AWS re:Invent

S V S 4 0 2 - R

Building APIs from front to back

Eric Johnson
Senior Developer Advocate – Serverless
Amazon Web Services

Who am I?

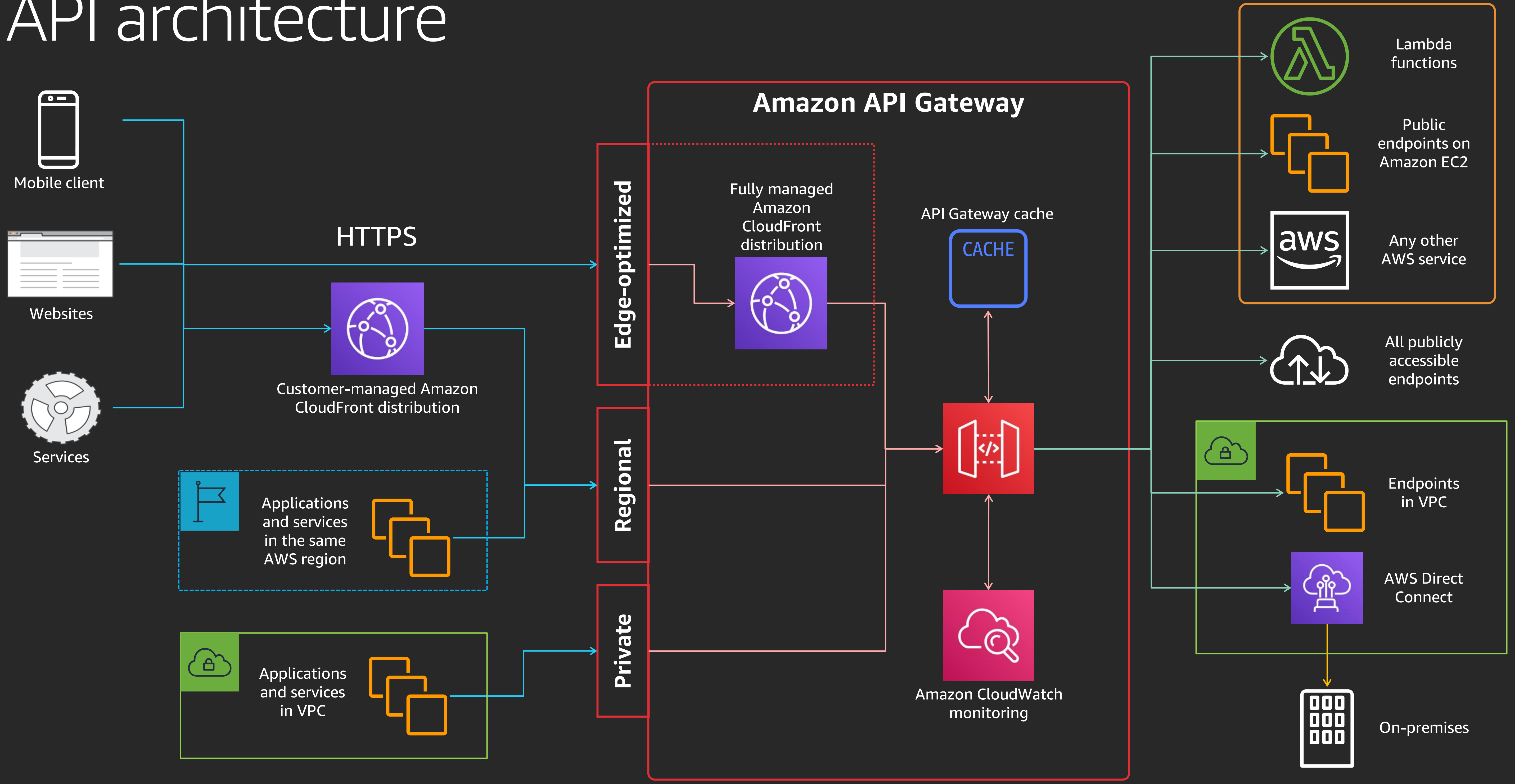
- Eric Johnson – @edjgeek
- Sr. Developer Advocate – Serverless, AWS
- Serverless/tooling/automation geek
- Software Architect/Solutions Architect
- Music lover
- Pizza and Diet Dr. Pepper fanatic

APIs on AWS

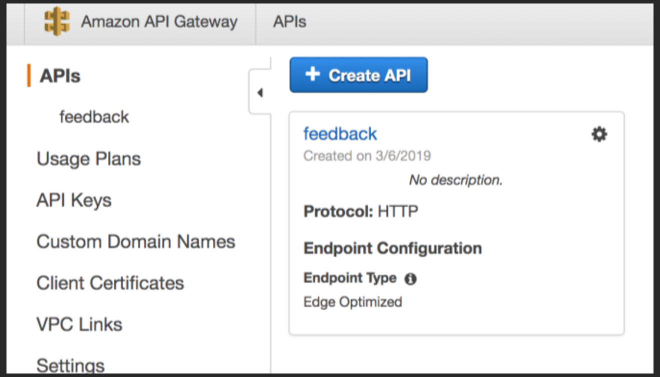
Amazon API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale

API architecture


Mobile client
Websites
Services
HTTPS
Customer-managed Amazon CloudFront distribution
Applications and services in the same AWS region
Applications and services in VPC
Edge-optimized
Regional
Private
Amazon API Gateway
Fully managed Amazon CloudFront distribution
API Gateway cache
CACHE
Amazon CloudWatch monitoring
Lambda functions
Public endpoints on Amazon EC2
aws
Any other AWS service
All publicly accessible endpoints
Endpoints in VPC
AWS Direct Connect
On-premises

API Gateway management



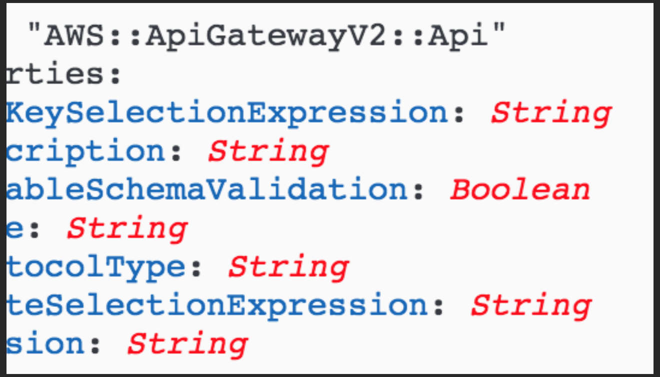
feedback
Created on 3/6/2019
No description.
Protocol: HTTP
Endpoint Configuration
Endpoint Type
Edge Optimized

AWS Management Console

```
APIGATEWAYV2()                                                    APIGATEWAYV2()

NAME
       apigatewayv2 -

DESCRIPTION
       Amazon API Gateway V2

AVAILABLE COMMANDS
       o create-api

       o create-api-mapping

       o create-authorizer

       o create-deployment
```

AWS CLI

AWS SAM



```
 "AWS::ApiGatewayV2::Api"
rties:
KeySelectionExpression: String
cription: String
ableSchemaValidation: Boolean
e: String
tocolType: String
teSelectionExpression: String
sion: String
```

AWS CloudFormation

Swagger/OpenAPI

AWS Cloud Development Kit

AWS SAM templates

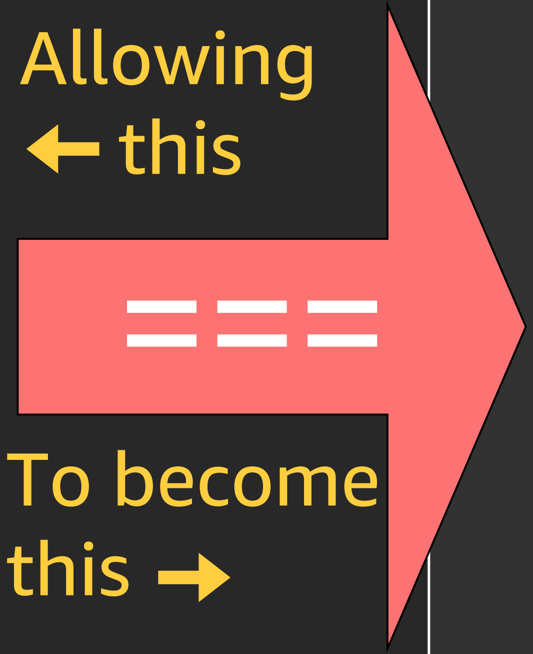
```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetProductsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.getProducts
      Runtime: nodejs10.x
      CodeUri: src/
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref ProductTable
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /products/{productId}
            Method: get
  ProductTable:
    Type: AWS::Serverless::SimpleTable
```

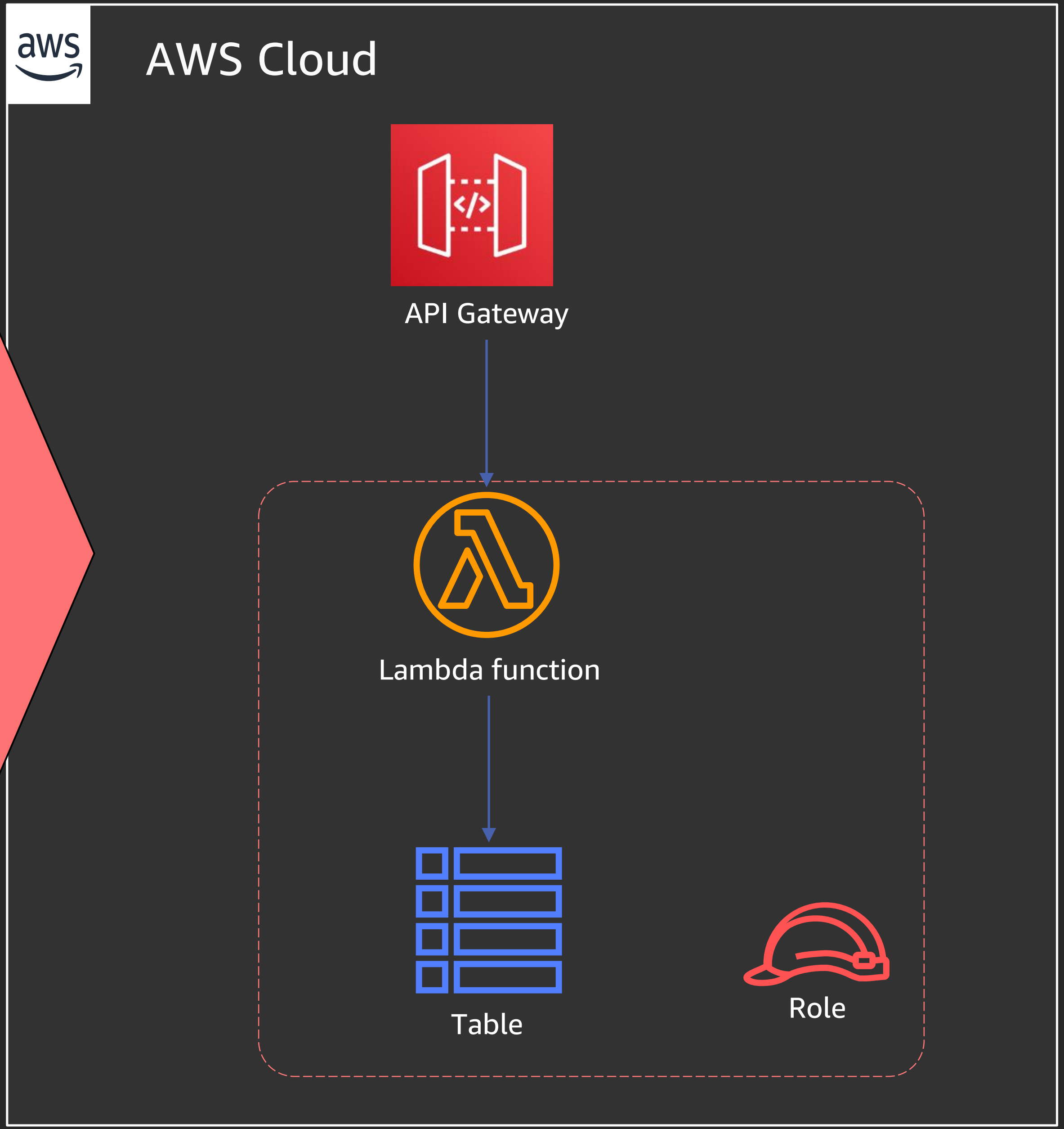
Just 20 lines to create:

- Lambda function
- IAM role
- API Gateway
- DynamoDB table

AWS SAM templates

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  GetProductsFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.getProducts
      Runtime: nodejs10.x
      CodeUri: src/
      Policies:
        - DynamoDBReadPolicy:
            TableName: !Ref ProductTable
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /products/{productId}
            Method: get
  ProductTable:
    Type: AWS::Serverless::SimpleTable
```





What are we going to build?

Meet Angus and Elly

- Newly married
- Want to keep track of each other
- Budding developers
- Want to build it themselves
- Want it to be secure
- Want to use serverless

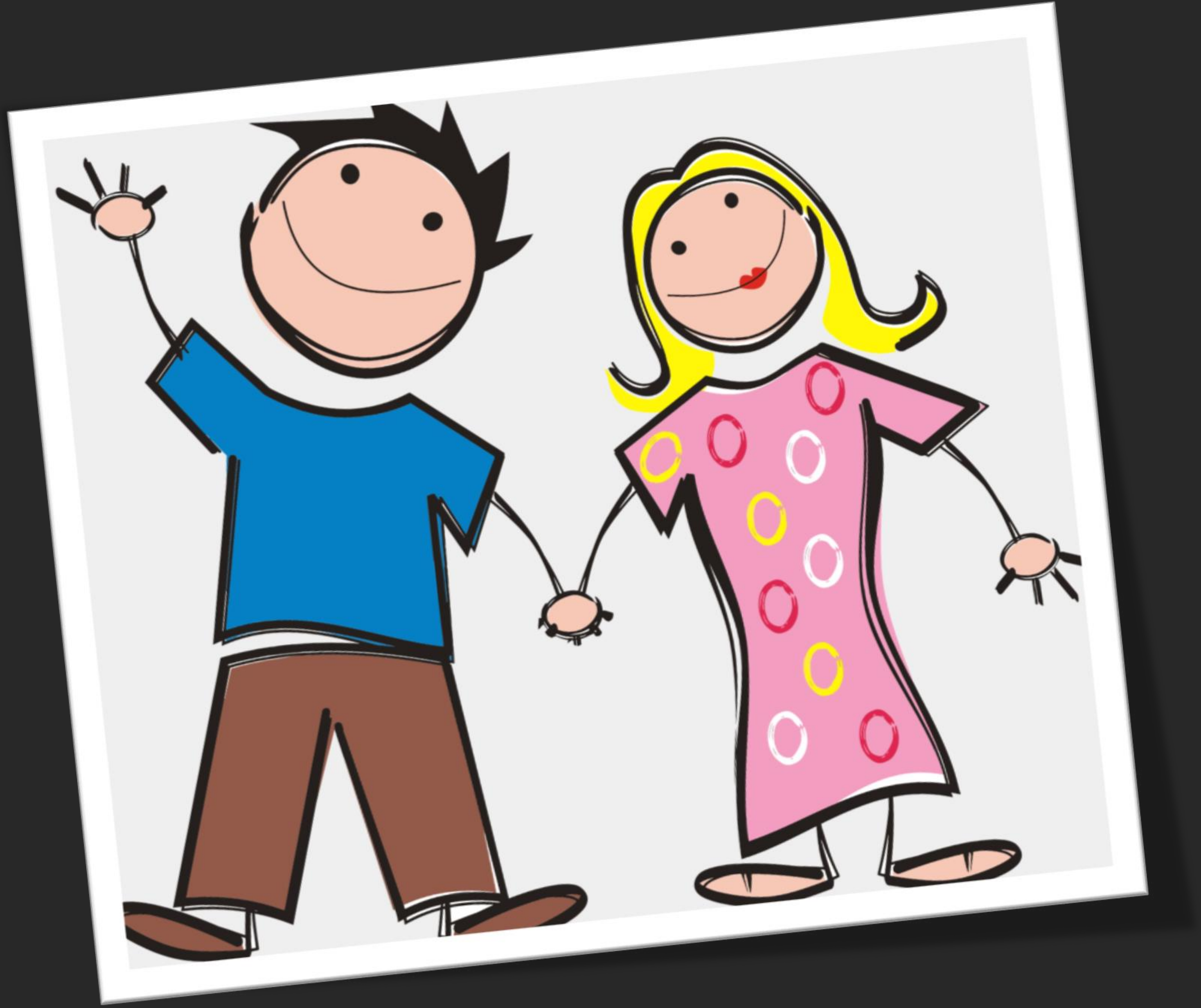
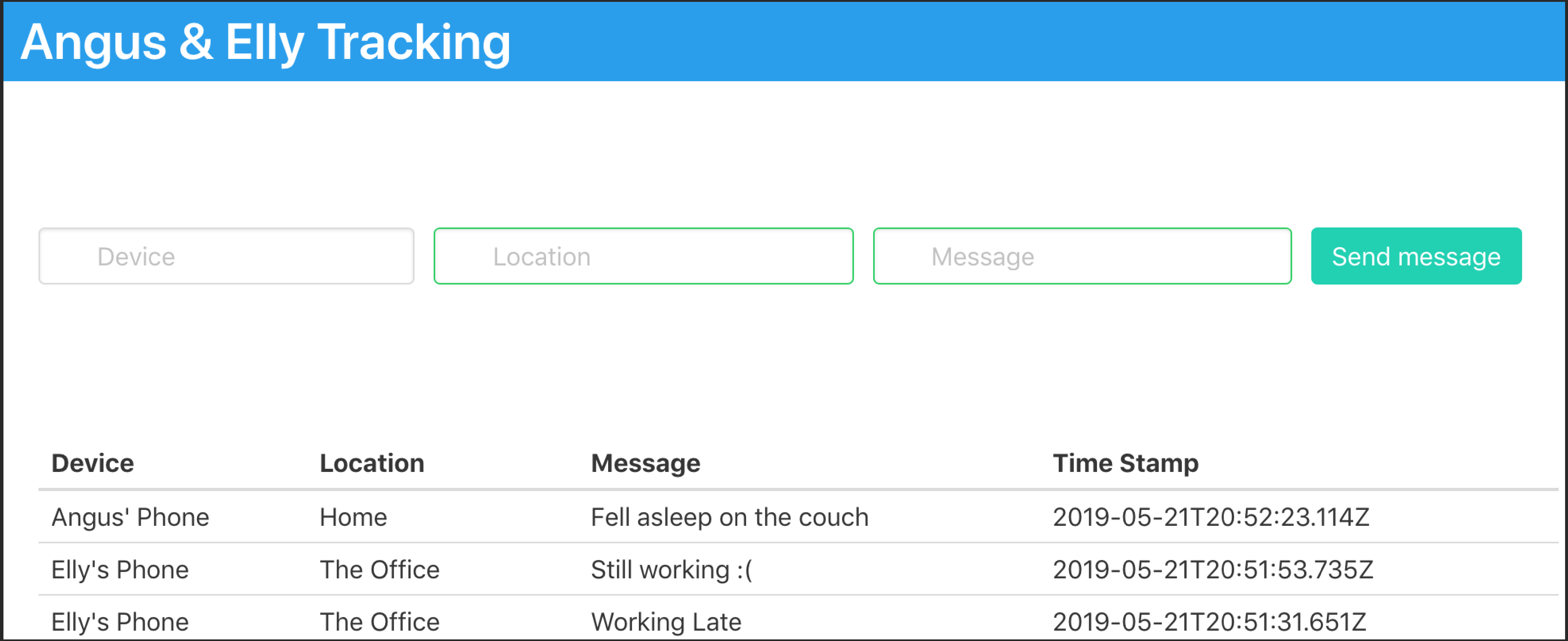


Image source: https://pixabay.com/vectors/boy-colorful-comic-characters-1298928/

The family website

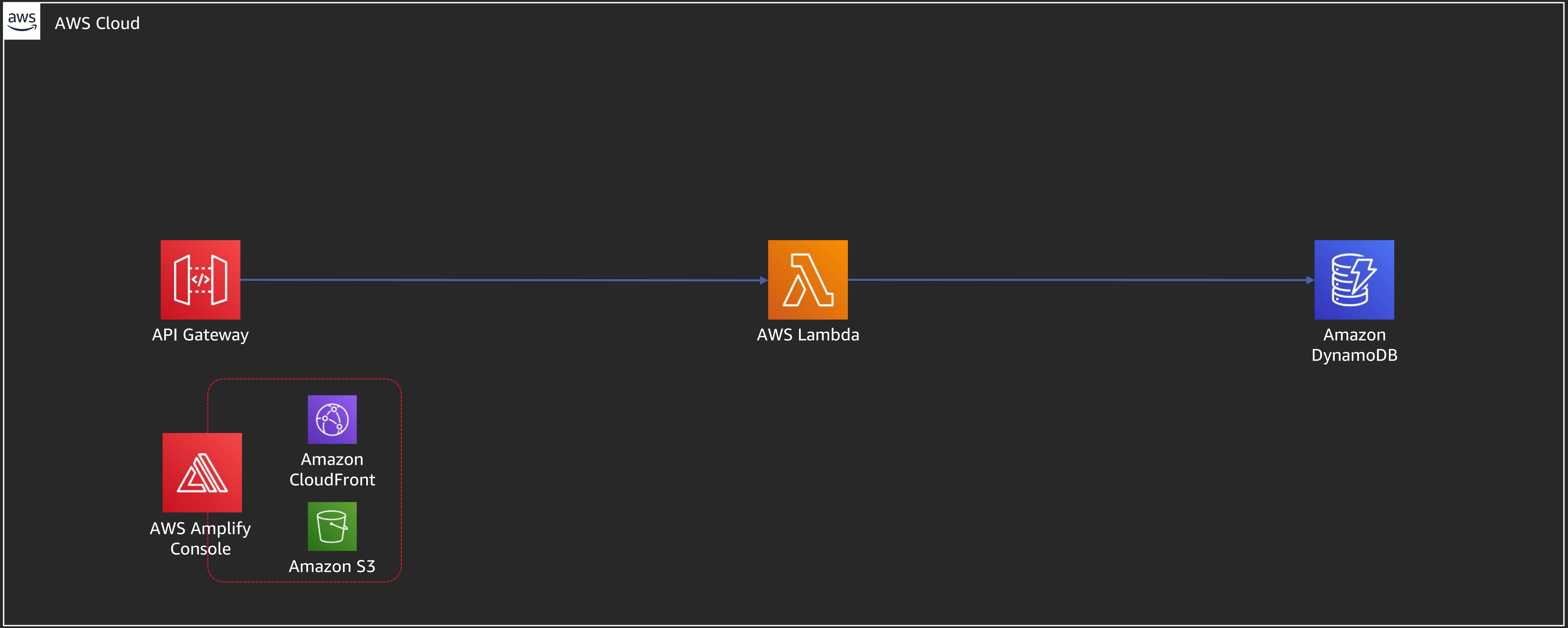


Angus & Elly Tracking

Device	Location	Message	Time Stamp
Angus' Phone	Home	Fell asleep on the couch	2019-05-21T20:52:23.114Z
Elly's Phone	The Office	Still working :(	2019-05-21T20:51:53.735Z
Elly's Phone	The Office	Working Late	2019-05-21T20:51:31.651Z

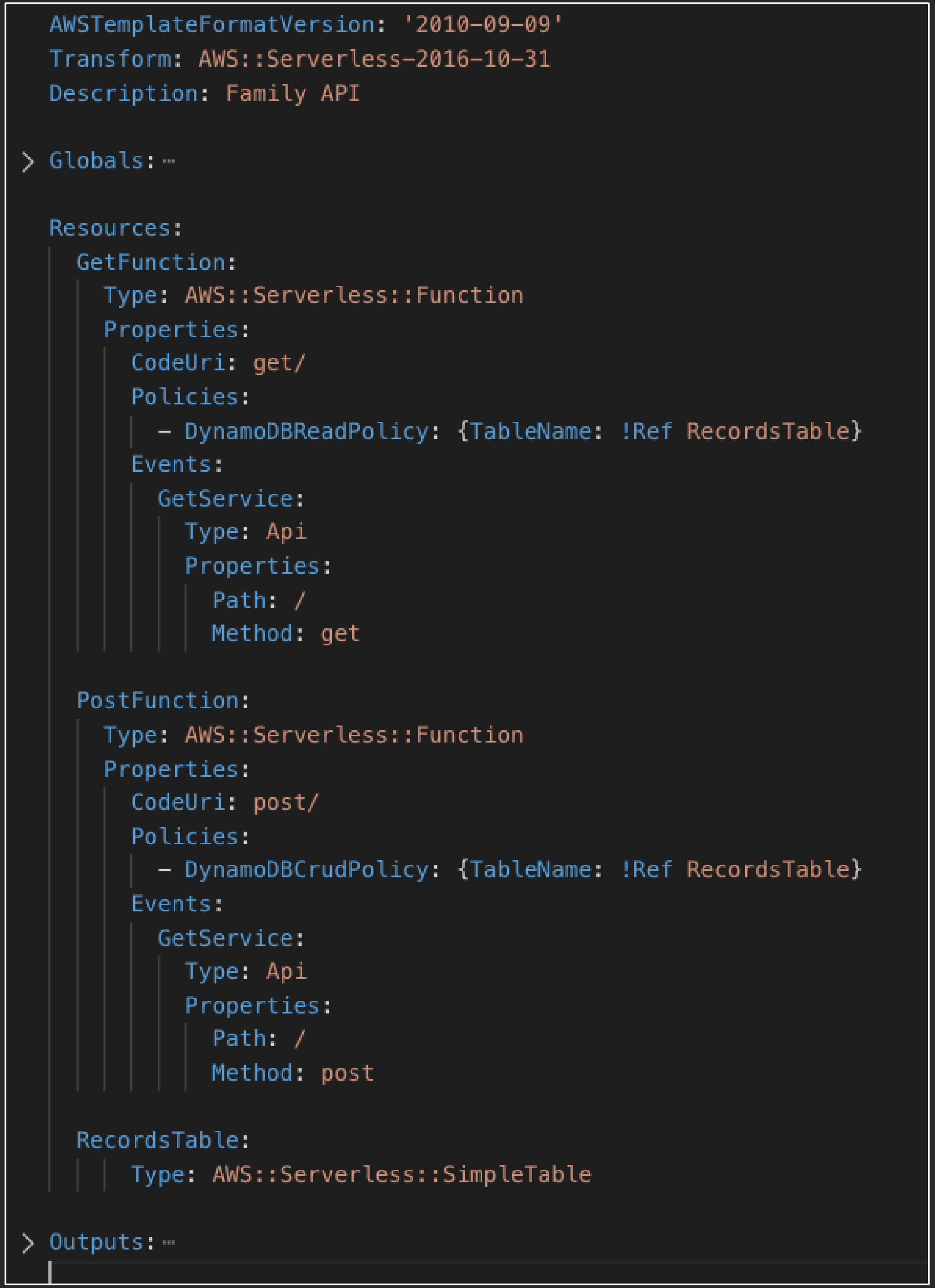
Phase one: A basic family website

What services shall we start with?



Show me code!

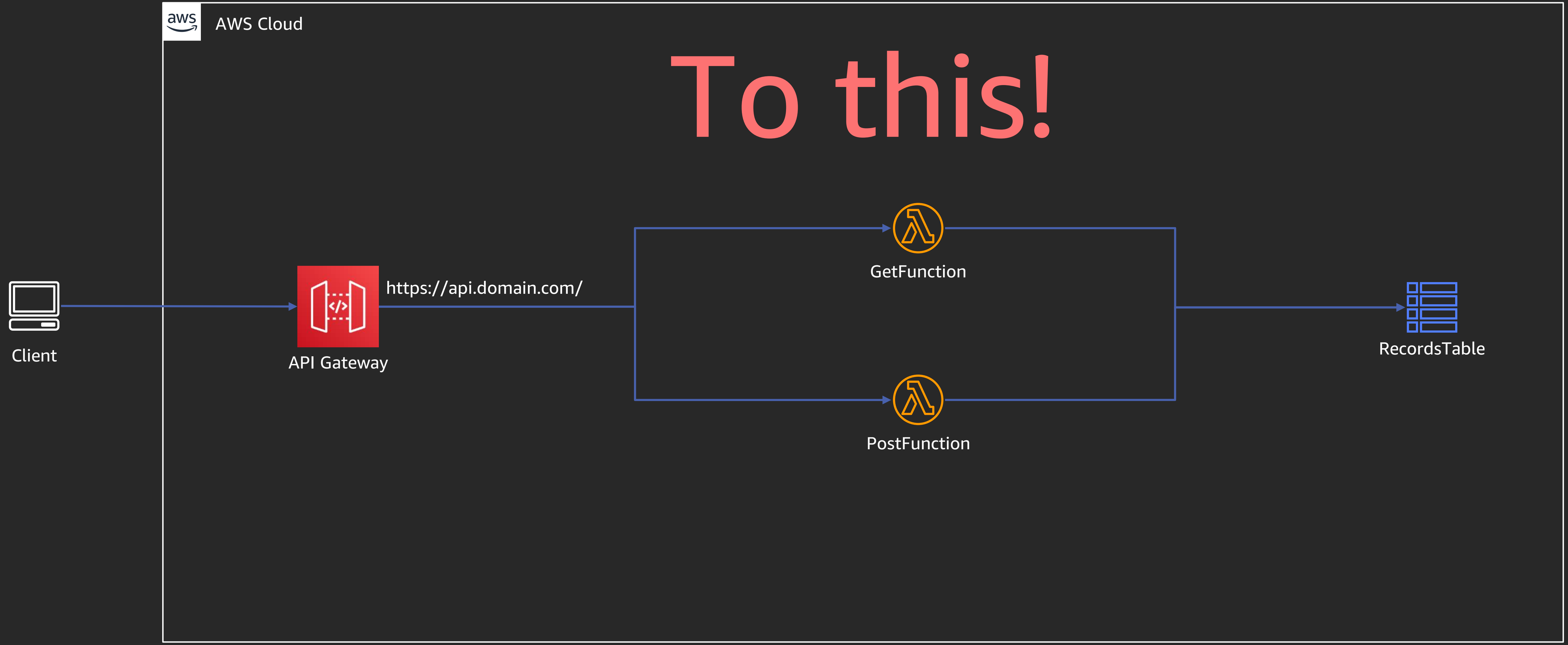
Phase one summary



