*[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.**

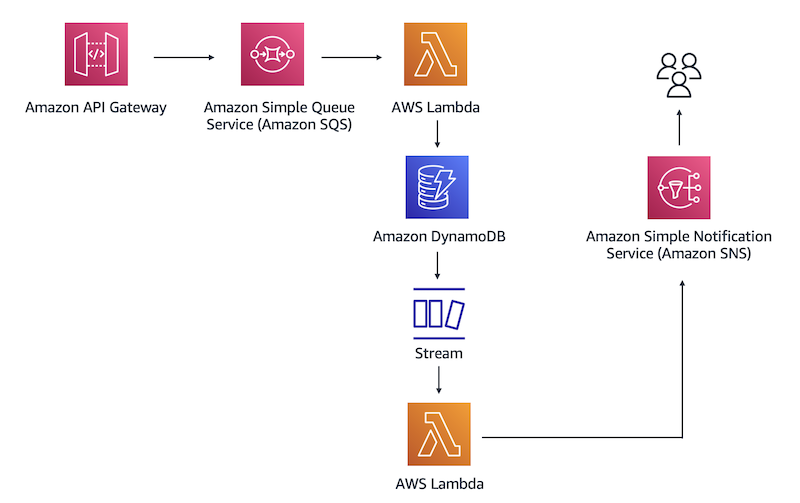
Familiarize yourself with [**DynamoDB pricing**](https://aws.amazon.com/dynamodb/pricing/), [**Amazon SQS pricing**](https://aws.amazon.com/sqs/pricing/), [**Amazon SNS pricing**](https://aws.amazon.com/sns/pricing/), [**Lambda pricing**](https://aws.amazon.com/lambda/pricing/), [**API Gateway pricing**](https://aws.amazon.com/api-gateway/pricing/), [**CloudWatch Logs pricing**](https://aws.amazon.com/cloudwatch/pricing/), and the [**AWS Free Tier**](https://aws.amazon.com/free/).

**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.

1. In the JSON tab, paste the following code:
2. {
3. "Version": "2012-10-17",
4. "Statement": [
5. {
6. "Sid": "VisualEditor0",
7. "Effect": "Allow",
8. "Action": [
9. "dynamodb:PutItem",
10. "dynamodb:DescribeTable"
11. ],
12. "Resource": "\*"
13. }
14. ]

}

 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.

 Choose **Next: Tags** and then choose **Next: Review**.

 For the policy name, enter Lambda-Write-DynamoDB.

 Choose **Create policy**.

 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.

1. 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
2. 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:
2. import boto3, uuid
3. client = boto3.resource('dynamodb')
4. table = client.Table("orders")
5. def lambda\_handler(event, context):
6. for record in event['Records']:
7. print("test")
8. payload = record["body"]
9. print(str(payload))

table.put\_item(Item= {'orderID': str(uuid.uuid4()),'order': payload})

1. 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.

1. 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.

1. 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.

1. 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:
2. import boto3, json
3. client = boto3.client('sns')
4. def lambda\_handler(event, context):
5. for record in event["Records"]:
6. if record['eventName'] == 'INSERT':
7. new\_record = record['dynamodb']['NewImage']
8. response = client.publish(
9. TargetArn='<Enter Amazon SNS ARN for the POC-Topic>',
10. Message=json.dumps({'default': json.dumps(new\_record)}),
11. MessageStructure='json'

)

1. **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.

1. 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.

1. 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.

1. 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:
3. { "item": "latex gloves",

"customerID":"12345"}

1. 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.

