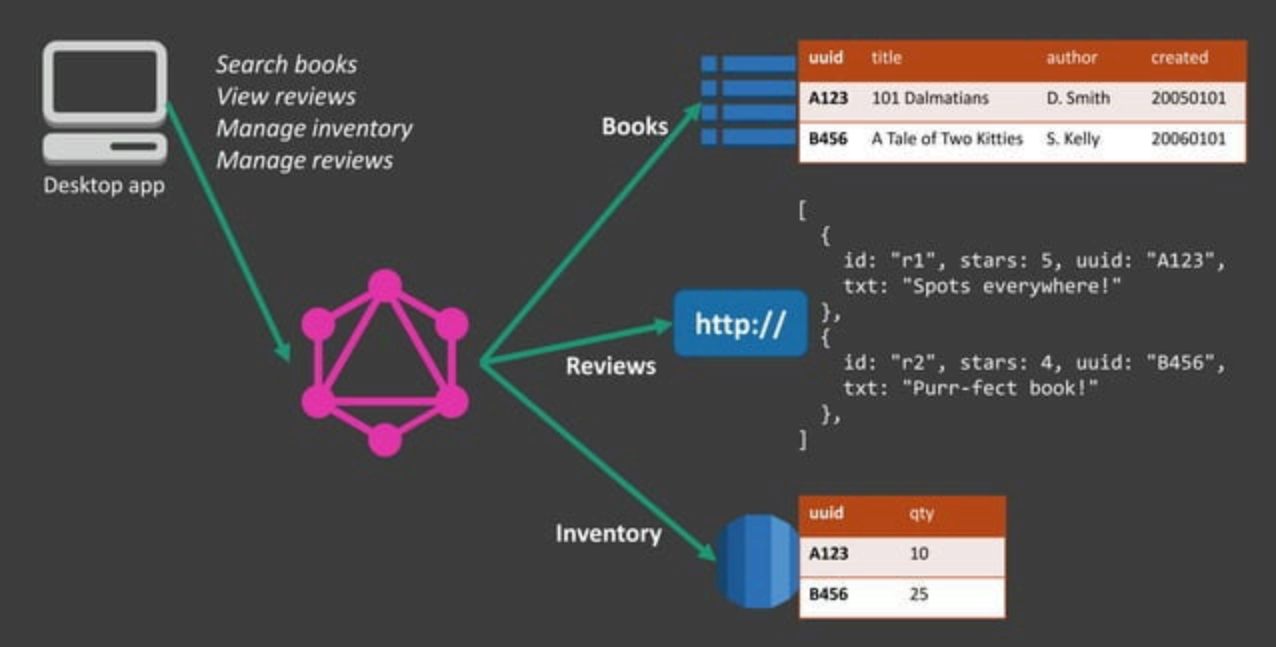
Features of GraphQL

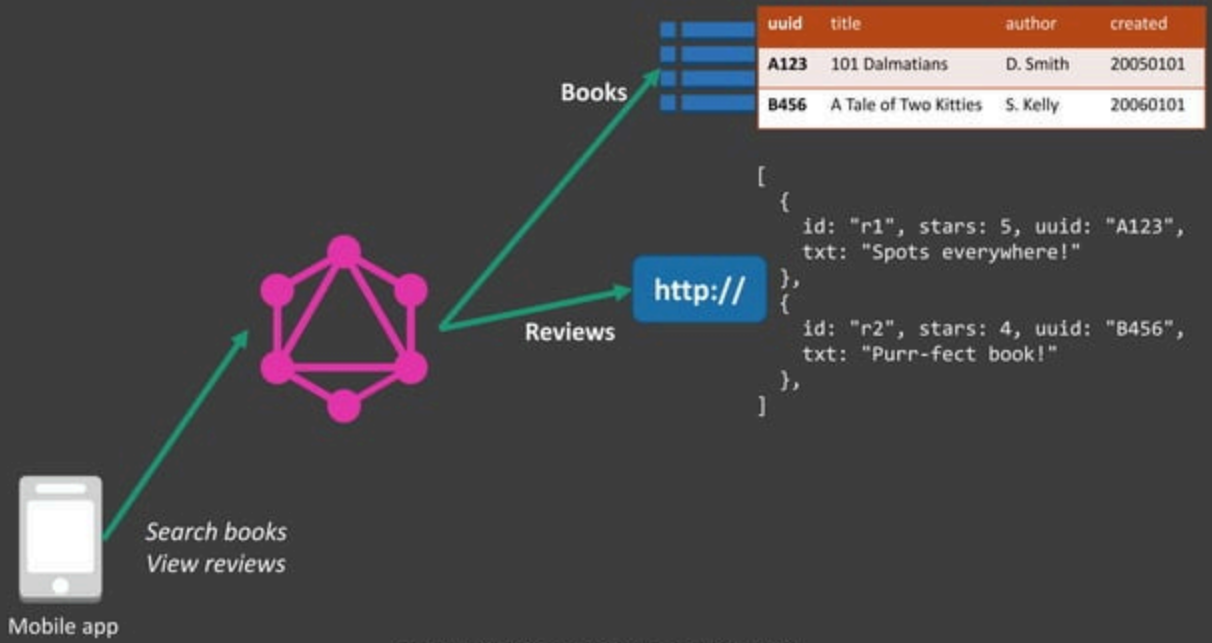
- Strongly typed
- Code generation and introspection
- Queries, mutations, subscriptions
- Transport agnostic
- Client-specified shape of response
- Efficient