```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Family API

> Globals:⋯

Resources:
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: get/
      Policies:
        - DynamoDBReadPolicy: {TableName: !Ref RecordsTable}
      Events:
        GetService:
          Type: Api
          Properties:
            Path: /
            Method: get

  PostFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: post/
      Policies:
        - DynamoDBCrudPolicy: {TableName: !Ref RecordsTable}
      Events:
        GetService:
          Type: Api
          Properties:
            Path: /
            Method: post

  RecordsTable:
      Type: AWS::Serverless::SimpleTable

> Outputs:⋯
```

We went from this …

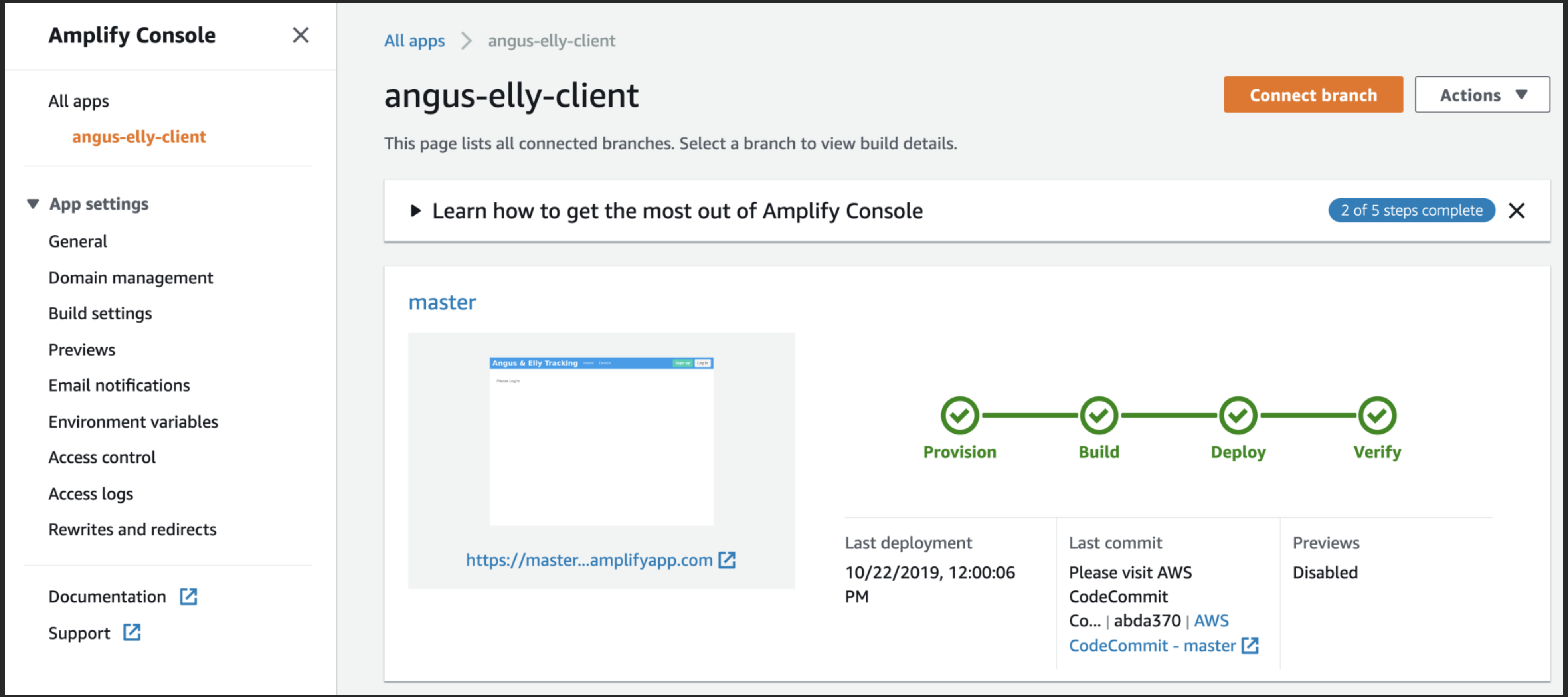
Phase one summary



Hosting the front end

AWS Amplify Console

Git-based workflow for deploying and hosting full-stack serverless web applications


Amplify Console
All apps
angus-elly-client
App settings
General
Domain management
Build settings
Previews
Email notifications
Environment variables
Access control
Access logs
Rewrites and redirects
Documentation
Support

All apps > angus-elly-client

angus-elly-client

Connect branch | Actions

This page lists all connected branches. Select a branch to view build details.

Learn how to get the most out of Amplify Console — 2 of 5 steps complete

master

https://master...amplifyapp.com

Provision — Build — Deploy — Verify

Last deployment	Last commit	Previews
10/22/2019, 12:00:06 PM	Please visit AWS CodeCommit Co... | abda370 | AWS CodeCommit - master	Disabled

AWS Amplify Console makes life easy!

Phase two: Securing and optimizing the family website

Not an exhaustive list

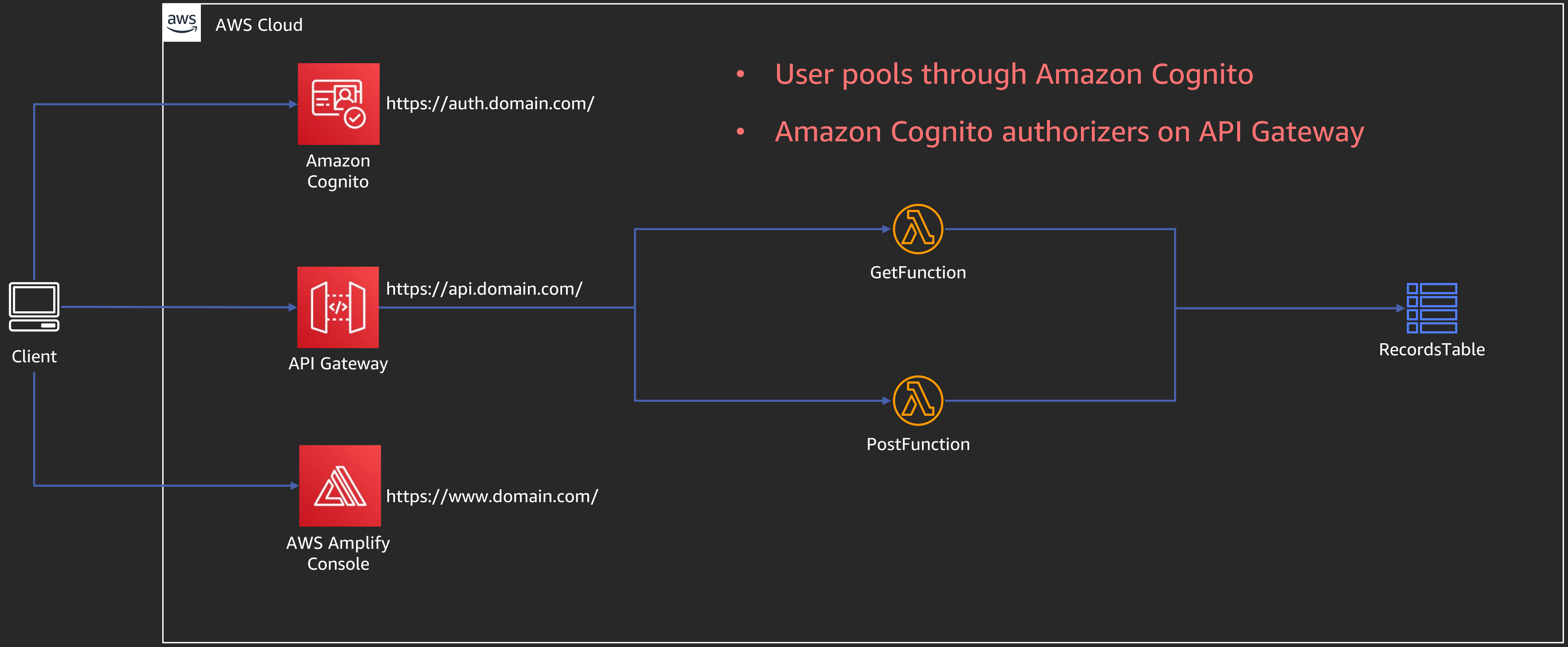
Covering

- Amazon Cognito
- Throttling
- Resource policies
- AWS WAF
- Data models

Not covering

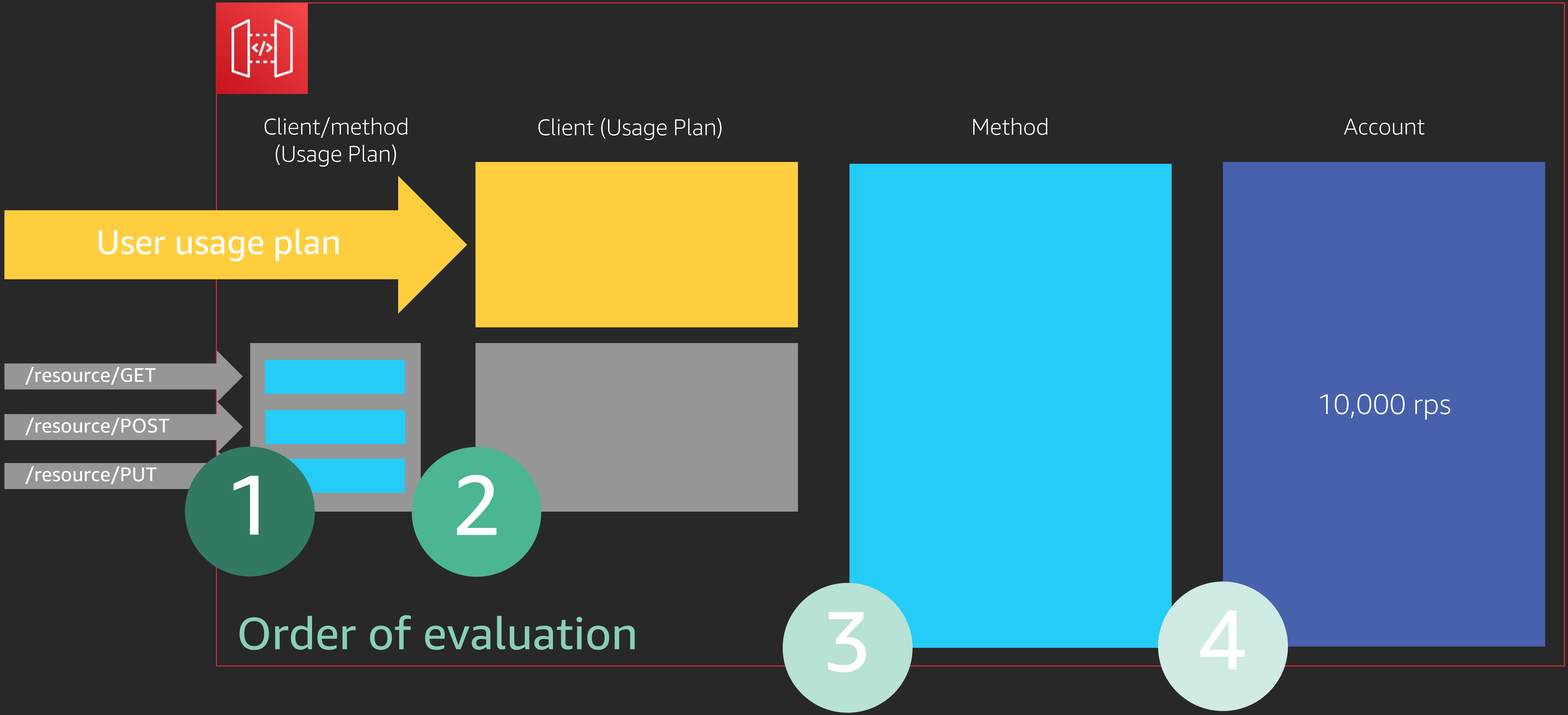
- Cache
- CloudFront

Authentication and authorization

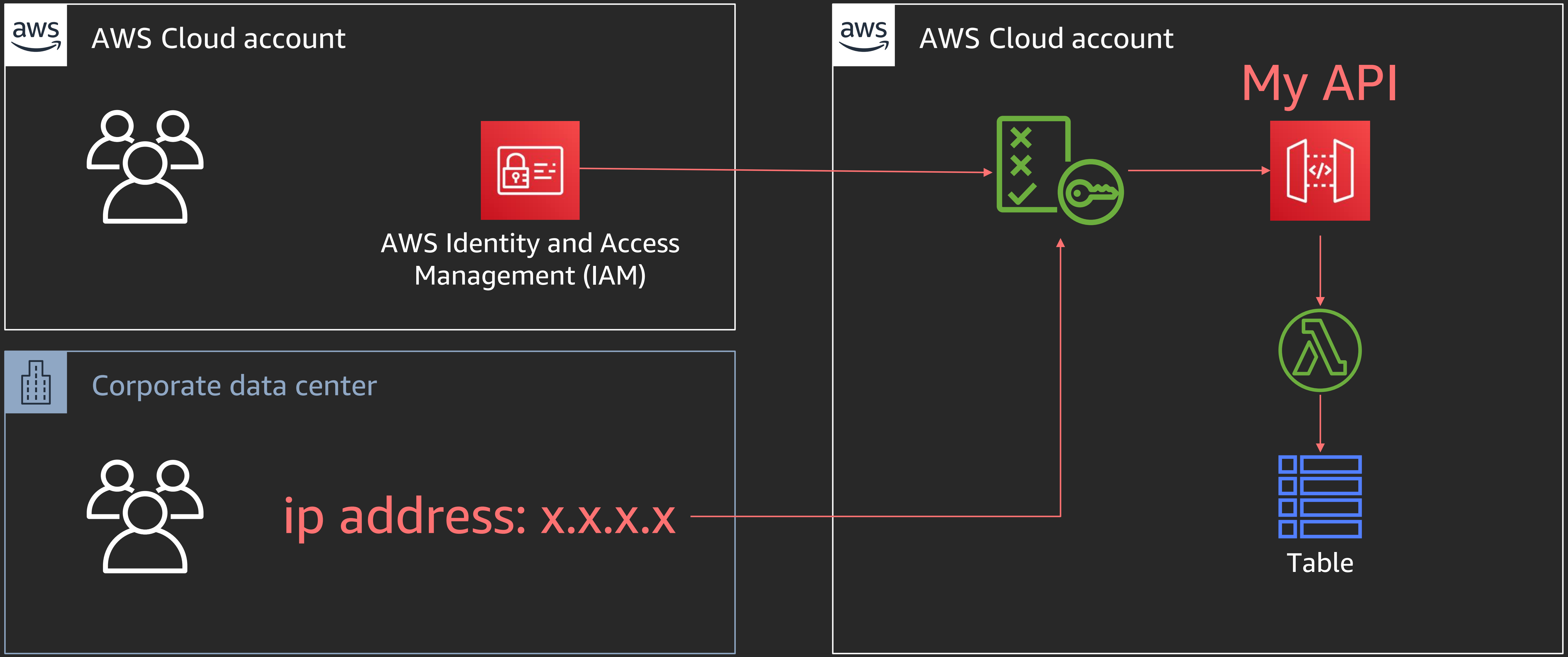


- User pools through Amazon Cognito
- Amazon Cognito authorizers on API Gateway

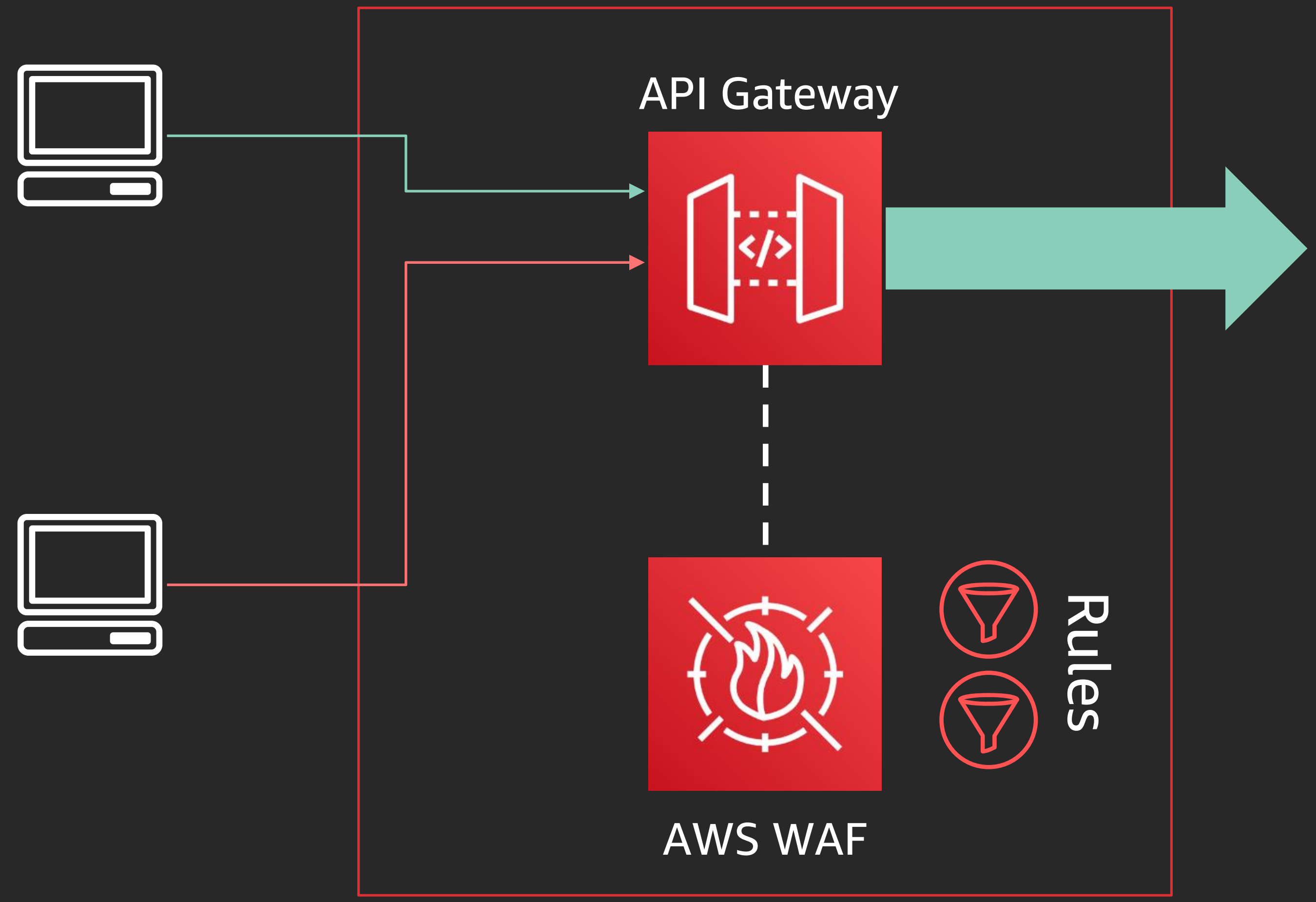
Throttling


Client/method
(Usage Plan)
Client (Usage Plan)
Method
Account
User usage plan
/resource/GET
/resource/POST
/resource/PUT
10,000 rps
1
2
3
4
Order of evaluation

Resource policies



AWS Web Application Firewall (AWS WAF)



- Protect API Gateway APIs from common web exploits, such as SQL injection and cross-site scripting (XSS) attacks
- Block requests from specified IP address ranges or CIDR blocks
- Block requests originating from a specific country or region
- Match specified string or regular expression pattern in HTTP headers, method, query string, URI, and the request body
- Block attacks from specific user-agents, bad bots, and content scrapers

Data modeling and validation

