Pokemon Backend Setup

This document will cover the backend setup for our Pokedex Service, This includes having Lambdas that handles requests to retrieve Pokemon information from DynamoDB, and having an API gateway that triggers our Lambdas.

Step 1: Install Required Tools

Step A: Install Java Development Kit (JDK) 11

1. Visit the Oracle JDK download page or AdoptOpenJDK to download the JDK 11 installer compatible with your operating system.
2. Follow the installation instructions provided by the installer for your specific operating system.
3. After installation, set the `JAVA_HOME` environment variable to point to the directory where JDK 11 is installed. This step is crucial for Maven and other tools to locate the Java runtime environment.

Step B: Install Apache Maven

1. Visit the Apache Maven download page and download the latest stable version of Apache Maven.
2. Extract the downloaded Maven archive to a directory on your system.
3. Add the `bin` directory of the extracted Maven folder to your system's `PATH` environment variable.
4. Verify the Maven installation by running `mvn --version` command in your terminal or command prompt. You should see the Maven version and other details if the installation is successful.

Step C: Install AWS CLI

1. Install AWS CLI using your system's package manager (e.g., `apt`, `yum`, `brew`) or by downloading the installer from the AWS CLI official website.
2. ~~After installation, configure AWS CLI by running `aws configure` command and providing your AWS access key, secret key, default region, and output format when prompted.~~
3. Verify the AWS CLI installation by running `aws --version` command in your terminal or command prompt. You should see the AWS CLI version if the installation is successful.

Step D: Install AWS SAM CLI

1. Install AWS SAM CLI using your system's package manager (e.g., `brew`, `pip`) or by following the installation instructions provided on the AWS SAM CLI GitHub repository.
2. After installation, verify the AWS SAM CLI installation by running `sam --version` command in your terminal or command prompt. You should see the SAM CLI version if the installation is successful.

Step E: Verify Installation

1. To ensure everything is set up correctly, open a new terminal or command prompt window.
2. Run the following commands to verify the installations:

```
java --version #Should display the installed Java version.
mvn --version #Should display the installed Maven version.
aws --version #Should display the installed AWS CLI version.
sam --version #Should display the installed AWS SAM CLI version.
```

If all the above commands display the respective versions without any errors, it indicates that the tools are successfully installed and configured on your system, and you're ready to proceed with creating your Java 11 Lambda function using Maven and AWS SAM with Amazon API Gateway.

