Exercise: SQS - Java

In this exercise you will:

1. Create an SQS queue
2. Write a Java program that sends messages to your queue
3. Write a Java Lambda function that processes messages sent to your queue

Assumptions

You have already configured your AWS credentials on your computer. This should have been done when you installed the AWS CLI on your computer.

Intellij is installed on your computer.

Steps

1. Go the the AWS Simple Queue Service (SQS) web console

2. Create an SQS queue
    a. Click the “Create New Queue” button
    b. Enter a name for your queue
    c. Select “Standard Queue”
    d. Click the “Quick-Create Queue” button

3. Send a message to your queue through the SQS web console
    a. On the main SQS console page, select the new queue by checking the box next to its name
    b. In the “Queue Actions” menu, select “Send a Message”
    c. Type some text in the message body
    d. Click the “Send Message” button
    e. Click “Close”

4. Verify that the message was sent
    a. On the main SQS console page, select your queue
    b. In the “Queue Actions” menu, select “View/Delete Messages”
    c. Click “Start Polling for Messages”
    d. You should see the message you sent in the queue (click More Details to see more details)
    e. Click “Close”

5. Next, you will write a Java program that sends messages to your queue.
6. Create a directory for this exercise

7. Create a new Intellij project

8. In the new project, add dependencies on the AWS SDK Java library
    a. Select the File -> Project Structure menu
    b. Select “Modules” on the left side of the “Project Structure” dialog
    c. Select the “Dependencies” tab
    d. Click the “plus” icon in the top-right
    e. Select “Library”
    f. Select “From Maven”
    g. Enter “com.amazonaws:aws-java-sdk-core:1.11.547”, and click OK. This will add a dependency on the core AWS Java SDK library.
    h. Using the same process, add a Maven dependency on “com.amazonaws:aws-java-sdk-sqs:1.11.547”. This will add a dependency on the SQS portion of the AWS Java SDK library.

9. Create a class named SqsClient containing the following code. The queue URL can be found in the SQS console (in the Details tab).

```
import com.amazonaws.services.sqs.AmazonSQS;
import com.amazonaws.services.sqs.AmazonSQSClientBuilder;
import com.amazonaws.services.sqs.model.SendMessageRequest;
import com.amazonaws.services.sqs.model.SendMessageResult;

public class SqsClient {
   public static void main(String[] args) {

      String messageBody = "*** PUT YOUR MESSAGE BODY HERE ***";
      String queueUrl = "*** PUT YOUR QUEUE URL HERE ***";

      SendMessageRequest send_msg_request = new SendMessageRequest()
            .withQueueUrl(queueUrl)
            .withMessageBody(messageBody);
            .withDelaySeconds(5);

      AmazonSQS sqs = AmazonSQSClientBuilder.defaultClient();
      SendMessageResult send_msg_result = sqs.sendMessage(send_msg_request);

      String msgId = send_msg_result.getMessageId();
      System.out.println("Message ID: " + msgId);
   }
}
```

10. More information on accessing SQS from Java can be found here:

    a. https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs-messages.html

11. Run your program a few times, and then go to the SQS console to verify that the messages sent by your program are showing up in the queue.

12. Next you will write a Java Lambda function that processes messages sent to your queue.

13. Your Lambda function needs to run with an IAM role that has SQS permissions. Modify the IAM role you created in the "Lambda / IAM" exercise to allow SQS access:
    a. Go to the IAM web console
    b. On the left, select "Roles"
    c. Goto the role you created previously for the "Lambda / IAM" exercise by clicking on its name (e.g., cs340lambda)
    d. Click the "Attach Policies" button
    e. On the "Attach Permissions" screen, attach the AmazonSQSFullAccess policy to your role. This will allow your Lambda function to access your SQS queues.

14. To the IntelliJ project created earlier, add dependencies on the AWS SDK Java library
    a. Select the File -> Project Structure menu
    b. Select "Modules" on the left side of the "Project Structure" dialog
    c. Select the "Dependencies" tab
    d. Click the "plus" icon in the top-right
    e. Select "Library"
    f. Select "From Maven"
    g. Enter "com.amazonaws:aws-lambda-java-core:1.2.0", and click OK. This will add a dependency on the Lambda portion of the AWS Java SDK library.
    h. Using the same process, add a Maven dependency on "com.amazonaws:aws-lambda-java-events:2.2.5". This library contains the SQSEvent class, which will be used by your Lambda function.

15. In your project, create a Java package named "sqs".

16. In the "sqs" package, create a class named QueueProcessor that will contain the "handler" for your Lambda function. Add the following code to this file.

```
package sqs;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.SQSEvent;

public class QueueProcessor implements RequestHandler<SQSEvent, Void> {
```

```
    @Override
    public Void handleRequest(SQSEvent event, Context context) {

        for (SQSEvent.SQSMessage msg : event.getRecords()) {

            // TODO:
            // Add code to print message body to the log

        }

        return null;
    }

}
```

17. Fill in the code that logs the message body

18. Once your program compiles, add an “artifact” to your Intellij project that creates a JAR file containing your compiled code.
    a. Select the File -> Project Structure menu
    b. Select “Artifacts” on the left side of the “Project Structure” dialog
    c. To add a new artifact, click the “plus” button at the top of the dialog
    d. Select JAR -> From modules with dependencies
    e. In the “Create JAR from Modules” dialog, keep the default options, and click OK
    f. Check the “Include in project build” check box
    g. Rebuild the project, which will result in a JAR file being created in the “out/artifacts/<DIR>/ folder. This JAR file can be used to deploy your Lambda function to AWS

19. Next, deploy your Lambda function.
    a. Login to the AWS console
    b. Navigate to the Lambda service
    c. Select “Functions” on the left side of the Lambda screen
    d. Click the “Create Function” button in the top-right corner
    e. Give your function a name (e.g., queue_processor)
    f. As your function’s Runtime select “Java 8”
    g. Under Permissions, click “Choose or create an execution role”
    h. In the “Execution role” drop-down, select “Use an existing role”
    i. In the “Existing role” drop-down, select the name of the IAM role that you configured earlier (e.g., cs340lambda)
    j. Click the “Create function” button. This will actually create the function, and take you to a screen that lets you further configure the function.
    k. Scroll down to the “Function code” section of the function configuration screen.

l. In the “Handler” field, specify the name of the method that contains your Lambda function handler (e.g., sqs.QueueProcessor::handleRequest)
m. Click the Function package “Upload” button and select the JAR file created in the previous step
n. Scroll to the top, and click the “Save” button. This will deploy your Lambda function, which means it can now be called

20. Next, connect your Lambda function to your SQS queue, as follows:

21. Go back to the SQS web console

22. Select your queue by checking the box next to its name

23. In the “Queue Actions” menu, select “Configure Trigger for Lambda Function”

24. Select the name of the Lambda function you previously created (e.g., queue_processor)

25. Click the “Save” button

26. Now, whenever a message is sent to your SQS queue, the message will be sent to your Lambda function for processing. Verify that this is happening by running your Java program that sends messages to your queue, or send messages manually through the SQS console.

27. IMPORTANT: After completing the exercise, remove the lambda trigger from your queue, because SQS constantly polls the queue to detect new messages so it can call the lambda appropriately. All of this polling adds to your AWS charges, so you will want to remove the lambda trigger to avoid this.

28. Submit your Java code through Canvas.

More In formation

The full AWS SDK documentation can be found here in the SDKs section:
https://aws.amazon.com/tools/