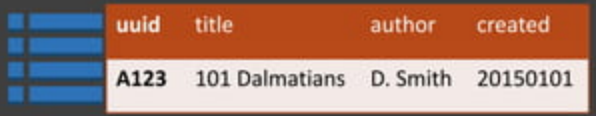
Build a GraphQL API





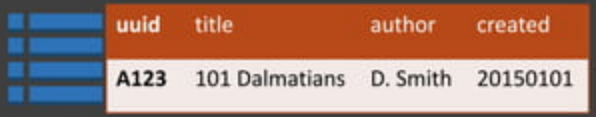






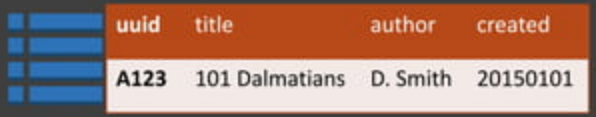
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {

}
```



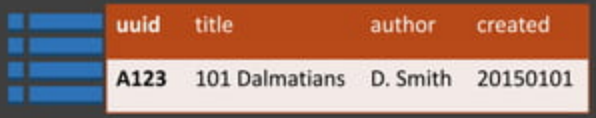
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {

}
```



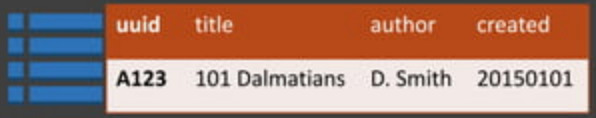
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {

}
```



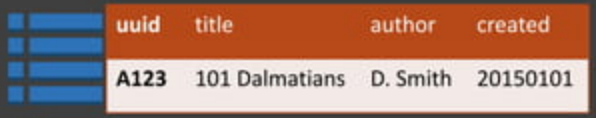
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {
  uuid: ID
}
```



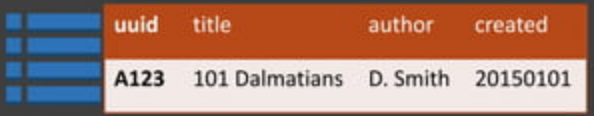
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {
  uuid: ID
}
```



uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {
  uuid: ID
}
```



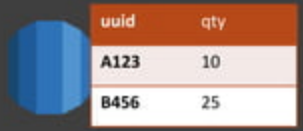
uuid	title	author	created
A123	101 Dalmatians	D. Smith	20150101

```
type Book {
  uuid: ID
  title: String
  author: String
}
```

```
[
  {
    id: "r1", stars: 5, uuid: "A123",
    txt: "Spots everywhere!"
  },
  {
    id: "r2", stars: 4, uuid: "B456",
    txt: "Purr-fect book!"
  },
]
```

http://

```
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
```



uuid	qty
A123	10
B456	25

```
type Inventory {
  uuid: ID
  qty: Int
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
}
```

Books

uuid	title	author	created
A123	101 Dalmatians	D. Smith	20050101
B456	A Tale of Two Kitties	S. Kelly	20060101

```
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
```

Reviews

http://

```
[
  {
    id: "r1", stars: 5, uuid: "A123",
    txt: "Spots everywhere!"
  },
  {
    id: "r2", stars: 4, uuid: "B456",
    txt: "Purr-fect book!"
  },
]
```

```
type Inventory {
  uuid: ID
  qty: Int
}
```

Inventory

uuid	qty
A123	10
B456	25

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
}

type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
```

Books

uuid	title	author	created
A123	101 Dalmatians	D. Smith	20050101
B456	A Tale of Two Kitties	S. Kelly	20060101

Reviews

http://

```
[
  {
    id: "r1", stars: 5, uuid: "A123",
    txt: "Spots everywhere!"
  },
  {
    id: "r2", stars: 4, uuid: "B456",
    txt: "Purr-fect book!"
  },
]
```

Inventory

uuid	qty
A123	10
B456	25

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
```

Books

uuid	title	author	created
A123	101 Dalmatians	D. Smith	20050101
B456	A Tale of Two Kitties	S. Kelly	20060101

Reviews http://

```
[
  {
    id: "r1", stars: 5, uuid: "A123",
    txt: "Spots everywhere!"
  },
  {
    id: "r2", stars: 4, uuid: "B456",
    txt: "Purr-fect book!"
  },
]
```

Inventory

uuid	qty
A123	10
B456	25

Reading data

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
```

```
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
search(q: "tale") {
  uuid
  title
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
search(q: "tale") {
  uuid
  title
}
```

```
[
  {
    "uuid": "B456"
    "title": "A Tale of Two Kitties"
  }
]
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
getBook(uuid: "A123") {
  uuid
  title
  reviews {
    id
    txt
    stars
  }
  inventory {
    qty
  }
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
{
  "uuid": "A123",
  "title": "101 Dalmatians",
  "reviews": [
    id: "r1",
    txt: "Spots everywhere!",
    stars: 5
  ],
  "inventory" {
    "qty": 10
  }
}
```

Writing data

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
```

```
type MyMutation {
  addStar(num: Int): Review
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
```

```
addStar(num: 1) {
  stars
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
```

```
addStar(num: 1) {
  stars
}
```

```
{
  stars: 55
}
```

