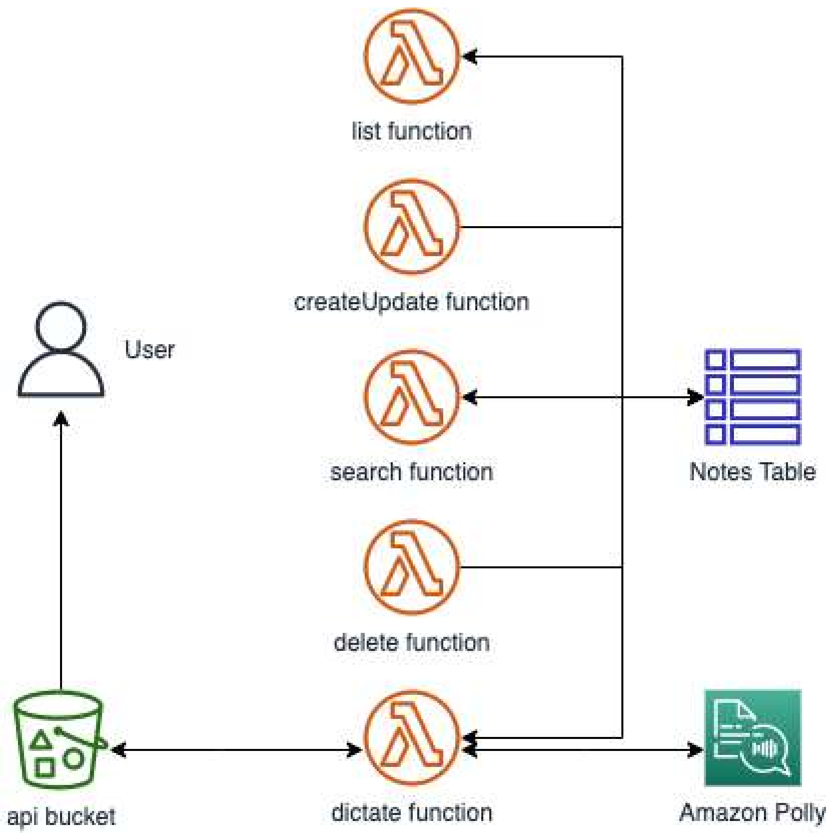
Lab 4 (Java) - Develop Solutions Using AWS Lambda

Lab Overview

Now that you have notes stored in an Amazon DynamoDB table, you can create the application logic to interact with the files. In prior labs, you have used the AWS SDK to interact with AWS services. Now you will run the code you have been developing.

With AWS, you can use the Amazon Elastic Compute Cloud (Amazon EC2) to run your code when you need full control over the operating system and instance size. If you don't need that level of control, you can run your code in a prebuilt run-time environment that supports your language. If you need to control the orchestration of those containers, you cam use Amazon Elastic Container Service (Amazon ECS) or Amazon Elastic Kubernetes Service (Amazon EKS). But, if you just want to run your code with minimum configuration, you will use AWS Lambda.

In this lab, you will create the backend logic for your application by deploying your code with AWS Lambda functions.



Objectives

After completing this lab, you will be able to:

- Create AWS Lambda functions and interact programmatically using AWS SDKs and the AWS CLI.
- Configure Lambda functions to use the environment variables and integrate with other services.
- Generate Amazon S3 pre-signed URLs using AWS SDKs and verify the access to bucket objects.
- Deploy the Lambda functions with .zip file archives through your IDE and test as needed.
- Invoke AWS Lambda functions using the AWS Console and AWS CLI.

Duration

This lab requires approximately **60 minutes** to complete.

Task 0: Connect to your development environment

Task 0: Connect to your development environment

In this task, you will get connected to your development environment.

1. Choose a method to connect to the development environment from the options below:

- Connect to Your Windows Dev Instance using Apache Guacamole
- Connect to Your Windows Dev Instance from a MacOS Machine using RDP
- Connect to Your Windows Dev Instance from a Windows Machine using RDP

NOTE:: When lab instructions in subsequent sections require you to issue commands, you will use the **Intellij Terminal**.

Consider: This lab is designed for both experienced and newer developers:

- For more experienced developers who enjoy a challenge, there are **High-Level Instructions** before each task that should provide you enough information to help you complete the task.
- Once you complete the updates test your code to ensure it works, troubleshoot if needed, and then move on to the next task.
- For newer developers, there are **Detailed Instructions** to guide you through each step of the lab.

Task 0.1: Open and configure the IntelliJ IDE

Prepare the development environment for code development.

