**Aim** :- Implement the messaging system using Amazon SNS and Amazon SQS.

**Lab overview and objectives**

In this lab, you will use Amazon Simple Queue Service (Amazon SQS) and Amazon Simple Notification Service (Amazon SNS) to set up a system to receive, queue, and send data for an application to process. You will use a Python publisher to send messages to a notification topic. You will also review the Node.js consumer that a web application will use to retrieve and process the data from a queue.

After completing this lab, you should be able to:

* Configure SNS topics and SQS queues to support programmatic receipt of messages
* Develop an Amazon SNS publisher to send messages to an SNS topic
* Develop an Amazon SQS consumer to read messages from an SQS queue

**AWS service restrictions**

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

**Scenario**

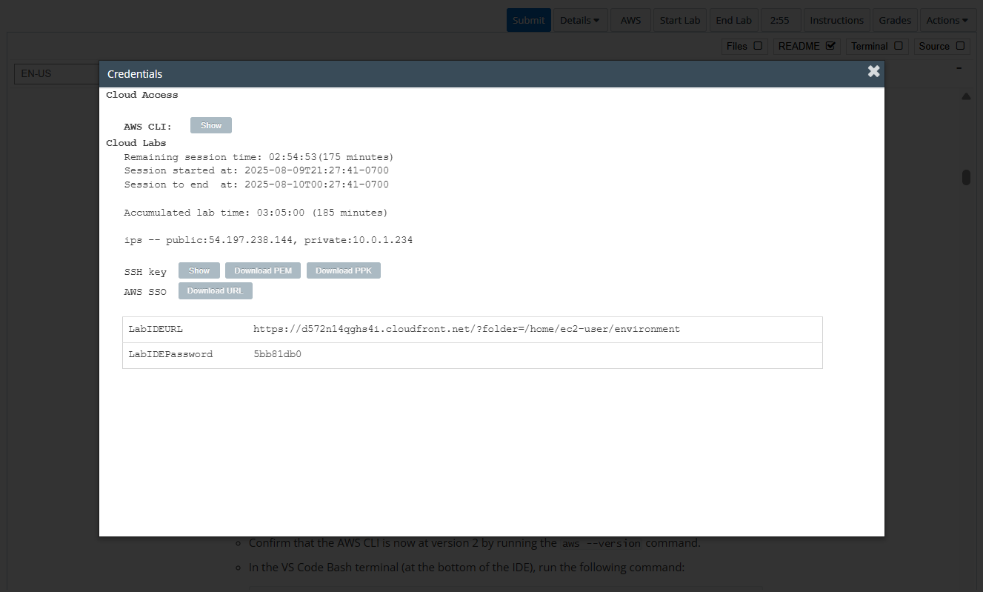
Customers love being able to buy coffee beans from the café, but maintaining the bean inventory is time-consuming. Café employees must call each coffee supplier and then use the coffee suppliers web application to manually update each record. Frank has asked Sofía if she can find a better way to keep the coffee inventory current. He would like café staff to spend more time helping customers and less time doing data entry.

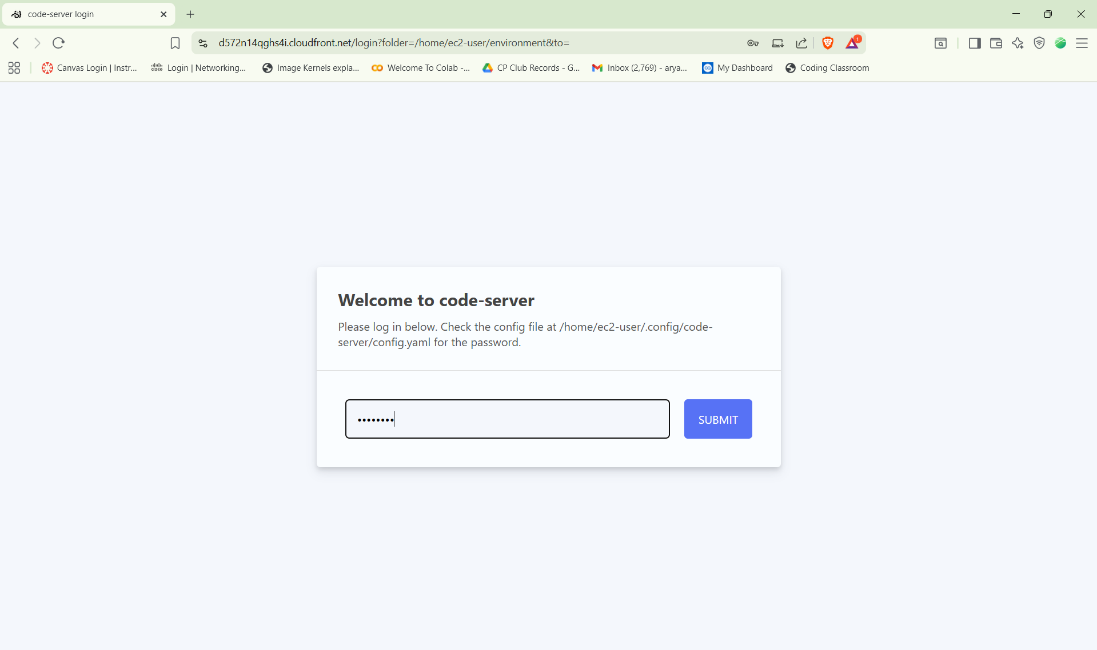
Sofía has researched setting up a messaging system to automatically receive and process inventory updates. Mateo, a café regular and AWS consultant, suggested using an SNS topic to receive messages from suppliers, and an SQS queue to store the messages until the application is ready to process them. This way, the café application won't lose any messages if the application or database happens to be unavailable. He also highly recommended that she create a *dead-letter queue* to handle messages that cannot be processed. Mateo also explained that suppliers will need to run a script called a *producer* to publish their updates to the SNS topic. The coffee suppliers application code will need to include a *consumer* to poll and retrieve messages from the SQS queue.