Listening for data

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
```

```
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
onReview(uuid: "A123") {
  uuid
  id
  stars
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
onReview(uuid: "A123") {
  uuid
  id
  stars
}
```

<listen>

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
onReview(uuid: "A123") {
  uuid
  id
  stars
}
```

```
{
  "uuid": "A123",
  "id": "r576",
  "stars": 54
}
```

```
{
  "uuid": "A123",
  "id": "r9845",
  "stars": 55
}
```

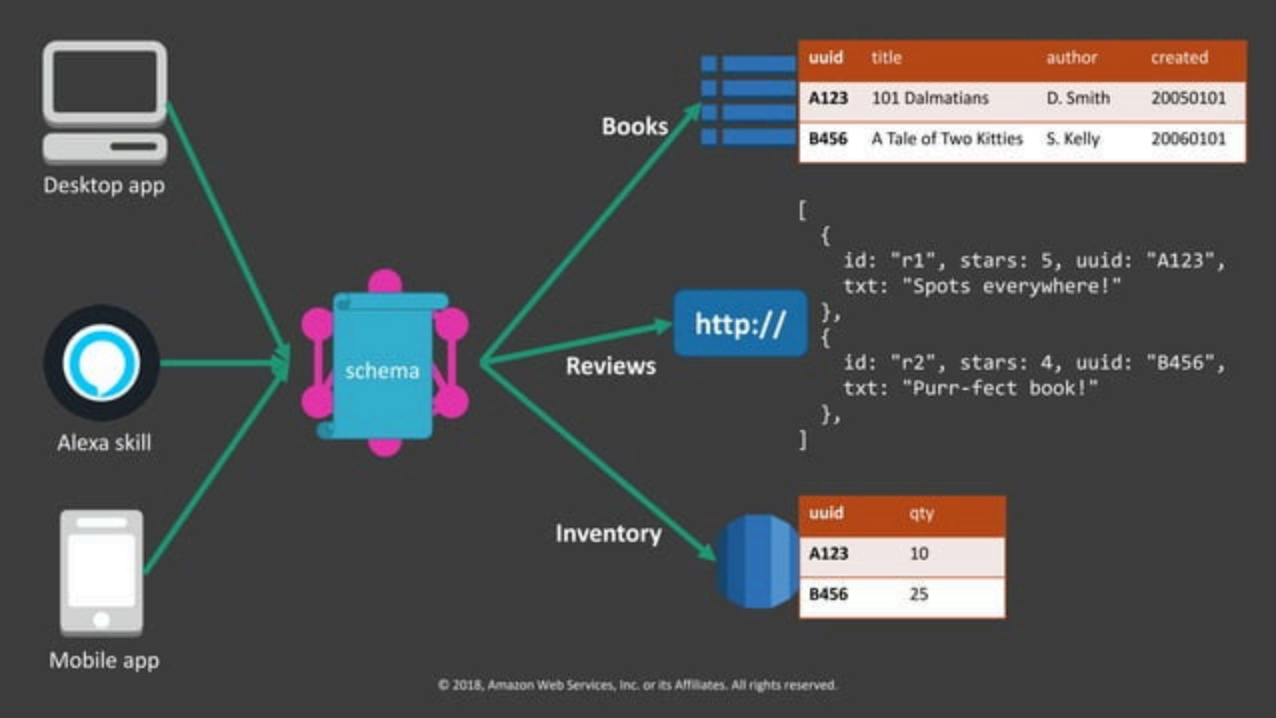
```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
```

```
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
type Review {
  id: ID
  uuid: String
  stars: Int
  txt: String
}
type Inventory {
  uuid: ID
  qty: Int
}
```

```
type MyQuery {
  search(q: String): [Book]
  getBook(uuid: ID): Book
}
type MyMutation {
  addStar(num: Int): Review
}
type MySubscription {
  onReview(uuid: ID): Review
}
```

```
schema {
  query: MyQuery
  mutation: MyMutation
  subscription: MySubscription
}
```



How are queries executed?

```
type MyQuery {
  getBook(uuid: ID): Book
}
```

Books

```
type MyQuery {
  getBook(uuid: ID): Book
}
```

resolver
f(x)

Books

```
type MyQuery {
  getBook(uuid: ID): Book
}
```

resolver
f(x)

```
var client = new DocumentClient();
var params = {
    TableName: "Books"
};
client.getItem(params, onItem);

function onItem(err, data) {
    if (data) {
      //...
    }
}
```

Books

```
type MyQuery {
  getBook(uuid: ID): Book
}
```

GetBookResolver

Books

http://

Reviews

Inventory

```
type MyQuery {
  getBook(uuid: ID): Book
}

type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
```

GetBookResolver

Books

http://

Reviews

Inventory

```
type MyQuery {
  getBook(uuid: ID): Book
}

type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
```

GetBookResolver

Books

http://

Reviews

GetReviewsResolver

Inventory

```
type MyQuery {
  getBook(uuid: ID): Book
}

type Book {
  uuid: ID
  title: String
  author: String
  reviews: [Review]
  inventory: Inventory
}
```