2. Open **IntelliJ** from the desktop icon.

- From the **LICENSE AGREEMENT** window, select the check box reading ☑ I confirm that I have read and accept the terms of this User Agreement and then choose **Continue**
- From the DATA SHARING window, choose **Don't Send**

3. From the **Welcome** screen, choose **Open** > navigate to C:\code\dictateFunction > select pom.xml > OK > Open as Project > ☑ Trust projects in C:/code > Trust Project

⚠ **CAUTION:** Wait for the underlying processes to complete as they download plugins, dependencies and other tasks. If you do not wait for those tasks to complete, this can cause errors. You can observe the status at the bottom of the screen.

4. Close the **Tip of the Day** window.
5. Set the **AWS Connections Setting** to use the default profile and set the region to the value to the left of these instructions.

NOTE:: You can set this from the option at the bottom-right of the screen (if you open the **AWS Explorer** tab from the lower-left of the screen). Also set the region to match the region value to the left of these instructions.

6. You will now assign the SDK to the project. Choose **File > Project Structure... > Project SDK:** 11 Amazon Corretto ve > OK

Task 1: Create an AWS Lambda function

Task 1: Create an AWS Lambda function

In the first function you create an on-demand audio version of your note. This function will use Amazon DynamoDB, Amazon Polly, and Amazon S3.

You will start the lab work by using the AWS Console where you can see a representation of the options that are available. However, most of the work in this lab will be performed in the AWS CLI and your code.

You are given with the incomplete source code. You will use the clues provided to complete the missing pieces of the code at **TODO** placeholders. Once the syntax-free code is filled out, you will run the code and review the results.

High-Level Instructions

- Use the AWS Console to create the dictateFunction
- Set the runtime to **Java 11 (Corretto)**.
- Use **lambdaPollyRole** to grant the Lambda function permissions

Detailed Instructions

1. From the **AWS Management Console**, use the **AWS search bar** to search for `Lambda` and then choose the service from the list of results.
2. Choose **Create function**
3. Choose **Author from scratch**
4. For **Function name** enter `dictateF`
5. For **Runtime**, select **Java 11 (Corretto)**.
6. Expand ▶ **Change default execution role** and choose **Use an existing role**.
7. For **Existing role**, select **lambdaPollyRole**.
8. Choose **Create function**

The AWS Lambda Console allows you to create, configure, monitor, and invoke Lambda functions. If your function code is written in a language that does not require compilation (Node.js, Python, or Ruby), and is less than 3MB, you can also view and modify the function code in the console. The **lambdaPollyRole** provides permissions for the function to access DynamoDB, Amazon S3, Amazon Polly and the Logs services.

Congratulations! You have created the Lambda function named **dictate-function**.

Task 2: Add processing logic to the AWS Lambda function code

Task 2: Add processing logic to the AWS Lambda function code

The Lambda function handler is the method in your function code that processes events. When your function is invoked, Lambda runs the handler method. When the handler exits or returns a response, it becomes available to handle another event.

A class named Handler defines a handler method named handleRequest. The handler method takes an event and context object as input and returns a string.

In your development environment using InetelliJ, you have been provided with a skeleton function that has all dependencies imported, clients and resources created, and user parameters created. This function will return a signed URL that grants access to an MP3 file hosted in an Amazon S3 bucket. The MP3 file is an audio version of a note stored in your database.

This function completes **3 main steps** listed below:

- Retrieve the text of a note from Amazon DynamoDB.
- Convert that text to speech with Amazon Polly and save that audio stream as a local file.
- Upload that file to Amazon S3 and generate a pre-signed URL to return from the function.

High-Level Instructions

- Assign values to use these two environment variables:
    - **MP3_BUCKET_NAME** (use bucket name from your lab environment with **apibucket** in its name)
    - **TABLE_NAME** (Notes)

- From your development environment, complete Handler.java **in java[dictate] > src > main > java > dictate** folder using the parameters passed to each function:
    - Add code to **TODO 1** that returns the note text from Amazon DynamoDB.
    - Add code to **TODO 2** that creates a local MP3 file in the /tmp folder with an audio version of that note using Amazon Polly and returns the file location.
    - Add code to **TODO 3** that uploads the MP3 file to Amazon S3, removes the local file, and returns a pre-signed URL to that object.

Detailed Instructions

Task 2.1: Add code to retrieve the note text from Amazon DynamoDB

1. Return to the RDP session to access your development environment.

2. From the **IntelliJ Project** menu, expand **dictateFunction[dictate] > src > main > java > dictate >** open **Handler**

The class starts with all required dependencies imported. The next section initializes the SDK clients and resources outside of the handler function, which follows AWS best practices for improving performance. Next, the handler function that is called when your Lambda function is invoked.