ChatGPT

Explain all the below with steps and examples whether other So you have the frontend clients making requests, sending in orders through their phones, browsers, et cetera. All these requests are going to be directed to Amazon API Gateway, which acts as the front door for your API. API Gateway will handle the authentication for the request, and it will also validate the format of the incoming request to verify that all the necessary fields are included in the payload of the request. Once it passes through the authentication and validation, API Gateway will then send the message to an SQS queue. The message will remain in the queue until an AWS Lambda function is spun up to process the message. This happens quickly, and it's all automated because there is a polling mechanism built into AWS Lambda that will read the messages from the queue. Putting SQS between API Gateway and Lambda is decoupling the API from compute. So that way, if you have a large scaling event and you reach any predefined limits for the Lambda, the messages will be in the queue and will not be lost. Then, Lambda can churn through the messages and catch up. Or to catch up even faster, you can raise your Lambda limits and process the messages that are in the queue. Now, for the Lambda function. This Lambda function will contain the application code for the order service related to order processing and storing the orders. The orders will be stored in an Amazon DynamoDB table. Once an order is stored in the table, an entry gets added to the DynamoDB stream, which then will need to be processed to send the order to the downstream functions. This is where another Lambda function comes into play. And this Lambda function will read the information on the DynamoDB stream and publish the order information to Amazon SNS. SNS, then following a fan-out pattern that will send that message to all of the subscribed endpoints. Those endpoints being the three downstream services for fulfillment, accounting, and inventory. - [Raf] Wow! Okay, this looks awesome. But I have a question for you. What about monitoring and logging? - [Morgan] Yeah, so all of these services are serverless in nature, and have built-in integrations to send metrics and logs to Amazon CloudWatch and Amazon CloudWatch Logs. - [Raf] Right. It's good to hear that we won't need to do a ton of extra configuration to that, get working and in place. What about scaling? Is everything here using managed scaling? - [Morgan] Yes, so the scaling is all managed for these services.

ChatGPT

Certainly! Let’s break down the entire process step-by-step, along with examples to better understand each part of the system.

### 1. Frontend Clients Make Requests

**Description:**

* Users send orders through their phones, browsers, or other devices.
* These requests contain data such as order details (e.g., item ID, quantity, user information).

**Example:**

* A user places an order for a book via a mobile app.

### 2. Amazon API Gateway

**Description:**

* Acts as the entry point for your API.
* Handles authentication and request validation.

**Example:**

* The order request hits the API Gateway, which checks if the user is authenticated (using tokens, API keys, etc.) and ensures the request body includes all necessary fields (e.g., item ID, quantity).

### 3. Sending Message to SQS Queue

**Description:**

* After authentication and validation, API Gateway forwards the request to an Amazon Simple Queue Service (SQS) queue.
* SQS decouples the API Gateway from the compute resources.

**Example:**

* The validated order request is converted into a message and placed into an SQS queue named OrderQueue.

### 4. AWS Lambda Function Processing SQS Messages

**Description:**

* AWS Lambda functions are triggered to process messages from the SQS queue.
* The Lambda function reads the message, processes the order, and stores it.

**Example:**

* A Lambda function, ProcessOrderLambda, reads the message from OrderQueue, processes the order data, and stores it in an Amazon DynamoDB table.

### 5. Storing Orders in DynamoDB

**Description:**

* Orders are stored in an Amazon DynamoDB table, a NoSQL database.
* Each order becomes an item in the table.

**Example:**

* The processed order is stored in a DynamoDB table named OrdersTable with attributes like OrderID, UserID, ItemID, Quantity, and Status.

### 6. DynamoDB Streams

**Description:**

* DynamoDB Streams captures changes to items in the DynamoDB table.
* Changes (e.g., new orders) generate stream records.

**Example:**

* A new order is added to OrdersTable, and a corresponding record is created in the DynamoDB Stream.

### 7. Lambda Function Processing DynamoDB Streams

**Description:**

* Another Lambda function is triggered by changes in the DynamoDB Stream.
* This function reads the stream record and processes it further.

**Example:**

* DynamoDBStreamProcessorLambda reads the stream record and prepares the order information to be sent to downstream services.

### 8. Publishing to Amazon SNS

**Description:**

* The Lambda function publishes the order information to an Amazon Simple Notification Service (SNS) topic.
* SNS follows a fan-out pattern to send messages to all subscribed endpoints.

**Example:**

* DynamoDBStreamProcessorLambda publishes the order details to an SNS topic OrderNotifications.

### 9. SNS Fan-Out to Subscribed Endpoints

**Description:**

* SNS sends the message to all its subscribers, which could be other Lambda functions or other services.
* Subscribed endpoints perform specific actions based on the order information.

**Example:**

* The SNS topic OrderNotifications has three subscribers: FulfillmentServiceLambda, AccountingServiceLambda, and InventoryServiceLambda.
* Each Lambda function handles the order data for its respective domain (e.g., fulfilling the order, updating accounting records, and managing inventory).

