The Amazon API Gateway Workshop

Introduction
Getting Started
Module 1 - Introduction to Amazon API Gateway
Module 2 - Deploy your first API with IaC
Module 3 - API Gateway REST Integrations
Module Goals
Mock
HTTP Integration
AWS Lambda
AWS Step Functions
Amazon SQS
Set up your AWS SAM Project
Build Amazon SQS Integration with AWS SAM and OpenAPI
Test Integration
Amazon SNS
Amazon Kinesis
Amazon DynamoDB
Amazon EventBridge
Private Integration
Amazon S3
Clean up
Module 4 - Observability in API Gateway
Module 5 - WebSocket APIs
Module 6 - Enable fine-grained access control for your APIs
Clean up
Resources

Build Amazon SQS Integration with AWS SAM and OpenAPI

You can integrate an API method to put a message into SQS Queue using **AWS type integration**.

Use AWS SAM and OpenAPI to create an API Gateway REST API with SQS integration

1. Using AWS Cloud9 console, return to the root folder `module-3/sqs`
2. This code belongs in your SAM template file `template.yaml`

   Review the code and then **copy/paste** it into the `template.yaml` file.

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: >
   module3-lambda-rest-api: Sample SAM Template for module3-sqs-rest-api

Globals:

    # Enable Logs
    Api:
        MethodSettings:
            - ResourcePath: "/*"
              HttpMethod: "*"
              DataTraceEnabled: True
              LoggingLevel: INFO
              MetricsEnabled: True
    Function:
        Timeout: 3
        Runtime: nodejs18.x

Resources:

    # SQS Queue for Orders
    OrderQueue:
        Type: AWS::SQS::Queue
        Properties:
            QueueName: OrderQueue

    # Example REST API Gateway Integrated with SQS
    APIWithSQSIntegration:
        Type: AWS::Serverless::Api
        Properties:
            StageName: dev
            OpenApiVersion: 3.0.3
            DefinitionBody: # an OpenApi definition
                "Fn::Transform":
                    Name: "AWS::Include"
                    Parameters:
                        Location: "openapi.yaml"
            EndpointConfiguration:
                Type: REGIONAL

     # IAM Role For APIGW Integration with SQS
    IAMRoleForSQSIntegration:
        Type: AWS::IAM::Role
        Properties:
            AssumeRolePolicyDocument:
                Version: "2012-10-17"
                Statement:
                  - Effect: Allow
                    Principal:
                      Service:
                        - apigateway.amazonaws.com
                    Action:
                      - 'sts:AssumeRole'
            Policies:
                - PolicyName: PolicyForAPIGWSQSIntegration
                  PolicyDocument:
                    Version: "2012-10-17"
                    Statement:
                      - Effect: Allow
                        Action: 'sqs:SendMessage'
                        Resource:
                            - !GetAtt OrderQueue.Arn

    #Lambda Function to consumes SQS Queue
    SQSConsumerFunction:
        Type: 'AWS::Serverless::Function'
        Properties:
            CodeUri: ./handlers
            Handler: queue-consumer.handler
            Policies:
            - SQSPollerPolicy:
                QueueName: !GetAtt OrderQueue.QueueName
            - DynamoDBCrudPolicy:
                TableName: !Ref OrderTable
            Environment:
                Variables:
                    SAMPLE_TABLE: !Ref OrderTable
                    REGION: !Sub "${AWS::Region}"
            Events:
                SQSTrigger:
                  Type: SQS
                  Properties:
                    Queue: !GetAtt OrderQueue.Arn

    OrderTable:
        Type: AWS::Serverless::SimpleTable
        Properties:
            PrimaryKey:
                Name: id
                Type: String
            ProvisionedThroughput:
                ReadCapacityUnits: 2
                WriteCapacityUnits: 2

Outputs:

    Endpoint:
        Description: API Gateway Endpoint
        Value:
            Fn::Sub: https://${APIWithSQSIntegration}.execute-api.${AWS::Region}.amazonaws.com/dev/order
```

- The resource `OrderQueue` defines a **SQS Queue**.
- The resource `IAMRoleForSQSIntegration` defines an **IAM Role** responsible for allowing API Gateway to place a message in SQS with the least possible privilege.
- The `SQSConsumerFunction` resource defines the **Lambda** that will **consume** the request messages that Amazon API Gateway will put on SQS.

> Note that the `SQSPollerPolicy` and `DynamoDBCrudPolicy` polices, respectively, add permissions to the Lambda execution role to pull messages from the queue and save them in DynamoDB, respecting the principle of the **least-privilege**. To learn more visit AWS SAM policy templates.

3. This code belongs in your OpenAPI definition file `openapi.yaml`

   Review the code and then **copy/paste** it into the `openapi.yaml` file