Step 2: Create a New Maven Project

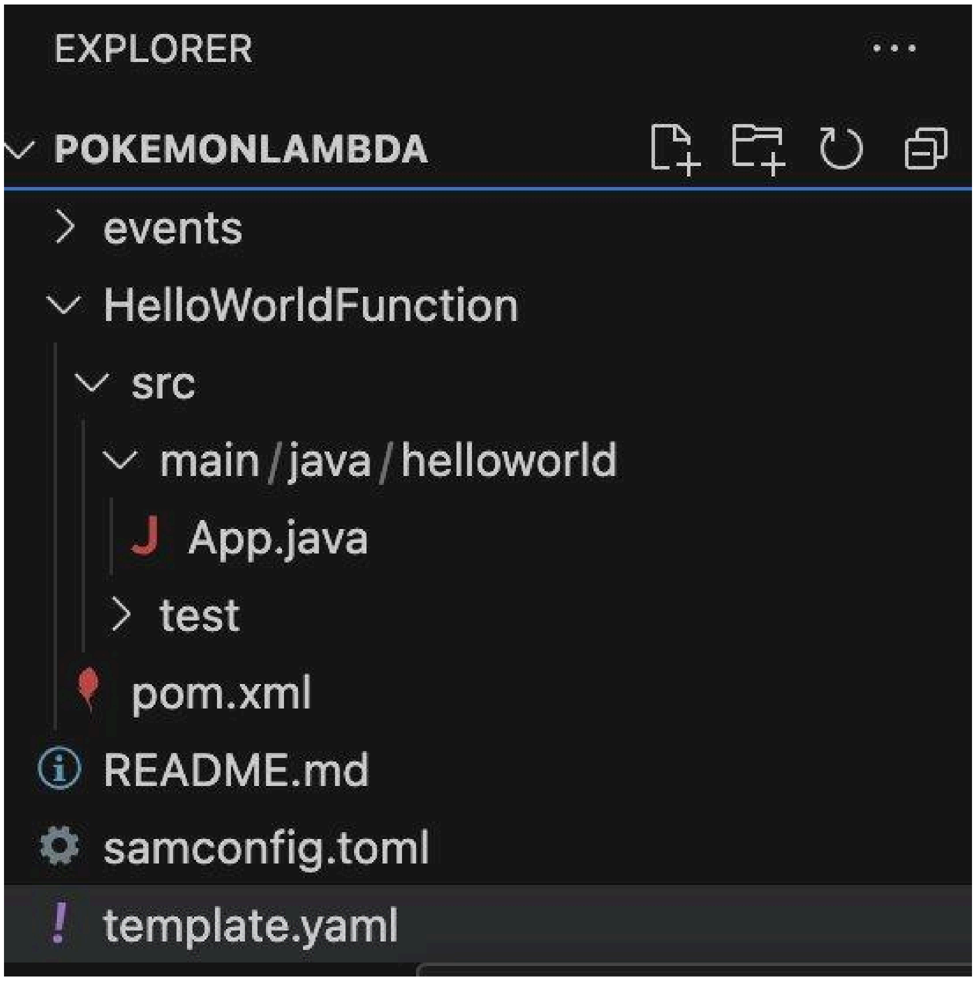
1. Open your terminal or command prompt.
2. Navigate to the directory where you want to create your project.
3. Run the following Sam command to create a new Maven project:

```
sam init -n pokemonLambda -r java11
--app-template hello-world && cd pokemonLambda
```

