Issue #69: AWS SQS Queue Setup - Complete Implementation Guide

Issue: #69 - Set Up AWS SQS Queue (Development Environment)
Author: Abhishek Singh
Date: October 19, 2025
Status: Ready for Implementation

Executive Summary

This document provides a complete implementation guide for setting up an AWS SQS Standard Queue for the automotive telemetry data pipeline. The queue serves as a critical buffer between the FastAPI receiver (Issue #70) and the batch consumer (Issue #71), enabling decoupled, scalable message processing.

Deliverables:

1. Python setup script (`setup_sqs.py`)
2. Complete setup documentation
3. IAM policy templates
4. Testing and validation procedures
5. Cost analysis and monitoring guidance

1. Overview & Architecture Context

1.1 Queue Role in Data Pipeline

```
FastAPI Receiver  →  SQS Queue  →  Batch Consumer  →  JSONL Storage
    (Issue #70)     (Issue #69)     (Issue #71)
```

Purpose:

- **Decoupling**: Allows receiver and consumer to scale independently
- **Buffering**: Handles traffic spikes without data loss
- **Reliability**: 4-day message retention for disaster recovery
- **Scalability**: Unlimited throughput with Standard queue

1.2 Design Decisions (from ADR 0001)

Queue Type: Standard (not FIFO)

Rationale:

- **Cost**: $0.40/million requests vs $0.50/million for FIFO
- **Throughput**: Unlimited vs 3,000 msg/sec for FIFO
- **Ordering**: Not required for hourly analytics
- **Duplicates**: <0.1% expected, acceptable for use case

Configuration Parameters:

Parameter	Value	Rationale
Message Retention	4 days	Disaster recovery window
Visibility Timeout	30 seconds	Processing time allowance
Long Polling	20 seconds	Cost optimization (95% reduction)
Max Message Size	256 KB	Sufficient for telemetry
Encryption	Optional (SSE)	Enable for production

2. Prerequisites

2.1 AWS Account Requirements

- Active AWS account (Free Tier sufficient)
- Access to SQS service in us-east-1 region
- IAM user with appropriate permissions

2.2 Local Environment Setup

Install AWS CLI:

```
# macOS
brew install awscli

# Linux
pip install awscli

# Windows
# Download from: https://aws.amazon.com/cli/
```

Configure AWS Credentials:

```
aws configure
# Provide:
```

```
# - AWS Access Key ID
# - AWS Secret Access Key
# - Default region: us-east-1
# - Default output: json
```

Install Python Dependencies:

```
pip install boto3
```

2.3 Required IAM Permissions

Your IAM user/role needs:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sqs:CreateQueue",
        "sqs:GetQueueUrl",
        "sqs:GetQueueAttributes",
        "sqs:SetQueueAttributes",
        "sqs:TagQueue",
        "sqs:SendMessage",
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage"
      ],
      "Resource": "arn:aws:sqs:us-east-1:*:telemetry-queue-*"
    }
  ]
}
```

3. Setup Methods

3.1 Method 1: Using Python Script (Recommended)

Step 1: Place Script in Repository

The provided `setup_sqs.py` script automates the entire setup process.

Location: `scripts/setup_sqs.py`

Step 2: Run Setup Script

Basic Setup (recommended for most users):

```
python scripts/setup_sqs.py
```

Advanced Options:

```
# Custom queue name
python scripts/setup_sqs.py --queue-name my-telemetry-queue

# Different region
python scripts/setup_sqs.py --region us-west-2

# Enable encryption
python scripts/setup_sqs.py --encrypt

# Output environment config
python scripts/setup_sqs.py --output-env

# Output IAM policies
python scripts/setup_sqs.py --output-iam

# Use specific AWS profile
python scripts/setup_sqs.py --profile dev-account

# Skip test messages
python scripts/setup_sqs.py --no-test

# All options combined
python scripts/setup_sqs.py \\
  --queue-name telemetry-queue-dev \\
  --region us-east-1 \\
  --encrypt \\
  --output-env \\
  --output-iam
```

Step 3: Expected Output

```
================================================================================
Creating SQS Queue: telemetry-queue-dev
================================================================================

