

Developing on AWS - Lab 7 - Developing an end to end Application with AWS



.Net version

© 2020 Amazon Web Services, Inc. and its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Corrections, feedback, or other questions? Contact us at [AWS Training and Certification](#).

Corrections, feedback, or other questions? Contact us at [AWS Training and Certification](#).

Overview

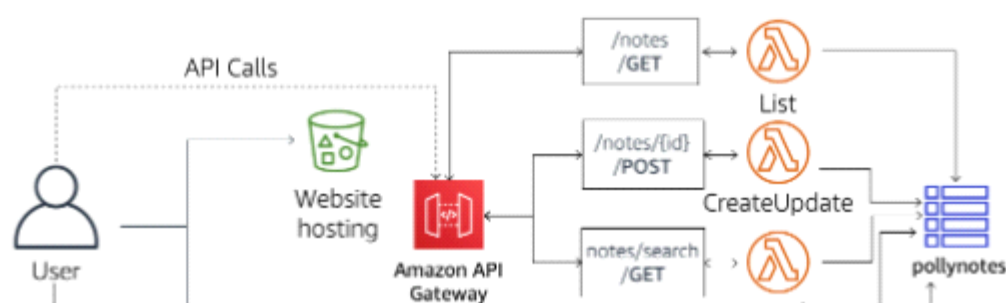
In this lab, you will learn how to use the AWS SDK and several AWS services to build an end to end serverless web application instead of the little pieces you have been building in the previous labs.

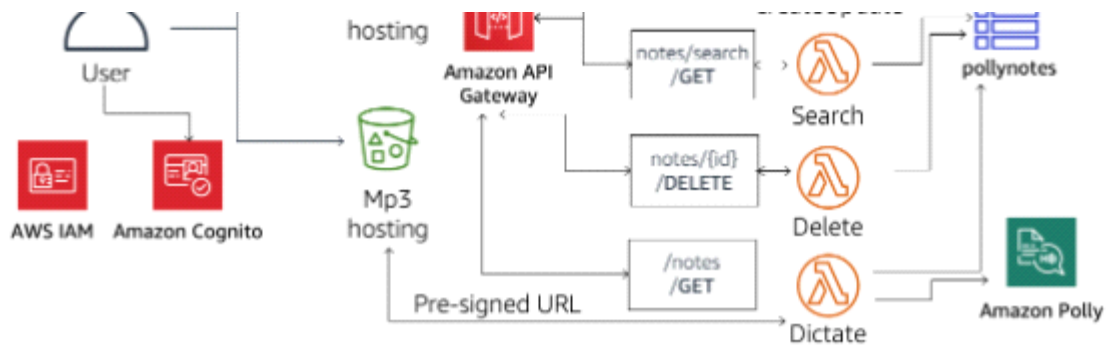
The idea of this web application is to provide a way for users to create and manage text notes and play them back audibly in different accents and voices by using Amazon Polly.

The web app will have the following:

- A frontend based on Angular whose files will be hosted on a static website in S3.
- A backend based on Amazon Cognito User Pools, API Gateway, Lambda, DynamoDB, and Polly.

There is a need to develop one Lambda function as part of this lab and use four Lambda functions that have already been coded for you. Once you complete the lab and you still have time remaining, take the challenge by coding three of those four Lambda functions yourself.





Objectives

After completing this lab, you will be able to:

- Host a Static Website in an S3 bucket.
- Create an IAM Policy and Role to give specific access to Lambda.
- Create an Amazon Cognito User Pool to authenticate users and control access to API Gateway.
- Create Lambda functions to perform CRUD operations on a DynamoDB table.
- Create a Restful API on API Gateway to front Lambda functions.
- Access the Polly service through API.

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi running Microsoft Windows or macOS.
- An Internet browser such as Chrome, Firefox, or IE9+. (previous versions of Internet Explorer are not supported)
- An SSH client, such as PuTTY, or a Microsoft Remote Desktop client to connect to your development EC2 instance.

! IMPORTANT

You can use an iPad or tablet device to access these directions in the lab console.

Duration

Duration

This lab will require around **2 hours** to complete.

Start Lab

1. At the top of your screen, launch your lab by choosing **Start Lab**

This starts the process of provisioning your lab resources. An estimated amount of time to provision your lab resources is displayed. You must wait for your resources to be provisioned before continuing.

i If you are prompted for a token, use the one distributed to you (or credits you have purchased).

2. Open your lab by choosing **Open Console**

This opens an AWS Management Console sign-in page.

3. On the sign-in page, configure:

- **IAM user name:** `awsstudent`
- **Password:** Paste the value of **Password** from the left side of the lab page
- Choose **Sign In**

⚠ Do not change the Region unless instructed.

Common Login Errors

Common Login Errors

Error: You must first log out

Amazon Web Services Sign In

You must first log out before logging into a different AWS account.

To logout, [click here](#)

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose **click here**
- Close your browser tab to return to your initial lab window
- Choose [Open Console](#) again

Task 1: Connecting to Your Development Environment

4. Establish a Connection to the Windows Dev Instance.

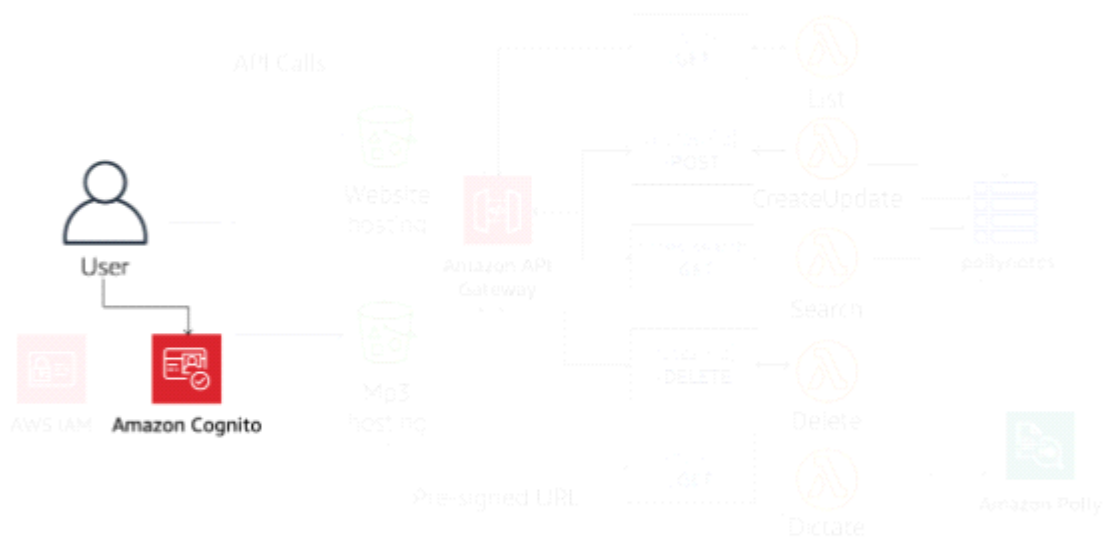
For detailed directions, see: [Appendix 3: Connecting to Your Development Environment](#).