### 10. Monitoring and Logging

**Description:**

* All serverless services (API Gateway, Lambda, SQS, DynamoDB, SNS) integrate with Amazon CloudWatch for logging and monitoring.
* Metrics and logs are automatically sent to CloudWatch.

**Example:**

* You can set up CloudWatch dashboards to monitor metrics like the number of processed orders, Lambda invocation counts, and SQS queue depth.
* Logs for each Lambda function are available in CloudWatch Logs for debugging and auditing purposes.

### 11. Managed Scaling

**Description:**

* All components (API Gateway, Lambda, SQS, DynamoDB, SNS) automatically scale based on demand.
* This ensures the system can handle varying loads without manual intervention.

**Example:**

* During a high-traffic event, Lambda functions automatically scale to handle increased order processing, and SQS ensures that no messages are lost even if Lambda invocation rates temporarily lag.

By following this architecture, you ensure that your system is scalable, reliable, and decoupled, making it easier to handle high loads and maintain overall system stability.

Explain all the below with steps and examples First, what we need to do is create some different IAM policies and roles. So let's go ahead and navigate to the IAM console and select IAM. And I have already built these out, but I want to show you what they look like so that you can follow along as well. And like I said, we will include these instructions as well. So the first one that I want to show is this Lambda-DynamoDBStreams-Read. So if you think about our architecture, we have API Gateway pushing a message to SQS. Lambda then reads the message off of SQS, writes to a DynamoDB table, which pushes that information into a stream. A second Lambda function reads that information off of the stream and then publishes the message to SNS. So let's actually start with this Lambda-Read=SQS policy. And you can see here we have the API calls that are necessary for this Lambda function to read from SQS. We have DeleteMessage, ReceiveMessage, GetQueueAttributes, and ChangeMessageVisibility. So these are all the API calls that Lambda needs to read from that queue. Next, we also have Lambda-Write-DynamoDB. This policy will also be attached to a role that will be used by the Lambda function to write to DynamoDB once it reads the message off of the queue. So for this policy, we have the actions as PutItem and DescribeTable. Now that's all of the permissions that that Lambda function needs. The second Lambda function needs to read from DynamoDB Streams and write to SNS. So let's first read the Lambda-DynamoDBStreams-Read policy. And you can see here we have these four API calls, GetShardIterator, DescribeStream, ListStreams and GetRecords. And then for the writing to SNS, we have the policy Lambda-SNS-Publish, which includes the API calls Publish, GetTopicAttributes and ListTopics. So these are all customer managed policies that you can see here. And then there are also AWS managed policies. So it's good to know the difference between the two of those. Customer managed policies are going to be managed by you, whereas AWS managed policies are managed by AWS. So what that means is, for example, if we have something like AWS Marketplace Full Access, if you expand this, you can see that it includes all of the API calls that are needed for this particular service. If there was a new feature that came out, AWS would update this policy, whereas these are custom policies up here, so if we needed an extra API call here to use a different feature, we would have to go in there and edit this ourselves. All right, next, we need to attach these policies to roles, which again, I have already done. So we have some roles we're going to be using. One of them is APIGateway-SQS, and this role, if I click on this, I'm going to go ahead and copy the ARN and paste that into a text editor that I have off the side. We're going to be using the ARNs for these roles in upcoming steps, and you can see the policies that exist for these as well. So for this one, we have this policy that allows API Gateway to push to CloudWatch Logs, and this is an AWS managed policy. Now let's go ahead and go back to the roles. And then for our next role, I want to look at this Lambda-SQS-DynamoDB. This includes the policies for Lambda-Read-SQS, which we just looked at, Lambda-Write-DynamoDB, which we also looked at. And then we have this LambdaBasicExecutionRole, which allows the Lambda function to write to CloudWatch Logs. So we can go ahead and collapse these policies. And then I want to copy the ARN for this role and paste that into a text editor in another screen. Now back to the roles screen here. Let's go ahead and grab the next role, which is Lambda-DynamoDBStreams-SNS. I'm going to go ahead and copy the ARN for this one and then I want to paste that off on another screen. And for the policies for this one, we have this Lambda function, can read from DynamoDB Streams, it can publish to SNS, and it also has that basic execution role, which will allow this function to write to CloudWatch Logs. All right, so that is all the roles that we need for this solution. So next, let's go ahead and create our SQS queue. Type in SQS and then select Simple Queue Service. And then click Create Queue. And from here you can select the type of queue you want, standard or FIFO. We're going to select Standard, and then we're going to name this POC Q-U-E-U-E, queue. And then we're going to accept all the default configurations. If you're planning on taking the exam, I would make sure that you understand what each of these configurations is for and what they do. But for now we're going to go ahead and scroll down, accept the defaults, and then what we need to change