Serverless Web Apps using Amazon DynamoDB - Part 2

Overview

Continue the mission! In this lab, you will continue building a mission dossier generator using DynamoDB. This is Part 2 of a three-part series of labs. You already created a DynamoDB table and set up IAM in Part 1. In Part 2, this lab, you will create and test Lambda functions that interact with DynamoDB and retrieve data in several different ways. You will then test your new functions. In Part 3, you will publish the app, utilizing all of the DynamoDB skills you learn through the three labs, and send your team of super heroes off to save the world!

Topics covered

By the end of this lab you will be able to:

- Create Lambda functions that retrieve data in a DynamoDB table in two different ways
- Test Lambda functions with a template
- Configure custom tests for Lambda functions

Prerequisites

You should familiarize yourself with key concepts by taking the **Introduction to Amazon DynamoDB** lab.

This is the second in a three-part series of labs. Each of the three labs can stand alone, but because the labs build upon what you learned previously, you should take the **Serverless Web Apps using Amazon DynamoDB - Part 1** prior to taking this lab.

Other AWS Services

Other AWS Services than the ones needed for this lab are disabled by an IAM policy during your access time in this lab. In addition, the capabilities of the services used in this lab are limited to what's required by the lab and in some cases are even further limited as an intentional aspect of the lab design. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad tech, IoT, and many other applications.

You can use Amazon DynamoDB to create a database table that can store and retrieve any amount of data, and serve any level of request traffic. Amazon DynamoDB automatically spreads the data and traffic for the table over a sufficient number of servers to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance.

DynamoDB Terminology

Tables

Similar to other database management systems, DynamoDB stores data in tables. A table is a collection of data. For example, in this lab you will work with a table named SuperMission, where mission information is stored.

Items
Each table contains multiple items. An item is a group of attributes that is uniquely identifiable among all of the other items. In the SuperMission table, each item represents Mission information.

Attributes
Each item is composed of one or more attributes. An attribute is a fundamental data element, something that does not need to be broken down any further. In the SuperMission table, items have attributes like SuperHero, MissionStatus, Villain1, etc. Attributes in DynamoDB are similar in many ways to fields or columns in other database management systems.

Primary Key
When you create a table, in addition to the table name, you must specify the primary key of the table. As in other databases, a primary key in DynamoDB uniquely identifies each item in the table, so that no two items can have the same key. When you add, update, or delete an item in the table, you must specify the primary key attribute values for that item. The key values are required; you cannot omit them. DynamoDB supports two different kinds of primary keys: **Partition Key** and **Partition Key and Sort Key**.

Secondary Indexes
In DynamoDB, you can read data in a table by providing primary key attribute values. If you want to read the data using non-key attributes, you can use a secondary index to do this. After you create a secondary index on a table, you can read data from the index in much the same way as you do from the table. By using secondary indexes, your applications can use many different query patterns, in addition to accessing the data by primary key values.

AWS Lambda

AWS Lambda is a compute service that provides resizable compute capacity in the cloud to make w eb-scale computing easier for developers. Upload your code to AWS Lambda and AWS Lambda takes care of provisioning and managing the servers that you use to run the code. AWS Lambda supports multiple coding languages: Node.js, Java, or Python.\
You can use AWS Lambda in tw o w ays:

- As an event-driven compute service w here AWS Lambda runs your code in response to events, such as uploading image files as you'll see in this lab.
- As a compute service to run your code in response to HTTP requests using Amazon API Gatew ay or API calls.

AWS Lambda passes on the financial benefits of Amazon's scale to you. Lambda executes your code only w hen needed and scales automatically, from a few requests per day to thousands per second. Lambda makes it easy to build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Amazon Kinesis, or create your ow n back end that operates at AWS scale, performance, and security.

Start Lab

- Open https://808477742599.signin.aw s.amazon.com/console
- Enter login credentials

Task 1: Check CloudFormation Status

3. In the **AWS Management Console**, on the Services menu, click **CloudFormation**.

4. You should see one stack w ith a status of **CREATE_IN_PROGRESS**.

5. Wait for the status to change to **CREATE_COMPLETE** before you proceed.

You can click the **refresh** button to refresh the screen.

Tip If you do not see a stack being created, double check that you are in the "N. Virginia" region.