Task 2: Amazon Cognito Authentication

Task 2: Amazon Cognito Authentication

In this section, you will:

- Create an Amazon Cognito user pool, which you are going to use to authenticate your end users and control access through API Gateway, making sure that your endpoint is secure.
- Create and confirm an Amazon Cognito user by using the command line. The Amazon Cognito user is used to test the end to end application.



For more information on Amazon Cognito User Pools, see: [Amazon Cognito User Pools](#)

Task 2.1: Creating an Amazon Cognito User Pool

5. Choose **Services** and select **Cognito**.

! IMPORTANT

Verify that you are in the **correct region**. If you are unsure of the region, verify the value with your instructor. Every service that is used in the lab (Amazon Cognito, API Gateway, Lambda, and DynamoDB) must be in the same region.

the value with your instructor. Every service that is used in the lab (Amazon Cognito, API Gateway, Lambda, and DynamoDB) must be in the same region. Take a note of this region and every time the instructions prompt you to verify your region, make sure it is the same.

6. Choose **Manage User Pools**.
 7. Choose **Create a user pool**.
 8. For **Pool name**, enter: `PollyNotesPool`
 9. For **How do you want to create your user pool?**, choose **Step through settings**.
 10. Go to the **Which standard attributes do you want to require?** section.
- ! EXTREMELY IMPORTANT**
- This step can't be modified once the User Pool is created. To correct this, you would need to delete the current User Pool and create a new one.
11. Unselect the checkbox next to ☐ **email** and choose **Next step**.
 12. In the **What password strength do you want to require?** section, you set the specifics required for the password. The password used is for testing purposes and you will be entering it many times so it has limited security settings.

Make the following configuration:

- **Minimum length** `6`

Unselect the following options:

- ☐ **Require numbers**
- ☐ **Require special character**
- ☐ **Require uppercase letters**
- ☐ **Require lowercase letters**

- ☐ **Require lowercase letters**

13. Choose **Next step**.

In the navigation menu on the left, you will find a list of all of the main steps for the creation of the Amazon Cognito User Pool. As many settings can stay as the defaults, you will skip ahead and go to the ones that need to be modified.

14. Choose **App clients**.

15. Choose **Add an app client**.

IMPORTANT

Make sure that you follow the next steps exactly as this configuration can't be undone. To correct, you would have to delete the current App client and create a new one.

- **App client name:** PollyNotesAngular
- Unselect ☐ **Generate client secret**.

16. Choose **Create app client**.

17. In the navigation menu on the left, choose **Review** and review all of the settings.

18. Choose **Create pool**.

19. Save the **Pool Id** and the **Pool ARN** from your Amazon Cognito Pool information into a separate file for use later in this lab.

Note

Make sure to note that it is for **Amazon Cognito Pool Id** and **Amazon Cognito Pool ARN** as well.

For example, in the Oregon region, the **Pool Id** would look like **us-west-2_YYYYYYYYYY** and the **Pool ARN** would look like **arn:aws:cognito-idp:us-**

For example, in the Oregon region, the **Pool Id** would look like **us-west-2_XXXXXXXXXX** and the **Pool ARN** would look like **arn:aws:cognito-idp:us-west-2:012345678901:userpool/us-west-2_XXXXXXXXXX**.

20. In the navigation pane on the left, under **General settings**, choose **App clients**.
21. Save the **App client id** in the form of xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx information into the same file. Make sure to note that it is for the **Amazon Cognito App client id**.

Task 2.2: Creating a User for the Amazon Cognito User Pool

In this section, you will create and confirm an Amazon Cognito user by using the command line. The Amazon Cognito user is used to test the end to end application.

Note

For this task, you should be using PowerShell from the Windows instance.

22. Establish a Connection to the **Windows Dev Instance**.
23. Open the PowerShell application on the Windows Instance or from the Linux instance, use the terminal session.

PowerShell is available as a shortcut in the taskbar menu. From PowerShell, you will run the sign-up script.

In the following command, change **change-me_app-client-id** with the value of the **App client id** that you copied to a file earlier and then, run the command:

```
aws cognito-idp sign-up --client-id change-me_app-client-id --  
username student --password student
```

```
aws cognito-idp sign-up --client-id change-me_app-client-id --  
username student --password student
```

Here is an example of the command and the output.

```
{  
  "UserConfirmed": false,  
  "UserSub": "abc70c2f-52v7-4abc-c195-012abc8560ab"  
}
```

Note

The values in the output should show your specific configuration.


You have now created a user in Amazon Cognito with the username `student` and a password of `student`. Even though this isn't secure, you are using a simple username and password for simplicity during testing.


24. Next, confirm the user that you created. You will need to change **change-me_pool-id** with the value of the **Pool Id** that you copied to a file earlier. To confirm the user, enter the following command:

```
aws cognito-idp admin-confirm-sign-up --user-pool-id change-  
me_pool-id --username student
```

If you don't see any output, it means that the command is successful and you can continue.

25. Go back to the **Amazon Cognito** console for the [User Pools](#). Under **General settings**, choose **Users and groups**.

You should see the username `student` and the status is **CONFIRMED**. If you do not see it, choose  to refresh the browser.

