Table Of Contents

|  |  |  |
| --- | --- | --- |
| **S.No** | **Contents** | **Page No** |
| 1 | Set UP AWS Environment | 2 |
| 2 | Getting Started with Lambda Functions | 4 |
| 3 | Getting Started with DynamoDB | 11 |
| 4 | Working On Dynamo DB CURD Operations. | 16 |
|  |  |  |

1. ***Set -Up AWS Environment***

In Order to Work with AWS project in Eclipse we need to install “AWS Took Kit “ plugin ,for that Open this below path

Eclipse Work space > help > Eclipse Market Place > Find “AWS Tool kit for Eclipse 2.0.” and then Install.

**Set up AWS Credentials**

To access Amazon Web Services with the AWS Toolkit for Eclipse, you must configure the AWS Toolkit for Eclipse with AWS account credentials.

**Get your AWS access keys**[**#**](http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/setup-credentials.html#get-your-aws-access-keys)

Access keys consist of an *access key ID* and *secret access key*, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the [AWS Management Console](https://console.aws.amazon.com/console/home). We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

**To get your access key ID and secret access key**

1. Open the [IAM console](https://console.aws.amazon.com/iam/home).
2. From the navigation menu, click **Users**.
3. Select your IAM user name.
4. Click **User Actions**, and then click **Manage Access Keys**.
5. Click **Create Access Key**.

Your keys will look something like this:

* + Access key ID example: AKIAIOSFODNN7EXAMPLE
  + Secret access key example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

1. Click **Download Credentials**, and store the keys in a secure location.

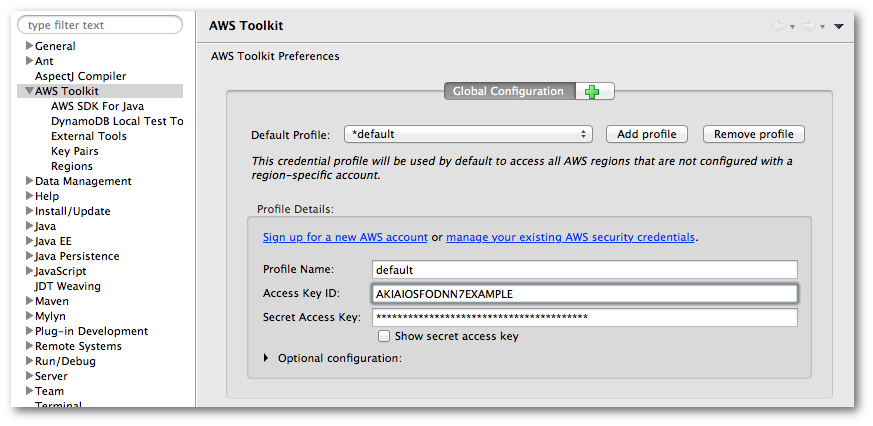
**NOTE**

Your secret key will no longer be available through the AWS Management Console; you will have the only copy. Keep it confidential in order to protect your account, and never email it. Do not share it outside your organization.

**To add your access keys to the AWS Toolkit for Eclipse**

1. Open Eclipse's **Preferences** dialog box and click **AWS Toolkit** in the sidebar.
2. Type or paste your AWS access key ID in the **Access Key ID** box.
3. Type or paste your AWS secret access key in the **Secret Access Key** box.
4. Click **Apply** or **OK** to store your access key information.

Here's an example of a configured set of default credentials:



***2. Getting Started with Lambda Functions***

This tutorial will guide you through the process of a typical AWS Lambda workflow and provide you with first-hand experience in using Lambda with the AWS Toolkit for Eclipse.

## *2.1 Create a Lambda handler class*

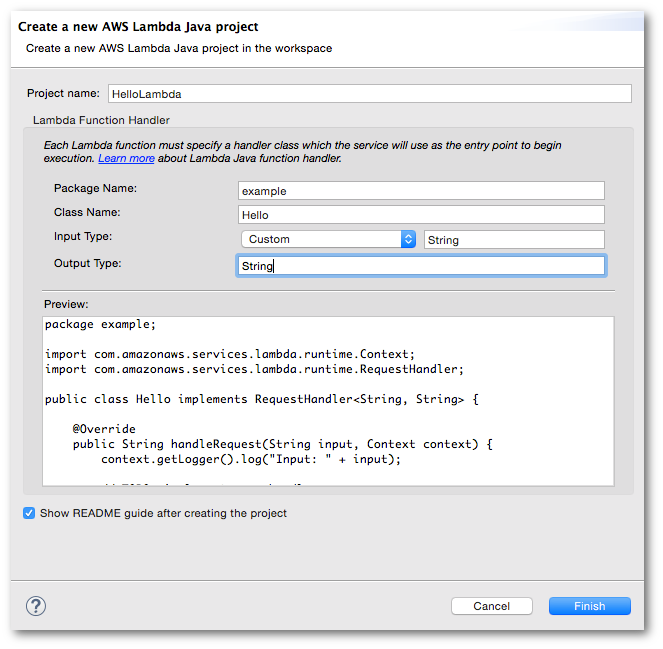
First, you will implement the code as a method in a handler class. The AWS Toolkit for Eclipse provides a new project wizard to help you create a new handler class.