The function reads a JSON payload from a source, calls DynamoDB to fetch the note text from the userId and noteId, calls the Polly synthesize_speech API to convert note text to speech, stores the resulting audio in an MP3 file in /tmp folder, uploads the MP3 file to Amazon S3, and creates a pre-signed URL for the MP3 file, then returns the URL as response object.

The handler function refers to two environment variables - **table name** to read the notes from a user and **bucket name** to upload the MP3 file of the audio converted notes.

You will first change the Lambda configuration to use these environment variables.

3. **Copy/Paste:** A bucket has already been created for you. Copy the value for the **MP3BucketName** to the left of these instructions as you will use it in the next step.

4. **Command:** Within the IntelliJ console, open a **Terminal** session (located in the bottom left of the console) and assign the environment variables **TABLE_NAME and MP3_BUCKET_NAME** with the command below:

NOTE:: Be sure to update the <MP3BucketName> placeholder with the value to the left of these instructions.

Copy Code

```
set env-parms=TABLE_NAME=Notes, MP3_BUCKET_NAME=<MP3BucketName>
```

Example:

```
**********************************
**** This is an EXAMPLE ONLY. ****
**********************************

set env-parms=TABLE_NAME=Notes, MP3_BUCKET_NAME=labstack-itkypjqcjpa-
pollynotesapibucket-1xuyfcr099i0b
```

Expected Output:

None, unless there is an error.

5. **Command:** Update the **dictateFunction** with the Environment Variables using the the command below:

Copy Code

```
aws lambda update-function-configuration --function-name dictateFunction --environment
Variables="{%env-parms%}"
```

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

Architectures:
- x86_64
CodeSha256: jYP1nUIZ6A7Mqr3J5GwlBCMva56NpgTZDLKGRkT7LQo=
CodeSize: 8928
Description: ''
Environment:
  Variables:
    MP3_BUCKET_NAME: labstack-hornmarc-8calzsqggsu-pollynotesapibucket-
```

```
knwlpqvy2zni
    TABLE_NAME: Notes
EphemeralStorage:
  Size: 512
FunctionArn: arn:aws:lambda:ap-southeast-
1:350214907448:function:dictateFunction
FunctionName: dictateFunction
Handler: example.Hello::handleRequest
LastModified: 2022-04-12T01:47:34.000+0000
LastUpdateStatus: InProgress
LastUpdateStatusReason: The function is being created.
LastUpdateStatusReasonCode: Creating
MemorySize: 512
PackageType: Zip
RevisionId: 37801d65-d7a6-463e-a9cd-311e6287b9d3
Role: arn:aws:iam::350214907448:role/lambdaPollyRole
Runtime: java11
State: Active
Timeout: 15
TracingConfig:
  Mode: PassThrough
Version: $LATEST
```

After the function has the required values, it then processes the core logic. You will have to decide which choice is correct for each of the remaining functions. Use the AWS Java SDK Table resource documentation for the next steps.

6. The first task is to retrieve the note text requested by the event. From **IntelliJ** in the Handler.java file, scroll to the **getNoteText** function to complete your first coding challenge.

7. **Copy/Paste:** For **TODO 1**, choose and insert the correct choice to retrieve the note text from the DynamoDB table.

- Choice A

Copy Code

```
var request = GetItemRequest.builder()
    .tableName(tableName)
    .key(key)
    .build();
```

- Choice B

Copy Code

```
var request = GetItemRequest.builder()
    .key(key)
    .build();
```

Answer: You can find the solution to this step in the TODO 1 Solution section at the bottom of these instructions.

Task 2.2: Add code to use Amazon Polly to convert the note text to an MP3 file

With the note text retrieved, you are ready to use Amazon Polly to convert it from text to audio, and Amazon Polly will return an audio stream if successful. You will then need to write the stream to an MP3 file

To complete the **TODO 2** section, use the provided parameters that are passed to the function.

Use the AWS SDK Java Polly Client Documentation for the next step.

8. **Copy/Paste:** For **TODO 2**, choose and insert the correct choice to convert the note text to a local MP3 file using the voiceId that user chose.

- Choice A

Copy Code

```
pollyclient.synthesizeSpeech(SynthesizeSpeechRequest.builder()
            .outputFormat("mp3")
            .text(noteText)
            .build(), destfile)
            .join();
```

- Choice B

Copy Code