In this lab, you will again play the role of Sofía as you develop the automated inventory processing for the café's coffee suppliers application.

**Task 1: Preparing the lab**

Connect to the VS Code IDE.

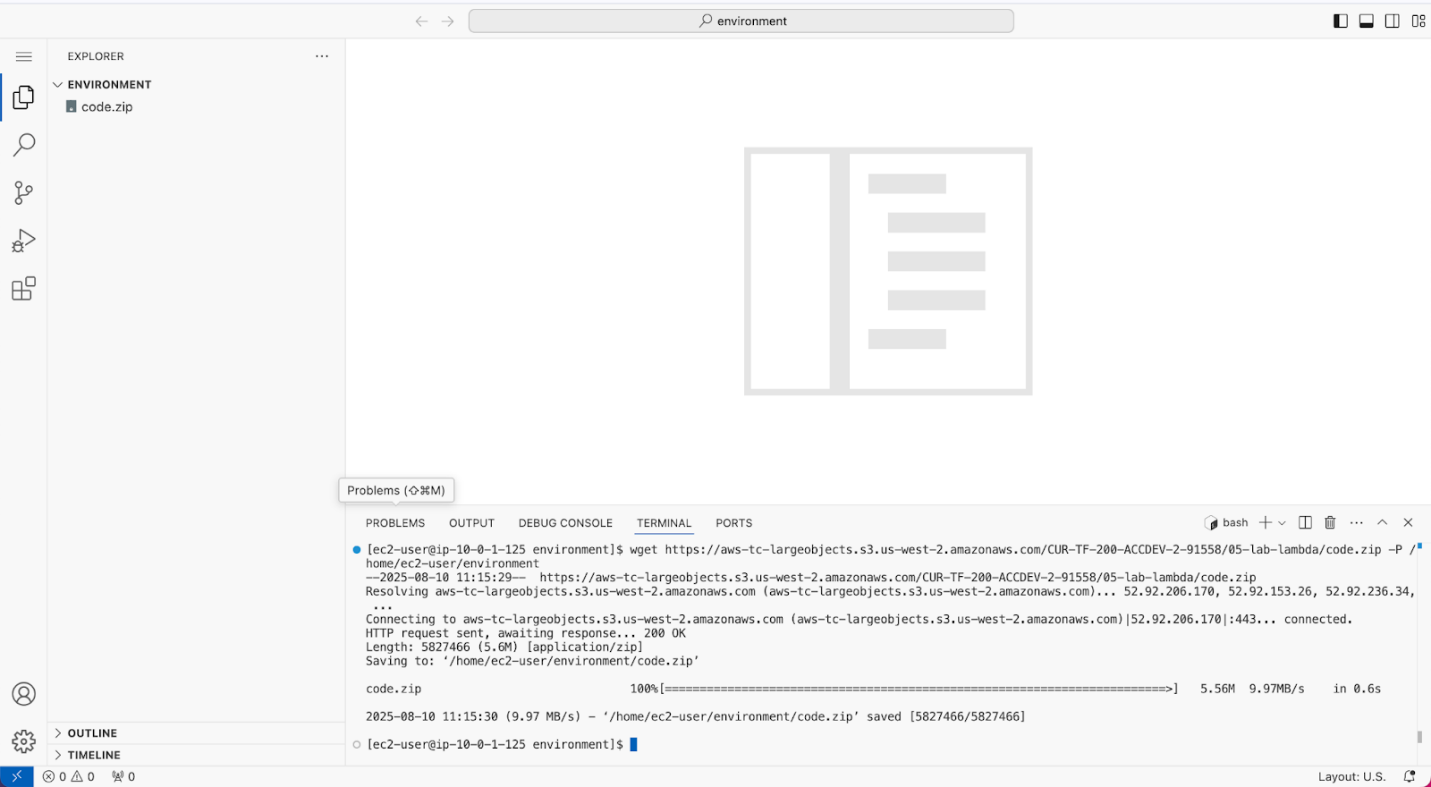
1. At the top of these instructions, choose Details followed by **AWS: Show**
2. Copy values from the table **similar** to the following and paste it into an editor of your choice for use later.
   1. **LabIDEURL**
   2. **LabIDEPassword**
3. In a new browser tab, paste the value for **LabIDEURL** to open the VS Code IDE.
4. On the prompt window **Welcome to code-server**, enter the value for **LabIDEPassword** you copied to the editor earlier, choose **Submit** to open the VS Code IDE.

