Homework Assignment #3

Estimated time: 240 – 360 minutes

Objectives

- Gain some experience in using Containers
- Gain some experience in using SQS

Overview

In this assignment, you will modify your Consumer program from HW2, so it can retrieve Widget Requests from an SQS queue. Also, you will package up the Consumer so you could run multiple concurrent instances of it in containers.

Instructions

Step 0 – Preliminaries

Be sure that you use the Learner Lab associated with CS5260 in the AWS Academy Canvas.

Also, make sure that your most up-to-date credentials are in your AWS credentials file on the machine that you are using for development.

Step 1 – Setup a queue

Using the SQS Console, create a SQS queue with the following specifications:

Type:	standard
Name:	cs5260-requests
Configuration settings:	use defaults
Access policy:	use defaults
Encryption:	disabled
Dead-letter queue:	disabled
No additional tags	

Take a snapshot of your queue's overview screen. Be sure to include the header bar that shows your account id.

Step 2 – Refine Your Design

This new version of the Consumer will need to include a new feature that will retrieve Widget Requests from the SQS queue created in Step 1. The user should be able to specify the name of queue on the command line. This new feature should not break any existing features. In

other words, the Consumer should still be able to retrieve a Widget Requests from an S3 Bucket, if so specified via command-line parameters.

Also, this new version will need to process Delete and Update Widget Requests, besides the Create requests. After retrieving a message (containing a request) and processing that request, your Consumer must delete the message the queue using the messages receipt handle.
To optimize the use of the queue, you can read up to 10 messages at a time. Your message retrieving software component could still return just one message but cache the others and return them in subsequent calls to that method or function.

Step 3 – Implement and Test the Changes

In this step you will implement your revised design. Also, as before, your implementation must contain executable unit test cases that test all non-trivial methods or functions reasonably thoroughly.

Also, your program should write meaningful messages to a log file. Meaningful logging and reasonable thoroughly testing will help you reduce your overall time spent on HW3 and HW4. For the logging, look for a 3rd library that handle all the heavy work. For Java, consider Log4J2. For Python, consider the "logging" module. There are good logging libraries for virtual every modern programming language. There is no need for you to write your own logging component(s) – simply use an existing library that have been a good reputation and following.

Before you move unto the next step, do some system testing by running the instructor-provided Producer and multiple instances of your Consumer program on your local machines. Run at least two instances of your Consumer program in separate windows and then the Producer in a third. You can use the "--max-widget-requests" (or "-mwr") command-line argument set the limit of requests, if you like, but try it with at least 100. You should see each instance of the Consumer processes different requests. You may see a few failed Update or Delete requests. Do not worry about these errors at this point but think about why they might be occurring.

Take snapshots of the screen that show at least two Consumers running concurrently and submit these with your deliverables for the assignment.

If you need to clear out (purge) all existing requests from the queue, you can do so through the SQS console or by using the AWS CLI. However, AWS limits how often you can do this.

Step 4 – Package into a Docker Image and Deploy to Local Containers

In this step, you will package and deploy your program so it can run in a Docker container. You can do this on your own workstation or on an EC2 instance that you are using as a workstation. So, if you don't have already have Docker installed, you will need to download it and install it.

To package your Consumer program into a Docker Image, you will need to create an appropriate Dockerfile that 1) specifies your runtime environment, 2) copies your program into the container, and 3) specifies a command that will run your program. The following screen snapshot shows a sample docker file and a build command that creates a docker image from that file.

```
swc@SWC-MacBook-Pro hw3 % more Dockerfile
FROM openjdk:8
COPY dist/consumer.jar consumer.jar
CMD ["java","-jar","consumer.jar", "-r", "us-east-1", "-rq", "https://sqs.us-east-1.amazonaws.com/351775707843/c
s5260-requests", "-dwt", "widgets"]
swc@SWC-MacBook-Pro hw3 % docker build -t consumer:2.3 .
Sending build context to Docker daemon  94.96MB
Step 1/3 : FROM openjdk:8
 ---> 9324460525ca
Step 2/3 : COPY dist/consumer.jar consumer.jar
 ---> Using cache
 ---> 5c6e3daf0337
Step 3/3 : CMD ["java","-jar","consumer.jar", "-r", "us-east-1", "-rq", "https://sqs.us-east-1.amazonaws.com/351
775707843/cs5260-requests", "-dwt", "widgets"]
 ---> Using cache
 ---> 8926ab9b32bd
Successfully built 8926ab9b32bd
Successfully tagged consumer:2.3
swc@SWC-MacBook-Pro hw3 %
```

The first command in the above screenshot displays the Dockerfile. In this case, the Dockerfile use an image that supports OpenJDK 8 as the base layer. The second line in file copies a jar file from the local file system into the image. The third line defines a command that will be run when the container starts up. This Dockerfile is just an example. Yours can and probably will be a little different.