```yaml
1   openapi: "3.0.1"
2   info:
3      title: "module-3-sqs-integration"
4      description: "API Gateway example for using SQS as direct service integration"
5      version: "1.0"
6
7   paths:
8
9      /order:
10        post:
11           produces:
12           - "application/json"
13           responses:
14              "200":
15                 description: "200 response"
16                 schema:
17                    $ref: "#/definitions/Empty"
18           x-amazon-apigateway-integration:
19              credentials:
20                 Fn::Sub: ${IAMRoleForSQSIntegration.Arn}
21              httpMethod: "POST"
22              uri:
23                 Fn::Sub: "arn:aws:apigateway:${AWS::Region}:sqs:path/${AWS::AccountId}/${OrderQueue.QueueName}"
24              responses:
25                 default:
26                    statusCode: "200"
27              requestTemplates:
28                 application/json: "Action=SendMessage&MessageBody=$input.body"
29              requestParameters:
30                 integration.request.header.Content-Type: "'application/x-www-form-urlencoded'"
31              passthroughBehavior: "when_no_match"
32              type: "aws"
33
34  definitions:
35     Empty:
36        type: "object"
37        title: "Empty Schema"
```

The x-amazon-apigateway-integration object **(line 18)** specifies details of the backend integration used for this method.

On **line 20** we have `${IAMRoleForSQSIntegration.Arn}`, responsible for **associate the IAM Role**, that will allow API Gateway put message on SQS, with the **API method**.

On **line 23** we have the **uri definition**. For **Amazon SQS**, we override the path with the following format: **example-account-id/example-sqs-queue-name**. This definition `"arn:aws:apigateway:${AWS::Region}:sqs:path/${AWS::AccountId}/${OrderQueue.QueueName}"` will result in this configuration and format.

On **line 32** we have `type: "aws"`, responsible for configuring the integration as **AWS** type to expose AWS service actions.

With AWS integration it is possible to have more control in the request and response workflow. In this example we are transforming the input payload before passing it to SQS using Apache Velocity Template Language (VTL).

In **Line 28** we have the request transformation template definition `Action=SendMessage&MessageBody=$input.body` with defines the action as **SendMessage** and put the request body to SQS as **message**.

In **line 30** we have the **Content-type** header definition as `application/x-www-form-urlencoded`. Failure to provide this header will result in the following error: **UnknownOperationException**.

Deploy the project

1. To deploy the Amazon API Gateway and the AWS Lambda to your AWS account, run the following commands from the application root `module-3/sqs`, where the `template.yaml` file for the sample application is located:

```
sam build && sam deploy --guided
```

   The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you need to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure as it is above.

   - **Stack Name**: `module-3-sqs-integration`
   - **AWS Region**: Put the chosen region to run the workshop. **e.g.** `us-east-1`
   - **Confirm changes before deploy**: `Y`
   - **Allow SAM CLI IAM role creation**: `Y`
   - **Disable rollback**: `N`
   - **Save arguments to configuration file**: `Y`
   - **SAM configuration file** and **SAM configuration environment** leave blank

```
Configuring SAM deploy
======================

        Looking for config file [samconfig.toml] :  Found
        Reading default arguments  :  Success