****

1. Download and extract the files that you need for this lab.

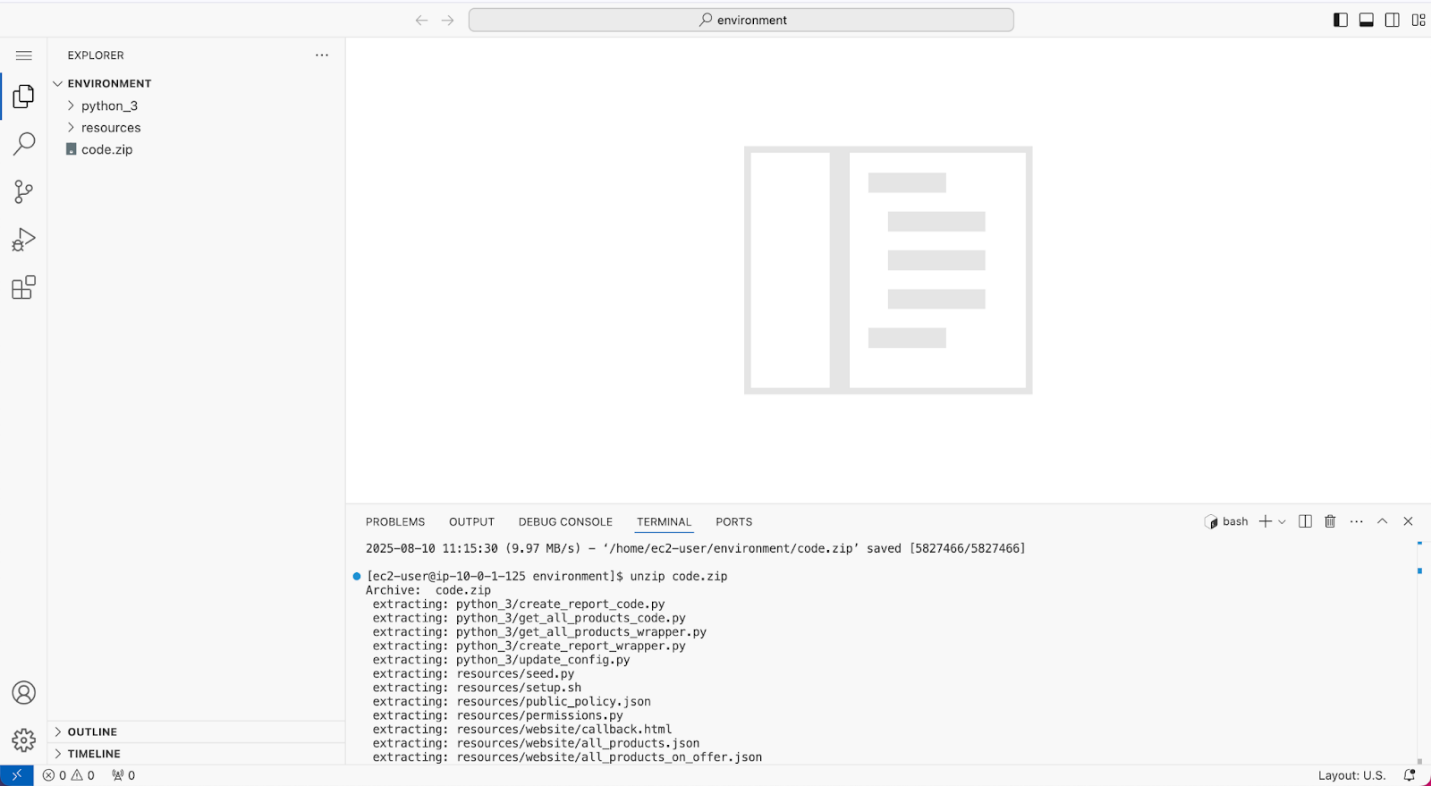
* In the VS Code bash terminal (located at the bottom of the IDE), run the following commands:

**wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/10-lab-sqs/code.zip -P /home/ec2-user/environment**



1. You should see that the **code.zip** file was downloaded to the VS Code IDE and is now in the left navigation pane.

* Extract the file by running the following command:

unzip code.zip

1. Run a script that upgrades the version of the AWS CLI installed on the VS Code IDE.

* To set permissions on the script and then run it, run the following commands in the Bash terminal:

chmod +x ./resources/setup.sh && ./resources/setup.sh

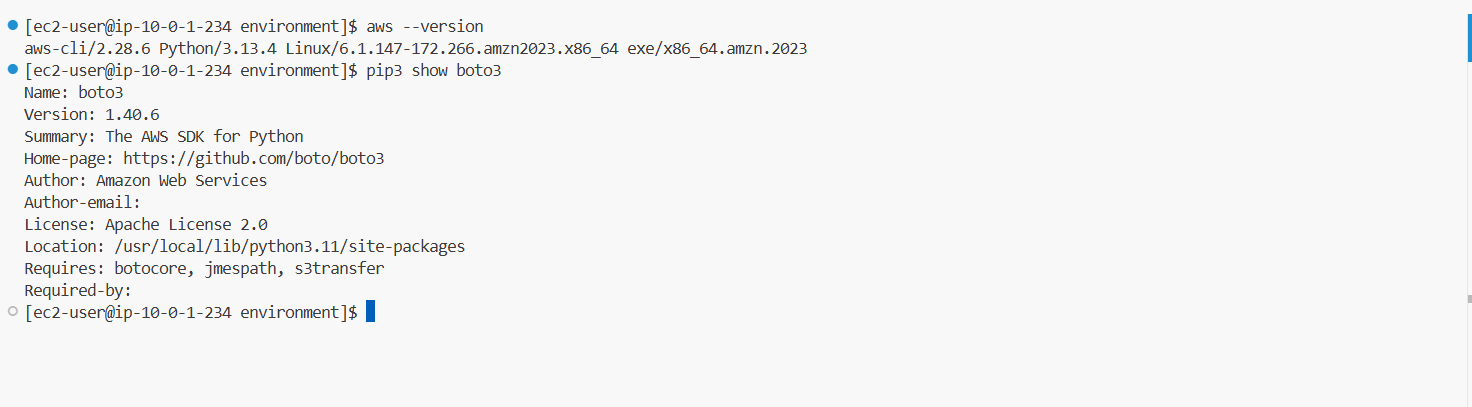
The script will prompt you for the **IP address** by which your computer is known to the internet.

Use [www.whatismyip.com](http://www.whatismyip.com/) to discover this address and then paste the IPv4 address into the command prompt and finish running the script.



1. Verify the AWS CLI version and also verify that the SDK for Python is installed.

* Confirm that the AWS CLI is now at version 2 by running the **aws --version** command.
* In the VS Code Bash terminal (at the bottom of the IDE), run the following command:

**pip3 show boto3**

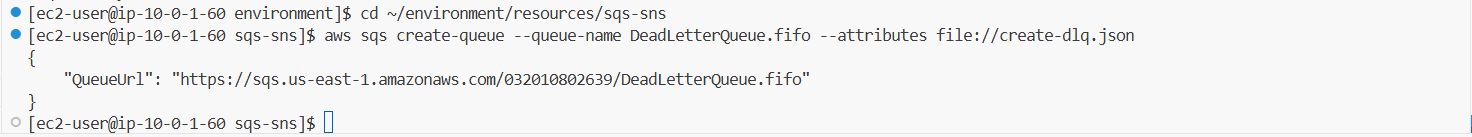
**Task 2: Configuring the Amazon SQS dead-letter queue**