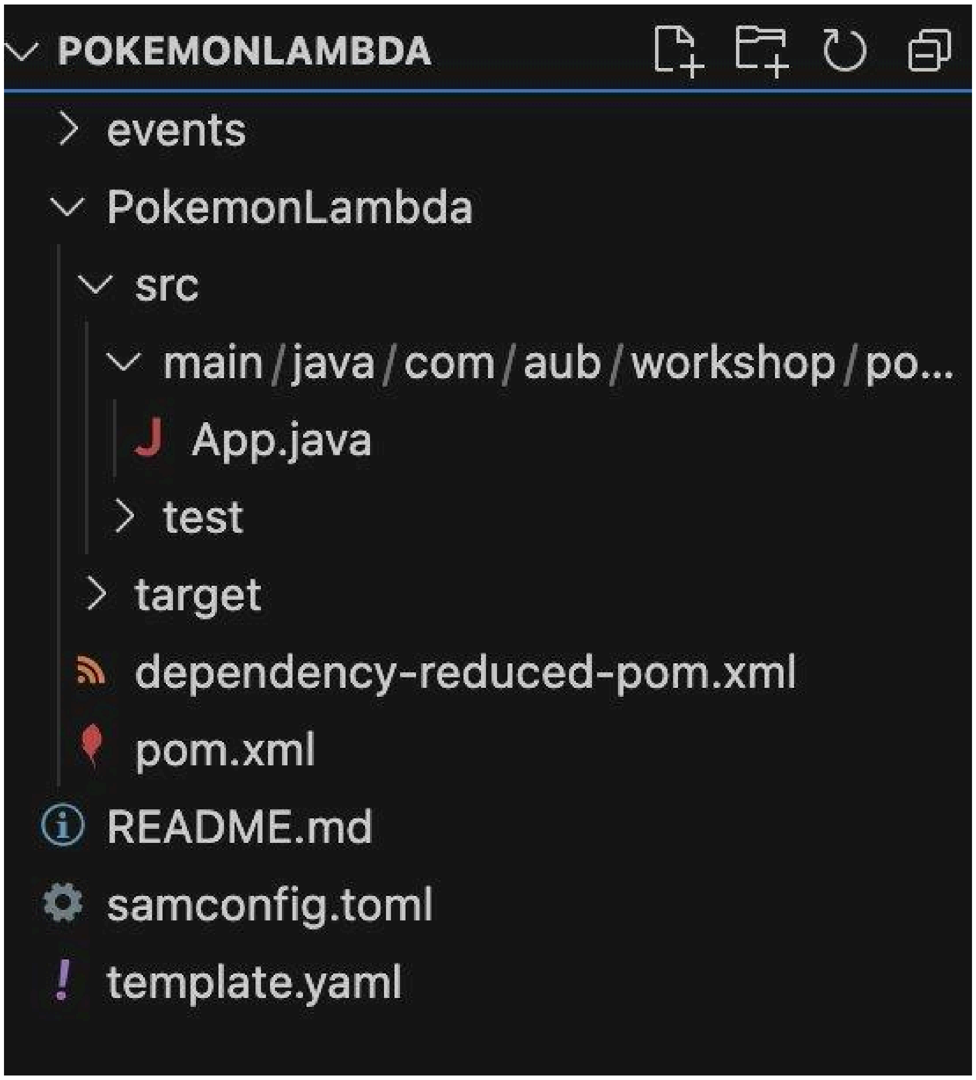
This command will create a basic Lambda function using Java 11.
you will receive an interactive guide for the generation follow the below answers : 2, N, N , Y

```
Based on your selections, the only Template available is Hello World Example.
We will proceed to selecting the Template as Hello World Example.

Based on your selections, the only Package type available is Zip.
We will proceed to selecting the Package type as Zip.

Which dependency manager would you like to use?
   1 - gradle
   2 - maven
>> Dependency manager: 2

>> Would you like to enable X-Ray tracing on the function(s) in your
application?  [y/N]: N

>> Would you like to enable monitoring using CloudWatch Application Insights?
For more info, please view
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch-applic
ation-insights.html [y/N]: N

>> Would you like to set Structured Logging in JSON format on your Lambda
functions?  [y/N]: y
Structured Logging in JSON format might incur an additional cost. View
https://docs.aws.amazon.com/lambda/latest/dg/monitoring-cloudwatchlogs.html#moni
toring-cloudwatchlogs-pricing for more details
```

your app will look like the following



Rename HelloWorldFunction to be PokemonLambda
and rename the folder helloworld to be com/aub/workshop/pokemon
do the same in the tests folder
your project should now look like



```
POKEMONLAMBDA
  > events
  v PokemonLambda
    v src
      v main / java / com / aub / workshop / po...
          App.java
      > test
    > target
      dependency-reduced-pom.xml
      pom.xml
  README.md
  samconfig.toml
  template.yaml
```

Run the following to step inside PokemonLambda and install dependencies:

```
cd PokemonLambda && mvn install
```

it should build the project successfully.

If you have opened the project through VSCode and the Java project is not working correctly, try right clicking on the main/java folder and pressing **Add Folder to Java Source Path**.

Step 3: Configure AWS Credentials

From the AWS workshop studio, you should see "Get AWS CLI credentials" button. Press that to get credentials for your AWS CLI. It will provide a command that you can paste in your terminal to configure your session. **If you open a new terminal window, you should repeat this.**

Step 4: Implement Your Lambda Function

Commons

1. create a file called Constants.java and add your static values and config like the name of the DynamoDB Table and a constant for the page size of the List API (more on that later)

```
package com.aub.workshop.pokemon;

public class Constants {
    public static final String DYNAMODB_TABLE_NAME = "Pokemons"; //
Name of your DynamoDB table
    public static final int PAGE_SIZE = 50; // Threshold for
pagination
}
```

List API

1. Navigate to the `com/aub/workshop/pokemon` directory.
2. Rename the `App.java` file to `PokemonListHandlerLambda.java`.
3. Add your Lambda function logic inside the `handleRequest` method.