```
pollyclient.synthesizeSpeech(SynthesizeSpeechRequest.builder()
            .outputFormat("mp3")
            .text(noteText)
            .voiceId(voice)
            .build(), destfile)
            .join();
```

Answer: You can find the solution to this step in the TODO 2 Solution section at the bottom of these instructions.

Task 2.3: Add code to upload the MP3 file to Amazon S3 and generate a pre-signed URL

Complete **TODO 3** to upload the file and generate and return a pre-signed URL variable. The URL needs to allow access to anyone who has the link and only be valid for a limited amount of time. This access needs to be granted without making the bucket or object public.

9. **Copy/Paste:** For **TODO 3**, choose and insert the correct choice to upload the MP3 file to Amazon S3 and generate a pre-signed URL active for 5 minutes.

- Choice A

Copy Code

```
var presigner = S3Presigner.builder().build();
        var url = presigner.presignGetObject(GetObjectPresignRequest.builder()
                .signatureDuration(Duration.ofMinutes(5))
                .getObjectRequest(GetObjectRequest.builder()
                        .bucket(bucket)
                        .key(key)
                        .build())
                        .build())
                        .url();

        response = url.toString();
```

- Choice B

Copy Code

```
var presigner = S3PreSigner.builder().build();
        var url = presigner.presignGetObject(GetObjectPresignRequest.builder()
                .putObjectRequest(GetObjectRequest.builder()
                        .bucket(bucket)
                        .key(key)
                        .build())
                        .build())
                        .url();

        response = url.toString();
```

Answer: You can find the solution to this step in the TODO 3 Solution section at the bottom of these instructions.

10. Save and close Handler.java .

Task 3: Publish an AWS Lambda function

Task 3: Publish an AWS Lambda function

Your code for the AWS Lambda function consists of scripts or compiled programs and their dependencies. You use a deployment package to deploy your function code to Lambda. Lambda supports two types of deployment packages: container images and .zip file archives.

A .zip file archive includes your application code and its dependencies. When you author functions using the Lambda console or a toolkit, Lambda automatically creates a .zip file archive of your code.

When you create functions with the Lambda API, command line tools, or the AWS SDKs, you must create a deployment package. You also must create a deployment package if your function uses a compiled language, or to add dependencies to your function. To deploy your function's code, you upload the deployment package from Amazon Simple Storage Service (Amazon S3) or your local machine.

High-Level Instructions

- Run Maven install to generate the project build.
- Update the code of the function, created in task 1, with your deployment package.
- Update the function configuration for the handler, set memory to 500 and the timeout to 60.

Detailed Instructions

1. Change directories to c:\code\dictateFunction :

Copy Code

```
cd C:\code\dictateFunction
```

✓ **Expected Output:**

None, unless there is an error.

2. **Command:** Run the Maven install command below to generate the project build:

Copy Code

```
mvn clean install
```

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing C:\code\dictateFunction\target\dictateFunction.jar
```

```
with C:\code\dictateFunction\target\dictate-1.0-shaded.jar
[INFO] Dependency-reduced POM written at:
C:\code\dictateFunction\dependency-reduced-pom.xml
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ dictate
---
[INFO] Installing C:\code\dictateFunction\target\dictateFunction.jar to
C:\Users\Administrator\.m2\repository\dictateFunction\dictate\1.0\dictate-
1.0.jar
[INFO] Installing C:\code\dictateFunction\dependency-reduced-pom.xml to
C:\Users\Administrator\.m2\repository\dictateFunction\dictate\1.0\dictate-
1.0.pom
[INFO] ----------------------------------------------------------------
--------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------------------
--------
[INFO] Total time:  7.018 s
[INFO] Finished at: 2022-04-12T01:50:32Z
[INFO] ----------------------------------------------------------------
--------
```

3. **Command:** Upload the new code to the **dictateFunction** with the command below:

Copy Code

aws lambda update-function-code --function-name dictateFunction --zip-file fileb://target/dictateFunction.jar

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

Architectures:
- x86_64
CodeSha256: 3sCC40sQOyTNxW7K1KqoER3T7nvJE9aMUnwwqhAKXdM=
CodeSize: 20105943
Description: ''
Environment:
  Variables:
    MP3_BUCKET_NAME: labstack-hornmarc-8calzsqggsu-pollynotesapibucket-
knwlpqvy2zni
    TABLE_NAME: Notes
EphemeralStorage:
  Size: 512
FunctionArn: arn:aws:lambda:ap-southeast-
1:350214907448:function:dictateFunction
FunctionName: dictateFunction
```