is this access policy. So this access policy will determine who's allowed to write to the queue and who's allowed to receive messages from the queue. So if you remember our architecture, API Gateway is going to be publishing messages to the queue. So you want to paste the ARN for the role that API Gateway is going to use here so that we define who can send the messages to the queue. This role, APIGateway-SQS. Now who can receive messages from the queue? We want that to be our first Lambda function. So I'm going to go ahead and grab the ARN for Lambda-SQS-DynamoDB and paste that here. And then you can scroll through here if you would like to and you can read through this access policy in JSON. So now we're going to scroll down and click Create Queue. Now our queue has been created, I'm going to copy the ARN for this queue, and then I'm going to paste it in my text editor so I can use it later. Now, let's go ahead to DynamoDB and select DynamoDB here. And I'm just going to double check to make sure that I am still in North Virginia so that everything's being created in the same account. And then I'm going to go ahead and click Create Table. And here we want to give this table a name. And I have the instructions open in another screen, so that's how I'm remembering each of these little things here. So the table name is going to be 'orders'. Partition Key we will say is OrderID. Then we can scroll down, keep all the default settings and click Create table. All right, so now that we have our SQS queue and we have our DynamoDB table, we need to create our Lambda function. So I'm going to go to the Lambda service and select Lambda. And then I want to click Create Function. And from here I want to Author from scratch, give this function a name, POC-Lambda-1. And then for the runtime, I'm going to select Python 3.9. And then for the permissions, I'm going to change the default execution role and I want to use an existing role. And then I want to select this Lambda-SQS-DynamoDB, and then click Create Function. All right, our function has now been created. We need to now add a trigger. So the SQS queue is going to be the trigger for our function. So we add a trigger, and then we want to select a source, type in SQS, select SQS, and then select the POC-Queue. And then we can just click Add. All right, so now we have the triggers set up for this queue. The next thing that we need to do is modify the code. So right now, this code is just basically a blank stub that just says "Hello from Lambda!". I'm going to copy in, copy and paste some code that we have pre-written for you for this. And this code imports Boto3, which is the Python SDK for AWS. And then we're setting up the clients for DynamoDB and Orders for the table. So the client, this is going to be how you interact with DynamoDB. Then you can see we're saying client, which is DynamoDBClient.table for this specific table called Orders. And then we have this table object here. And then what we're doing is we're processing all of the records in the incoming event, so there's going to be a payload passed into this function. And then we're doing a little print just to test to make sure that it worked. You could take this out if you wanted to. And then we are setting the payload is going to be the body of the message that came in. I'm doing another print here again just to make sure that it worked and you can see that in logs if you wanted to do that. And then I'm calling table.put\_item. We're creating a UUID for that order ID and then including the order information, the payload off of the request. So I'm going to go ahead and click Deploy. And then what I want to do next is create a test event in the Lambda console. So I'm copying some information from the instructions. Then we'll go ahead and click Test and we want to create a new test event. We'll just call this test. And then you can go ahead and paste in the information from the instructions. I clicked format JSON to make it look a little bit prettier. So we should see incoming order information. We'll go here and then we'll go ahead and click Save. And then from here, if we click Test, we should be able to see that our- We have test incoming order information over here. That's that printing that I was saying, that I had those print statements here. So we can see that it did run. But now if we want to verify that this actually wrote to DynamoDB the way that we would anticipate, we can go back to DynamoDB, select DynamoDB, click on Tables, select the Orders table, go to Explore Table Items, and then we can go ahead and see that we have our order here with that information that we sent in from that test payload. Now that test payload was mocking what the payload would look like if it came from SQS. So we're mocking the structure of what SQS will pass to that Lambda function. All right, so next what I want to do on this DynamoDB table is I want to click back here to DynamoDB, back to Tables, select this Orders table, and then select the Exports and Streams tab and then scroll down. We want to enable stream details. And then what we want to do is you want to select new and old images. Or actually, let's just do new image. We'll do the new image whenever you enable the streams. So what that means is what version of the items do you want to push onto the stream? So do you want just the key attributes? You might use this if you have like a large body. Then you might want to just have the key where then it can go look up the payload, maybe the payload's somewhere else. Maybe it lives in S3, maybe it's in DynamoDB, just depends on the size. We're going to go ahead and select New Image. And so this will include all of the information on that item once it's published or added to the