**Create a Decoupled Backend Architecture Using Lambda, SQS and DynamoDB**

Created by[Stephane Maarek | AWS Certified Cloud Practitioner,Solutions Architect,Developer](https://deloittedevelopment.udemy.com/user/stephane-maarek/)

**AWS Workspace**

**60-120 minutes**

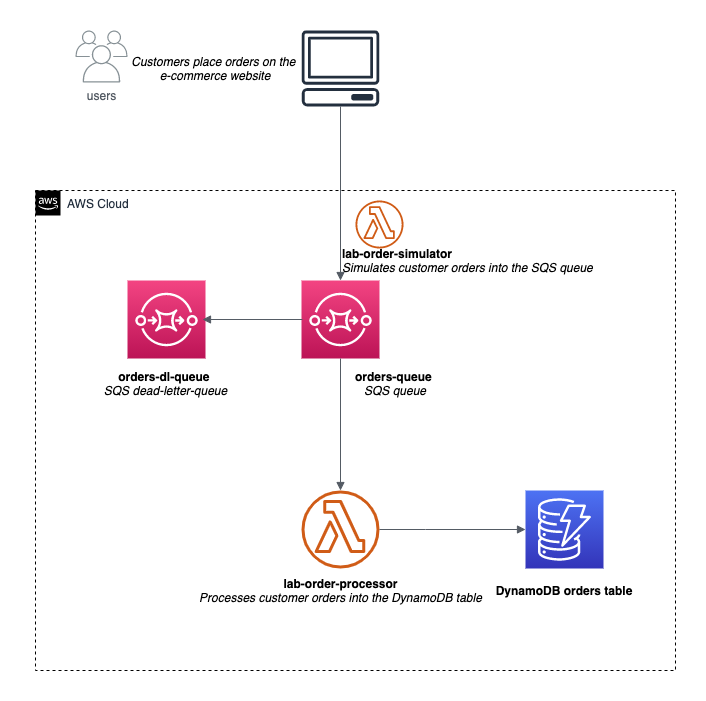
In this lab, you will experience what its like to be a backend engineer working for an e-commerce company. The company is gearing up for the Thanksgiving sale that is the biggest annual shopping event for the company. Your job is to implement a decoupled backend architecture that is able to handle the spike in traffic for the duration of the sale and ensure that none of the customer orders are lost.

Our team has been given a mandate to recommend an upgrade for the backend architecture to prepare for the upcoming Thanksgiving sale. Historically, our company has seen a 5X traffic spike for e-commerce for the duration of this sale. In the past, there have been cases where the system went down and customer orders were lost, leading to significant revenue loss for the company. We need you to devise a decoupled serverless backend architecture that is able to handle the spike in traffic for the duration of the sale.

To build your solution, you will leverage an SQS standard queue to send, store, and receive order messages and then configure the SQS Event Source to trigger the Lambda function which in turn processes the orders into a DynamoDB table.

**How you'll work**

Your project has been broken into a set of tasks. To complete these tasks, use the provided workspace. You can launch your workspace by clicking below or using the button in the top right of the screen.



Tasks

1-Create SQS standard queue to send, store, and receive order message

2-create Dynamodb table to capture the processed order data

3-create a lambda function to simulate orders into the SQS queue

4-Create a Lambda function to process orders into Dynamodb table

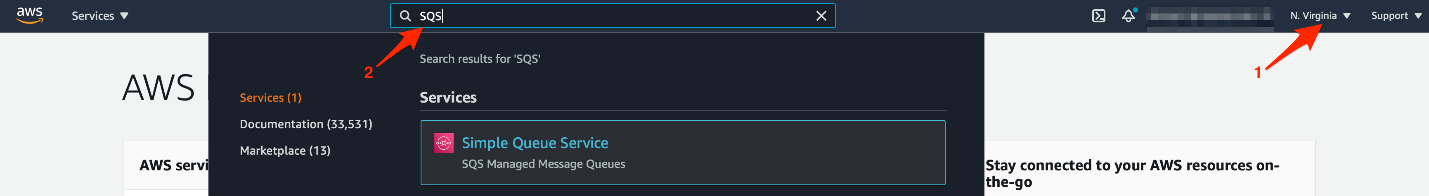
5-Validate the end-to-ed decoupled backend architecture

6-Clean up

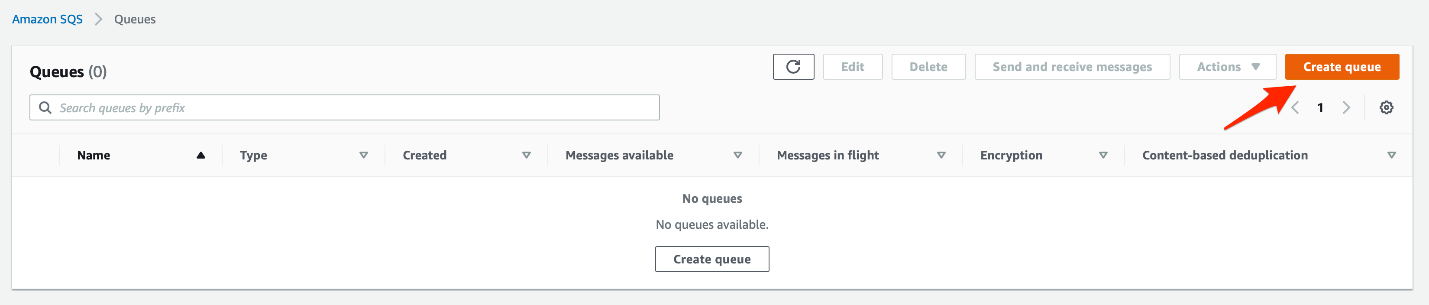
Create SQS standard queue to send, store, and receive order message

You will need to create an SQS standard queue to process the customer orders captured on the e-commerce website. You are also required to create a dead-letter-queue to ensure that you dont lose any orders that might have missing data or badly formatted data. Dead-letter queues are important because they provide a way to reprocess failed orders and debug application issues.

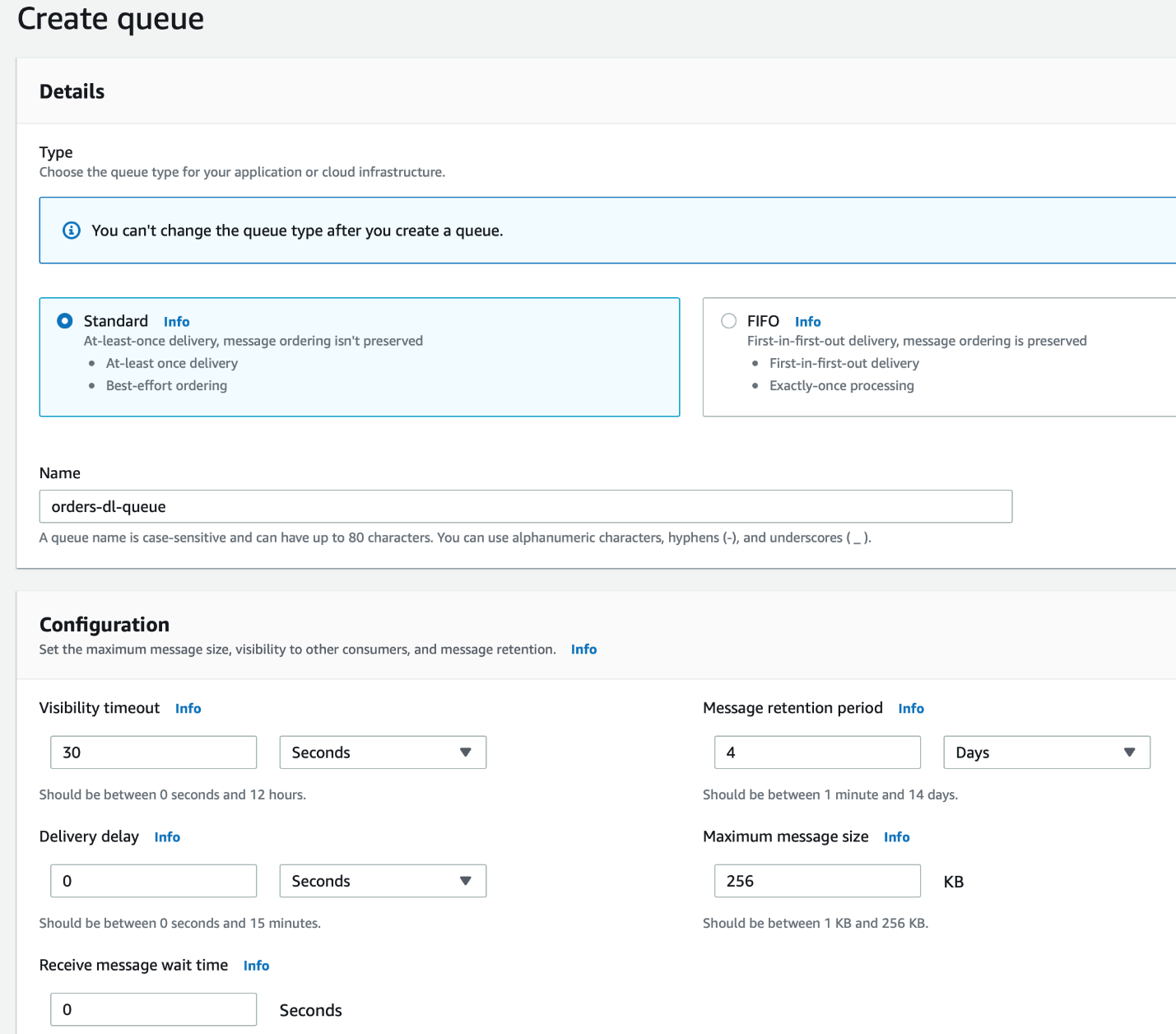
1. Make sure that you are in the N.Virginia AWS Region on the AWS Management Console. Enter **SQS** in the search bar and select **Simple Queue Service**.

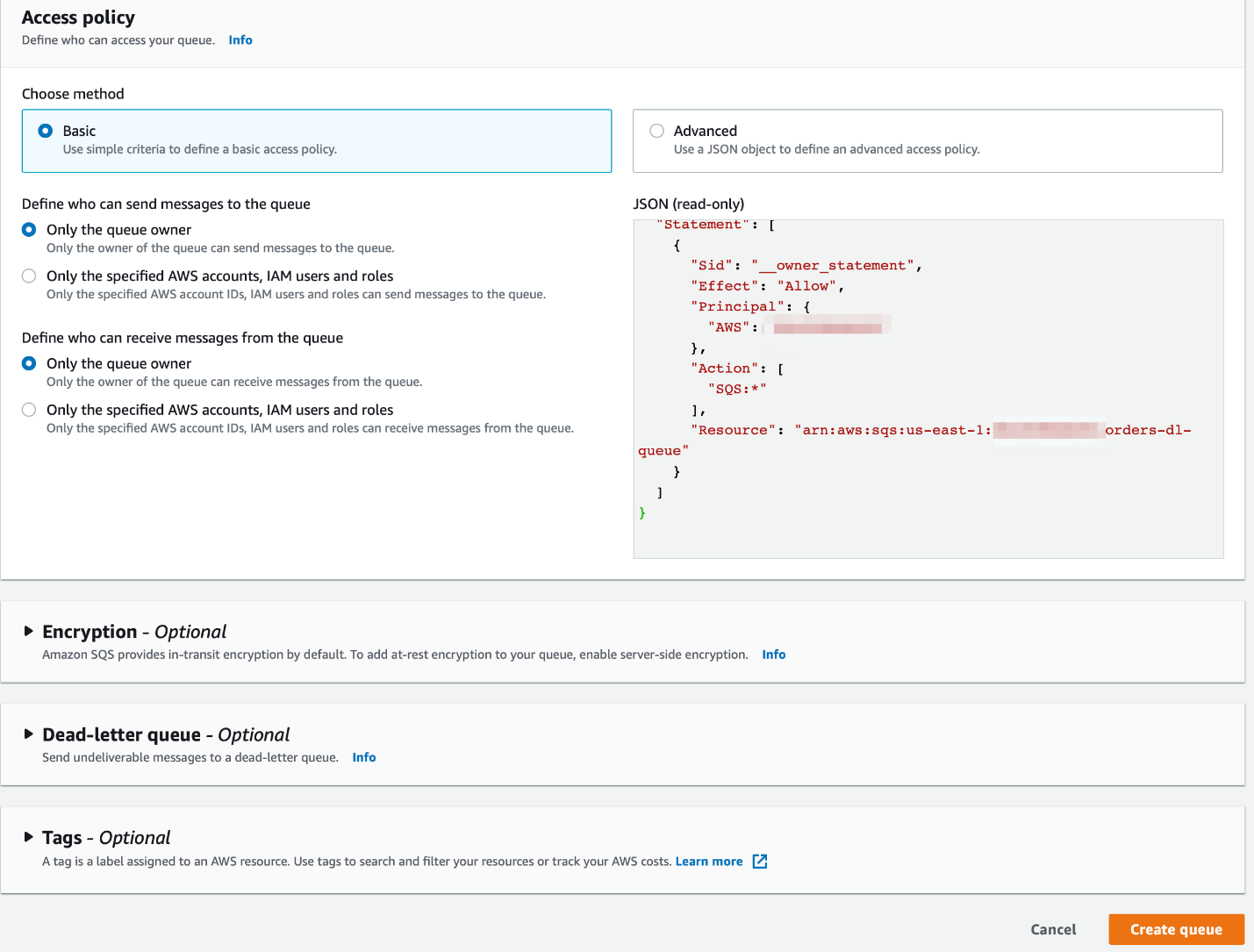


1. Click **Create queue**

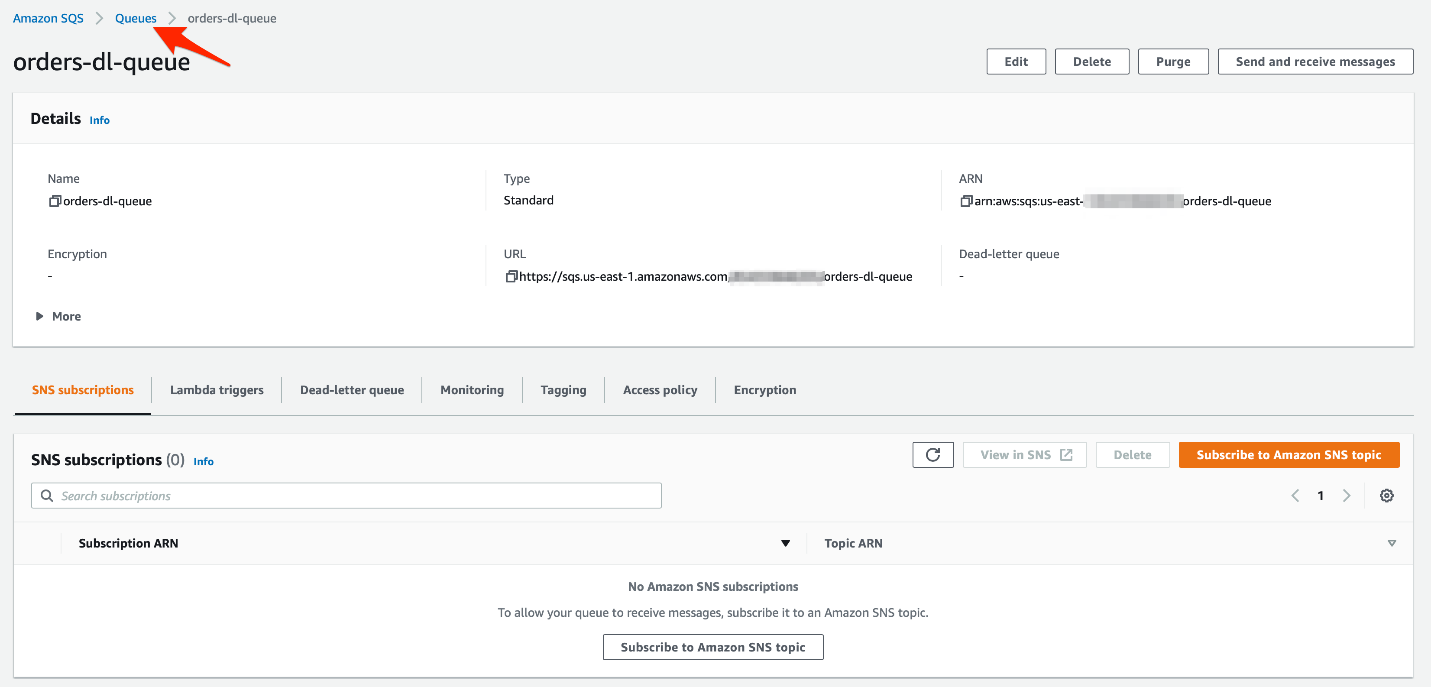


1. Lets create a SQS standard queue which will be used as a dead-letter-queue to ensure that we dont lose any orders that might have missing data or badly formatted data to process the customer orders. You can create this queue by filling in the values as shown below:
   1. Select the **Standard**type
   2. Enter queue name as **orders-dl-queue**
   3. Leave the default values unchanged for the *Configuration, Access policy, Encryption, Dead-letter queue*and*Tags*sections.Then click on **Create** **queue** to set up the queue.