```
Handler: dictate.Handler
LastModified: 2022-04-12T01:56:06.000+0000
LastUpdateStatus: InProgress
LastUpdateStatusReason: The function is being created.
LastUpdateStatusReasonCode: Creating
MemorySize: 500
PackageType: Zip
RevisionId: 8f0cc25c-b7b4-4867-a3cb-195ecccc52a8
Role: arn:aws:iam::350214907448:role/lambdaPollyRole
Runtime: java11
State: Active
Timeout: 60
TracingConfig:
  Mode: PassThrough
Version: $LATEST
```

This Lambda function has the handler set to the value you set earlier.You need to change the handler of the Lambda function so that the service knows what Java method to run when it is invoked. You can also update other configurations of your function such as the memory allocation and set the timeout value.

4. **Command:** Run the command below to update the **function handler**, **memory allocation** and **timeout values**:

Copy Code

aws lambda update-function-configuration --function-name dictateFunction --handler dictate.Handler --memory 500 --timeout 60

Expected output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

Architectures:
- x86_64
CodeSha256: 3sCC40sQOyTNxW7K1KqoER3T7nvJE9aMUnwwqhAKXdM=
CodeSize: 20105943
Description: ''
Environment:
  Variables:
    MP3_BUCKET_NAME: labstack-hornmarc-8calzsqggsu-pollynotesapibucket-
knwlpqvy2zni
    TABLE_NAME: Notes
EphemeralStorage:
  Size: 512
FunctionArn: arn:aws:lambda:ap-southeast-
1:350214907448:function:dictateFunction
FunctionName: dictateFunction
```

```
Handler: example.Hello::handleRequest
LastModified: 2022-04-12T01:55:01.000+0000
LastUpdateStatus: InProgress
LastUpdateStatusReason: The function is being created.
LastUpdateStatusReasonCode: Creating
MemorySize: 512
PackageType: Zip
RevisionId: 5b5c211c-562b-4844-99a4-feec6a40a57a
Role: arn:aws:iam::350214907448:role/lambdaPollyRole
Runtime: java11
State: Active
Timeout: 15
TracingConfig:
  Mode: PassThrough
Version: $LATEST
```

Task 4: Invoke an AWS Lambda function

Task 4: Invoke an AWS Lambda function

It is time to test your function and make sure it works. In this task, you will use the **AWS CLI** and **AWS Management Console** to invoke your function.

High-Level Instructions

- Use the event below to invoke the dictate function.

Copy Code

```
{
"UserId": "newbie",
"NoteId": "2",
"VoiceId": "Joey"
}
```

- Use AWS CLI to invoke function using the above event and capture the response separately.
- Use AWS Lambda Console to use Test feature using the above event and review the output.

Detailed Instructions

Task 4.1: Invoke the AWS Lambda function in your IntelliJ terminal

You can use the **AWS CLI** to invoke your Lambda functions as a test or to offload processing from your scripts. The AWS CLI can invoke a function synchronously (and wait for the response), or asynchronously. To invoke a function asynchronously, use the **--invocation-type Event** option. For type **Event**, the command will succeed if the event was successfully submitted.

There is one more option for the **Invocation Type** option and that is **DryRun. DryRun** is helpful to validate parameter values and verify that the user or role has permission to invoke the function.

Answer: If you need additional guidance to complete this task, you can view the complete **Solution-dictateHandler.java** file in the **solutions** folder.

1. **Copy/Paste:** In the **src > main > resources** folder, update event.json with the following JSON object.

Copy Code

```
{
"UserId": "newbie",
"NoteId": "2",
"VoiceId": "Joey"
}
```

2. Save and close **event.json**.

3. **Command:** Run the command below to change into the resources directory:

Copy Code

```
cd C:\code\dictateFunction\src\main\resources
```

Expected Output:

None, unless there is an error.

4. **Command:** Run the command below to invoke the **dictateFunction**:

Copy Code

```
aws lambda invoke --function-name dictateFunction --payload fileb://event.json response.txt
```

Expected Output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************
```

```
ExecutedVersion: $LATEST
StatusCode: 200
```

The command should return a **200 status** saying that it was successful. There is now a new **response.txt** file in the **resources** folder. The output returned from the **dictateFunction** is captured in the **response.txt** file.

NOTE:: If you do not see the **response.txt** in the **resources** folder, you may need to update your project for it to show up.

5. Open the **response.txt** file.

This is a **pre-signed URL** that links to an **MP3** file in your bucket. To listen to it on your computer, paste the link (without the surrounding quotes) in a browser and you will hear the note that you requested.

Task 4.2: Invoke the AWS Lambda function in the AWS console

You can also manually invoke functions with the **AWS Management Console**. This method is graphical and gives you immediate access to all logs generated by the function. The only drawback is that it is not easily automated.

6. From the **AWS Management Console**, use the **AWS search bar** to search for Lambda and then choose the service from the list of results.
7. Choose the function name of **dictateFunction**.

8. Choose the Test tab and make the following selections under Test event:

- **Test event action:** Create new event

- **Event name:** testPolly

- **Event sharing settings:** Private

- **Template - optional:** hello-world

- **Event JSON:** Replace the default JSON object with the following:

Copy Code