**To create an Lambda handler class**

1. On the Eclipse toolbar, open the drop-down Amazon Web Services menu (identified by the AWS icon) and select  **New AWS Lambda Java project...**
2. Add your Java *project name*, *package name*, and *class name* in the associated input boxes. You can choose any valid names that you want. This tutorial will use the following sample values:
   * **Project name**: *HelloLambda*
   * **Package name**: *example*
   * **Class name**: *Hello*

While you type, the code in the **Source preview** will change to reflect the changes you make in the dialog.

1. For **Input Type**, choose *Custom*. For information about each of the available input types, see [New AWS Lambda Java Project Dialog](http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/lambda-ref-create-project.html).
2. The second **Input Type** field is name of the actual Java type that will be returned, which must be a Java class (not a primitive type such as float, int or boolean). It defaults to Object. Change this value to String.
3. Change the value of **Output Type** to String, as well.
4. Verify that your entries look like the following screenshot (modify them if they are not), and then click **Finish**.



1. Once you click **Finish**, your project's directory and source files will be generated in your eclipse workspace, and a new web browser window will appear, displayingREADME.html (which was created for you in your project's root directory). README.html provides instructions to help guide you through the next steps of implementing, testing, uploading and invoking your new Lambda function. Read through it to gain some familiarity with the next steps that will be covered here.

Next, we'll implement the function in the HelloLambda Java project that was just created for you in Eclipse.

***2.2 Implement the handler method***

The **Create New Project** dialog creates a skeleton project for you, but it's up to you to fill in the code that will be run when your Lambda function is invoked (in this case, by a custom event that sends a String to your function, as you specified when setting your method's input parameter.

**To implement your Lambda handler method**

1. Using Eclipse's **Project Explorer**, open Hello.java in the **HelloLambda** project. It will contain code similar to the following:

**package** **example**;

**import** **com.amazonaws.services.lambda.runtime.Context**;

**import** **com.amazonaws.services.lambda.runtime.RequestHandler**;

**public** **class** **Hello** **implements** RequestHandler<String, String> {

**@Override**

**public** String handleRequest(String input, Context context) {

context.getLogger().log("Input: " + input);

*// TODO: implement your handler*

**return** **null**;

}

}

1. Replace the contents of the handleRequest function with the following custom code .

**@Override**

**public** String handleRequest(String input, Context context) {

String output = "Hello, " + input + "!";

**return** output;

}

Your Hello.java file should now contain:

**package** **example**;

**import** **com.amazonaws.services.lambda.runtime.Context**;

**import** **com.amazonaws.services.lambda.runtime.RequestHandler**;

**public** **class** **Hello** **implements** RequestHandler<String, String> {

**@Override**

**public** String handleRequest(String input, Context context) {

String output = "Hello, " + input + "!";

**return** output;

}

}

**Allow Lambda to assume an IAM role**

In order for Lambda to access your Lambda function, you will need to create an IAM role that gives it access to your AWS resources. The easiest way to do this is with the AWS Management Console.

**To create an IAM role for Lambda**

1. Sign in to the [AWS Management Console](https://console.aws.amazon.com/console/home).
2. Open the [IAM console](https://console.aws.amazon.com/iam/home).
3. Select **Roles** on the sidebar, then **Create New Role**.
4. Add a name for your role, such as hello-lambda-role, and click **Next Step**.
5. On the **Select Role Type** page, select *AWS Lambda* within the **AWS Service Roles** list.
6. For **Attach Policy**, check *AmazonS3FullAccess*, which allows Lambda to access your Amazon S3 resources, and then click **Next Step** to continue.

**Note**

Amazon S3 is required because Lambda will upload your code to an Amazon S3 bucket when you deploy and run your Lambda function. You can use a bucket that you create (this is covered in the next section) or use an existing bucket.