1. Create an Amazon SQS dead-letter queue.

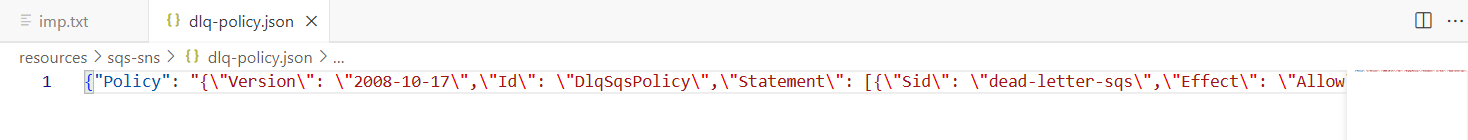
* In the VS Code IDE bash terminal, change to the *sns-sqs* directory:

**cd ~/environment/resources/sqs-sns**

* Create a dead-letter queue named *DeadLetterQueue.fifo*:

**aws sqs create-queue --queue-name DeadLetterQueue.fifo --attributes file://create-dlq.json**

1. Update the policy statement with your AWS account ID.

* In the environment window, under *resources/sqs-sns*, open the *dlq-policy.json* file.
* The *Policy* attribute's value is the same as the policy that you just reviewed. However, to work with the AWS Command Line Interface (AWS CLI), the entire policy must be formatted on a single line. In addition, the double quotes (") that are part of the policy definition must be escaped with a backslash (\) for the AWS CLI to correctly interpret the policy definition.
* ****In the *dlq-policy.json* file, replace the two *<FMI\_1>* placeholders with your AWS account ID.

1. In the terminal window, run the following command to update the dead-letter queue's access policy. Replace the <FMI\_1> placeholder with the DeadLetterQueue QueueURL from your text file:

**aws sqs set-queue-attributes --queue-url "<FMI\_1>" --attributes file://dlq-policy.json**

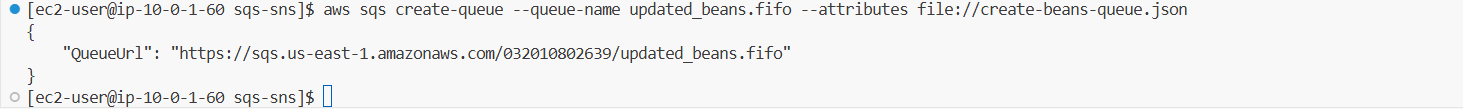


**Task 3: Configuring the SQS queue**

1. Review the attributes that will define your SQS queue.

* In the environment window, under *resources/sqs-sns*, open the *create-beans-queue.json* file.
* In the *create-beans-queue.json* file, replace the *<FMI\_1>* placeholder with your AWS account ID.

1. In the terminal window, run the following command to create a queue named *updated\_beans.fifo*:

**aws sqs create-queue --queue-name updated\_beans.fifo --attributes file://create-beans-queue.json**

1. Update the policy statement with your AWS account ID.

* In the environment window, under *resources/sqs-sns*, open the *beans-queue-policy.json* file.
* Replace all eight *<FMI\_1>* placeholders with your AWS account ID.

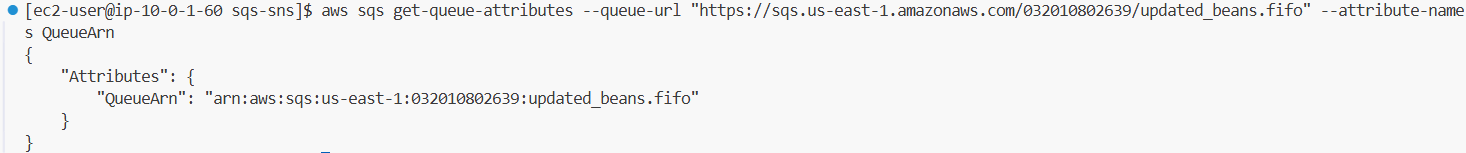
1. In the terminal window, run the following command to update the updated\_beans.fifo queue's policy. Replace the <FMI\_1> placeholder with the updated\_beans.fifo QueueURL that you saved earlier:

**aws sqs set-queue-attributes --queue-url "<FMI\_1>" --attributes file://beans-queue-policy.json**

1. Retrieve the ARN for the *updated\_beans.fifo* queue to use later.

* Run the following command. Replace the *<FMI\_1>* placeholder with your AWS account ID.