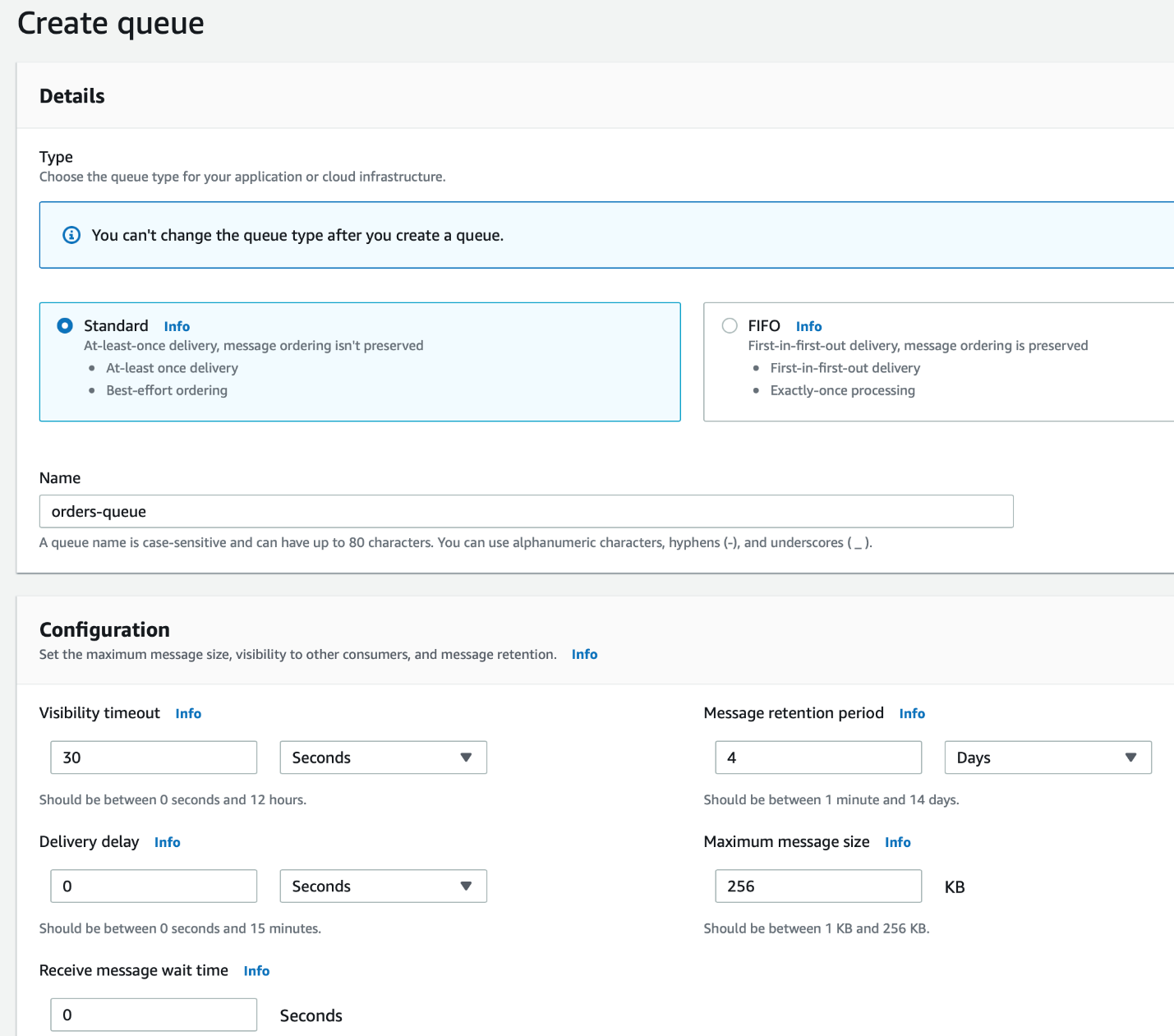


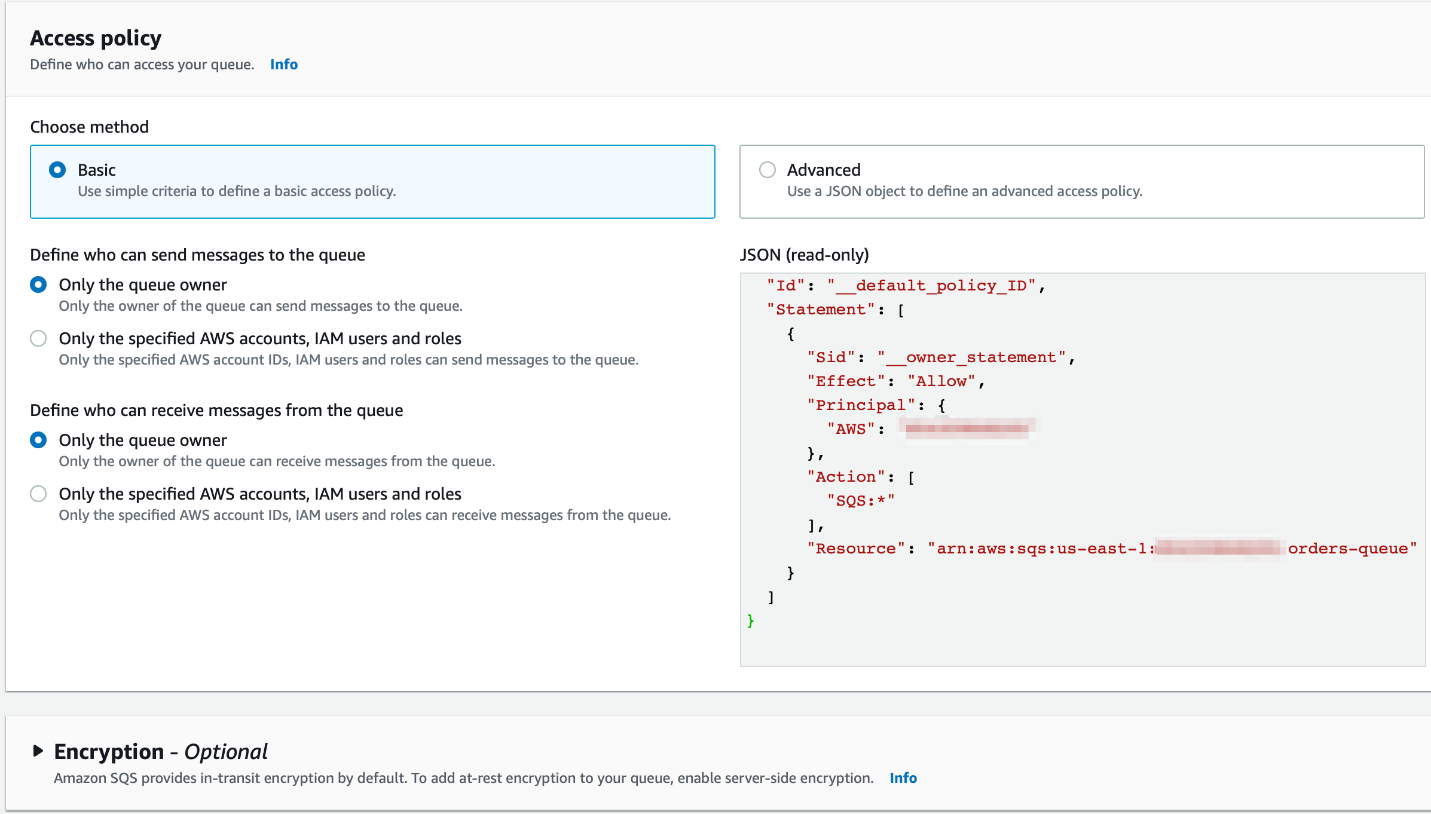


1. You should see that your **orders-dl-queue** has been successfully created. Now lets create the SQS standard queue to process the customer orders. Click on the **Queues** link shown below to go back and then click on the **Create queue**button.

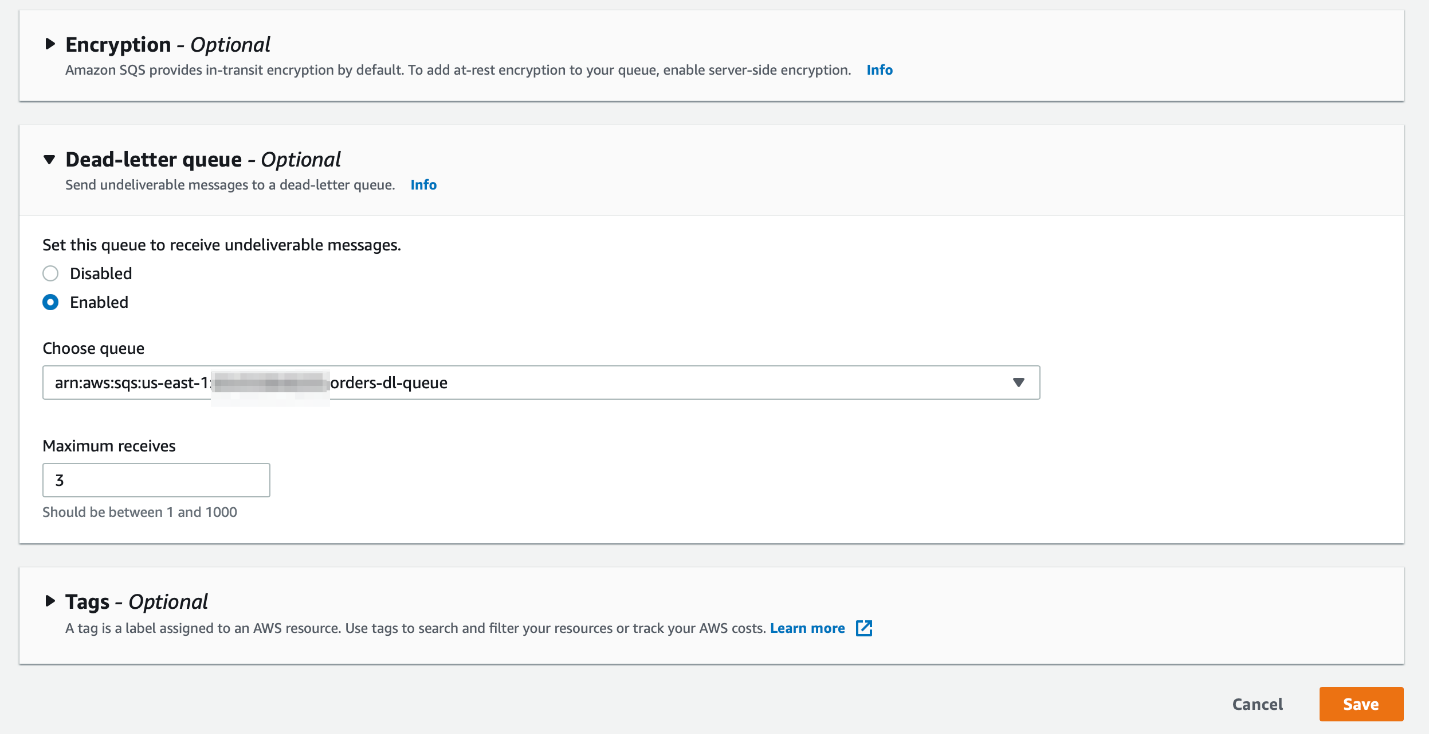


1. Lets fill in the values for the customer orders queue as shown below:
   1. Select the **Standard**type
   2. Enter queue name as **orders-queue**
   3. Leave the default values unchanged for the *Configuration, Access policy, Encryption*and*Tags*sections.





* 1. For the *Dead-letter queue*section, you should enable the option to receive undeliverable messages. Then, you should select the **orders-dl-queue**queue that you created in Step 3. Lastly, you should also note that the **Maximum receives** is set to 3. Then click on **Create** **queue**.



1. You should see that your **orders-queue** has been successfully created



<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html>

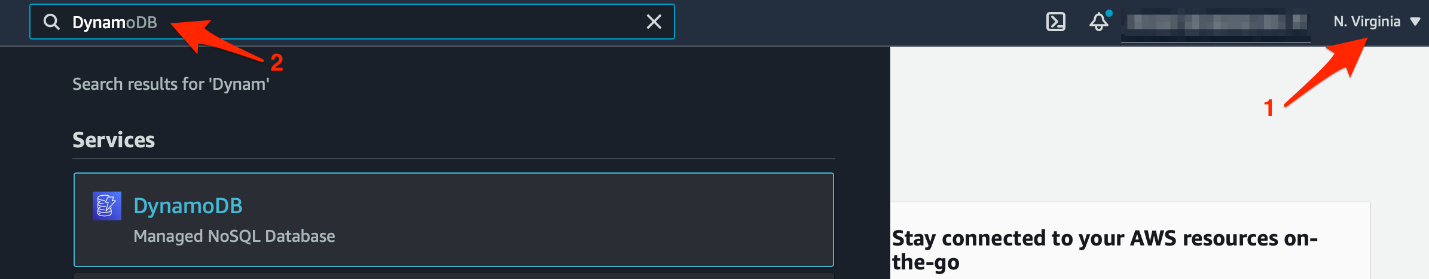
<https://aws.amazon.com/blogs/aws/amazon-sqs-new-dead-letter-queue/>

<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-configure-queue-parameters.html>

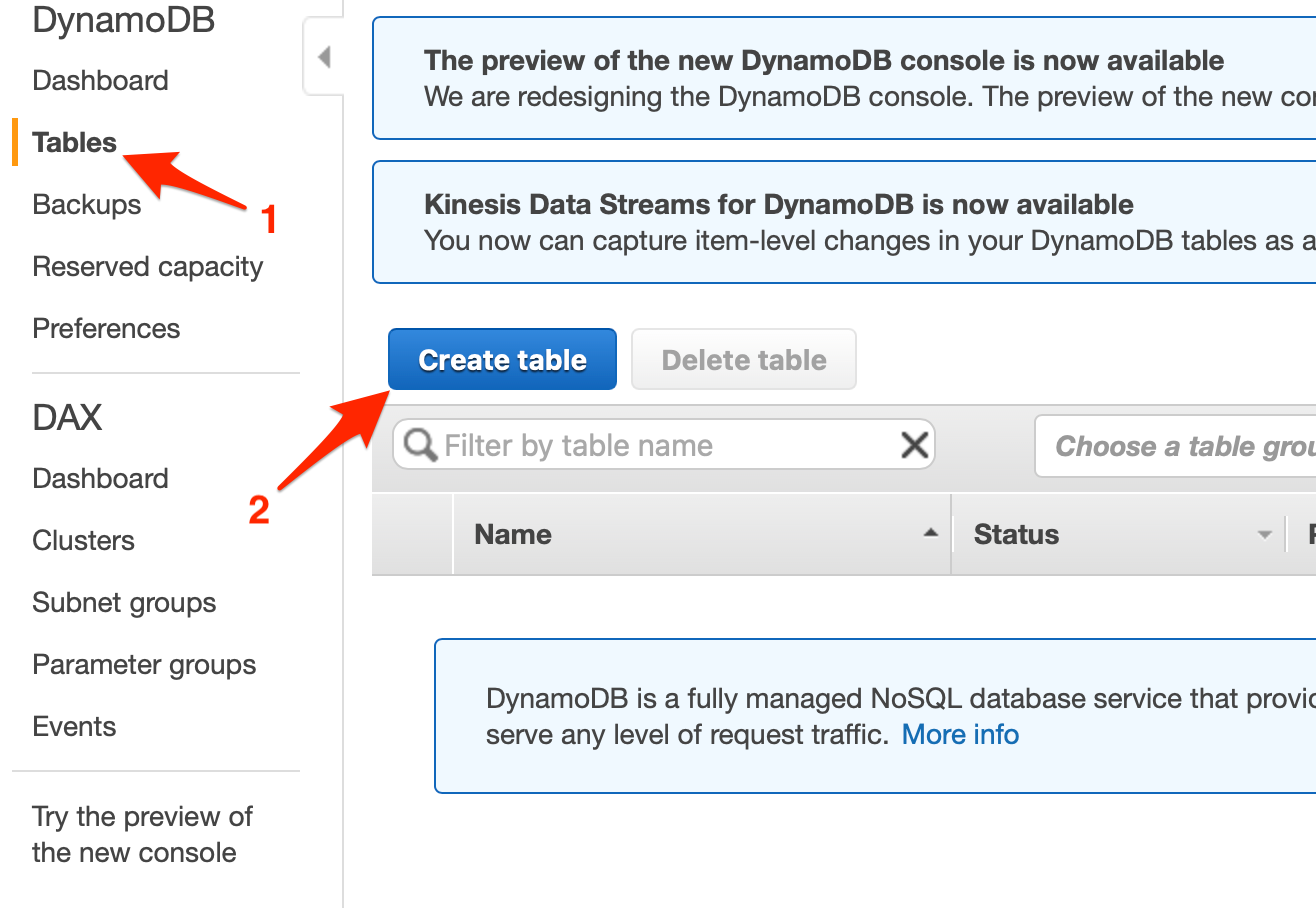
create Dynamodb table to capture the processed order data

You need to create a DynamoDB Table to capture the processed order data. You can use the default settings for creating the table in provisioned mod

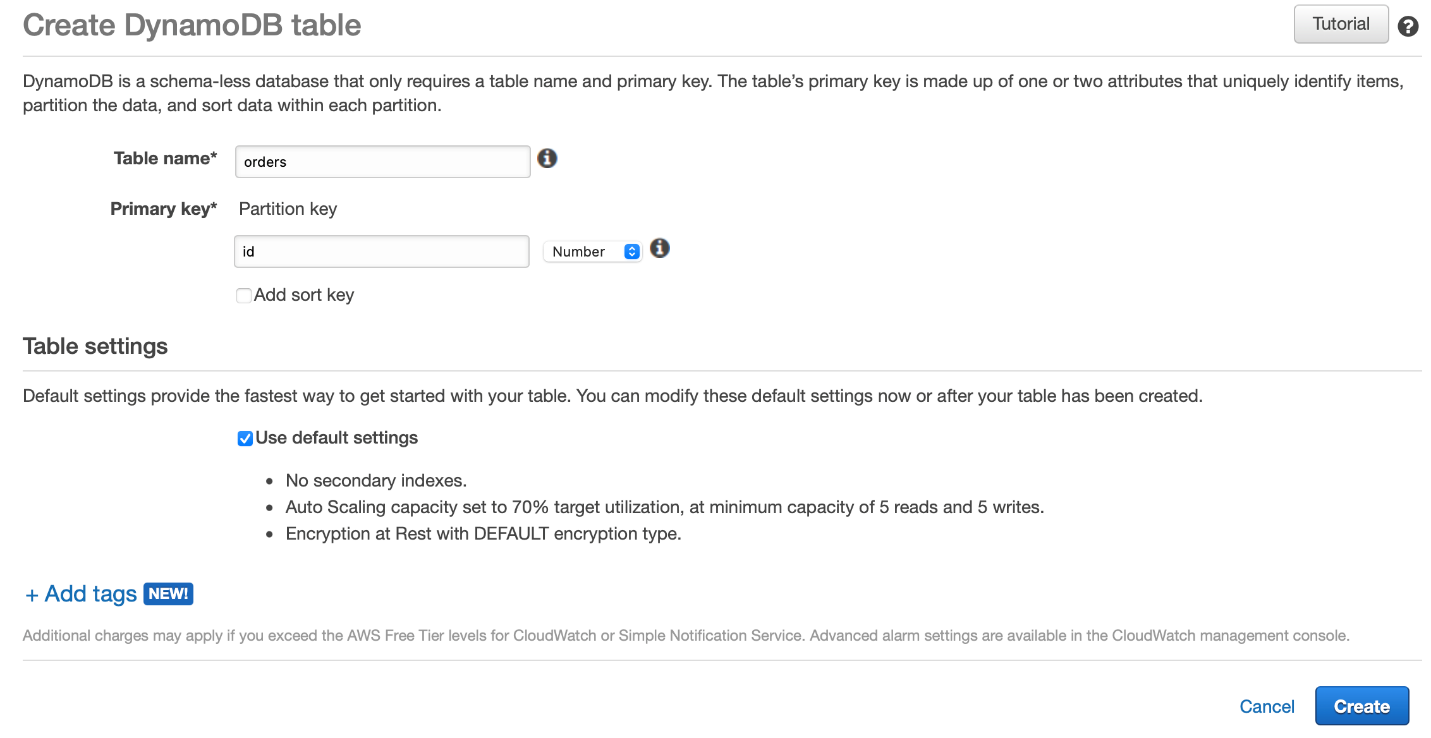
1. Make sure that you are in the N.Virginia AWS Region on the AWS Management Console. Enter **DynamoDB** in the search bar and select **DynamoDB**service.



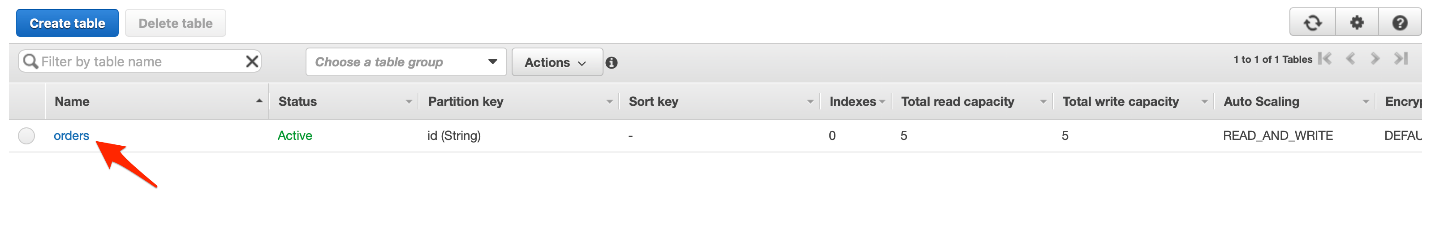
1. Select **Tables** from the left sidebar and then click on **Create table**button



1. Enter the required values for the table as follows:
   1. Table name: **orders**
   2. Primary key: **id**
   3. Make sure that you select the type as **Number** for the Primary key **id**
   4. Select the option to **Use default settings**
   5. Click **Create** to complete the configuration for the table.



