Cloud Application Development

Week -10

WEEK-10: AWS, SIMPLE QUEUE SERVICE(SQS) Design a protocol and use Simple Queue Service (SQS) to implement the barrier synchronization after the first phase.

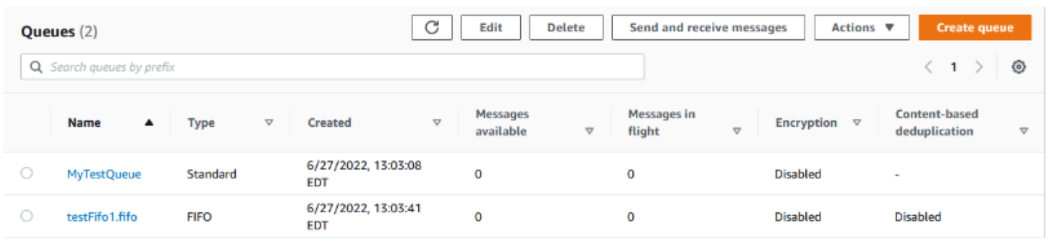
Prerequisites

Before you begin, complete the steps in Setting up Amazon SQS.

Understanding the Amazon SQS console

When you open the Amazon SQS console, choose **Queues** from the navigation pane. The **Queues** page provides information about all of your queues in the active region.

Each queue entry provides essential information about the queue, including its type and key attributes. Standard queues, optimized for maximum throughput and best-effort message ordering, are distinguished from First-In-First-Out (FIFO) queues, which prioritize message ordering and uniqueness for applications requiring strict message sequencing.



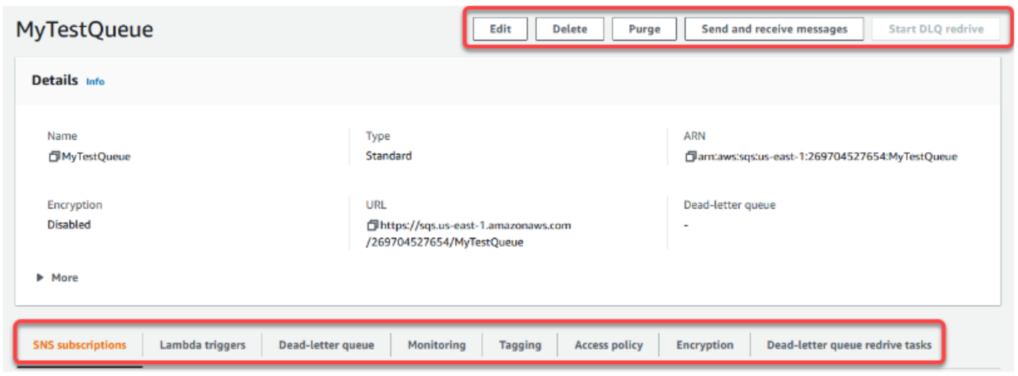
	Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication
○	MyTestQueue	Standard	6/27/2022, 13:03:08 EDT	0	0	Disabled	-
○	testFifo1.fifo	FIFO	6/27/2022, 13:03:41 EDT	0	0	Disabled	Disabled

Interactive elements and actions

From the Queues page, you have multiple options for managing your queues:

1. **Quick Actions** – Adjacent to each queue name, a dropdown menu offers quick access to common actions such as sending messages, viewing or deleting messages, configuring triggers, and deleting the queue itself.

2. **Detailed View and Configuration** – Clicking on a queue name opens its Details page, where you can delve deeper into queue settings and configurations. Here, you can adjust parameters like message retention period, visibility timeout, and maximum message size to tailor the queue to your application's requirements.



MyTestQueue

Details

Name	Type	ARN
MyTestQueue	Standard	arn:aws:sqs:us-east-1:269704527654:MyTestQueue
Encryption	**URL**	**Dead-letter queue**
Disabled	https://sqs.us-east-1.amazonaws.com/269704527654/MyTestQueue	-

Region selection and resource tags

Ensure you're in the correct AWS Region to access and manage your queues effectively. Additionally, consider utilizing resource tags to organize and categorize your queues, enabling better resource management, cost allocation, and access control within your AWS shared environment.