table. So we'll go ahead and click Enable Stream here. So now we can see DynamoDB Streams is enabled. And the next thing that I want to do is I want to create our SNS topic. So I'm going to go ahead and navigate to SNS, select SNS, and then I'm going to go ahead and give this topic a name. We can type in POC-Topic, and then click Next Step. And we're going to go ahead and select Standard here. And you can see that we have a difference in subscription protocols. So SQS, Lambda, HTTP, SMS, email, mobile application is supported by Standard, whereas for FIFO, it's only SQS. So we're going to be testing that this works via email, so you want to make sure that you're selecting Standard here. All right, so we're going to select all of the defaults here except for the access policy. So this looks very familiar to SQS, right? So now we're looking at who can publish messages to the topic. What I'm going to do is select this Advanced radio button here, and then we're just going to edit this JSON directly. So what I want to do is let's go ahead and read through this really quick. We have allow star and then that's going to allow people, anybody to publish to this particular SNS topic. So what I want to do is I want to change this to instead reference the ARN of the role that the Lambda function is going to be using to publish to this particular topic. So let's go ahead and scroll down, and then we should be able to click Create topic. All right, so now let's go ahead to Topics. And then if we select POC-Topic, we can now create a subscription. So you want to go ahead and create a subscription here. And for this subscription, you would want to use email. And then you can put your email address in here. I'm not going to do that right now based off of just the protection of my email address, but this is where you would put in your email. You also could put in your phone number here. So it depends on where you are in the world on how this is going to work. So this works for US destinations, which is why we're going to go ahead and select Email, so that way you can put your email in here. Okay, so I'm going to come back once I've set up my email, so you'll see me again in just a second. Okay, so now we have our topic created. I'm going to go ahead and copy the ARN of this topic. And then I'm going to create the second Lambda function. So let's go back to the Lambda console, select Lambda. And now let's create a new function and let's call this one POC-Lambda-2. And then we're going to select that Python 3.9 runtime again, we're going to change the default execution role to use an existing role. And this time we want to select Lambda-DynamoDBStreams-SNS, and then click Create Function. All right, so now what we want to do is we want to go ahead and add a trigger. And then for this trigger we want to use DynamoDB and we want to select our Orders table and then we can go ahead and click Add. Okay, so now we have our DynamoDB table set up as the trigger for this function. Next, we will click on the Code tab and then I'm going to copy and paste the code in from the instructions, so let's go ahead and do that. Now here you can see that we're setting up a client for SNS, and then we are going to be taking the records that were passed in from DynamoDB Streams. And then we are going to say if this was an insert, meaning it was a new item, then we're going to grab the information from that new image. So now you can see why it was important that we use that new image flag for enabling DynamoDB Streams, because we are calling on that attribute directly in this code. And then we are calling client.publish, and then you have to fill in here where are we publishing the message? So client.publish. Publish is the API call that we're sending to SNS. TargetArn, where are we publishing this message? Message, what is the message we are publishing? And then MessageStructure, what shape is this message in? Which it's going to be JSON. So let's go ahead and paste in the ARN for our topic here and then we should be able to deploy this code. All right, so now what we want to do is we can create the API Gateway and then we can test if this whole thing works from there. So let's go scroll up and then we can go to API Gateway, select API Gateway, and then I want to build a REST API. So I'm going to go ahead and click Build for REST API and then click Okay. And now I want to create a new API, so brand new slate and I want to call this POC-API. And then I'm going to click Create API. All right, so you can see now we have basically a blank API. We need to set up our methods and our resources. So what I want to do first is I'm going to create a resource and let's go ahead and call this Order and then create the resource. And then under here, I want to click Create Method. And we're going to do a POST because theoretically, there would be a front end sending an HTTP post to a back end with the path /order that would come in here. So now what I'm going to do is I'm going to select AWS Service for the API because API Gateway has a direct integration between API Gateway and different AWS services. So we're going to use that API integration with SQS. So next what we need to do is select the region, which we want to use North Virginia here. So I'm going to type in us-east-1. And then for the AWS service, we want to use SQS, so we should be able to see that here. There you go. And then we're going to leave the AWS Subdomain blank. We're going to select the HTTP method to be a post. So the reason for that is because API Gateway is going to be posting a message directly to SQS. So then we want to go ahead and select the path override. And from here, what we want to do for the path override is we want to actually provide the, a