TEST 1

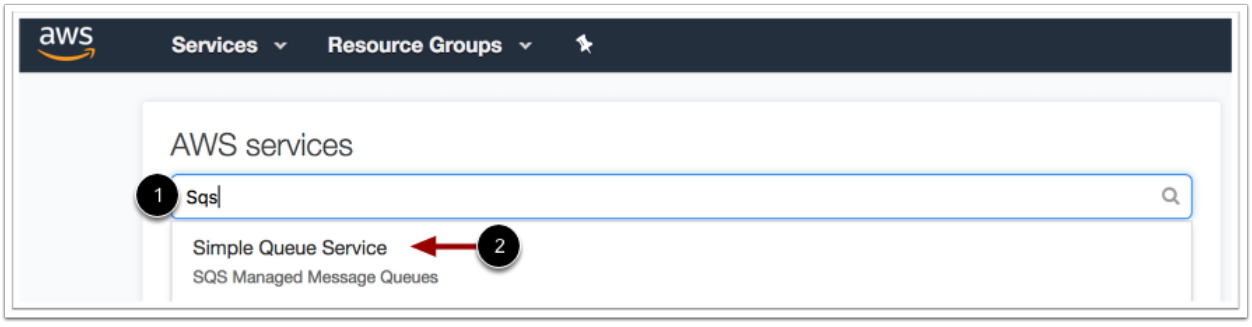
Setup and design a data pipeline that can parallely fetch and process 1 TB of data every single day and save it to mongo db
The above pipeline should be built on Kubernetes, please also explain how the workflow between the nodes will be managed.
Since we are going to handle large amounts of data, explain the strategy for redundancy and stability for the MongoDB cluster
Explain the stack that should be used to monitor the above pipeline.

Conclusion

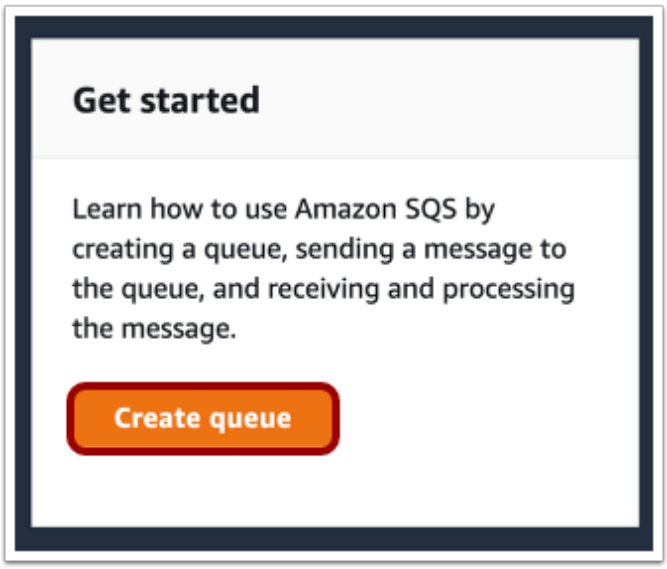
Considering the data is stored in s3 bucket and needs to be parsed and shipped to mongodb through python script

1. Write a bash script with awscli / python script with boto3 to load the filenames from s3 to aws Sqs. This approach will come in handy to keep track of data and avoid data loss. Queue will be flushed only if
each of the pipeline is successful

