

The background features a vibrant, multi-colored gradient. It starts with a dark blue on the left, transitions through purple and magenta, and then into bright orange and yellow towards the right. A diagonal line separates the darker blue on the left from the lighter colors on the right.

AWS  
re:Invent

**S V S 3 3 8 - R**

# API patterns and architectures: When and how to use RESTful and GraphQL APIs

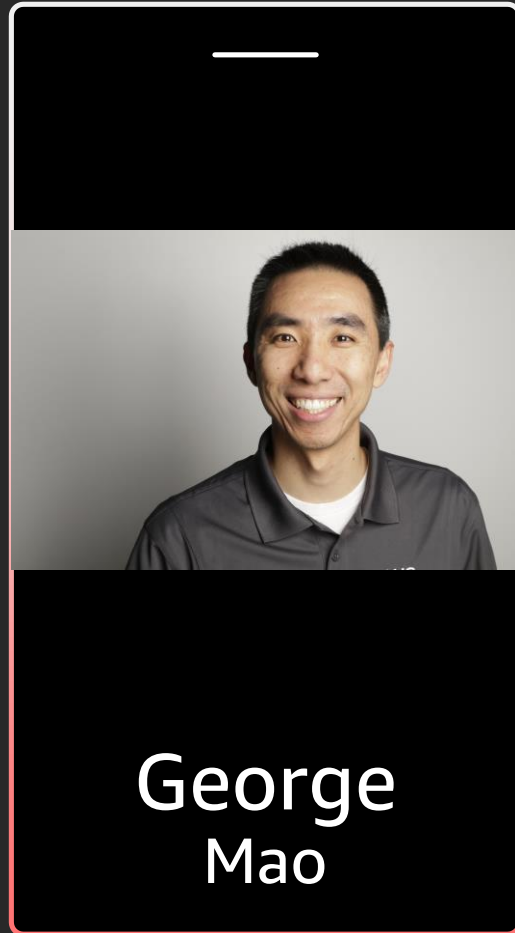
## **George Mao**

Principal Serverless Specialist  
Amazon Web Services

## **Matt Trescot**

Sr. Manager, Serverless  
Amazon Web Services

# About Us



George Mao  
Serverless  
Specialist

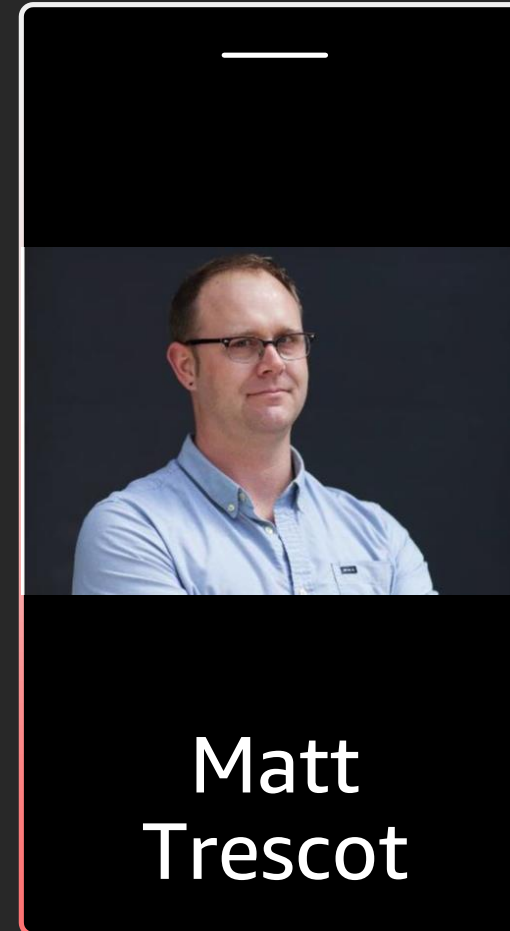
[georgmao@amazon.com](mailto:georgmao@amazon.com)

@georgemao (Twitter)

@georgemao (Slack  
#awsdevelopers)

George  
Mao

Principal  
Serverless  
Specialist



Matt Trescot  
Sr. Manager,  
Serverless

[mtrescot@amazon.com](mailto:mtrescot@amazon.com)

@m\_trescot (Twitter)

Matt  
Trescot

Sr. Manager,  
Serverless

# Related breakouts

SVS338-R1 (**Repeat of this session**), Wednesday, 8:30 a.m., MGM

SVS338-R2 (**Repeat of this session**), Thursday, 1 p.m., Bellagio

SVS305-R/-R1 – **How to secure your serverless APIs**

SVS327-R to -R3 – **Build Serverless APIs with the AWS CDK**

# Agenda

The Scenario

Discussion

Demo

Q&A

# The scenario

# Travel portal requirements

List all available travel destinations

Find a destination by?