```
{
  deviceType: “angus phone”,
  location: “the house”,
  message: “eating”,
}
```

```
{
  deviceType: “angus phone”,
  message: “eating”,
}
```

```
{
  location: “the house”,
  message: “eating”,
}
```

```
{
  deviceType: “angus phone”,
  location: “the house”,
}
```

```
{
  deviceType: “angus phone”,
  location: “the house”,
  message: { success: true }
}
```

```
{
  "type” : "object",
  "required” : [ "deviceType", "location" ],
  "properties” : {
    "deviceType” : { "type" : "string” },
    "location” : { "type" : "string” },
    "message" : { "type" : "string” }
  }
}
```

Data modeling and validation

```
{
  deviceType: “angus phone”,
  location: “the house”,
  message: “eating”,
}
```

=

```
{
  deviceType: “angus phone”,
  message: “eating”,
}
```

!=

```
{
  location: “the house”,
  message: “eating”,
}
```

!=

```
{
  deviceType: “angus phone”,
  location: “the house”,
}
```

=

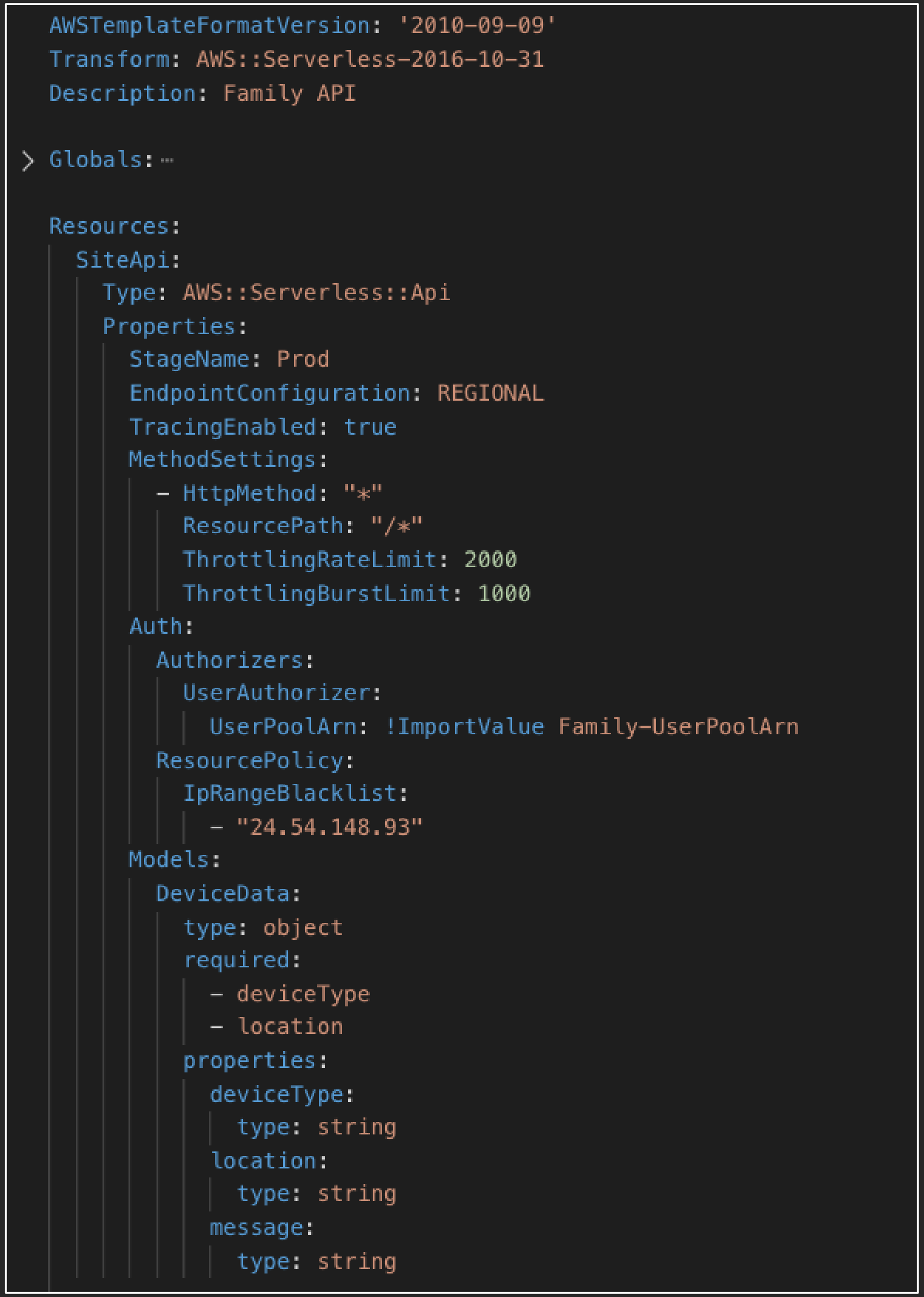
```
{
  deviceType: “angus phone”,
  location: “the house”,
  message: { success: true }
}
```

!=

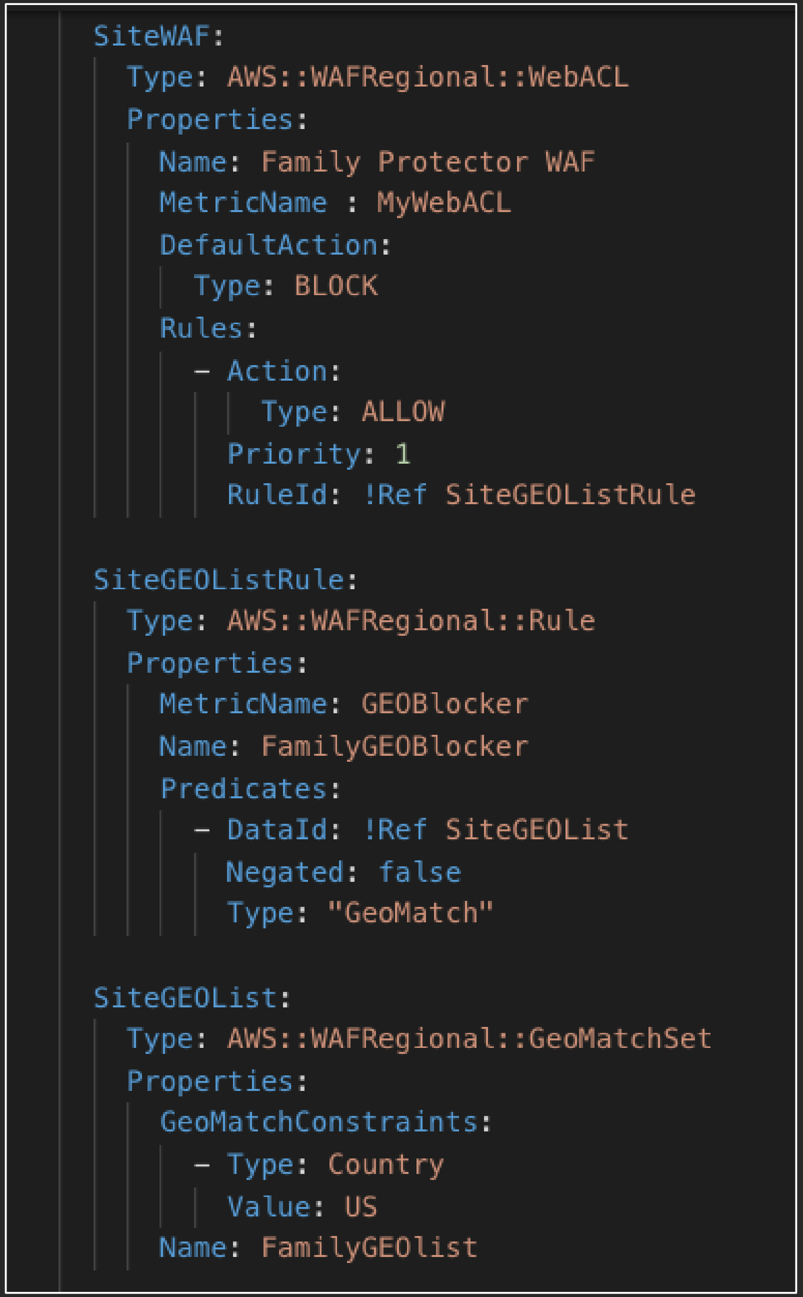
```
{
  "type” : "object",
  "required” : [ "deviceType", "location" ],
  "properties” : {
    "deviceType” : { "type" : "string” },
    "location” : { "type" : "string” },
    "message" : { "type" : "string” }
  }
}
```

Show me code already!

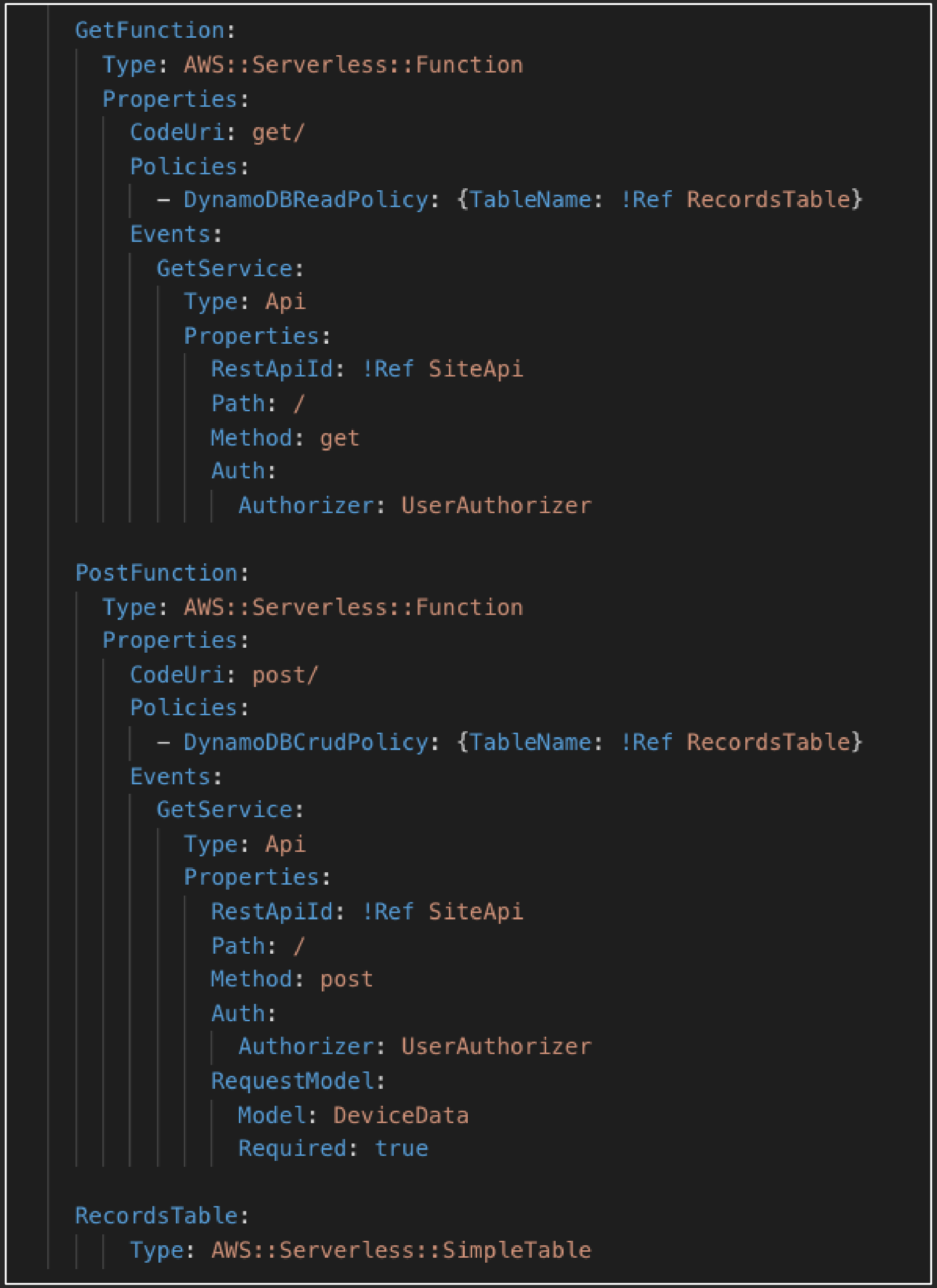
Phase two summary



```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Family API

Globals:⋯

Resources:
  SiteApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      EndpointConfiguration: REGIONAL
      TracingEnabled: true
      MethodSettings:
        - HttpMethod: "*"
          ResourcePath: "/*"
          ThrottlingRateLimit: 2000
          ThrottlingBurstLimit: 1000
      Auth:
        Authorizers:
          UserAuthorizer:
            UserPoolArn: !ImportValue Family-UserPoolArn
        ResourcePolicy:
          IpRangeBlacklist:
            - "24.54.148.93"
      Models:
        DeviceData:
          type: object
          required:
            - deviceType
            - location
          properties:
            deviceType:
              type: string
            location:
              type: string
            message:
              type: string
```

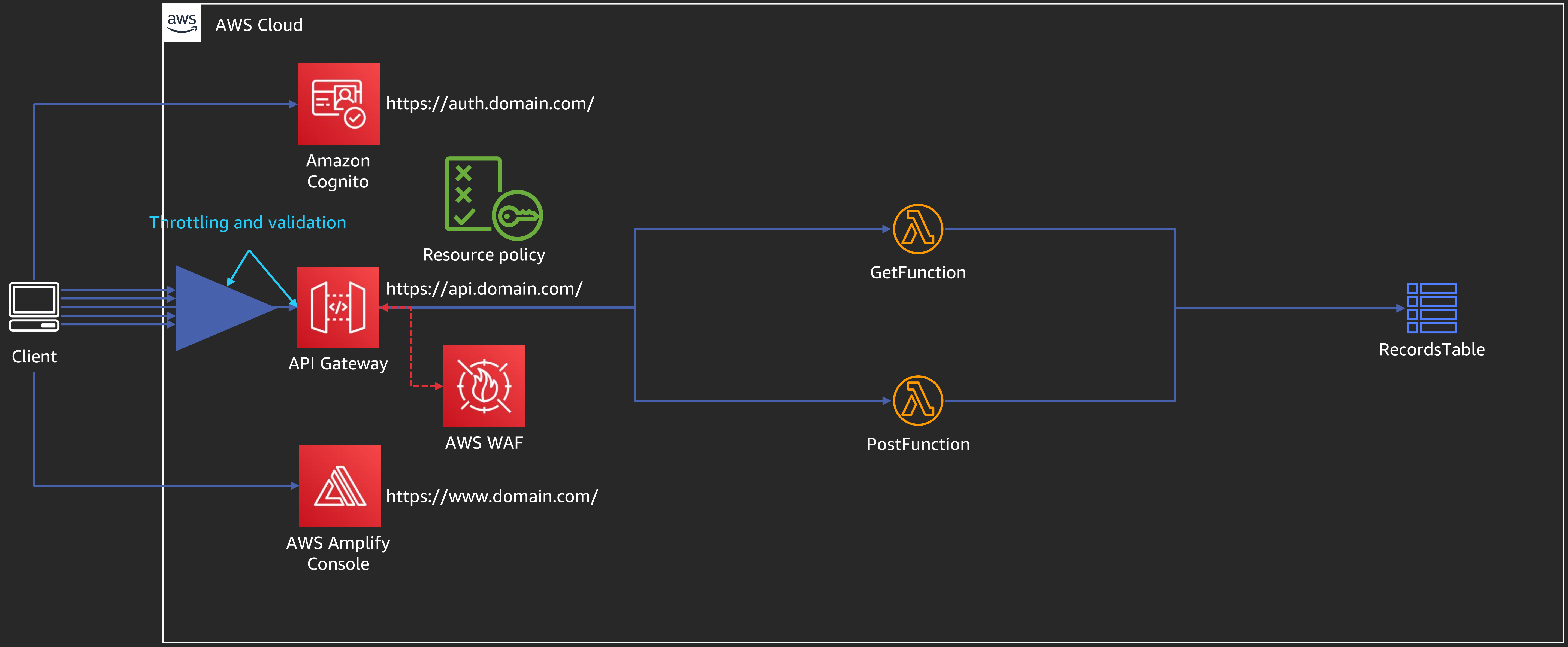


```yaml