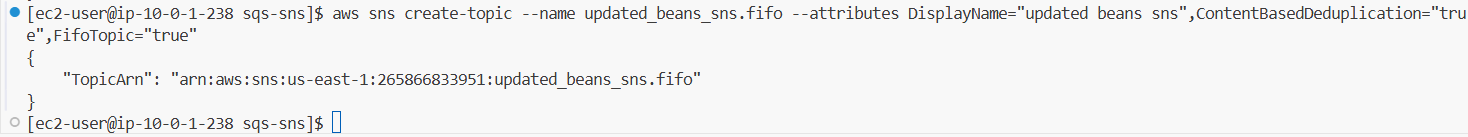
**aws sqs get-queue-attributes --queue-url "https://sqs.us-east-1.amazonaws.com/<FMI\_1>/updated\_beans.fifo" --attribute-names QueueArn**



**Task 4: Configuring the SNS topic**

1. To create the SNS topic, run the following command:

**aws sns create-topic --name updated\_beans\_sns.fifo --attributes DisplayName="updated beans sns",ContentBasedDeduplication="true",FifoTopic="true"**



1. Update the policy statement with your AWS account ID.

* In the environment window, under *resources/sqs-sns*, open the *beans-topic-policy.json* file.
* Update the placeholders:
  + Replace the *<FMI\_1>* placeholder with the *updated\_beans\_sns.fifo TopicArn* that you saved earlier.
  + Replace the five *<FMI\_2>* placeholders with your AWS account ID.

1. To apply the policy to your queue, run the following command.

**aws sns set-topic-attributes --cli-input-json file://beans-topic-policy.json**

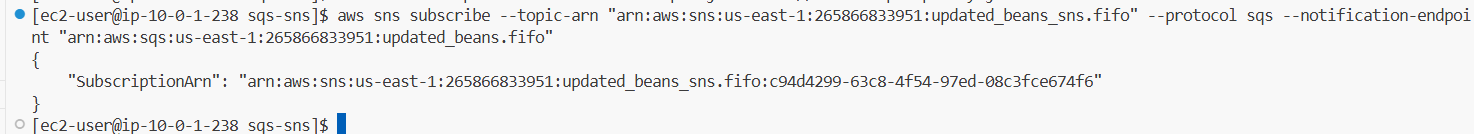


**Task 5: Linking Amazon SQS and Amazon SNS**

1. Create a subscription from the SQS queue to the SNS topic.

* Update the following command:
  + Replace the *<FMI\_1>* placeholder with the *update\_beans\_sns.fifo TopicArn*.
  + Replace the *<FMI\_2>* placeholder with the *updated\_beans.fifo QueueArn*.