# Open Amazon SQS Console

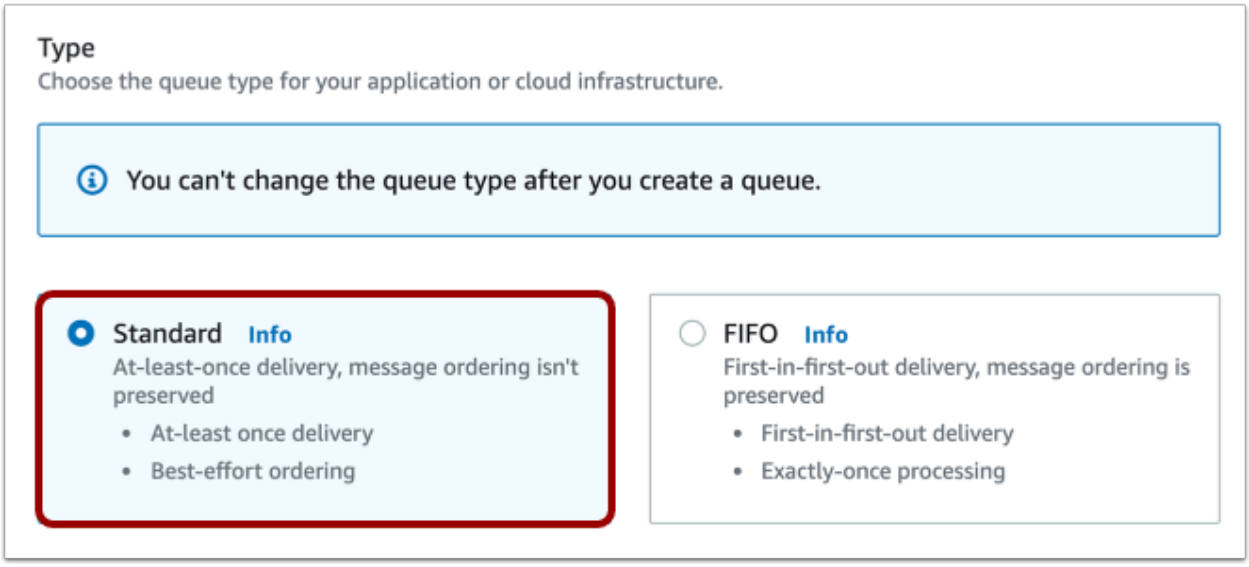


## Create New Queue




In the Amazon SQS console, click the **Create queue** button.

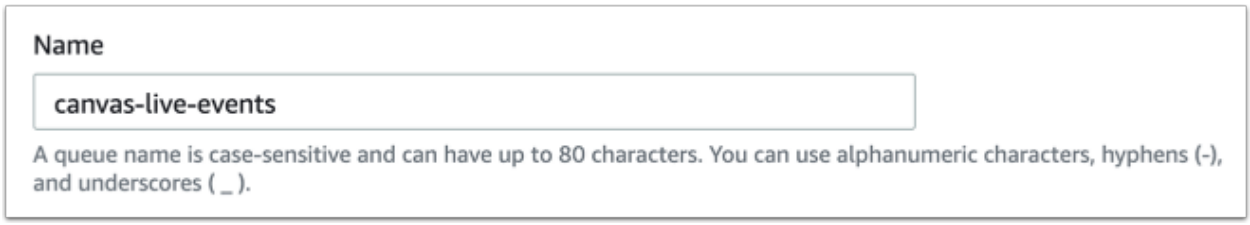
## Select Standard Queue




In the Type section, select the **Standard** option.

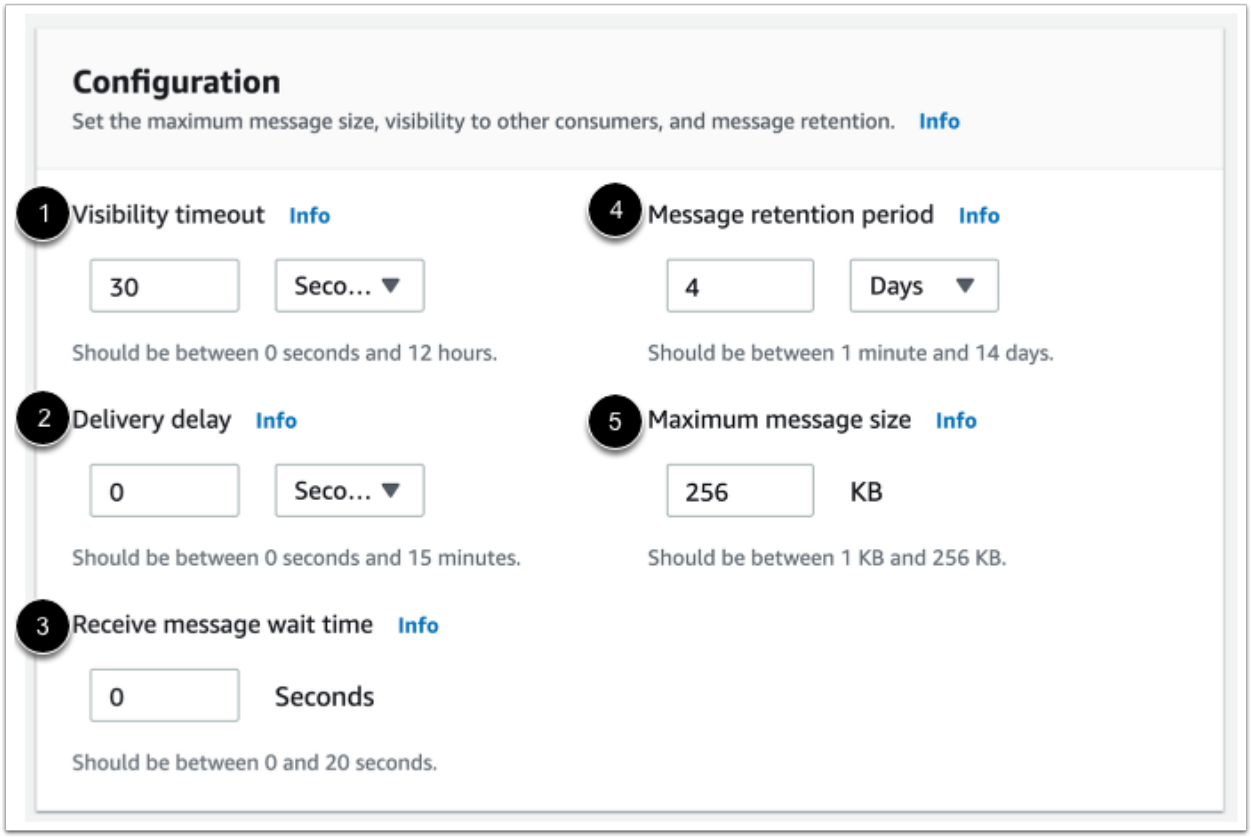
**Note**: FIFO Queues are not currently supported.

