

Tutorial: How to use AWS Lambda with S3 for real-time data processing

 examples.javacodegeeks.com/software-development/amazon-aws/tutorial-use-aws-lambda-s3-real-time-data-processing/

A company recently asked me to develop a solution to receive batch data from a third party data vendor, run business logic against the data, and store the resulting values in a database. This demo will show how to build such a process using AWS and Java. We will receive a sample grade book via a batch file in AWS S3, calculate an overall grade for each student in the grade book using an AWS Lambda function and store the results in a DynamoDB table.

1. Introduction

Why use AWS Lambda? Lambda (not to be confused with lambda expressions introduced in Java 8) is a serverless compute platform (Function as a Service) where our Java code will run. As you might expect, there is still a server, however the burden of responsibility has been placed on AWS. This means that provisioning, monitoring, patching, scaling, and other infrastructure activities are managed by AWS. After code is deployed it can be established to run automatically when some external event occurs such as a RESTful call or a new S3 object received. For services that only need to run on occasion and in response to a trigger, Lambda is an excellent solution.

Lambda is an excellent choice compared to cloud compute platforms like AWS EC2 when it isn't necessary to have control of server instances, when code is only going to run in response to another event, or for code that is run infrequently. For example, Netflix uses Lambda to help automate the encoding process of media files.

S3 (Simple Storage Service) is a storage solution for bulk data. Amazon describes it as "secure, durable, highly-scalable cloud storage." Common uses include hosting static websites, big data objects, and holding objects for processing by other AWS services.

DynamoDB is a NoSQL database which is extremely fast, with single-millisecond latency, and delivers consistent read and write executions. Like Lambda, it's fully managed and scalable. It provides an advantageous database solution for simple read and write processes without complex joins.

2. Setup

First you will need to get an AWS account and set up the CLI. Amazon has instructions to do this at <http://docs.aws.amazon.com/lambda/latest/dg/setup.html>

Additionally, we will follow setup using Eclipse IDE and AWS SDK plugin. Amazon has setup instructions here: <http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/getting-started.html>

Once you have set up CLI and your `adminuser` account, log in at <https://signin.aws.amazon.com/console>

2.1 S3

We will create two buckets, one to store the raw data to be processed and a second to store our Java code.

After logging in to the AWS console, select from the top menu:

- AWS > Storage and Content Delivery > S3
- Select the 'Create Bucket' button

Amazon policy allows for names with lowercase letters, numbers, periods (.), and hyphens (-). Your bucket must have

a unique name across all of AWS. Amazon recommends DNS compliant names.

For this tutorial I suggest creating your first bucket with the name `yournames3gradebookexample`. While my examples use US West (Oregon), also referred to as us-west-2, there are a number of considerations to make when choosing a region that we will not explore in this tutorial, including cost, SLA, and latency. For this demo US West (Oregon) will be a good default.

There is no need to copy any settings; choose the 'Create' button on the bottom left.

The screenshot shows the 'Create bucket' wizard in the AWS Management Console. The title bar is blue with the text 'Create bucket' and the 'Java Code Geeks' logo. Below the title bar is a progress bar with four steps: 1. Name and region (active), 2. Set properties, 3. Set permissions, and 4. Review. The main content area is dark blue. Under the heading 'Name and region', there is a 'Bucket name' field with an information icon, containing the text 'yournames3gradebookexample'. Below this is a 'Region' dropdown menu showing 'US West (Oregon)'. Further down is a section titled 'Copy settings from an existing bucket' with a dropdown menu showing 'Select bucket (optional)' and '4 Buckets'. At the bottom of the form are three buttons: 'Create' (blue), 'Cancel' (white), and 'Next' (blue). The footer of the console shows 'Create S3 bucket'.

Repeat the same steps and create a second bucket to store code. Call this `yournamelambda code`. Create this bucket in the same region.