The second command in the above screenshot, i.e., "docker build -t consumer:2.3 .", creates a Docker image using the current directory as the context and tags it as consumer:2.3.

Once you are able to build a Docker image, test it running the Producer program with parameters that cause it generate 100+ request and then by launching the Docker image in one container. Look at the output to make sure everything is working. You may see a few Update and Delete requests failing. Think about why this is so, even when you are running just one Consumer. Step 7 will have you answer questions related to this behavior.

Below is a screenshot that illustrates the launching a container in the foreground (i.e., so the output to the console is immediately visible). Just before this container was launcher, the Producer was executed to place 100+ requests into the queue.

```
swc@SWC-MacBook-Pro hw3 % docker run --env-file docker.env consumer:2.3
21:59:44.686 [main] INFO  edu.usu.cs5260.common.Settings - help: false
21:59:44.691 [main] INFO  edu.usu.cs5260.common.Settings - profile: default
21:59:44.691 [main] INFO  edu.usu.cs5260.common.Settings - region: us-east-1
21:59:44.691 [main] INFO  edu.usu.cs5260.common.Settings - request-bucket: null
21:59:44.692 [main] INFO  edu.usu.cs5260.common.Settings - request-queue: https://sqs.us-east-1.amazonaws.com/351775
21:59:44.692 [main] INFO  edu.usu.cs5260.common.Settings - max-runtime: 30000
21:59:44.692 [main] INFO  edu.usu.cs5260.consumer.ConsumerSettings - widget-bucket: null
21:59:44.693 [main] INFO  edu.usu.cs5260.consumer.ConsumerSettings - widget-key-prefix: widgets/
21:59:44.693 [main] INFO  edu.usu.cs5260.consumer.ConsumerSettings - dynamodb-widget-table: widgets
21:59:44.693 [main] INFO  edu.usu.cs5260.consumer.ConsumerSettings - queue-wait-timeout: 10
21:59:44.693 [main] INFO  edu.usu.cs5260.consumer.ConsumerSettings - queue-visibility-timeout: 10
21:59:47.009 [Thread-1] INFO  edu.usu.cs5260.consumer.ConsumerDynamoWidgetAdapter - Put or update in DynamoDB table
21:59:47.902 [Thread-1] INFO  edu.usu.cs5260.consumer.ConsumerDynamoWidgetAdapter - Put or update in DynamoDB table
21:59:48.218 [Thread-1] INFO  edu.usu.cs5260.consumer.ConsumerDynamoWidgetAdapter - Put or update in DynamoDB table
21:59:48.540 [Thread-1] INFO  edu.usu.cs5260.consumer.ConsumerDynamoWidgetAdapter - Put or update in DynamoDB table
```

Take a screen snapshot of launching and running of the container, like above. Be sure that the snapshot shows the command line and at least 15 lines of output. Submit this snapshot with your other deliverables for the assignment.

Important Note:
To run the Consumer in a container, your will need to authorize the container or pass your credentials to the container so the Consumer can access your AWS resources. AWS provides several ways to do this. Some (like using ECS Local Endpoints) are sophisticated and follow best practices but use concepts that we have not covered yet. Some can only be used if the containers are running within the AWS cloud. Others (like passing environment variables into the containers) work for containers running anywhere and use concepts that you should know but are not considered best practices.