GetBookResolver

Books

GetReviewsResolver

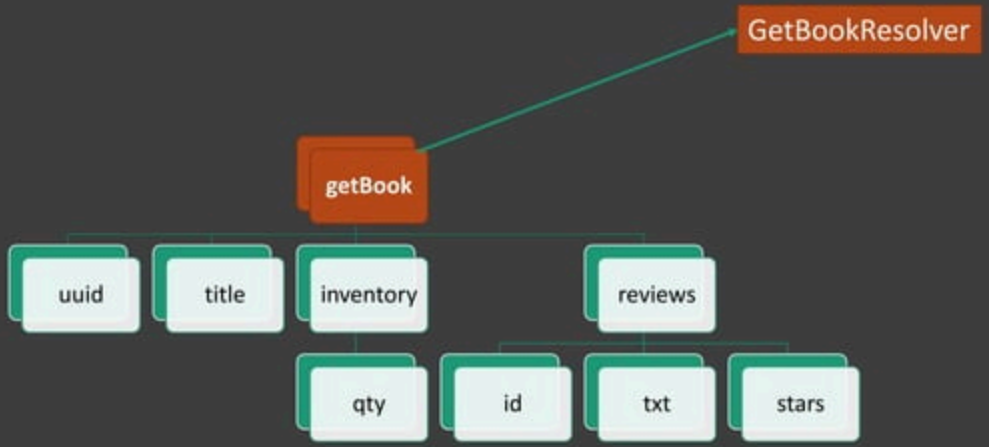
http://

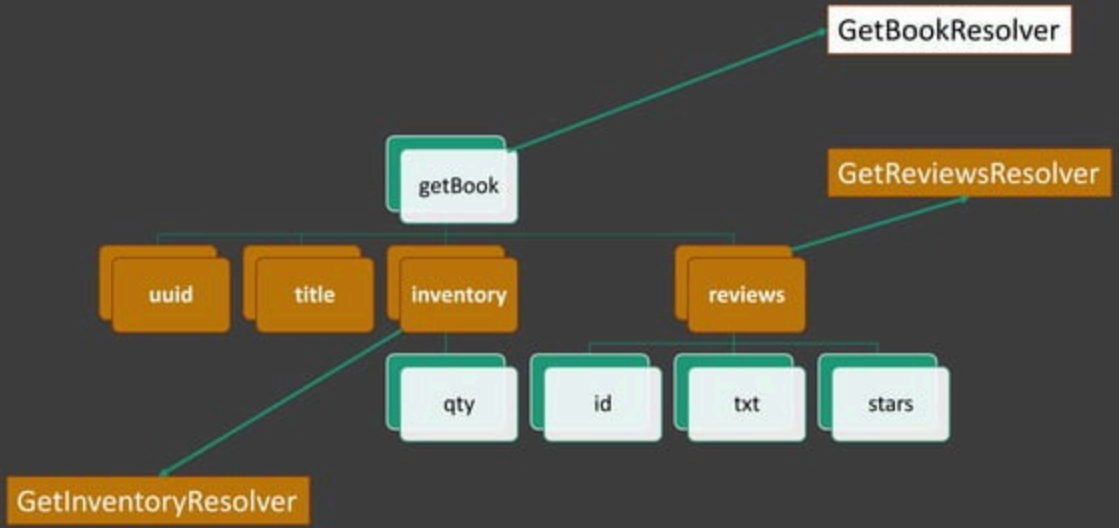
Reviews

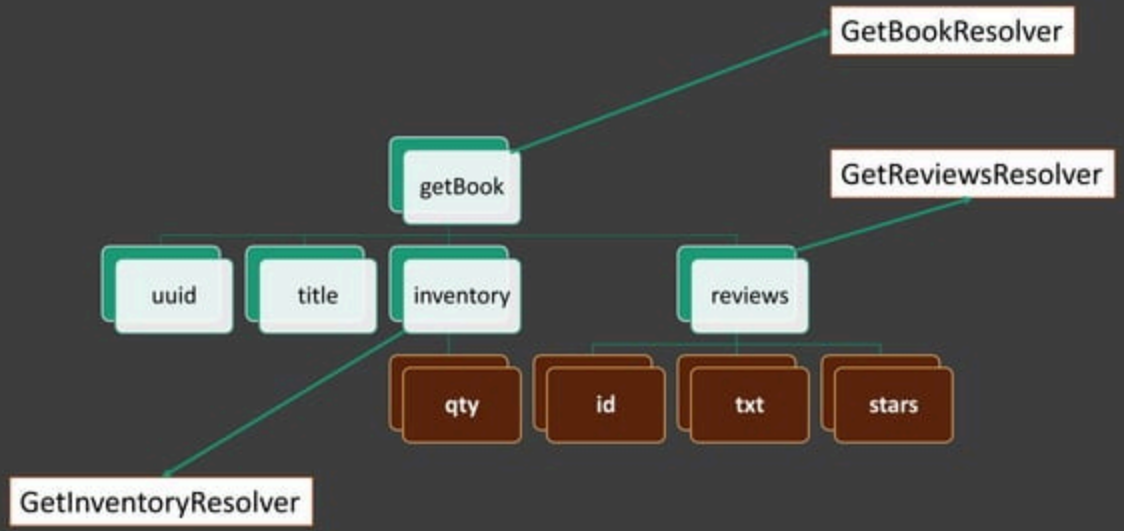
GetInventoryResolver

Inventory

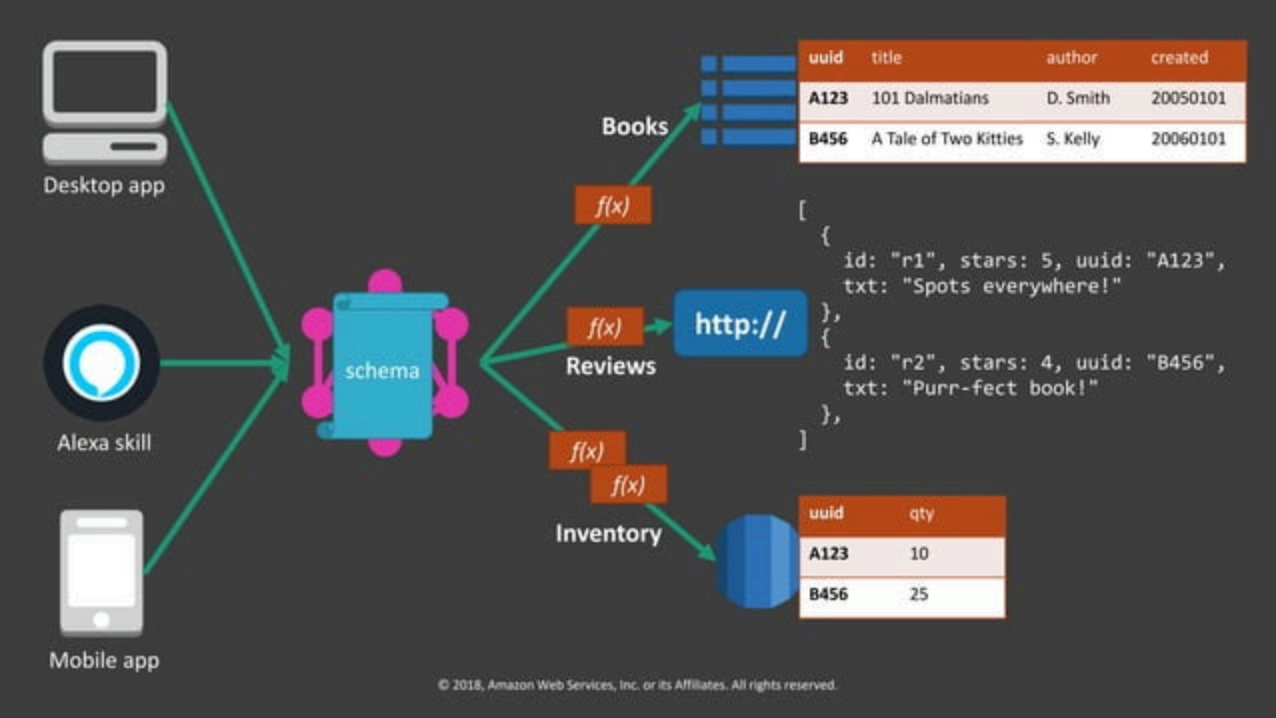
```
getBook(uuid: "A123") {
  uuid
  title
  reviews {
    id