1. You should see that the **orders** table has been successfully created



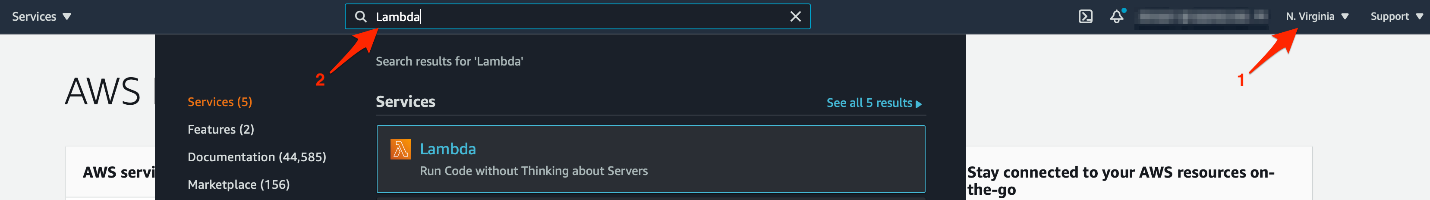
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html>

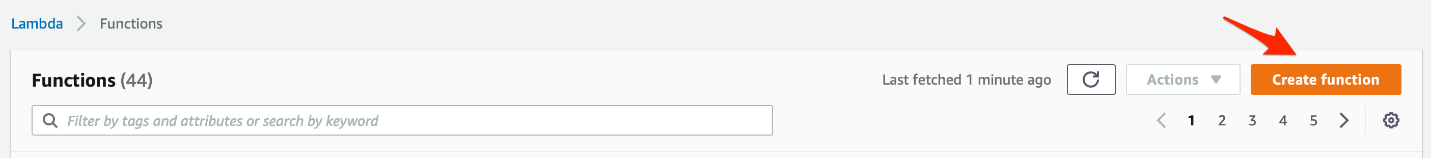
create a lambda function to simulate orders into the SQS queue

You will need to create a Lambda function to put some customer order messages into the designated SQS queue. These messages would later be used to trigger the order processing lambda that will eventually store the order data in the DynamoDB table.

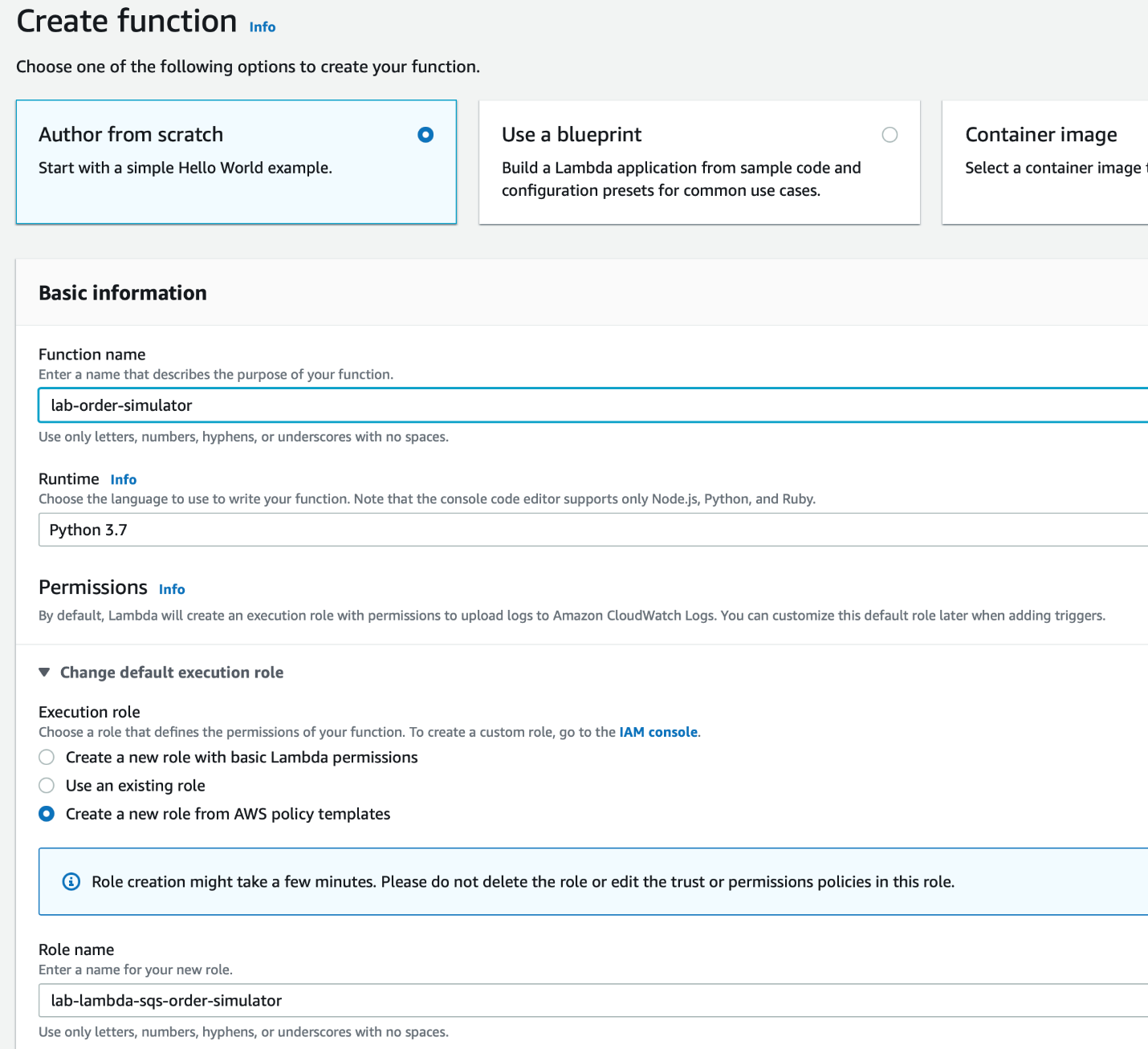
1. Make sure that you are in the N.Virginia AWS Region on the AWS Management Console. Enter **Lambda** in the search bar and select **Lambda**service.



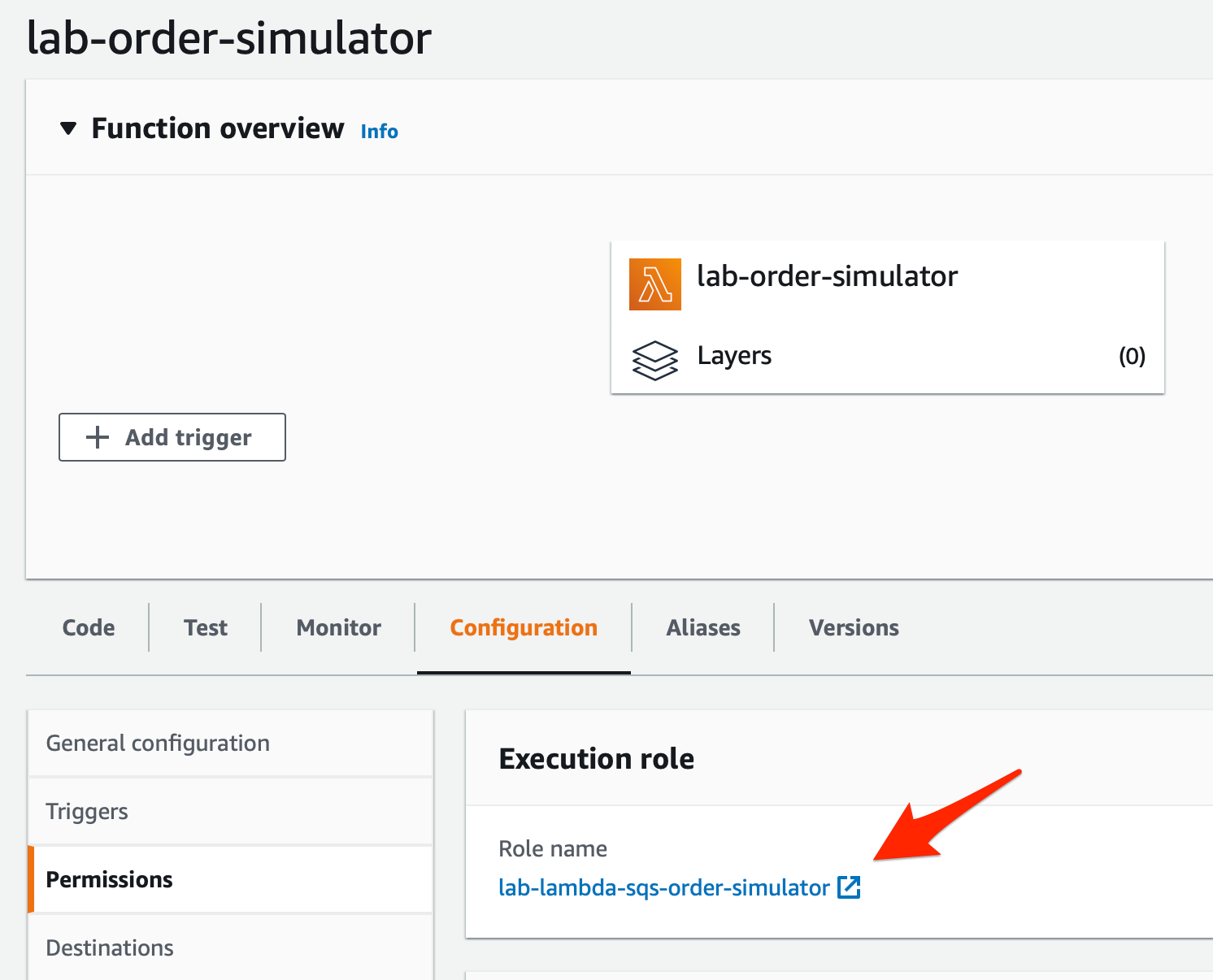
Click on **Create function** button



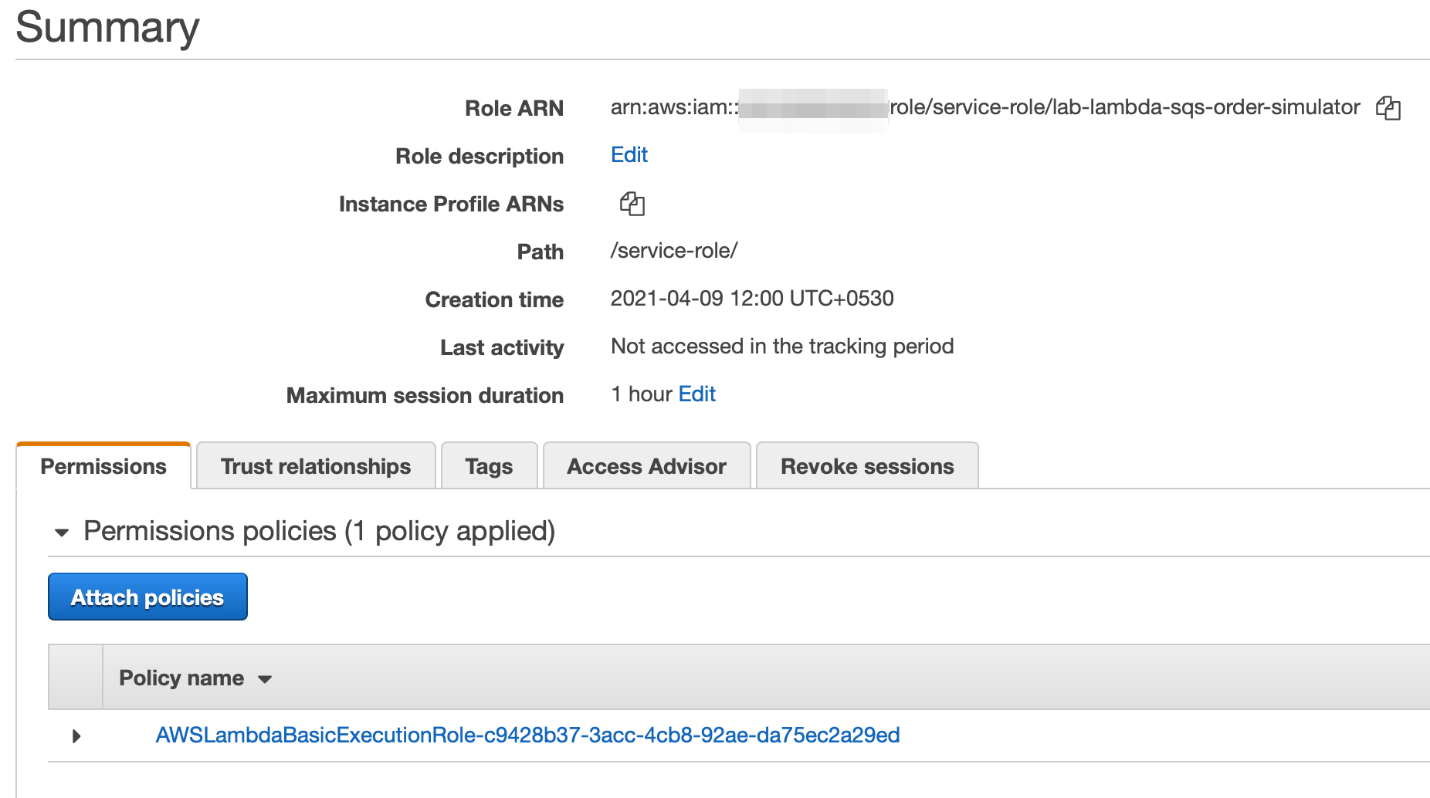
1. Lets create the Lambda function by filling in the following values:
   1. Select the **Author from scratch** template
   2. Function name: **lab-order-simulator**
   3. Runtime: **Python 3.7**
   4. For execution role, select the option that says **Create a new role from AWS policy templates**
   5. Role name: **lab-lambda-sqs-order-simulator**
   6. Click **Create function** to complete this step



1. Configure the Lambda to modify the permissions for its execution role.
   1. Select the **Configuration** tab and click on the **Permissions** link on the left sidebar. Then, click on the hyperlink for the execution role.



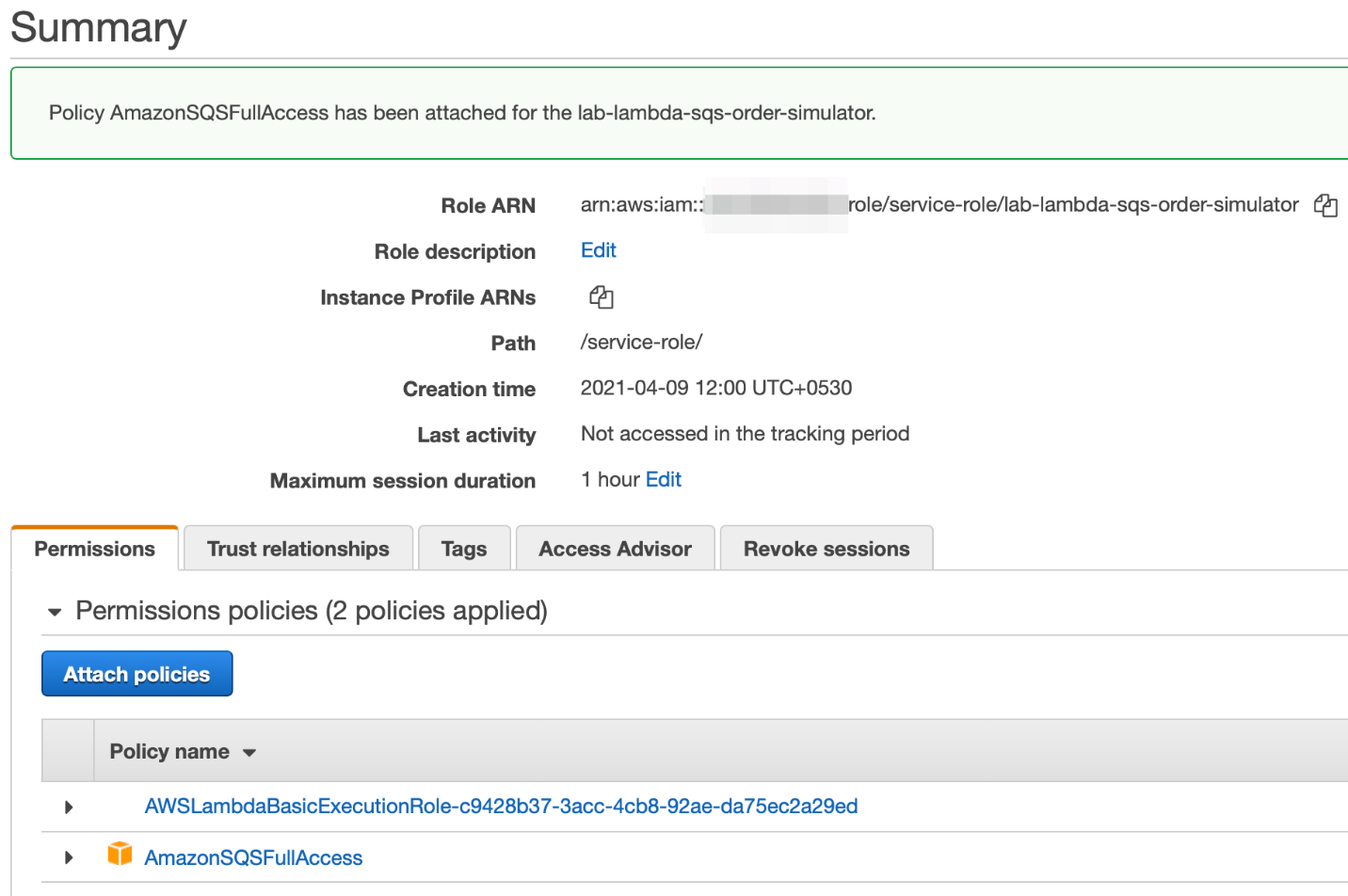
* 1. You will notice that only the **AWSLambdaBasicExecutionRole** policy is attached to the Lambda in the Permissions tab. Click on the **Attach policies**button.



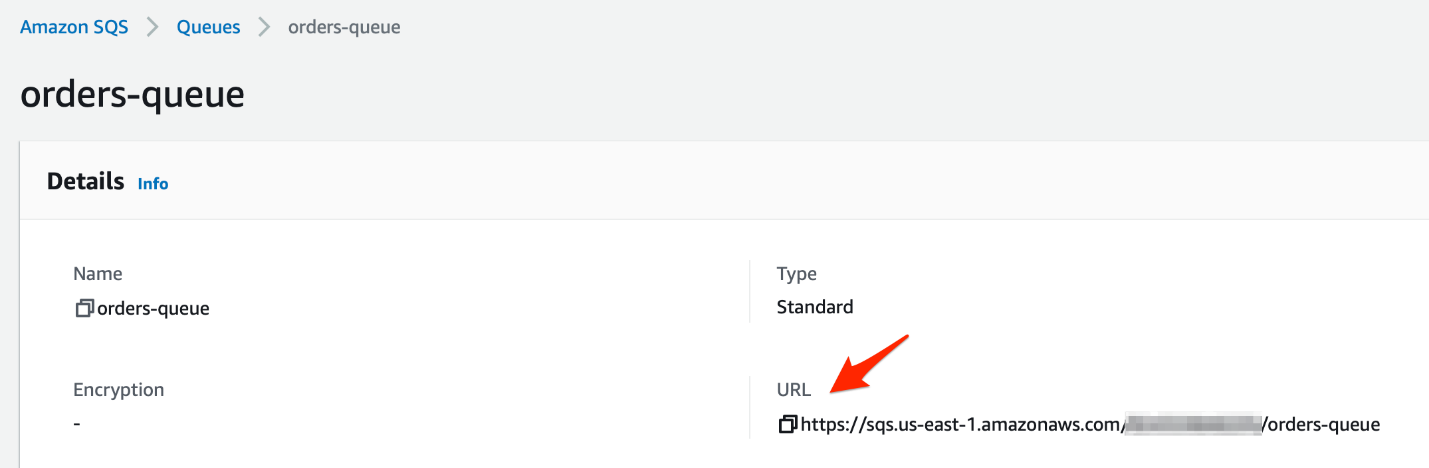
* 1. Enter **SQS** in the search bar and select the policy **AmazonSQSFullAccess** for the Lambda so that it can write order messages into the SQS queue. Click on the **Attach policy** button to add this policy.