**aws sns subscribe --topic-arn "<FMI\_1>" --protocol sqs --notification-endpoint "<FMI\_2>"**

****

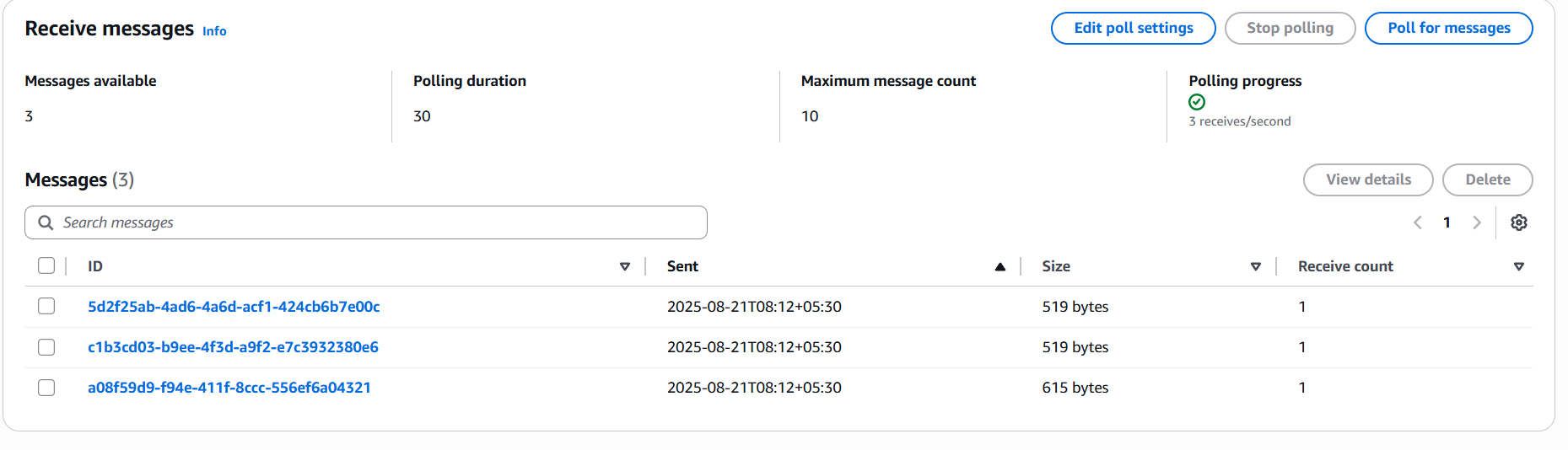
**Task 6: Testing message publishing**

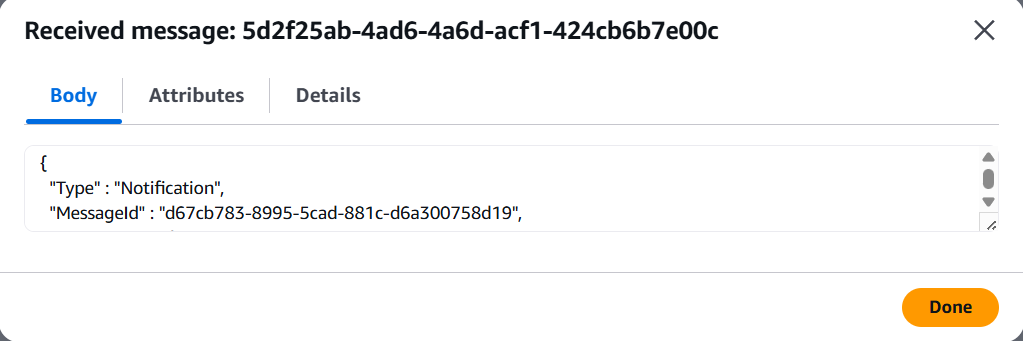
1. In the *send\_beans\_update.py* file, replace the *<FMI\_1>* placeholder with your AWS account ID to update the *sns\_topic* value.
2. In the terminal, use the following commands to run the Python publisher:

**cd ~/environment/resources/sqs-sns**

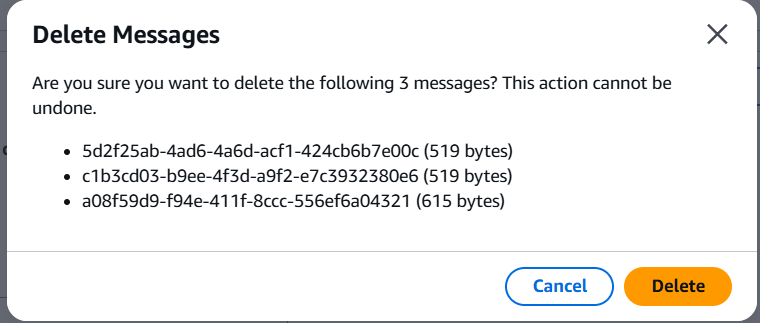
**python3 send\_beans\_update.py beans\_update\_1.txt**

1. Navigate to the Amazon SQS console.

* In the AWS Management Console, search for and select Simple Queue Service
* The *DeadLetterQueue.fifo* and *updated\_beans.fifo* queues are listed.
* Choose the link for the *updated\_beans.fifo* queue and choose **Send and receive messages**
* Poll for new messages:In the bottom pane, choose **Poll for messages**.
* Locate the message with the largest **Size**.
* To examine the message details, choose the **ID** hyperlink.
* Review the message body.

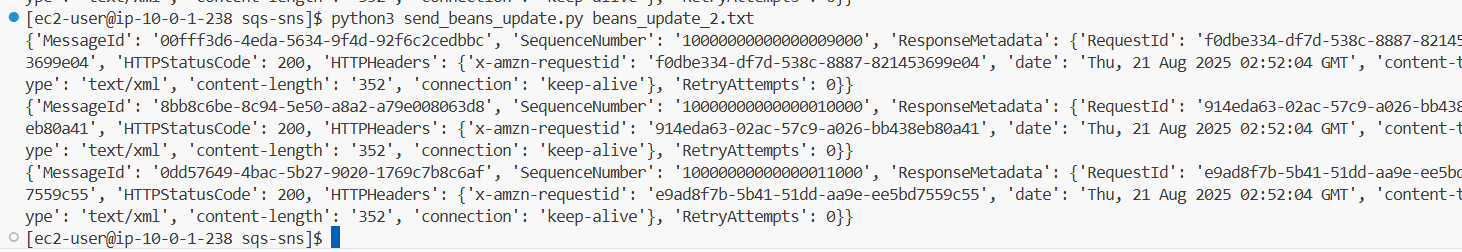


1. Delete messages from the queue.

* Select all messages in the queue.
* Choose **Delete**, and choose **Delete** again when you are prompted to confirm this action.