## Enter Queue Name



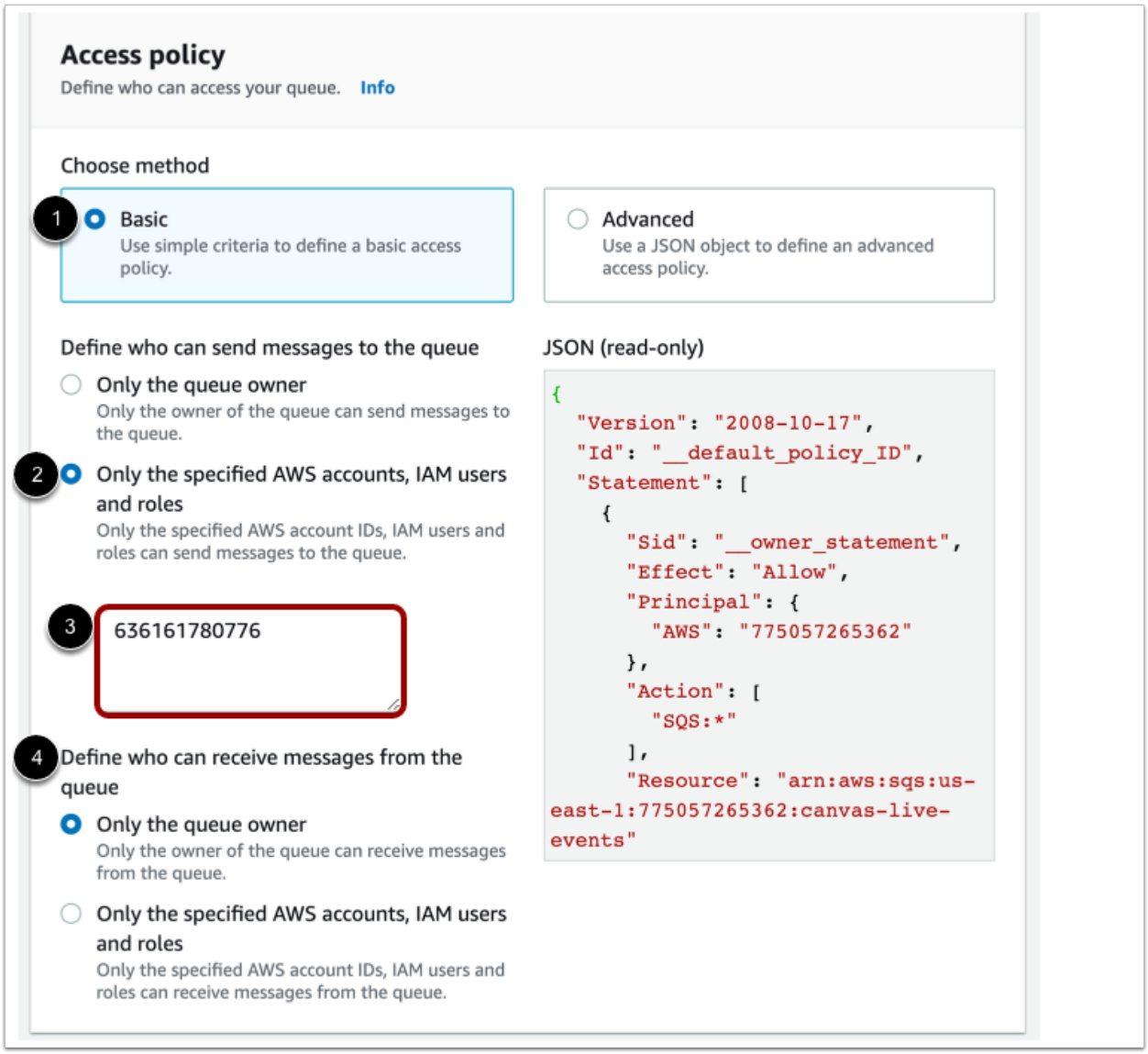

Enter a name for the queue. The name of the queue must begin with *canvas-live-events*.