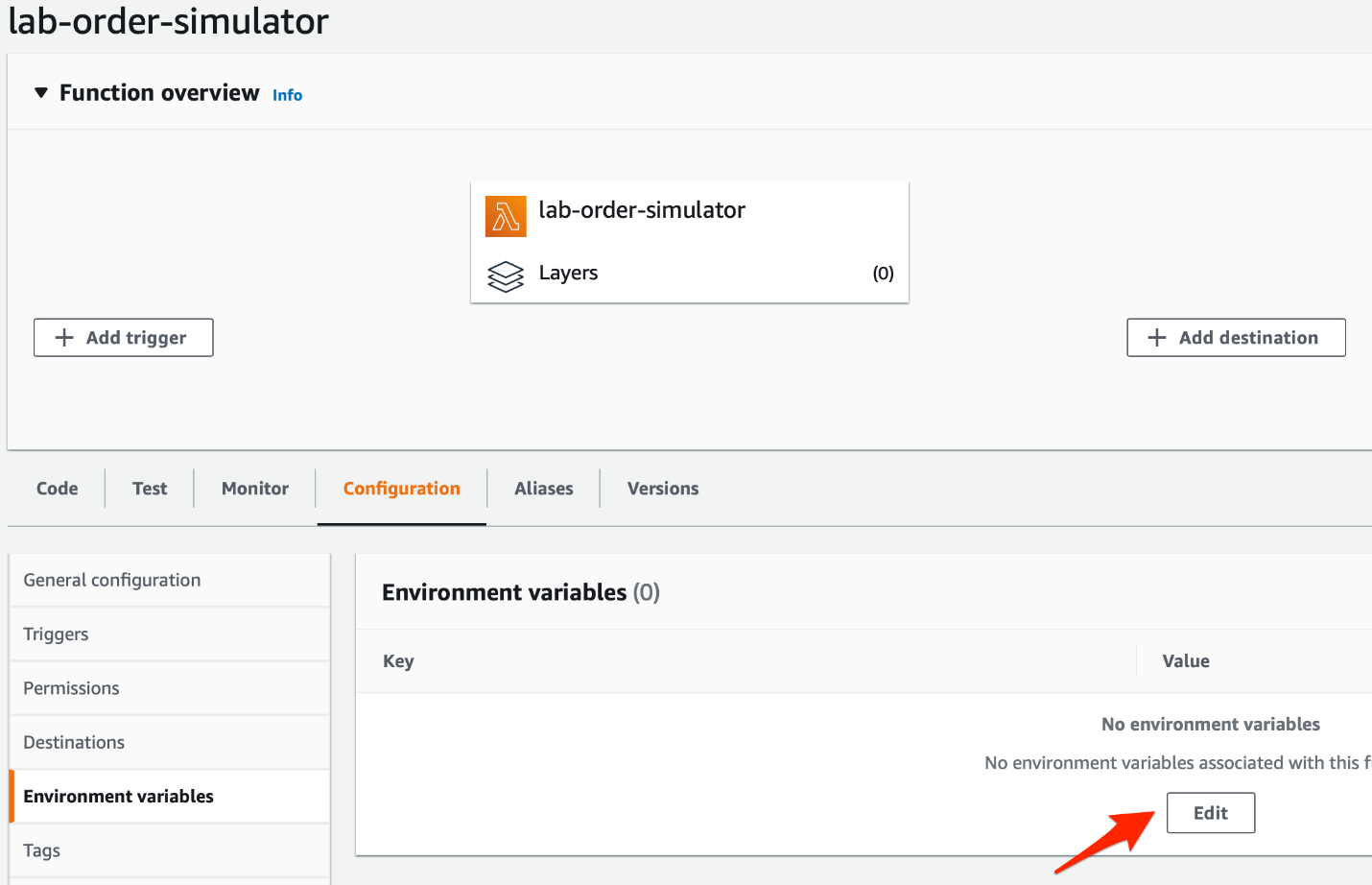
* 1. You should see a confirmation message that says that the **AmazonSQSFullAccess**policy has been attached to the Lambda



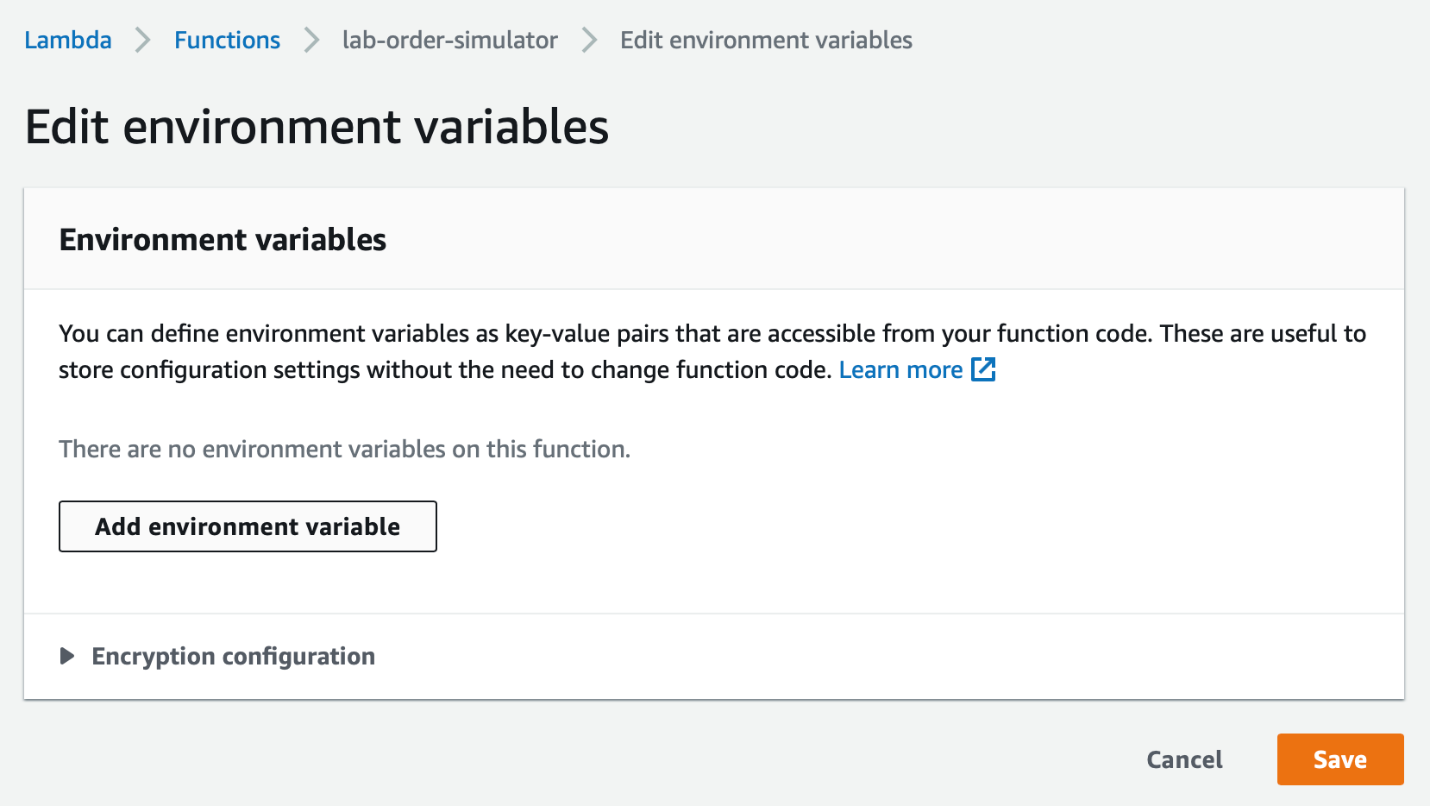
1. Configure the Lambda to set up the environment variable for the target SQS queue.
   1. Go to the SQS queue **orders-queue** created in Task 1 and copy the queue URL



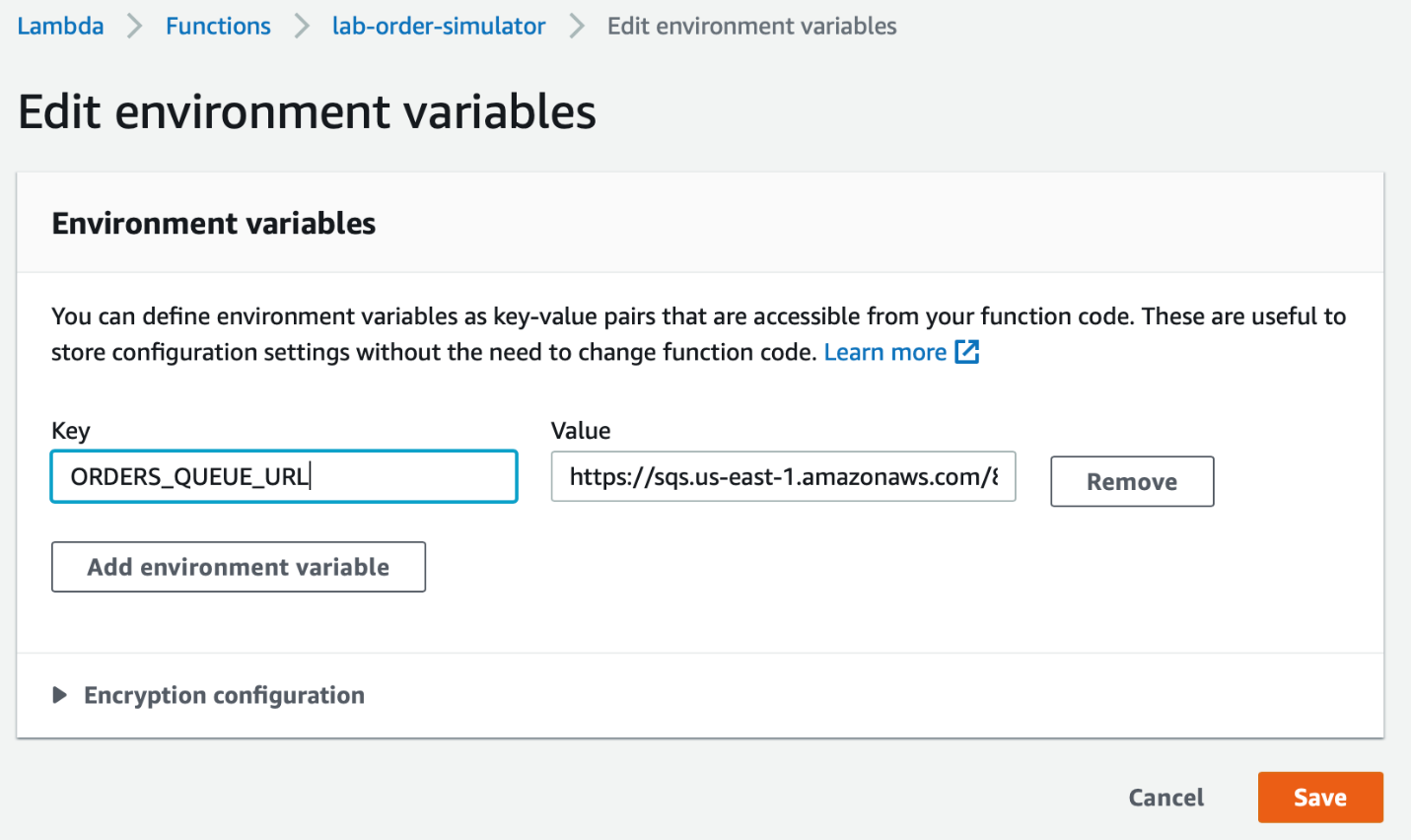
* 1. Select the **Configuration** tab and click on the **Environment variables** link and then click on the **Edit** button.



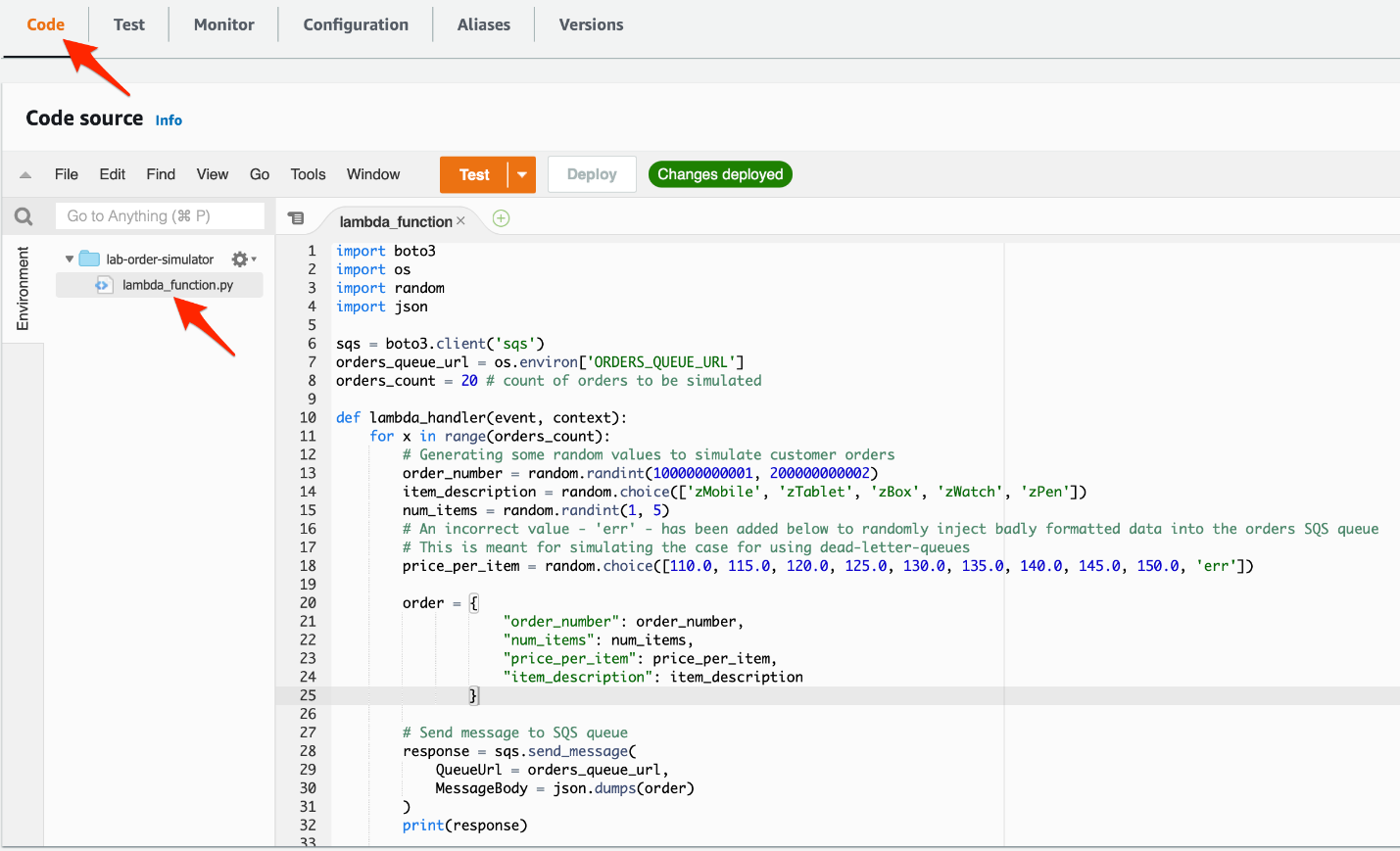
* 1. Click **Add environment variable**



* 1. Enter **ORDERS\_QUEUE\_URL** as the **Key.**Paste the **orders-queue**URL that you copied into the **Value**field. Click on **Save**.



1. Click on the **Code** tab and then select the lambda\_function.py file. Now copy the code from the **lab-order-simulator.py** file provided in the assets and paste in the Lambda function code editor and click **Deploy**.



Ref lab-order-simulator.py

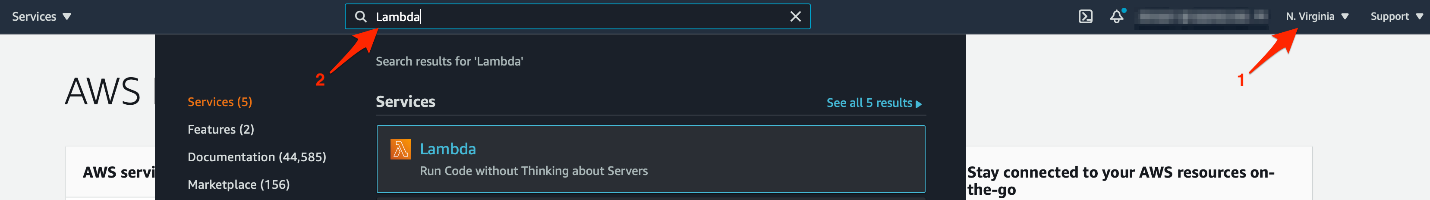
<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>

<https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html>

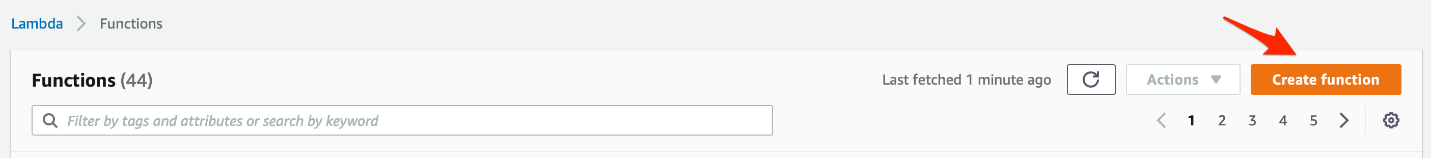
Create a Lambda function to process orders into Dynamodb table

You need to create a Lambda function that processes each SQS message containing the order data. You also need to configure the Lambda function to use the customer orders queue as the event source. The Lambda function will complete the order processing by writing the order data into a DynamoDB table.

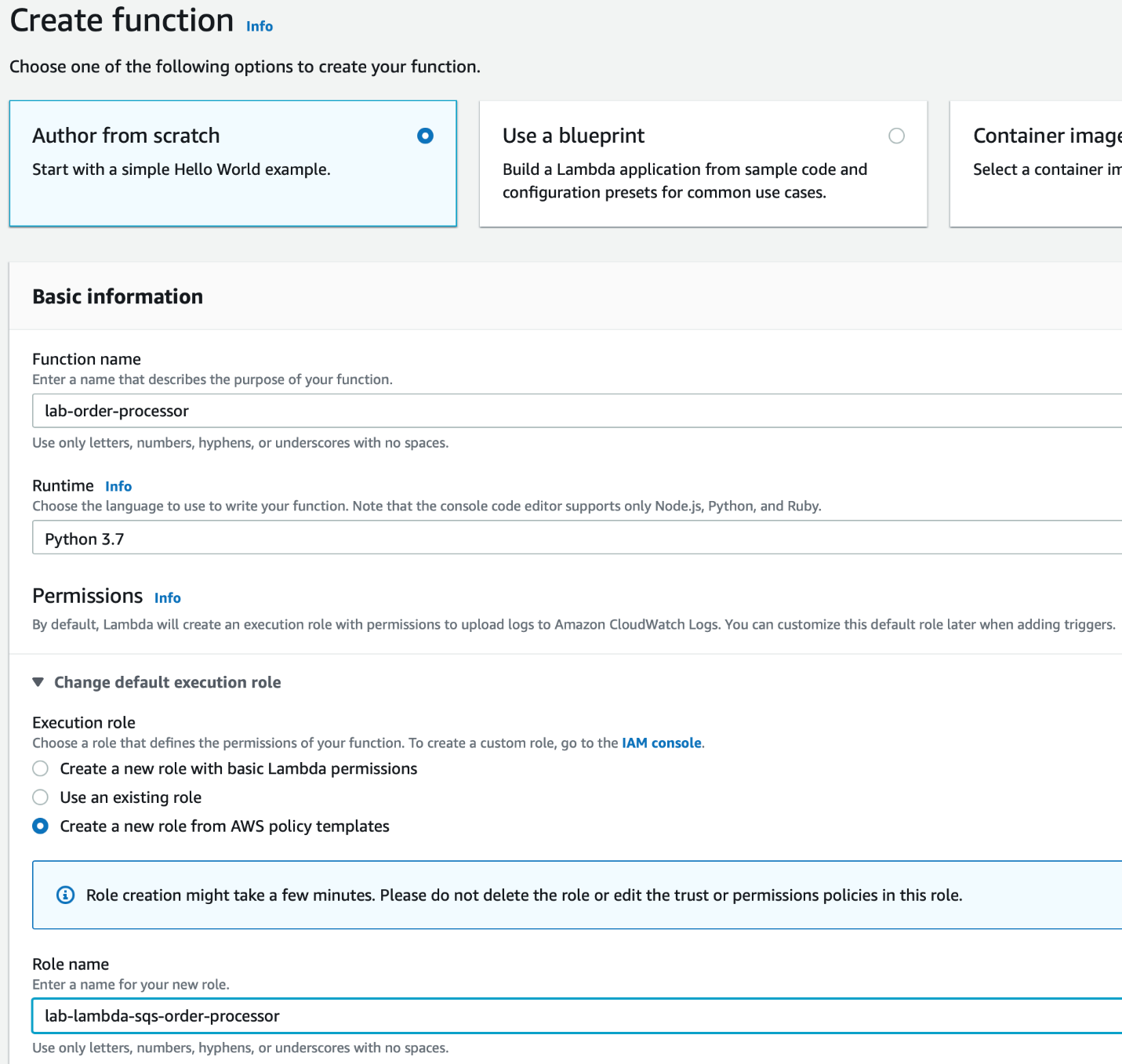
1. Make sure that you are in the N.Virginia AWS Region on the AWS Management Console. Enter **Lambda** in the search bar and select **Lambda**service.



