Toronto, CA | September 11th, 2024

aws SUMMIT

DEV101

Build a Serverless API in Go

Dr. Joanne Skiles

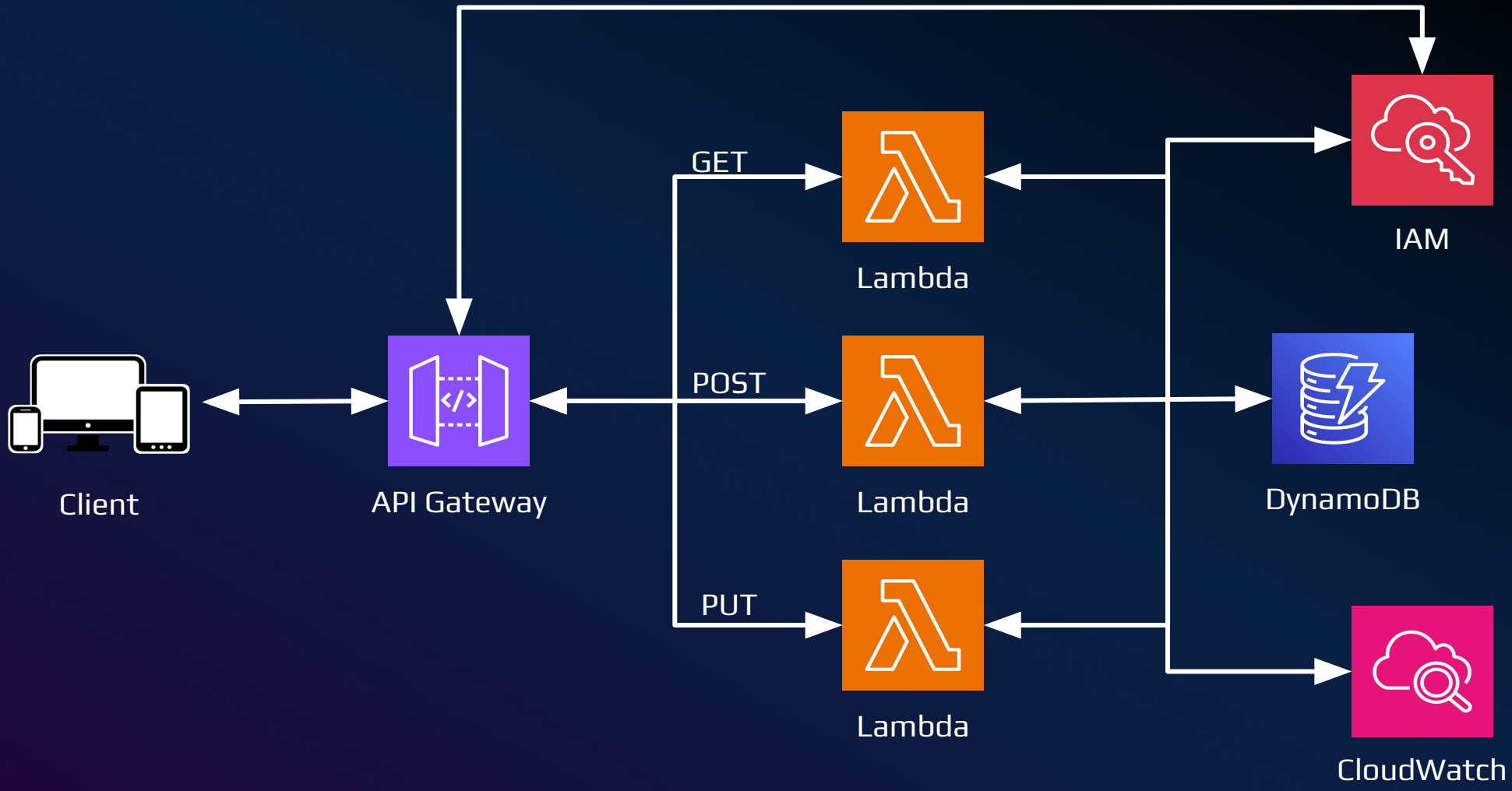
Senior Software Engineering Manager
Capital One

Agenda

1. Overview
2. Why GoLang for Serverless Development
3. Building the REST API
4. Integrating with AWS Services
5. Deployment and Management
6. Best Practices
7. Conclusion

