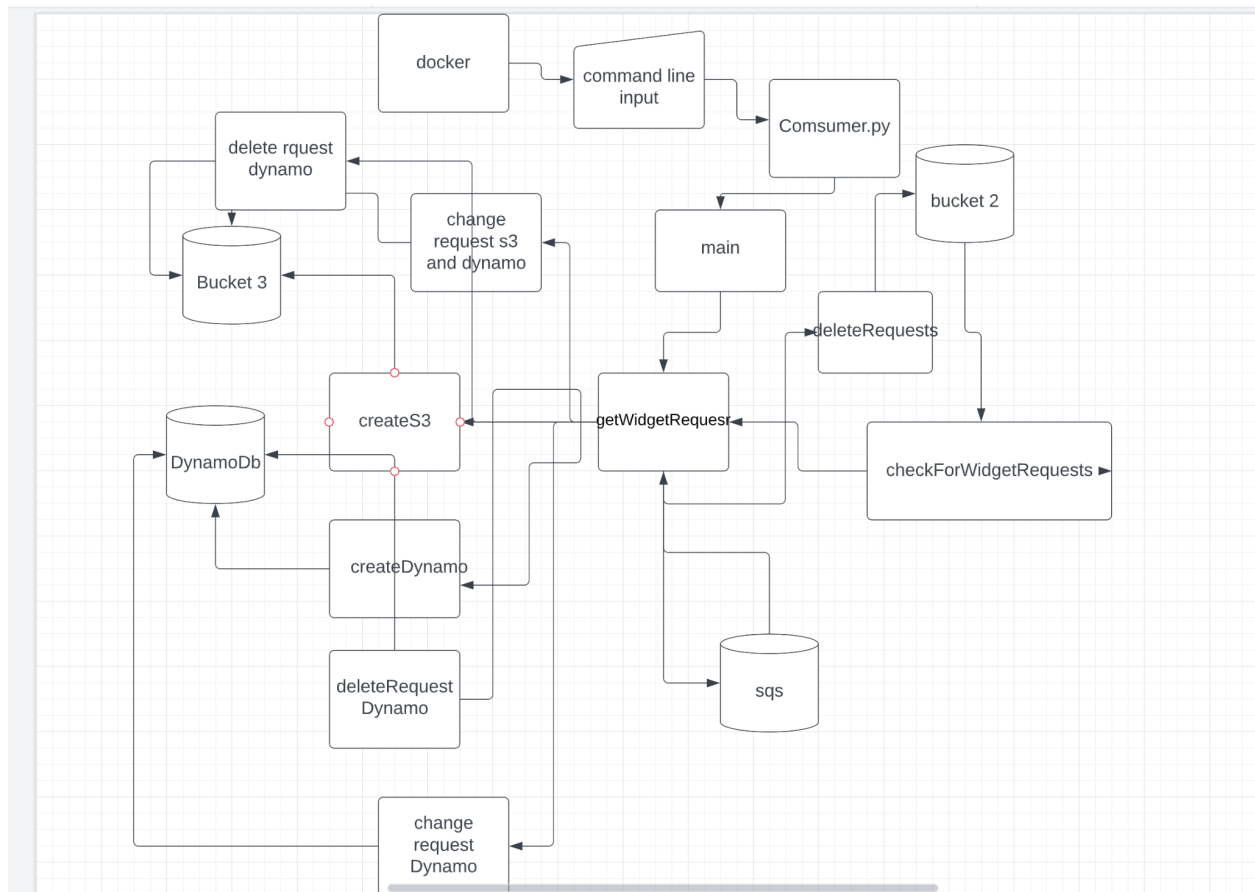


HW3

Changed design



sqs

The screenshot shows the Amazon SQS console for the queue 'cs5260-requests'. The 'Details' tab is active, displaying the following information:

Name	Type	ARN
cs5260-requests	Standard	arn:aws:sqs:us-east-1:514149339831:cs5260-requests

Additional details shown include Encryption (Disabled), URL (https://sqs.us-east-1.amazonaws.com/514149339831/cs5260-requests), and Dead-letter queue (-).

Below the details, the 'SNS subscriptions' tab is selected. It shows a subscription region of 'us-east-1' and a list of 'SNS subscriptions (0)'. A button 'Subscribe to Amazon SNS topic' is visible.