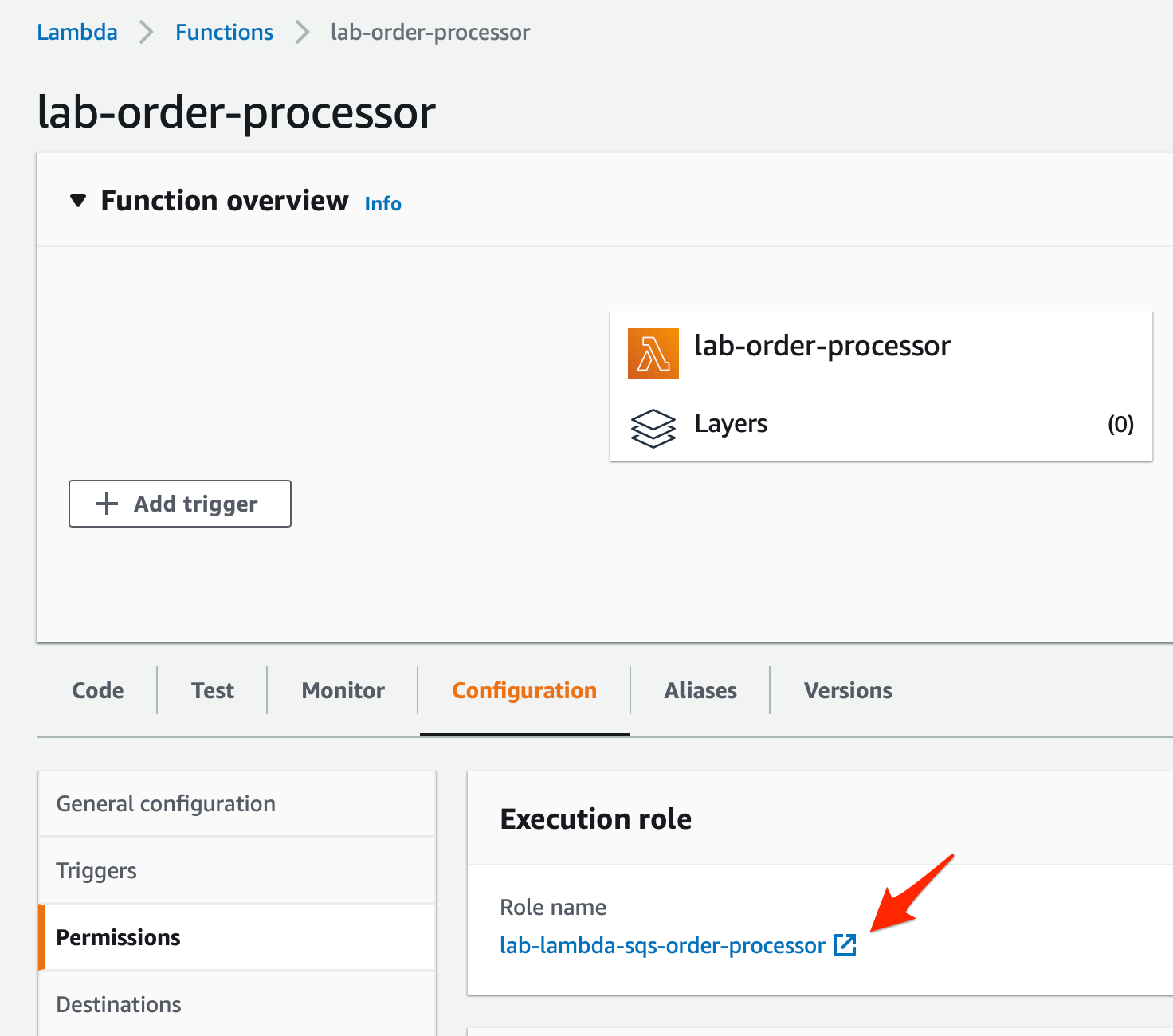
Click on **Create function** button



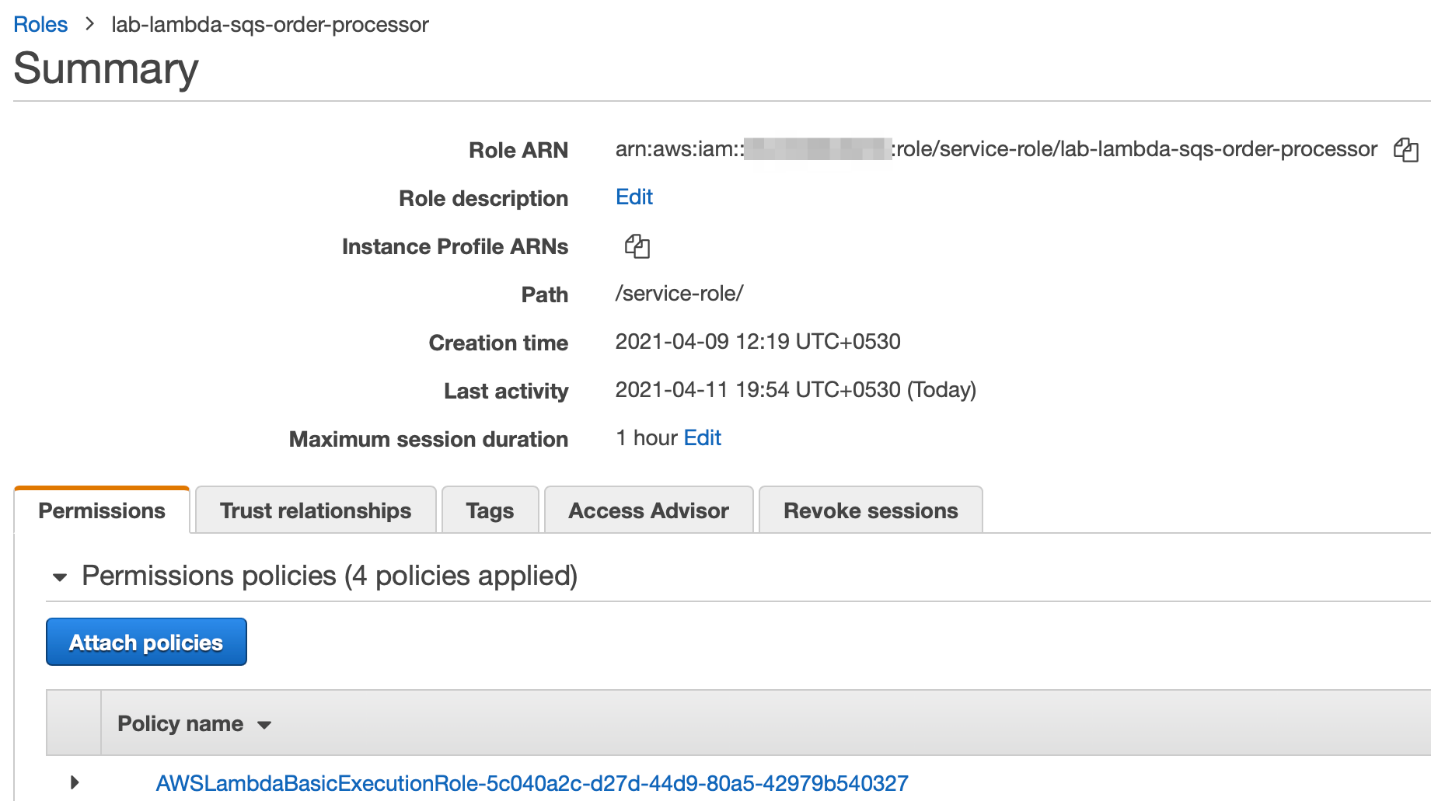
1. Lets create the Lambda function by filling in the following values:
   1. Select the **Author from scratch** template
   2. Function name: **lab-order-processor**
   3. Runtime: **Python 3.7**
   4. For execution role, select the option that says **Create a new role from AWS policy templates**
   5. Role name: **lab-lambda-sqs-order-processor**
   6. Click **Create function** to complete this step



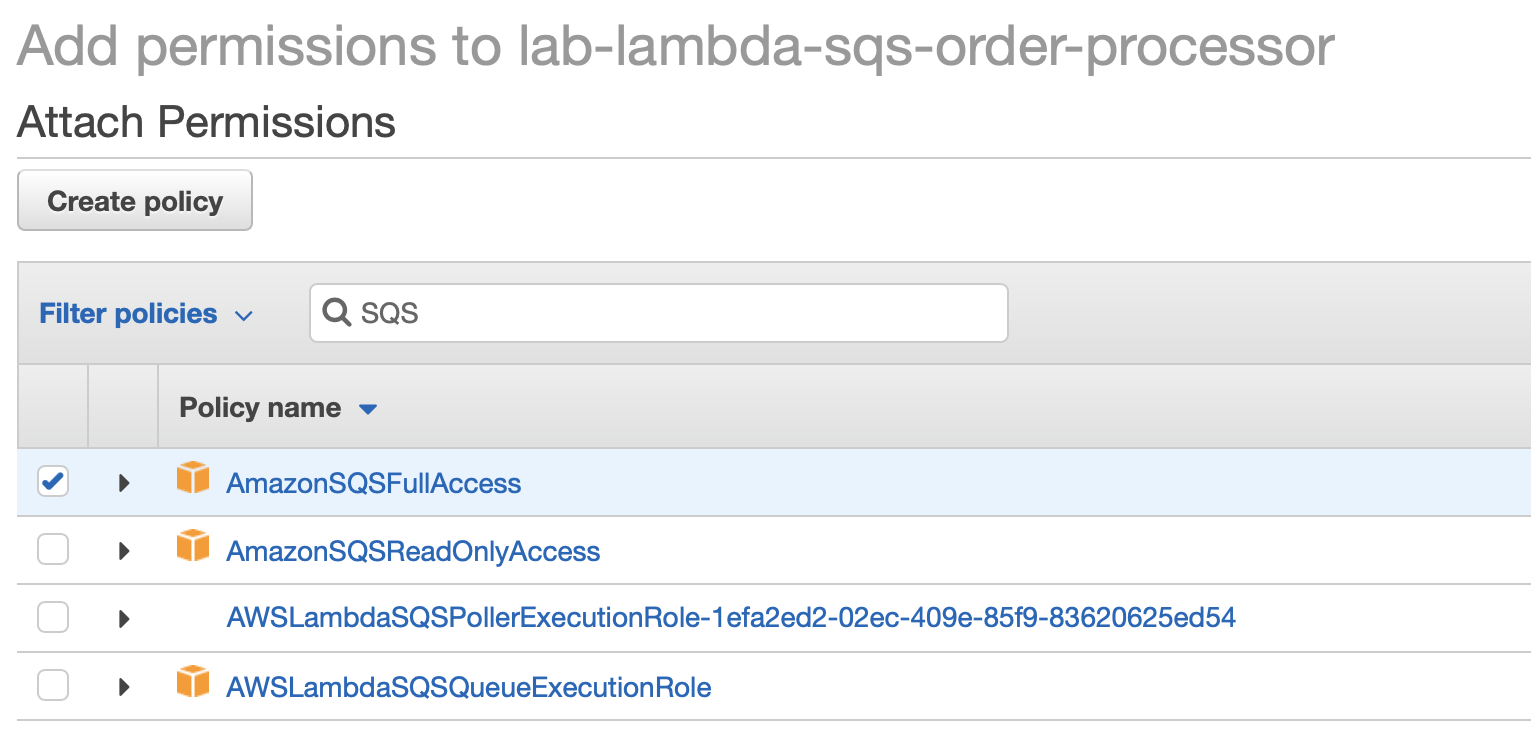
1. Configure the Lambda to modify the permissions for its execution role.
   1. Select the **Configuration** tab and click on the **Permissions** link on the left sidebar. Then, click on the hyperlink for the execution role.



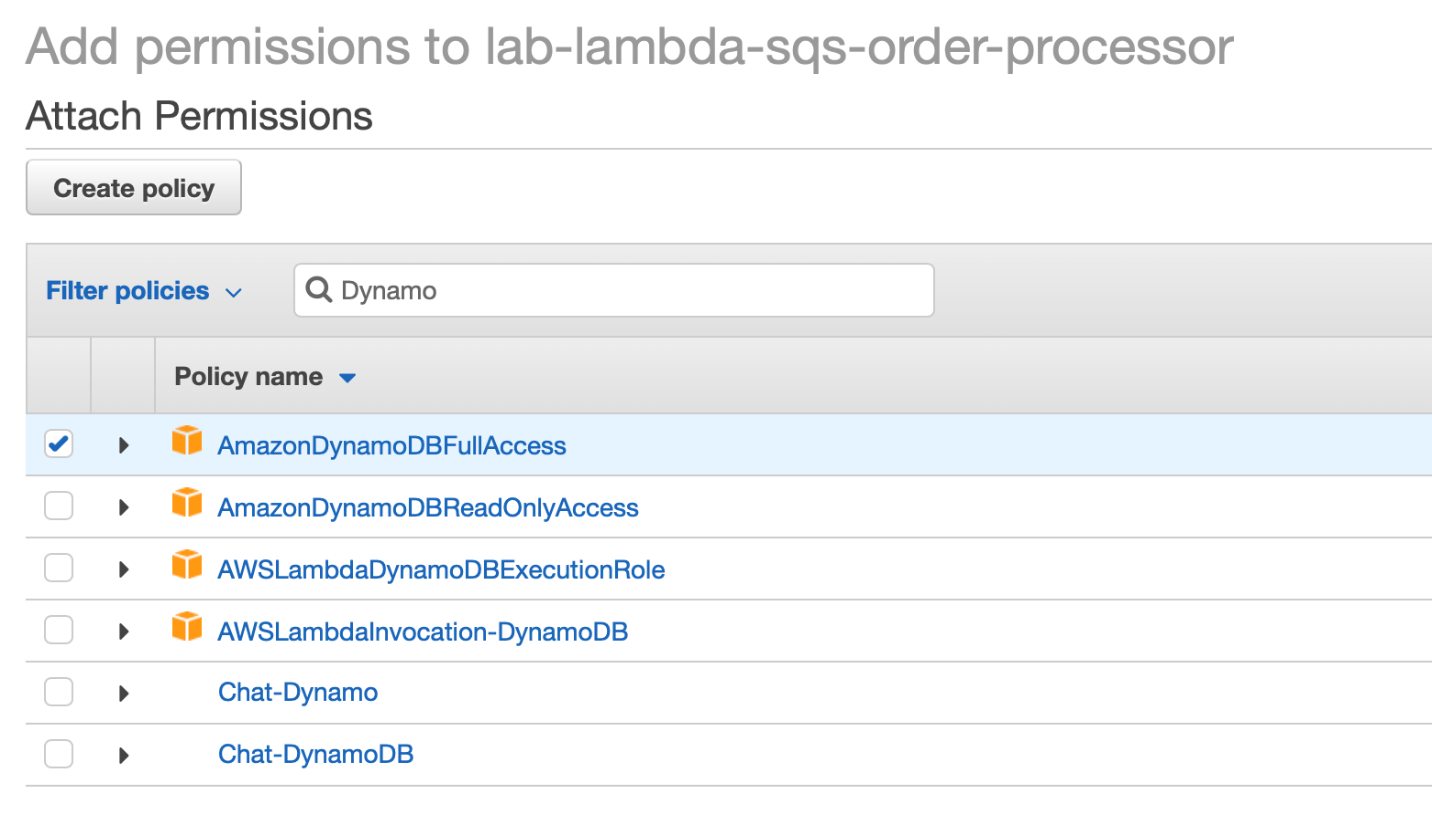
* 1. You will notice that only the **AWSLambdaBasicExecutionRole** policy is attached to the Lambda in the Permissions tab. Click on the **Attach policies**button.



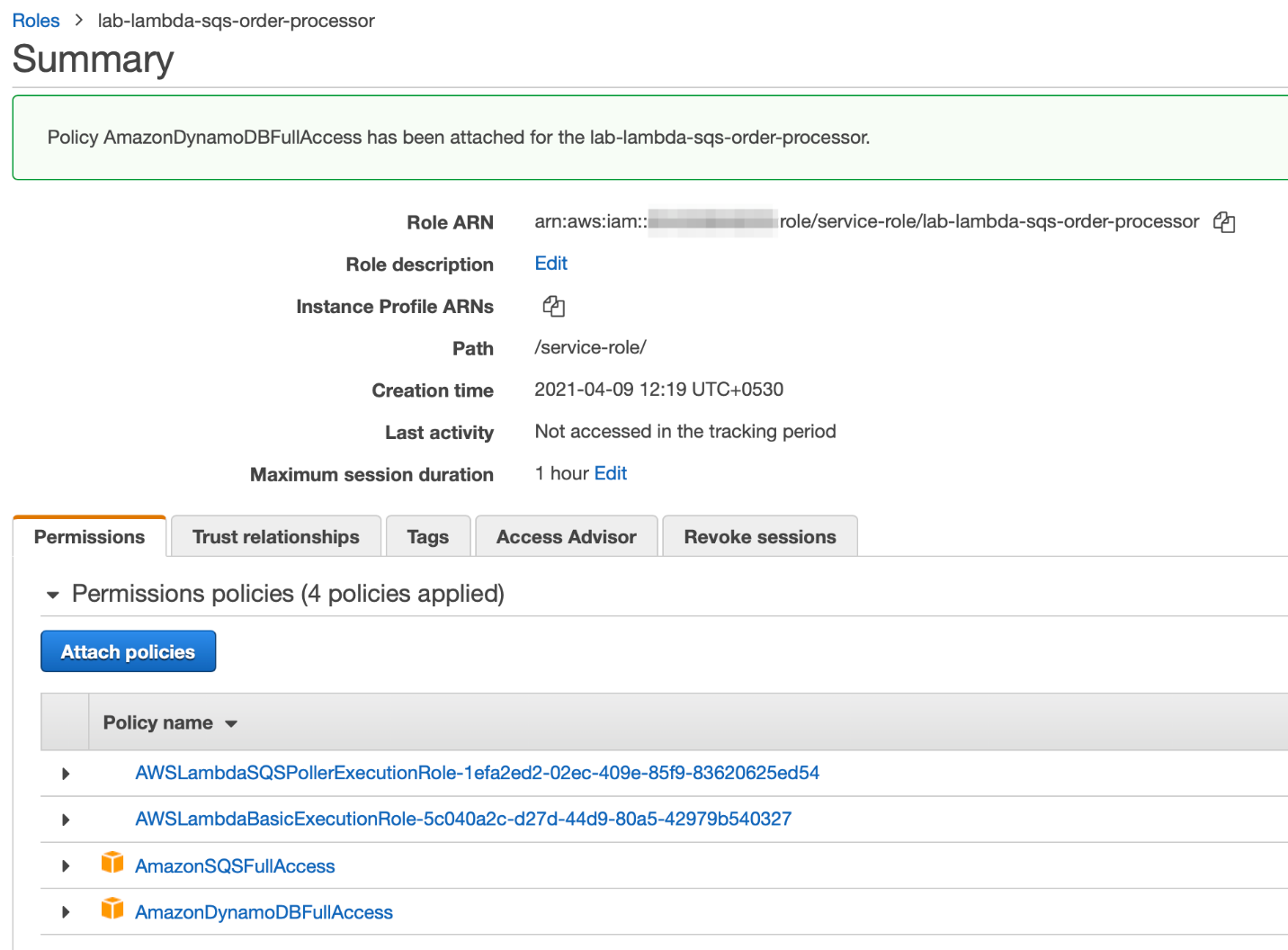
* 1. Enter **SQS** in the search bar and select the policy **AmazonSQSFullAccess** so that the queue can trigger the AWS Lambda function. Click on the **Attach policy** button to add this policy.