pointer to the queue. So the way that we're going to do that is you type in your account name and then, or the account number, and then slash the name of your queue. So I just grabbed this off of the end of the ARN and then just changed the colon to a slash, and that's what we need there. So next, for the execution role, this is going to be using that IAM role that we had created in the first step, so I'll go ahead and paste that here. And then I'm just double checking to make sure that everything looks good according to the instructions, and it does look like we have everything ready so I'm going to go ahead and click Save here. All right, so now we're going to go back to this integration request, and then under HTTP Headers, I want to add a header. And for this one I want to call it Content-Type. And then Mapped From, it's single quote application/x-www-form-urlencoded, application/x-www-form-urlencoded, and then single quote. So then I'm going to go ahead and click the Save button here. And then we want to scroll down to Mapping Templates, and from here I want to select Never and then I want to add a mapping template, give it the type application/json, click on the checkbox and then we can go down here. And what we need to type in down here, you can copy and paste this from the instructions. So it's going to go ahead and state Action, and this is written in VTL. So we're essentially transforming the payload so that it can interact with the SQS APIs directly. So SendMessage and MessageBody equals dollar input.body. So again, make sure you're double checking things as you're going along here. There's a lot of little pieces that can go wrong. So we have action equals SendMessage and MessageBody equals dollar input.body. So again, we are taking the payload of the request and we are creating a string. We're creating the language that the SQS API is going to understand so that we can integrate these two things directly. So now I'm going to go ahead and click Save. Make sure you're clicking Save here, that's very important. And then we're going to go ahead and click Back to Method Execution. And now what we can do is we can actually test this from the API Gateway console here. So you click Test and then down to the request body, add in some information that would, that's like it's coming in off of an order. So we have item, latex gloves, customer ID 12345. I'm going to go ahead and click Test. And then over here on the left hand side, we can read the logs. So we can see sending requests to that queue. And then we received the response status 200, which means that it did send. So one fast way to check this is did you get an email? Did you actually receive the email? And then if you did, you'll know that it went through the entire thing. So let me come back and check my email and see if I got it. One second. All right, so I did go ahead and confirm that I do have the email, but we can also go check the DynamoDB table, see if it got that far. So for troubleshooting, something you can do is like check each step, like we just saw that we got the 200 response from SQS, but that doesn't guarantee that the first Lambda function read it from the queue and then wrote it to the DynamoDB table. So we're going to go ahead and check the table to see if the information is there. If we go to Explore Items or we can then, we can see our item is right here. So we know that the integration between SQS and Lambda is working. Now, I did get the email, so I know this thing is working all the way through, but if you didn't get the email, I would go about it that way. First, check the logs between API Gateway and SQS. Are you seeing that 200 response? If you are, next, come look at the DynamoDB table. Is your item here? If it isn't, then that means that you likely have an issue with the Lambda function between SQS. The Lambda function isn't actually reading from SQS, or maybe it's not writing to DynamoDB. Most of the time, these are permissions problems if you are having issues. And then if, let's say, that you have the item here, but it's not in your email, then the next place I would check is the second Lambda function. Is it able to read from the DynamoDB Stream? Is your DynamoDB Stream configured correctly? And then if it is, does your Lambda function then have the correct permissions to publish to SNS, right? Did you make sure your email is typed correctly is another another easy one. All right, so we know that this is working now. I did get the email. So what we want to do now is go ahead and delete everything. All right, so let's go ahead and delete everything. I'm going to delete the table first. Type in delete, click Delete Table. All right, and then let's go to our Lambda functions. Lambda. And then I'm going to select all my functions and I'm going to delete them. We can type in delete, click Delete, close this, looking good. Now let's go to API Gateway. And then from here, I'm going to select that API Gateway that I created and then delete it. And next, let's go to SQS. And from here I'm going to select SQS queue and click Delete. And just as I'm cleaning this up, if you want to do like a real challenge for yourself, I would recommend trying to do this entire lab using either the AWS CLI or using CloudFormation. So instead of doing everything in a point and click fashion, challenge yourself to do infrastructure as code, or to even try to just use the CLI. Just do something that you wouldn't usually think about doing. All right, so we have the DynamoDB table's been deleted, which deletes the stream. We have both Lambda functions are deleted, SNS has been deleted, SQS has been deleted, so has API Gateway, so I think we are all set. All right, Good luck. End of transcript. Skip to the start.