For this assignment, the recommendation is that you pass AWS_ACCESS_KEY_ID, AWS_SECRET_KEY, and AWS_SESSION_TOKEN environment variables to your container, by putting them credentials in a file (that is **not** committed to Git) and running the container with a "–env-file" parameter, as illustrated below:

```
swc@SWC-MacBook-Pro hw3 % more docker.env
AWS_ACCESS_KEY_ID=ASIAVDZ3XKLBVUWP7F5C
AWS_SECRET_ACCESS_KEY=vsbNVxV6DSHFHP5TlzySvZcDLuon/0XNNnuHYtNO
AWS_SESSION_TOKEN=FwoGZXIvYXdzEIb//////////wEaDEUzAGuuN1l9GIHSjiK8AYmcjRp58OKJirBSz9GU5bWvtHwvHZfFu8YxGZi7nnQVJ2L
HhCuvukiU42qYm1/100WZz9KRrURy8UgxQqDgcdcRSgzSbIgAzJGRnRUcpy4HyjRoPDbeb9tA0e9HLxEuy7qnSfQqrFHrIfNTqCaRxPRSlORFTWe7
Vjt8AgtHD/U42U0=
swc@SWC-MacBook-Pro hw3 % docker run --env-file docker.env -d consumer:2.3
54bf2dc0dbd56d48ad0bbaa3325f6ee70ab86842ca817366d581ccea52ff2395
swc@SWC-MacBook-Pro hw3 % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS     NAMES
54bf2dc0dbd5   consumer:2.3   "java -jar consumer.…"   8 seconds ago   Up 7 seconds             elegant_taussig
swc@SWC-MacBook-Pro hw3 %
```

In this screenshot, the first command shows the contents of file that contains a list of environment variables. In this case, those variables are for authentication. The default authentication mechanism in your SDK should automatically look for this environment variables. If not, you may have to edit your code to provide a customer AWS credential provider chain. See the documentation for your SDK on how to authenticate using environment variables.

The second command launches a container using an existing docker image. The "–env-file" parameter specifies the file that contains the environment variables. The "-d" parameter tells Docker to run the container in the background.

The third command shows the running containers.

Step 5 – Push Your image to a Docker Image repository

In this step, you will first create a private Docker Image repository using the AWS ECR console and the following specifications:

Name:	cs5260
Scan on push:	enabled

Then, you will push your Docker image to that repository. For instructions on how to push images in general, read:

https://docs.aws.amazon.com/AmazonECR/latest/userguide/docker-push-ecr-image.html

Since the repository is private, you will need to configure Docker with the necessary credentials for pushing images to that repository. The recommended way to do this is to install and config a docker credential helper. For more details, see the following GitHub project and review its README file.

https://github.com/awslabs/amazon-ecr-credential-helper

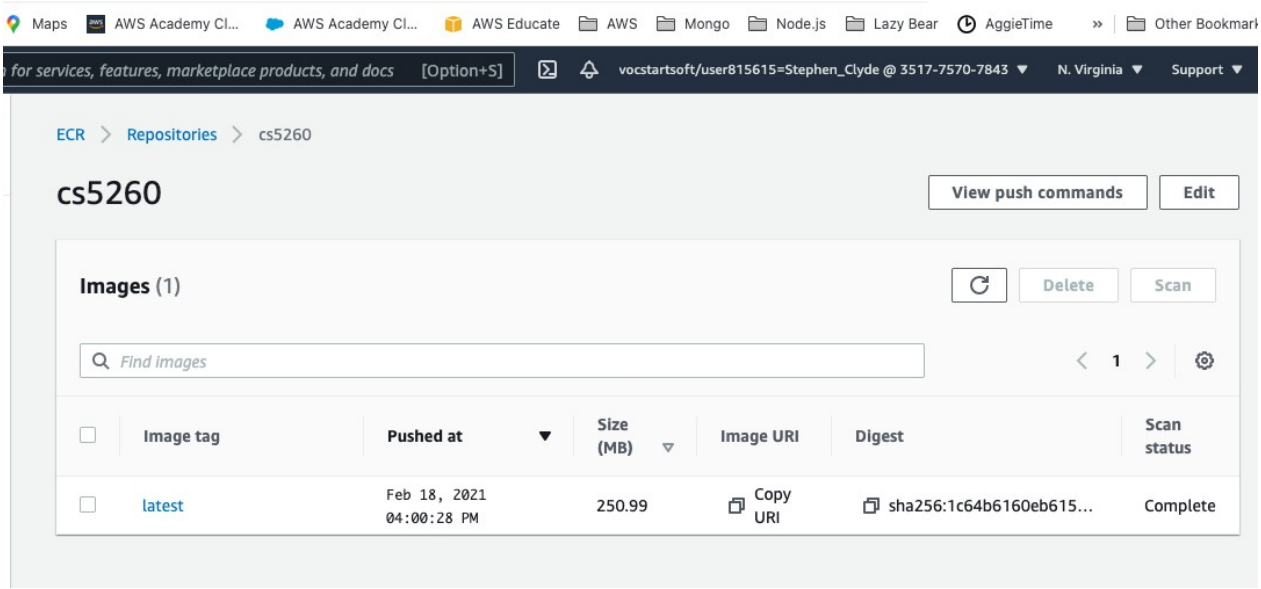
Setting up this ECR credential helper should be sufficient, but if you need more information about using private repositories with AWS ECR, read the following article.