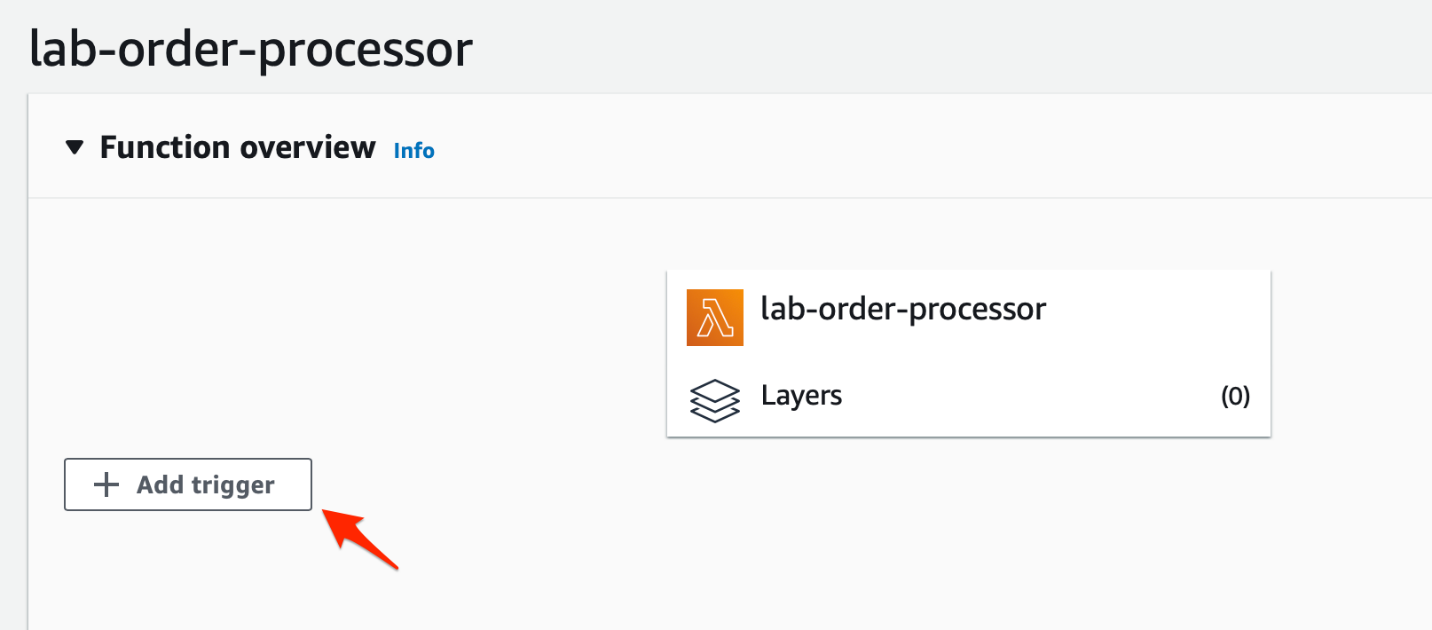
* 1. On the Permissions tab, click on the **Attach policies**button again and enter **DynamoDB** in the search bar and select the policy **AmazonDynamoDBFullAccess** for the Lambda so that it can write the order data into the DynamoDB table. Click on the **Attach policy** button to add this policy.



* 1. You should be able to see that all policies are attached to the Lambda function



1. Configure the Lambda to set up a trigger from SQS
   1. Go to the **Function overview** section of the Lambda and click on **Add trigger**



* 1. Set the values for the Trigger configuration as follows:

Trigger source: **SQS**

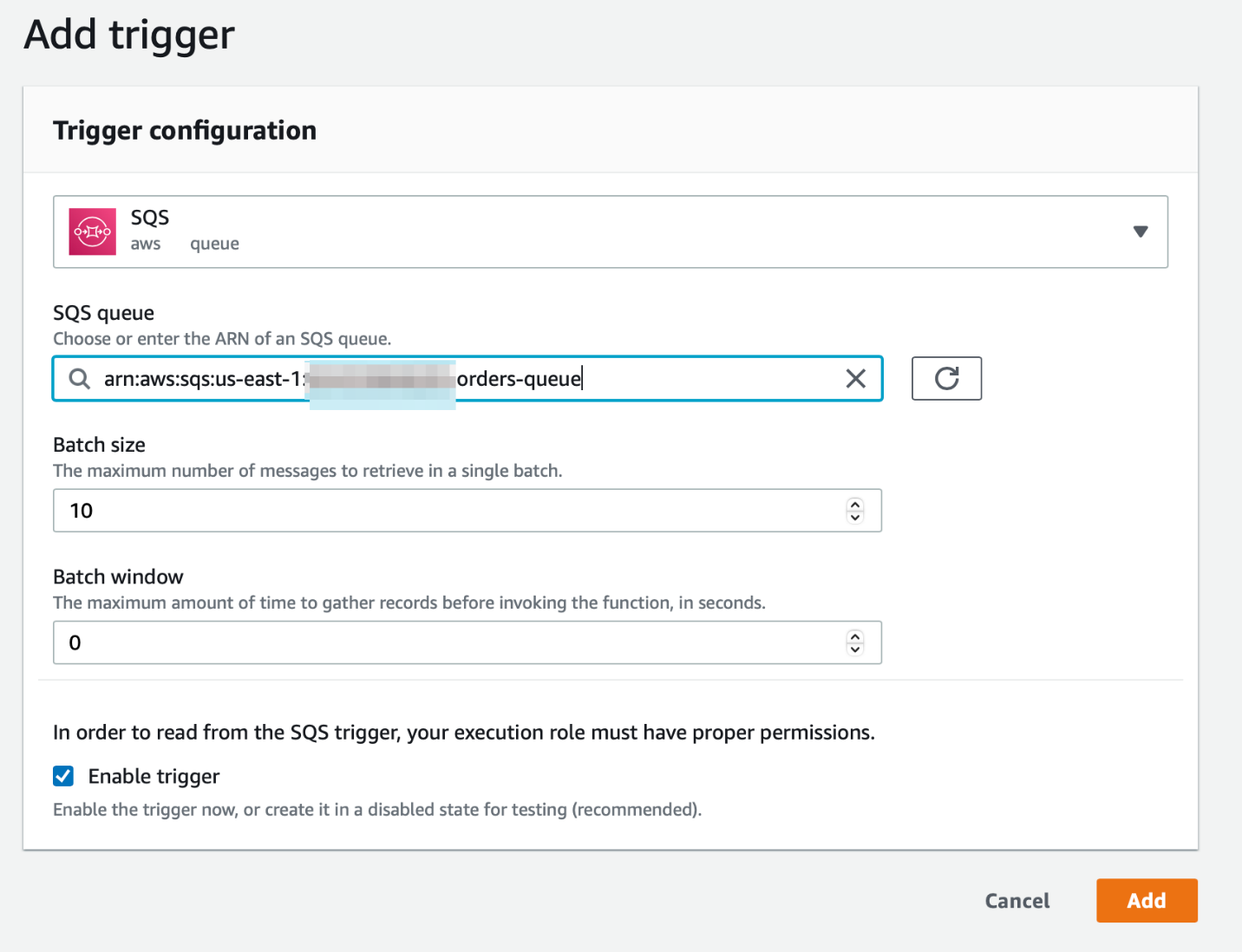
SQS queue: select the **orders-queue**as the source queue

Batch size: **10**

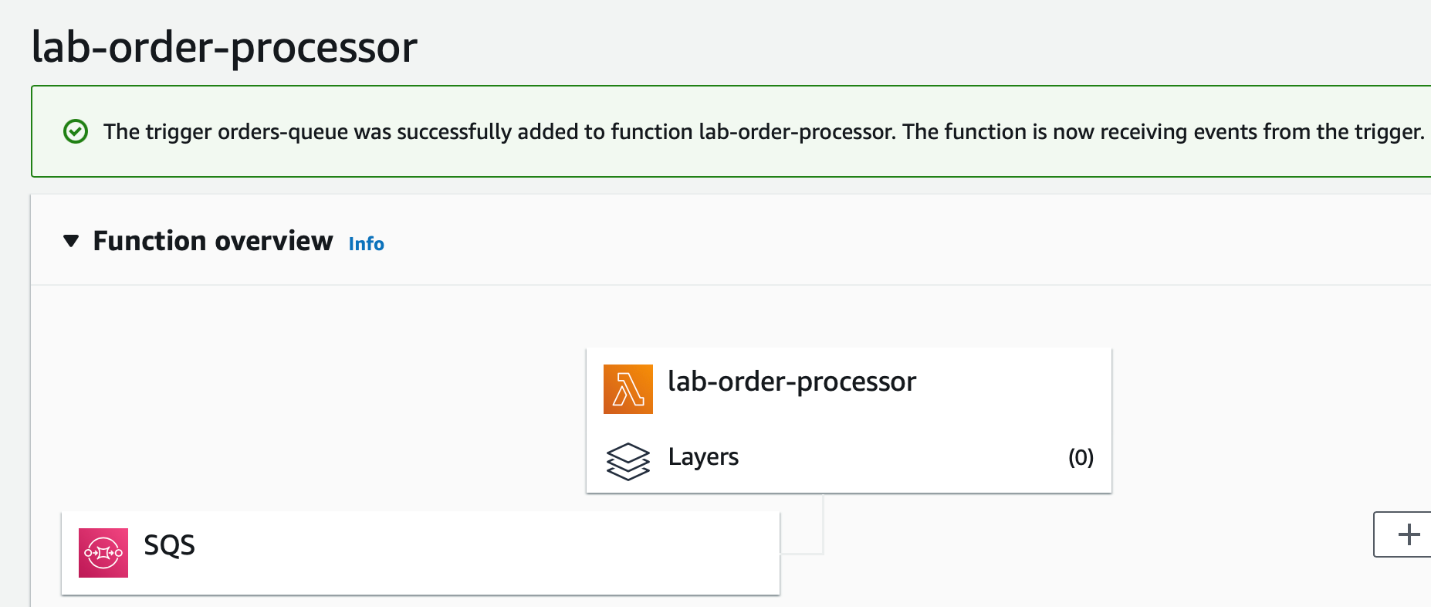
Batch window: **0**

Select the **Enable trigger** checkbox

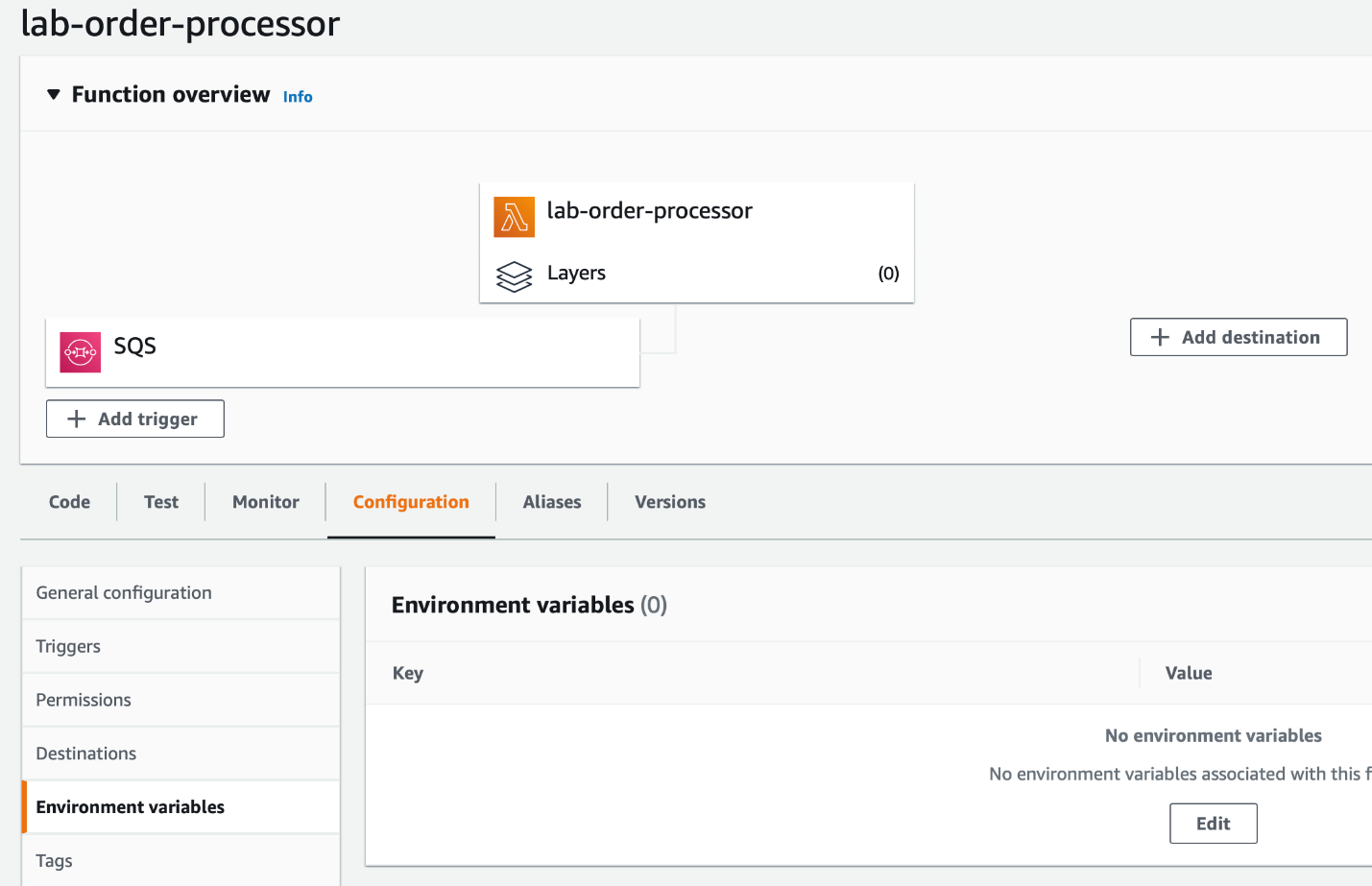
Click on **Add** to complete this step



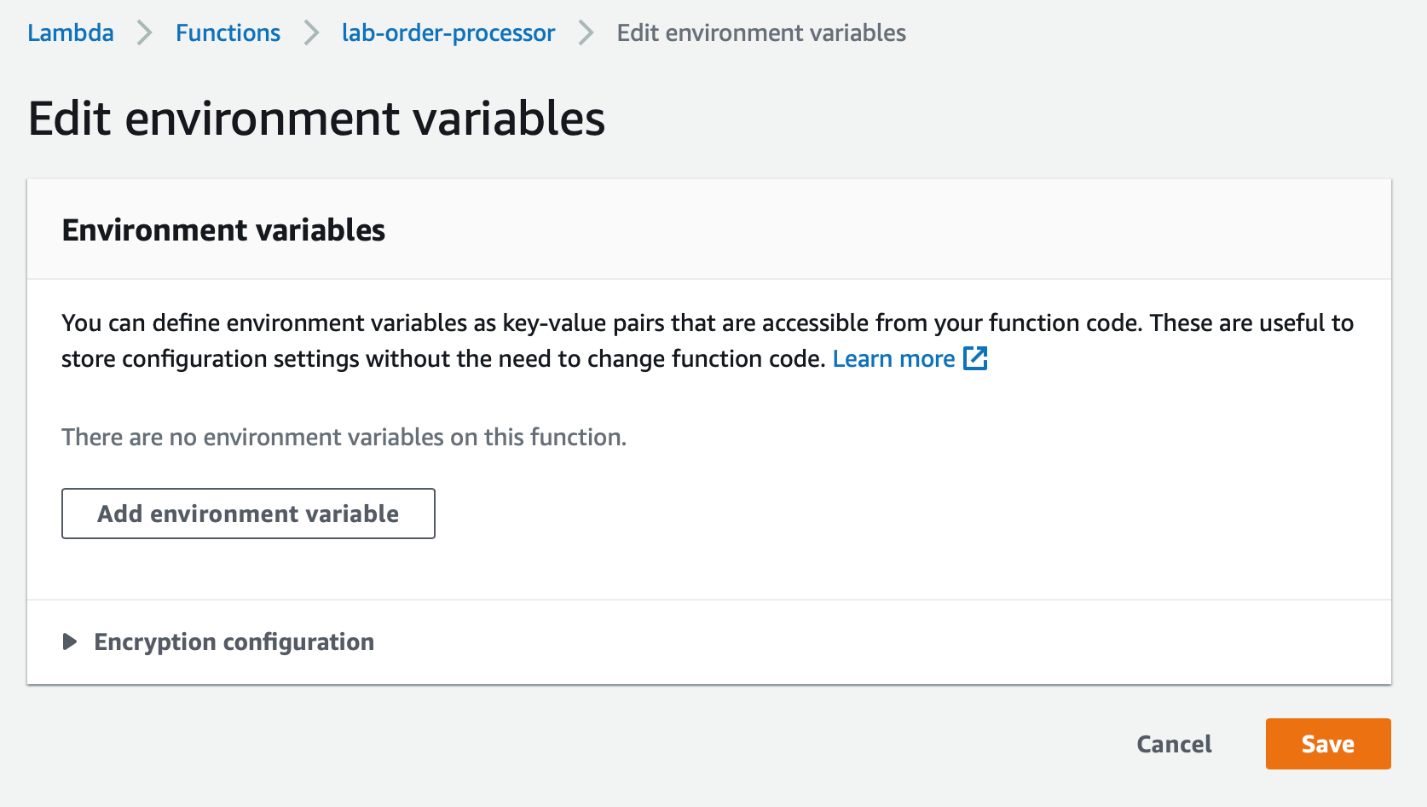
* 1. You should be able to see that trigger has been set up successfully



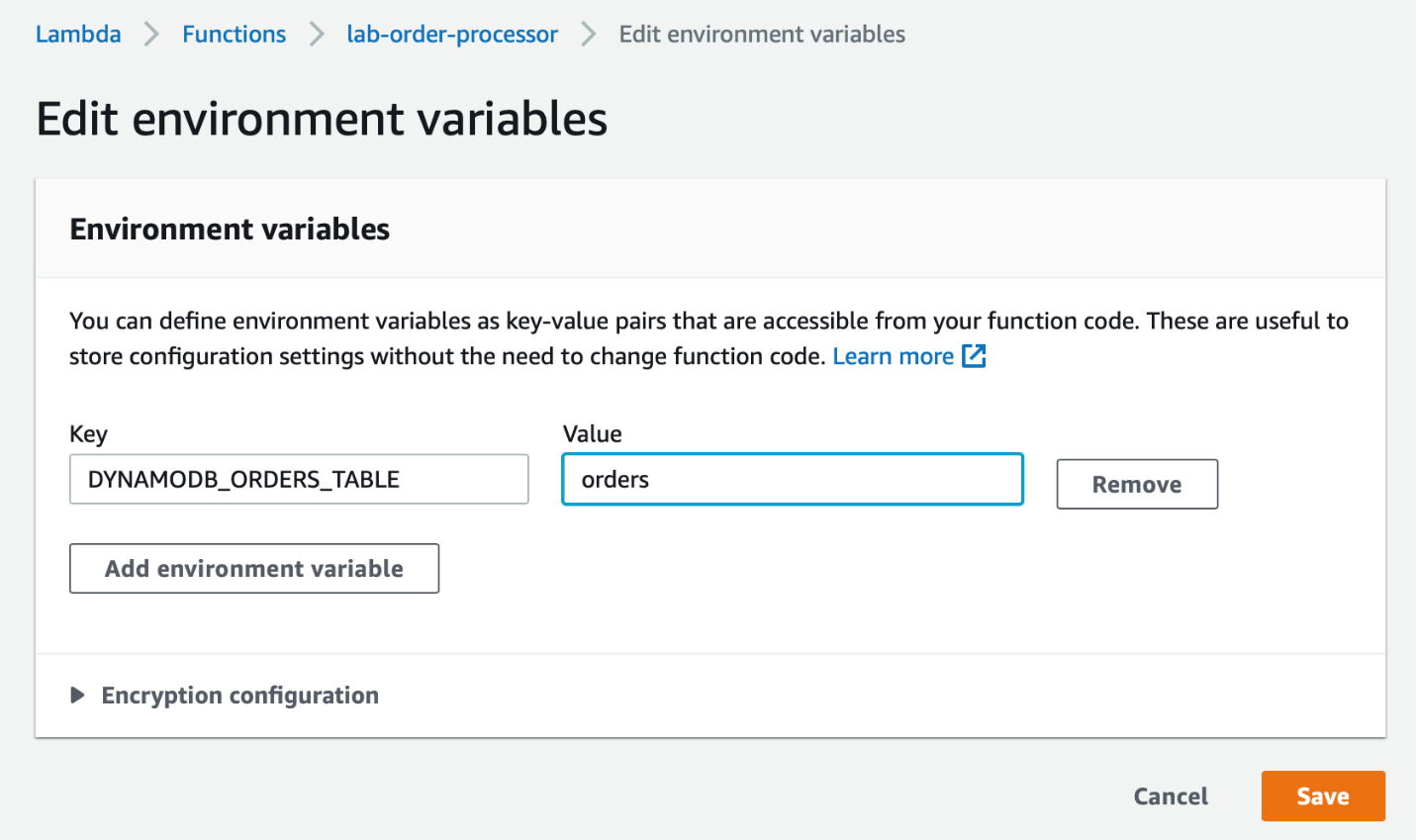
1. Configure the Lambda to set up the environment variable for the DynamoDB table used to store the order data.
   1. Click on the **Configuration** tab and select the **Environment variables** link in the left sidebar. Click on the **Edit** button.