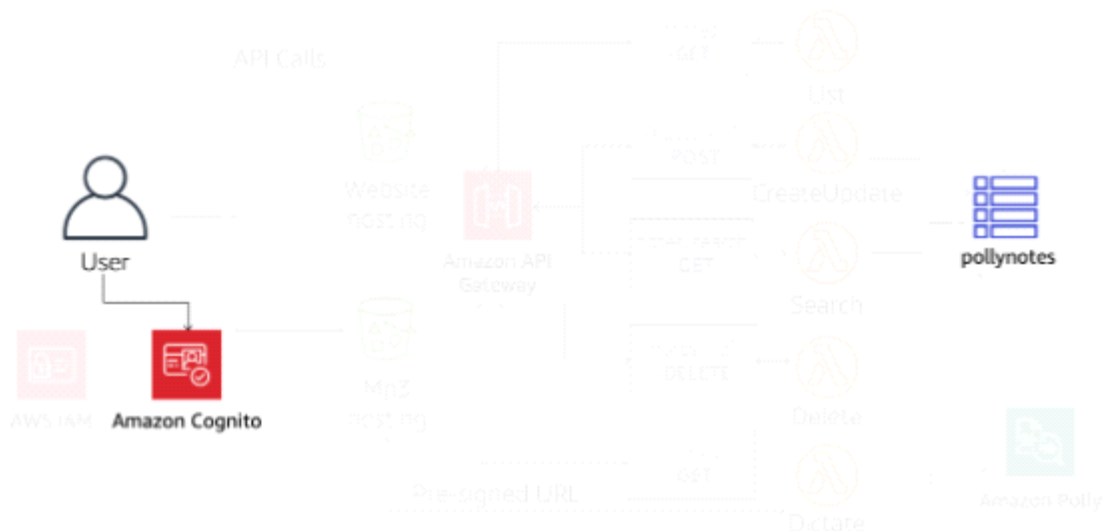
 Congratulations! You have successfully created an Amazon Cognito User Pool, manually added a user, and confirmed that user.

Pool, manually added a user, and confirmed that user.

Normally, the sign-up process would be done via the application. However, the application that you are using doesn't provide such a function. So instead, it is manually created.

Task 3: Creating a DynamoDB table

In this section, you will create a table in DynamoDB called **pollynates**. This table is going to be used to store your user id, along with your notes.



26. In the AWS Management Console, choose **Services** and select **DynamoDB**.

27. Choose the **Tables** link on the navigation menu on the left side.

! IMPORTANT Verify that you are in the **correct region** based on what you have noted before. If you are unsure of the region, verify the value with your instructor. Make sure your region name is consistent

have noted before. If you are unsure of the region, verify the value with your instructor. Make sure your region name is consistent.

! IMPORTANT

The entries below are case sensitive, make sure that you copy and paste the entries from the lab console.

28. Choose **Create table**.

29. **Table name:** pollynotes

30. **Primary key:** userId (String)

31. Enable **Add sort key**.

32. **Add sort key:** noteId (String)

Create DynamoDB table Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* pollynotes ⓘ

Primary key* Partition key

userId String ⓘ

☒ Add sort key

noteId String ⓘ

Choose **Create**.

It may take up to 15 seconds for the DynamoDB table to be created. There will be a spinning loading logo next to your table name. Wait for that to disappear before moving to the next step.

33. You should be placed on the pollynotes table pane. If not, choose the **pollynotes** table.

34. Choose the **Items** tab.

35. Choose **Create item**.

36. In the **VALUE** field next to **userId** enter: testuser

36. In the **VALUE** field next to **userId**, enter: `testuser`
37. In the **VALUE** field next to **noteld**, enter: `001`
38. Choose the **+** (plus circle icon) to the left of **noteld**.
39. Select **+ Append ▼**, and then choose **String**.
40. In the **FIELD**, field enter: `note`
41. In the **VALUE** field next to **note**, enter: `My note to myself`
42. Choose **Save**.
43. Create a few more items by going through the Create item process again.

For example:

- Your next **noteld** should be `002`, then `003`, and more.
- The **userId** should always be `testuser`.
- The **note** can be anything that you would like, but beware that it will be recited back by Polly at some point.

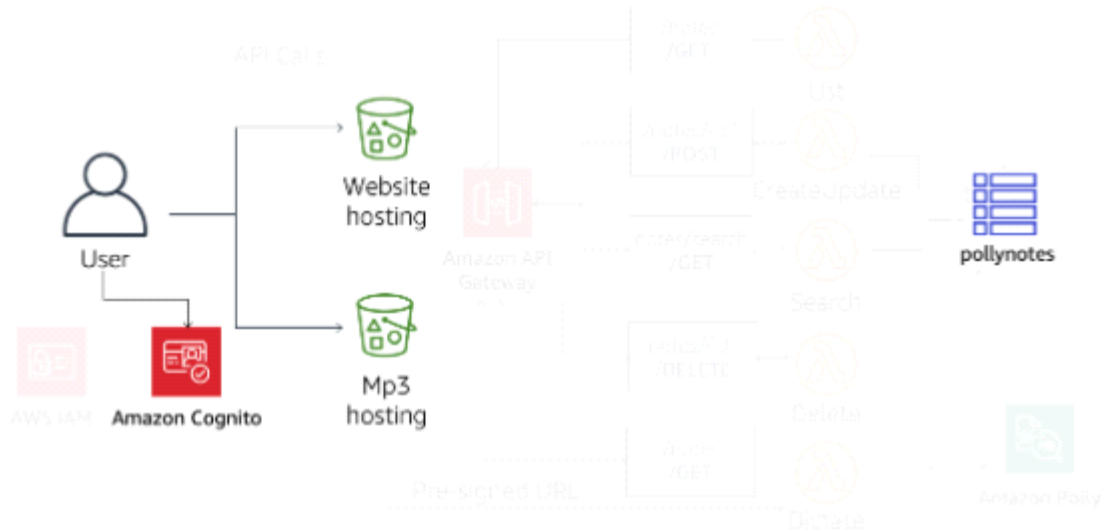
Task 4: Creating S3 buckets

In this section, you will create two S3 buckets:

- A website bucket which will host your front end website.
- An MP3 bucket which will store your MP3 files.

- An MP3 bucket which will store your MP3 files.

You will only be creating the buckets in this section and you will configure them in a later step.



44. Choose **Services** and select **S3**.

! IMPORTANT

You can see that the region is Global. This is because S3 does not require region selection. However, S3 creates buckets in a region you specify. When creating your buckets, select the correct region based on what you noted before. If you are unsure of the region, verify the value with your instructor.

Task 4.1: Create the Web Bucket

45. Choose **Create bucket**.

46. **Bucket name:** (all lowercase):

`polly-notes-web-<firstname>-<lastname>`

Note

For example, if your name is John Smith, your bucket name would be **polly-notes-web-john-smith**.


For example, if your name is John Smith, your bucket name would be **polly-notes-web-john-smith**.

47. Make sure that the **Region** field is set to the appropriate region that you have noted earlier.
48. Choose **Create bucket**.

Task 4.2: Create the MP3 Bucket

In this task you will create another S3 bucket to hold the Polly audio files.