2.2 DynamoDB

Similar to the first steps above, select from the top menu:

- AWS > Database > DynamoDB
- Select 'Create Table'
- For Table Name, enter 'Students'.
- For Partition Key, enter 'StudentID' and set the type to Number.
 - The sort key option is only needed if the partition key can have duplicate entries. We won't need one for our example.
- Leave the 'default settings' option checked.
- Select the 'Create' button at the bottom of the page.

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*

Students

i

Primary key*

Partition key

StudentID

Number

i

☐ Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☒ Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel

Create

Create DynamoDB table

2.3 Lambda


- From Eclipse, choose File > New > Project... > AWS > AWS Lambda Java Project
- Add a project name. I called mine Gradebook. This is only used for your local file system.

- Group ID and Artifact ID are Maven specific. It is ok to use the default if you're unsure what these are for. I used the names `com.zackroppel.lambda` and `gradebook` for these.
- Package name should match a combination of Group ID and Artifact ID, you don't need to change this.
- Call your class `LambdaGradebook`.
- Set Input Type to S3 Event.
- Click Finish.

New AWS Lambda Maven Project

Create a new AWS Lambda Java project

Create a new AWS Lambda Java project in the workspace



Project name:

Group ID:

Artifact ID:

Lambda Function Handler

Each Lambda function must specify a handler class which the service will use as the entry point to begin execution. [Learn more](#) about Lambda Java function handler.

Package Name:

Class Name:

Input Type:

An Amazon S3 trigger that retrieves metadata for the object that has been updated.

Preview:

```
package com.amazonaws.lambda.gradebook;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.S3Event;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.S3Object;


public class LambdaFunctionGradebook implements RequestHandler<S3Event, String> {


    private AmazonS3 s3 = AmazonS3ClientBuilder.standard().build();

    public LambdaFunctionGradebook() {}

}
```

☒ Show README guide after creating the project

 **Java Code Geeks**
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER



Create Lambda function

What you now have is a simple Lambda example which can be deployed to AWS. In your `src/main/java` directory in your project, you'll see your `LambdaGradebook` class. This class implements the AWS `RequestHandler` interface and overrides `handleRequest()`, a method in that interface. As generated this method looks for a file in a given S3 bucket and returns a `String` containing the content type (we will see the input for this shortly).

We will first test to make sure the skeleton setup is complete and allows us to read from S3 in our Lambda function,



and further in the tutorial we will make modifications to this class to calculate grades and store results in DynamoDB.

2.4 IAM Role

You will need to create an IAM role in order for your Lambda function. Doing this sets the access levels that your Lambda function has within AWS. To do this:

- Go to the AWS console > IAM
- Select Roles > Create New Role
- In the dialog for *Select role type*, choose *AWS Lambda* in the *AWS Service Role* category
- Attach Policy > Select boxes for both *AWSLambdaExecute* and *AmazonDynamoDBFullAccess*
- Call your role *lambda-s3-execution-role*
- Create the role.

You have now created a role with full access on DynamoDB resources and the ability to read and write S3 resources.

 **Services** ▾ **Resource Groups** ▾ 


Create role

Step 1 : [Select role type](#)

Step 2 : [Establish trust](#)

Step 3 : [Attach policy](#)



Step 4 : [Set role name and review](#)

 **Java Code Geeks**
JAVA 2 | JAVA DEVELOPERS RESOURCE CENTER

Attach Policy

Select one or more policies to attach. Each role can have up to 10 policies attached.

