[version_1.0]

Note

The exercises in this course will have an associated charge in your AWS account. In this exercise, you will create the following resources:

- AWS Identity and Access Management (IAM) policy and user (policies and users are AWS account features, offered at no additional charge)
- Amazon DynamoDB table
- AWS Lambda functions
- Amazon Simple Queue Service (Amazon SQS) queue
- Amazon Simple Notification Service (Amazon SNS) topic
- Amazon API Gateway
- Amazon CloudWatch Logs

The final task in this exercise includes instructions to delete all the resources that you create.

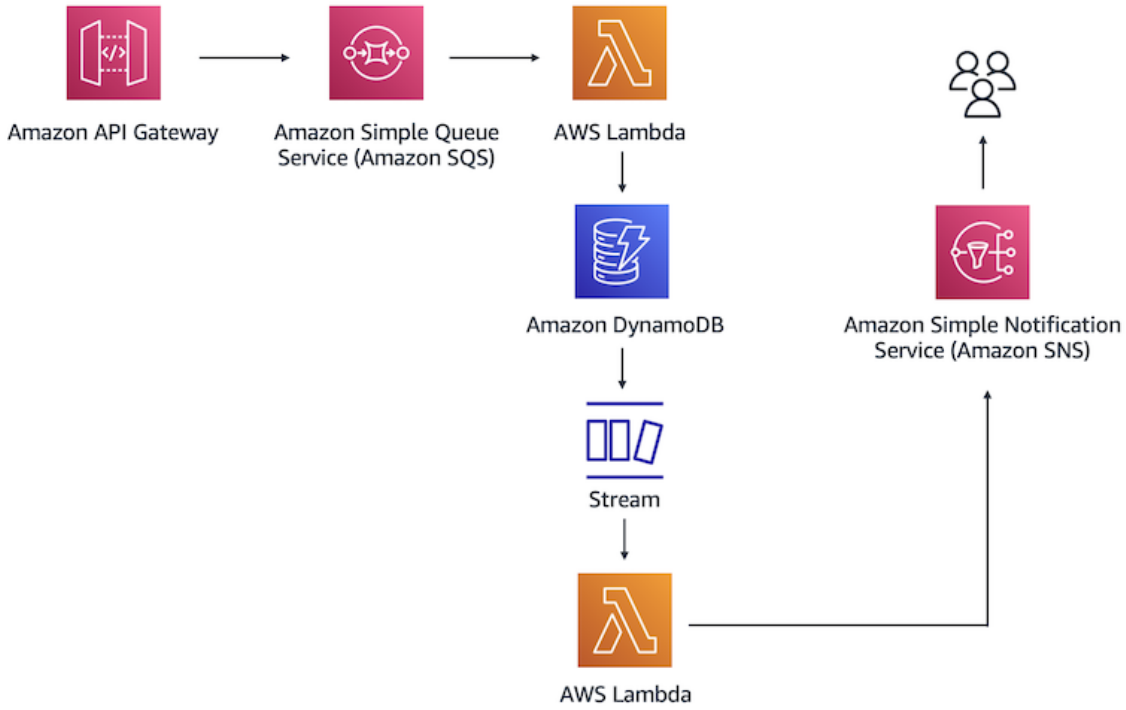
Familiarize yourself with **DynamoDB pricing**, **Amazon SQS pricing**, **Amazon SNS pricing**, **Lambda pricing**, **API Gateway pricing**, **CloudWatch Logs pricing**, and the **AWS Free Tier**.

Exercise 1. Architecting Solutions: Building a Proof of Concept for a Serverless Solution

This exercise provides you with instructions for how to build a proof of concept for a serverless solution in the AWS Cloud.

Suppose you have a customer that needs a serverless web backend hosted on AWS. The customer sells cleaning supplies and often sees spikes in demand for their website, which means that they need an architecture that can easily scale in and out as demand changes. The customer also wants to ensure that the application has decoupled application components.