At first, for *experimentation*, I recommend removing all methods except `handleRequest`, deploying, running, then adding more and more logic. Feel free to just copy the whole class directly.

```
package com.aub.workshop.pokemon;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.Base64;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.dynamodbv2.model.ScanRequest;
import com.amazonaws.services.dynamodbv2.model.ScanResult;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
```

```java
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import static com.aub.workshop.pokemon.Constants.DYNAMODB_TABLE_NAME;
import static com.aub.workshop.pokemon.Constants.PAGE_SIZE;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;


public class PokemonListHandlerLambda implements RequestHandler<APIGatewayProxyRequestEvent,
APIGatewayProxyResponseEvent>{
   private static final String QUERY_PARAM = "nextPage";

   private final AmazonDynamoDB client = AmazonDynamoDBClientBuilder.standard().build();
   private final DynamoDB dynamoDB = new DynamoDB(client);
   private final Gson gson = new Gson();
   private LambdaLogger logger;

   @Override
   public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent input, Context
context) {
       // Lambda Logger
       logger = context.getLogger();
       logger.log("Starting Index lambda");

       String nextPage = input.getQueryStringParameters() != null ?
               input.getQueryStringParameters().get(QUERY_PARAM) : null;
       logger.log("Got Query param next page: " + nextPage);
       Map<String, AttributeValue> lastKeyEvaluated =
decodeLastEvaluatedKeyFromParameters(nextPage);


       String jsonOutput = scanDBAndFormatResponse(lastKeyEvaluated);

       APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent();
       response.setStatusCode(200);
       response.setBody(jsonOutput);
       return response;
   }
```

```java
    private Map<String, AttributeValue> decodeLastEvaluatedKeyFromParameters(String nextPage) {
        if (nextPage!=null){
            byte[] decodedBytes = Base64.getUrlDecoder().decode(nextPage);
            String decodedString = new String(decodedBytes);

            Type type = new TypeToken<Map<String, AttributeValue>>(){}.getType();
            return gson.fromJson(decodedString, type);
        }
        return null;
    }

    private String encodeLastEvaluatedKey(Map<String, AttributeValue> lastKey) {
        // turn tree to JSON
        String marshalledLastKey = gson.toJson(lastKey);

        // Base64 URL safe encoding
        return Base64.getUrlEncoder().encodeToString(marshalledLastKey.getBytes());
    }

    private  String scanDBAndFormatResponse(Map<String, AttributeValue> lastKeyEvaluated) {
         // setup Query and Scan DB
         dynamoDB.getTable(DYNAMODB_TABLE_NAME);
         ScanRequest scanSpec = new ScanRequest()
                 .withTableName(DYNAMODB_TABLE_NAME)
                 .withLimit(PAGE_SIZE)
                 .withExclusiveStartKey(lastKeyEvaluated);
        ScanResult result = client.scan(scanSpec);



        List<Map<String, Object>> items = new ArrayList<>();
        result.getItems().forEach(item -> {
            Map<String, Object> parsedItem = new HashMap<>();
                       item.forEach((key, value) ->
                    {
                        if (value.getS() != null) {
                            parsedItem.put(key, value.getS());
// Assuming most attributes are strings
```

```
                        } else {
                            parsedItem.put(key, value.getN());
// Assuming rest of attributes are Number
                        }
                    }
            );

            items.add(parsedItem);
        });

        // Add Last Key + returned Pokemons to response Map
        Map<String, Object> responseMap = new HashMap<>();
        responseMap.put("items", items);

        // get Last Evaluated Key in Page and Encode it to be returned in Response
        lastKeyEvaluated = result.getLastEvaluatedKey();
        if (lastKeyEvaluated != null && !lastKeyEvaluated.isEmpty()){
            String encodedKey = encodeLastEvaluatedKey(lastKeyEvaluated);
            logger.log("key: " + lastKeyEvaluated + " encoded: " + encodedKey);
            responseMap.put("nextPage", encodedKey);
        }

        // turn response to JSON and return
        return gson.toJson(responseMap);
    }
}
```

4. update the unit test file name to PokemonListHandlerLambdaTest.java and for now remove the `successfulResponse` unit test and the `@Test` annotation.