Type. State. ZIP. Name.

What is the **weather** like at a destination?

What are some **hotels** at a destination?

What are some **things to do** at a destination?

What else?

Requirement 1

?

Requirement 2

?



# Let's implement it!

# What is REST?

REST is designed for servers

Combine **resource path** + **HTTP method**

**/destinationAPI**

**/getAllDestinations**

•*GET*

**/getAllDestinationsWithWeather**

•*GET*

**/getDestinationByState, / getDestinationByZip**

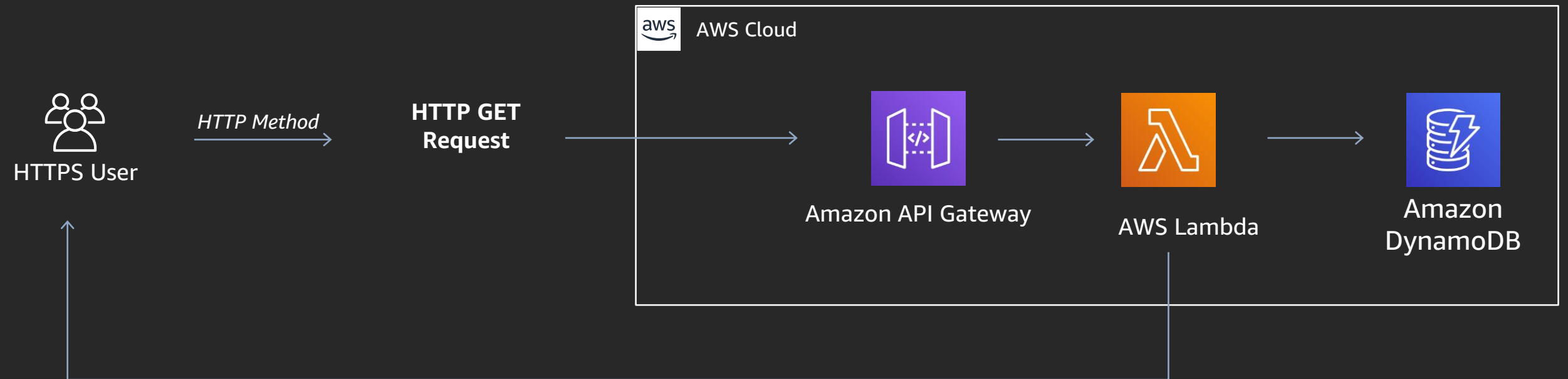
•*GET*

**/saveDestination, /deleteDestination**

•*PUT or POST*

# List all travel destinations via REST

**GET** <http://api.mycompany.com/getAllDestinations>



{

```
"name" : "Disney Land",  
"address" : "123 Disney Lane",  
"state" : "FL",  
"zip" : "92802",  
"type" : "Amusement Park"
```

}

# What if you wanted to include weather information?

Add a new resource:

GET <http://api.mycompany.com/getAllDestinationsWithWeather>

Support query params:

GET <http://api.Mycompany.Com/getalldestinations?Options=weather>

# List all travel destinations via REST

**GET** <http://api.mycompany.com/getAllDestinationsWithWeather>

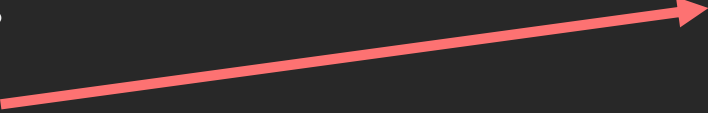
```
{
  "name" : "Disney Land",
  "address" : "123 Disney Lane",
  "state" : "FL",
  "zip" : "44001",
  "type" : "Amusement Park"
}
→
{
  "name" : "Disney Land",
  "address" : "123 Disney Lane",
  "state" : "FL",
  "zip" : "44001",
  "type" : "Amusement Park",
  "weather" : {
    "high" : "70",
    "low" : "50"
  }
}
```

