AWS Architecture Blog

How to Architect APIs for Scale and Security

by George Mao | on 30 JUL 2019 | in Advanced (300), Amazon API Gateway, Architecture, AWS AppSync, AWS Lambda, Intermediate (200) | Permalink | Share

We hope you've enjoyed reading our posts on best practices for your serverless applications. This series of posts will focus on best practices and concepts you should be familiar with when you architect APIs for your applications. We'll kick this first post off with a comparison between REST and GraphQL API architectures.

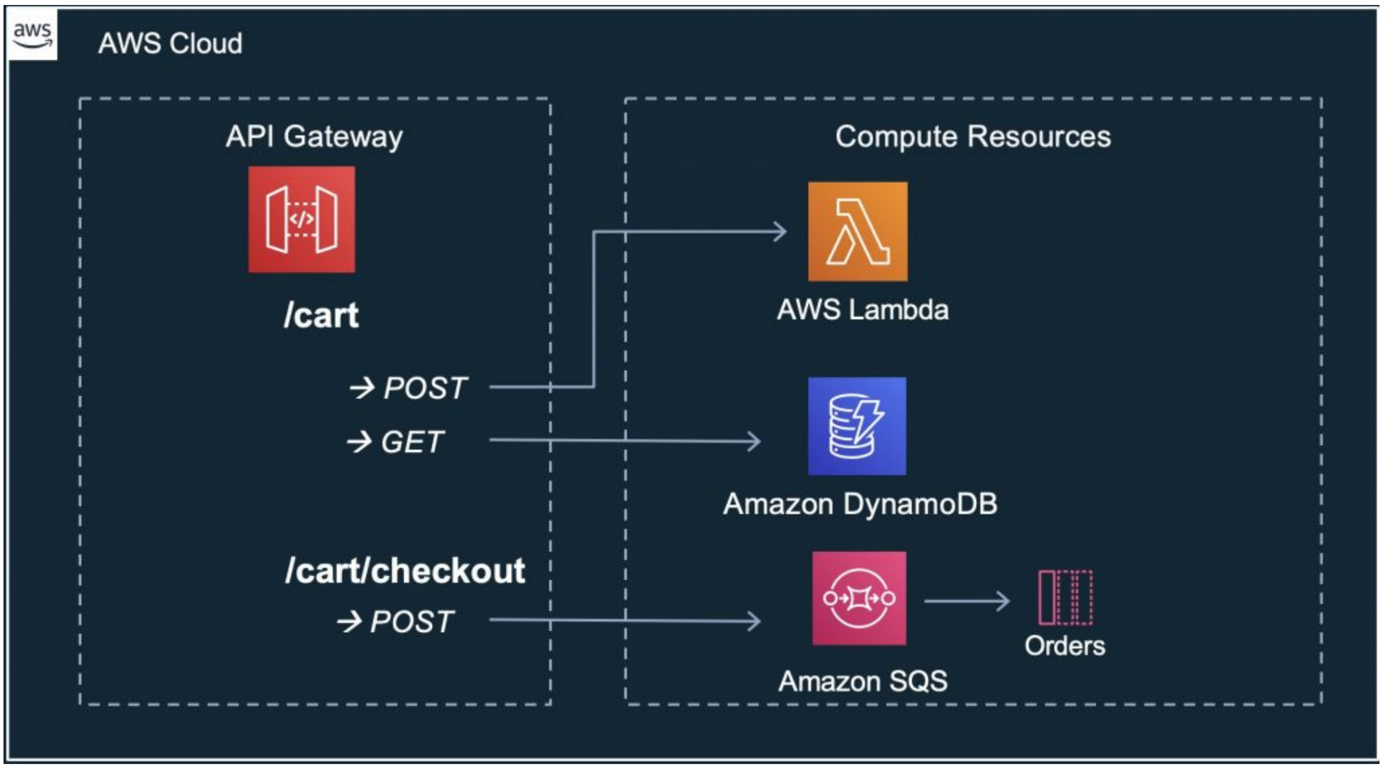
Introduction

Developers have been creating RESTful APIs for a long time, typically using HTTP methods, such as GET, POST, DELETE to perform operations against the API. Amazon API Gateway is designed to make it easy for developers to create APIs at any scale without managing any servers. API Gateway will handle all of the heavy lifting needed including traffic management, security, monitoring, and version/environment management.