    txt
    stars
  }
  inventory {
    qty
  }
}
```







```
getBook(uuid: "A123") {
  uuid
  title
  inventory
  reviews
}
```

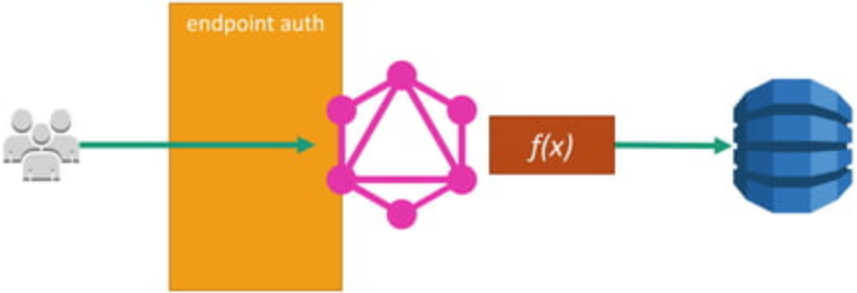


How do I

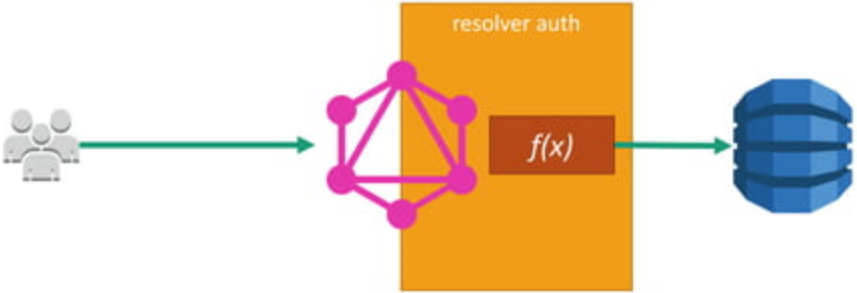
Run GraphQL in production

GraphQL sounds legit!
I want to run my own GraphQL server!!