1. Review your role parameters, then click **Create Role** to finish creating the IAM role.

**Create an Amazon S3 bucket for your Lambda code**[**#**](http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/lambda-tutorial.html#lambda-tutorial-create-bucket)

AWS Lambda requires an Amazon S3 bucket to store your Java project when you upload it. You can either use a bucket that already exists in the AWS region in which you'll run your code, or you can create a new one specifically for use by Lambda (recommended).

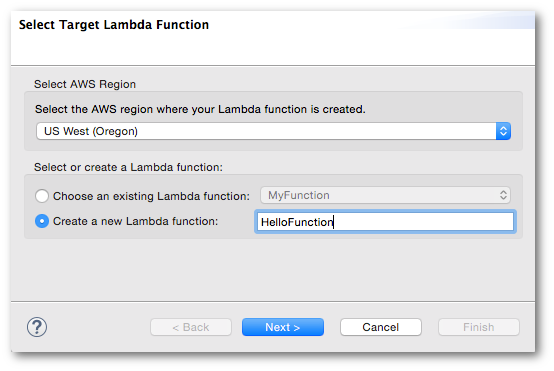
**To create an Amazon S3 bucket for use with Lambda**

1. Log in to AWS and go to the [S3 console](https://console.aws.amazon.com/s3/home).
2. Click **Create Bucke**t.
3. Enter a bucket name and select an [AWS region](https://docs.aws.amazon.com/general/latest/gr/glos-chap.html#region) for your bucket. This region should be the same one in which you intend to run your Lambda function. For a list of regions supported by Lambda see the [Regions and Endpoints](https://docs.aws.amazon.com/general/latest/gr/rande.html#lambda_region) topic in the *Amazon Web Services General Reference*.
4. Click **Create** to finish creating your bucket.

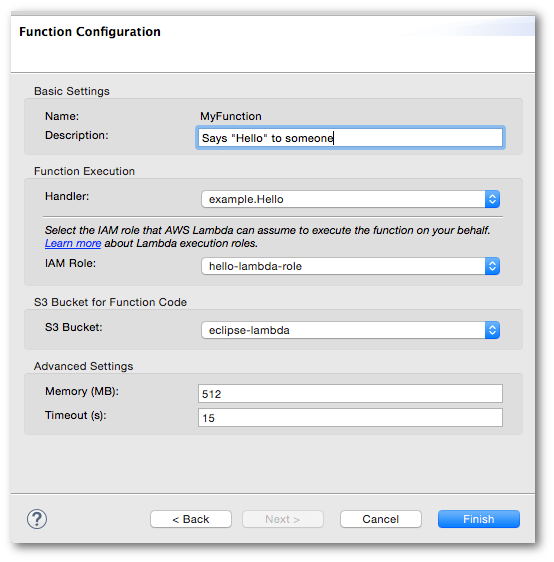
**Upload your Function to Lambda**

Next, we'll upload your code to AWS Lambda in preparation for invoking it using the AWS Management Console.

1. Right-click in your code window and select **AWS Lambda**, then **Upload function to AWS Lambda.**
2. In the **Select Target Lambda Function** dialog that appears, select the AWS region to use. This should be the same region that you chose for your [Amazon S3 bucket](http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/lambda-tutorial.html#lambda-tutorial-create-bucket).



1. Select **Create a new Lambda function** and enter the name of your function (such as **HelloFunction**).
2. Click **Next** to proceed to **Function Configuration**.
3. Enter a description for your target Lambda function. You can leave the rest of the options as they are; the AWS Toolkit for Eclipse chooses default values for you.



1. Click **Finish** to upload your Lambda function to AWS. If the upload succeeds, you will see the Lambda function name that you chose appear next to your Eclipse project name in the **Project Explorer** view:

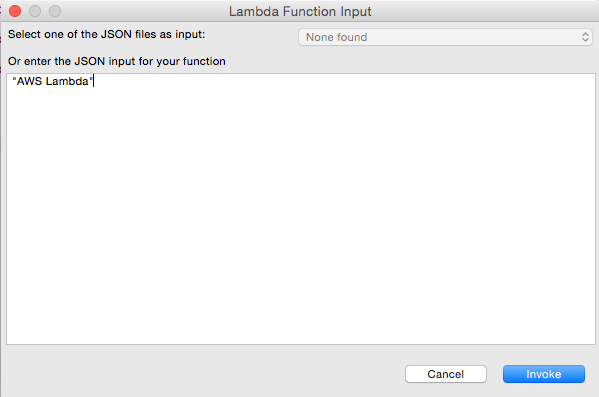
C:\Users\NK00484143\Desktop\lambda_tutorial_upload_function_success.png

If you don't see this happen, you should open Eclipse's **Error Log** view. Lambda will write information about failures to upload or run your function to the error log for further debugging.

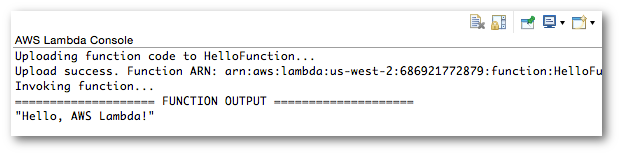
**Invoke the Lambda function**[**#**](http://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/lambda-tutorial.html#lambda-tutorial-invoke-function)

You can now invoke the function on AWS Lambda.

1. Right-click in your code window and select **AWS Lambda**, then **Run on AWS Lambda**.
2. In the input box, type a valid JSON string, such as "AWS Lambda".



1. Click **Invoke** and it will send your input data to your Lambda function. If you have set up everything correctly, you should see the return value of your function printed out in the Eclipse **Console** view (which will automatically appear if it isn't already shown).



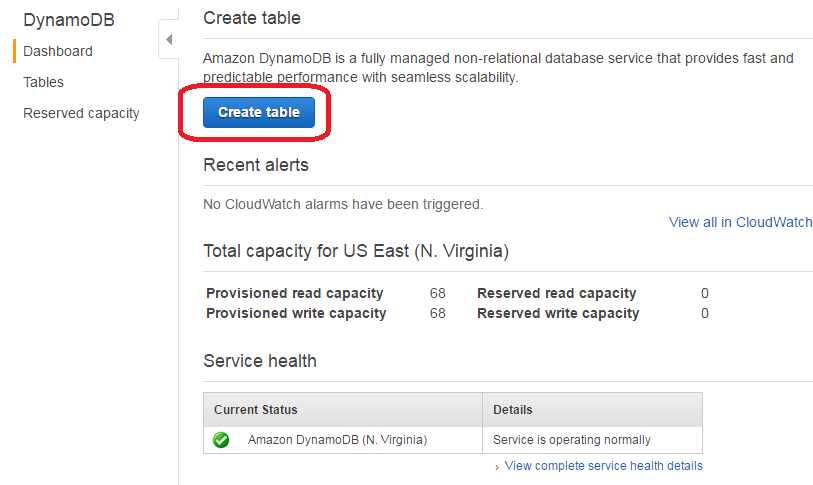
Congratulations, you've just run your first Lambda function directly from the Eclipse IDE!

1. ***Getting Started with DynamoDB***

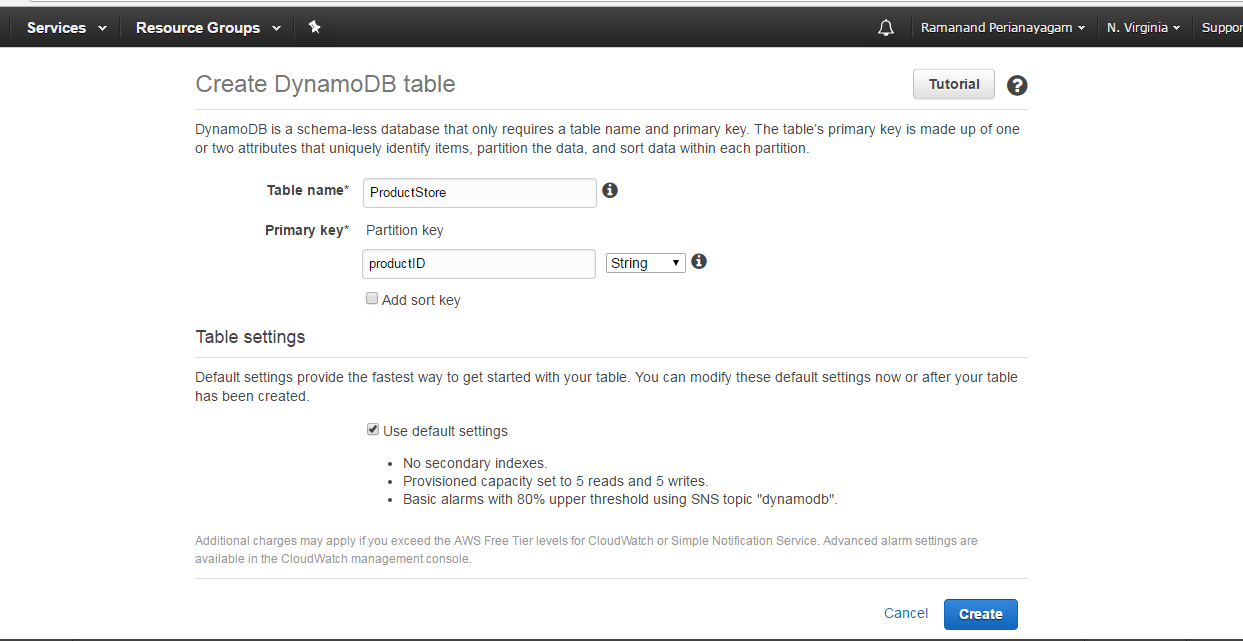
Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance. The AWS Toolkit for Eclipse provides functionality for working with DynamoDB in a development context.