https://docs.aws.amazon.com/AmazonECR/latest/userguide/registry_auth.html

The following screenshot illustrates the push of an image, after the docker credential helper has been installed and configured. The first command lists the existing docker images. The second command add a new tag the "consumer:2.3" image. This tag makes reference a repository to which we want to push the image. In this tag, the "3570177570783" is the AWS account Id and AWS cs5260 is the name of the ECR repository. Your account Id will be different and can be found in the top header bar of your AWS Management Console. Note, that it is display there with dashes every four digits. Do not include the dashes when you make use of the account Id.

```
swc@SWC-MacBook-Pro ~ % docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
consumer     2.3       8926ab9b32bd   2 hours ago   536MB
openjdk      8         9324460525ca   9 days ago    514MB
swc@SWC-MacBook-Pro ~ % docker tag 8926ab9b32bd 351775707843.dkr.ecr.us-east-1.amazonaws.com/cs5260
swc@SWC-MacBook-Pro ~ % docker push 351775707843.dkr.ecr.us-east-1.amazonaws.com/cs5260
Using default tag: latest
The push refers to repository [351775707843.dkr.ecr.us-east-1.amazonaws.com/cs5260]
2f0068b1a841: Pushed
02412b9dda81: Pushed
d7b2c55f7e50: Pushed
02f0a7f763a3: Pushed
da654bc8bc80: Pushed
4ef81dc52d99: Pushed
909e93c71745: Pushed
7f03bfe4d6dc: Pushed
latest: digest: sha256:1c64b6160eb615304165fd31984c4b1ac5b4bbecc2b1d9967d6fe5811a8fe7fa size: 2006
swc@SWC-MacBook-Pro ~ %
```

After you successfully push your Docker image, go to the ECR console and access the cs5260 repository. Take a screenshot of contents of the repository. Be sure the snapshot includes the account id in the header bar. Below is an example of what it might look like.



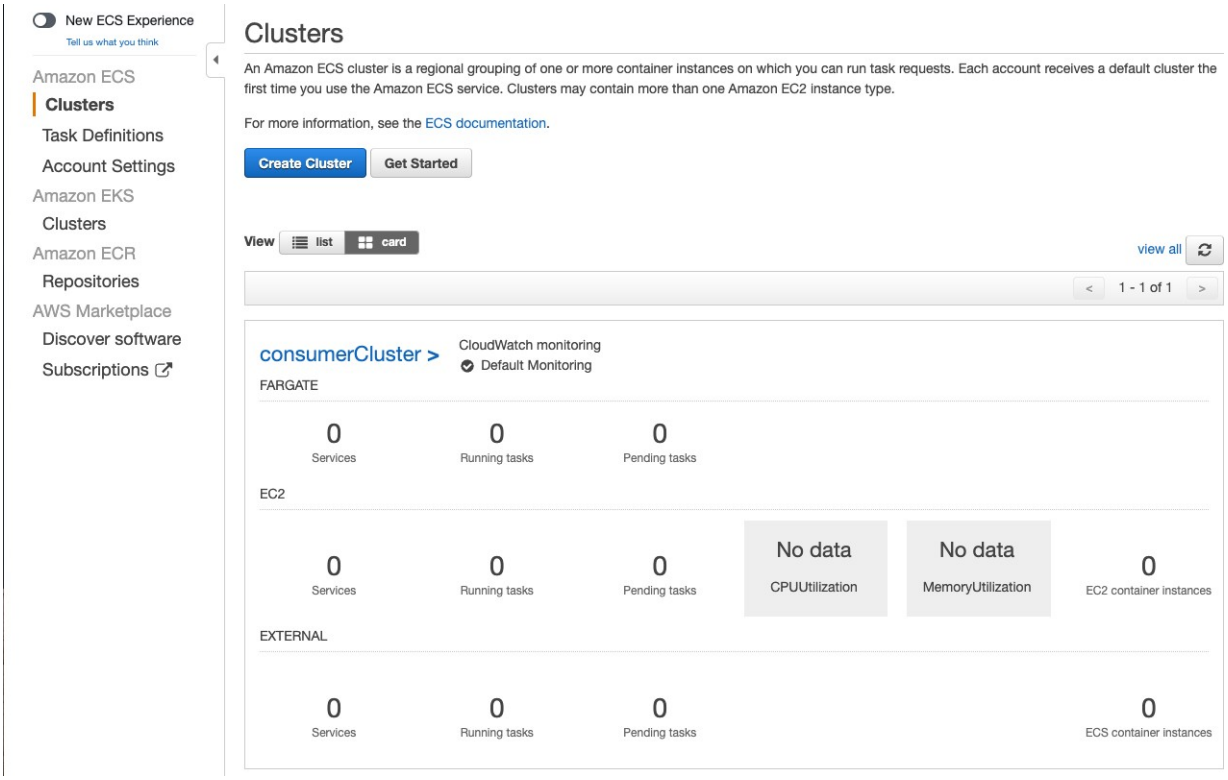
Step 6 – Deploy Containers on a ECS Cluster (Optional – Extra Credit)