Security



Security



Security

```
getBook(uuid: "A123") {
  uuid
  title
  qty
}
```

```
{
  uuid: "A123"
  title: "101 Dalmatians"
  qty: null
}
```

Nested queries

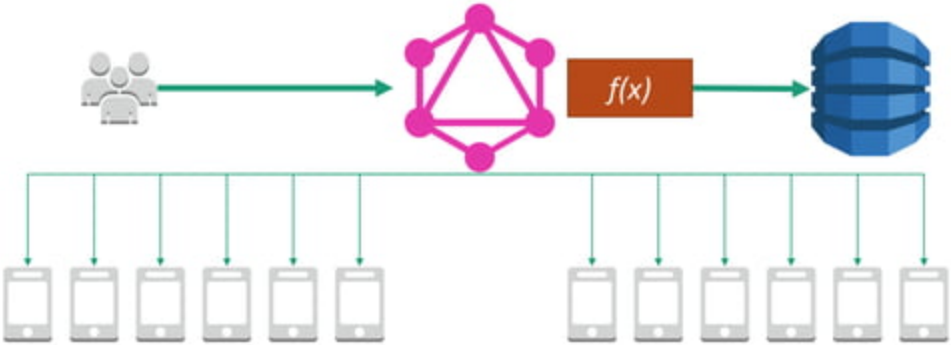
```
query {
  friends(id: "123") {
    id
    name
    friends {
      id
      name
      friends {
        id
        name
        friends {
          ...
        }
      }
    }
  }
}
```

Query Execution

- Bounding
- Throttling
- Batching

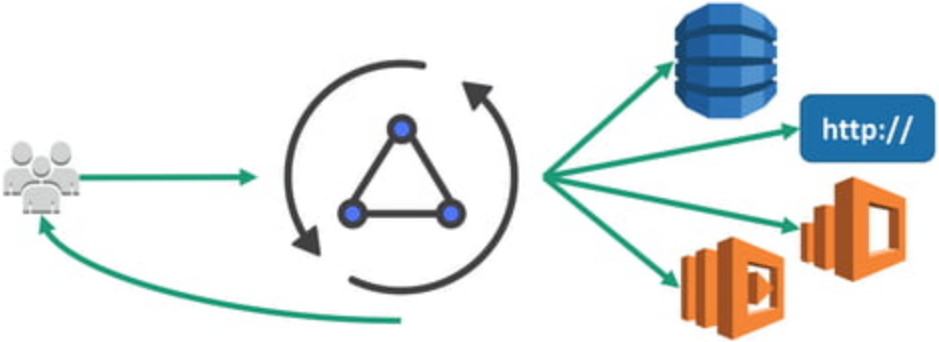
```
getBook(uuid: "A123") {
  uuid
  title
  reviews {
    id
    txt
    stars
  }
  price {
    currency
  }
  inventory {
    qty
  }
}
```

Scaling subscriptions



Focus on Apps
Not Infrastructure

AWS AppSync – managed GraphQL



https://aws.amazon.com/appsync

AWS AppSync

Managed Serverless GraphQL service

Connect to data sources in your account

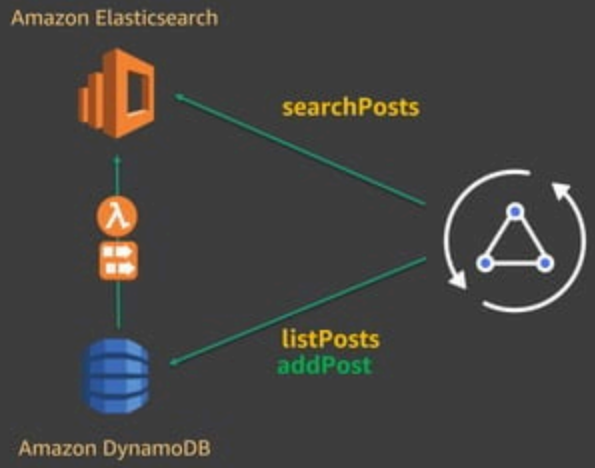
Add data sync, real-time and offline capabilities for any data source or API

GraphQL façade for any AWS service

Conflict detection and resolution in the cloud

Enterprise security features: IAM, Cognito, OIDC, API keys

Mix/Match Data Sources on GraphQL Types




```
type Query {
    listPosts: [Post]
    searchPosts: [Post]
}

type Mutation {
    addPost: Post
}

type Post {
    id: ID!
    content: String
    description: String
    ups: Int
    downs: Int
}
```

AWS AppSync benefits

Clients receive the data they ask for. Nothing more, nothing less

Get many resources from many data sources with a single request

Self-documenting APIs with Introspection

React Native, Android, iOS, and Web (JS) using the Apollo GraphQL client

Data persistence across application restarts

Write-through mutations with Optimistic UI

How does AppSync work?