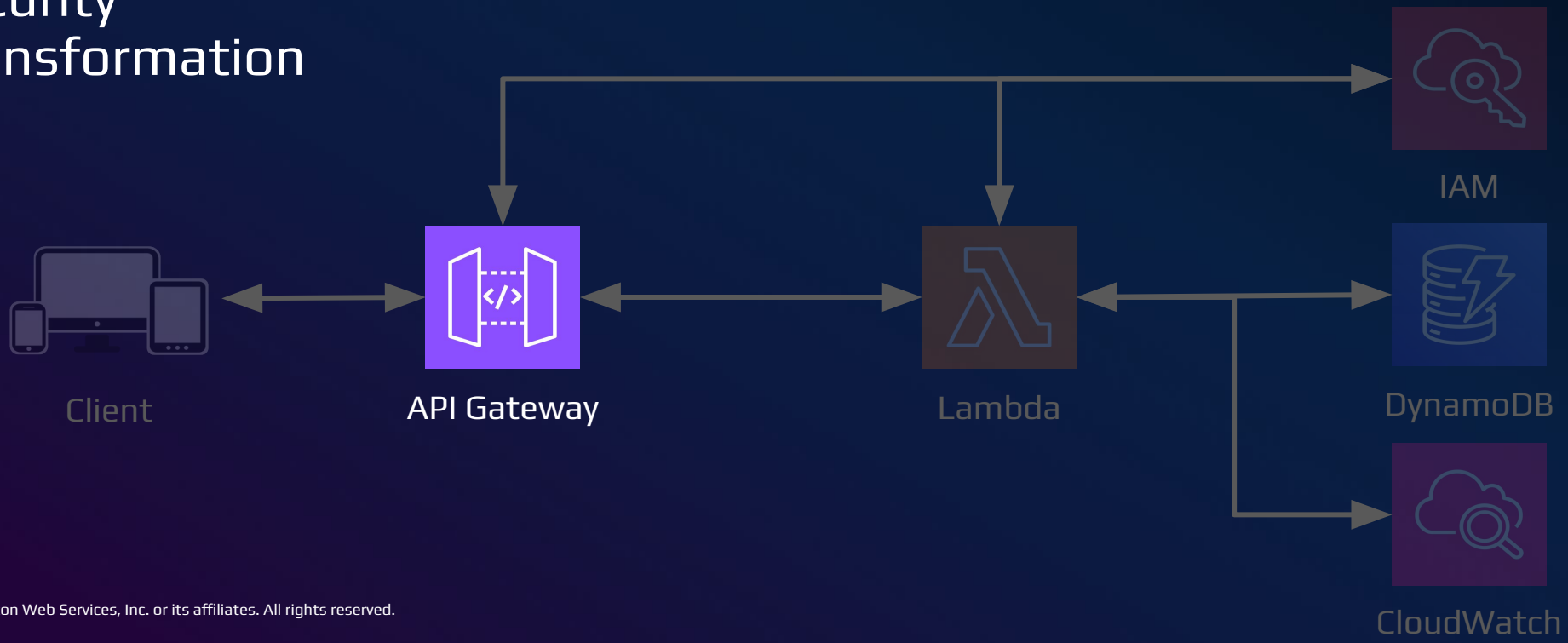
Overview

What are we building?