1. Login to the AWS Account and Selectthe **“Dynamo DB “** service in the AWS Services Console .

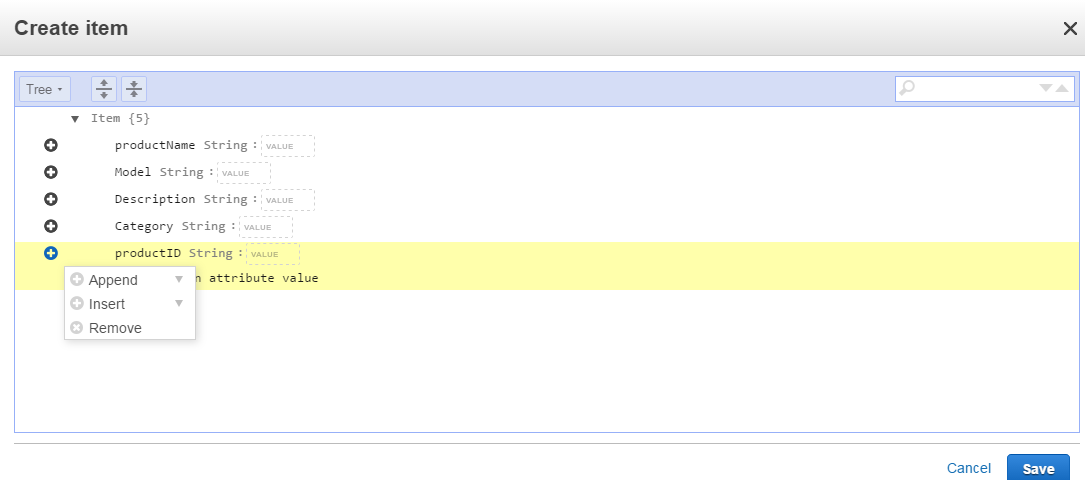
2. Click on Create table button.



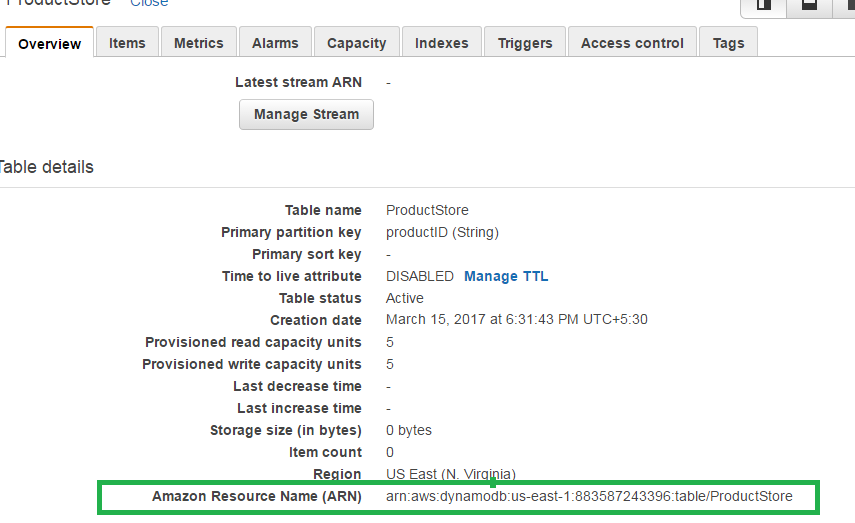
3. Provide Table name and Primary Key.