GraphQL APIs are relatively new, with a primary design goal of allowing clients to define the structure of the data that they require. AWS AppSync allows you to create flexible APIs that access and combine multiple data sources.

REST APIs

Architecting a REST API is structured around creating combinations of resources and methods. Resources are paths that are present in the request URL and methods are HTTP actions that you take against the resource. For example, you may define a resource called "cart": *http://myapi.somecompany.com/cart*. The cart resource can respond to HTTP POSTs for adding items to a shopping cart or HTTP GETs for retrieving the items in your cart. With API Gateway, you would implement the API like this:



Behind the scenes, you can integrate with nearly any backend to provide the compute logic, data persistence, or business work flows. For example, you can configure an AWS Lambda function to perform the addition of an item to a shopping cart (HTTP POST). You can also use API Gateway to directly interact with AWS services like Amazon DynamoDB. An example is using API Gateway to retrieve items in a cart from DynamoDB (HTTP GET).

RESTful APIs tend to use Path and Query parameters to inject dynamic values into APIs. For example, if you want to retreive a specific cart with an id of abcd123, you could design the API to accept a query or path parameter that specifies the cartID:

```
/cart?cartId=abcd123 or /cart/abcd123
```

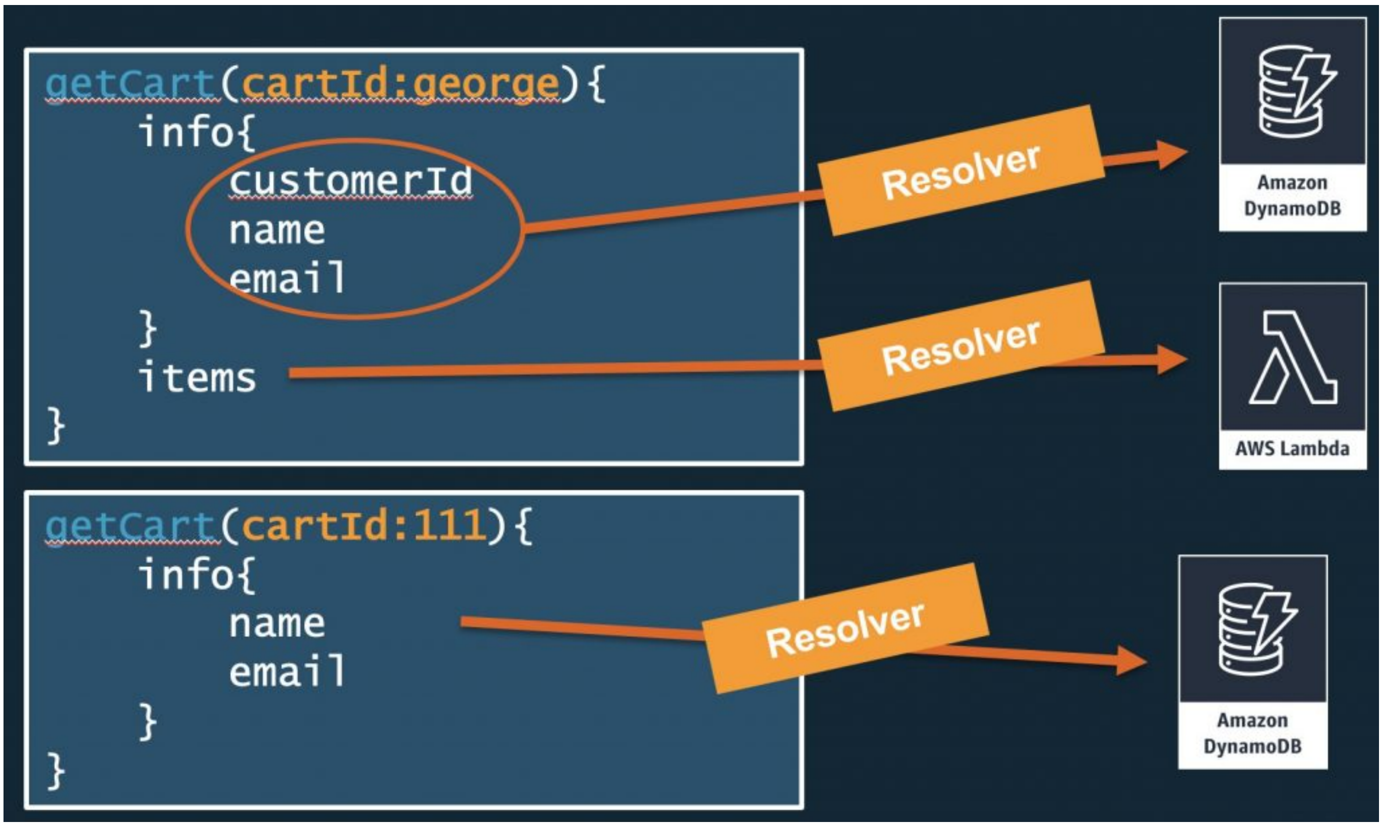
Finally, when you need to add functionality to your API, the typical approach would be to add additional resources. For example, to add a checkout function, you could add a resource called /cart/checkout.

GraphQL APIs

Architecting GraphQL APIs is not structured around resources and HTTP verbs, instead you define your data types and configure where the operations will retrieve data through a resolver. An operation is either a query or a mutation. Queries simply retrieve data while mutations are used when you want to modify data. If we use the same example from above, you could define a cart data type as follows:

```
type Cart {

  cartId: ID!

  customerId: String

  name: String

  email: String
  items: [String]
}
```

Next, you configure the fields in the Cart to map to specific data sources. AppSync is then responsible for executing resolvers to obtain appropriate information. Your client will send a HTTP POST to the AppSync endpoint with the exact shape of the data they require. AppSync is responsible for executing all configured resolvers to obtain the requested data and return a single response to the client.


```
getCart(cartId:george){
    info{
        customerId
        name
        email
    }
    items
}
```
```
getCart(cartId:111){
    info{
        name
        email
    }
}
```

With GraphQL, the client can change their query to specify the exact data that is needed. The above example shows two queries that ask for different sets of information. The first getCart query asks for all of the static customer (customerId, name, email) and a list of items in the cart. The second query just asks for the customer's static information. Based on the incoming query, AppSync will execute the correct resolver(s) to obtain the data. The client submits the payload via a HTTP POST to the same endpoint in both cases. The payload of the POST body is the only thing that changes.

As we saw above, a REST based implementation would require the API to define multiple HTTP resources and methods or path/query parameters to accomplish this.

AppSync also provides other powerful features that are not possible with REST APIs such as real-time data synchronization and multiple methods of authentication at the field and operation level.

Summary

As you can see, these are two different approaches to architecting your API. In our next few posts, we'll cover specific features and architecture details you should be aware of when choosing between API Gateway (REST) and AppSync (GraphQL) APIs. In the meantime, you can read more about working with API Gateway and Appsync.

TAGS: amazon api gateway, app sync, architecture best practices, aws lambda, GraphQL APIs, REST APIs

Resources

AWS Well-Architected
AWS Architecture Monthly
AWS Whitepapers
AWS Training and Certification
This Is My Architecture
AWS Answers

Follow

Twitter
Facebook
LinkedIn
Twitch
Email Updates

Related Posts

Orchestrating an application process with AWS Batch using AWS CDK

Scale your Remote Access VPN on AWS

Managing resources using AWS CloudFormation Resource Types

In-Depth Strategies from Infosys to Help Customers Re-Platform Mainframes on AWS

How to trigger AWS CodeBuild jobs for selective file changes in AWS CodeCommit

Running SQL on Amazon Athena to Analyze Big Data Quickly and Across Regions

How to analyze well drilling reports using natural language processing

Importing Azure Active Directory users and groups into Alexa for Business Directory using AWS Lambda