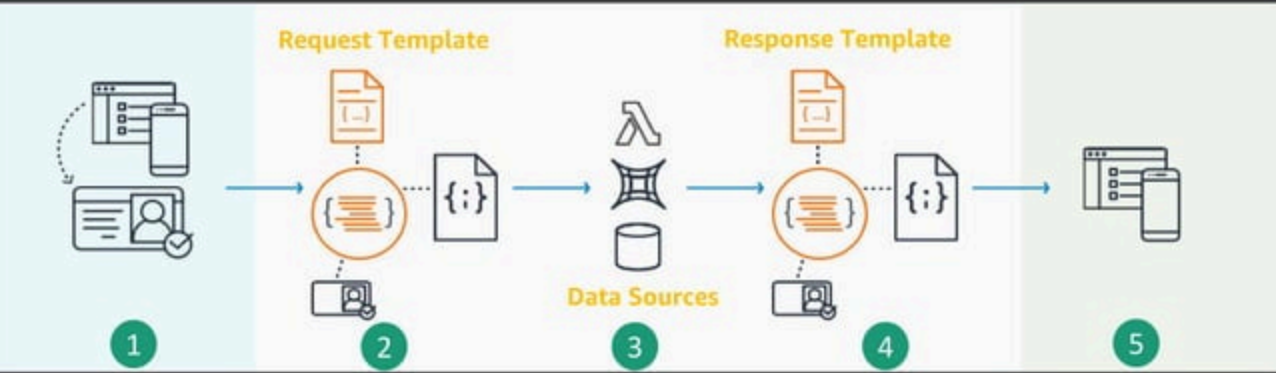
Create, Import or Upload GraphQL Schema

Connect Data Sources

AppSync Updates Data in Real Time and Manages Data when Offline

Data flow and security

GraphQL data flow in AWS AppSync



Fine-grained access control

Get token

User logs into their application using Amazon Cognito User Pools, receiving a token containing identity information.

Send request

A GraphQL operation is invoked from an AWS AppSync client, sending the logged-in user's token as part of the request.

Conditional check

The request mapping template adds a conditional check with the user's identity.

Run operation

If the conditional check matches the specified field in the database (username == author) then the operation succeeds.

Access control checks

Simple

Only users in certain groups can perform actions

```
#if($hasPermission || $ctx.result.public == 'yes')
    $utils.toJson($ctx.result)
#else
    $utils.unauthorized()
#end
```

Only users in certain groups can perform actions

```
#if($ctx.result["Owner"] == $ctx.identity.username)
    $utils.toJson($context.result);
#else
    $utils.unauthorized()
#end
```

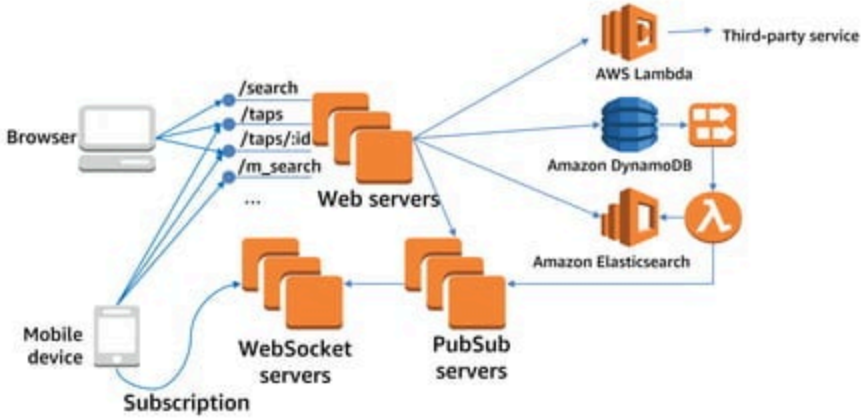
Advanced

Only users in certain groups can perform actions

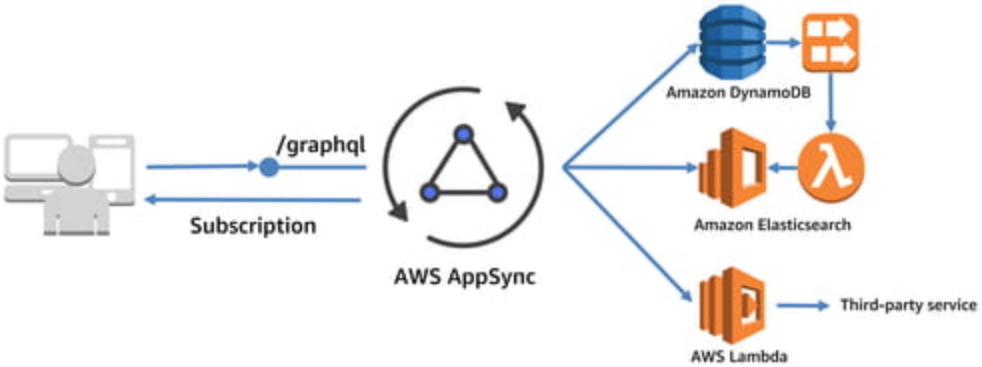
```
#foreach($group in $ctx.identity.claims.get("cognito:groups"))
    #if($group == "Admin")
        #set($inCognitoGroup = true)
    #end
#end
#if($inCognitoGroup)
{
            …write/read from datasource
}
#else
    $utils.unauthorized()
#end
```