Creating queue in region: us-east-1
Configuration:
  - Message Retention: 345600s (4.0 days)
  - Visibility Timeout: 30s
  - Long Polling: 20s
  - Max Message Size: 256 KB
✓ Server-side encryption (SSE) enabled

✅ Queue created successfully!
```

```
Queue URL: https://sqs.us-east-1.amazonaws.com/123456789012/telemetry-queue-dev
Queue ARN: arn:aws:sqs:us-east-1:123456789012:telemetry-queue-dev

================================================================================
TESTING MESSAGE FLOW
================================================================================
Sending test message...
✅ Test message sent successfully!
Message ID: 12345678-1234-1234-1234-123456789012
MD5 of body: abc123def456...

Receiving test message...
✅ Test message received!
Message ID: 12345678-1234-1234-1234-123456789012
Vehicle ID: test-vehicle-001
Timestamp: 2025-10-19T18:00:00.000Z
✅ Test message deleted

================================================================================
SQS QUEUE SETUP SUMMARY
================================================================================

✅ Queue Name: telemetry-queue-dev
✅ Region: us-east-1
✅ Queue URL: https://sqs.us-east-1.amazonaws.com/123456789012/telemetry-queue-dev
✅ Queue ARN: arn:aws:sqs:us-east-1:123456789012:telemetry-queue-dev

✅ Environment config written to: .env.sqs
✅ IAM policies written to: iam-policies/
   - sqs-producer-policy.json
   - sqs-consumer-policy.json
```

3.2 Method 2: Using AWS Console (Manual)

Step 1: Navigate to AWS Console → SQS → Create queue

Step 2: Configure Queue

- **Type**: Standard
- **Name**: telemetry-queue-dev
- **Visibility timeout**: 30 seconds
- **Message retention**: 4 days (345600 seconds)
- **Receive wait time**: 20 seconds
- **Max message size**: 256 KB

Step 3: Add Tags

- Project: automotive-devops-platform
- Component: can-data-pipeline
- Environment: development

- Issue: #69

Step 4: Create queue and note the Queue URL

3.3 Method 3: Using AWS CLI

```
# Create queue
aws sqs create-queue \\
  --queue-name telemetry-queue-dev \\
  --attributes \\
    MessageRetentionPeriod=345600,\\
    VisibilityTimeout=30,\\
    ReceiveMessageWaitTimeSeconds=20,\\
    MaximumMessageSize=262144 \\
  --tags \\
    Project=automotive-devops-platform \\
    Component=can-data-pipeline \\
    Environment=development \\
    Issue=#69

# Get queue URL
aws sqs get-queue-url --queue-name telemetry-queue-dev

# Output:
# {
#   "QueueUrl": "https://sqs.us-east-1.amazonaws.com/123456789012/telemetry-queue-dev"
# }

# Verify queue attributes
aws sqs get-queue-attributes \\
  --queue-url &lt;QUEUE_URL&gt; \\
  --attribute-names All
```

4. Configuration Files

4.1 Environment Variables (.env)

Add to your `.env` file:

```
# AWS SQS Configuration (Issue #69)
AWS_REGION=us-east-1
SQS_QUEUE_URL=https://sqs.us-east-1.amazonaws.com/123456789012/telemetry-queue-dev
SQS_QUEUE_NAME=telemetry-queue-dev

# SQS Client Configuration
SQS_MAX_MESSAGES=100
SQS_WAIT_TIME_SECONDS=20
SQS_VISIBILITY_TIMEOUT=30
```

Security Note: Never commit `.env` to git. Add to `.gitignore`.

4.2 IAM Policy: Producer (FastAPI Receiver)

File: `iam-policies/sqs-producer-policy.json`

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSQSPublish",
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:GetQueueUrl",
        "sqs:GetQueueAttributes"
      ],
      "Resource": "arn:aws:sqs:us-east-1:ACCOUNT_ID:telemetry-queue-dev"
    }
  ]
}
```

Usage:

1. Replace `ACCOUNT_ID` with your AWS account ID
2. Attach to IAM role used by FastAPI receiver
3. Used in Issue #70 implementation