```
{
"UserId": "newbie",
"NoteId": "2",
"VoiceId": "Joey"
}
```

9. Choose **Save changes** to save the test event.

10. Choose Test

11. In the **Execution result: succeeded (logs)** box, expand ▶ **Details**

12. Copy the URL returned (without the surrounding quotes) and browse to that URL.

Once again you see a URL to the MP3 file that you can test in your browser.

Optional Challenge Task 5: Create the remaining AWS Lambda functions used in the course application

Optional Challenge Task 5: Create the remaining AWS Lambda functions used in the course application

Subsequent labs will access DynamoDB with Lambda functions. In this task, you will create those functions. You have already explored how to access DynamoDB with the SDK in the previous lab, so the function code is provided for you. Review the Handler.java" file to make sure you understand what is happening as you use them to create and test the functions.

High-Level Instructions

- Using the aws lambda create-function command and the **jar files** in the **solutions** folder create the remaining functions:
  - createUpdateFunction
  - deleteFunction
  - listFunction
  - searchFunction
- Assign the lambdaPollyRole for all functions
- Test all of the functions

Detailed Instructions

Task 5.1: Create the functions

Lambda requires permissions to interact with other AWS services. This is granted with an AWS IAM role. You already know the name of the role created for use in this lab from **Task 1**. Use the

AWS CLI to create a variable with that role's ARN.

1. **Command:** Change directories to C:\code\dictateFunction\solutions directory with the following command:

Copy Code

```
cd C:\code\dictateFunction\solutions
```

Expected Output:

None, unless there is an error.

2. **Command:** Create a variable for the **ARN** of the **lambdaPollyRole** that the functions will use with the command below:

Copy Code

```
aws iam list-roles --output text --query "Roles[?contains(RoleName, `lambdaPollyRole`) == `true`].Arn"
```

Expected Output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

arn:aws:iam::123456789101:role/lambdaPollyRole
```

3. **Copy/Paste:** Copy the output for the **role ARN** as it will be used in the nex step.

4. **Command:** Run the command below to set the **role ARN** variable:

NOTE:: Be sure to replace the **role ARN** from the previous command in the place holder.

Copy Code

```
set roleArn=<role-ARN-goes-here>
```

Example:

```
**********************************
**** This is an EXAMPLE ONLY. ****
**********************************
```

```
set roleArn=arn:aws:iam::123456789101:role/lambdaPollyRole
```

Expected Output:

None, unless there is an error.

5. **Command:** Choose the correct command and use it to create the **createUpdateFunction** for **TODO 4**.

- Choice A

Copy Code

```
aws lambda delete-function --function-name createUpdateFunction ^
--handler createupdate.Handler ^
--runtime java11 ^
--memory 300 ^
--timeout 60 ^
--role "%roleArn%" ^
--environment Variables="{TABLE_NAME=Notes}" ^
--zip-file fileb://createupdateFunction.jar
```

- Choice B

Copy Code

```
aws lambda create-function --function-name createUpdateFunction ^
--handler createupdate.Handler ^
--runtime java11 ^
--memory 300 ^
--timeout 60 ^
--role "%roleArn%" ^
--environment Variables="{TABLE_NAME=Notes}" ^
--zip-file fileb://createupdateFunction.jar
```

Expected Output:

```
******************************
**** This is OUTPUT ONLY. ****
******************************

Architectures:
- x86_64
CodeSha256: x06pA9cPvyJUyZJQ38bAYj1z40cO8GrgZxPzKmx2QEg=
CodeSize: 20068113
Description: ''
Environment:
  Variables:
    TABLE_NAME: Notes
```

```
EphemeralStorage:
  Size: 512
FunctionArn: arn:aws:lambda:ap-southeast-
1:350214907448:function:createUpdateFunction
FunctionName: createUpdateFunction
Handler: createupdate.Handler
LastModified: 2022-04-12T02:05:50.101+0000
MemorySize: 300
PackageType: Zip
RevisionId: 6c216ad6-05ee-4dd9-a70b-65f9d8edda90
Role: arn:aws:iam::350214907448:role/lambdaPollyRole
Runtime: java11
State: Pending
StateReason: The function is being created.
StateReasonCode: Creating
Timeout: 60
TracingConfig:
  Mode: PassThrough
Version: $LATEST
```

Answer: You can find the solution to this step in the TODO 4 Solution section at the bottom of these instructions.