## Enter Configuration Details




Enter the **Configuration** details. You can enter your preferences for visibility timeout [1], delivery delay [2], receive message wait time [3], message retention period [4], and maximum message size [5].

# Enter Access Policy Details




Enter the details for your access policy.

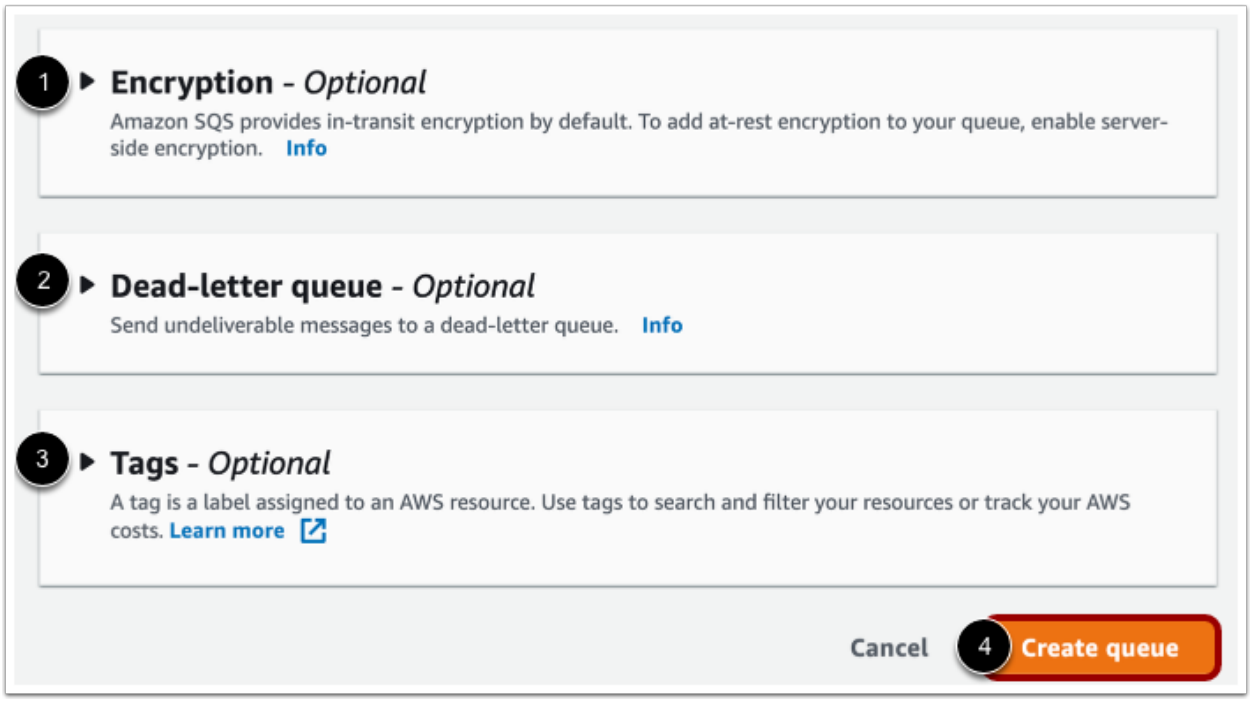
In the Choose method section, select the **Basic** option [1].

In the Define who can send messages to the queue section, select the **Only the specified AWS accounts, IAM users and roles** option [2].

In the account ID field, enter the account number **636161780776** [3]. This account number is required for the queue to receive Live Events data.

You can also select who will receive messages in the **Define who can receive messages from the queue** section [4].