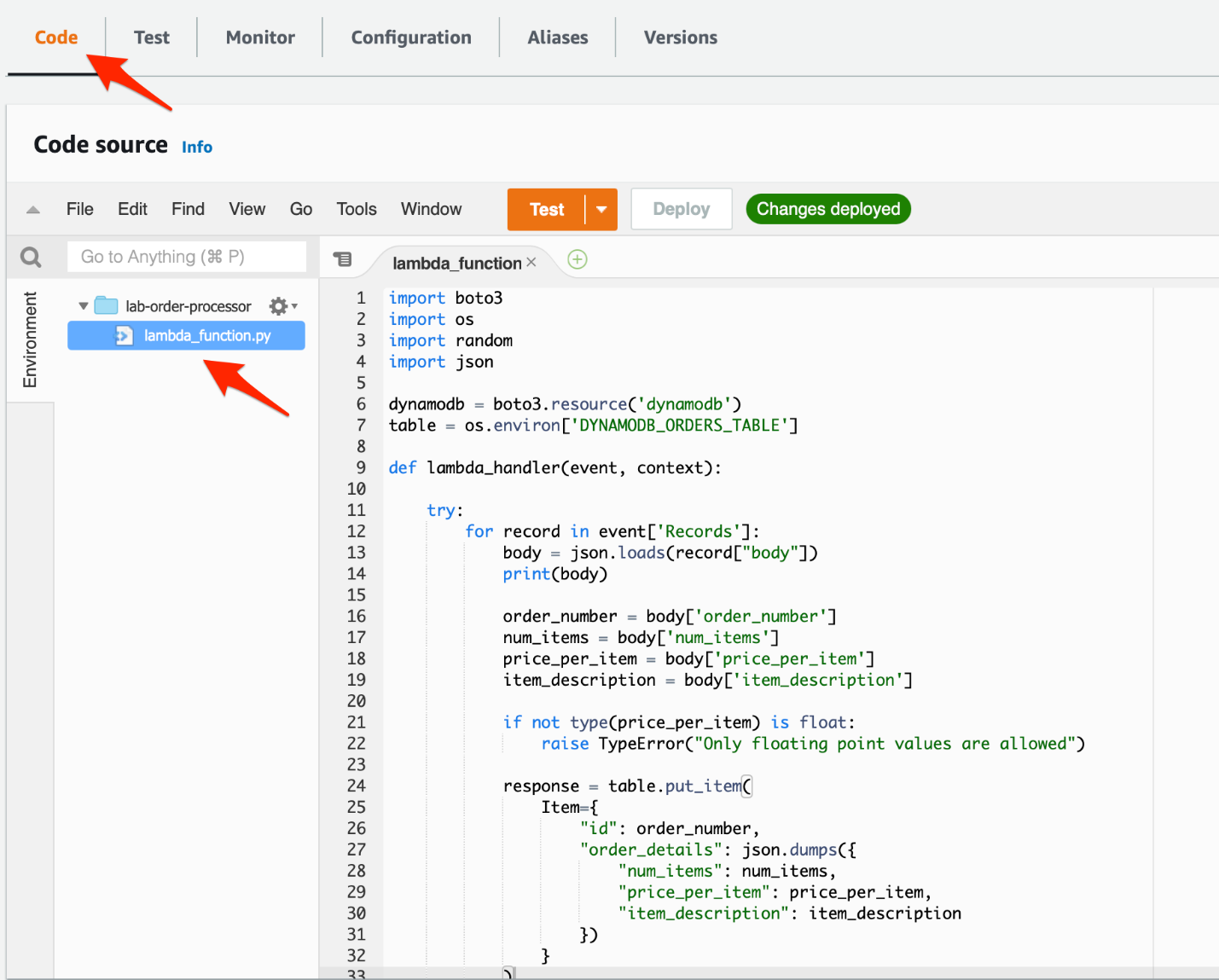
* 1. Click on the **Add environment variable** button.



* 1. Enter **DYNAMODB\_ORDERS\_TABLE** as the **Key.**Put **orders**as the **Value**. Click on **Save**.



1. Click on the **Code** tab and then select the lambda\_function.py file. Now copy the code from the **lab-order-processor.py** file provided in the assets and paste in the Lambda function code editor and click **Deploy**.



Ref lab-order-processor.spy

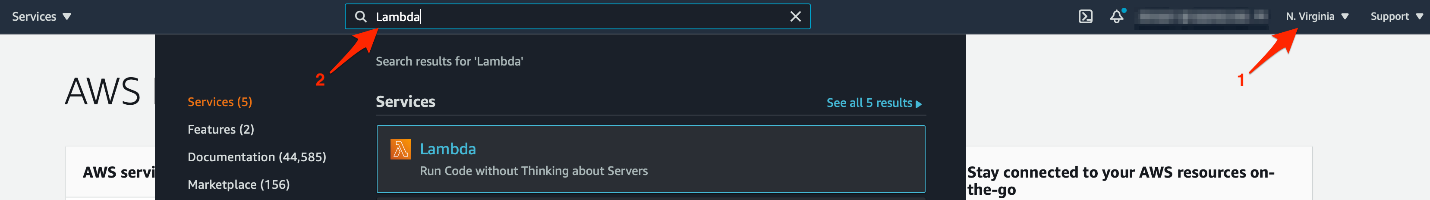
<https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html>

<https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html>

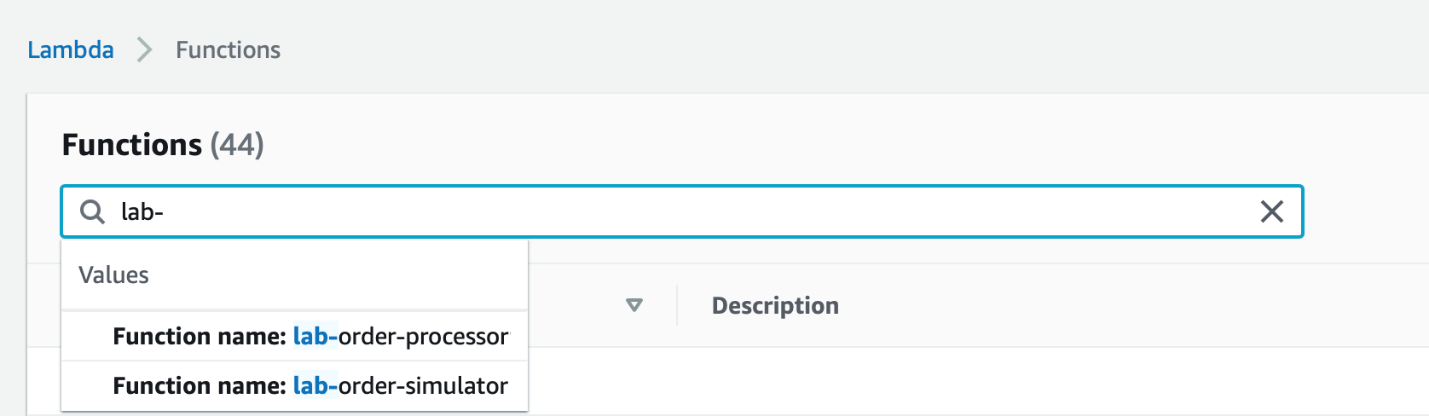
Validate the end-to-ed decoupled backend architecture

To test your end-to-end architecture, you need to create customer orders by executing the order simulator Lambda (created in Task 3) which puts customer orders into the SQS queue. This, in turn, triggers the order processor Lambda (created in Task 4) that finally captures the customer order data in a DynamoDB table. Thus, the given architecture decouples the order-creating component with the order-processing component. The architecture can also handle any missing data or badly formatted data without breaking the workflow.

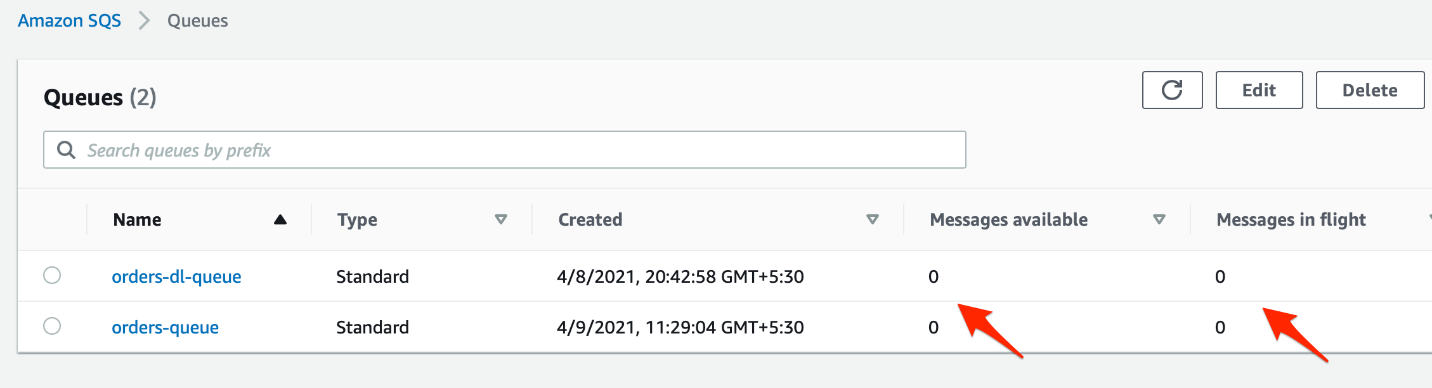
1. Make sure that you are in the N.Virginia AWS Region on the AWS Management Console. Enter **Lambda** in the search bar and select **Lambda**service.

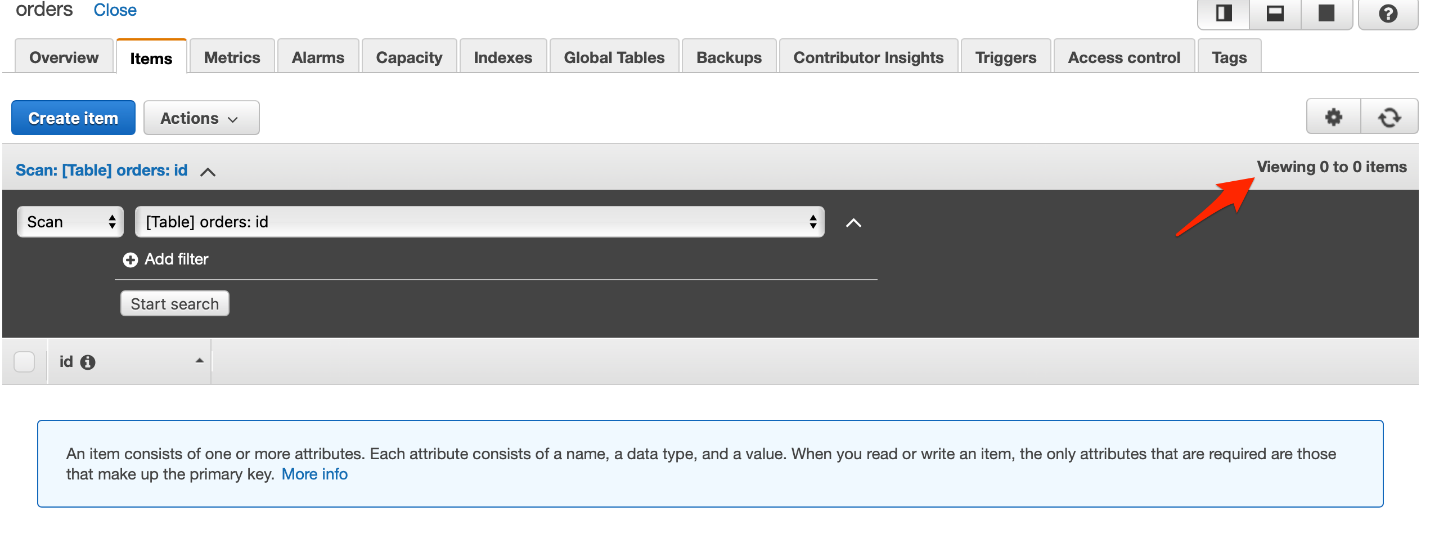


Search and select the **lab-order-simulator** lambda

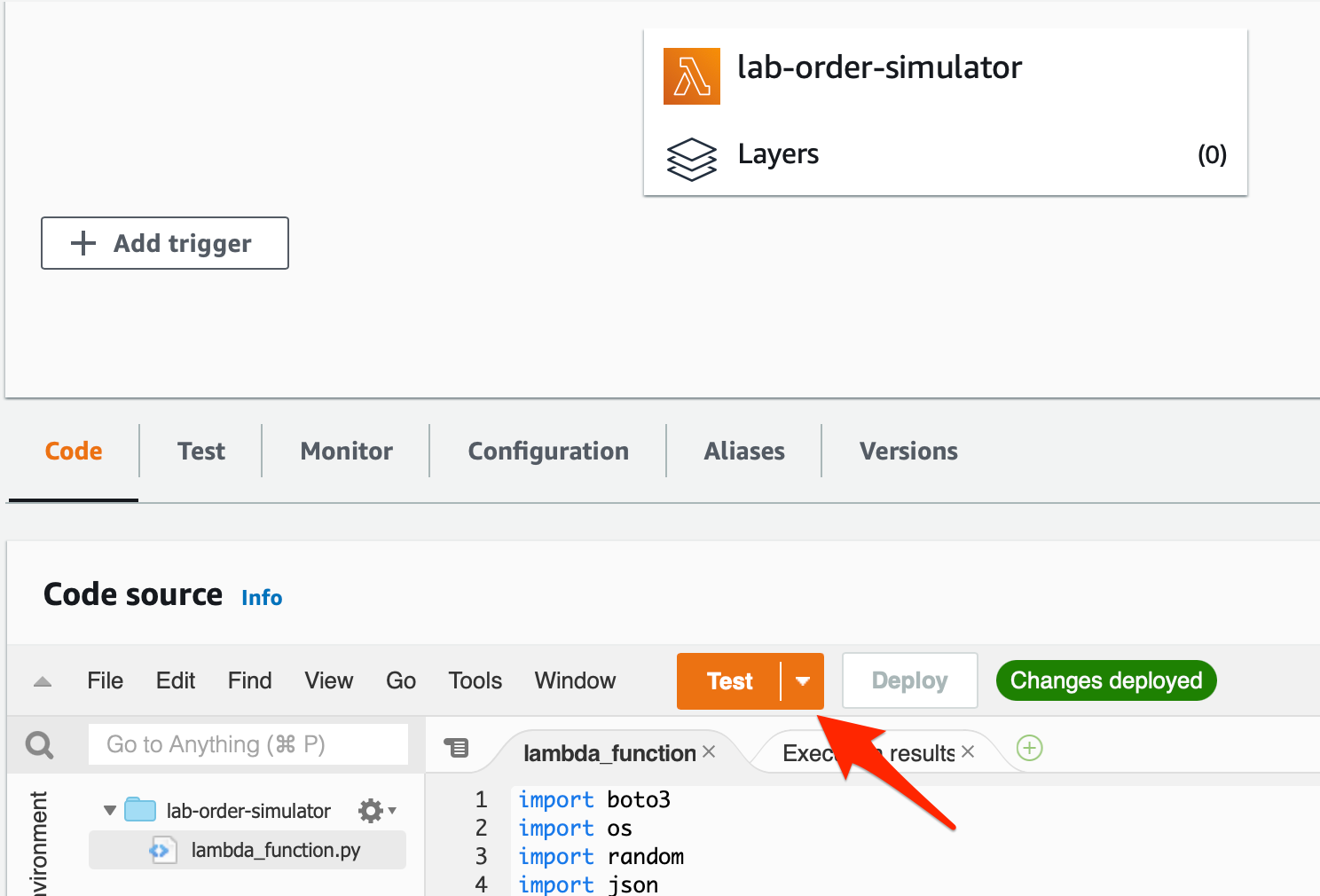


1. Before creating the orders, lets make sure that there are no messages in the queues and the DynamoDB **orders** table has no records.