GET API

With Get API we'll be able to request one single pokemon from the Lambda.

1. create a file called PokemonGetHandlerLambda.java
2. implement the Lambda function your file should look like below.

To make it work, we'll need to add some dependencies.

```
package com.aub.workshop.pokemon;

import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.document.DynamoDB;
import com.amazonaws.services.dynamodbv2.document.Item;
import com.amazonaws.services.dynamodbv2.document.Table;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyRequestEvent;
import com.amazonaws.services.lambda.runtime.events.APIGatewayProxyResponseEvent;
import static com.aub.workshop.pokemon.Constants.DYNAMODB_TABLE_NAME;
import com.google.gson.Gson;


public class PokemonGetHandlerLambda implements
RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>{
    private final AmazonDynamoDB client =
AmazonDynamoDBClientBuilder.standard().build();
    private final DynamoDB dynamoDB = new DynamoDB(client);

    @Override
    public APIGatewayProxyResponseEvent handleRequest(APIGatewayProxyRequestEvent
input, Context context) {
        LambdaLogger logger = context.getLogger();

        String pokemonId = input.getPathParameters().get("id");
        logger.log("Received call to get pokemon with id: " + pokemonId);

        int pokemonNumber = Integer.parseInt(pokemonId);

        Table table = dynamoDB.getTable(DYNAMODB_TABLE_NAME);

        // Retrieve item from DynamoDB using pokemonId as the key
        Item item = table.getItem("pokemonNumber", pokemonNumber);

        // Convert item to JSON string
        Gson gson = new Gson();
        String jsonOutput = gson.toJson(item.asMap());

        APIGatewayProxyResponseEvent response = new APIGatewayProxyResponseEvent();
        response.setStatusCode(200);
        response.setBody(jsonOutput);
        return response;
    }
}
```

At this step, maven will not work yet. You should add the dependencies to make it work.

Step 5: Install AWS SDK Dependencies for Lambda and DynamoDB

Navigate to the root directory of your Maven project.
Open the `pom.xml` file in a text editor.
Replace the pom file with the following file instead

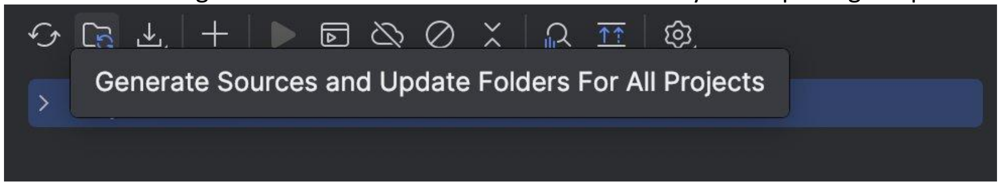
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.aub.workshop.pokemon</groupId>
    <artifactId>pokemonLambda</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.shade.plugin.version>3.2.1</maven.shade.plugin.version>
        <maven.compiler.plugin.version>3.6.1</maven.compiler.plugin.version>
        <exec-maven-plugin.version>1.6.0</exec-maven-plugin.version>
        <aws.java.sdk.version>2.25.31</aws.java.sdk.version>
        <aws.lambda.java.version>1.2.3</aws.lambda.java.version>
```

```xml
        <junit5.version>5.10.1</junit5.version>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-bom</artifactId>
                <version>1.12.630</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.2</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-events</artifactId>
            <version>3.11.1</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-s3</artifactId>
            <version>1.12.630</version>
        </dependency>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-dynamodb</artifactId>
            <version>1.11.563</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.8</version>
        </dependency>