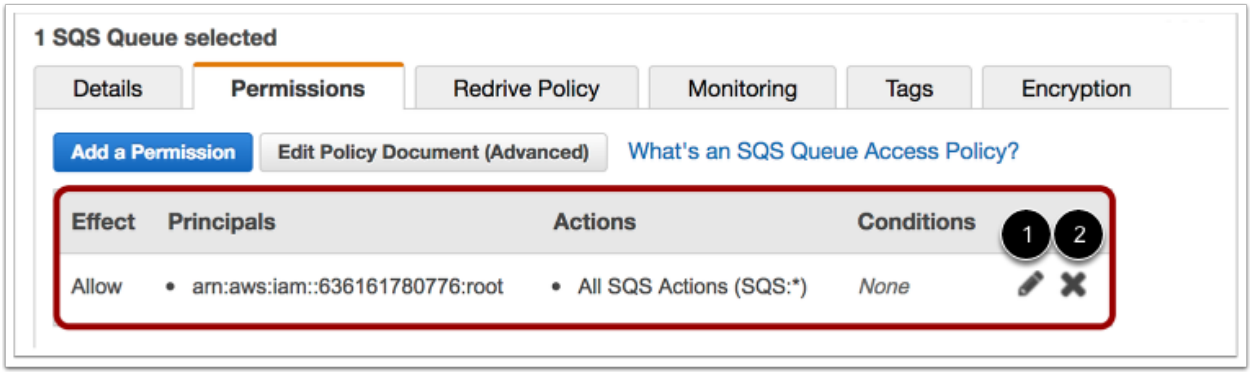
# Save Queue




You can add additional details in **Encryption** settings [1], **Dead-letter queue** settings [2], and **Tags** settings [3]. All of these settings are optional.

To create your queue, click the **Create queue** button [4].

# View Queue Permission




In the queue details area, the permission will display in the Permissions tab.

2. Once the queue is populated with objects. We will write a python script which will be using pymongo,boto3,json,sys modules to parse each object and feed it to mongoDB.

Python.py

```
import os
Import sys
Import boto3
from pprint import pprint

import json
from dotenv import load_dotenv
import pymongo

#AWS creds
Aws_secret_access_key='XXXXXXX'
aws_secret_access_key='XXXXXXXXXXXXXXX'

# Load config from a .env file:
load_dotenv(verbose=True)
MONGODB_URI = os.environ["MONGODB_URI"]

# Connect to your MongoDB cluster:
client = pymongo.MongoClient(MONGODB_URI)

# Get a reference to the "sample_mflix" database:
db = client["sample_mflix"]

# Get a reference to the "movies" collection:
movie_collection = db["movies"]


pipeline = [
  {
    "$match": {
        "title": "String data is here"
```

```
    }
  },
  {
    "$sort": {
        "year": pymongo.ASCENDING
    }
  },
]
results = movie_collection.aggregate(pipeline)
for movie in results:
  print(" * {title}, {first_castmember}, {year}".format(
        title=movie["title"],
        first_castmember=movie["cast"][0],
        year=movie["year"],
  ))
```

3. Dockerize a python script to run in kubernetes pods

```
FROM python3
RUN apt-get update
ADD Python.py /home/Python.py
PORT 3000
EXPOSE 3000
ENTRYPOINT ["python3 /home/Python.py"]
```

4. This procedure can take from 2 - 5 hours to process 1TB data depending on the processing power of the worker nodes from 4-16Gb of RAM, recommended to configure scale in and scale out policy to balance loads

5. The Replicas would be around 30-50 depending on the time frame needs to be achieved

Same can be achieved through AWS EMR and AWS Glue

# Monitoring

Pipelines can be monitored through ELK stack ,airflow, luigi,argo .

Below pipeline monitoring  is deployed with argo on kubernetes

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: test-pipeline-
spec:
  entrypoint: pipeline
  templates:
  - name: pipeline
    dag:
      tasks:
      - name: data-processing-1
        dependencies: []
        template: data-processing-1
        arguments:
          parameters: [{name: method, value: download_data.py}]
      - name: pca-1
        dependencies: [data-processing-1]
        template: pca-1
        arguments:
          parameters: [{name: method, value: pca.py}]
      - name: interactions-1
        dependencies: [pca-1]
        template: interactions-1
        arguments:
          parameters: [{name: method, value: first_order_interactions.py}]
  - name: data-processing-1
    inputs:
      parameters:
      - name: method
    container:
      imagePullPolicy: Always
      image:
      command: [python, "{{inputs.parameters.method}}"]
  - name: pca-1
    inputs:
      parameters:
      - name: method
    container:
      imagePullPolicy: Always
      image:
      command: [python, "{{inputs.parameters.method}}"]
  - name: interactions-1
```

You can have prometheus integrated to be notified after each executed pipeline and monitor each pipeline simultaneously .

| | |
|---|---|
| NAME | test-pipeline-2d4r6.data-processing-1 |
| TYPE | Pod |
| POD NAME | test-pipeline-2d4r6-3382753853 |
| PHASE | Succeeded |
| START TIME | 33 minutes ago |
| END TIME | 33 minutes ago |
| DURATION | 00:08 min |

YAML LOGS