6. Use the steps in this task to create the remaining functions:

- deleteFunction
- listFunction
- searchFunction

Task 5.2: Test the new functions

Using what you learned in Task 4 and the included event.json file, complete the following tests on your new functions in the following order:

- List all notes.
- Create a new note.
- Search for that new note.
- Delete the note.

Consider: Need help?

Answer: In the **solutions** folder, refer to the OptionalChallengeSolution.txt file for **commands** and **test events**.

Code Challenge Solutions

Code Challenge Solutions

TODO 1 Solution:

- **Choice A is the correct code.**

Copy Code

```
        var request = GetItemRequest.builder()
                .tableName(tableName)
                .key(key)
                .build();
```

- Choice B is incorrect because it is missing the **.tableName(tablename)** property to be able to target the proper table

Return to the instructions

TODO 2 Solution:

- Choice A is incorrect because it does not specify the **.voiceId(voice)** property.
- **Choice B is the correct code.**

Copy Code

```
    pollyclient.synthesizeSpeech(SynthesizeSpeechRequest.builder()
                    .outputFormat("mp3")
                    .text(noteText)
                    .voiceId(voice)
                    .build(), destfile)
                    .join();
```

Return to the instructions

TODO 3 Solution:

- **Choice A is the correct code.**

Copy Code

```
    var presigner = S3Presigner.builder().build();
            var url = presigner.presignGetObject(GetObjectPresignRequest.builder()
                    .signatureDuration(Duration.ofMinutes(5))
                    .getObjectRequest(GetObjectRequest.builder()
                    .bucket(bucket)
                    .key(key)
                    .build())
                    .build())
                    .url();

            response = url.toString();
```

- Choice B is incorrect because it doesn't specify the duration of link to be expired by.

Return to the instructions

TODO 4 Solution:

- Choice A is incorrect because it is attempting delete action when it should be creating the function.

- **Choice B is the correct command.**

Copy Code

```
 aws lambda create-function --function-name createUpdateFunction ^
--handler createupdate.Handler ^
--runtime java11 ^
--memory 300 ^
--timeout 60 ^
--role "%roleArn%" ^
--environment Variables="{TABLE_NAME=Notes}" ^
--zip-file fileb://createupdateFunction.jar
```

Return to the instructions

RDP Connection Options

RDP Connection Options

Connect to Your Windows Dev Instance using Apache Guacamole

In this task you manually verify the **Public IPv4 address** is running by connecting to a **Apache Guacamole Server** hosted on the Amazon EC2 instance.

1. Search for EC2 using the **Unified Search Bar** at the top of the AWS Console, and choose the service from the listed results.
2. Choose **Instances** from the Instances sub-menu in the Amazon EC2 navigation bar.
3. Fill the checkbox ☑ for the **Apache Guacamole Server**
4. Scroll down in **Details** tab and copy **Public IPV4 address** value and paste in a **notepad**
5. Copy the **GuacamoleLink** http://Public IPv4 addre and paste it into a browser **replacing** *Public IPV4 address* of the **Apache Guacamole Server** copied from the previous task. .

   .
6. Go to the Apache Guacamole sign-in, in the browser and sign in using the following steps:

- For**Username**, enter: student
- For **Password**, use value for AWSPa$.
- Choose **Log In**.

The connection to your remote instance should start momentarily. Once you open a connection, you will see an image of the Dev instance desktop. You can interact with this image just as you would your normal desktop or any remote desktop client.

You are now connected to your Windows Dev instance in the browser using Guacamole.

NOTE: You will be prompted with a Networks pop-up window asking: **Do you want to allow your PC to be discoverable by other PCs and devices on this network?** Choose **No**.

NOTE: How to copy and paste when using Guacamole browser session?

- Open the Clipboard editor by choosing **Ctrl -> Alt -> Shift** on Windows or **command -> control -> shift** on MacOS.
- Copy your text from lab instructions and paste it to the Clipboard editor of the session.
- Close the Clipboard editor by choosing **Ctrl -> Alt -> Shift** on Windows or **command -> control -> shift** on MacOS.
- Now, you can use regular paste commands in your session from Clipboard. You can also either edit or replace the clipboard for the subsequent copies.

Return to the instructions

Connect to the Windows Dev Instance from a MacOS Machine

In this section, you connect to a Windows Amazon EC2 instance from your MacOS machine.

In this task you manually verify the **Public IPv4 address** is running by connecting to a **Windows Dev Instance** hosted on the Amazon EC2 instance.