The CloudFormation template is preparing w hat you need to complete this lab. It may take a few minutes for all the resources to be ready. While you w ait, you can observe the creation process for the follow ing resources.

Task 2: Create Your Lambda Functions

In this task, you w ill create tw o Lambda functions. The first w ill retrieve a list of super heroes that are stored in the **SuperMission** DynamoDB table. The second w ill retrieve the mission details from the same DynamoDB table.

6. On the Services menu, click **Lambda**.

7. Click **Create function** Then configure:

- **Name:** `getheroeslist`
- **Runtime:** *Node.js 8.10*
- **Existing role:** *SuperDynamoDBScanRole*
- Click **Create function**

6. In the **Function code** section, remove all of the code in *index.js*

7. Within the **index.js** tab, paste the code shown below:

```
var doc = require('aws-sdk');
var dynamo = new doc.DynamoDB();

exports.handler = function(event, context) {
    var getParams = {
        TableName:'SuperMission'
   };

    dynamo.scan(getParams, function(err, data){
        if (err) console.log(err, err.stack); // an error occurred
        else {
             context.succeed(data);
        }
    });
};
```

10. Examine the code and review what the function does.

This function retrieves a list of super heroes stored in the **SuperMission** DynamoDB table.

11. At the top of the page, click Save

12. Click Test then configure:

- **Event Name:** `myTest`
- Scroll to the bottom of the screen, then click Create

11. At the top of the page, click Test

12. Scroll to the top of the page.

You will see a message telling you that **Execution Result: Succeeded**.

15. Expand **Details**.

You will see a text box displaying the contents of the SuperMission DynamoDB table.

Congratulations! You created and tested a Lambda function that retrieve list of Super heroes that are stored in the **SuperMission** DynamoDB table.

Next, you will create a second Lambda function to retrieve the mission details from the SuperMission DynamoDB table.

16. At the top of the page, click Functions

17. Click **Create function** then configure:

- **Name:** `getmissiondetails`
- **Runtime:** *Node.js 8.10*
- **Existing role:** *SuperDynamoDBQueryRole*
- Click **Create function**

16. In the editor, remove all of the placeholder code.

17. In the **Function code** section, remove all of the code in *index.js*

18. Within the **index.js** tab, paste the code show n below :

```
var doc = require('aws-sdk');
var dynamo = new doc.DynamoDB();

exports.handler = function(event, context) {
        condition = {};
    condition["SuperHero"] = {
                ComparisonOperator: 'EQ',
                AttributeValueList:[{S: event.superhero}]
            }

    var getParams = {
        TableName:'SuperMission',
        ProjectionExpression:"SuperHero, MissionStatus, Villain1, Villain2, Villain3",
        KeyConditions: condition
   };

    dynamo.query(getParams, function(err, data){
        if (err) console.log(err, err.stack); // an error occurred
        else {
             context.succeed(data);
        }
    });
};
```

21. Examine the code and review w hat the function does.

This function retrieves the mission details from the DynamoDB table.

22. At the top of the screen, click Save

23. Click Test then configure:

- **Event Name:** `myTest2`

22. In the editor, remove all of the placeholder code.

23. Copy the code below and paste it into the editor.

This code looks for details of Batman's mission in the DynamoDB table.

```
{
  "superhero": "Batman"
}
```

26. Scroll to the bottom of the screen, then click Create

27. Click Test

Near the top of the page, you will see a message telling you that **Execution Result: Succeeded**. Below that, there will be a text box displaying the mission details for "Batman".

28. Expand **Details**.

29. Review the details.

Congratulations! You created and tested a Lambda function that retrieves mission details from the DynamoDB table.

If you have time…

Go back to the last test you ran. Replace "Batman" with another super hero (Superman, Iron Man) and practice retrieving details for other missions.

End Lab

Follow these steps to close the console, end your lab, and evaluate the experience.

30. Return to the AWS Management Console.

31. On the navigation bar, click **<yourusername\>@<AccountNumber\>**, and then click **Sign Out**.

Conclusion

Congratulations! You have completed this lab. You have successfully:

- Created Lambda functions that retrieve data in a DynamoDB table
- Tested Lambda functions using a template
- Configured custom tests for Lambda functions

Additional Resources

- Amazon DynamoDB
- AWS Lambda