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [module-3-sqs-integration]: module-3-sqs-integration
        AWS Region [us-east-1]:
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [Y/n]: y
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]: y
        #Preserves the state of previously provisioned resources when an operation fails
        Disable rollback [y/N]: n
        Save arguments to configuration file [Y/n]: y
        SAM configuration file [samconfig.toml]:
        SAM configuration environment [default]:
```

2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.

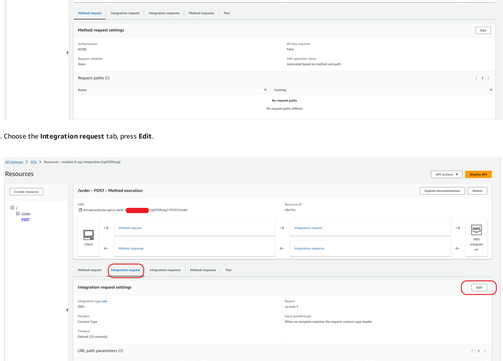
```
CloudFormation stack changeset
-------------------------------------------------------------------------------------------------------------------------
Operation          LogicalResourceId                             ResourceType                          Replacement
-------------------------------------------------------------------------------------------------------------------------
+ Add              APIWithSQSIntegrationDeploymente2493af871     AWS::ApiGateway::Deployment           N/A
+ Add              APIWithSQSIntegrationdevStage                 AWS::ApiGateway::Stage                N/A
+ Add              APIWithSQSIntegration                         AWS::ApiGateway::RestApi              N/A
+ Add              IAMRoleForSQSIntegration                      AWS::IAM::Role                        N/A
+ Add              OrderQueue                                    AWS::SQS::Queue                       N/A
+ Add              OrderTable                                    AWS::DynamoDB::Table                  N/A
+ Add              SQSConsumerFunctionRole                       AWS::IAM::Role                        N/A
+ Add              SQSConsumerFunctionSQSTrigger                 AWS::Lambda::EventSourceMapping       N/A
+ Add              SQSConsumerFunction                           AWS::Lambda::Function                 N/A
-------------------------------------------------------------------------------------------------------------------------
```

Test Integration

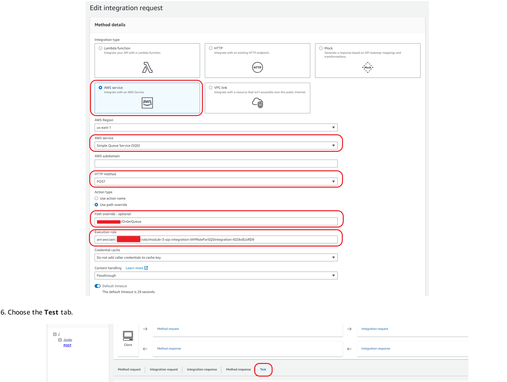
After you create your API Gateway REST API with SQS integration, you can test the API.

Test SQS Integration using API Gateway console

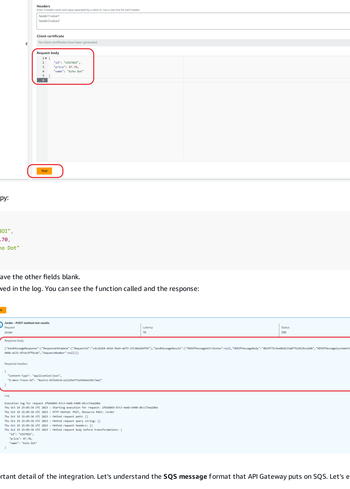
1. Open the Amazon API Gateway console and sign in.
2. Choose your REST API named, `module-3-sqs-integration`.
3. In the **Resources** pane, you can select the method you want to test. Click on the `POST` method.



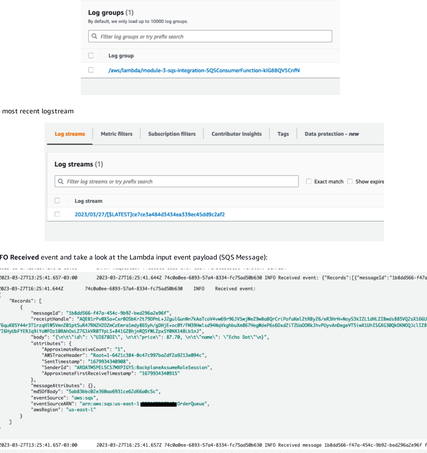
4. Choose the **Integration request** tab, press **Edit**.



5. Review the configuration details for integration. Press **Cancel** once done reviewing.



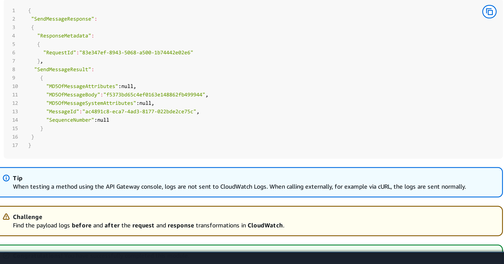
6. Choose the **Test** tab.



- For **Request Body** copy:

```
{
  "id": "1U567BO1",
  "price": 87.70,
  "name": "Echo Dot"
}
```

7. Click on **Test** button. Leave the other fields blank.
8. The test result are showed in the log. You can see the function called and the response:



9. Now let's check an important detail of the integration. Let's understand the **SQS message** format that API Gateway puts on SQS. Let's explore **SQS Lambda consumer** logs.
10. Navigate to CloudWatch in the AWS console.
11. Click on **Log Groups**, and then click on a lambda log group with the prefix `module-3-sqs-integrati...`

12. Click on the most recent logstream

13. Look for **INFO Received** event and take a look at the Lambda input event payload (SQS Message):

Test the deployed API using cURL

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the following cURL command and paste it into the terminal window, replacing `<api-id>` with your APIs API ID and `<region>` with the region where your API is deployed. You may have copied this URL in from the CloudFormation output in the last step. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**.

```
curl -X POST https://<api-id>.execute-api.<region>.amazonaws.com/dev/order \
-H 'Content-Type: application/json' \
-d '{"id": "0f854123","price": 200.00,"name": "Karaoke"}'
```

3. The **Response Body** output should be

```
{
  "SendMessageResponse":
  {
    "ResponseMetadata":
    {
      "RequestId": "83e347ef-8943-5068-a500-1374442e02e6"
    },
    "SendMessageResult":
    {
      "MD5OfMessageAttributes":null,
      "MD5OfMessageBody":"f5373bd65c4ef8163e148862f0499944",
      "MD5OfMessageSystemAttributes":null,
      "MessageId":"ac4891c8-ea7-4ad3-8177-822bde2ce75c",
      "SequenceNumber":null
    }
  }
}
```

Tip
When testing a method using the API Gateway console, logs are not sent to CloudWatch Logs. When calling externally, for example via cURL, the logs are sent normally.

Challenge
Find the payload logs **before** and **after** the **request** and **response** transformations in **CloudWatch**.