# Let's evaluate GraphQL instead ... What is GraphQL?

GraphQL is strongly typed and designed to allow clients to ask for data  
Combine a **static resource path** + **HTTP POST**

Define a schema:

- types
- queries
- mutations



```
type Destination {  
  id: ID!  
  name: String!  
  address: String!  
  state: String!  
  zip: String!  
  weather: Weather!  
  type: Status  
}
```

# The GraphQL schema

```
type Destination {  
  id: ID!  
  name: String!  
  address: String!  
  state: String!  
  zip: String!  
  weather: Weather!  
  type: String!  
}
```

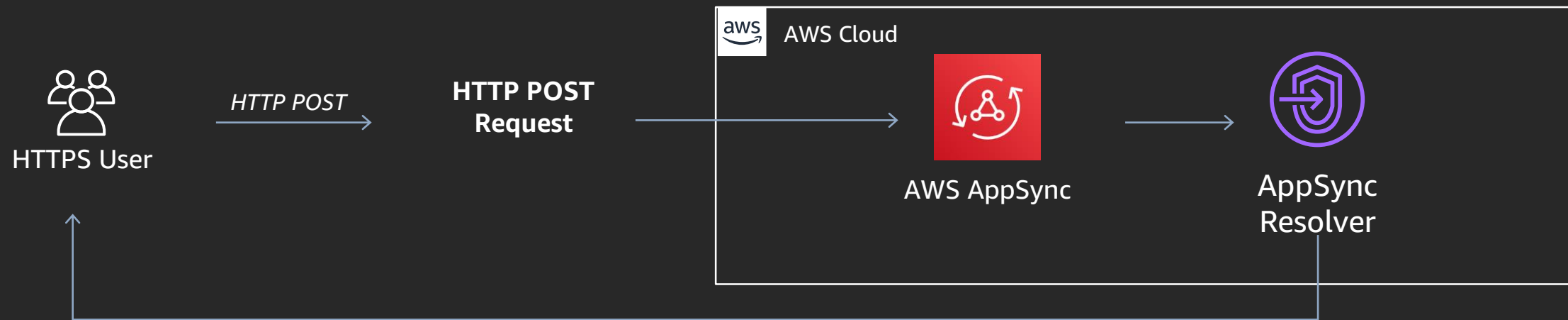
```
type Weather {  
  temp: String!  
  description: String!  
}
```

```
type Query {  
  getAllDestinations: [Destination]  
  getDestination(id: ID!, zip: String): Destination  
  getDestinationsByState(state: String!): [Destination]  
}
```

```
type Mutation {  
  saveDestination: [Destination]  
  deleteDestination: [Destination]  
}
```