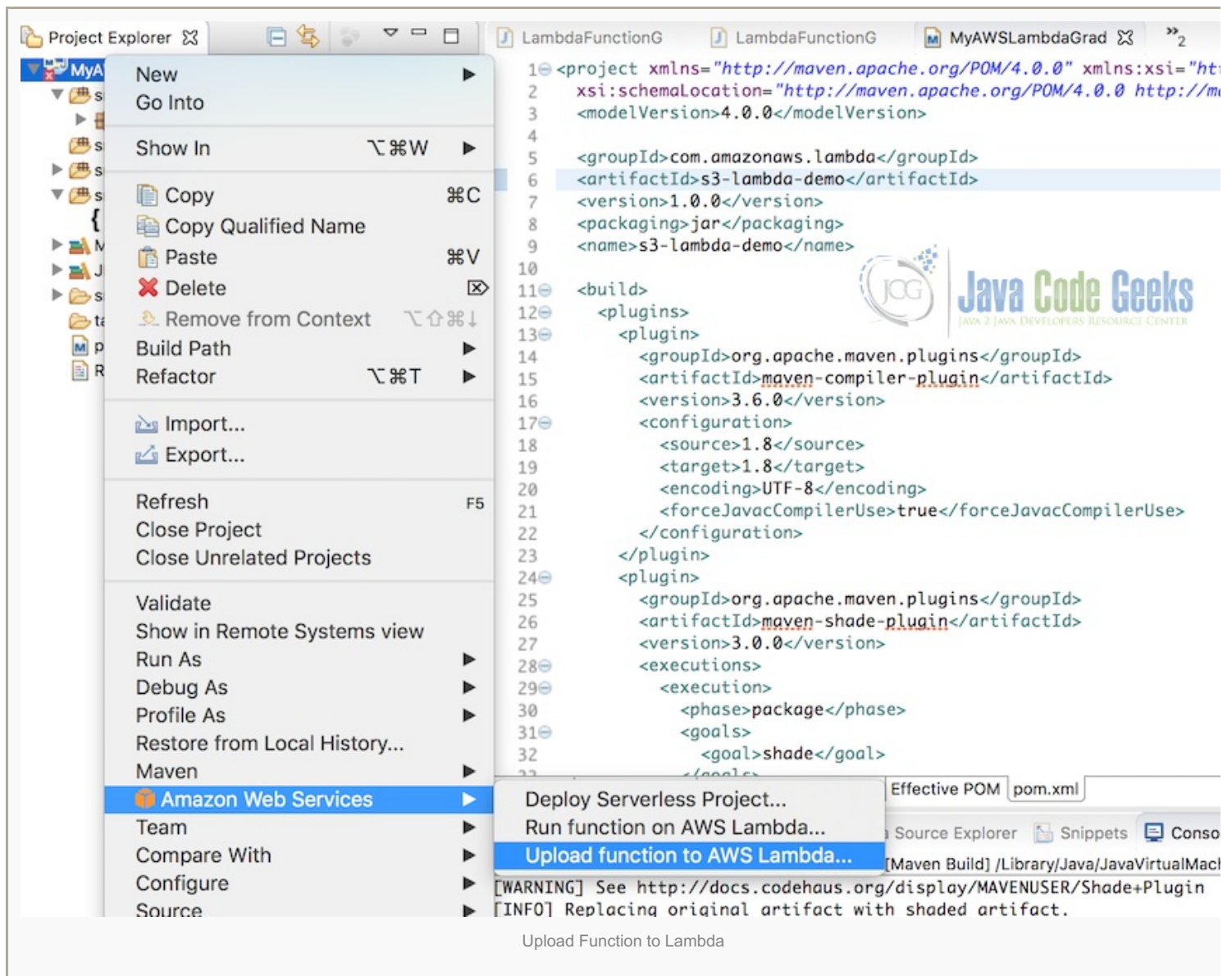
Filter: [Policy Type](#) ▾

	Policy Name ↕	Attached Entities ↕
<input type="checkbox"/>	AWSLambdaBasicExecutionRole...	1
<input checked="" type="checkbox"/>	 AWSLambdaExecute	1
<input type="checkbox"/>	AWSLambdaS3ExecutionRole-b1...	1
<input type="checkbox"/>	AWSLambdaVPCLAccessExecutio...	1
<input type="checkbox"/>	 AWSLambdaBasicExecutionRole	0
<input type="checkbox"/>	AWSLambdaBasicExecutionRole...	0

Attach IAM Policies

2.5 Deploy Lambda code to AWS

In Eclipse, right click your code and select Amazon Web Services > Upload function to Lambda...




In the dialog:

- Make sure the region is consistent with what you chose for S3.
- Create a new Lambda function: `Gradebook`
- Click to the next page
- Description is optional
- Handler should be preselected, leave as is
- Check that your IAM role matches the `lambda-s3-execution-role` set in step 2.4
- Match the S3 bucket with the previously created `yournamelambdacloud`.