4.3 IAM Policy: Consumer (Batch Processor)

File: `iam-policies/sqs-consumer-policy.json`

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSQSConsume",
      "Effect": "Allow",
      "Action": [
        "sqs:ReceiveMessage",
        "sqs:DeleteMessage",
        "sqs:GetQueueAttributes",
        "sqs:GetQueueUrl"
      ],
      "Resource": "arn:aws:sqs:us-east-1:ACCOUNT_ID:telemetry-queue-dev"
    }
  ]
}
```

Usage:

1. Replace `ACCOUNT_ID` with your AWS account ID
2. Attach to IAM role used by batch consumer

3. Used in Issue #71 implementation

5. Validation & Testing

5.1 Test 1: AWS CLI Message Flow

Send Test Message:

```
aws sqs send-message \\
  --queue-url &lt;YOUR_QUEUE_URL&gt; \\
  --message-body '{"vehicle_id": "test-001", "timestamp": "2025-10-19T19:00:00Z", "test":

# Expected output:
# {
#   "MessageId": "12345678-abcd-1234-abcd-123456789012",
#   "MD5OfMessageBody": "abc123..."
# }
```

Receive Test Message:

```
aws sqs receive-message \\
  --queue-url &lt;YOUR_QUEUE_URL&gt; \\
  --max-number-of-messages 1 \\
  --wait-time-seconds 20

# Expected: JSON with Messages array
```

Delete Test Message:

```
aws sqs delete-message \\
  --queue-url &lt;YOUR_QUEUE_URL&gt; \\
  --receipt-handle &lt;RECEIPT_HANDLE&gt;
```

5.2 Test 2: Python Validation Script

Create `scripts/test_sqs_connection.py`:

```
import boto3
import json
from datetime import datetime
import sys

def test_sqs_connection(queue_url):
    sqs = boto3.client('sqs', region_name='us-east-1')

    # Send test message
    test_msg = {
        "timestamp": datetime.utcnow().isoformat() + "Z",
        "vehicle_id": "test-vehicle-001",
        "test": True
    }

    print(f"Sending test message...")
    response = sqs.send_message(
        QueueUrl=queue_url,
        MessageBody=json.dumps(test_msg)
    )
    print(f"✅ Sent. MessageId: {response['MessageId']}")

    # Receive test message
    print("Receiving message...")
    response = sqs.receive_message(
        QueueUrl=queue_url,
        MaxNumberOfMessages=1,
        WaitTimeSeconds=20
    )

    if 'Messages' in response:
        msg = response['Messages'][0]
        print(f"✅ Received. MessageId: {msg['MessageId']}")

        # Delete message
        sqs.delete_message(
            QueueUrl=queue_url,
            ReceiptHandle=msg['ReceiptHandle']
        )
        print("✅ Deleted")
        print("\\n✅ SQS test PASSED!")
    else:
        print("⚠  No messages received")

if __name__ == '__main__':
    if len(sys.argv) &lt; 2:
        print("Usage: python test_sqs_connection.py &lt;QUEUE_URL&gt;")
        sys.exit(1)
    test_sqs_connection(sys.argv[1])
```

Run Test:

```
python scripts/test_sqs_connection.py &lt;YOUR_QUEUE_URL&gt;
```

5.3 Test 3: CloudWatch Metrics Validation

Navigate to CloudWatch Console:

1. AWS Console → CloudWatch → Metrics
2. Select "SQS" namespace
3. Select queue: `telemetry-queue-dev`

Expected Metrics (after sending test messages):

- `NumberOfMessagesSent`: Should be > 0

- `NumberOfMessagesReceived`: Should be > 0
- `NumberOfMessagesDeleted`: Should match received
- `ApproximateNumberOfMessages`: Should be 0 (if all deleted)

6. Monitoring & Observability

6.1 CloudWatch Metrics (Automatic)

AWS automatically publishes these metrics every 5 minutes:

Queue Depth:

- `ApproximateNumberOfMessages` - Available messages
- `ApproximateNumberOfMessagesNotVisible` - In-flight (processing)
- `ApproximateNumberOfMessagesDelayed` - Delayed messages

Throughput:

- `NumberOfMessagesSent` - Messages published
- `NumberOfMessagesReceived` - Messages polled
- `NumberOfMessagesDeleted` - Successfully processed

Age:

- `ApproximateAgeOfOldestMessage` - Backlog age (seconds)

6.2 CloudWatch Alarms (Recommended)

Alarm 1: High Queue Depth