Real-time data

Real-time app before AppSync



Real-time app with AppSync

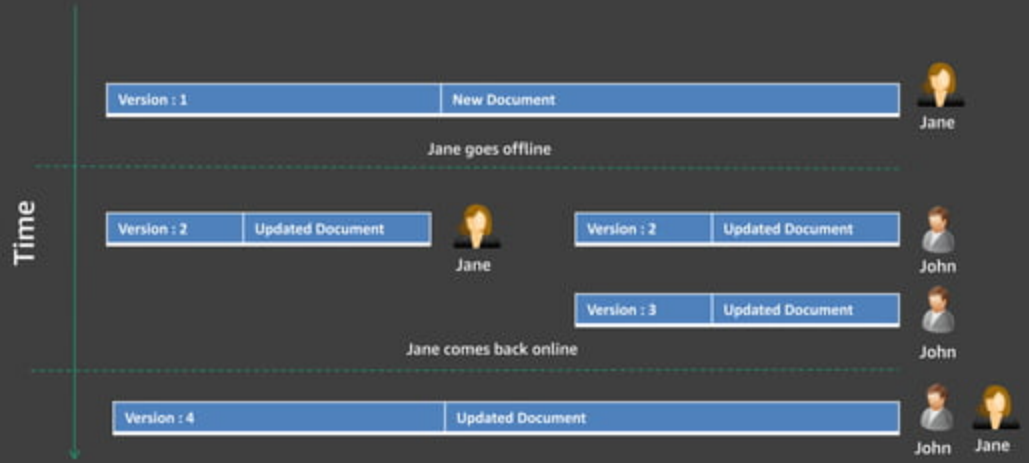


Real-time updates

- GraphQL subscriptions
- Event stream of mutation results
- Data synchronisation with MQTT + WebSockets
- Client managed WebSocket connection
- Automatic catch-up snapshots

Offline data and conflicts

Data conflicts



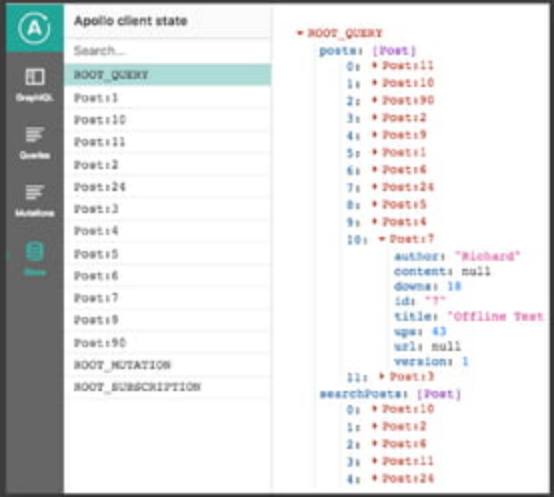
Handle data conflicts in the cloud

- Conflict detection with optimistic version check
- Conflict resolution strategies:
  - Client wins
  - Server wins
  - Silent reject
  - Custom logic in AWS Lambda
- Client callback for conflict resolution is available

```
"condition" : {
    "expression" : "someExpression"
    "conditionalCheckFailedHandler" : {
        "strategy" : "Custom",
        "lambdaArn" : "arn:..."
    }
}
```

Offline capabilities – AppSync offline storage

- Offline is a write-through "Store"
- Persistent storage mediums back the Apollo normalised cache
  - Local Storage for web
  - AsyncStorage for React Native
  - SQLite on native platforms
- Database can be preloaded
- Offline client can be configured
  - WiFi only
  - WiFi and cellular



AWS Amplify CLI

GraphQL transformer

```
$amplify push
```

```
type Post
@model
@auth(rules:
[{allow: owner}])
@searchable{
    id: ID!
    content: String
    description: String
    ups: Int
    downs: Int
}
```

Amazon Cognito User Pools

mutations

queries

AWS AppSync

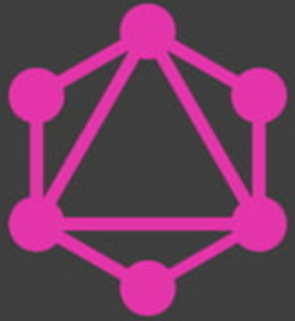
searchPosts

Amazon Elasticsearch

Amazon DynamoDB

createPost
readPost
updatePost
deletePost

Let's build a GraphQL API with AWS AppSync




F I GB USA MEX
ONE SIZE
DOES NOT
FIT ALL
30°C
Von links bügeln

GraphQL or REST?

Use GraphQL	Use REST
When data drives UI • Structured Data • Complex Data • Query-driven • Real-time/Offline	**When you leverage HTTP** • Caching • Content Types • Hypermedia (HATEOAS)
Client-driven development	For Resources (e.g. Kinesis)
Pros: Contract-driven, Introspection, Relations, Types **Conns**: Not as ubiquitous as REST	**Pros**: HTTP Client, Golden Standard, HTTP/2 Performance gains **Conns**: Over fetching/Under fetching

GraphQL or REST?

Bottom Line

It depends on the use case and, most importantly…

Good API Design!

Recap

- What is GraphQL
- Why was GraphQL created
- How to build a GraphQL API
- How to operate a GraphQL API in production
- We built a GraphQL API

Thank you 🙏😁👋

https://aws.amazon.com/appsync

follow me on twitter

@appwiz