Upload Function to AWS Lambda

Function Configuration



Basic Settings

Name:

Gradebook

Description:

The description for the function (optional)

Function Execution

Handler:


com.zackroppel.lambda.gradebook.LambdaGradebook

Select the IAM role that AWS Lambda can assume to execute the function on your behalf. [Learn more](#) about Lambda execution roles.

IAM Role:

lambda-s3-execution-role

Create



S3 Bucket for Function Code

S3 Bucket:

zackslambdacode

Create


Advanced Settings

Memory (MB):

512

Timeout (s):

15



< Back

Next >

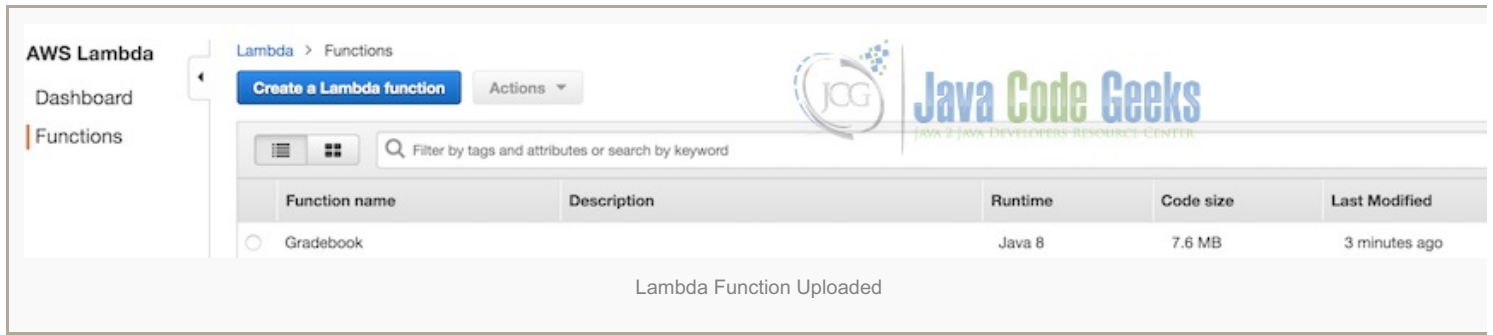
Cancel

Finish

Lambda Function Configuration

Once your function successfully uploads you'll see the name of your lambda function in brackets in your Eclipse project explorer appended to your project name. If you return to your AWS Console for Lambda you'll see the function

just created.



2.6 Input file

We need to create a sample CSV file to work with. In a text editor, create a file with the following two lines

```
1 100,91,88,79,99
```

```
2 101,88,75,90,83
```

Save the file in your local filesystem as `grades.csv` and upload it via the AWS Console at S3
>yournames3gradebookexample > Upload > Add Files > Upload.

2.7 Manual job processing

Now we'll invoke the function manually to see that set up is successful. In the AWS console, go to Lambda > Gradebook > Actions > Configure Test Event

In the dialog that appears, choose a sample event template of S3 Put from the dropdown. Then make the following edits as shown in the sample below:

Use the editor below to enter an event to test your function with. You can edit the event again by choosing an event in the Actions list. Note that changes to the event will only be saved locally.

Sample event template: S3 Put

```

12  "eTag": "0123456789abcdef0123456789abcdef",
13  "sequencer": "0A1B2C3D4E5F678901",
14  "key": "grades.csv",
15  "size": 1024
16  },
17  "bucket": {
18    "arn": "arn:aws:s3:::zackroppels3gradebookexample",
19    "name": "zackroppels3gradebookexample",
20    "ownerIdentity": {
21      "principalId": "EXAMPLE"
22    }
23  },
24  "s3SchemaVersion": "1.0"
25  },
26  "responseElements": {
27    "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmbdaisawesome/mnop
28    "x-amz-request-id": "EXAMPLE123456789"
29  },
30  "awsRegion": "us-west-2",
31  "eventName": "ObjectCreated:Put",
32  "userIdentity": {
33    "principalId": "EXAMPLE"
34  },
35  "eventSource": "aws:s3"
36  }
37

```

Test Configuration

- In the lines for bucket ARN and bucket name, update the sample bucket with your first S3 bucket `yournames3gradebookexample`.
- Update your S3 key to `grades.csv`.
- Make sure your AWS region matches as previously used.
- Click 'Save and test'.

