Use Case:

Trigger a Kubernetes Scaled Job (KSJ) when an S3 bucket event writes a message to an SQS queue. The job must read the message, execute custom logic (e.g., file processing), and delete the message only if successful. In case of failure, the message must not be lost and should be retried gracefully.

High-Level Flow:

S3 File Upload
↓
S3 Event Notification
↓
SQS Message Sent
↓
KEDA ScaledObject detects message
↓
KSJ Launched (Kubernetes Job Pod)
↓
Pod reads and processes message
↓
→ If Success → Delete SQS Message
→ If Failure → Log error, DO NOT delete message

System Architecture Diagram:

```
  +--------------+     +------------+     +----------------+
|  Developer/API | -----> |   S3    | -----> |    SQS      |
  +--------------+     +------------+     +----------------+
                                              |
                                              v
                                   +--------------------------+
                                   | KEDA ScaledObject |
                                   | (SQS Trigger)          |
                                   +--------------------------+
                                              |
                                              v
                                    +-------------------------+
                                    | K8s ScaledJob (KSJ)|
                                    +-------------------------+
                                              |
                  +----------------------+-----------------+----------------------+
                        |                        |                              |
                  Read SQS Msg          Execute Business Logic        Log + Retry Policy
                        |                          |                      |
                  On Success                  On Success                On Failure
                       |                         |                         |
                 Delete SQS Msg           Terminate Pod          Let message timeout (remain in
                                   queue)
```

Key Design Considerations:

1. 📬 **Message Visibility Timeout**

Strategy	Description
Dynamic	Adjust visibility timeout after reading message based on job execution
Fixed (Recommended)	Set VisibilityTimeout = Max job duration (e.g., 10 min)

Why?
Prevent other pods from picking up the same message mid-processing

✅ Default Behavior: Set to max processing time. Consider customizing per-job-type.

2.🔄 Message Lifecycle: Read → Process → Delete

- Always delete message only on success.
- If processing fails, do not delete: message will become visible again after timeout.
- Add unique message deduplication ID (if needed for idempotency).

3. ✋ Handling SIGTERM (Pod Termination)

- Ensure SIGTERM traps are in place in job containers:
- If terminated, do not delete message, visibility timeout ensures retry.
- Persist partial state if needed (to prevent duplicate writes on retry).

4. ❌ Handling Failures

Failure Scenario	Action
Business Logic Fails	Log error, return non-zero exit, let visibility timeout expire
Retry Count Exceeded	Move to DLQ (dead-letter queue) if configured
Network/Timeout Errors	Retry with backoff or leave for next pod instance

✅ Add x-death or similar message attributes (optional) for retry count tracking.

5. 📖 Logging & Observability

- Use centralized logging (e.g., Fluent Bit + ELK/Grafana).
- Log at:
  - Job start and finish
  - SQS message ID
  - Message content (or hash/metadata)
  - Any errors
- Include job_id, pod_name, and message_id for traceability.

6. ⚙️ KEDA Configuration (Recommended Defaults)

Config Option	Default	Description
pollingInterval	10s	How frequently KEDA checks SQS
cooldownPeriod	300s	Time to wait before scaling down
minReplicaCount	0	Only spawn pods when messages exist
maxReplicaCount	10 (custom)	Based on system capacity and throttling limits
queueLength	1	One message per job
scaleTargetRef	KubernetesJob	Tied to job spec (KSJ)

Sample KEDA ScaledJob YAML

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
  name: sqs-file-processor
spec:
  jobTargetRef:
    template:
      spec:
        containers:
        - name: processor
          image: registry/sqs-job:latest
        restartPolicy: Never
  pollingInterval: 10
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 5
  maxReplicaCount: 10
  triggers:
  - type: aws-sqs-queue
    metadata:
      queueURL: https://sqs.us-east-
1.amazonaws.com/1234567890/myqueue
      awsRegion: us-east-1
      queueLength: "1"
```

Sample Job Logic (Pseudocode)

```
def main():
    msg = sqs.receive_message(timeout=5min)
    try:
        process_file(msg.body)
        sqs.delete_message(msg.receipt_handle)
    except Exception as e:
        log_error("Failed to process message", msg.id, str(e))
        # Don't delete message — SQS will re-deliver after timeout

def process_file(file_info):
    # Do something useful with the S3 file
    pass
```

Edge Cases & Recommendations

- **Idempotency**: Make processing logic idempotent to handle retries safely.
- **DLQ**: Configure SQS with a dead-letter queue for poisoned messages.
- **Backpressure**: Monitor pod crash loops; use maxReplicaCount to prevent overload.
- **Metrics**: Use Prometheus to track:
    - Jobs launched
    - Messages processed
    - Failures / retries
- **Security**: Ensure IAM role has minimum SQS + S3 permissions required.