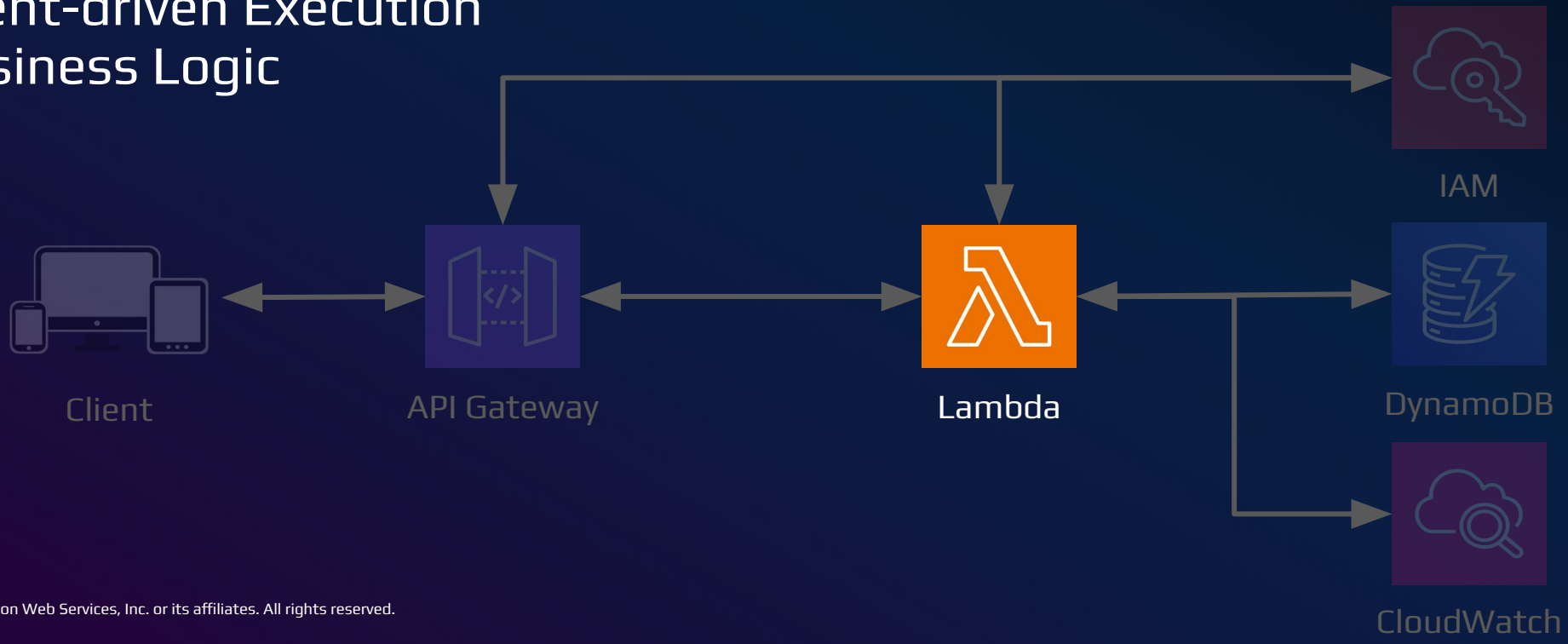
API Gateway

- Acts as the entry point for all client requests.
  - Routes HTTP requests to the appropriate AWS Lambda function.
- Features:
  - Request Routing
  - Security
  - Transformation



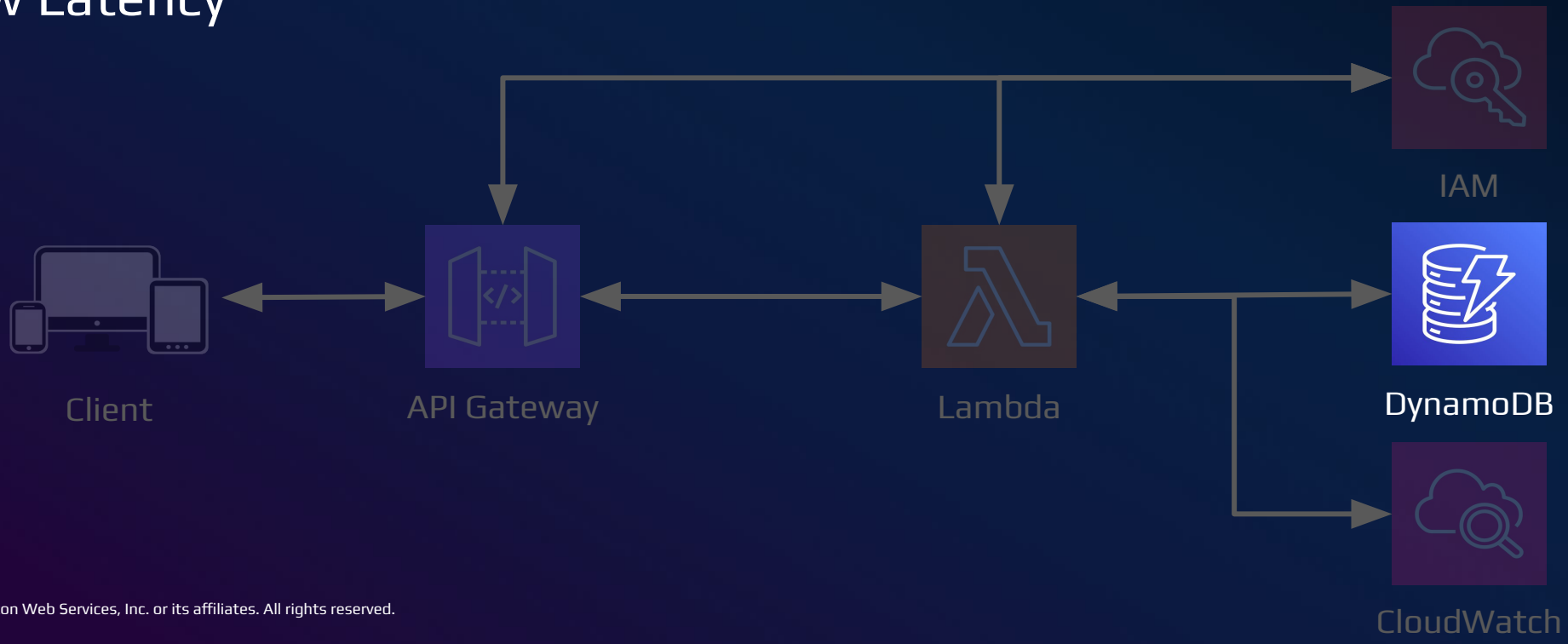
AWS Lambda

- Hosts the Go-based business logic that processes incoming requests and returns responses.
- Features:
  - Stateless Execution
  - Event-driven Execution
  - Business Logic