The application will run and you will see a successful result which logs the content type as "text/csv" on the second line of the log output.

Execution result: succeeded (logs)

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

"text/csv"

Summary

Code SHA-256	x20vk+HmyL/kh6fpEETYo/iD4HciUfzYz zOe1UgGhCo=
Request ID	f4200742-49b4-11e7-b491- 8143e21fd4e7
Duration	3369.72 ms

Log output

The area below shows the logging calls in your code. These correspond to a single row within corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: f4200742-49b4-11e7-b491-8143e21fd4e7 Version: $LATEST
Received event: com.amazonaws.services.lambda.runtime.events.S3Event@7d7758beEND
REPORT RequestId: f4200742-49b4-11e7-b491-8143e21fd4e7 Duration: 3369.72 ms
```

Successful Test

2.8 Gradebook logic

We're going to modify our code to calculate grades and store results in a DynamoDB table. The code will be updated like the following:

```
01 package com.yourname.lambda.gradebook;
02
03 import java.io.BufferedReader;
04 import java.io.IOException;
05 import java.io.InputStreamReader;
06
07 import com.amazonaws.regions.Regions;
08 import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
09 import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
10 import com.amazonaws.services.dynamodbv2.document.DynamoDB;
11 import com.amazonaws.services.dynamodbv2.document.Item;
12 import com.amazonaws.services.dynamodbv2.document.Table;
```

```
13 import com.amazonaws.services.lambda.runtime.Context;
14 import com.amazonaws.services.lambda.runtime.RequestHandler;
15 import com.amazonaws.services.lambda.runtime.events.S3Event;
16 import com.amazonaws.services.s3.AmazonS3;
17 import com.amazonaws.services.s3.AmazonS3ClientBuilder;
18 import com.amazonaws.services.s3.model.GetObjectRequest;
19 import com.amazonaws.services.s3.model.S3Object;
20
21 RequestHandler<S3Event, String>
    public class LambdaGradebookimplements {
22
23     AmazonS3 s3 =
        private AmazonS3ClientBuilder.standard().build();
24
25     private
        AmazonDynamoDB client =
        AmazonDynamoDBClientBuilder.standard().withRegion(Regions.US_WEST_2).build();
26
27     DynamoDB dynamoDB
        private = new DynamoDB(client);
28
29     String tableName
        private static = "Students";
30
31     LambdaGradebook()
        public {
32
33     }
34 }
```

```
31


---


32


---


33     LambdaGradebook(AmazonS3 s3)
        {


---


34         .s3 =
            thiss3;


---


35     }


---


36


---


37     @Override


---


38     String handleRequest(S3Event event, Context context)
        public {


---


39         "Received event: " +
            context.getLogger().log(" " + event);


---


40


---


41


---


42         String bucket =
            event.getRecords().get(0).getS3().getBucket().getName();


---


43         String key =
            event.getRecords().get(0).getS3().getObject().getKey();


---


44         try {


---


45             S3Object response =
                GetObjectRequest(bucket,
                s3.getObject(
                new key));


---


46             String contentType =
                response.getObjectMetadata().getContentType();


---


47
```



```

48         BufferedReader br
           = new BufferedReader(new
InputStreamReader(response.getObjectContent()));

```

```

49

```

```

50         String
           csvOutput;

```

```

51         ((csvOutput = br.readLine())
           != null)
           while != null{

```

```

52         String[] str =
           csvOutput.split(
           ",");

```

```

53         int total
           = 0;

```

```

54         int average
           = 0;

```

```

55         for (int i = 0; i < str.length; i++)
           for (int j = 0; j < str[i].length(); j++)

```

