Nisarga Nagaraj
nisarganarmada@gmail.com

24th February 2021

Amazon Web Services Technical Design Document

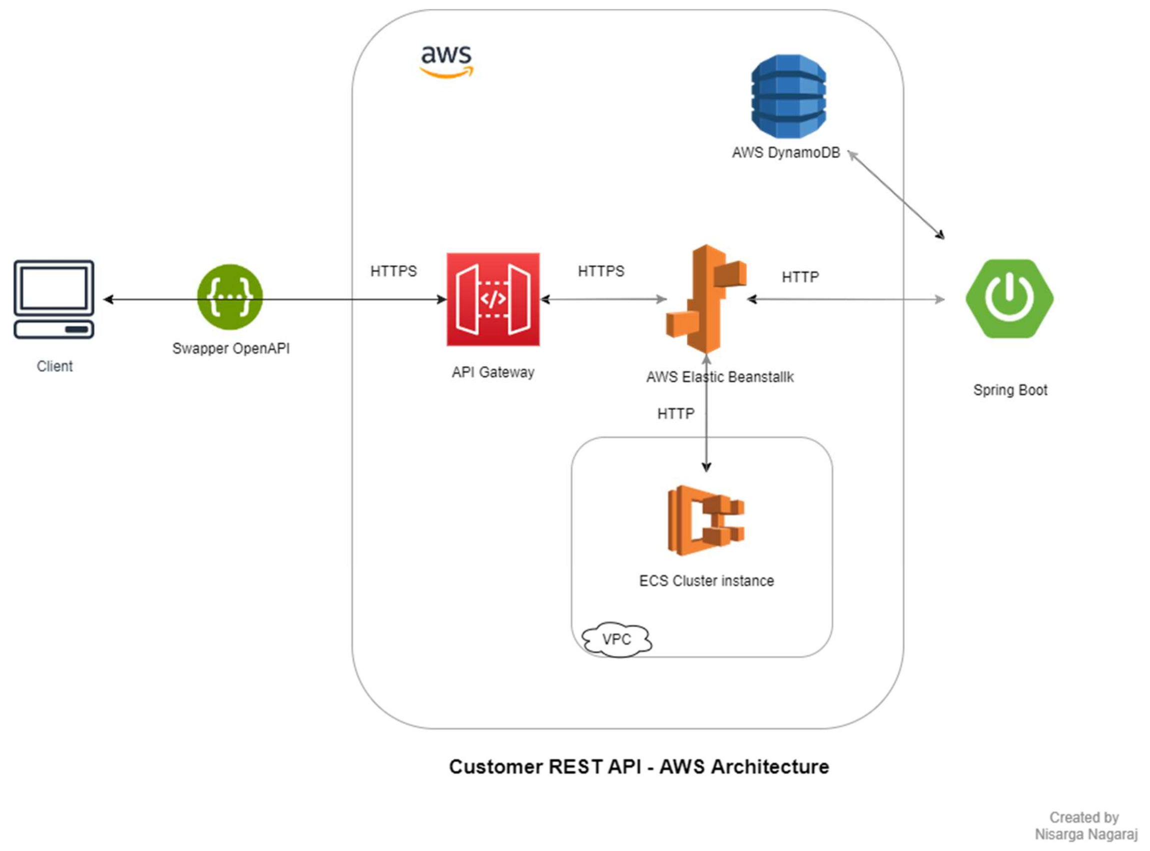
PROJECT OVERVIEW

Aim of this mini project is to implement a simple CRUD RESTful API on the Amazon Web Services platform and to create a supporting technical architecture.

GOALS

1. Creation of CRUD REST API using Open API Specification containing a single resource "Customers".
2. Development of Technical Architecture for the implementation of REST API on the Amazon Web Services (AWS)
3. Detailed explanation of the API Deployment on AWS Platform.

1 ARCHITECTURE DIAGRAM



Customer REST API - AWS Architecture

Note: As part of deliverable of this project, the Architecture diagram has been added in git repository in pdf format.

GitHub Repo - https://github.com/nisarga-nishu/AWS-CustomerAPI

Git Repo Folder name: Technical Design Document

2 TECHNICAL OVERVIEW

Customer REST API Specification

Language/format - JSON/YAML

IDE/Editor - Swagger editor

GitHub Repo - https://github.com/nisarga-nishu/AWS-CustomerAPI

Git Repo Folder name: API Specification

The REST API uses Open API Specification format. Swagger editor was used to design the Open API Spec.

The swagger documentation for the REST API can be found in the below link,

https://app.swaggerhub.com/apis-docs/nisarga-nishu/CusomerRestAPI/1.0.0

The REST API can be accessed at swagger hub in the below mentioned link. API is currently public since the private version is for a paid license.

https://app.swaggerhub.com/apis/nisarga-nishu/CusomerRestAPI/1.0.0

The API template is available in YAML format, API is downloaded as JSON Unresolved type and is uploaded into github.

As per the requirement, a single resource “**Customer**” is used to allow a user to perform the,

1. Creation of customer
2. Fetching the customers by ID.
3. Updating the customer details
4. Deleting a customer.
5. Listing all the customers.

