# SQS Queue Listener Implementation Guide

## Overview

This guide explains how to listen to AWS SQS queues and process messages in the billing-service.

## Implementation Details

### 1. SqsWorkOrderQueueAdapter

The main listener class that polls messages from SQS queue.

**Location**: `infrastructure/adapter/out/messaging/SqsWorkOrderQueueAdapter.java`

**Key Features:**

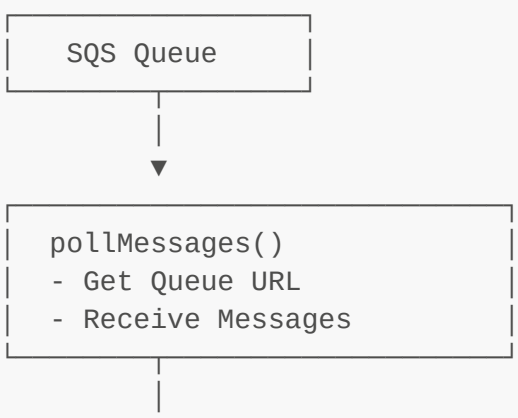
- **Scheduled Polling**: Automatically polls queue every 10 seconds using `@Scheduled`
- **Long Polling**: Uses 20-second wait time for efficient message retrieval
- **Batch Processing**: Receives up to 10 messages per poll
- **Automatic Deletion**: Deletes messages after successful processing
- **Error Handling**: Logs errors and prevents message loss

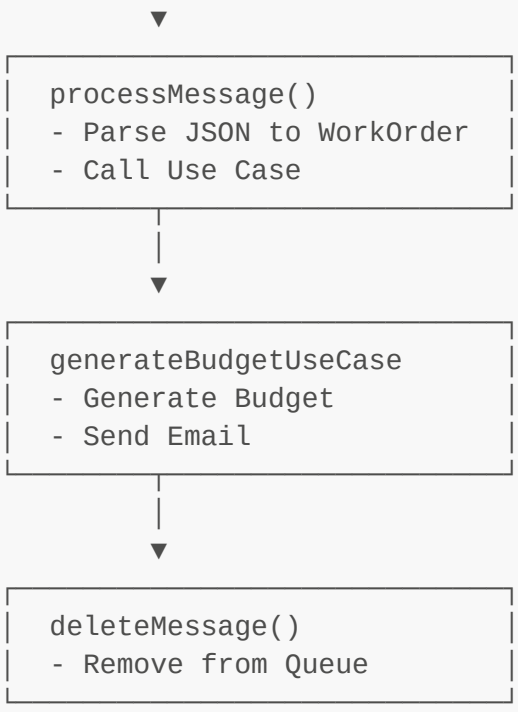
**How It Works:**

```
@Scheduled(fixedDelay = 10000) // Runs every 10 seconds
public void pollMessages() {
    // 1. Get queue URL
    // 2. Receive messages (up to 10)
    // 3. Process each message
    // 4. Delete processed messages
}
```

PROF

### 2. Message Processing Flow






### 3. Message Format

Expected JSON format for work order messages:

```
{
  "workOrderId": "70e6215d-b5c6-4896-987c-144a6333a2b3",
  "customer": {
    "Id": "70e6215d-b5c6-4896-987c-144a6312a2b3",
    "name": "John Doe",
    "email": "john.doe@example.com",
    "documentNumber": "12345678900"
  },
  "vehicle": {
    "Id": "80e6215d-b5c6-4896-987c-144a6312a2b4",
    "make": "Toyota",
    "model": "Camry",
    "year": 2020
  },
  "parts": [
    {
      "part": "Oil Filter",
      "quantity": 2,
      "amount": 15.0
    }
  ],
  "services": [
    {
      "service": "Oil Change",
      "quantity": 1,
      "amount": 50.0
```

PROF

```
    }
  ]
}
```

## 4. Configuration

**application.yml**

```yaml
# AWS Configuration
aws:
  region: us-east-1
  credentials:
    access-key: ${AWS_ACCESS_KEY}
    secret-key: ${AWS_SECRET_KEY}
  sqs:
    work-order-queue: work-order-queue

# Scheduling Configuration
spring.task.scheduling:
  enabled: true
```

**Environment Variables**

```
export AWS_ACCESS_KEY=your-access-key
export AWS_SECRET_KEY=your-secret-key
```

## 5. Testing the Implementation

PROF

**Method 1: Using Test Controller**

Send a test message via REST API:

```
# Send test work order
curl -X POST http://localhost:8080/api/test/send-test-message

# Send custom work order
curl -X POST http://localhost:8080/api/test/send-custom-message \
  -H "Content-Type: application/json" \
  -d '{
    "workOrderId": "70e6215d-b5c6-4896-987c-144a6333a2b3",
    "customer": {
      "Id": "70e6215d-b5c6-4896-987c-144a6312a2b3",
      "name": "John Doe",
      "email": "john.doe@example.com",
      "documentNumber": "12345678900"
```

```
    },
    "vehicle": {
      "Id": "80e6215d-b5c6-4896-987c-144a6312a2b4",
      "make": "Toyota",
      "model": "Camry",
      "year": 2020
    },
    "parts": [
      {
        "part": "Oil Filter",
        "quantity": 2,
        "amount": 15.00
      }
    ],
    "services": [
      {
        "service": "Oil Change",
        "quantity": 1,
        "amount": 50.00
      }
    ]
  }'
```