        <!-- Test Dependencies -->

        <dependency>
          <groupId>junit</groupId>
          <artifactId>junit</artifactId>
          <version>4.13.2</version>
          <scope>test</scope>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.13.0</version>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.4</version>
                <configuration>
                    <createDependencyReducedPom>false</createDependencyReducedPom>
                </configuration>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers>
                                <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>com.aub.workshop.pokemon.PokemonListHandlerLambda</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

now run in the terminal

```
mvn install
```

and then click on the generate sources button in intelliJ for it to sync the package dependencies



Step 6: Build Your Project

1. Navigate to the root directory of your Maven project.
2. Run the following Maven command to build your project, make sure you are in `pokemonLambda/PokemonLambda` folder.

```
mvn clean package shade:shade
```

Step 7: Update SAM Template with Lambda Function, API Gateway, and Policies Configuration

1. Open the `template.yaml` file (or your SAM template file) in a text editor and update the handler name, CodeUri and memorySize

```
Handler: com.aub.workshop.pokemon.PokemonListHandlerLambda::handleRequest
Runtime: java11
CodeUri: .
MemorySize: 1024
```

1. Our Lambda function requires additional permissions to read and scan DynamoDB, update the `Policies` section accordingly. For example:

```
      Policies:
        - DynamoDBReadPolicy:
            TableName: "Pokemons"
        - DynamoDBCrudPolicy:
            TableName: "Pokemons" # Add this to allow Scan operation
```

This will ensure that the IAM policies specified allow the necessary permissions for your Lambda function to execute successfully.

1. Add a new resource for Amazon API Gateway under the `Resources` section. For example:

```
PokemonApi:
 Type: AWS::Serverless::Api
 Properties:
    StageName: Prod
    DefinitionBody:
    swagger: "2.0"
    info:
    title: "Pokemon API"
    version: "1.0"
    paths:
    /pokemon:
    get:
    responses:
    '200':
    description: "200 response"
    x-amazon-apigateway-integration:
    uri:
    Fn::Sub:
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${Pokemon
Function.Arn}/invocations"
    httpMethod: POST
    type: aws_proxy
```

add Event to our lambda function so that we can make the API call the lambda trigger

```
Events:
    PokemonApi:
        Type: Api
        Properties:
        Path: /pokemon
        Method: GET
```

6. Save the changes to the SAM template file.

```
# final Sam file should look like AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'

Resources:
  PokemonFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler:
com.aub.workshop.pokemon.PokemonListHandlerLambda::handleRequest
      Runtime: java11
      CodeUri: PokemonLambda
      MemorySize: 1024
      Timeout: 60
      Policies:
        - DynamoDBReadPolicy:
            TableName: "Pokemons"
        - DynamoDBCrudPolicy:
            TableName: "Pokemons"  # Add this to allow Scan operation
      Events:
        PokemonApi:
          Type: Api
          Properties:
            Path: /pokemon
            Method: GET
          RestApiId: !Ref PokemonApi
          Auth:
            ApiKeyRequired: false
          Cors: # Enable CORS
              AllowMethods: "'GET,OPTIONS'"
              AllowHeaders:
"'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'"
              AllowOrigin: "'*'"
  PokemonGetFunction:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: PokemonLambda
      Handler:
com.aub.workshop.pokemon.PokemonGetHandlerLambda::handleRequest
      Runtime: java11
      MemorySize: 1024
      Timeout: 60
      Policies:
        - DynamoDBReadPolicy:
            TableName: "Pokemons"
        - DynamoDBCrudPolicy:
            TableName: "Pokemons"  # Add this to allow Scan operation
      Events:
        PokemonApi:
          Type: Api
          Properties:
            Path: /pokemon/{id}  # Updated path to include {id} for receiving
Pokemon ID
            Method: GET
          RestApiId: !Ref PokemonApi
          Cors: # Enable CORS
            AllowMethods: "'GET,OPTIONS'"
            AllowHeaders:
"'Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token'"
            AllowOrigin: "'*'"
  PokemonApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: Prod
      DefinitionBody:
        swagger: "2.0"
        info:
```

```yaml
          title: "Pokemon API"
          version: "1.0"
        paths:
          /pokemon:
            get:
              parameters:
                - name: nextPage
                  in: query
                  required: false
                  type: string
                  default: ""  # Default value for the limit
              responses:
                '200':
                  description: "200 response"
              x-amazon-apigateway-integration:
                uri:
                  Fn::Sub:
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${Pokemon
Function.Arn}/invocations"
                httpMethod: GET
                type: aws_proxy
          /pokemon/{id}:
            get:
              responses:
                '200':
                  description: "200 response"
              x-amazon-apigateway-integration:
                uri:
                  Fn::Sub:
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${Pokemon
Function.Arn}/invocations"
                httpMethod: GET
                type: aws_proxy

Outputs:
  PokemonApi:
    Description: "API Gateway endpoint URL for Pokemon Function"
    Value: !Sub
"https://${ServerlessRestApi}.execute-api.${AWS::Region}.amazonaws.com/Prod/p
```

```
okemon"
    Export:
      Name: "PokemonApiUrl"
```