49. Choose **Create bucket**.
50. **Bucket name:** (all lowercase):
`polly-notes-mp3-<firstname>-<lastname>`

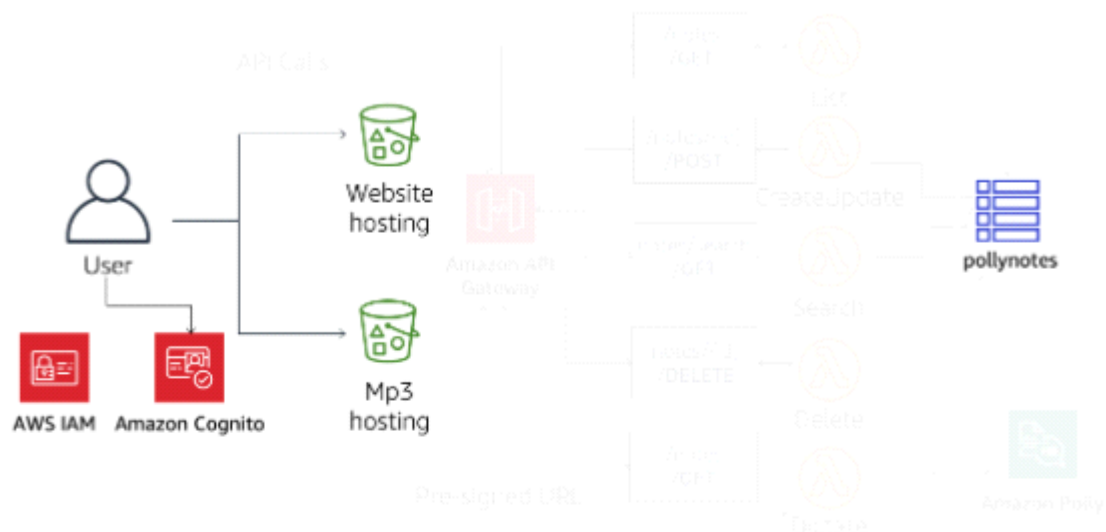
 **Note** For example, if your name is John Smith, your bucket name would be **polly-notes-mp3-john-smith**.
51. Make sure that the **Region** field is set to the appropriate region that you have noted earlier.
52. Choose **Create bucket**.

Task 5: Reviewing an IAM Managed Policy and Creating an IAM Role

In this section, you will review an IAM Managed Policy and create an IAM Role to allow your Lambda functions to:

In this section, you will review an IAM Managed Policy and create an IAM Role to allow your Lambda functions to:

- Perform CRUD operations on DynamoDB.
- Call the Polly API.
- Perform the PUT operation on S3 for the MP3.
- Log events to Cloudwatch.



53. Choose **Services** and select **IAM**.

! IMPORTANT

You can see that the region is Global. IAM does not require region selection because it's a global service.

Task 5.1: Review an IAM Policy

54. To save time the **lambda_ddd_s3_polly** managed policy has already been created.


55. The policy includes the following permissions:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "s3:PutObject",  
      "Resource": "arn:aws:s3:::lambda-ddd-s3-polly/*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "s3:DeleteObject",  
      "Resource": "arn:aws:s3:::lambda-ddd-s3-polly/*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "dynamodb:List*",  
      "Resource": "arn:aws:dynamodb:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:List*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:Describe*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:StartSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:StopSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:DeleteSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:List*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:Describe*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:StartSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:StopSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    },  
    {  
      "Effect": "Allow",  
      "Action": "polly:DeleteSpeechSynthesis*",  
      "Resource": "arn:aws:polly:*:*:*:*:*:*:*:*:*:*",  
      "Condition": {  
        "StringEquals": {  
          "aws:PrincipalTag": "polly",  
          "aws:PrincipalTag": "pollynotes"  
        }  
      }  
    }  
  ]  
}
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:UpdateItem",
        "dynamodb:DescribeTable",
        "polly:SynthesizeSpeech",
        "s3:PutObject",
        "s3:GetObject",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": ["*"]
    }
  ]
}
```

Task 5.2: Creating an IAM Role

In this section, you will create an IAM Role called **PollyNotesRole** and attach the **lambda_ddb_s3_polly** policy. This Role will be used by the Lambda functions in the next sections. This IAM role allows you to delegate access with defined permissions in the above policy to trusted entities without having to hard code your credentials in your code.

56. In the navigation pane, choose **Roles**, and then choose [Create role](#).
57. Under **Select type of trusted entity**, choose  **AWS service**.
58. Go to the **Choose a use case** section. Choose **Lambda**, and then, choose [Next: Permissions](#).
59. In the **Search** field, enter: `lambda_ddb_s3_polly`

59. In the **Search** field, enter: `lambda_ddb_s3_polly`

60. Select the ☒ `lambda_ddb_s3_polly` policy.

61. Choose **Next: Tags**.

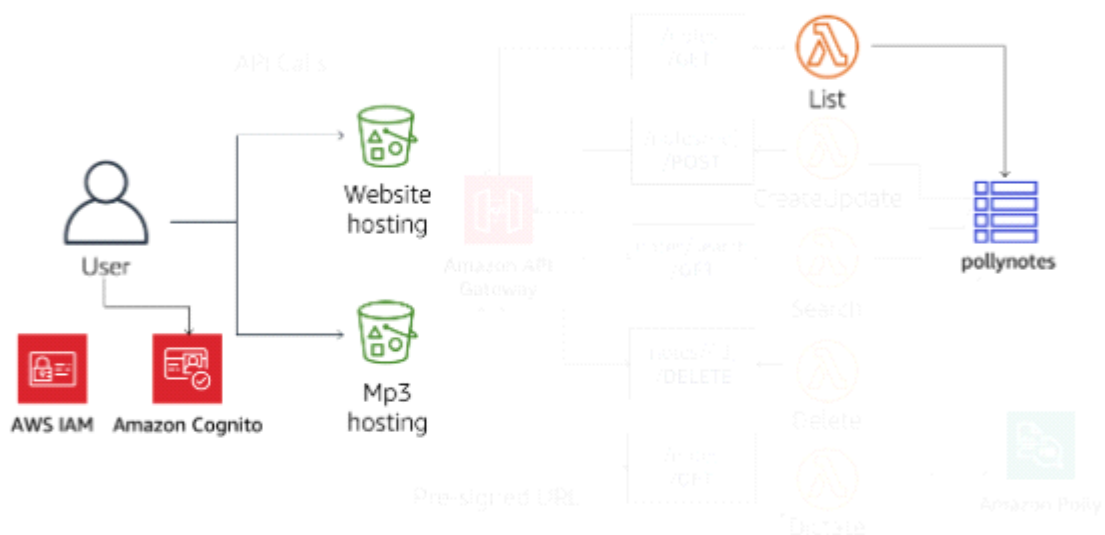
62. Choose **Next: Review**.

63. For **Role name**, enter: `PollyNotesRole`

64. Choose **Create role**.

Task 6: Creating the Lambda ListFunction


In this section, you will create your Lambda List function. To do this, you edit the *ListFunction.sln* file.



Task 6.1: Getting to the ListFunction code

65. Go back to the **Windows Dev Instance**.

66. Navigate using **File Explorer** to the folder **C:\temp\workdir\lab7CSharpLab** and open the file **PollyNotes.sln**.

 **Note:** If you get a prompt asking *How do you want to open this type of file (.sln)?*, select **Visual Studio 2017** and choose **Not now, maybe later** on the next screen.

Leave the defaults on the next screen and choose **Start Visual Studio**.

There are many projects inside of this solution. For this specific Lambda function, you will focus on the following two:

- **ListFunction:** The project that you will use to develop the ListFunction Lambda function.
- **ListFunctionSolution:** The solution of the ListFunction that you will develop. You can use this for reference.

67. Open the file **ListFunction.cs** under the ListFunction project.

It may seem that there is a lot of code already written for you. However, you will quickly realize that only the structure for the Lambda function has been created so that you don't have to spend much time on this.

68. Take a look at the current code to get a better understanding. Examine the following:

- The function call that you will develop is called **FunctionHandler**.
- The **FunctionHandler** takes in the Note as a first argument. The *context* argument isn't used here.
- The other function is a **Main()** which will be used to test your function

argument isn't used here.

- The other function is a `Main()` which will be used to test your function once it is coded. You can look at it to understand that only a `userId` is being passed to your **`FunctionHandler()`** function. That `Main` is serializing JSON into a `Note` POCO and passing it to your `FunctionHandler`.

Task 6.2: Coding the ListFunction

69. It is now time for you to code! You will need to edit the **`ListFunction.cs`** under the **`FunctionHandler`** only. If you would rather not code and just use the Solution file, skip to [Task 6.3](#)

- Your goal is to develop a Lambda function that will receive a `userId` as an input and return a list of `Note` objects.
- .NET Core is a strong type language which means that receiving JSON isn't as easy to parse as some other languages. Instead, Lambda and .NET work together to deserialize the JSON payload into an object.
- The input from Lambda will look similar to the following:

```
{
  "userId": "testuser"
}
```

- It may look difficult to parse at first glance. However, the Lambda .NET Core will do the work for you by serializing this JSON into a `Note` object that will have its `userId` property set and that you receive as an argument to your Lambda function as you can see in the function that you need to develop.

```
public List<Note> FunctionHandler(Note note, ILambdaContext
context)
```

- The output that Lambda is expecting is a List of Notes that should be

-
- The output that Lambda is expecting is a List of Notes that should be similar to the following:

```
[
  {
    "userId": "testuser",
    "noteId": "001",
    "note": "My note to myself"
  },
  {
    "userId": "testuser",
    "noteId": "002",
    "note": "My new note to myself"
  }
]
```

This may sound like a challenge to return this kind of data. However, by returning a `List<Note>`, the Lambda .Net core will serialize the list of Note and output the above JSON data.

- Comments have been left in the code to give you some help. However, it is your responsibility to make good use of those comments or to do it on your own. There are many ways to Query DynamoDB and the Solution code makes use of the Object Persistence Model that was discussed in the class. You should take a look at the following:

- [Query and Scan](#)
- [QueryAsync](#)

Because .NET Core can't use synchronous invocations of the C# SDK, you will have to rely on the Async functions for all of the calls to the APIs of AWS. This adds some complexity to the code as you will need to get a Task from the Async call and run a `Wait()` on that task. You will not be able to use the usual `() =>` or `await` (unless you create another function). Depending on the call you will make, the return value may be an [AsyncSearch](#), which you can then convert into a Task by using the [GetRemainingAsync](#) call.

the call you will make, the return value may be an [AsyncSearch](#), which you can then convert into a Task by using the [GetRemainingAsync](#) call.

The return call of the Async calls Task mentioned is typically what you are looking for. For example, if you were to create a `Task<List<Note>>` from your query, you should be able to return the `Result` directly.

Overall, the work you have to do is to query DynamoDB to find all of the notes of the userId based on the **note** argument and return a `List<Note>` based on the response of your query. Good luck and remember that there is always a Solution file that can help you out! Once you think you have something working or would like to test things out, follow the next steps.

70. To test your code locally instead of having to upload your code to Lambda and invoke it from there. You will run the project as a Console program, which will call the **Main()** function. This will simulate a call to your Lambda handler with the userId `testuser`. Even though this test runs locally, it's only your Lambda function that runs locally, your code is making an actual call to DynamoDB.

1. Make sure **ListFunction** is selected to be run (top of the screen).
2. Select **Debug -> Start Without Debugging** or press **Ctrl+F5**. You can also troubleshoot by using breakpoints and running the code by using **Debug -> Start Debugging** or **F5**. The output should look similar to the following:

```
[{"userId":"testuser","noteId":"001","note":"My note to myself"},  
{"userId":"testuser","noteId":"002","note":"My new note to  
myself"}]
```

The output shows a listing all of the notes that you created in your DynamoDB table in a prior steps.

Task 6.3: Upload your ListFunction to Lambda

Task 6.3: Upload your ListFunction to Lambda

71. It is time to upload your Lambda function to Lambda. To do so, you will use the AWS ToolKit that has been installed in Visual Studio. This simplifies the process instead of having to use the command line. What is done in the background for you is to create a ZIP of your code, creating the Lambda function by using the ZIP, and associating the IAM Role that you created earlier.

72. Depending on if you are deploying your own code or the Solution code, you will need to **Right-click** the appropriate project:

- If you have coded your function, **Right-click** on the **ListFunction** project in the Solution Explorer. Then, select **Publish to AWS Lambda...**
- If you want to use the Solution Code, **Right-click** the **ListFunctionSolution** project in the Solution Explorer. Then, select **Publish to AWS Lambda...**

73. For **Region**, select the Region for your lab as you noted it before.

Note that everything else is pre-populated for you. This is due to the *aws-lambda-tools-defaults.json* file that has been set with the appropriate Function Name, Language Runtime, Framework, and more. You can take a look at this file once you are done creating the Lambda function.

74. Choose **Next**.

75. For **Role Name**, select `PollyNotesRole`.

76. Choose **Upload**.

You have now uploaded your Lambda function!

The next step is to test the Lambda function from the Lambda UI.

77. Choose **Services** and select **Lambda**.

77. Choose **Services** and select **Lambda**.

78. Choose **PollyNotes-ListFunction**. If you don't see your function, make sure that you are in the correct Region.

79. Choose **Test**.

80. For **Event name**, enter: **ListTest**

81. Replace the JSON payload with the following:

```
{
  "userId": "testuser"
}
```

It should look very familiar as you are testing again to get the list of items.

82. Choose **Create**.

83. Choose **Test**.

It is normal for it to take a little longer as this is the first time you are running your Lambda function.

84. On the left, there should be a green check mark just under the name of your Lambda function name with the message *Execution result: succeeded (logs)*. Choose **Details** and you should see the output of your Lambda function listing a few notes that you have created in the DynamoDB section. For example:

```
[
  {
    "userId": "testuser",
    "noteId": "001",
    "note": "My note to myself"
  },
  {
    "userId": "testuser",
```

```

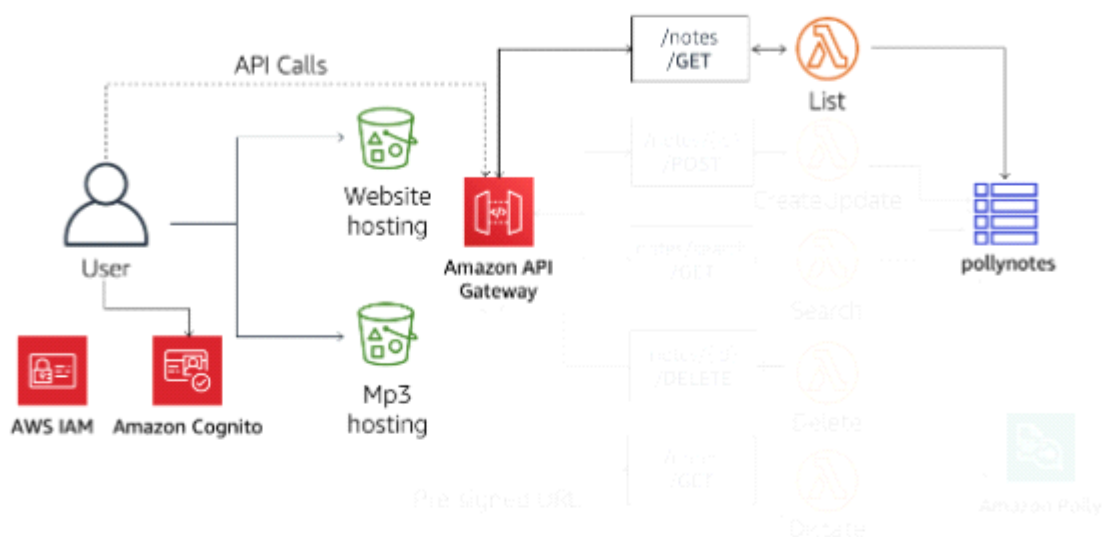
{
  "userId": "testuser",
  "noteId": "002",
  "note": "My new note to myself"
}
]

```

👍 Congratulations! You have just created the List Lambda function that will act as your backend when a user will need to list their notes.

Task 7: Creating the API Gateway

In this section, you will create a Restful API to front the back-end Lambda function. The API will use Amazon Cognito identities for authentication. For the application, you will be using the Amazon API Gateway service.



85. Choose **Services** and select **API Gateway**.

! IMPORTANT

Verify that you are in the **correct region** based on what you have noted

! IMPORTANT

Verify that you are in the **correct region** based on what you have noted before. If you are unsure of the region, verify the value with your instructor. Make sure your region name is consistent.

86. Under **Choose an API type**, select **REST API** and choose **Build**.

Note: There is a similar option labeled as **REST API Private**. Please be mindful not to select this option.

87. In the **Create your first API** dialog box, choose **OK**.

88. Under **Create new API**, select **New API**.

89. Enter the following formation:

- **API name**, enter: `PollyNotesAPI`
- **Description**, enter: `PollyNotesAPI`
- **Endpoint Type**, select `Regional`.

90. Choose **Create API**.

The screenshot shows the 'Create new API' dialog box in the AWS IAM console. At the top, it says 'Create new API' and 'An Amazon API Gateway API is a collection of resources and methods that can be invoked through HTTP endpoints.' Below this, there are three tabs: 'New API' (selected), 'Import from Swagger', and 'Example API'. Under the 'Settings' section, it says 'Choose a friendly name and description for your API'. There are two input fields: 'API name' with the value 'PollyNotesAPI' and 'Description' with the value 'PollyNotesAPI'. Below these is a dropdown for 'Endpoint type' with 'Regional' selected. At the bottom right, there is a 'Create API' button. A small asterisk and 'Required' text are visible at the bottom left.

Task 7.1: Creating the Amazon Cognito Authorizer

In this section, you create the Amazon Cognito Authorizer. The Authorizer will allow you to control access to the API with the Amazon Cognito User Pool.

will allow you to control access to the API with the Amazon Cognito User Pool.

91. In the left navigation menu, choose **Authorizers**.

92. Choose **+ Create New Authorizer**.

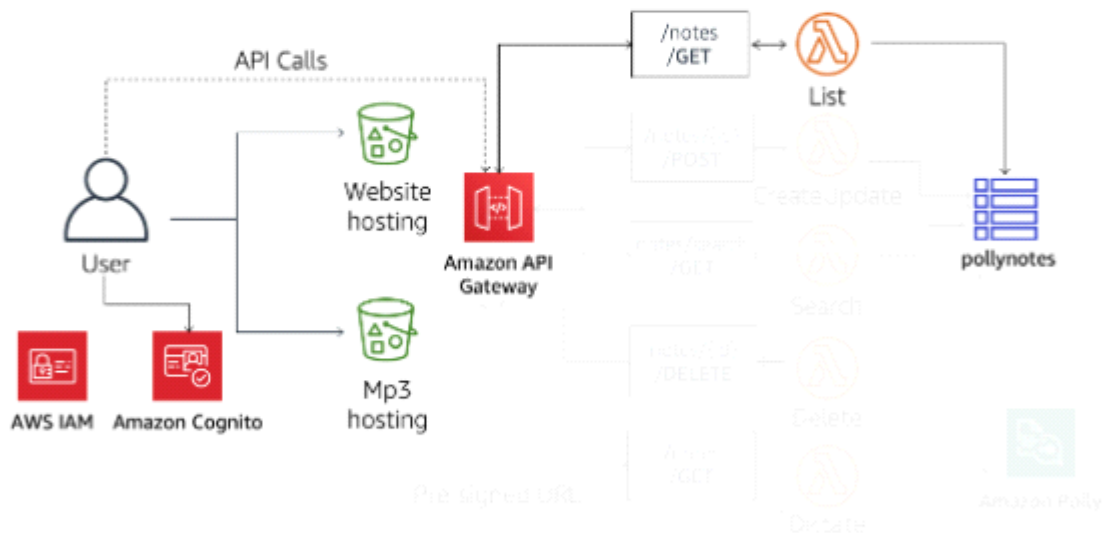
93. Enter the following information:

- **Name:** PollyNotesPool
- **Type:** Cognito
- **Cognito User Pool:** PollyNotesPool
- **Token Source:** Authorization
- **Token Validation:** Leave empty.

94. Choose **Create**.

Task 7.2: Creating the Resources

In this section, you create your API Resources.



95. In the navigation pane, choose **Resources**.

96. Under **Resources**, select / (root).

97. Choose **Actions ▾**, and then select **Create Resource**.

98. Under **Resource Name**, enter the following (the Resource path will populate automatically): `notes`

99. Choose **Create Resource**.

Now that you have created your resource, you need to create your Methods.

Task 7.3: Creating the API GET Method

100. If not already selected, choose the **Resources** link and select **/notes**.

101. Choose **Actions ▾**, and then select **Create Method**.

102. Select **GET** and then choose ☒ (check mark tick).


103. For **Integration type**, select ☒ **Lambda Function**.



104. Under **Lambda Function**, type an uppercase `P`, then select your Lambda **PollyNotes-ListFunction**.

105. Choose **Save**.

106. In the **Add Permission to Lambda Function** dialog box, choose **OK**.

107. Choose **Method Request**.

108. In the **Settings** section, next to **Authorization**, choose  (Edit).

108. In the **Settings** section, next to **Authorization**, choose  (Edit).
109. Select **PollyNotesPool** and then, choose ☒ (check mark tick).
110. In the breadcrumb, choose [← Method Execution](#).
111. Choose [Integration Request](#).
112. Expand the **Mapping Templates** section.
113. Select ☒ **When there are no templates defined (recommended)**.
114. Under **Content-Type**, choose  [Add mapping template](#).
115. Enter `application/json` and choose ☒ (check mark tick).
116. In the box that appeared below, enter the following:

```
{
  "userId": "testuser"
}
```

Generate template:

```
1 {
2   "userId": "testuser"
3 }
```

Note

You are hard coding the value of *userId* here so that when you do your first test from the web interface, you can see the original notes you have created in DynamoDB. Since you won't be creating the CreateUpdate Function until after doing your test, you would have no way to create a note for your user. The first step once you are done fully testing the List function, will be to change this mapping template to be dynamic based on the Amazon Cognito

The first step once you are done fully testing the List function, will be to change this mapping template to be dynamic based on the Amazon Cognito ID of the user login.

117. Choose **Save**.

118. Choose the **/notes** resource.

119. Choose **Actions** and select **Enable CORS**.

120. Under **Gateway Responses for PollyNotes API API**, check option for ☒ **DEFAULT 4XX** and ☒ **DEFAULT 5XX**.

121. Choose **Enable CORS and replace existing CORS headers**.

122. In the **Confirm method changes** dialog box, choose **Yes, replace existing values**.

123. Choose **Actions**, then select **Deploy API**. Make the following updates:

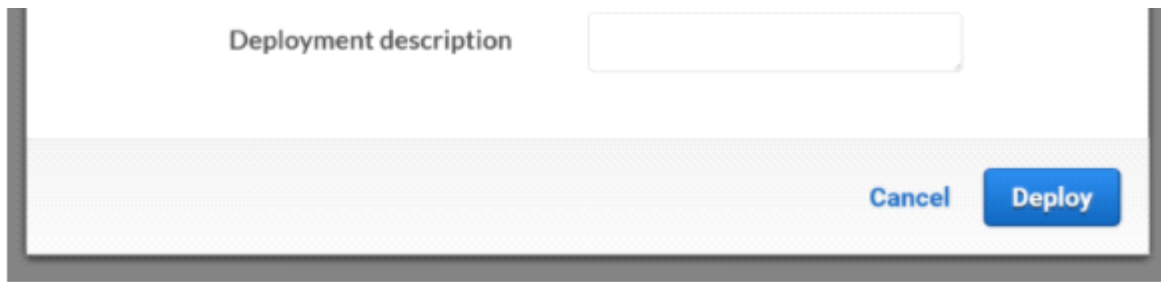
- **Deployment stage:** [New Stage]
- **Stage name:** prod

124. Choose **Deploy**.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	prod
Stage description	
Deployment description	

A screenshot of a deployment dialog box. At the top, there is a label "Deployment description" followed by a text input field. At the bottom right, there are two buttons: "Cancel" and "Deploy".

Deployment description

Cancel Deploy

125. Copy the **Invoke URL** and paste it into a file. You will need it later in the lab.

126. In the **prod Stage Editor**, choose the **Logs/Tracing** tab and make the following updates:

- **Cloudwatch Settings**
 - **Enable CloudWatch Logs** ☒
 - **Log level** INFO
 - **Log full requests/responses data** ☒


127. Choose **Save Changes**.

Task 8: Creating the Front-End Website

In this task, you create the front-end website.

128. Download and extract the following zip file:

Source file: [ZIP File](#)

 **Note:** If downloaded file says "zip can't be downloaded securely, click ^ and select Keep.

and select Keep.

129. Modify the *main.bundle.js* file (lines 2-4) and change the following values:

```
// Change the following 3 variable's value
var API_GATEWAY_INVOKE_URL = "https://your-api-url"
var COGNITO_POOL_ID = 'us-east-1-xxxxxxxxxxxxx'
var COGNITO_APP_CLIENT_ID = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

You should have noted the needed values from prior lab steps. If you do not have details, you can get them by using the Console and the following paths:

- **Services -> API Gateway -> PollyNotesAPI -> Stages -> prod (API Gateway Invoke URL)**
- **Services -> Cognito -> Manage User Pools -> PollyNotesPool (Cognito Pool ID)**
- **Services -> Cognito -> Manage User Pools -> PollyNotesPool -> App clients (Cognito App client ID)**

130. Next, you need to enable CORS on the MP3 Bucket. When users access your website, they will be loading the website from the static Website bucket. Then, they will use javascript in their browser to connect to two different endpoints; API Gateway and the MP3 bucket. You already enabled CORS on the API Gateway, but you now need to enable CORS on your MP3 bucket, so that it can set response headers.