**Method 2: Using AWS CLI**

```
# Get queue URL
aws sqs get-queue-url --queue-name work-order-queue

# Send message
aws sqs send-message \
  --queue-url <your-queue-url> \
  --message-body '{
    "workOrderId": "70e6215d-b5c6-4896-987c-144a6333a2b3",
    "customer": {
      "Id": "70e6215d-b5c6-4896-987c-144a6312a2b3",
      "name": "John Doe",
      "email": "john.doe@example.com"
    },
    ...
  }'
```

PROF

**Method 3: Using AWS Console**

1. Go to AWS SQS Console
2. Select your queue
3. Click "Send and receive messages"
4. Paste the JSON message
5. Click "Send message"

## 6. Monitoring and Logs

The listener provides detailed logging:

```
2026-02-07 10:00:00 INFO  - Polling messages from SQS queue: work-order-
queue
2026-02-07 10:00:01 INFO  - Received 3 messages from queue
2026-02-07 10:00:01 INFO  - Processing message: abc-123-def
2026-02-07 10:00:01 INFO  - Message body: {"workOrderId":"..."}
2026-02-07 10:00:02 INFO  - Successfully processed work order:
70e6215d-...
2026-02-07 10:00:02 INFO  - Message deleted from queue
```

## 7. Error Handling

### Successful Processing

- Message is processed
- Budget is generated
- Email is sent
- Message is deleted from queue

### Failed Processing

- Error is logged
- Message remains in queue
- Will be retried after visibility timeout
- After max retries, moves to DLQ (if configured)

## 8. Performance Tuning

PROF

### Adjust Polling Frequency

```
@Scheduled(fixedDelay = 5000) // Poll every 5 seconds
```

### Adjust Batch Size

```
ReceiveMessageRequest.builder()
    .maxNumberOfMessages(5) // Receive up to 5 messages
```

### Adjust Wait Time

```
```

```
ReceiveMessageRequest.builder()
    .waitTimeSeconds(10) // Wait up to 10 seconds
```

## 9. Alternative: Spring Cloud AWS

For a more Spring-native approach, you can use Spring Cloud AWS:

**Add Dependency**

```
<dependency>
    <groupId>io.awspring.cloud</groupId>
    <artifactId>spring-cloud-aws-messaging</artifactId>
</dependency>
```

**Simplified Listener**

```
@SqsListener("${aws.sqs.work-order-queue}")
public void receiveMessage(String message) {
    WorkOrder workOrder = parseMessage(message);
    generateBudgetUseCase.generateAndSendBudget(workOrder);
}
```

## 10. Production Considerations

**Dead Letter Queue (DLQ)**

Configure a DLQ for failed messages:

PROF

```
aws sqs create-queue --queue-name work-order-queue-dlq

aws sqs set-queue-attributes \
  --queue-url <your-queue-url> \
  --attributes '{
    "RedrivePolicy": "{
      \"deadLetterTargetArn\":\"arn:aws:sqs:...:work-order-queue-dlq\",
      \"maxReceiveCount\":\"3\"
    }"
  }'
```

**Message Visibility Timeout**

Set appropriate visibility timeout (default: 30 seconds):

```
aws sqs set-queue-attributes \
  --queue-url <your-queue-url> \
  --attributes VisibilityTimeout=300  # 5 minutes
```

**Monitoring**

- Set up CloudWatch alarms for queue depth
- Monitor processing time
- Track error rates

# Troubleshooting

## Issue: Messages not being received

**Check**:

1. Queue name is correct in configuration
2. AWS credentials have SQS permissions
3. Scheduling is enabled (`@EnableScheduling`)
4. Application logs for errors

## Issue: Messages deleted but not processed

**Check**:

1. Message format matches expected structure
2. All required fields are present
3. UUID strings are valid
4. Numeric values are properly formatted

## Issue: Too many API calls

PROF

**Solution**:

- Increase polling delay
- Use long polling (waitTimeSeconds)
- Implement exponential backoff when queue is empty

# Summary

The SQS listener implementation provides:

- ✅ Automatic message polling
- ✅ Batch processing
- ✅ Error handling
- ✅ Message parsing
- ✅ Automatic deletion
- ✅ Comprehensive logging

- ✅ Easy testing via REST API

The listener will automatically start when the application runs and continuously poll for new messages to process.