  SiteWAF:
    Type: AWS::WAFRegional::WebACL
    Properties:
      Name: Family Protector WAF
      MetricName : MyWebACL
      DefaultAction:
        Type: BLOCK
      Rules:
        - Action:
            Type: ALLOW
          Priority: 1
          RuleId: !Ref SiteGEOListRule

  SiteGEOListRule:
    Type: AWS::WAFRegional::Rule
    Properties:
      MetricName: GEOBlocker
      Name: FamilyGEOBlocker
      Predicates:
        - DataId: !Ref SiteGEOList
          Negated: false
          Type: "GeoMatch"

  SiteGEOList:
    Type: AWS::WAFRegional::GeoMatchSet
    Properties:
      GeoMatchConstraints:
        - Type: Country
          Value: US
      Name: FamilyGEOlist
```



```yaml
  GetFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: get/
      Policies:
        - DynamoDBReadPolicy: {TableName: !Ref RecordsTable}
      Events:
        GetService:
          Type: Api
          Properties:
            RestApiId: !Ref SiteApi
            Path: /
            Method: get
            Auth:
              Authorizer: UserAuthorizer

  PostFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: post/
      Policies:
        - DynamoDBCrudPolicy: {TableName: !Ref RecordsTable}
      Events:
        GetService:
          Type: Api
          Properties:
            RestApiId: !Ref SiteApi
            Path: /
            Method: post
            Auth:
              Authorizer: UserAuthorizer
            RequestModel:
              Model: DeviceData
              Required: true

  RecordsTable:
    Type: AWS::Serverless::SimpleTable
```

Phase two summary



Phase three: A change in requirements

AWS re:Invent

Meet Rufus and Beatrice

- New family members
- Same goals for tracking
- Need a simple device

Image Source: https://pixabay.com/vectors/boy-comic-characters-dad-daughter-1299084/

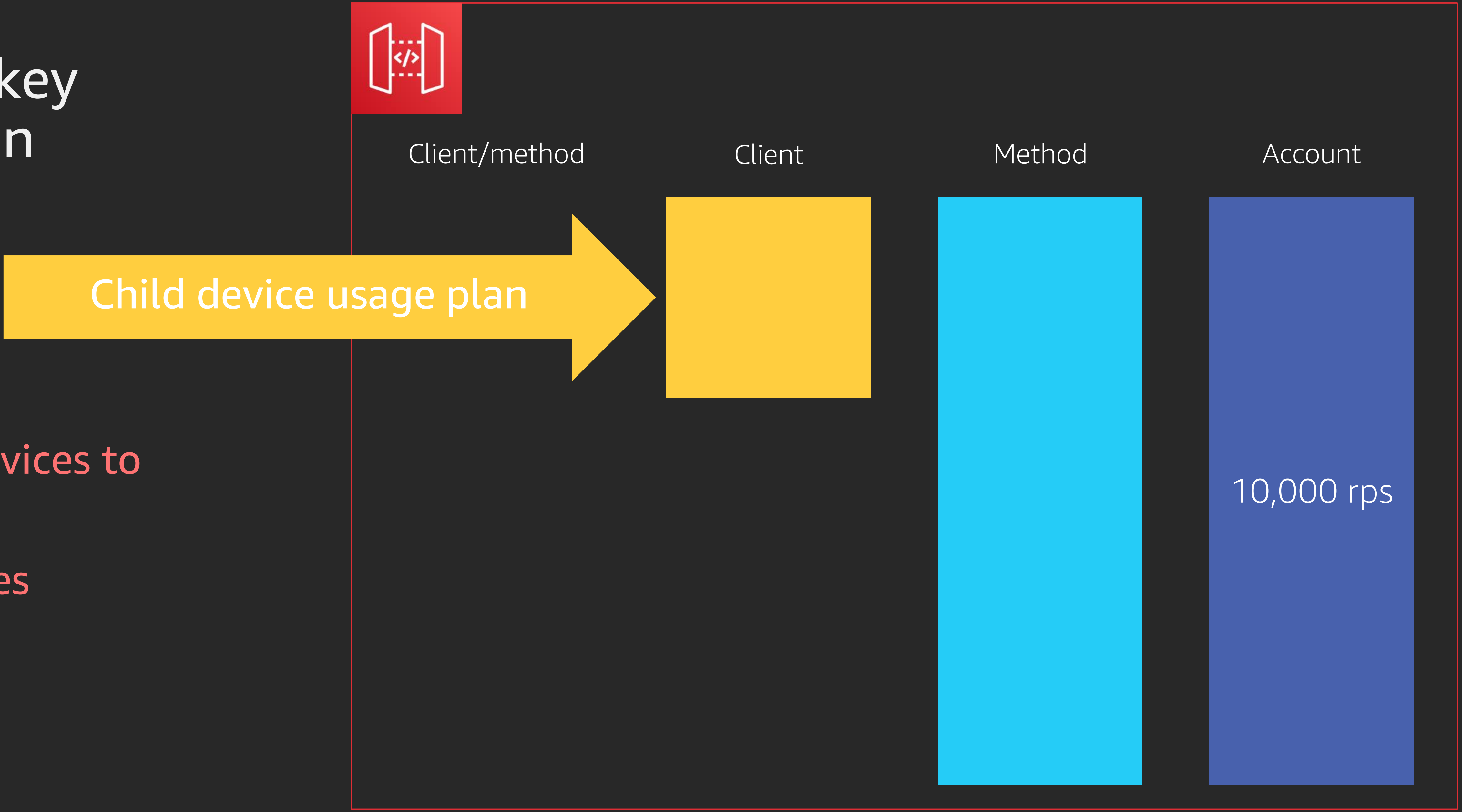
Challenge

Simple phone-location service can be too chatty

Image Source: https://pixabay.com/illustrations/smartphone-tablet-emoji-yellow-3170621/

Solution: API key

Require an API key and a usage plan



- API key allows devices to connect to API
- Data plan throttles connections

Challenge

Simple phone cannot modify outgoing payload

Image Source: https://pixabay.com/illustrations/smartphone-tablet-emoji-yellow-3170621/

Solution: Transform the data