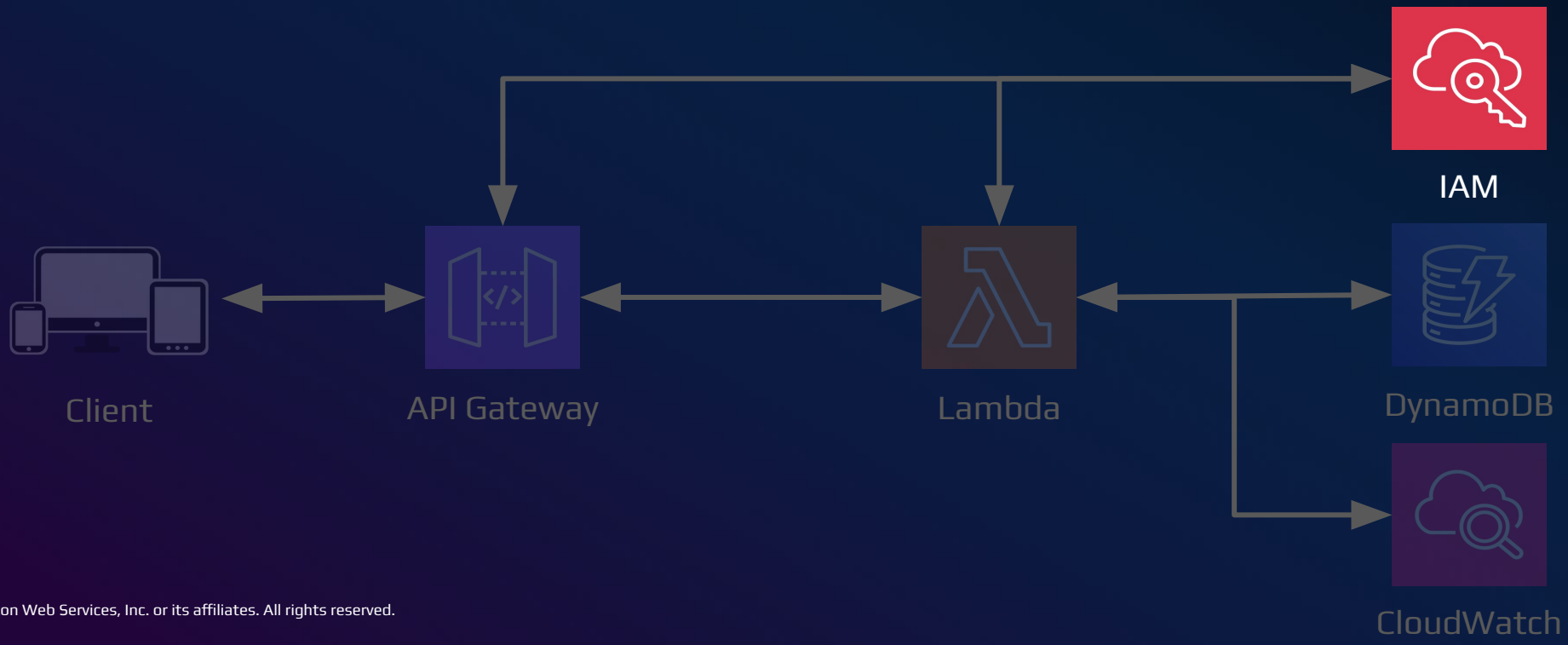
DynamoDB

- Serves as a NoSQL database for persistent data storage
- Features:
  - Scalability
  - Event-driven Integration
  - Low Latency



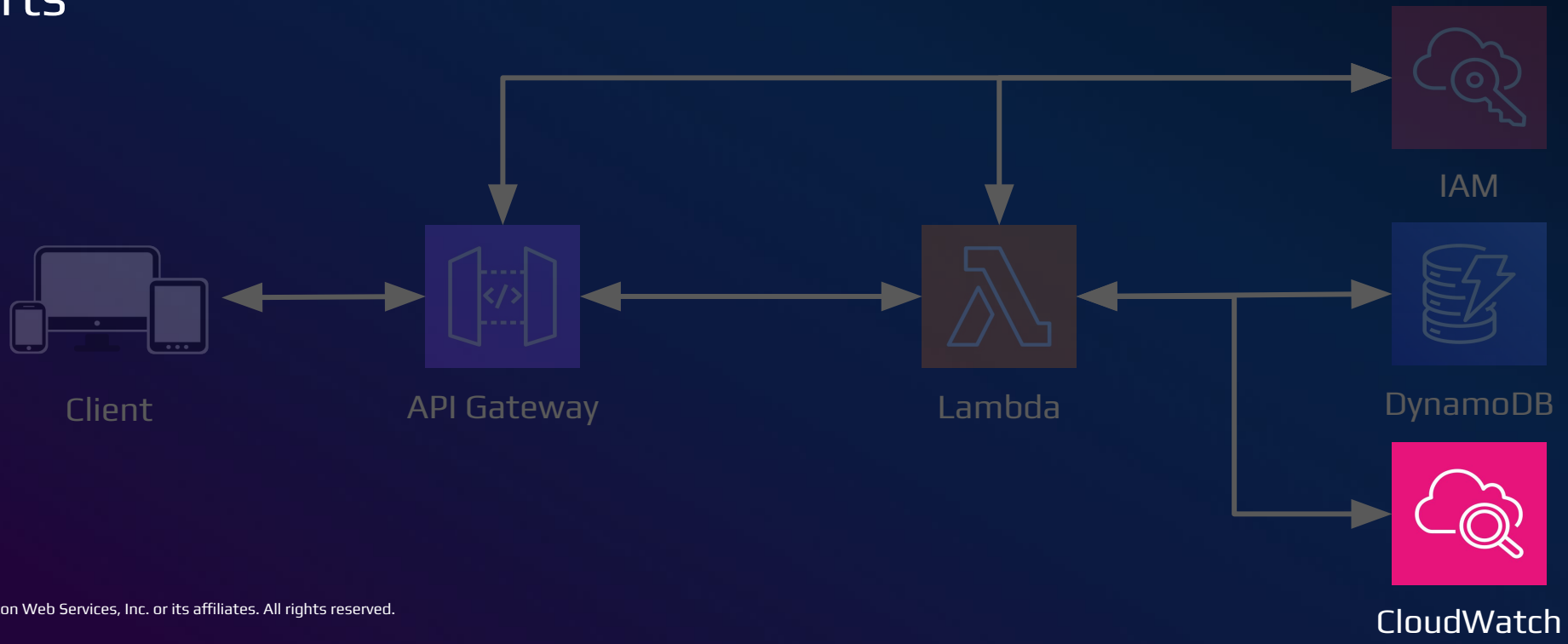
AWS IAM

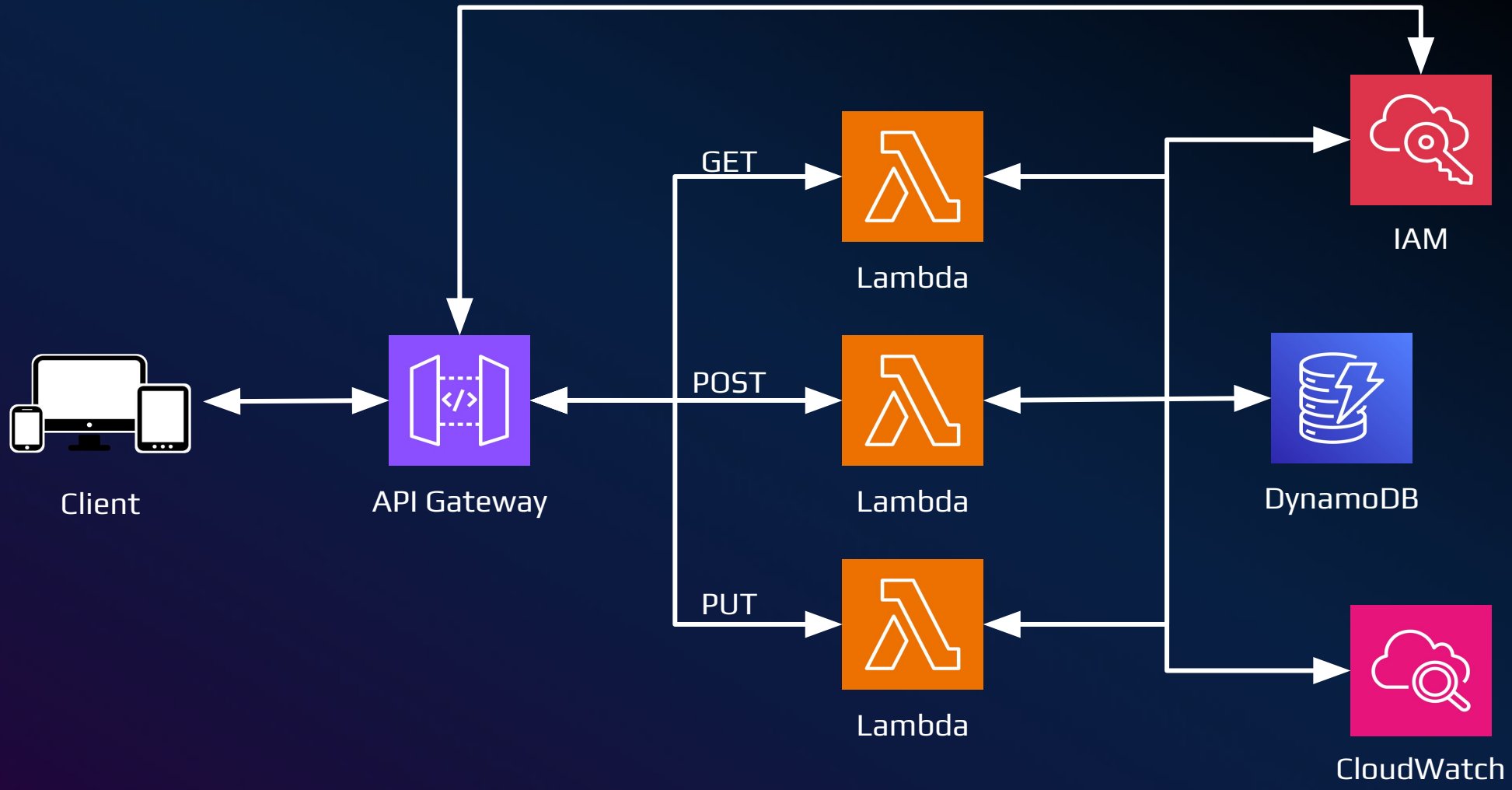
- Manages permissions and security for accessing AWS services.
- Features:
  - Lambda Permissions
  - API Gateway Access



CloudWatch

- Monitors and logs API requests and Lambda executions
- Features:
  - Logging
  - Metrics
  - Alerts

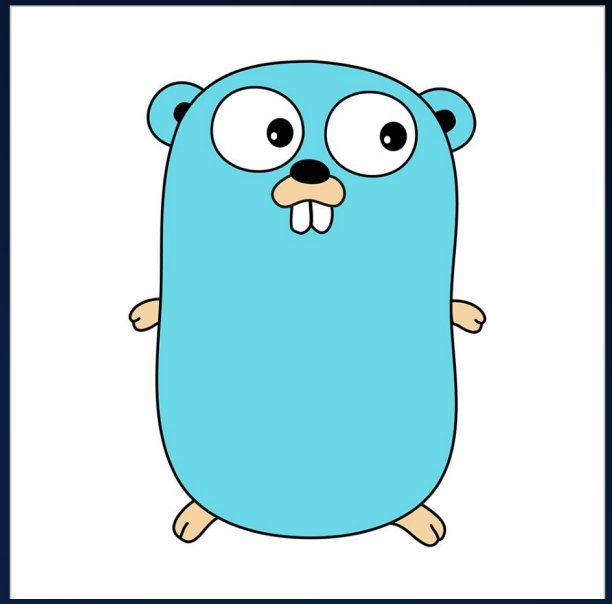




GoLang for Serverless Development

Why GoLang for Serverless?

- High Performance
  - Compiled Language
  - Efficient Concurrency
- Simplicity & Readability
  - Clean Syntax
  - Minimalistic Design
- Small Binary Size
  - Optimized Deployment
  - Easy Distribution
- Strong Ecosystem & Tooling
  - Mature Libraries
  - Built-in Support
- Growing Adoption in Cloud Computing
  - Industry Adoption
  - Community & Support

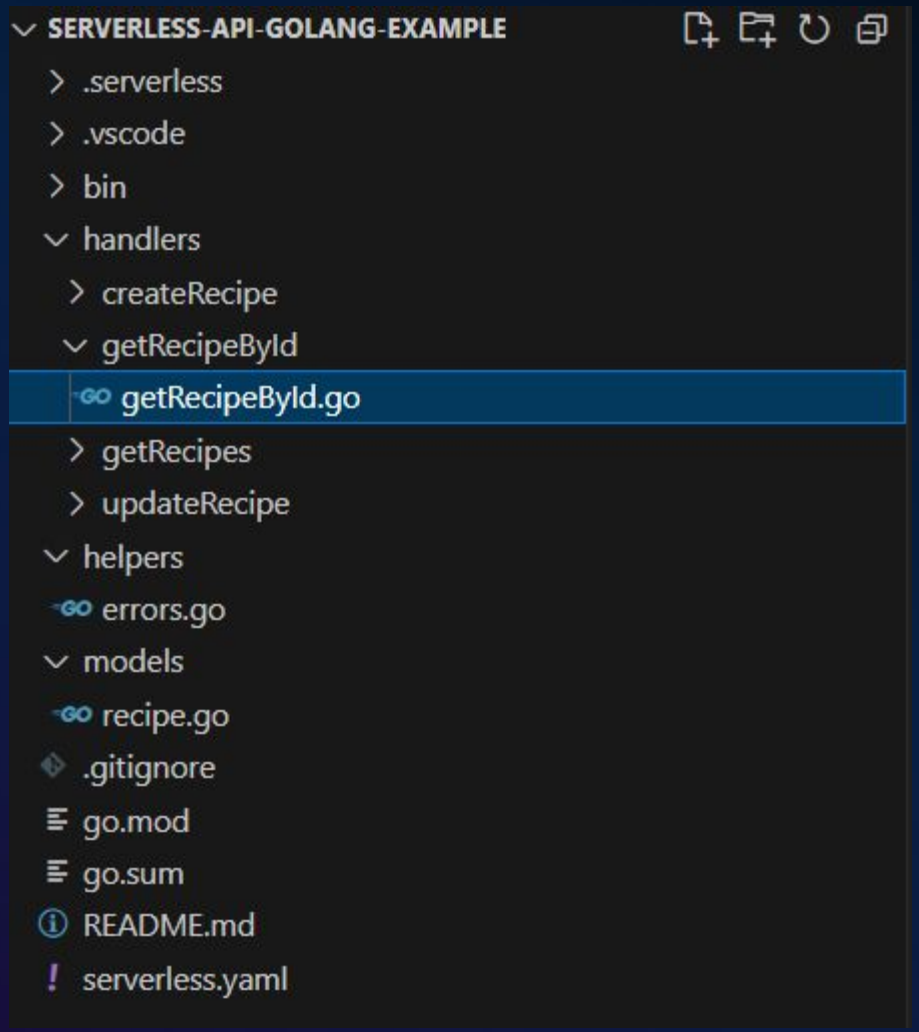


When NOT to Use Go

- Cold Start Sensitivity
- Limited Native Library Support
  - Specialized Libraries
  - Community Support
- Complex Build and Deployment
  - Cross-Compilation
  - Binary Size
- High Development Speed Requirements
- Frequent Code Changes

Building/Deploying the REST API

Setting Up the Code



Recipe Struct

```go
package models

type Recipe struct {
    ID           string   `json:"id"`
    Title        string   `json:"title"`
    Ingredients  []string `json:"ingredients"`
    Instructions []string `json:"instructions"`
}
```

Example Function

```go
func getRecipeByID(ctx context.Context, req events.APIGatewayProxyRequest) (events.APIGatewayProxyResponse, error) {
    id := req.PathParameters["id"]
    if id == "" {
        return helpers.ClientError(http.StatusBadRequest, "id is required")
    }

    cfg, err := config.LoadDefaultConfig(ctx)
    if err != nil {
        log.Error().Err(err).Msg("failed to load config")
        return helpers.ServerError(err)
    }

    client := dynamodb.NewFromConfig(cfg)

    output, err := client.GetItem(ctx,
        &dynamodb.GetItemInput{
            TableName: aws.String("Recipes"),
            Key: map[string]types.AttributeValue{
                "id": &types.AttributeValueMemberS{Value: id},
            },
        })

    if err != nil {
        log.Error().Err(err).Msg("failed to get item")
        return helpers.ServerError(err)
    }
```

Example Function

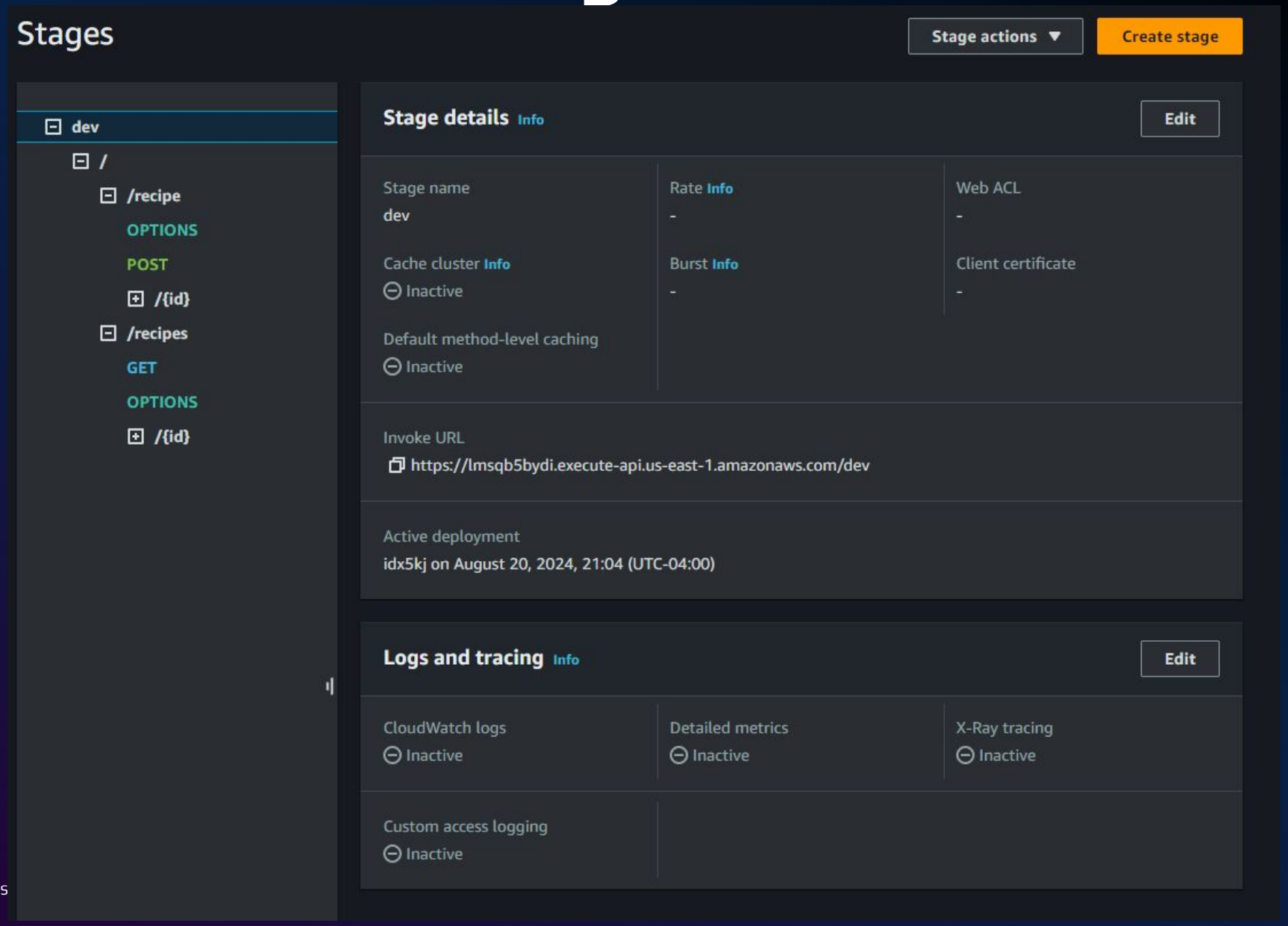
```
    if output.Item == nil {
        return helpers.ClientError(http.StatusNotFound, "recipe not found")
    }

    recipe := models.Recipe{}
    err = attributevalue.UnmarshalMap(output.Item, &recipe)
    if err != nil {
        log.Error().Err(err).Msg("failed to unmarshal recipe")
        return helpers.ServerError(err)
    }

    body, err := json.Marshal(recipe)
    if err != nil {
        log.Error().Err(err).Msg("failed to marshal recipe")
        return helpers.ServerError(err)
    }

    return events.APIGatewayProxyResponse{
        StatusCode: http.StatusOK,
        Body:       string(body),
        Headers: map[string]string{
            "Content-Type": "application/json",
        },
    }, nil
}
```

Serverless.yaml

```yaml
# "org" ensures this Service is used with the correct Serverless Framework Access Key.
org: chaotictoejam
# "app" enables Serverless Framework Dashboard features and sharing them with other Services.
app: serverless-api-go-example
# AWS service name.
service: serverless-api-go-example

custom:
  recipesTableName: ${self:service}-${self:provider.stage}-recipes
  recipesTableArn: # ARNs are addresses of deployed services in AWS space.
    Fn::Join:
    - ":"
    - - arn
      - aws
      - dynamodb
      - Ref: AWS::Region
      - Ref: AWS::AccountId
      - table/${self:custom.recipesTableName}

provider:
  name: aws
  runtime: provided.al2023
  architecture: arm64
  stage: dev
  region: us-east-1
  environment:
    RECIPES_TABLE_NAME: ${self:custom.recipesTableName}
  iamRoleStatements: # Defines what other AWS services our lambda functions can access.
    - Effect: Allow # Allow access to DynamoDB tables.
      Action:
        - dynamodb:Scan
        - dynamodb:GetItem
        - dynamodb:PutItem
        - dynamodb:UpdateItem
        - dynamodb:DeleteItem
      Resource:
        - ${self:custom.recipesTableArn}
```

```yaml
package:
  individually: true

functions:
  createRecipe: ···
  getRecipeById:
    handler: bootstrap
    package:
      artifact: ./build/getRecipeById.zip
    events:
      - http:
          path: recipes/{id}
          method: get
          cors: true
      #- cloudwatchLog: '/aws/lambda/getRecipeById'
  getRecipes: ···
  updateRecipe: ···

resources:
  Resources:
    recipesTable: # Define a new DynamoDB Table resource to store items
      Type: AWS::DynamoDB::Table
      Properties:
        TableName: ${self:custom.recipesTableName}
        ProvisionedThroughput:
          ReadCapacityUnits: 1
          WriteCapacityUnits: 1
        AttributeDefinitions:
          - AttributeName: id
            AttributeType: S
        KeySchema:
          - AttributeName: id
            KeyType: HASH
```

Deployment and Management



Best Practices

Best Practices

- Optimize Cold Start Performance
  - Minimize Binary Size
  - Use Environment Variables
- Test Locally
  - Using AWS SAM
  - Mocking Services
  - Unit & Integration Testing
- Secure Your API
- Implement Proper Monitoring & Alerts:
  - CloudWatch Metrics
  - Alerts
- Optimize Costs:
  - Use Provisioned Concurrency
  - Monitor Usage

Key Takeaways

Key Takeaways

- GoLang Performance
- Efficient Concurrency
- Small Binary Size
- Strong Ecosystem
- Best Practices

And remember always use the right tool for the job!

Github Repo

Thank you!

Please complete the session survey in the mobile app

Dr. Joanne Skiles

@drjoanneskiles

linkedin.com/in/jlskiles