```

56             total +=
               Integer.valueOf(str[i].charAt(j));

```

```

57         }

```

```

58         average = total / (str.length
           - 1);

```

```

59         createDynamoItem(Integer.valueOf(str[0].charAt(average)),
           createDynamoItem(Integer.valueOf(str[0].charAt(average)),

```

```

60         }

```

```

61         return contentType;

```

```

62         } catch (IOException e)
           } catch {

```

```

63         e.printStackTrace();

```

```

64         context.getLogger().log(String.format(
            "Error getting object %s from bucket %s. Make sure they exist
            and"

65         " your bucket is in the same region as this
            +function."

            , bucket,
            key));

66         return e.toString();

67     }

68

69     }

70

71         grade)
        private void createDynamoItem(int studentId,int {

72

73         Table table =
            dynamoDB.getTable(tableName);

74         try {

75

76             Item item
                = new Item().withPrimaryKey("StudentID"studentId).withInt(
                "Grade"grade);

77         table.putItem(item);

78

79         (Exception e)
        }catch {

80         "Create item
            System.err.println(failed."
            );

```

```
81         System.err.println(e.getMessage());
```

```
82
```

```
83     }
```

```
84 }
```

```
85 }
```

The logic now computes an average student grade and calls the method `createDynamoItem()` to add the student ID and grade to the Students table.

Repeat the steps in 2.5 using the existing Lambda function `Gradebook`. Return to the AWS console in your browser. Go to `Lambda > Functions > Gradebook > Test`. The test configuration previously used will be run again.

In the AWS console, go to `DynamoDB > Tables > Students > Items`. You should see student records as shown below:

Students

Close

Overview

Items

Metrics

Alarms

Capacity

Indexes

Create item

Actions

JCG

Java Code Geeks

JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Scan: [Table] Students: StudentID

Scan

[Table] Students: StudentID

+ Add filter

Start search

	StudentID	Grade
<input type="checkbox"/>	100	89
<input type="checkbox"/>	101	84

DynamoDB Students

2.9 Automation

Now that we have a working pipeline, we want to automate processing of data when new CSV files are received. Run the following CLI command in your terminal, replacing the values in parentheses:

```
1 aws lambda add-permission \
```

```
2 --function-name  
  Gradebook \
```

```
3 --region (us-west-2)
  \


---


4 --statement-id (gradebook-unique-id)
  \


---


5 --action lambda:InvokeFunction
  \


---


6 --principal s3.amazonaws.com
  \


---


7 --source-arn arn:aws:s3:::(yournamelambdacode)
  \


---


8 --source-account (bucket-owner-account-id)
  \


---


9 --profile adminuser
```

- Check that your region matches
- Statement ID can be anything you want and does not necessarily need to be changed
- Append your source bucket to the end of source-arn
- Append your account ID to the source-account, removing any dashes.
 - Your account ID is a 12 digit number which appears on the top right side when you log in to the browser console. Enter it without dashes.

Repeat step 2.6 with a new .csv file that has different student IDs (I added 103 and 104 to mine).

Verify in your DynamoDB console that your new records are in the table:

Students

Close

Overview

Items

Metrics


Alarms

Capacity

Indexes

Create item

Actions

 **Java Code Geeks**
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Scan: [Table] Students: StudentID

Scan

[Table] Students: StudentID

+ Add filter

Start search

	StudentID	Grade
<input type="checkbox"/>	104	81
<input type="checkbox"/>	100	89
<input type="checkbox"/>	103	80
<input type="checkbox"/>	101	84

Students DynamoDB table with 4 values

3. Summary

We have set up an AWS environment from scratch that takes batch files, applies business logic to them, and saves the results in a NoSQL database. We were able to do this without having any responsibility on the infrastructure. We were able to deploy our code automatically using the AWS Toolkit for Eclipse. The result is a highly available and scalable solution that runs automatically as needed and allows us to focus on our code.

4. Download The Source Code

Download

You can download the full source code of this example here: [Gradebook](#)