# List all travel destinations via GraphQL

**POST** <http://api.mycompany.com/graphql>

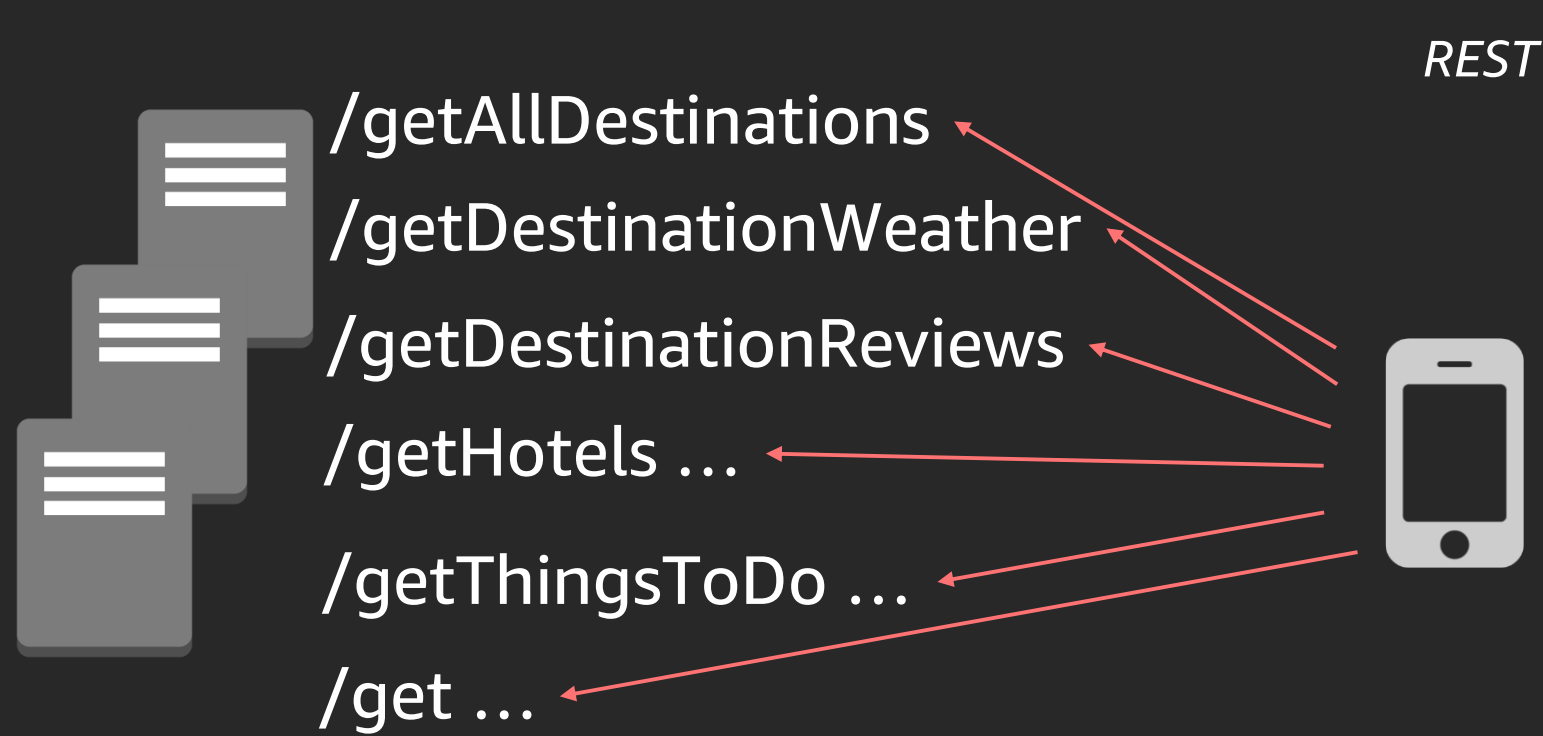


```
query getAllDestinations{
  getAllDestinations(){
    name
    address
    state
    zip
    type
  }
}
```

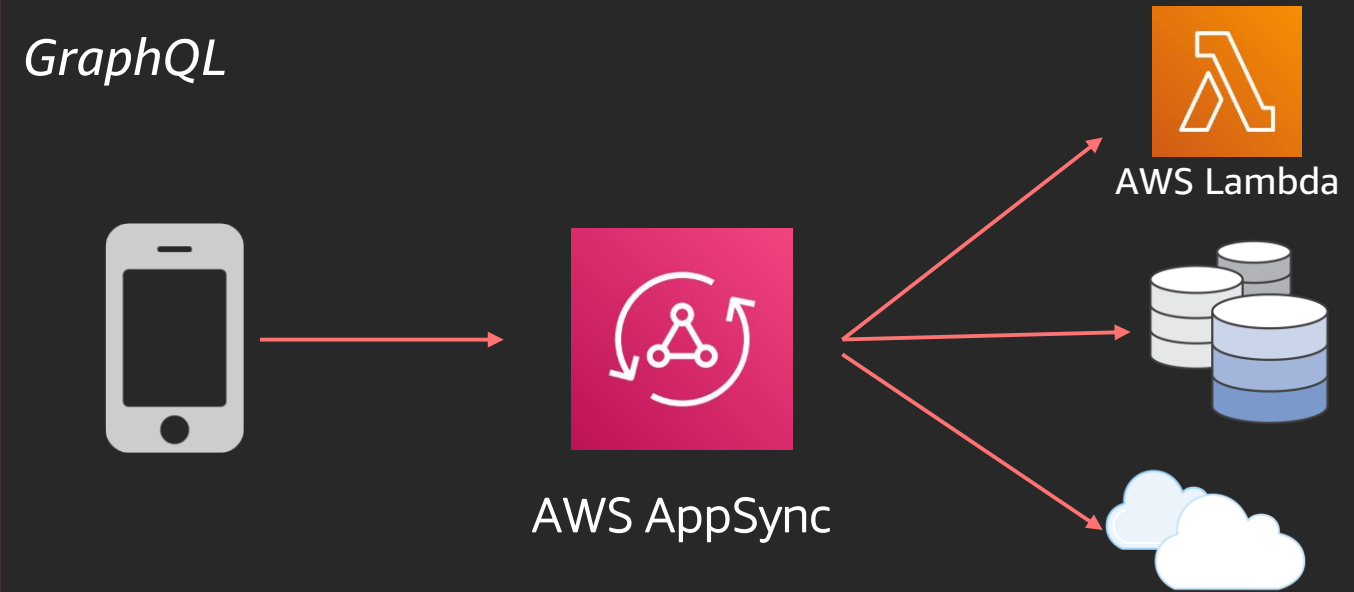
```
query getAllDestinations{
  getAllDestinations(){
    name
    state
    type
  }
}
```



# REST API vs. GraphQL



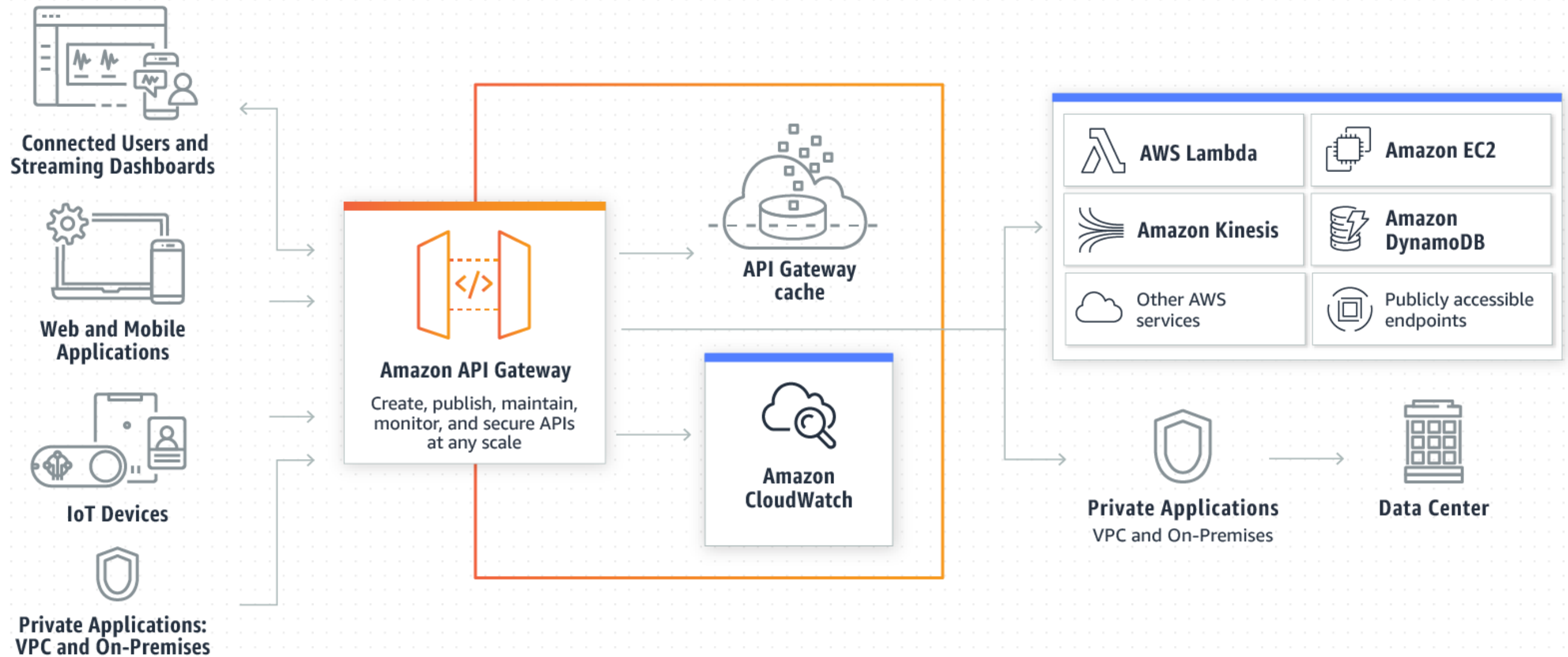
With a REST-based interface, I need to make several calls sequentially based on responses from the previous call



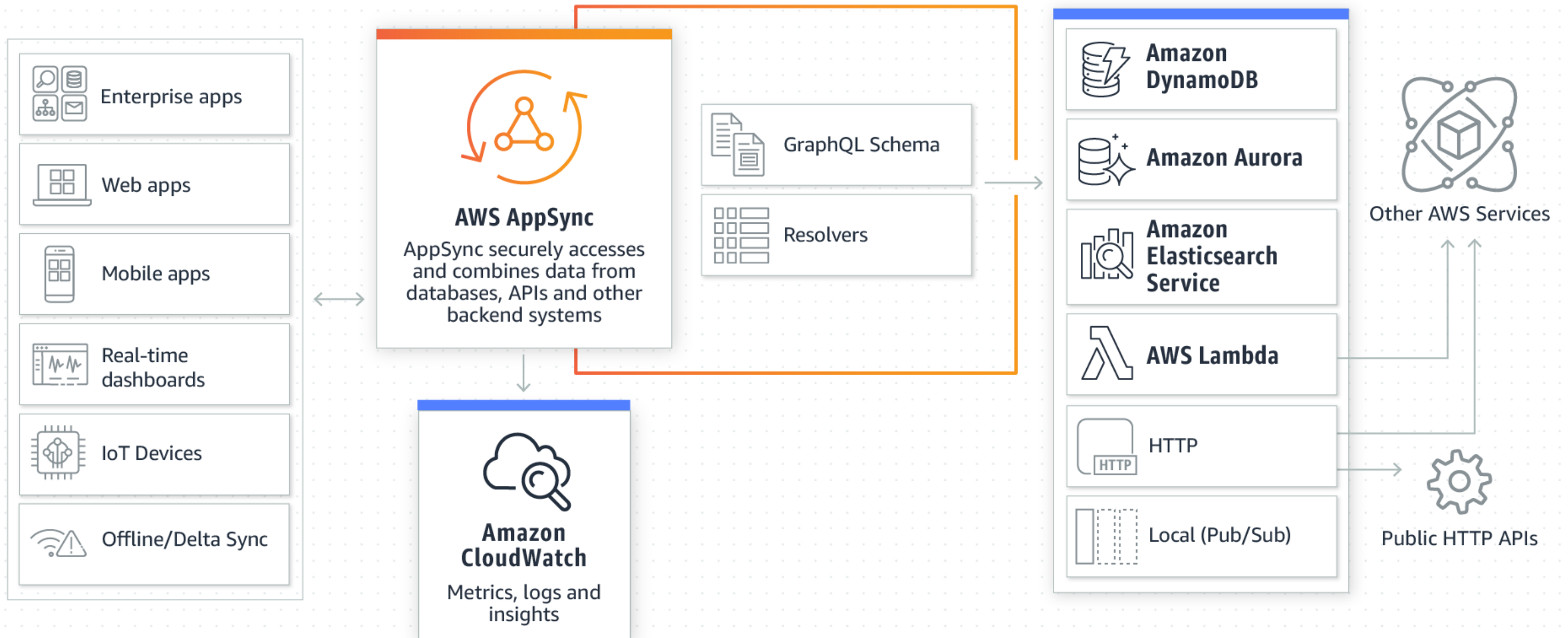
```
query {  
  getAllDestinations {  
    ...  
  }  
}
```

With GraphQL I can make one call

# Amazon API Gateway

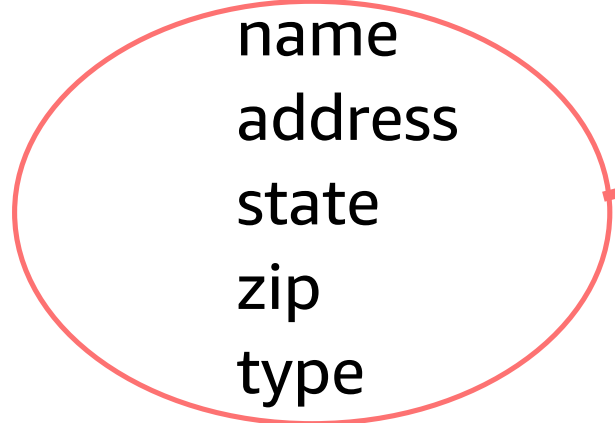


# AWS AppSync

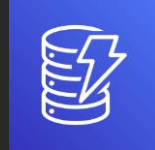


# AWS AppSync query resolution

```
query getAllDestinations{  
  getAllDestinations(){  
    name  
    address  
    state  
    zip  
    type  
    weather {  
      temp  
      description  
    }  
  }  
}
```



Resolver



DynamoDB

Resolver



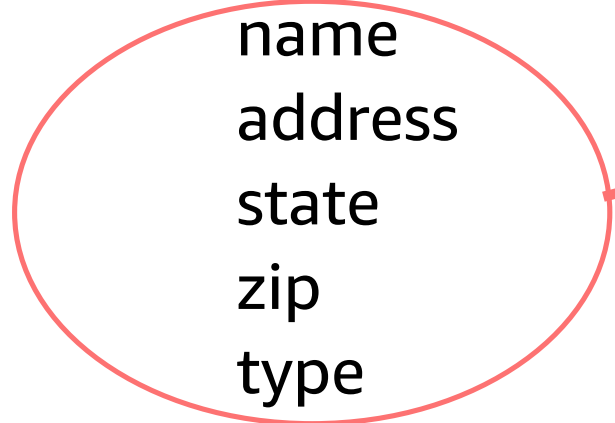
Lambda

Scan

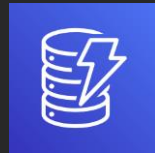
```
{  
  "TableName" : "Destinations"  
}
```

# AWS AppSync query resolution

```
query getAllDestinations{  
  getAllDestinations(){  
    name  
    address  
    state  
    zip  
    type  
  }  
}
```



Resolver



DynamoDB

Scan

```
{  
  "TableName" : "Destinations"  
}
```

# Modifying data

**POST** /saveDestination?id=some-ID-here HTTP/1.1

Host: api.mycompany.com

Authorization: [...]

```
{  
  "name" : "YellowStone",  
  "address": "Yellowstone national park",  
  "state": "WY",  
  "zip" : "82190",  
  "type" : "National Park"  
}
```