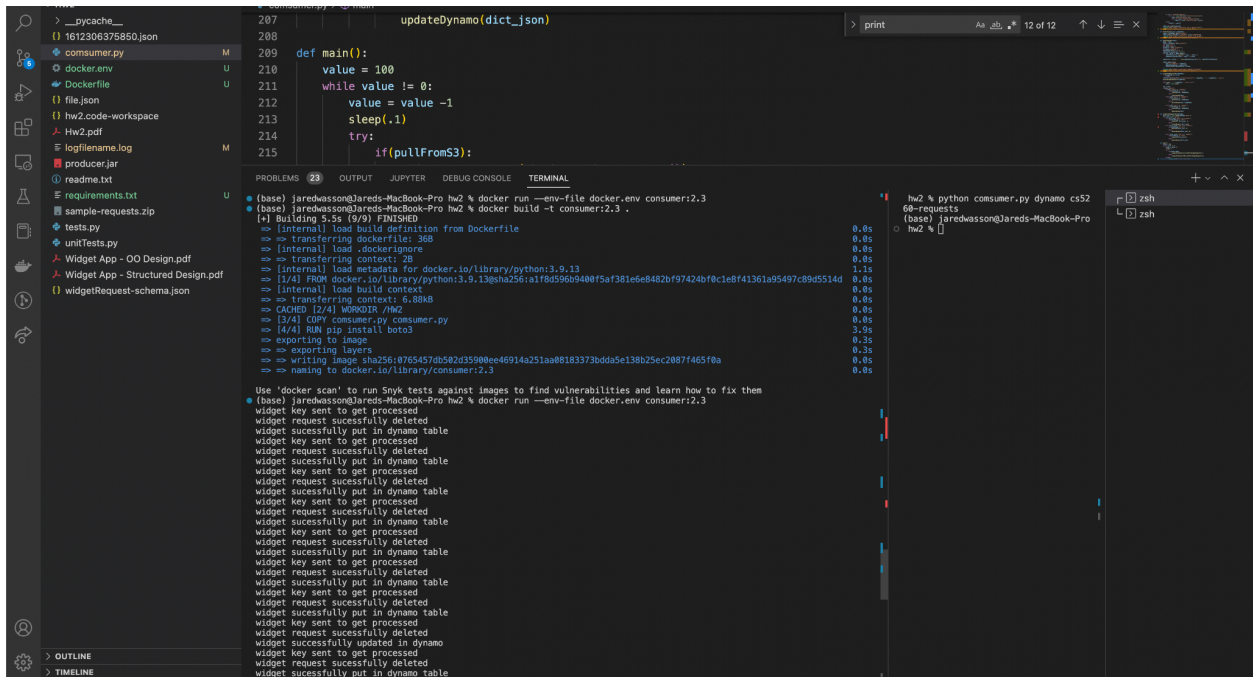
Running two consumers

The screenshot shows a code editor with a Python script named 'consumer.py'. The script is designed to consume messages from an SQS queue and store them in an S3 bucket. It includes a test function 'test_checkForWidgetRequests()' and a function 'test_createS3()'.