131. Choose **Services** and select **S3**.

132. Choose your `polly-notes-mp3-<firstname>-<lastname>` bucket.

133. Choose the **Permissions** tab.

134. Navigate down to the **Cross-origin resource sharing** section, then click

Edit

135. Paste the following into the text box, then click **Save changes**

135. Paste the following into the text box, then click **Save changes**

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

136. Next, go to your Website bucket. In the breadcrumb trail, choose **Amazon S3**, and then, choose your `polly-notes-web-<firstname>-<lastname>` bucket.

137. Choose **Upload** from Objects section.

138. Upload all the files and folders in the folder named `polly-notes-web-bucket-root` by choosing **Add Folder** and Click Upload.

139. Move to the bottom of the screen, then, choose **Upload**.

- There should be a total of 10 files and folders in the S3 Bucket Console.

140. Choose **Exit**.

141. Choose the **Permissions** tab.

142. In the **Block public access (bucket settings)** card choose **Edit**.

143. Unselect ☐ **Block all public access**.

144. Choose **Save**.

143. Unselect ☐ **Block all public access**.

144. Choose **Save changes**.

145. A confirmation box will appear; type `confirm` and choose **Confirm**.

146. Move down to the **Bucket policy** card and choose **Edit**.

147.  Copy and paste the following bucket policy.

 Ensure that you replace your bucket name.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::polly-notes-web-<firstname>-<lastname>/*"
    }
  ]
}
```

148. Choose **Save changes**.

149. In the **Bucket overview** card, your bucket should now be flagged with **Access** as **Public**.

150. Choose the **Properties** tab.

151. In the **Static website hosting** card, choose **Edit**.

Configure the following settings:

- **Static website hosting:** ☒ Enable
- **Index document:** `index.html`
- **Error document:** `error.html`

- **Index document:** `index.html`
- **Error document:** `error.html`

152. Choose **Save changes**.

153. Move to the **Static website hosting** card and copy your *Bucket website endpoint* URL to a text editor.

You will use this again in the lab.

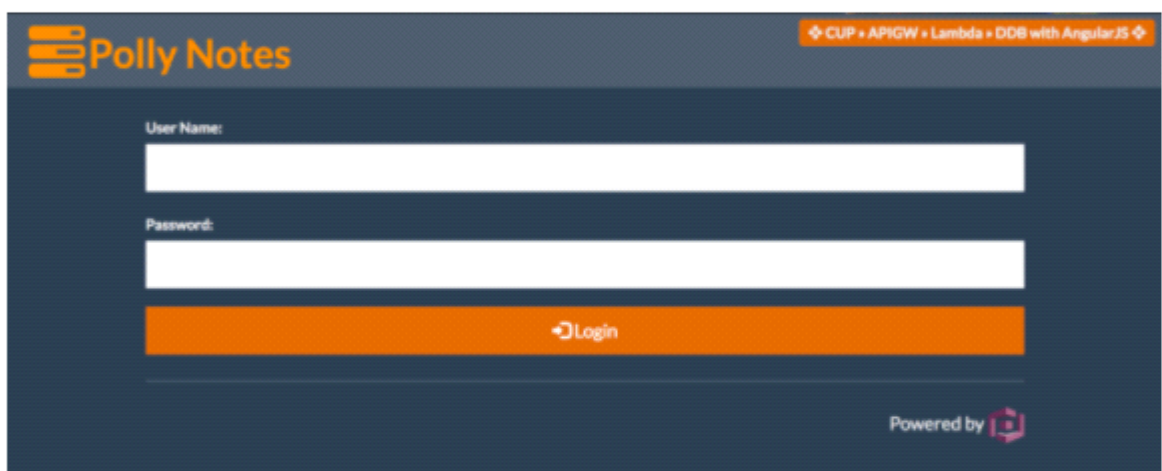
Task 9: Testing the Web Application

In this task, you test the web application by loading your S3 Hosted URL.

Note

You can obtain your S3 Hosted URL by going into the S3 Console. Select your web hosting bucket. Choose the **Properties** tab, and then choose, **Static website hosting**.

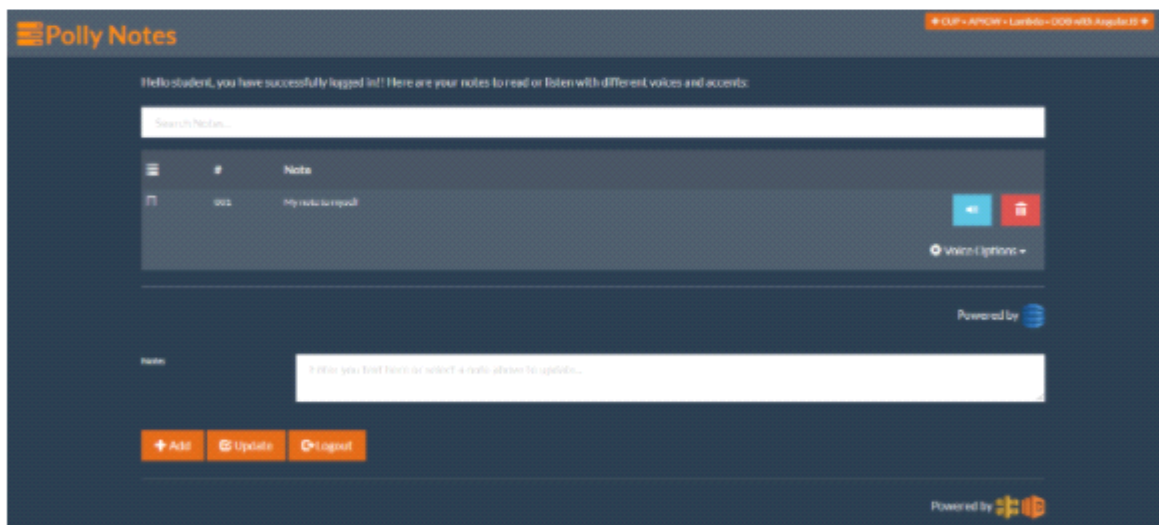
154. In a browser, enter your Static website hosting URL. You should be presented with a login page.



155. Login with the following:

- For **UserName**, enter: `student`
- For **Password**, enter: `student`

156. You should be presented with a page that looks like the following:



You should expect to see **at least one** test note returned. All other functionality, such as adding notes, updating, deleting, or playing will not work. You need to create the API methods and Lambda functions to enable this functionality.

Task 10: Creating the Remaining Lambda Functions

You have two options for the remaining Lambda functions:

You have two options for the remaining Lambda functions:

