GUTHEMBERG DA SILVA SAMPAIO

PROJECT HAUL

TECHNICAL REPORT

HAUL

Um app de logística que oferece encomendas vindas de e-commerce a passageiros de avião cadastrados e verificados, para transporte expresso até o destinatário em troca de uma quantia.

Arquitetura

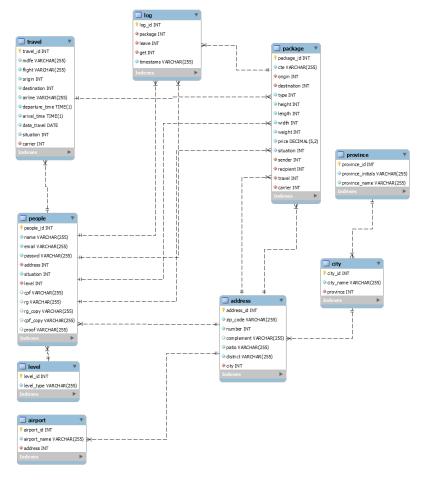
A arquitetura utilizada no seguinte projeto é a Serverless (sem servidor), devido a sua facilidade e rapidez no desenvolvimento e a alta escalabilidade da aplicação.

Sua facilidade é devido à necessidade "zero" de configuração do servidor, sendo necessário apenas o foco nos objetivos de negócios da aplicação.

1. Back-End (Em desenvolvimento)

a. DynamoDB

Inicialmente pensou-se em utilizar um banco de dados relacional para o armazenamento de dados da aplicação.



Devido a integração nativa do banco NoSQL DynamoDB com o AWS Lambda foi feita a mudança para agilizar o processo de desenvolvimento. Como o banco escolhido não é relacional foi feita uma adaptação, como por exemplo, retirar a normalização, colocando as colunas com mais de uma palavra em forma de árvore.

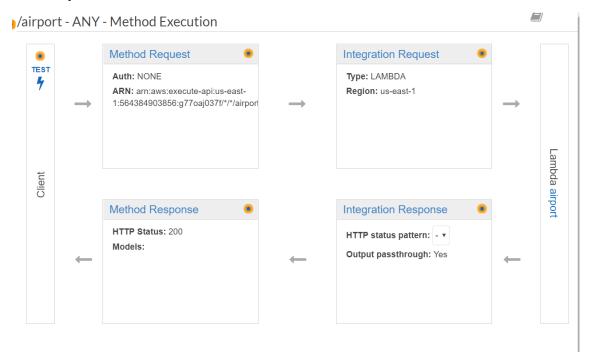
b. AWS Lambda

AWS Lambda é onde o código ou função é implementada, no caso de exemplo temos o código de um Endpoint completo que é utilizada na nossa aplicação.

```
T
                       x +
       index.js
  1
     console.log('Loading function');
  2
     const doc = require('dynamodb-doc');
  3
      const dynamo = new doc.DynamoDB();
     exports.handler = (event, context, callback) => {
          const done = (err, res) => callback(null, {
    statusCode: err ? '400' : '200',
  5
  6
  7
               body: err ? err.message : res,
  8
  9
                    'Content-Type': 'application/json',
 10
              },
           });
 11
 12
           switch (event.httpMethod) {
 13
               case 'DELETE'
                   dynamo.deleteItem(event.body, done);
 14
               break;
case 'GET':
 15
 16
 17
                   if(event.body.Key){
 18
                       dynamo.getItem(event.body, done);
 19
 20
                   else{
 21
                       dynamo.scan({ TableName: event.queryStringParameters.TableName }, done);
 22
               break;
case 'POST':
 23
 24
 25
                   dynamo.putItem(event.body, done);
 26
                   break;
               case 'PUT':
 27
 28
                   dynamo.updateItem(event.body, done);
 29
 30
               default:
 31
                   done(new Error(`Unsupported method "${event.httpMethod}"`));
 32
 33
```

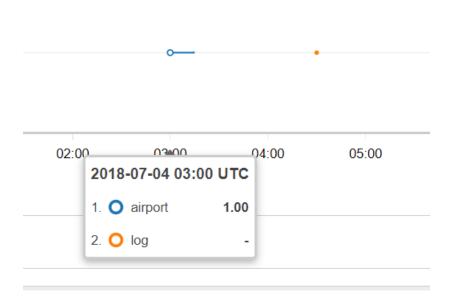
c. Amazon API Gateway

O Gateway da Amazon é responsável por interligar as requisições do front-end com a função lambda. A imagem mostra a tela de teste da API Gateway.



d. CloudWatch

O CloudWatch é responsável por salvar os logs das chamadas das funções Lambdas, por monitorar o uso de recursos da aplicação e por acionar eventos caso aconteça algum imprevisto.



e. IAM

Identity and Access Management é responsável pelo controle de acesso de toda aplicação, onde é definido as regras e as politicas de uso, por exemplo, a comunicação do API Gateway com a função Lambda, autorizar a comunicação externa com a API.

Aqui é onde termina o que é necessário para se ter uma aplicação Serverless na AWS.

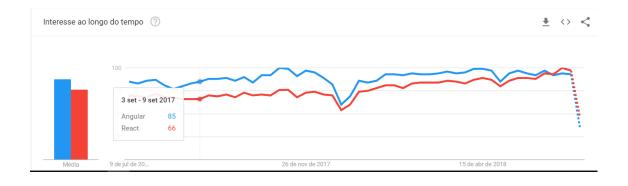


f. Amazon Cognito(Em desenvolvimento)

O Cognito é responsável pelo acesso do usuário, devido à necessidade de um front-end esta parte ainda está em desenvolvimento.

2. Front-End(Em planejamento)

ReactJS ou Angular 6?



Referências Bibliográficas

https://serverless-stack.com/

http://viniciusgarcia.me/development/o-que-eh-arquitetura-serverless/

https://aws.amazon.com/pt/documentation/

https://trends.google.com.br/trends/explore?q=Angular,%2Fm%2F012l1vxv