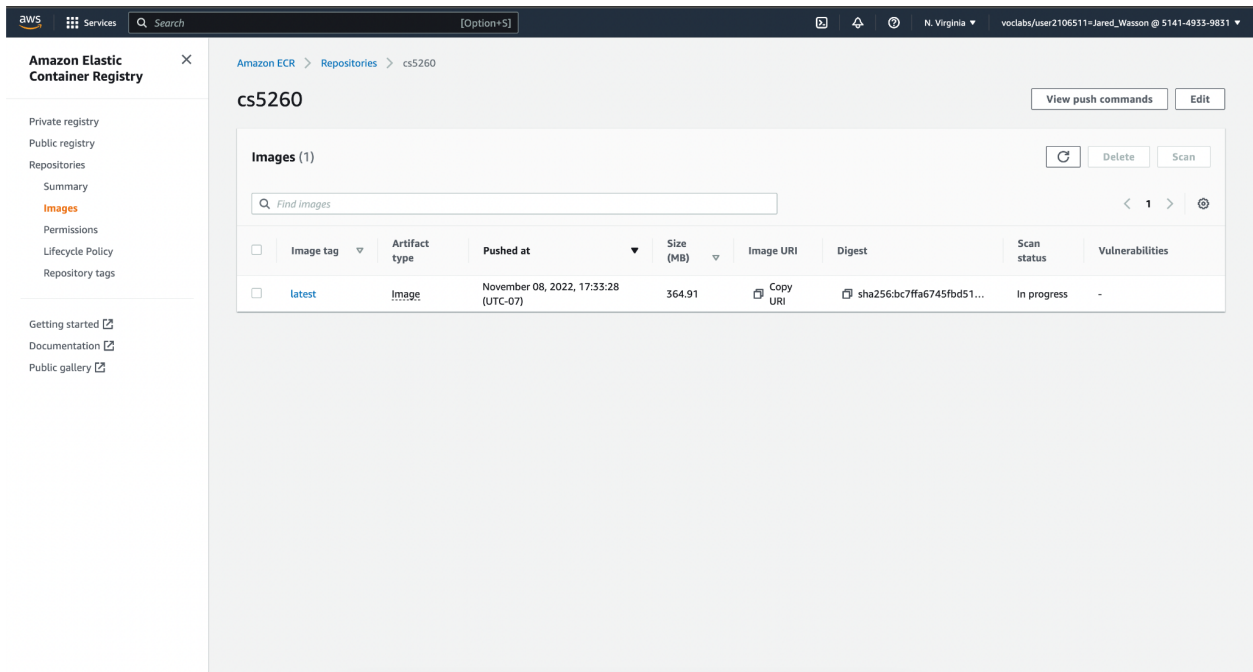
```
1 from unittests import *
2 import json
3 import os
4 import sys
5
6 testJson = {
7     "type": "create", "requestId": "6f6121c7-14d6-4d8a-a431-9f1a6dde8808", "widgetId": "7f0b5fd22-876f-42f2-937d-cdf048caec2a", "ow
8     "jared-wasson, 2 hours ago + added some tests
9     s3 = boto3.resource('s3')
10     my_bucket3 = s3.Bucket('jared-blue-bucket-3')
11
12 def test_checkForWidgetRequests():
13     assert checkForWidgetRequests() == '1612306375850'
14     print('test_checkForWidgetRequests() passed')
15
16 def test_createS3():
17     key = False
18     s3 = boto3.resource('s3')
19     my_bucket3 = s3.Bucket('jared-blue-bucket-3')
20     with open('./1612306375850.json') as f:
21         data = json.load(f)
22         createS3(data, '1612306375850')
23         widgetsInBucket = []
24         for file in my_bucket3.objects.all():
25             widgetInBucket.append(file.key)
```

The terminal at the bottom shows the execution of the script. Two instances of the script are running, each in a separate terminal window. The first instance is running 'python consumer.py s3' and the second is running 'python consumer.py dynamo cs5260-requests'. Both instances are running successfully.

Docker container running



Ecr



Commit history

Commits on Nov 8, 2022

finished

Jared-Wasson committed 1 minute ago

38b5538

added some tests

Jared-Wasson committed 4 hours ago

5711c72

Commits on Nov 5, 2022

missed sqs stuff

Jared-Wasson committed 3 days ago

7fcf340

added update and delete

Jared-Wasson committed 3 days ago

240fd5f

Commits on Nov 4, 2022

add sqs queue

Jared-Wasson committed 4 days ago

7fc6696

Commits on Oct 26, 2022

add pdf

Jared-Wasson committed 13 days ago

582142e

added unit tests

Jared-Wasson committed 13 days ago

48c0fee

Commits on Oct 25, 2022

added readme

Jared-Wasson committed 14 days ago

2ea4ca9

added logs

Jared-Wasson committed 14 days ago

c42d845

inital work

Jared-Wasson committed 14 days ago

67ddc36

Questions

#1: Update and delete could fail because the consumer polls from the SQS which does not guarantee any order of the messages. This means that an update could be pulled before the create. Another thing to note is that when the producer adds messages onto the SQS there is also a small delay from when the message can be accessed.

2# With using SQS, Design patterns should have a dead letter queue to make sure that unprocessed messages will eventually be processed if they continue to fail. Consumers should constantly be polling the SQS to confirm if there are messages that are available to process.