7. Search for `EC2` using the **Unified Search Bar** at the top of the AWS Console, and choose the service from the listed results.
8. Choose **Instances** from the Instances sub-menu in the Amazon EC2 navigation bar.
9. Fill the checkbox ☑ for the **Windows Dev Instance**
10. Scroll down in **Details** tab and copy **Public IPV4 address** value and paste in a **notepad**
11. Install Microsoft Remote Desktop if it is not already installed. To install, complete the following:

- At the top of the screen select the Apple icon.
- Select **About This Mac**.
- Take note of your MacOS version
- From the Dock launch **App store**.
- Search for the following string: Microsoft Remote Desktop .
- Choose **Install** or **GET** to install the appropriate version.
- At the time of this writing, the current MacOS version is OSX 10.15.7 and you install Microsoft Remote Desktop Version 10.5.2. The connection instructions will be based on this version.

12. To open **Microsoft Remote Desktop** on the Dock select **Launchpad**, then select **Microsoft Remote Desktop**.
13. To create a new connection, from the top menu select **Connections > Add PC** and fill in the following fields:

- **PC name:** *Paste the **Public IPV4 address** of the Windows Dev Instance copied from the previous task.*
- **User account:** `Administ`
- **Friendly name:** `Windows Dev Ir`
- **Reconnect if the connection is dropped:** *Select this option if not already selected* .
- Choose **Add**.

14. For **Password**, use value for `AWSPa$`.
15. Go back to your Microsoft Remote Desktop connection window, and open the context menu for the **Windows Dev Instance** connection and choose **Open**

16. You will see the following message, **You are connecting to the RDP host "IP.Address.Listed.Here". There certificate couldn't be verified back to a root certificate. Your connection may not be secure. Do you want to continue?** Choose **Continue**.

17. Once prompted for the password, enter the password that you copied to the clipboard and choose **Continue**.

Result

Your connection to your remote instance should start momentarily. When lab instructions in subsequent sections require a command window, open or use a PowerShell window.

NOTE: You will be prompted with a Networks pop-up window asking: **Do you want to allow your PC to be discoverable by other PCs and devices on this network?** Choose **No**.

Return to the instructions

Connect to Your Windows Dev Instance from a Windows Machine

In this section, you connect to a Windows Amazon EC2 instance .

In this task you manually verify the **Public IPv4 address** is running by connecting to a **Windows Dev Instance** hosted on the Amazon EC2 instance.

18. Search for `EC2` using the **Unified Search Bar** at the top of the AWS Console, and choose the service from the listed results.
19. Choose **Instances** from the Instances sub-menu in the Amazon EC2 navigation bar.

20. Fill the checkbox ☑ for the **Windows Dev Instance**

21. Scroll down in **Details** tab and copy **Public IPV4 address** value and paste in a **notepad**

22. Open the Remote Desktop Connection application on your computer.
23. Choose the **Start** icon, and choose the **Search** icon. Type in `Remote Deskto` . Choose the application when it appears in the **Programs** list.
24. For **Computer**, paste the **Public IPV4 address** of the *Windows Dev Instance copied from the previous task*.

25. Choose **Connect**.

26. Remote Desktop Connection will prompt you with a login dialog asking for your username and password. By default, the application will use your current Windows username and domain. To change this, choose **Use another account**.

NOTE: On Windows 10, select **More Choices** before choosing **Use a different account**.

27. For your login credentials, use the following values:

- For **User name**, enter: `\Adminis`
- For **Password**, use value for `AWSPa$`.

NOTE: The `\` in the user name is important because it tells Remote Desktop Connection that you are logging in as the local Administrator and not as a domain user.

28. To connect to your instance, choose **OK**. If you receive a prompt that the certificate used to verify the connection was not a known, trusted root certificate, choose **Yes**.

Result

Your connection to your remote instance should start momentarily. When lab instructions in subsequent sections require a command window, open or use a Powershell window.

NOTE: You will be prompted with a Networks pop-up window asking: **Do you want to allow your PC to be discoverable by other PCs and devices on this network?** Choose **No**.

Return to the instructions

Summary:

Congratulations on completing the lab! For the **Java** version you can now:

- Create AWS Lambda functions and interact programmatically using AWS SDKs and the AWS CLI.
- Configure Lambda functions to use the environment variables and integrate with other services.
- Generate Amazon S3 pre-signed URLs using AWS SDKs and verify the access to bucket objects.
- Deploy the Lambda functions with .zip file archives through your IDE and test as needed.
- Invoke AWS Lambda functions using the AWS Console and AWS CLI.

Additional Resources

- AWS Lambda Documentation.
- Tutorial: Creating a Lambda function using Java
- Best practices for working with AWS Lambda functions

End Lab

Click to go up

Lab Complete