The following architectural diagram shows the flow for the serverless solution that you will build.



Architecture diagram for exercise 1

In this architecture, you will use a REST API to place a database entry in the Amazon SQS queue. Amazon SQS will then invoke the first Lambda function, which inserts the entry into a DynamoDB table. After that, DynamoDB Streams will capture a record of the new entry in a database and invoke a second Lambda function. The function will pass the database entry to Amazon SNS. After Amazon SNS processes the new record, it will send you a notification through a specified email address.

In this exercise, you will learn how to do the following:

- Create IAM policies and roles to follow best practices of working in the AWS Cloud.
- Create a DynamoDB table to store data.
- Create an Amazon SQS queue to receive, store, and send messages between software components.
- Create Lambda functions and set up triggers to invoke actions in different AWS services.
- Enable DynamoDB Streams to capture modifications in the database table.
- Configure Amazon SNS to receive email or text notifications.
- Create a REST API to insert data into a database.

Notes:

To complete the instructions in this exercise, choose the **US East (N. Virginia) us-east-1** Region in the navigation pane of the AWS Management Console.

The instructions might prompt you to enter your account ID. Your account ID is a 12-digit account number that appears under your account alias in the top-right corner of the AWS Management Console. When you enter your account number (ID), make sure that you remove hyphens (-).

Task 1. Setup: Creating IAM policies and roles

When you first create an account on AWS, you become a root user, or an account owner. We don’t recommend that you use the account root user for daily operations and tasks. Instead, you should use an IAM user or IAM roles to access specific services and features. IAM policies, users, and roles are offered at no additional charge.

In this task, you create custom IAM policies and roles to grant limited permissions to specific AWS services.

Step 1.1: Creating custom IAM policies

1. Sign in to the AWS Management Console.
2. In the search box, enter **IAM**.
3. From the results list, choose **IAM**.
4. In the navigation pane, choose **Policies**.
5. Choose **Create policy**.

   The **Create policy** page appears. You can create and edit a policy in the visual editor or use JSON. In this exercise, we provide JSON scripts to create policies. In total, you must create four policies.
6. In the JSON tab, paste the following code:

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Sid": "VisualEditor0",
               "Effect": "Allow",
               "Action": [
                   "dynamodb:PutItem",
                   "dynamodb:DescribeTable"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

   This JSON script grants permissions to put items into the DynamoDB table. The asterisk (*) indicates that the specified actions can apply to all available resources.
7. Choose **Next: Tags** and then choose **Next: Review**.
8. For the policy name, enter `Lambda-Write-DynamoDB` .
9. Choose **Create policy**.
10. After you create the Lambda-Write-DynamoDB policy, repeat the previous steps to create the following policies:

- A policy for Amazon SNS to get, list, and publish topics that are received by Lambda:
  - **Name**: `Lambda-SNS-Publish`
  - **JSON**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sns:Publish",
                "sns:GetTopicAttributes",
                    "sns:ListTopics"
            ],
                "Resource": "*"
        }
    ]
 }
```

- A policy for Lambda to get records from DynamoDB Streams:
  - **Name**: `Lambda-DynamoDBStreams-Read`
  - **JSON**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetShardIterator",
                "dynamodb:DescribeStream",
                "dynamodb:ListStreams",
                "dynamodb:GetRecords"
            ],
            "Resource": "*"
        }
    ]
}
```

- A policy for Lambda to read messages that are placed in Amazon SQS:
  - **Name**: `Lambda-Read-SQS`
  - **JSON**:

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:ReceiveMessage",
                "sqs:GetQueueAttributes",
                "sqs:ChangeMessageVisibility"
            ],
            "Resource": "*"
        }
    ]
}
```

Step 1.2: Creating IAM roles and attaching policies to the roles

Because AWS follows the principle of least privilege, we recommend that you provide role-based access to only the AWS resources that are required to perform a task. In this step, you create IAM roles and attach policies to the roles.

1. In the navigation pane of the IAM dashboard, choose **Roles**.
2. Choose **Create role** and in the **Select trusted entity** page, configure the following settings:
   - **Trusted entity type**: *AWS service*
   - **Common use cases**: *Lambda*
3. Choose **Next**.
4. On the **Add permissions** page, select **Lambda-Write-DynamoDB** and **Lambda-Read-SQS**.
5. Choose **Next**
6. For **Role name**, enter `Lambda-SQS-DynamoDB` .
7. Choose **Create role**.
8. Follow the previous steps to create two more IAM roles:
   - An IAM role for AWS Lambda: This role grants permissions to obtain records from the DynamoDB streams and send the records to Amazon SNS. Use the following information to create the role.
     - **IAM role name**: `Lambda-DynamoDBStreams-SNS`
     - **Trusted entity type**: *AWS service*
     - **Common use cases**: *Lambda*
     - **Attach policies**: *Lambda-SNS-Publish* and *Lambda-DynamoDBStreams-Read*
   - An IAM role for Amazon API Gateway: This role grants permissions to send data to the SQS queue and push logs to Amazon CloudWatch for troubleshooting. Use the following information to create the role.
     - **IAM role name**: `APIGateway-SQS`
     - **Trusted entity type**: *AWS service*

- **Common use cases**: *API Gateway*
- **Attach policies**: *AmazonAPIGatewayPushToCloudWatchLogs*

Task 2: Creating a DynamoDB table

In this task, you create a DynamoDB table that ingests data that's passed on through API Gateway.

1. In the search box of the AWS Management Console, enter **DynamoDB**.
2. From the list, choose the **DynamoDB** service.
3. On the **Get started** card, choose **Create table** and configure the following settings:
   - **Table**: `orders`
   - **Partition key**: `orderID`
   - Data type: Keep **String**
4. Keep the remaining settings at their default values, and choose **Create table**.

Task 3: Creating an SQS queue

In this task, you create an SQS queue. In the architecture for this exercise, the Amazon SQS receives data records from API Gateway, stores them, and then sends them to a database.

1. In the AWS Management Console search box, enter `SQS` and from the list, choose **Simple Queue Service**.
2. On the **Get started** card, choose **Create queue**.

   The **Create queue** page appears.
3. Configure the following settings:
   - **Name**: `POC-Queue`
   - **Access Policy**: *Basic*
   - **Define who can send messages to the queue**:
     - Select *Only the specified AWS accounts, IAM users and roles*
     - In the box for this option, paste the Amazon Resource Name (ARN) for the *APIGateway-SQS* IAM role
     - **Note**: For example, your IAM role might look similar to the following: `arn:aws:iam::<account ID>:role/APIGateway-SQS .`
   - **Define who can receive messages from the queue**:
     - Select *Only the specified AWS accounts, IAM users and roles*.
     - In the box for this option, paste the ARN for the *Lambda-SQS-DynamoDB* IAM role.
     - **Note**: For example, your IAM role might look similar to the following: `arn:aws:iam::<account_ID>:role/Lambda-SQS-DynamoDB`
4. Choose **Create queue**.

Task 4: Creating a Lambda function and setting up triggers

In this task, you create a Lambda function that reads messages from the SQS queue and writes an order record to the DynamoDB table.

Step 4.1: Creating a Lambda function for the Lambda-SQS-DynamoDB role

1. In the AWS Management Console search box, enter **Lambda** and from the list, choose **Lambda**.
2. Choose **Create function** and configure the following settings:
   - **Function option**: *Author from scratch*
   - **Function name**: `POC-Lambda-1`
   - **Runtime**: *Python 3.9*
   - **Change default execution role**: *Use an existing role*
   - **Existing role**: `Lambda-SQS-DynamoDB`
3. Choose **Create function**.

Step 4.2: Setting up Amazon SQS as a trigger to invoke the function

1. If needed, expand the **Function overview** section.
2. Choose **Add trigger**.
3. For **Trigger configuration**, enter `SQS` and choose the service in the list.
4. For **SQS queue**, choose **POC-Queue**.
5. Add the trigger by choosing **Add**.

Step 4.3: Adding and deploying the function code

1. On the **POC-Lambda-1** page, in the **Code** tab, replace the default Lambda function code with the following code:

```
import boto3, uuid

client = boto3.resource('dynamodb')
table = client.Table("orders")

def lambda_handler(event, context):
    for record in event['Records']:
        print("test")
        payload = record["body"]
        print(str(payload))
        table.put_item(Item= {'orderID': str(uuid.uuid4()),'order':  payload})
```

2. Choose **Deploy**.

   The Lambda code passes arguments to a function call. As a result, when a trigger invokes a function, Lambda runs the code that you specify.

   When you use Lambda, you are responsible only for your code. Lambda manages the memory, CPU, network, and other resources to run your code.

Step 4.4: Testing the POC-Lambda-1 Lambda function

1. In the **Test** tab, create a new event that has the following settings:
   - **Event name**: `POC-Lambda-Test-1`
   - **Template-Optional**: `SQS`

   The SQS template appears in the **Event JSON** field.

2. Save your changes and choose **Test**.

   After the Lambda function runs successfully, the “Execution result: succeeded” message appears in the notification banner in the **Test** section. This means that the Lambda function sent a test message “Hello from SQS!” from the SQS template to the DynamoDB table.

Step 4.5: Verifying that the Lambda function adds the test message to a database

1. In the AWS Management Console search box, enter `DynamoDB` and from the list, choose **DynamoDB**.

2. In the navigation pane, choose **Explore items**.

3. Select the **orders** database. Under **Items returned**, the **orders** table returns “Hello from SQS!” from the Lambda function test.

Task 5: Enabling DynamoDB Streams

In this task, you enable DynamoDB Streams. A DynamoDB stream captures information about every modification to data items in the table.

1. In the DynamoDB console, in the **Tables** section of the navigation pane, choose **Update settings**.

2. In the **Tables** card, make sure that the **orders** table is selected.

3. Choose the **Exports and streams** tab.

4. In the **DynamoDB stream details** section, choose **Enable**.

5. For **View type**, choose **New image**.

6. Choose **Enable stream**.

   After the Lambda function reads messages from the SQS queue and writes an order record to the DynamoDB table, DynamoDB Streams captures the primary key attributes from the record.

Task 6: Creating an SNS topic and setting up subscriptions

In this task, you create an SNS topic and set up subscriptions. Amazon SNS coordinates and manages delivering or sending messages to subscriber endpoints or clients.

Step 6.1: Creating a topic in the notification service

1. In the AWS Management Console, search for `SNS` and choose **Simple Notification Service**.
2. On the **Create topic** card, enter `POC-Topic` and choose **Next step**.
3. In the **Details** section, keep the **Standard** topic type selected and choose **Create topic**.
4. On the **POC-Topic** page, copy the ARN of the topic that you just created and save it for your reference.

   You will need the ARN for the SNS topic later in this exercise.

Step 6.2: Subscribing to email notifications

1. On the **Subscriptions** tab, choose **Create subscription**.
2. For **Topic ARN**, make sure that the box contains the ARN for POC-Topic.
3. To receive notifications, for **Protocol**, choose **Email**.
4. For **Endpoint**, enter your email address.
5. Choose **Create subscription**.

   The confirmation message is sent to the email address that you specified.
6. After you receive the confirmation email message, confirm the subscription. If you don't receive an email message within a few minutes, check the spam folder.

Task 7: Creating an AWS Lambda function to publish a message to the SNS topic

In this task, you create a Lambda function for the Lambda-DynamoDBStreams-SNS role. The second Lambda function uses DynamoDB Streams as a trigger to pass the record of a new entry to Amazon SNS.

Step 7.1: Creating a POC-Lambda-2 function

1. In the AWS Management Console, search for and open AWS Lambda.
2. Create a new Lambda function by choosing **Create function**, and configure the following settings:
   - **Function option**: *Author from scratch*
   - **Function name**: `POC-Lambda-2`
   - **Runtime**: *Python 3.9*
   - **Change default execution role**: *Use an existing role*
   - **Existing role**: *Lambda-DynamoDBStreams-SNS*

   This role grants permissions to get records from DynamoDB Streams and send them to Amazon SNS.
3. Choose **Create function**.

Step 7.2: Setting up DynamoDB as a trigger to invoke a Lambda function

1. In the **Function overview** section, choose **Add trigger** and configure the following settings:
   - **Trigger configuration**: Enter `DynamoDB` and from the list, choose **DynamoDB**.
   - **DynamoDB table**: *orders*
2. Keep the remaining default settings and choose **Add**.
3. In the **Configuration** tab, make sure that you are in the **Triggers** section and that the DynamoDB state is “Enabled.”

Step 7.3: Configuring the second Lambda function

1. Choose the **Code** tab and replace the Lambda function code with the following code:

```
import boto3, json

client = boto3.client('sns')

def lambda_handler(event, context):

    for record in event["Records"]:

        if record['eventName'] == 'INSERT':
            new_record = record['dynamodb']['NewImage']
            response = client.publish(
                TargetArn='<Enter Amazon SNS ARN for the POC-Topic>',
                Message=json.dumps({'default': json.dumps(new_record)}),
                MessageStructure='json'
            )
```

   Note: In the function code, replace the **TargetArn** value with the ARN for the Amazon SNS POC-Topic. Make sure that you remove the placeholder angle brackets (<>).

   Your ARN might look similar to the following: `arn:aws:sns:us-east-1:<account ID>:POC-Topic`.

2. Choose **Deploy**.

Step 7.4: Testing the POC-Lambda-2 Lambda function

1. On the **Test** tab, create a new event and for **Event name**, enter `POC-Lambda-Test-2`.
2. For **Template-optional**, enter `DynamoDB` and from the list, choose **DynamoDB-Update**.

   The DynamoDB template appears in the **Event JSON** box.
3. Save your changes and choose **Test**.

   After the Lambda function successfully runs, the “Execution result: succeeded” message should appear in the notification banner in the **Test** section.

In a few minutes, an email message should arrive at the email address that you specified in the previous task.

4. Confirm that you received the subscription email message. If needed, check both your inbox and spam folder.

Task 8: Creating an API with Amazon API Gateway

In this task, you create a REST API in Amazon API Gateway. The API serves as a communication gateway between your application and the AWS services.

1. In the AWS Management Console, search for and open **API Gateway**.

2. On the **REST API** card with a public authentication, choose **Build** and configure the following settings:
   - **Choose the protocol**: *REST*
   - **Create new API**: *New API*
   - **API name**: `POC-API`
   - **Endpoint Type**: *Regional*

3. Choose **Create API**.

4. On the **Actions** menu, choose **Create Method**.

5. Open the method menu by choosing the down arrow, and choose **POST**. Save your changes by choosing the check mark.

6. In the **POST - Setup** pane, configure the following settings:
   - **Integration type**: *AWS Service*
   - **AWS Region**: *us-east-1*
   - **AWS Service**: *Simple Queue Service (SQS)*
   - **AWS Subdomain**: Keep empty
   - **HTTP method**: *POST*
   - **Action Type**: *Use path override*
   - **Path override**: Enter your account ID followed by a slash (/) and the name of the POC queue
     - **Note**: If *POC-Queue* is the name of the SQS queue that you created, this entry might look similar to the following: `/<account ID>/POC-Queue`
   - **Execution role**: Paste the ARN of the APIGateway-SQS role
     - **Note**: For example, the ARN might look like the following: `arn:aws:iam::<account ID>:role/APIGateway-SQS`
   - **Content Handling**: *Passthrough*

7. Save your changes.

8. Choose the **Integration Request** card.

9. Scroll to the bottom of the page and expand **HTTP Headers**.

10. Choose **Add header**.

11. For **Name**, enter `Content-Type`

12. For **Mapped from**, enter `'application/x-www-form-urlencoded'`

13. Save your changes to the **HTTP Headers** section by choosing the check mark.
14. Expand **Mapping Templates** and for **Request body passthrough**, choose **Never**.
15. Choose **Add mapping template** and for **Content-Type** , enter `application/json`
16. Save your changes by choosing the check mark.
17. For **Generate template**, do not choose a default template from the list. Instead, enter the following command: `Action=SendMessage&MessageBody=$input.body` in a box.
18. Choose **Save**.

Task 9: Testing the architecture by using API Gateway

In this task, you use API Gateway to send mock data to Amazon SQS as a proof of concept for the serverless solution.

1. In the API Gateway console, return to the **POST - Method Execution** page and choose **Test**.
2. In the **Request Body** box, enter:

```
{  "item": "latex gloves",
"customerID":"12345"}
```

3. Choose **Test**.

   If you see the “Successfully completed execution” message with the 200 response in the logs on the right, you will receive an email notification with the new entry. If you don’t receive an email, but the new item appears in the DynamoDB table, troubleshoot the exercise instructions starting from *after* you set up DynamoDB. Ensure that you deploy all of the resources in the *us-east-1* Region.

   After API Gateway successfully processes the request that you pasted in the **Request Body** box, it places the request in the SQS queue. Because you set up Amazon SQS as a trigger in the first Lambda function, Amazon SQS invokes the function call. The Lambda function code places the new entry into the DynamoDB table. DynamoDB Streams captures this change to the database and invokes the second AWS Lambda function. This function gets the new record from DynamoDB Streams and sends it to Amazon SNS. Amazon SNS, in turn, sends you an email notification.

Task 10: Cleaning up

In this task, you delete the AWS resources that you created for this exercise.

1. Delete the DynamoDB table.
   1. Open the DynamoDB console.
   2. In the navigation pane, choose **Tables**.
   3. Select the **orders** table.
   4. Choose **Delete** and confirm your actions.
2. Delete the Lambda functions.
   1. Open the Lambda console.

2. Select the Lambda functions that you created in this exercise: **POC-Lambda-1** and **POC-Lambda-2**.
3. Choose **Actions**, **Delete**.
4. Confirm your actions and close the dialog box.

3. Delete the SQS queue.
    1. Open the Amazon SQS console.
    2. Select the queue that you created in this exercise.
    3. Choose **Delete** and confirm your actions.
4. Delete the SNS topic and subscriptions.
    1. Open the Amazon SNS console.
    2. In the navigation pane, choose **Topics**.
    3. Select **POC-Topic**.
    4. Choose **Delete** and confirm your actions.
    5. In the navigation pane, choose **Subscriptions**.
    6. Select the subscription that you created in this exercise and choose **Delete**.
    7. Confirm your actions.
5. Delete the API that you created.
    1. Open the API Gateway console.
    2. Select **POC-API**.
    3. Choose **Actions**, **Delete**.
    4. Confirm your actions.
6. Delete the IAM roles and policies.
    1. Open the IAM console.
    2. In the navigation pane, choose **Roles**.
    3. Delete the following roles and confirm your actions:
        - **APIGateway-SQS**
        - **Lambda-SQS-DynamoDB**
        - **Lambda-DynamoDBStreams-SNS**
    4. In the navigation pane, choose **Policies**.
    5. Delete the following custom policies and confirm your actions:
        - **Lambda-DynamoDBStreams-Read**
        - **Lambda-SNS-Publish**
        - **Lambda-Write-DynamoDB**
        - **Lambda-Read-SQS**

Congratulations! You have successfully completed the exercise.