1. In the VS Code IDE bash terminal window, run the Python publisher again. This time, send a new set of records:

**python3 send\_beans\_update.py beans\_update\_2.txt**



**Task 7: Configuring the application to poll the queue**

1. View the coffee bean inventory.

* Return to the browser tab that is open to the Amazon SQS console.
* Choose **Services**, and choose **Elastic Beanstalk**.
* Choose the hyperlink for **MyEnv**.
* To open the currently deployed coffee suppliers application, choose the URL directly under **MyEnv**.
* The coffee suppliers application opens in a new browser tab.
* Append /beans to the URL in your browser tab.
* The URL is similar to the following:

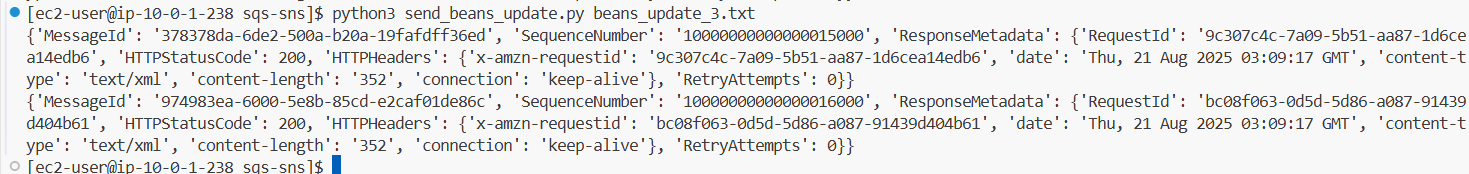
**https://myenv.eba-xxxxxxxx.us-east-1.elasticbeanstalk.com/beans**

1. Review the messages in the dead-letter queue.

* Return to the browser tab that is open to the Elastic Beanstalk console.
* Choose **Services**, and then choose **Simple Queue Service**.
* Choose the hyperlink for the **DeadLetterQueue.fifo** queue.
* Choose **Send and receive messages**.
* In the bottom pane, choose **Poll for messages**.
* Three messages are listed in the dead-letter queue.

1. Use the following command to run the Python publisher again. This time you will send a new set of records:

**python3 send\_beans\_update.py beans\_update\_3.txt**



**Conclusion:-**

In this lab, I successfully implemented a messaging system using **Amazon SNS** and **Amazon SQS** to enable automated coffee inventory updates. The following steps, configurations, and commands were performed:

* Connected to **VS Code IDE** using provided credentials.
* Downloaded and extracted the lab resources:
* **wget https://aws-tc-largeobjects.s3.us-west-2.amazonaws.com/CUR-TF-200-ACCDEV-2-91558/10-lab-sqs/code.zip -P /home/ec2-user/environment**
* **unzip code.zip**
* Updated AWS CLI and verified installation:
* **chmod +x ./resources/setup.sh && ./resources/setup.sh**
* **aws --version**
* **pip3 show boto3**
* **Created Dead Letter Queue (DLQ):**
* **cd ~/environment/resources/sqs-sns**
* **aws sqs create-queue --queue-name DeadLetterQueue.fifo --attributes file://create-dlq.json**
* **aws sqs set-queue-attributes --queue-url "<DLQ\_URL>" --attributes file://dlq-policy.json**
* **Created SQS queue (updated\_beans.fifo)** and applied policy:
* **aws sqs create-queue --queue-name updated\_beans.fifo --attributes file://create-beans-queue.json**
* **aws sqs set-queue-attributes --queue-url "<QUEUE\_URL>" --attributes file://beans-queue-policy.json**
* **aws sqs get-queue-attributes --queue-url "https://sqs.us-east-1.amazonaws.com/<AccountID>/updated\_beans.fifo" --attribute-names QueueArn**
* **Created SNS topic** and applied topic policy:
* **aws sns create-topic --name updated\_beans\_sns.fifo --attributes DisplayName="updated beans sns",ContentBasedDeduplication="true",FifoTopic="true"**
* **aws sns set-topic-attributes --cli-input-json file://beans-topic-policy.json**
* **Linked SQS with SNS (subscription):**
* **aws sns subscribe --topic-arn "<TopicArn>" --protocol sqs --notification-endpoint "<QueueArn>"**
* **Tested message publishing with Python publisher:**
  + **cd ~/environment/resources/sqs-sns**
  + **python3 send\_beans\_update.py beans\_update\_1.txt**
  + **python3 send\_beans\_update.py beans\_update\_2.txt**
  + **python3 send\_beans\_update.py beans\_update\_3.txt**
* Verified messages in **Amazon SQS Console** and reviewed processing with **Elastic Beanstalk application**.
* Checked **Dead Letter Queue** for failed messages.

**Result :-**