Lastly, the attributes included in the “Customer” resource are:

1. customerId
2. FirstName
3. LastName
4. Address
    a. State
    b. City
    c. pincode

Customer REST API Back-end Development

Language/format - Java (Java 1.8)

Framework- Spring boot

Database - AWS DynamoDB

IDE/Editor - Eclipse 2020-12

GitHub Repo - https://github.com/nisarga-nishu/AWS-CustomerAPI

Git Repo Folder name: Infrastructure as Code

Back-end project is mainly a maven project using the spring suit tool - 4 framework with the Lombok and Spring Web dependencies for RESTful application.

The source code includes a main class, a DB config file, 2 POJO class files and a repository class.

An extra dependency package "aws-java-sdk-dynamodb" is included in the pom.xml file to support the DynamoDB integration.

In summary, the project contains 5 java classes overall, and an application.properties file.

Server port is made available for 5000, port 8080 was failing when tried, hence used 5000 port.

Customer REST API Amazon Web Services Implementation

Application Services- Elastic Beanstalk

Data Services- Amazon DynamoDB

Integration Services- API Gateway

Cloud infrastructure hosting services - Amazon EC2

Networking- VPC

GitHub Repo - https://github.com/nisarga-nishu/AWS-CustomerAPI

Git Repo Folder name: AWS-Links

The project implementation includes above stated AWS technologies. As mentioned earlier, amazon dynamoDB is used as a database and is integrated in the back-end code. Amazon dynamodb was the given first preference as a choice for databases due to its built-in security, NoSQL property and high scalability . The blogs and articles provided by amazon were referred to connect the back-end functionality with the Amazon DynamoDB.

Takeaway lesson during Dynamo implementation:

Tried to assess the DB using the function **AmazonDynamoDBClientBuilder.defaultClient()**,but access was denied since this function creates a service client with the default configuration, using the default provider chain to load credentials and the AWS Region. On further investigation, settled on to the below snippet, the **standard()** function includes the methods to provide endpoint configuration details.

AmazonDynamoDBClientBuilder.standard().withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(SERVICE_ENDPOINT, REGION)).withCredentials(new AWSStaticCredentialsProvider(new BasicAWSCredentials(ACCESS_KEY, SECRET_KEY))).build();

Amazon Elastic Beanstalk has been used to deploy the Customer REST API Web application since it is easy-to-use, and a good candidate for the web applications developed in Java.

The project build (Jar executable) was uploaded and deployed in an Elastic Beanstalk environment running on 64bit Amazon Linux/2.11.3 Java 8 platform.

HTTPS configuration could not be done in Elastic Beanstalk since it had to be configured via Load Balancer, which needs a validated certificate provided by AWS Certificate Manager provided I have my own domain name.

As a workaround, the http endpoint hosted on AWS Elastic Beanstalk was pointed to the Amazon API Gateway to publish the Restful application. Hence, the request received by the API endpoint is internally routed to the http integration of ELB endpoint of microservice hosted on an EC2 cluster. An EC2 instance is launched into the subnets of Virtual Private Cloud by default generating a virtual private address.

The health status of the EC2 environment keeps toggling between green, orange and red. Cloudwatch alarm has been set to monitor the cloudwatch metrics for the ECS instance on which Rest API is deployed.

Most importantly, customized IAM policies, roles and groups were created to get access to dynamoDB, Elastic Beanstalk and EC2 environments.

Please refer to the architecture diagram for a summarized overview.

Below are the http and https endpoints generated by Elastic Beanstalk and API Gateway.

3 ENDPOINT CONFIGURATION

HTTP Endpoint by Elastic Beanstalk:

http://customerapi.ap-southeast-2.elasticbeanstalk.com/

Endpoint for GET:

http://customerapi.ap-southeast-2.elasticbeanstalk.com/getCustomers

http://customerapi.ap-southeast-2.elasticbeanstalk.com/getCustomers/{customerId}

Endpoint for POST:

http://customerapi.ap-southeast-2.elasticbeanstalk.com/saveCustomer

Endpoint for PUT:

http://customerapi.ap-southeast-2.elasticbeanstalk.com/editCustomer

Endpoint for DELETE:

http://customerapi.ap-southeast-2.elasticbeanstalk.com/deleteCustomer

HTTPS Endpoint by AWS API Gateway:

https://s3b1om8tb9.execute-api.ap-southeast-2.amazonaws.com/Test

Endpoint for GET, POST, PUT and DELETE

https://s3b1om8tb9.execute-api.ap-southeast-2.amazonaws.com/Test/customer

Endpoint for GET element by ID

https://s3b1om8tb9.execute-api.ap-southeast-2.amazonaws.com/Test/customer/{customerId}

4 TEST DATA

Json request for POST

```
{

  "address": {

    "city": "Melbourne",

    "state": "Victoria",
    "pinCode": 3006
  },
  "lastName": "Nagaraj",
  "firstname": "Nisarga"
}
```

Link with customerId to perform GET operation

https://s3b1om8tb9.execute-api.ap-southeast-2.amazonaws.com/Test/customer/9ef5f3dc-83d0-4c3e-9b21-32e0b39c59bd

4 EXPECTED IMPROVEMENTS

1. Initially tried using AWS Lambda function in the Java project, and tried to directly upload the build to AWS Lambda, but access failed due to AWS account setting issues in Eclipse. Need to focus on integrating the rest API using lambda functions.
2. Need to get customized user friendly endpoints.
3. The health of the EC2 environment to be addressed and improved. The status of health keeps toggling between orange and red.

5 REFERENCES

1. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Scan.html
2. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Java.html
3. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-cloudwatch-createalarm.html
4. https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_create.html