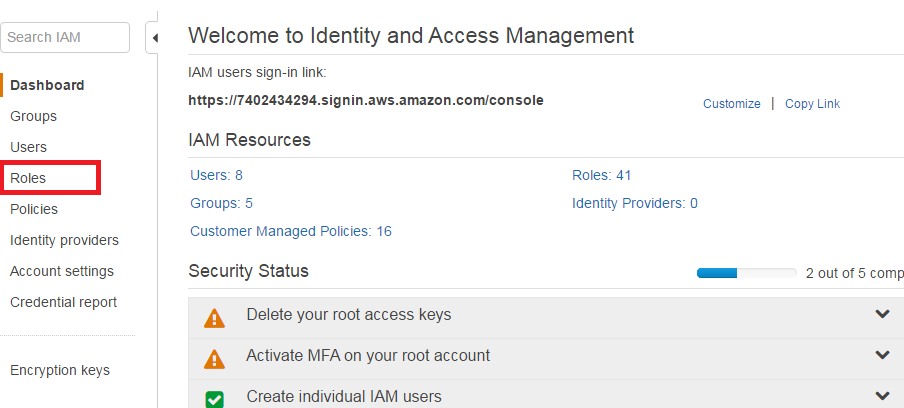
4. Select Items tab click on Create Item and provide the attributes with daya type and finally save the item.



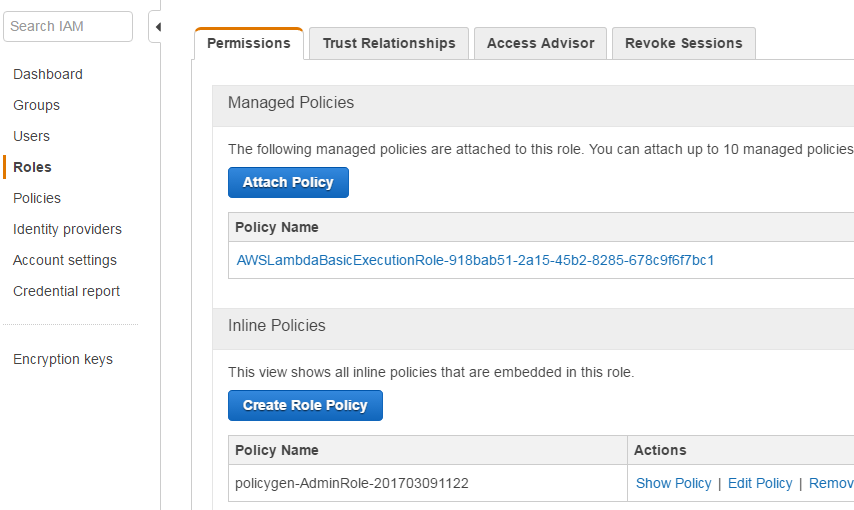
5. New Item will be reflected the Items section, click on Overview tab and selects ARN.



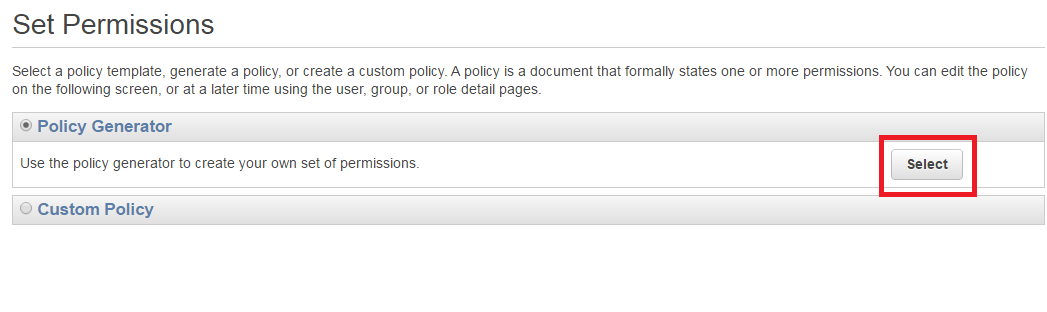
6. We are supposed to attach this ARN to some role,so that he is able to acess the table and allowed to perform selected Operations.

7. Select IAM service in the AWS Console and select “Roles”.

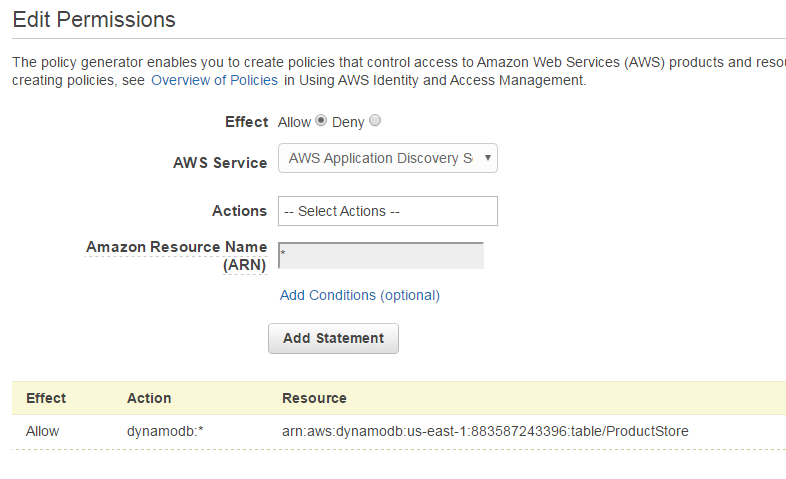
8. Select In line Policies and Create Role Policy .



