

Build a Serverless Web Application

Lab Overview

In this lab you will build a fully serverless application which will include the following services:

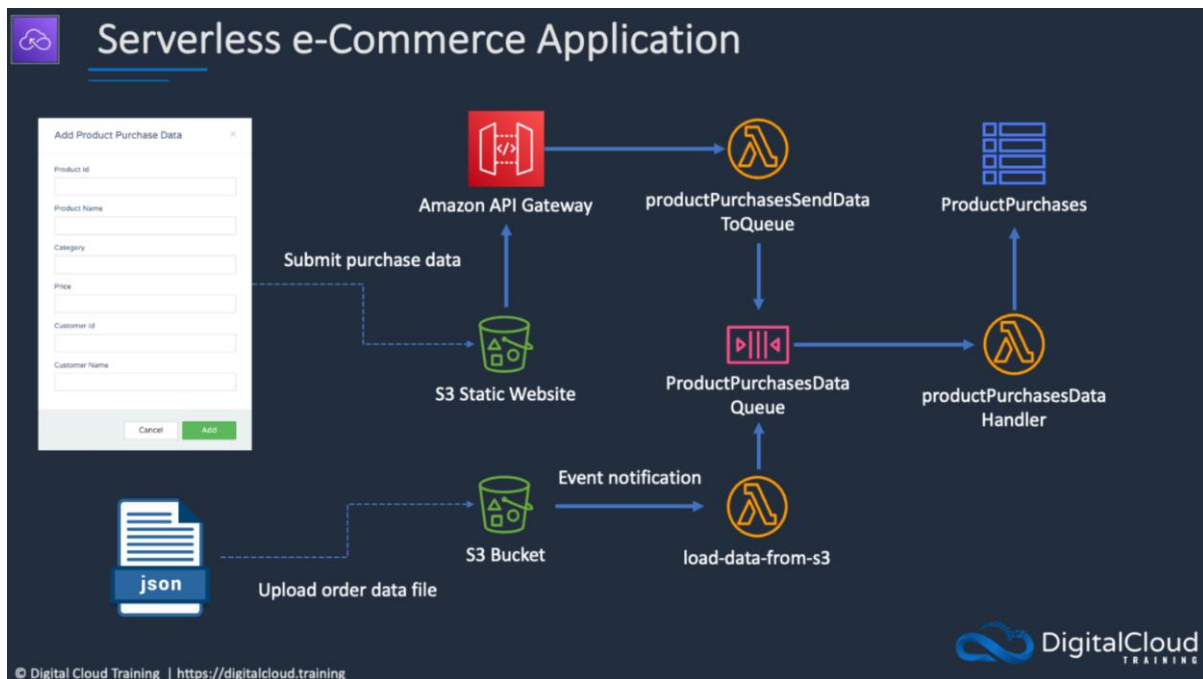
- Amazon SQS
- AWS Lambda
- Amazon DynamoDB
- Amazon S3
- Amazon API Gateway

The application uses a static website hosted in Amazon S3 that connects to an Amazon API Gateway. The website allows shop owners to manually submit product purchase information to the DynamoDB Table.

When data is entered into the webform running on S3, the API triggers an AWS Lambda function to place the data as a message in an SQS queue.

Another Lambda function is then triggered and processes the message in the queue and adds the data to the DynamoDB table.

The image below depicts the solution architecture:



Requirements (Prerequisites)

- **AWS Free Tier Account**
- **The AWS CLI or AWS CloudShell**

Resources

Please download the lab resources zip file [here](#).

Exercise Overview

Exercise 1 - Create the DynamoDB Table

Exercise 2 - Create the SQS Queue

Exercise 3 - Create the Function for Adding Data to DynamoDB

Exercise 4 - Test Adding Data to Queue

Exercise 5 - Create the Function for Adding Orders to the Queue

Exercise 6 - Create the API

Exercise 7 - Create the Static Website, and Test the application

Exercise 1 - Create the DynamoDB Table

Task 1 – Create the DynamoDB Table

First we will create the DynamoDB table which will store the transactions made from the front end.

1. Head over to the DynamoDB console and click on Create Table.
2. Call the table 'ProductPurchases' and enter 'ProductPurchaseKey' for the partition key.
3. Scroll down and click 'Create Table.'

Table details [Info](#)

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

engrmohiuddin4@gmail.com



1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.



1 to 255 characters and case sensitive.

Exercise 2 - Create the SQS Queue

Task 1 – Create the SQS Queue

First we will create the SQS Queue which will store the messages from the front end.

1. Head over to the SQS console, and click on Create Queue
2. Leave the queue as a Standard Queue and call it 'ProductPurchasesDataQueue'.
3. We don't need to make any more changes other than scrolling down and clicking 'Create Queue'.
4. Take note of the URL which you can see on the next screen by saving it in a notepad document.