Step 8: Deploy Your Server-less Application

1. Navigate to the root directory of your Maven project.
2. Run the following commands to deploy your serverless application using AWS SAM:

```
sam build
sam deploy --stack-name PokemonLambdaStack --guided
```

Make sure AWS CLI credentials are configured for your current session. You can run `echo $AWS_ACCESS_KEY_ID` to see if it's configured or not.

Follow the prompts to configure your deployment settings such as **AWS Region**, stack name, etc. as below

```
Configuring SAM deploy
======================

    Looking for config file [samconfig.toml] :  Not found

    Setting default arguments for 'sam deploy'
    =========================================
    Stack Name [PokemonLambdaStack]: <Enter>
    AWS Region [eu-north-1]: eu-west-2
    #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
    Confirm changes before deploy [y/N]: <y>
    #SAM needs permission to be able to create roles to connect to the resources in
your template
    Allow SAM CLI IAM role creation [Y/n]: <y>
    #Preserves the state of previously provisioned resources when an operation fails
    Disable rollback [y/N]: <N>
    PokemonFunction has no authentication. Is this okay? [y/N]: <y>
    Save arguments to configuration file [Y/n]: <y>
    SAM configuration file [samconfig.toml]: <Enter>
    SAM configuration environment [default]: <Enter>
```