```
aws cloudwatch put-metric-alarm \\
  --alarm-name telemetry-queue-depth-high \\
  --alarm-description "Alert when queue depth &gt; 1000" \\
  --metric-name ApproximateNumberOfMessages \\
  --namespace AWS/SQS \\
  --statistic Average \\
  --period 300 \\
  --evaluation-periods 2 \\
  --threshold 1000 \\
  --comparison-operator GreaterThanThreshold \\
  --dimensions Name=QueueName,Value=telemetry-queue-dev
```

Alarm 2: Old Messages (Backlog)

```
aws cloudwatch put-metric-alarm \\
  --alarm-name telemetry-queue-old-messages \\
  --alarm-description "Alert when oldest message &gt; 5 minutes" \\
  --metric-name ApproximateAgeOfOldestMessage \\
  --namespace AWS/SQS \\
  --statistic Maximum \\
  --period 300 \\
  --evaluation-periods 2 \\
  --threshold 300 \\
  --comparison-operator GreaterThanThreshold \\
  --dimensions Name=QueueName,Value=telemetry-queue-dev
```

7. Cost Analysis

7.1 Development Environment Estimate

Assumptions:

- 50 messages/second
- 24/7 operation
- 30 days/month
- Long polling enabled (20s)

Calculations:

```
Total messages: 50 msg/s × 86,400 s/day × 30 days = 129,600,000 messages

With long polling (20s):
- Empty receives reduced by ~95%
- Effective billable requests: ~130M

Cost:
- SQS Standard: $0.40 per 1M requests
- Monthly cost: 130M × $0.40/M = $52/month
```

7.2 Cost Optimization Techniques

1. **Long Polling** (Implemented)
    - Reduces empty receives by 95%
    - Saves: ~$50/month at this scale
2. **Batch Operations** (Issue #71)
    - Receive up to 100 messages per API call
    - Reduces API calls by 100x
3. **Standard vs FIFO**
    - Standard: $0.40/M
    - FIFO: $0.50/M
    - Savings: 20%

Total Monthly Cost: ~$52/month (with optimizations)

8. Security Best Practices

8.1 Least Privilege IAM

- ✅ Use specific resource ARNs (not wildcards)
- ✅ Separate producer and consumer policies
- ✅ Rotate access keys quarterly
- ✅ Use IAM roles for EC2/ECS (not access keys)

8.2 Encryption

Enable Server-Side Encryption:

```
python scripts/setup_sqs.py --encrypt
```

Or via AWS CLI:

```
aws sqs set-queue-attributes \\
  --queue-url &lt;QUEUE_URL&gt; \\
  --attributes SqsManagedSseEnabled=true
```

8.3 Access Control

- Queue is private by default (✅)
- Never make queue publicly accessible
- Use VPC endpoints for internal access (production)

8.4 Monitoring

- Enable CloudWatch alarms (see Section 6.2)
- Track failed deliveries
- Monitor queue age and depth

9. Troubleshooting

Issue: "AccessDenied" Error

Symptoms:

```
botocore.exceptions.ClientError: An error occurred (AccessDenied)
```

Causes:

1. Insufficient IAM permissions
2. Wrong resource ARN in policy
3. Credentials not configured

Solutions:

```
# Verify credentials
aws sts get-caller-identity

# Check IAM permissions
aws iam get-user-policy --user-name &lt;YOUR_USER&gt; --policy-name &lt;POLICY_NAME&gt;

# Attach SQS policy to user
aws iam attach-user-policy \\
  --user-name &lt;YOUR_USER&gt; \\
  --policy-arn arn:aws:iam::aws:policy/AmazonSQSFullAccess  # For testing only
```

Issue: Queue Not Created

Symptoms: Script runs but no queue appears in console

Solutions:

1. Check region mismatch:
   ```
   aws sqs list-queues --region us-east-1
   ```
2. Verify queue name doesn't conflict:
   ```
   python scripts/setup_sqs.py --queue-name telemetry-queue-dev-v2
   ```

Issue: Messages Not Received

Symptoms: Send succeeds, receive times out

Solutions:

1. Verify long polling enabled:
   ```
   aws sqs get-queue-attributes \\
     --queue-url &lt;QUEUE_URL&gt; \\
     --attribute-names ReceiveMessageWaitTimeSeconds
   ```
2. Check visibility timeout (messages in-flight):
   ```
   aws sqs get-queue-attributes \\
     --queue-url &lt;QUEUE_URL&gt; \\
     --attribute-names ApproximateNumberOfMessagesNotVisible
   ```

10. Next Steps

Immediate (After Issue #69)

1. ✅ Save queue URL to `.env` file
2. ✅ Commit IAM policy templates to repo
3. ✅ Document queue configuration in architecture docs

Issue #70: Implement SQS Publisher

Tasks:

- Add boto3 SQS client to FastAPI receiver
- Publish messages after validation
- Handle AWS SDK errors gracefully
- Add retry logic with exponential backoff

Issue #71: Build SQS Consumer

Tasks:

- Create async polling loop with long polling
- Receive messages in batches (up to 100)
- Process and write to JSONL
- Delete messages after successful write
- Implement error handling and logging

Issue #72: Add Latency Tracking

Tasks:

- Calculate queue latency (SQS timestamps)
- Calculate processing latency
- Log P50/P95/P99 metrics per batch
- Track end-to-end latency

11. Interview Talking Points

Q: "Why did you choose SQS Standard over FIFO?"

A: "For this MVP analytics use case:

- **Standard** provides unlimited throughput vs FIFO's 3,000 msg/sec limit
- **Cost**: 20% cheaper ($0.40 vs $0.50 per million)
- **Ordering not required**: Hourly aggregation doesn't need strict ordering
- **Duplicates acceptable**: Expected <0.1% duplicate rate, handled downstream
- **Documented upgrade triggers**: Will migrate to FIFO if duplicate rate exceeds 0.5%"

Q: "How does long polling reduce costs?"

A: "Long polling (20-second wait):

- **Reduces empty receives by ~95%**: Client waits for messages instead of immediate return
- **Fewer API calls**: Less network overhead and API charges
- **Cost savings**: ~$50/month at 50 msg/s throughput
- **Trade-off**: Slight increase in latency (acceptable for batch processing)"

Q: "What monitoring would you add in production?"

A: "CloudWatch alarms for:

1. **Queue depth > 1,000**: Indicates consumer falling behind
2. **Oldest message > 5 minutes**: Backlog building up
3. **Failed deliveries**: Track message processing errors
   Plus custom metrics: P95 latency, throughput trends, duplicate rate"

12. Acceptance Criteria Checklist

- [ ] SQS Standard queue created in us-east-1
- [ ] Queue name: `telemetry-queue-dev`
- [ ] Message retention: 4 days (345,600 seconds)
- [ ] Visibility timeout: 30 seconds
- [ ] Long polling: 20 seconds
- [ ] Max message size: 256 KB
- [ ] Queue URL saved to `.env` file
- [ ] Queue ARN documented
- [ ] IAM policies generated (producer + consumer)
- [ ] Test message sent successfully
- [ ] Test message received successfully

- [ ] Test message deleted successfully
- [ ] CloudWatch metrics visible
- [ ] Cost estimate documented
- [ ] Security best practices documented
- [ ] Troubleshooting guide complete

13. References

- **AWS SQS Documentation**: https://docs.aws.amazon.com/sqs/
- **boto3 SQS Reference**: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sqs.html
- **ADR 0001**: `docs/adr/0001-ingestion-transport.md`
- **Data Flow Architecture**: `docs/architecture/data-flow-architecture.md`
- **GitHub Issue #69**: Project issue tracker

Document Status: Ready for Implementation
Estimated Time: 1-2 hours
Closes Issue: #69