In this step, you will setup a Container Cluster and deploy your Docker Image to two containers using AWS ECS and AWS Fargate.

First, you will need to create a Contain Cluster that defines the environment in which your containers will execute. To create the cluster, do the following:

1. Go to the ECS Console in the AWS Manage Console
2. Go to Clusters

3. Click a new cluster
    a. Click on “Create Cluster”
    b. For the cluster template, select “Network Only”, which is the one to AWS Fargate.
    c. Give the cluster a name, e.g., “consumerCluster”
    d. Do not select the option that will cause a new VPC to be created. Later, you will launch containers into your existing cs5260-vpc.
4. View your cluster. It should look something like the image below



Take a snapshot of your cluster view, like the one above, and submit it with your homework assignment.

Next, you will need to setup a task definition that specifies how to launch container with the Consumer Image. Here are the steps:

1. Go the CloudWatch Console
2. Create a log group called /ecs/consumerTask
3. Return to the ECS Console
4. Go the Repositories and your cs5260 repository
5. View you latest Docker image and copy the Image URI. It will be something like:

    346941219394.dkr.ecr.us-east-1.amazonaws.com/cs5260:lastest

6. Go to Task Definitions
7. Create a new Task Definition
    a. Click on “Create new Task Definition”
    b. Select “Fargate” for the launch type compatibility
    c. Give the task definition a name of “consumerTask”
    d. For the Task Role, select “LabRole”.
    e. For the Task Execution Role, select “LabRole”.
    f. For the Task Member and CPU, list 0.5GB and 0.25 vCPU.
    g. Add a container
        i. Click on “Add container”
        ii. Give the container a name, e.g., “consumer”
        iii. Paste in your Docker image URI for the image.
        iv. Set a soft memory limit of 128
    h. Make sure “auto-configure CloudWatch logs” is enabled, with the following log options: awslogs-group = /ecs/consumerTask, awslogs-region = us-east-1, and awslogs-stream-prefix = ecs.
    i. No other options or configuration parameters than those mention above should be necessary.
8. View the Task Definition.

Take a snapshot of your view that shows at least the task definition name, network mode, compatibilities, task memory, and task CPU size. Submit this snapshot as one of the work artifacts for the assignment.