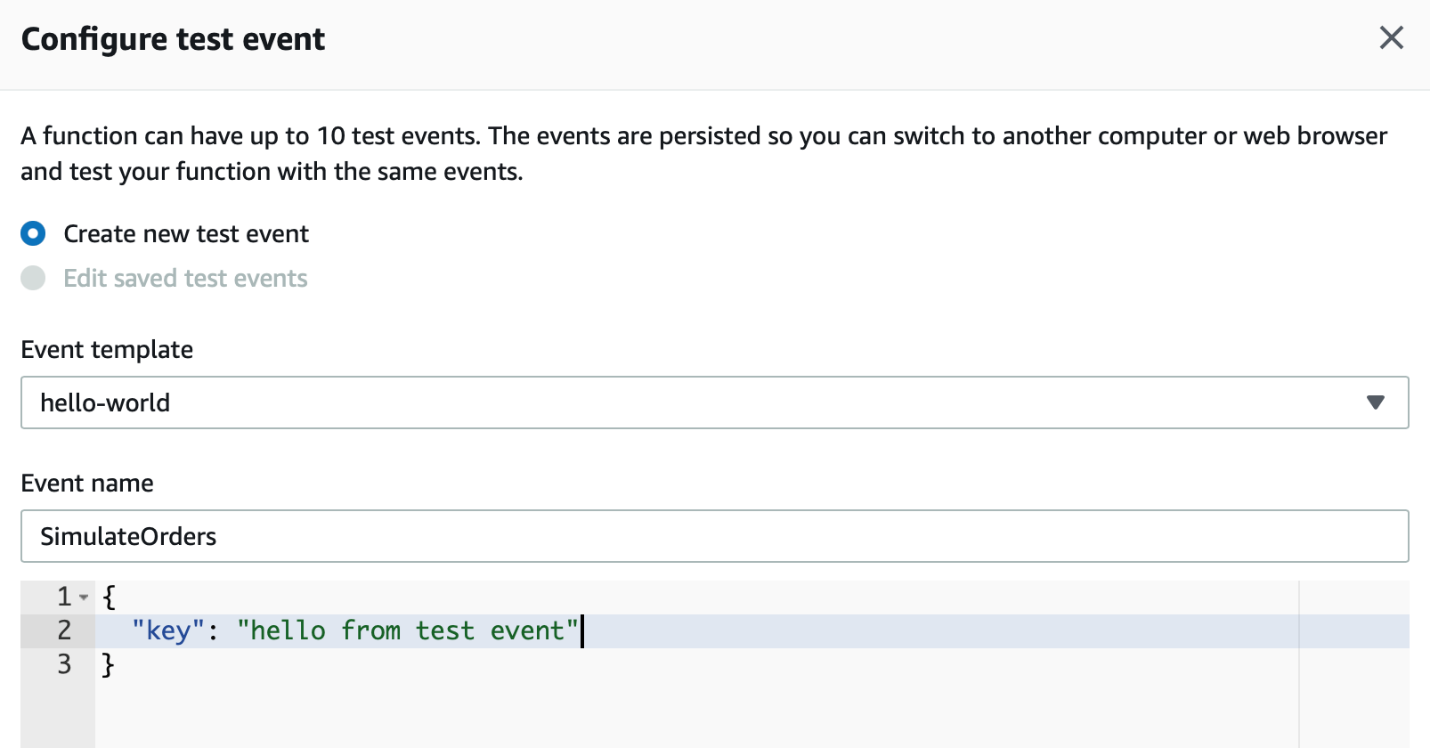




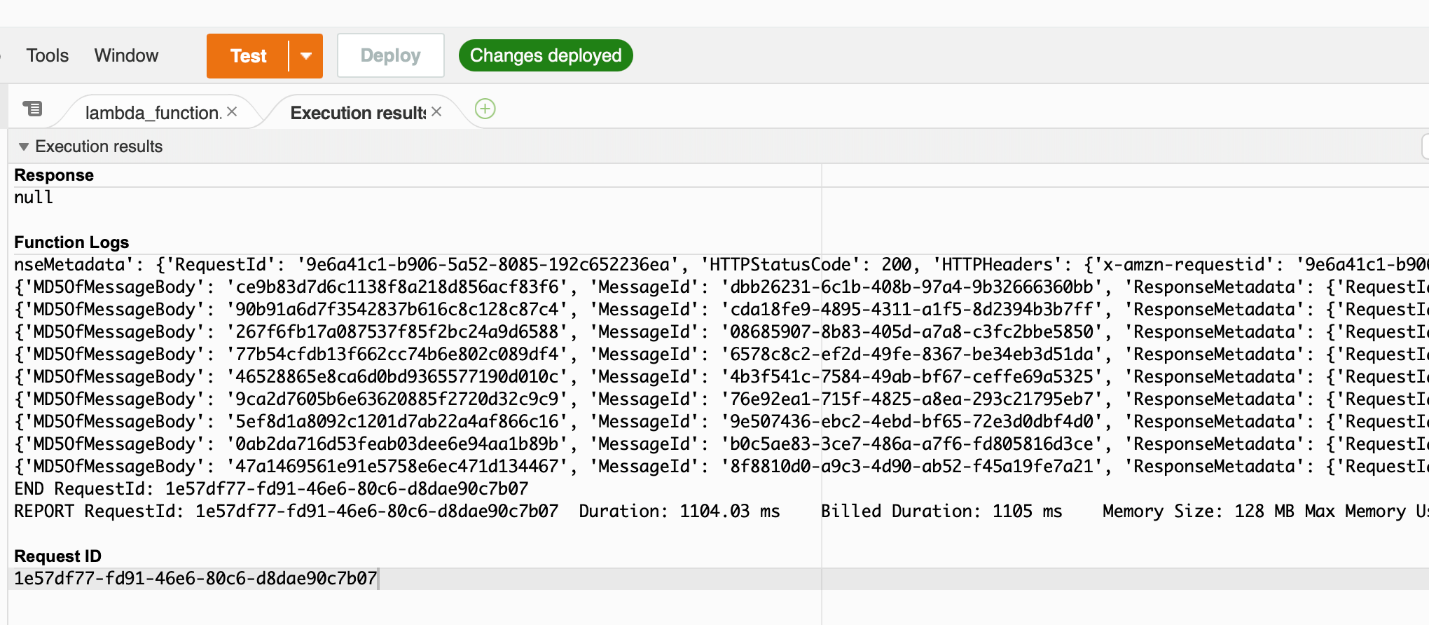
1. To invoke the **lab-order-simulator** lambda, you need to configure a test event. Click on the dropdown menu arrow next to **Test** and click on **Configure test event.**



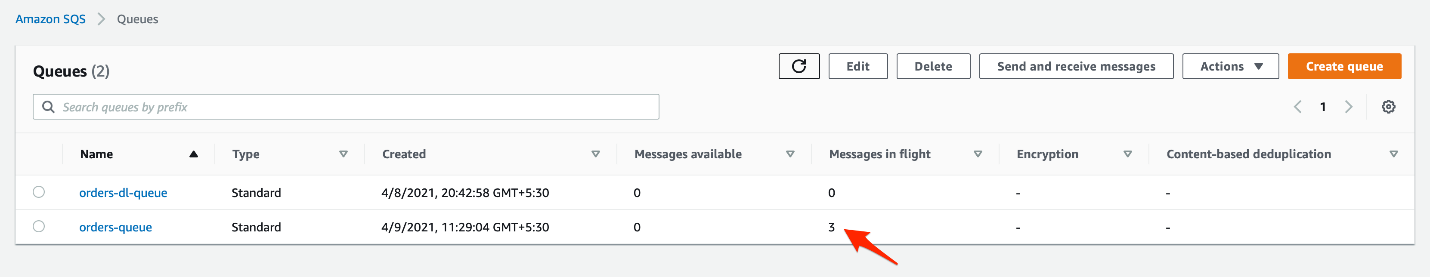
Select **Create new test event**and enter **SimulateOrders**as the event name. Provide a key and value as shown below. Click on **Create** to complete the set up.



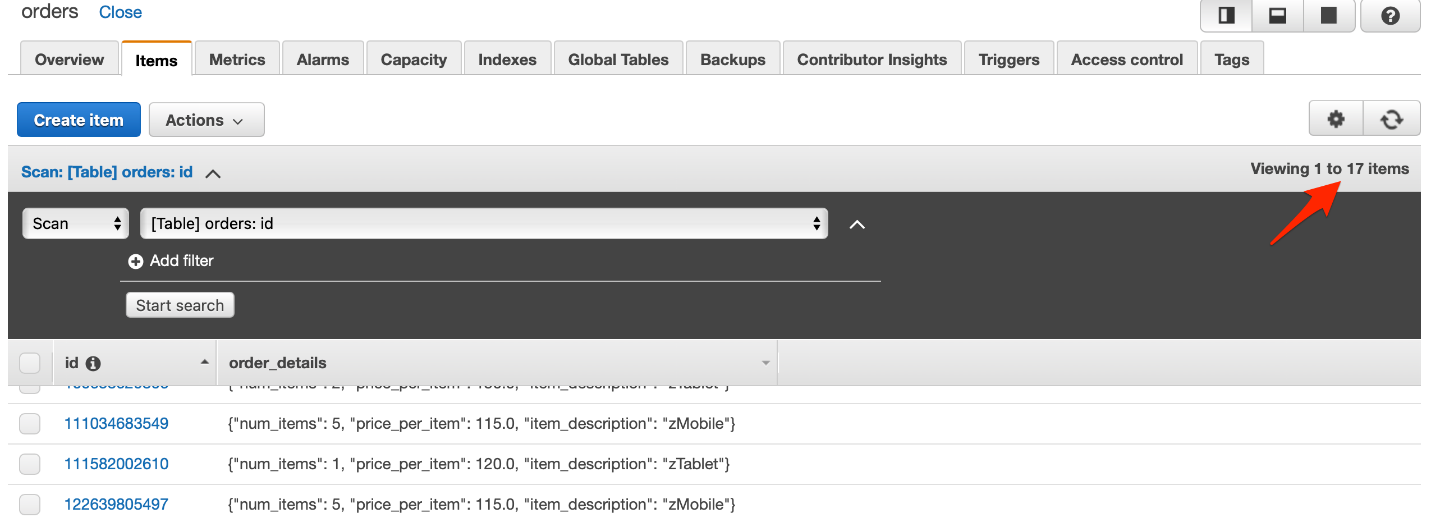
1. Now that the test event is ready, click on **Test**. You should notice the following:
   1. The execution results show that the Lambda ran successfully



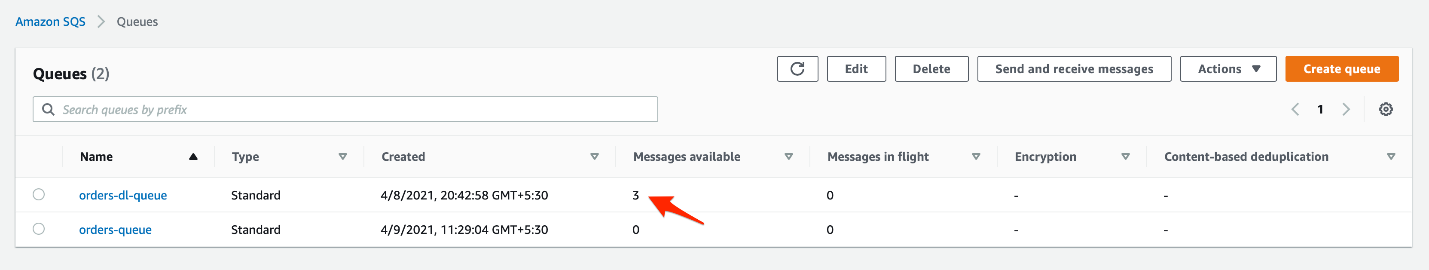
* 1. Go to SQS and check the summary stats for the queues. For this run, we see that the **orders-queue** shows there are 3 messages in flight. These are the 3 messages which have an err value as the price\_per\_item and therefore rejected by the order processor Lambda. *Please note that this number may be higher or lower for you as the value err is randomly generated.*We will look at the actual messages in a later step



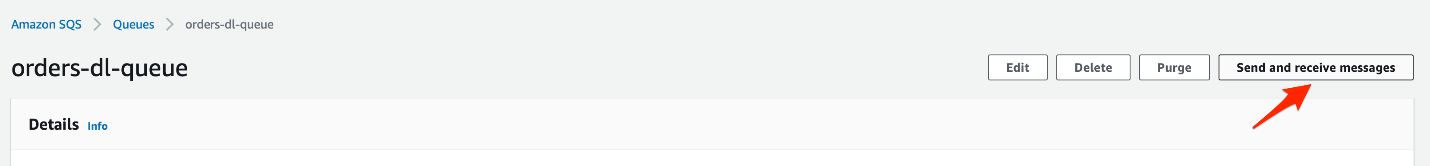
* 1. Go to DynamoDB and check the record count in the orders table. For this run, we see that the **orders** table shows that there are 17 records. This makes sense, as 3 out of the 20 records created were erroneous. *As mentioned earlier, these numbers may vary a little for you.*



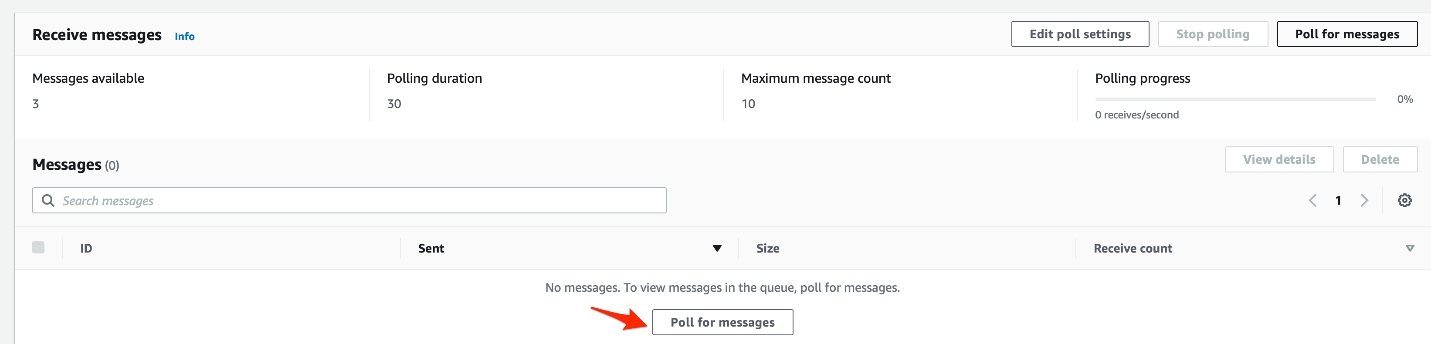
* 1. Lets go back to SQS and within a couple of minutes, you should see that the 3 erroneous in-flight messages for the **orders-queue** are now available in the dead-letter-queue **orders-dl-queue.**



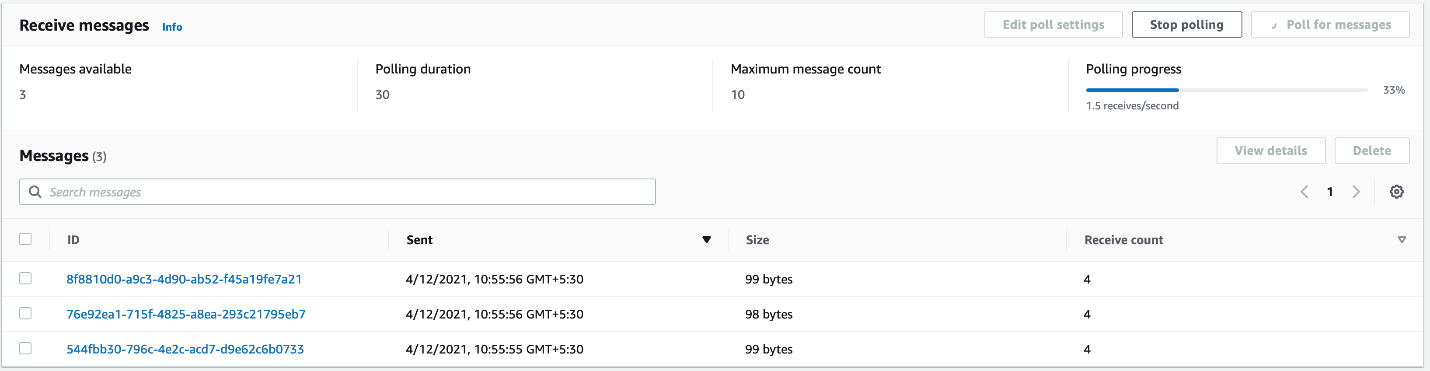
* 1. Click on the orders-dl-queue to inspect the messages in this dead-letter-queue. Then click on the **Send and receive messages** button.



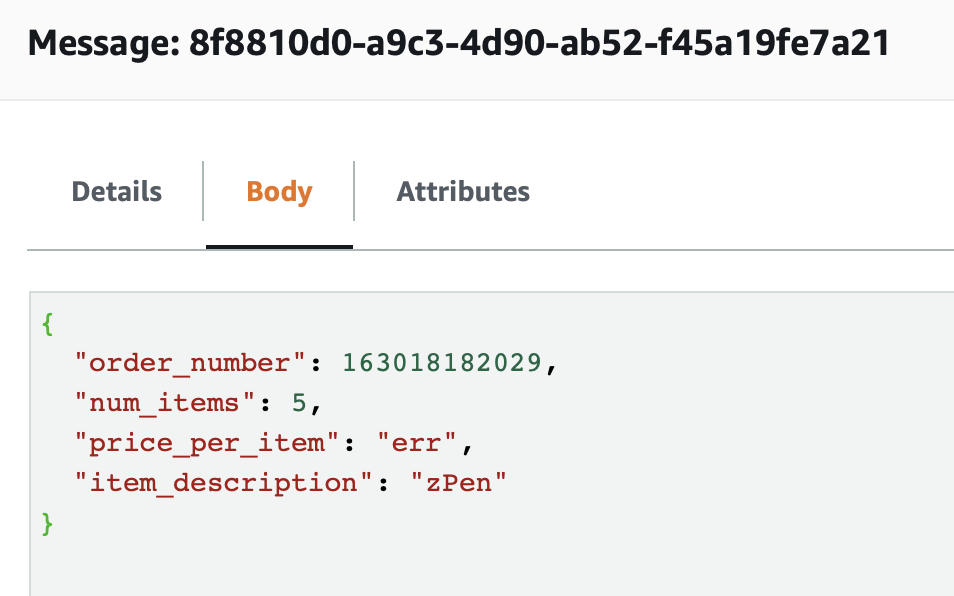
Click on **Poll for messages**to view the messages

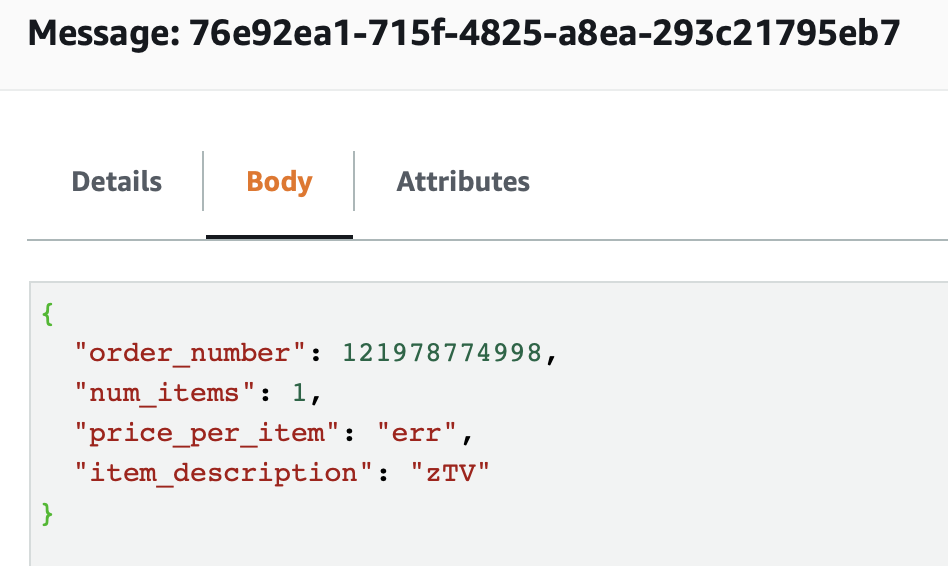


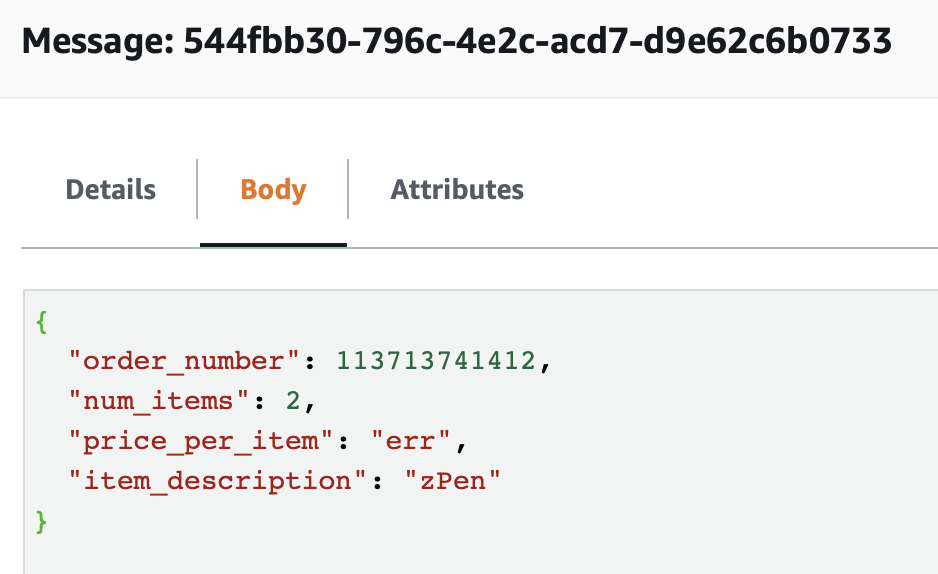
You should see the list of messages available in the dead-letter-queue:



Click on each of the messages to ascertain that each message has the value err for price\_per\_item key.







<https://aws.amazon.com/blogs/aws/amazon-sqs-new-dead-letter-queue/>

Clean up

Clean up the resources created in Task 1 through Task 5

1. Lets clean up the resources created for this lab:
   1. Delete the two lambda functions - **lab-order-simulator** and **lab-order-processor**
   2. Delete the two SQS queues - **orders-queue** and **orders-dl-queue**
   3. Delete the DynamoDB **orders** table
2. Congratulations for successfully completing this lab!