you should get the following message after

```
    Looking for resources needed for deployment:
    Creating the required resources...
    Successfully created!

    Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-4q2cihcrskiz
    A different default S3 bucket can be set in samconfig.toml and auto resolution
of buckets turned off by setting resolve_s3=False

    Saved arguments to config file
    Running 'sam deploy' for future deployments will use the parameters saved above.
    The above parameters can be changed by modifying samconfig.toml
    Learn more about samconfig.toml syntax at

https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serve
rless-sam-cli-config.html

    Uploading to PokemonLambdaStack/9854ee7b2a91aa876aebb5b2dc71f88d  17927803 /
17927803  (100.00%)

    Deploying with following values
    ===============================
    Stack name                   : PokemonLambdaStack
    Region                       : eu-north-1
    Confirm changeset            : True
    Disable rollback             : False
    Deployment s3 bucket         :
aws-sam-cli-managed-default-samclisourcebucket-4q2cihcrskiz
    Capabilities                 : ["CAPABILITY_IAM"]
    Parameter overrides          : {}
    Signing Profiles             : {}

Initiating deployment
=====================

    Uploading to PokemonLambdaStack/225f389e9be56af53145c53fe0878c68.template  1625
/ 1625  (100.00%)


Waiting for changeset to be created..
```

```
CloudFormation stack changeset
-------------------------------------------------------------------------------------
-------------
Operation                LogicalResourceId        ResourceType
Replacement
-------------------------------------------------------------------------------------
-------------
+ Add                    PokemonApiDeployment16   AWS::ApiGateway::Deplo   N/A
                         947a3939                 yment
+ Add                    PokemonApiProdStage      AWS::ApiGateway::Stage   N/A
+ Add                    PokemonApi               AWS::ApiGateway::RestA   N/A
                                                  pi
+ Add                    PokemonFunctionPokemon   AWS::Lambda::Permissio   N/A
                         ApiPermissionProd        n
+ Add                    PokemonFunctionRole      AWS::IAM::Role           N/A
+ Add                    PokemonFunction          AWS::Lambda::Function    N/A
+ Add                    ServerlessRestApiDeplo   AWS::ApiGateway::Deplo   N/A
                         yment0317c2dc9e          yment
+ Add                    ServerlessRestApiProdS   AWS::ApiGateway::Stage   N/A
                         tage
+ Add                    ServerlessRestApi        AWS::ApiGateway::RestA   N/A
                                                  pi
-------------------------------------------------------------------------------------
-------------


Changeset created successfully.
arn:aws:cloudformation:eu-north-1:665065268178:changeSet/samcli-deploy1714400791/71b
d6ef6-392d-4437-bc3b-cee5288677fc


Previewing CloudFormation changeset before deployment
======================================================
Deploy this changeset? [y/N]: y
```

and then it will start deploying the stack and show you the status of resource deployment and end on the following message

```
Outputs
```

```
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
Key                 PokemonApi
Description         API Gateway endpoint URL for Pokemon Function
Value
https://tm69a9tvnl.execute-api.eu-north-1.amazonaws.com/Prod/pokemon
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Successfully created/updated stack - PokemonLambdaStack in eu-north-1
```

If you make future changes, you can just run `sam deploy --stack-name PokemonLambdaStack` without `--guided`, it will use the previous deployment configurations.

Step 9: Test Your Lambda Function and APIs

After successful deployment, you can see that you have created 2 Lambda functions. While you can test them directly, I recommend trying to call the APIs.

After your SAM deployment finishes, you'll get an API Gateway endpoint URL in the outputs. Copy that.
It will have a format like this:
`https://xxx.execute-api.eu-west-2.amazonaws.com/Prod/pokemon`
This is your List endpoint. If you open it, it should return you the first 50 pokemons. To open the next page, you should add to the url `?nextPage=<nextPageTokenFromResponse>`

If you want to access a specific Pokemon, you can use the GET API with `/Prod/pokemon/<pokemonID>`. Example:
`https://xxx.execute-api.eu-west-2.amazonaws.com/Prod/pokemon/1`

Troubleshooting

The API calls fail. What should you do?

The API calls reach to API Gateway, from where they go to the AWS Lambda. If you have following the SAM template correctly, most likely the issue is with the lambdas.

Open the AWS Lambda page in the AWS Console, from where find the `PokemonLambdaStack-PokemonFunction-` lambda is responsible for the List API, while the `PokemonLambdaStack-PokemonGetFunction-` is responsible for the Get API. Based on which one was failing, open that one.

Then go to Monitoring tab, `View CloudWatch Logs`. Open the most recent log stream, you should see the exceptions there.

Bonus topics to learn about:

- How to remove unneeded Attributes in the scan request
- What is CORS, and how did we configure it here? Why do we need it? (hint: Check your session for tomorrow)
- How to remove unneeded Attributes in the scan request ?
- Your data ingestion lambda accessed an s3 bucket just by using its name. That's because s3 bucket names are globally unique. Explore this topic. What can you use this for?
- What is CloudFormation, did we use it today?

References [fill rest of refs]

https://docs.aws.amazon.com/lambda/latest/dg/java-package.html