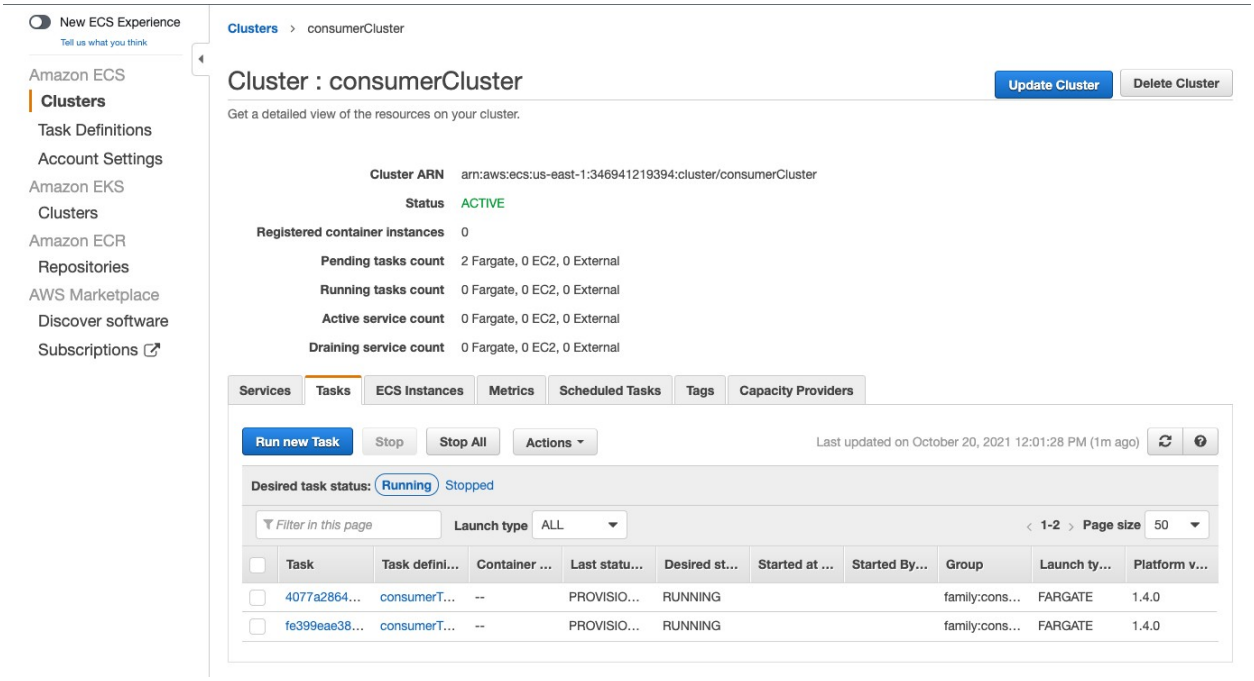
While the task is running check to see if the consumers are processing data. You can do this, by

a. Watching the number of requests in the SQS queue change.
b. Watching for objects being created in your DynamoDB table and or S3 Bucket 3.
c. Examining the CloudWatch logs associated with the task.

Take a snapshot of at least on these things to demonstrate the requests are being processed.

Finally, run the task by doing the following:

1. Clear out your DynamoDB table and your S3 Bucket 3
2. Run the Producer generate some requests
3. Go back to the Task Definitions screen in the ECS Console.
4. While looking at the task definition, click on “Actions” and “Run Task”
5. Select “Fargate” as the launch type.
6. Enter 2 for the number of tasks
7. Select your cs5260-vpc as the VPC.
8. Select the two available subnets.
9. Run the task and look at the Task for the cluster. It should look like the following:



Take a snapshot of your screen, like the one above, and submit it as one the work artifacts for this assignment.

On the Clusters Dashboard, watch for the tasks to move from “Pending” to “Running”. Once they do, your Consumers should be processing request. While they are running check to see if the consumers are processing data. You can do this, by

- Watching the number of requests in the SQS queue change.
- Watching for objects being created in your DynamoDB table and or S3 Bucket 3.
- Examining the CloudWatch logs associated with the task.

Take a snapshot of at least on these things to demonstrate the requests are being processed.

After your Consumers has processed all the requests, stop the tasks.

Note: If the tasks go from Pending to Running and then immediately disappear, there might be a problem with the Task Definition or with permission. You can click on the task in the Task tab of the cluster and then click on a Logs tab to investigate the problem. If you make mistake in the Task Definition, you will have to de-register the current version and recreate it.

Step 7 – Commit your work to a git repository

Manage your all project’s artifacts with Git, including project files, build instructions, etc. The only things you don’t need to keep in the Git repository are artifacts that are generated during the build process or at runtime. Commit to your Git repository frequently. Your Git commit log

will be examined during the grading and must show meaningful workflow. A single commit at the end is not acceptable.

Step 8 – Questions

1. Why is possible that Update and Delete Widget Requests may fail, even when you were running just one Consumer?
2. How would this possible behavior impact the design of distributed applications that use queues?

Hints and Other Thoughts

Consider standard queue behaviors. There is a small delay been the time when a message is submitted to an SQS queue and when it can be received. Even if there are messages logically in the queue, a receive-message operation may not return anything. This is an error, but an excepted behavior as a consequence of the way AWS replicates and distributes queued messages across multiple servers and availability zones.

Use a long polling and batch retrievals. Reduce the number of SQS operations by using long polling and by requesting up to 10 messages in a single receive-message operation.

Submission

Your submission must include:

- All screenshots mentioned in the above steps
- Answers to the questions in Step 8
- An archive file of your entire project, expect build-time or runtime-artifacts