**POST** /graphql HTTP/1.1

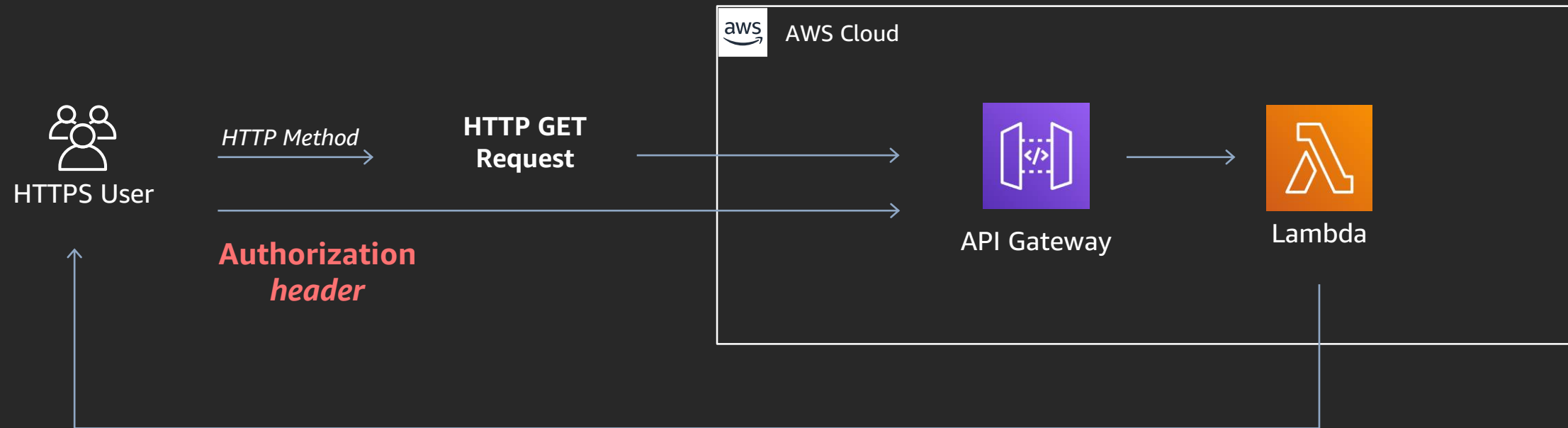
Host: api.mycompany.com

Authorization: [...]

```
mutation saveDestination{  
  saveDestination(  
    name : "YellowStone",  
    address: "Yellowstone national park",  
    state: "WY",  
    zips : "82190",  
    type : "National Park"  
  ){  
    id  
  }  
}
```

# Security

# API Gateway REST security



Authorization: Basic Base64(username:password)

Authorization: Bearer [JWT Token]

Authorization: AWS4-HMAC-SHA256 [....]



# AppSync GraphQL security with multi auth

```
type Destination {  
  id: ID!  
  name: String!  
  address: String!  
  state: String!  
  zip: String!  
  weather: Weather!  
  @aws_api_key  
  type: String!  
}
```

```
type Query {  
  getAllDestinations: [Destination]  
  getDestination(id: ID!, zip: String): Destination  
  getDestinationsByState(state: String!): [Destination]  
}
```

```
type Mutation {  
  saveDestination: [Destination]  
  @aws_auth(cognito_groups: ["Members"])  
  @aws_oidc  
  
  deleteDestination: [Destination]  
  @aws_auth(cognito_groups: ["Admins"])  
}
```

# Demo

# REST: Load a page with 1,000 destinations

REST with API Gateway	Requests
/getAllDestinations	1
/getDestinationWeather	1,000
/getHotels	1,000
/getThingsToDo	1,000
Total HTTP requests	3,001

REST with API Gateway	Requests
/getAllDestinations	1
/getDestinationDetails	1,000
Total HTTP requests	1,001

# GraphQL: Load a page with 1,000 destinations

GraphQL with AWS AppSync	Requests
getDestinations()	1
Total HTTP requests	1

# Summary

## REST with API Gateway

No strongly typed schema

All HTTP methods to a /resource

Single auth per resource

Server-controlled response

Tools and support in HTML and JavaScript

## GraphQL with AWS AppSync

Strongly typed schema

HTTP POSTS to /graphql

Multi auth supported

Client specifies response needed

Requires additional SDKs or APIs

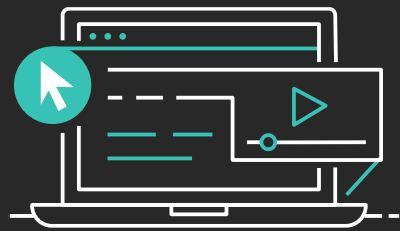
SO WHAT DOES THE FUTURE LOOK LIKE?

**ALL THE CODE YOU EVER WRITE IS BUSINESS LOGIC**



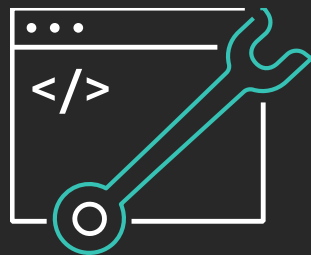
# Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development



Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures



Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at <https://aws.training>

# Thank you!

## **George Mao**

georgmao@amazon.com  
@georgemao (Twitter)  
@georgemao (Slack awsdevelopers)

## **Matt Trescot**

mtrescot@amazon.com  
@m\_trescot (Twitter)





Please complete the session survey in the mobile app.