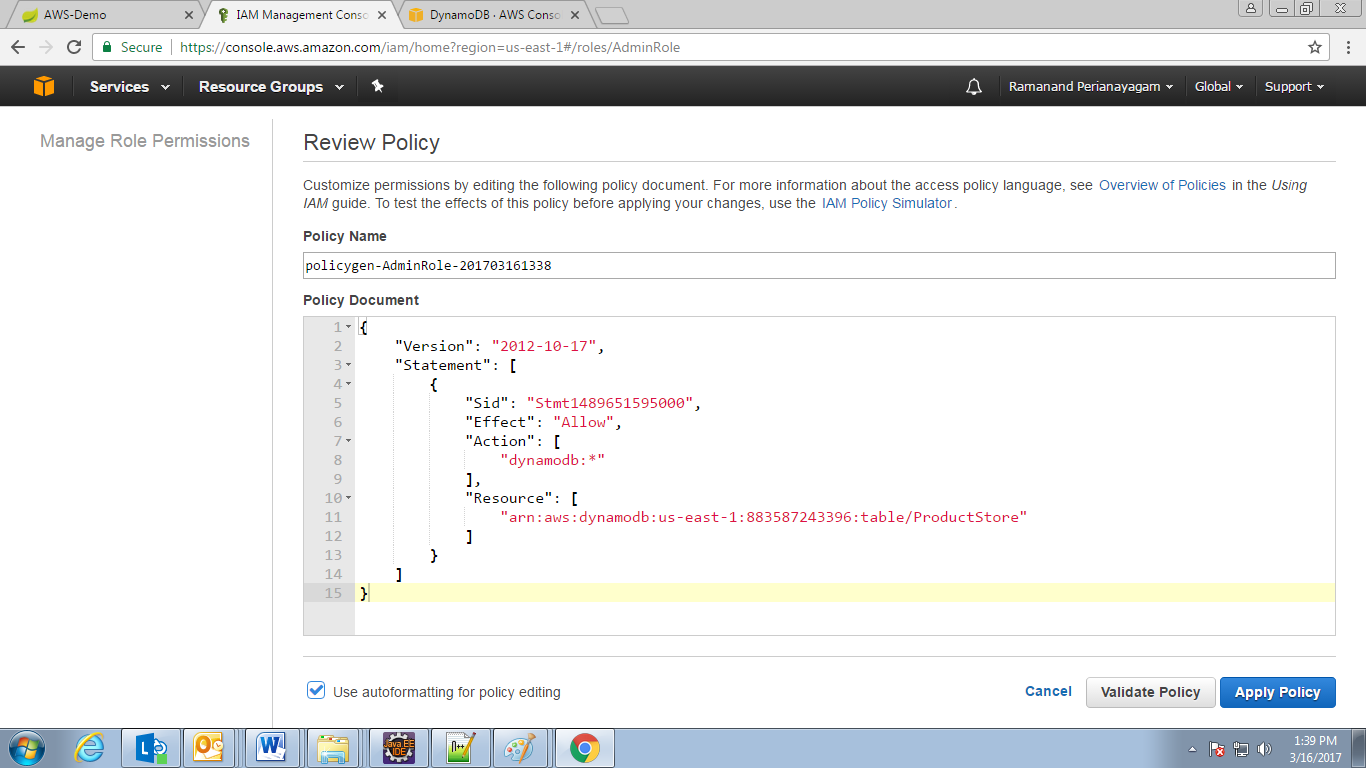
9. Select Policy Generator and click on Select button



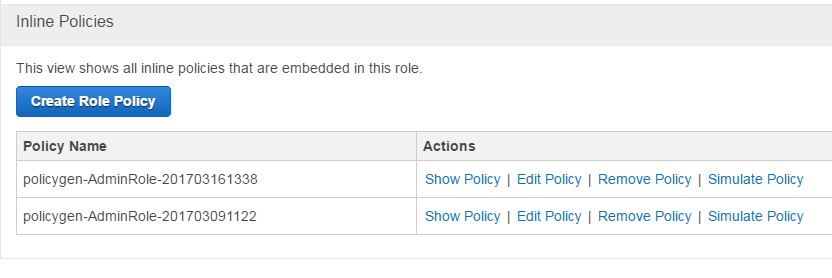
10. Select Allow option and select “AWS Dynamo DB” Option for the AWS Service and check all Action Provide the ARN which is copied from the Dynamo Db overview Section and Click on “Add Statement.”



11. Click on Apply Policy.



13. Now we are able to view new policy created, so that the particular role user can be able to do all selected operations on this Table.



Once this step is done, we are able to perform Operations on this table. Let’s Explore Some of the Curd Operations on this Dynamo Table.

1. ***Working On Dynamo DB CURD Operations.***

As Discussed, in the Section #2 (*Getting Started with Lambda Functions*) Create a Lambda java Project.

In the Custom Handler Method replace the Code with below content.

***1.Save Operation :***

This below Code will insert new product record in to the ***ProductStore***Table.

*public String handleRequest(ProductRequest inputRequest, Context context) {*

**private** PutItemOutcome persistData(ProductRequest *inputRequest*) {

AmazonDynamoDBClient client = **new** AmazonDynamoDBClient();

client.setRegion(Region.getRegion(REGION));

**this**.dynamoDb = **new** DynamoDB(client);

**return** **this**.dynamoDb.getTable(“ProductStore”)

.putItem(**new** PutItemSpec().withItem(**new** Item()

.withString("productID", productRequest.getProductID())

.withString("productName", productRequest.getProductName())

.withString("Category", productRequest.getCategory())

.withString("productModel", productRequest.getModel())

.withInt("Quantity",productRequest.getQuantity())

.withLong("Price",productRequest.getPrice())));

}

}

**Invoke the Lambda function :**

1. Right-click in your code window and select **AWS Lambda**, then **Run on AWS Lambda**.
2. In the input box, type a valid JSON string.

{

      "price": "65000",

     "productName": "Apple",

      "productID": "APPLE123",

      "category": "Smart phones",

        "model": "2017",

      "quantity":"1000"

    }

1. Click **Invoke** and it will send your input data to your Lambda function. If you have set up everything correctly, you should see the return value of your function printed out in the Eclipse **Console** view.

***2.Search Operation :***

This below Code will search existing product with primary key from the ***ProductStore***Table. In the Custom Handler Method replace the Code with below code.

**public** ScanResult handleRequest(String productId,Context context) {

ScanResult scanResult = **null** ;

AmazonDynamoDBClient dynamoDB = **new** ~~AmazonDynamoDBClient~~();

Region usWest2 = Region.*getRegion*(Regions.***US\_EAST\_1***);

dynamoDB.~~setRegion~~(usWest2);

HashMap<String, Condition> scanFilter = **new** HashMap<String, Condition>();

Condition condition = **new** Condition()

.withComparisonOperator(ComparisonOperator.***EQ***.toString())

.withAttributeValueList(**new** AttributeValue().withS(productId));

scanFilter.put("productID", condition);

ScanRequest scanRequest = **new** ScanRequest(“ProductStore”).withScanFilter(scanFilter);

scanResult = dynamoDB.scan(scanRequest);

}

**Invoke the Lambda function :**

1. Right-click in your code window and select **AWS Lambda**, then **Run on AWS Lambda**.
2. In the input box, type a valid JSON string.

{

      "productID": "APPLE123"

    }

1. Click **Invoke** and it will send your input data to your Lambda function. If you have set up everything correctly, you should see the return value of your function printed out in the Eclipse **Console** view.

***2.Delete Operation :***

This below Code will delete existing product with primary key from the ***ProductStore***Table.In the Custom Handler Method replace the Code with below code.

**public** DeleteItemOutcome handleRequest(String productId, Context context) {

AmazonDynamoDBClient client = **new** ~~AmazonDynamoDBClient~~();

client.~~setRegion~~(Region.*getRegion*(REGION));

**this**.dynamoDb = **new** DynamoDB(client);

Table table = dynamoDb.getTable(“ProductStore”);

DeleteItemOutcome outcome = table.deleteItem("**productID**", productId);

**return** outcome;

}

**Invoke the Lambda function :**

1. Right-click in your code window and select **AWS Lambda**, then **Run on AWS Lambda**.
2. In the input box, type a valid JSON string.

{

      "productID": "APPLE123"

    }

1. Click **Invoke** and it will send your input data to your Lambda function. If you have set up everything correctly, you should see the return value of your function printed out in the Eclipse **Console** view.