```
{
  deviceType: “”,
  location: “”,
  message: “”,
}
```

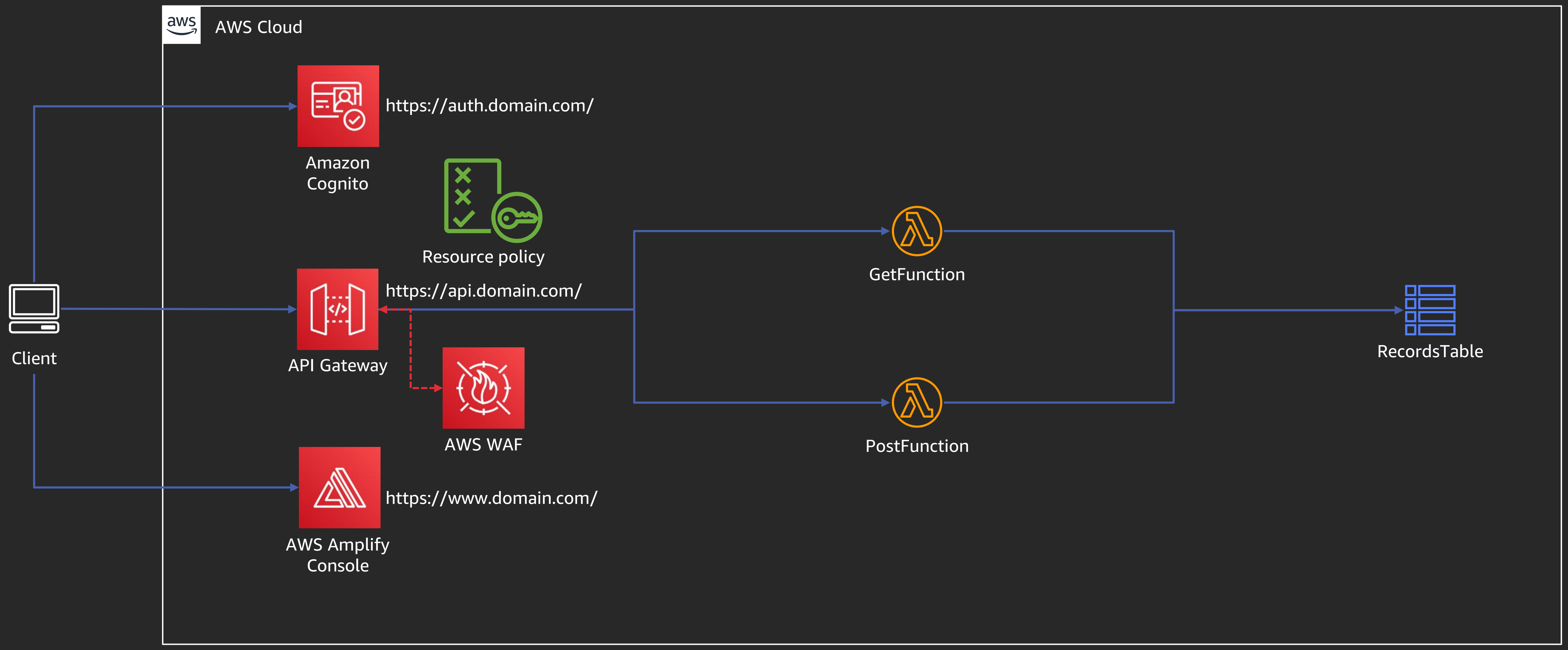
Current schema

```
{
  deviceId: “”,
  geoCoord: “”,
}
```

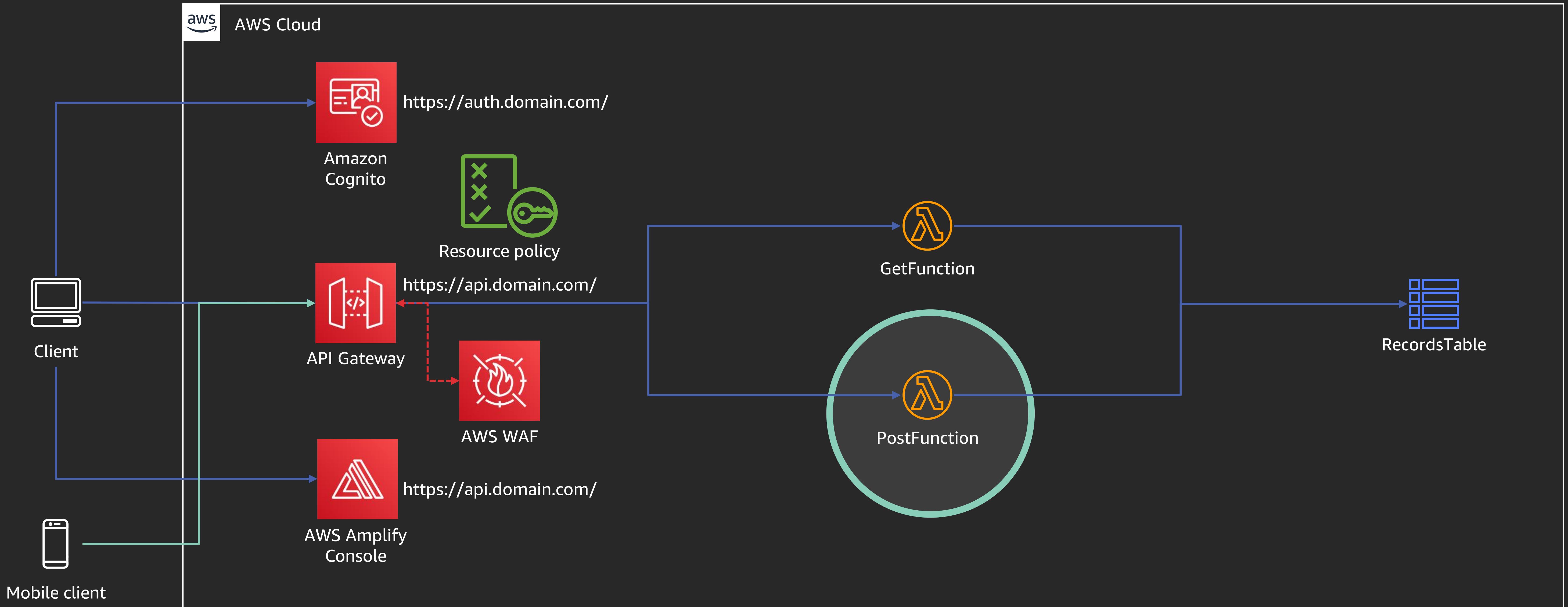
Device schema

Image Source: https://pixabay.com/illustrations/smartphone-tablet-emoji-yellow-3170621/

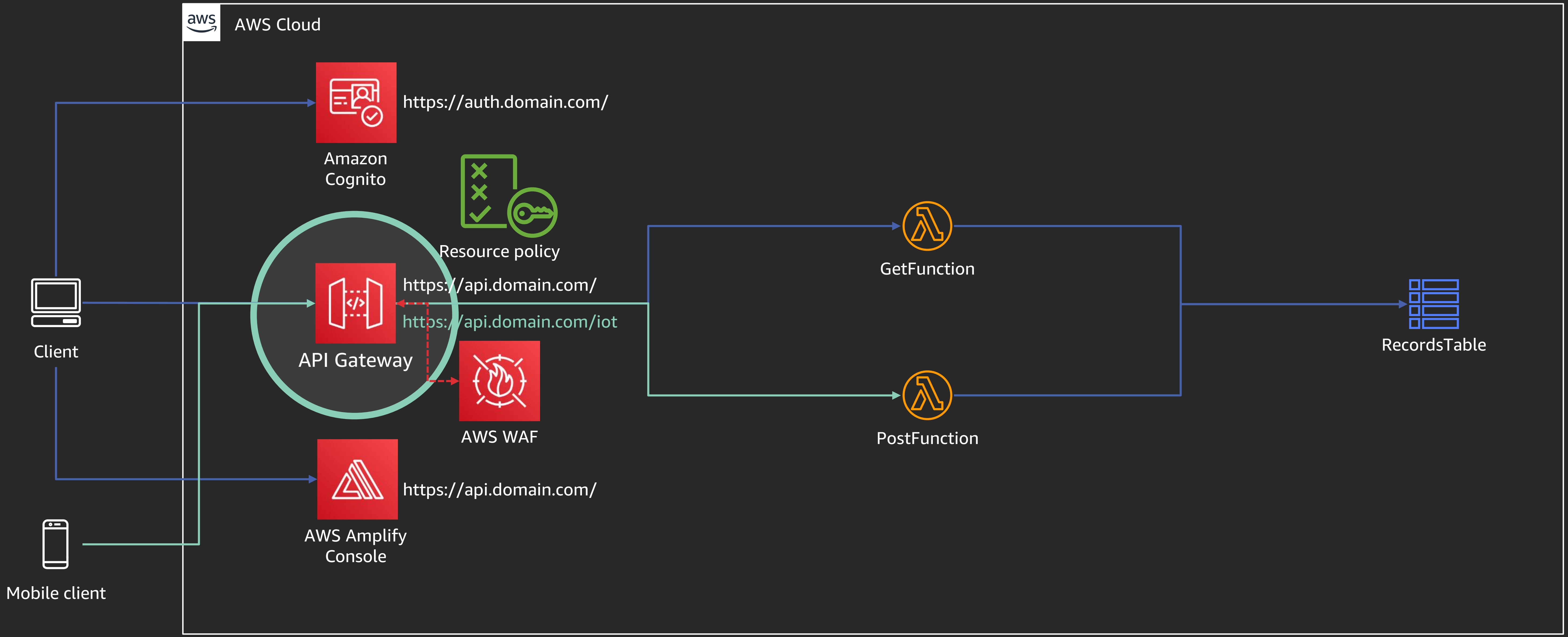
Where to handle the transformation?



Option A: Transform at the Lambda function



Option B: Transform at the API Gateway



Solution: Mapping template

Input

```
{
   deviceId: “”,
   geoCoord: “”,
}
```

Required

```
{
   deviceType: “”,
   location: “”,
   message: “”,
}
```

Data transformation

Solution: Mapping template

Input