Exercise 3 - Create the Function for Adding Data to DynamoDB

Task 1 – Create the IAM Role

Next, we will create the Lambda execution role.

1. Head over to the IAM console and click on Create policy.
2. Under the JSON tab find and copy contents of 'lambda-policy-for-sqs-ddb.json' into the JSON field, ensuring you replace your account number.
3. Click 'Create Policy' after calling the permissions policy "pushPurchasesToQueue".
4. We then need to attach this policy to a role. Select Roles in the IAM console and click 'Create Role'.
5. Under 'Use Case' select Lambda and click next.
6. Filter the permissions policy by typing in the name of the policy we just created ('pushPurchasesToQueue') attach it and click next.
7. We will then call the role 'pushPurchasesToQueue' and click 'Create Role'.

Task 2 – Create the first Lambda Function

Next, we will create the first Lambda function which processes messages from the queue and places the order entries into the DynamoDB table.

1. Head over to the Lambda console and click on Create Function.

2. We will call the Lambda function 'productPurchasesDataHandler' and change the runtime to 'Python 3.12'.
3. Click on 'Change Default Execution Role' and click 'Use and Existing Role' and use the 'pushPurchasesToQueue' we selected earlier.
4. Click 'Create Function'.
5. Now we can add the code. Under 'Code' copy and paste the contents of the main.py file from the Part 1 folder in the downloads in your code window in the lambda console.

Task 3 – Configure the SQS Queue

First we will configure the SQS Queue which triggers the Lambda function we just created.

1. Head back over to SQS and click on the 'ProductPurchasesDataQueue' Queue.
2. Click 'Lambda Triggers' and 'Configure a Lambda Trigger'.
3. Select the lambda function we just created and click save.

Lambda function

Set incoming messages to trigger a Lambda function.

Region

us-east-1

Specify an AWS Lambda function available for this queue.

arn:aws:lambda:us-east-1:475243172894:function:productPurchasesDataHandler ▼

Exercise 4 – Test Adding Data to Queue

Task 1 – Run the Test Code

We will run AWS CLI commands to test that messages placed in the queue are properly processed by Lambda and added to the DynamoDB table.

1. Open AWS CloudShell and upload and extract the zip file download. You can also use the wget command with the S3 download URL.
2. Unzip the download and change to the Part-1 folder in which the 'message-body-1.json' file is (build-a-serverless-app\Part-1).

3. Modify the queue URL for the command in the instructions.md file. Then run the command.

```
aws sqs send-message --queue-url **YOUR-QUEUE-URL**
--message-body file://message-body-1.json
```

If this works, we should see Lambda has written items to the DynamoDB Table which will correspond to the JSON file of whichever message-body you have chosen. Feel free to run this command with all 5 message bodies to see them populate the DynamoDB table.

Items returned (1)								
<input type="checkbox"/>	ProductPurchaseKey	Category	CustomerId	CustomerNa...	PricePerUnit	ProductId	ProductNa...	TimeOfVisit
<input type="checkbox"/>	2c7b95fd-2367-4141-...	Accessories	be44af0a-7...	John Doe	10	c96b49bb-...	Gloves	2021-01-31T16:23...

Exercise 5 – Create the Function for Adding Orders to the Queue

Task 1 – Create the IAM Role

Next, we will create the second Lambda execution role.

1. Head over to the IAM console and click on Create policy.
2. Copy the contents of the "lambda-policy.json" file (from the Part-2 folder) into the JSON field, ensuring you replace your account number.
3. Click 'Create Policy after calling the permissions policy 'productPurchasesSendMessage'.
4. We then need to attach this policy to a role. Select Roles in the IAM console and click 'Create Role'.
5. Under 'Use Case' select Lambda and click next.
6. Filter the permissions policy by typing in the name of the policy we just created ('productPurchasesSendMessage') attach it and click next.
7. We will then call the role 'productPurchasesSendMessage' and click 'Create Role'.

Task 2 – Create the Second Lambda function

We will now create another Lambda function. This function will receive order information from the frontend API and then place the order information into the queue for subsequent processing.

1. Navigate to the Part-2/DCTProductPurchaseForm\backend folder and edit the main.py file.
2. Update the queue URL in the main.py with your SQS queue URL and save the file (you should only need to change the account ID).

```
16  
17 QueueUrl: "https://sqs.us-east-1.amazonaws.com/821711655051/ProductPurchasesDataQueue"  
18 };
```

3. Create an AWS Lambda function and call it 'productPurchasesSendDataToQueue', select the Python 3.12 runtime.
4. Select the 'productPurchasesSendMessage' which we created earlier for the Lambda execution role.
5. Click 'Create Function'.
6. Copy the code from the edited main.py into the code editor and deploy the changes.