ChatGPT

Sure, let's go step-by-step to create and test the entire architecture described:

### 1. ****IAM Policies and Roles Creation****

**Step 1: Create IAM Policies**

1. **Lambda-Read-SQS Policy**
   * **Permissions**: DeleteMessage, ReceiveMessage, GetQueueAttributes, ChangeMessageVisibility
   * **Policy JSON**:

json

* 
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Action": [
* "sqs:DeleteMessage",
* "sqs:ReceiveMessage",
* "sqs:GetQueueAttributes",
* "sqs:ChangeMessageVisibility"
* ],
* "Resource": "arn:aws:sqs:<region>:<account-id>:<queue-name>"
* }
* ]
* }

 **Lambda-Write-DynamoDB Policy**

* **Permissions**: PutItem, DescribeTable
* **Policy JSON**:

json

* 
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Action": [
* "dynamodb:PutItem",
* "dynamodb:DescribeTable"
* ],
* "Resource": "arn:aws:dynamodb:<region>:<account-id>:table/<table-name>"
* }
* ]
* }

 **Lambda-DynamoDBStreams-Read Policy**

* **Permissions**: GetShardIterator, DescribeStream, ListStreams, GetRecords
* **Policy JSON**:

json

* 
* {
* "Version": "2012-10-17",
* "Statement": [
* {
* "Effect": "Allow",
* "Action": [
* "dynamodb:GetShardIterator",
* "dynamodb:DescribeStream",
* "dynamodb:ListStreams",
* "dynamodb:GetRecords"
* ],
* "Resource": "arn:aws:dynamodb:<region>:<account-id>:table/<table-name>/stream/\*"
* }
* ]
* }

 **Lambda-SNS-Publish Policy**

* **Permissions**: Publish, GetTopicAttributes, ListTopics
* **Policy JSON**:

json

* + {
  + "Version": "2012-10-17",
  + "Statement": [
  + {
  + "Effect": "Allow",
  + "Action": [
  + "sns:Publish",
  + "sns:GetTopicAttributes",
  + "sns:ListTopics"
  + ],
  + "Resource": "arn:aws:sns:<region>:<account-id>:<topic-name>"
  + }
  + ]
  + }

**Step 2: Create IAM Roles and Attach Policies**

1. **APIGateway-SQS Role**
   * Attach AWS managed policy AmazonSQSFullAccess.
2. **Lambda-SQS-DynamoDB Role**
   * Attach custom policies: Lambda-Read-SQS, Lambda-Write-DynamoDB.
   * Attach AWS managed policy AWSLambdaBasicExecutionRole.
3. **Lambda-DynamoDBStreams-SNS Role**
   * Attach custom policies: Lambda-DynamoDBStreams-Read, Lambda-SNS-Publish.
   * Attach AWS managed policy AWSLambdaBasicExecutionRole.

### 2. ****Create SQS Queue****

1. Navigate to the SQS console.
2. Click **Create Queue**.
3. Select **Standard Queue**.
4. Name the queue POC-Queue.
5. Under **Access Policy**:
   * Specify the role ARN for APIGateway-SQS to allow publishing.
   * Specify the role ARN for Lambda-SQS-DynamoDB to allow reading.
6. Click **Create Queue**.
7. Copy the ARN of the queue for later use.