```
{
  deviceId: “”,
  geoCoord: “”,
}
```

Mapping template

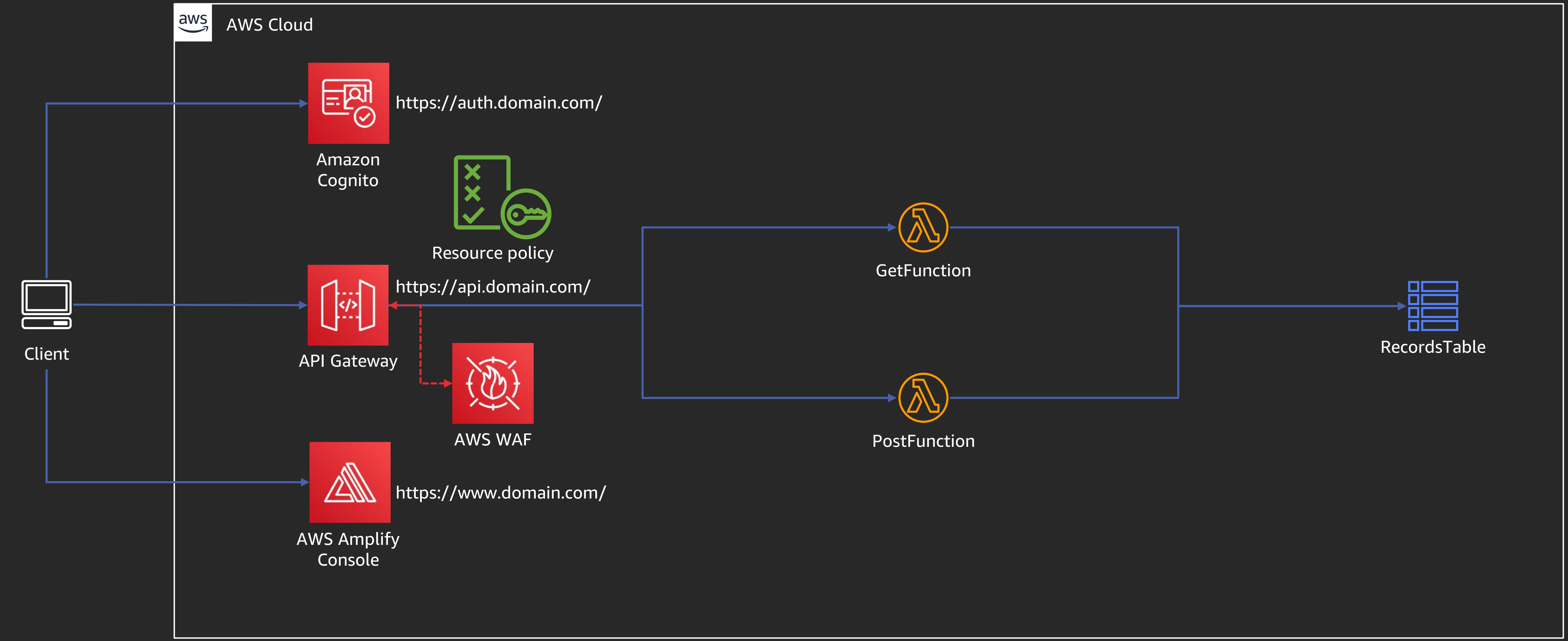
```
#set($inputRoot = $input.path('$’))
{
  ”deviceType": $inputRoot.deviceId,
  ”location": $inputRoot.geoCoord,
  ”message”: “NA”
}
```

Output

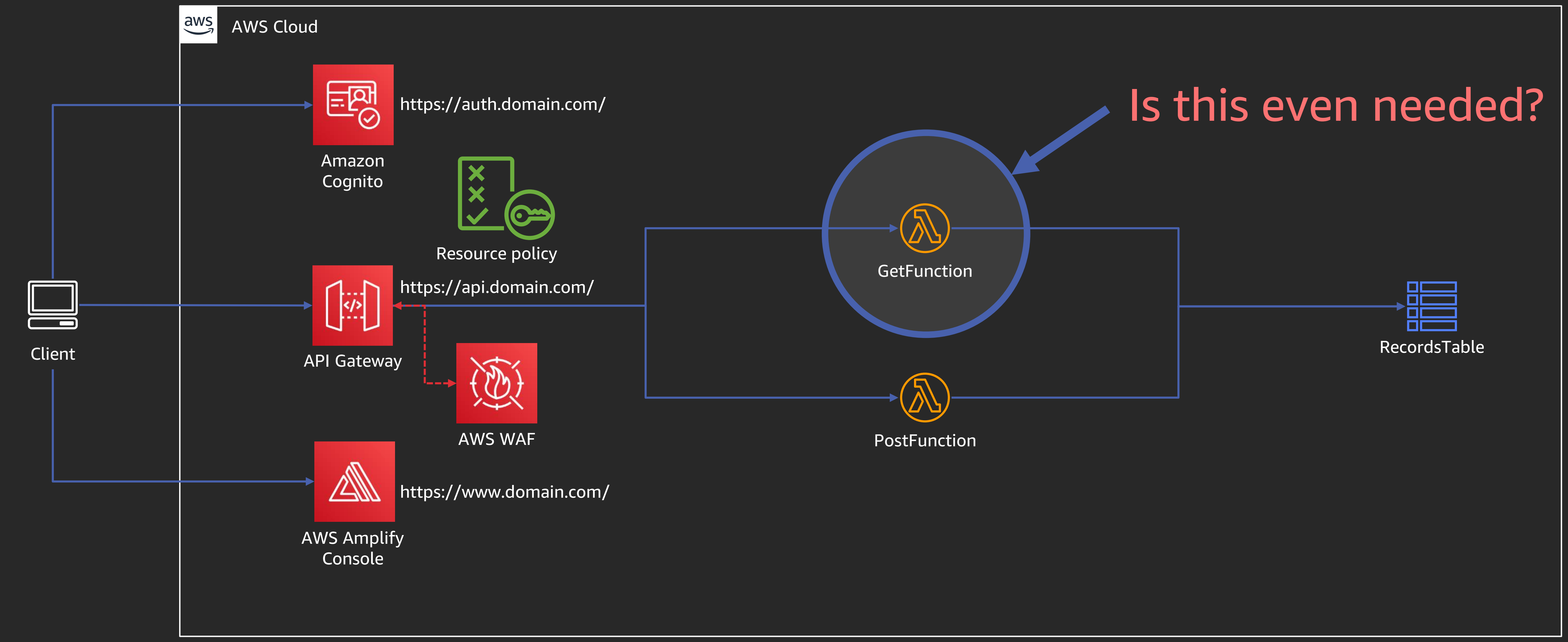
```
{
  deviceType: “”,
  location: “”,
  message: “”,
}
```

Using mapping templates allows you to reformat data as needed

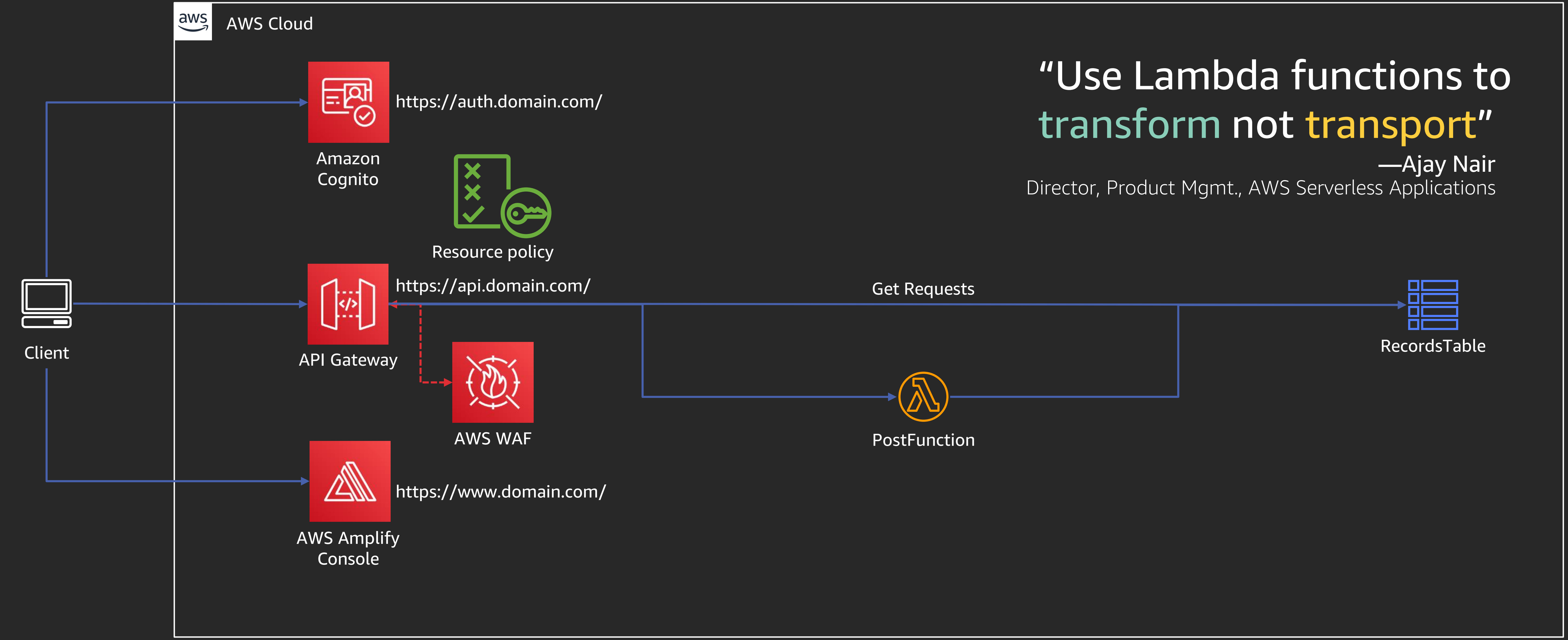
More with mapping templates



More with mapping templates



Service integration

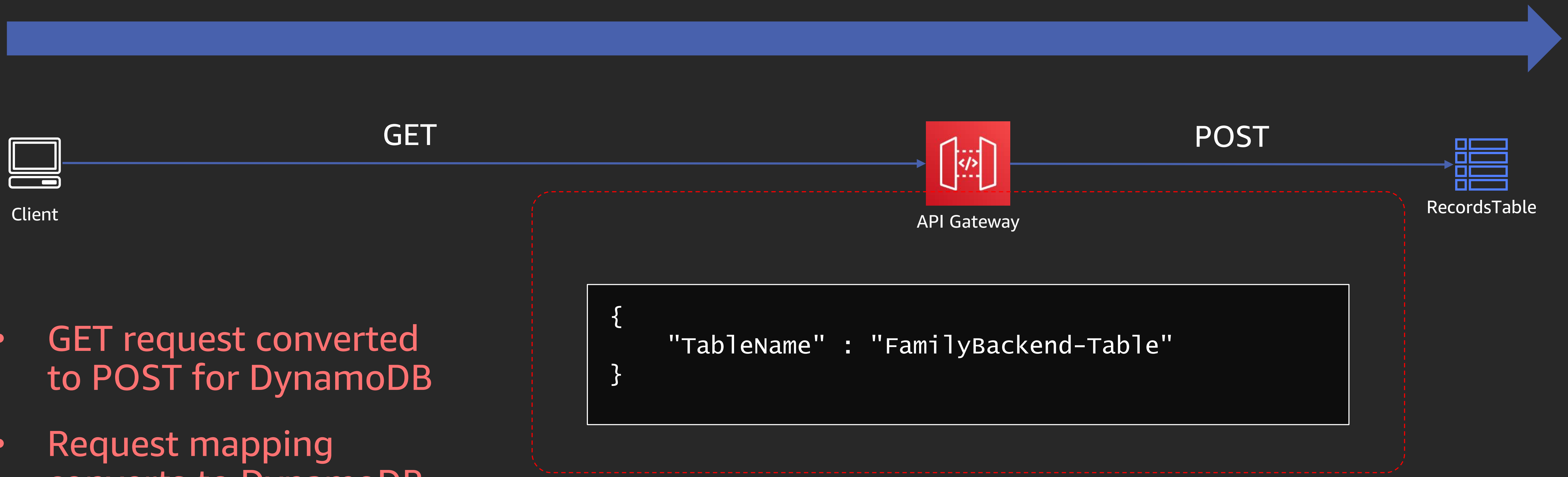


“Use Lambda functions to transform not transport”

—Ajay Nair

Director, Product Mgmt., AWS Serverless Applications

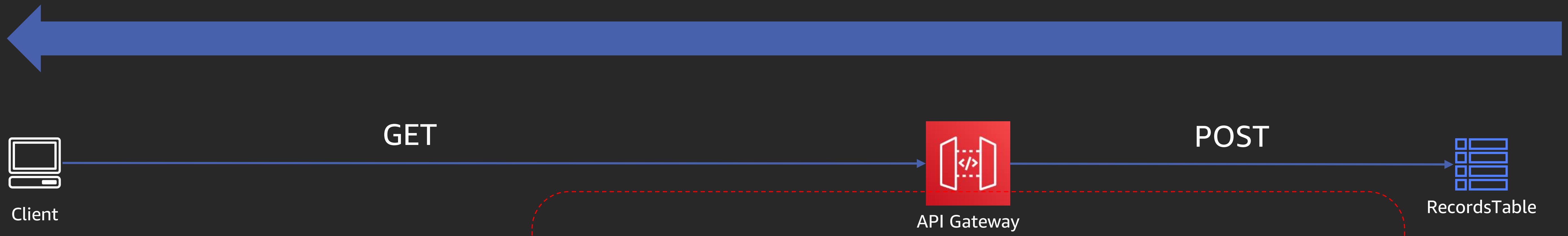
Service integration request mapping template