Exercise 6 – Create the API

Task 1 – Create the REST API.

We will now create the REST API.

1. Head to the API Gateway console, and click Build under the REST API – do not click the Private option, make sure you build this one:

REST API

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Import from Swagger or Open API 3 ☐ Example API

Settings

Choose a friendly name and description for your API.

API name*	<input type="text" value="productPurchase"/>
Description	<input type="text"/>
Endpoint Type	<input type="text" value="Regional"/> ⓘ

2. Select 'New API' and call it 'productPurchase'.
3. Click 'create resource' and call the resource 'productpurchase' and ensure the resource path is /productpurchase.
4. Enable Cross Origin Resource Sharing (CORS) and click 'Create Resource'.
5. Click 'Create Method' and select 'PUT' from the dropdown.
6. Enable a Lambda Proxy integration and select the 'productPurchasesSendDataToQueue' Lambda function.

Integration type ☒ Lambda Function ⓘ

☐ HTTP ⓘ

☐ Mock ⓘ

☐ AWS Service ⓘ

☐ VPC Link ⓘ

Use Lambda Proxy integration ☒ ⓘ

Lambda Region

Lambda Function

Use Default Timeout ☒ ⓘ

Click “Create Method”.

7. Under Actions click ‘Deploy API’ – and choose ‘new stage’ and call the stage ‘dev’. Click Deploy.
8. Next, copy the API endpoint (e.g. <https://jtsvxqdt3k.execute-api.us-east-1.amazonaws.com/dev>) and paste it on line 125 in the index.html in the ‘DCTProductPurchaseForm/frontend’ directory.
9. Make sure you keep the /productpurchase after /dev like this:

```
// Replace with your API Gateway endpoint URL
var apiGatewayUrl = 'https://5sg0yk3ma7.execute-api.us-east-1.amazonaws.com/dev/productpurchase';
```

Exercise 7 - Create the Static Website and test the application

Task 1 – Create the S3 Bucket.




We will now create the S3 bucket which will host the code for the static website.

1. Head over to the S3 console and click ‘Create Bucket’.
2. Call the bucket ‘product-purchases-webform-XXXX’ with the Xs numbers representing a random string of letters or numbers, so it remains globally unique.

3. Disable and acknowledge blocking all public access and click 'Create Bucket'.
4. Once the bucket is created, go to the properties tab, and enable static website hosting, and type 'index.html' for the index document.
5. Go to the Permissions section and add the permissions statements from the 'frontend-bucket-policy.json' file. Make sure to change the name of the bucket and save changes.
6. We will next upload the frontend code using CloudShell.
7. Using your terminal/command prompt, navigate to the 'DCTProductPurchaseForm/frontend' directory.
8. From the frontend directory, run the command, to copy the files to S3 (with your bucket name specified).

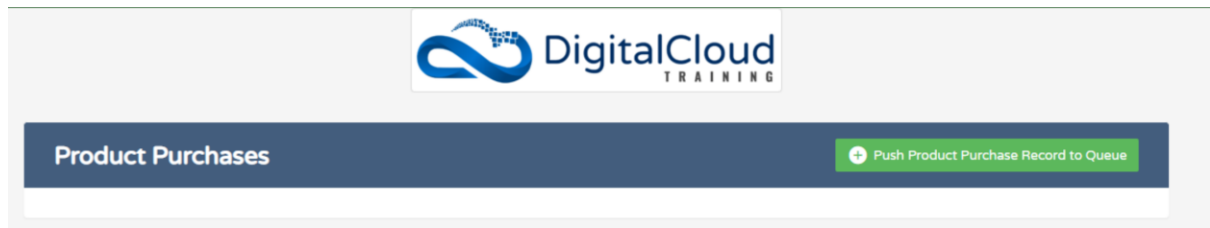
```
aws s3 sync ./ s3://product-purchases-webform-
```

9. You should see a series of commands running which show all the frontend contents being uploaded.
10. If we check the S3 bucket – there should be the code for the website!

<input type="checkbox"/>	Name	▲	Type
<input type="checkbox"/>	 favicon.ico		ico
<input type="checkbox"/>	 index.html		html
<input type="checkbox"/>	 logo.png		png

11. Click on the index.html document and copy the object URL and paste it into the browser.
12. Alternatively, you can use the static website endpoint (HTTP only).

The following website should load:



13. Click 'push product visit record to queue' and enter some information, like this, and click 'Add'.
14. It should have appeared in the DynamoDB table!

Thanks for taking part!