### 3. ****Create DynamoDB Table****

1. Navigate to the DynamoDB console.
2. Click **Create Table**.
3. Name the table orders.
4. Set the **Partition Key** as OrderID (String).
5. Click **Create**.

### 4. ****Create Lambda Function 1****

1. Navigate to the Lambda console.
2. Click **Create Function**.
3. Author from scratch:
   * Name: POC-Lambda-1.
   * Runtime: Python 3.9.
4. Change the execution role to use an existing role: Lambda-SQS-DynamoDB.
5. Click **Create Function**.
6. Add a trigger:
   * Trigger: SQS.
   * Queue: POC-Queue.

**Lambda Function Code**:

python

import boto3

import json

import uuid

dynamodb = boto3.resource('dynamodb')

table = dynamodb.Table('orders')

def lambda\_handler(event, context):

for record in event['Records']:

payload = record['body']

print(payload)

order\_id = str(uuid.uuid4())

table.put\_item(

Item={

'OrderID': order\_id,

'Order': json.loads(payload)

}

)

return {

'statusCode': 200,

'body': json.dumps('Order processed')

}

### 5. ****Enable DynamoDB Streams****

1. Navigate to DynamoDB console.
2. Select the orders table.
3. Go to the **Exports and streams** tab.
4. Enable stream with **New image**.

### 6. ****Create SNS Topic****

1. Navigate to the SNS console.
2. Click **Create Topic**.
3. Select **Standard**.
4. Name the topic POC-Topic.
5. Configure access policy:
   * Allow the Lambda-DynamoDBStreams-SNS role to publish.
6. Click **Create Topic**.
7. Create a subscription:
   * Protocol: Email.
   * Endpoint: <your-email>.

### 7. ****Create Lambda Function 2****

1. Navigate to the Lambda console.
2. Click **Create Function**.
3. Author from scratch:
   * Name: POC-Lambda-2.
   * Runtime: Python 3.9.
4. Change the execution role to use an existing role: Lambda-DynamoDBStreams-SNS.
5. Click **Create Function**.
6. Add a trigger:
   * Trigger: DynamoDB.
   * Table: orders.

**Lambda Function Code**:

python

import boto3

import json

sns = boto3.client('sns')

TOPIC\_ARN = 'arn:aws:sns:<region>:<account-id>:POC-Topic'

def lambda\_handler(event, context):

for record in event['Records']:

if record['eventName'] == 'INSERT':

new\_image = record['dynamodb']['NewImage']

order\_info = json.dumps(new\_image)

sns.publish(

TopicArn=TOPIC\_ARN,

Message=order\_info,

Subject='New Order'

)

return {

'statusCode': 200,

'body': json.dumps('Order notification sent')

}

### 8. ****Create API Gateway****

1. Navigate to the API Gateway console.
2. Create a new **REST API**.
3. Name the API POC-API.
4. Create a new resource:
   * Name: Order.
   * Path: /order.
5. Create a POST method for the resource.
6. Integrate with AWS Service:
   * Service: SQS.
   * Region: <region>.
   * Path: <account-id>/POC-Queue.
   * Execution role: APIGateway-SQS.
7. Configure the integration request:
   * HTTP Header: Content-Type, value: application/x-www-form-urlencoded.
   * Mapping Template: application/json:

vtl

* + Action=SendMessage&MessageBody=$input.body

1. Deploy the API.

### 9. ****Test the Architecture****

1. Test the API Gateway by sending a POST request to the endpoint.
2. Verify if the message is added to SQS, read by Lambda, stored in DynamoDB, and finally notified through SNS.
3. Check the email for the notification.

### 10. ****Clean Up Resources****

1. Delete all created resources to avoid unnecessary charges:
   * DynamoDB table.
   * Lambda functions.
   * API Gateway.
   * SQS queue.
   * SNS topic.
   * IAM roles and policies.

This step-by-step guide ensures that you set up and test the architecture as described.

Top of Form

You’ve reached your GPT-4o limit.

Responses will use GPT-3.5 until your limit resets in 1 minute.

Bottom of Form



ChatGPT can make mistakes. Check important info.

ChatGPT