By leveraging the features and functionalities offered within the Amazon SQS console, you can efficiently manage your messaging infrastructure, optimize queue performance, and ensure reliable message delivery for your applications.

Protocol Design for Barrier Synchronization with SQS

Overview

Barrier synchronization requires that all processes reach a defined checkpoint (barrier) before any can proceed further. Using SQS, each process sends a message to a queue to signal that it has completed the first phase. When the queue accumulates the expected number of messages (equal to the number of processes), each process is notified, allowing them to proceed to the next phase.

Key Assumptions

1. **N processes** need to be synchronized. Each process must complete Phase 1 before moving to Phase 2.
2. Each process is aware of the total number of participating processes NNN.
3. AWS SQS will serve as a central point for gathering messages that indicate the completion of Phase 1.

Steps in the Protocol

1. **Set Up the SQS Queue:**
   - Create an SQS queue named `BarrierQueue` to hold the completion messages from each process.
   - Configure the queue with appropriate message retention and visibility timeout settings to manage Phase 1 and Phase 2 duration.
2. **Process Phase 1 Execution:**
   - Each process completes its Phase 1 operations independently.
   - Upon reaching the end of Phase 1, each process constructs a message that signals it has completed Phase 1.
3. **Message Sending (Completion Notification):**
   - Each process sends a message to the `BarrierQueue` upon finishing Phase 1. The message can contain information like a unique process ID or a timestamp.
   - This message serves as the "check-in" signal for that process.
4. **Barrier Check (Counting Messages):**
   - A monitoring mechanism (can be a dedicated process or a lambda function) polls the `BarrierQueue`.
   - This monitor counts the messages in `BarrierQueue`. Once it detects NNN messages (one for each process), it means all processes have reached the barrier.
   - The monitor can then send a "barrier release" message to another queue (e.g., `ReleaseQueue`) or broadcast a release signal to each process directly.
5. **Processes Waiting at the Barrier:**
   - After each process sends its message, it continuously checks for the release signal.
   - Each process can poll the `ReleaseQueue` to wait for the barrier release message. Once the release signal is received, processes can proceed to Phase 2.
6. **Clean Up:**
   - After all processes move past the barrier, the messages in `BarrierQueue` can be deleted, and any necessary state reset for future synchronizations.

Sample Implementation Code

Here's a simplified example in Python, using the `boto3` library, to demonstrate the steps each process might take:

```
import boto3

import time



sqs = boto3.client('sqs', region_name='us-east-1')



def send_completion_message(queue_url, process_id):
    sqs.send_message(QueueUrl=queue_url, MessageBody=f"Process
{process_id} completed Phase 1")


def poll_for_barrier_release(release_queue_url):
    while True:
        messages = sqs.receive_message(QueueUrl=release_queue_url,
MaxNumberOfMessages=1)
        if 'Messages' in messages:
            print("Barrier released! Moving to Phase 2.")
            return
        time.sleep(1)  # Polling interval


# Example usage
barrier_queue_url = 'https://sqs.us-east-1.amazonaws.com/your-
account-id/BarrierQueue'
release_queue_url = 'https://sqs.us-east-1.amazonaws.com/your-
account-id/ReleaseQueue'
process_id = 'your-process-id'


# Step 1: Complete Phase 1
print("Process completed Phase 1.")


# Step 2: Send a completion message to BarrierQueue
send_completion_message(barrier_queue_url, process_id)
```

```
# Step 3: Wait for the release signal to proceed
poll_for_barrier_release(release_queue_url)
```

Additional Considerations

- **Visibility Timeout:** Set the visibility timeout of `BarrierQueue` messages to avoid messages being processed twice if there’s a delay in counting.
- **Lambda for Monitoring:** Consider using an AWS Lambda function to monitor `BarrierQueue`, count the messages, and send a release message to `ReleaseQueue` when the expected count NNN is reached.
- **Error Handling:** Ensure processes can handle potential delays or errors in message delivery by using retries or by implementing fallback synchronization logic.