```
{
    "TableName" : "FamilyBackend-Table"
}
```

- GET request converted to POST for DynamoDB
- Request mapping converts to DynamoDB scan request

Service integration response mapping template

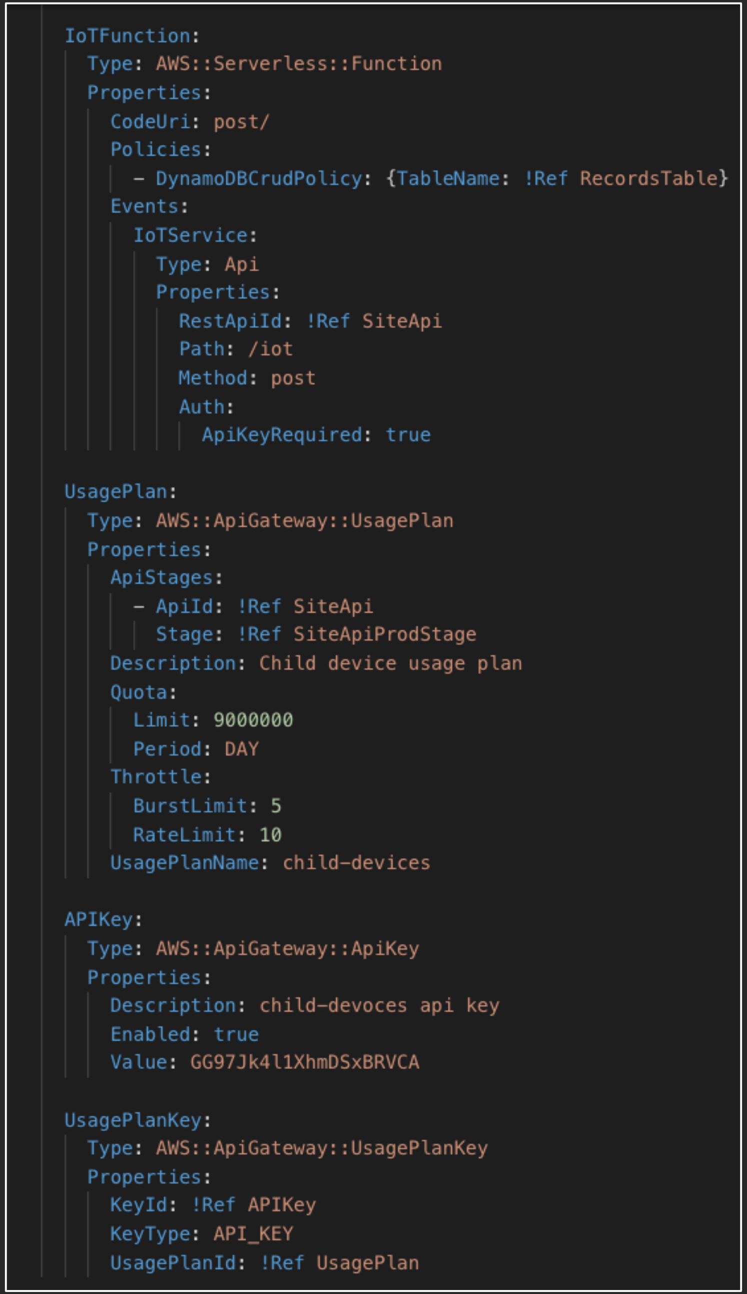


- Response mapping template converts data from DynamoDB schema

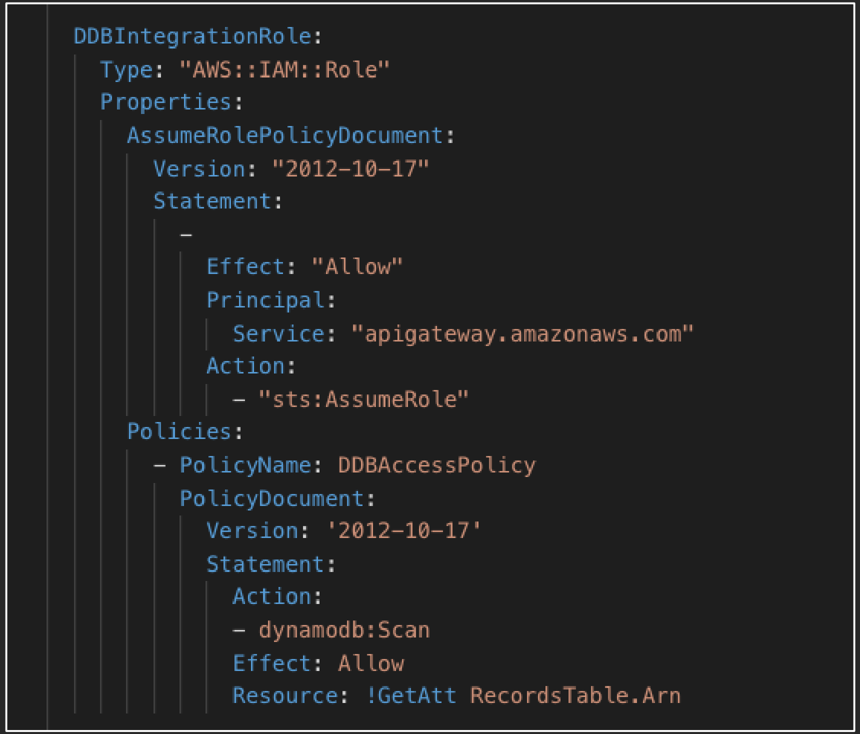
```
#set($inputRoot = $input.path('$'))
    [
        #foreach($elem in $inputRoot.Items) {
            "deviceType":"$elem.deviceType.S",
            "location": "$elem.location.S",
            "message": "$elem.message.S",
            "timestamp": $elem.timestamp.N,
            "id": "$elem.id.S"
        }
            #if($foreach.hasNext),
            #end
        #end
    ]
```

And, show me the code!

Phase three summary

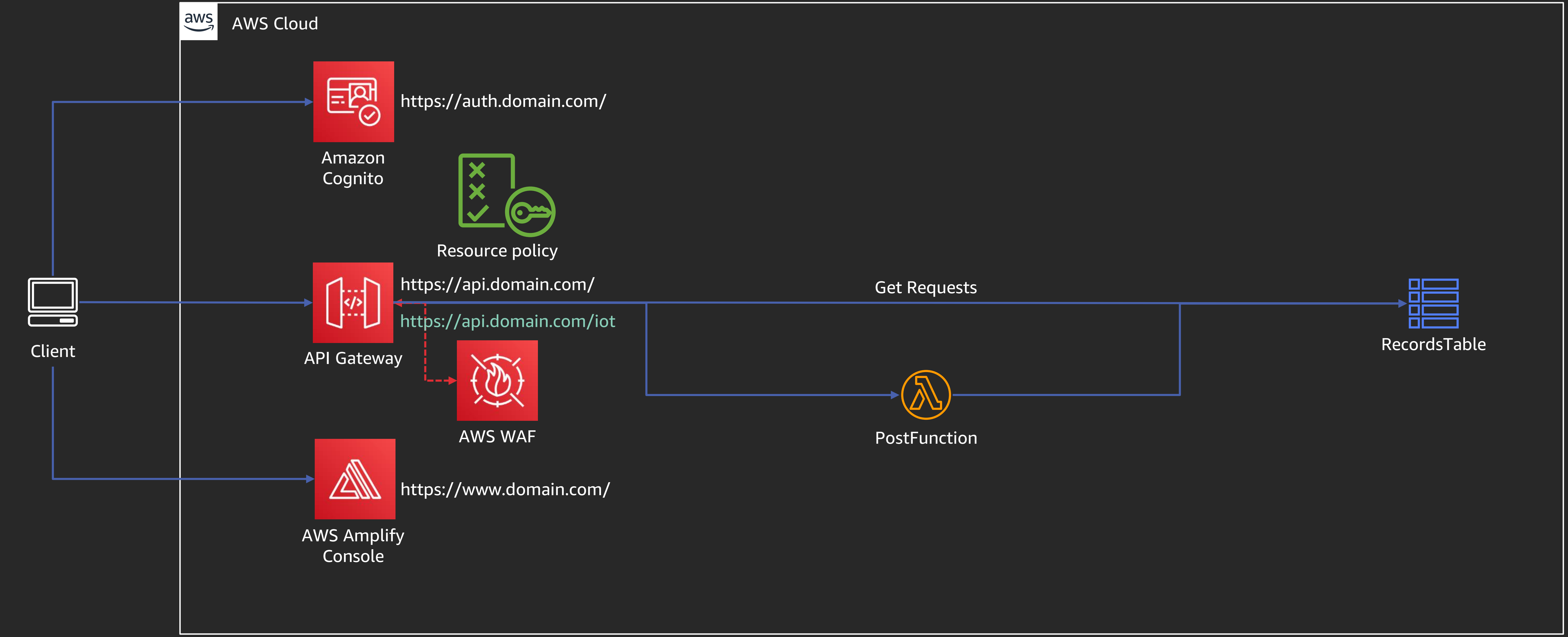


```yaml
IoTFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: post/
    Policies:
      - DynamoDBCrudPolicy: {TableName: !Ref RecordsTable}
    Events:
      IoTService:
        Type: Api
        Properties:
          RestApiId: !Ref SiteApi
          Path: /iot
          Method: post
          Auth:
            ApiKeyRequired: true

UsagePlan:
  Type: AWS::ApiGateway::UsagePlan
  Properties:
    ApiStages:
      - ApiId: !Ref SiteApi
        Stage: !Ref SiteApiProdStage
    Description: Child device usage plan
    Quota:
      Limit: 9000000
      Period: DAY
    Throttle:
      BurstLimit: 5
      RateLimit: 10
    UsagePlanName: child-devices

APIKey:
  Type: AWS::ApiGateway::ApiKey
  Properties:
    Description: child-devoces api key
    Enabled: true
    Value: GG97Jk4l1XhmDSxBRVCA

UsagePlanKey:
  Type: AWS::ApiGateway::UsagePlanKey
  Properties:
    KeyId: !Ref APIKey
    KeyType: API_KEY
    UsagePlanId: !Ref UsagePlan
```



```yaml
DDBIntegrationRole:
  Type: "AWS::IAM::Role"
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service: "apigateway.amazonaws.com"
          Action:
            - "sts:AssumeRole"
    Policies:
      - PolicyName: DDBAccessPolicy
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            Action:
            - dynamodb:Scan
            Effect: Allow
            Resource: !GetAtt RecordsTable.Arn
```

Phase three summary



Final thoughts

Final thoughts

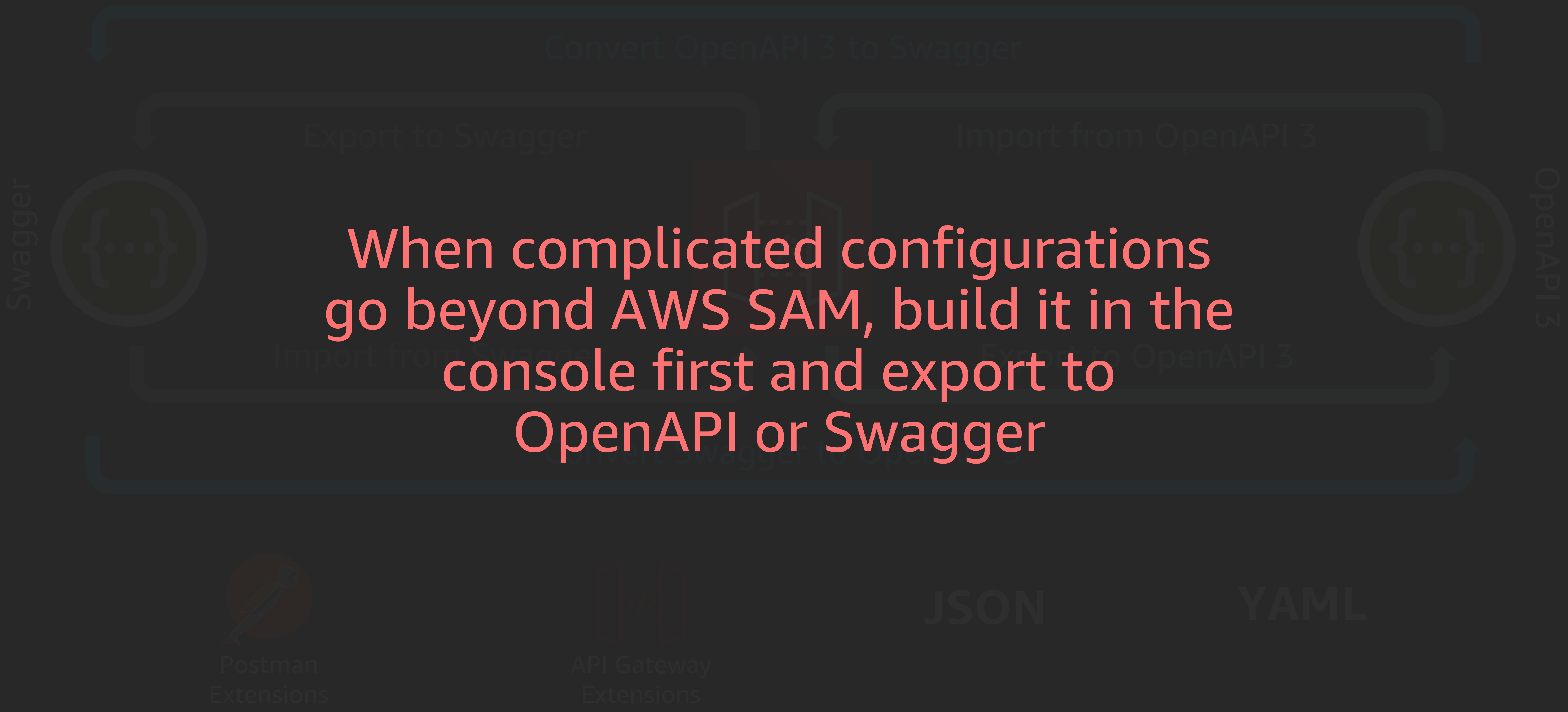
- Base website
- Authentication/authorization
- Throttling
- Resource policies
- AWS WAF
- API key/usage plan
- Mapping templates
- Service integration

Final thoughts

- Base website
- Authentication/authorization
- Throttling
- Resource policies
- AWS WAF
- API key / usage plan
- Mapping templates
- Service integration

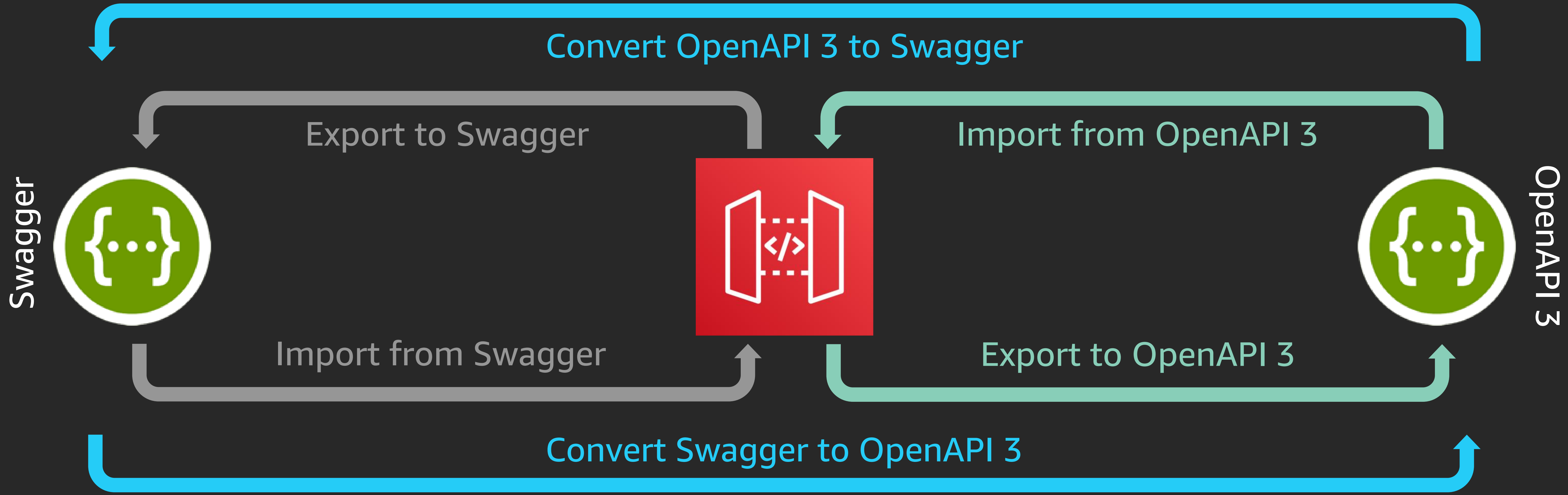
And … we used AWS SAM for most of it!

Final thoughts



When complicated configurations go beyond AWS SAM, build it in the console first and export to OpenAPI or Swagger

Final thoughts



Postman extensions

API Gateway extensions

JSON

YAML

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

aws training and certification

Thank you!

Eric Johnson

@edjgeek

AWS re:Invent

Please complete the session survey in the mobile app.