

**Workload Management for Data pipelines**

Anmol Gandhi

AWS Professional Services – Data & ML Engineer

July 2021

Table of Contents

Abstract 3

Use Cases 3

Architecture 3

Pre-requisites & Demo 7

Workload Management Repo Structure. 8

Conclusion 9

# Abstract

This document describes a workload management solution which can be used in data lake pipelines to manage different types of workloads based on source-dataset and priority. It can help to reduce the throttling errors and unnecessary clogging of resource from a dominant source

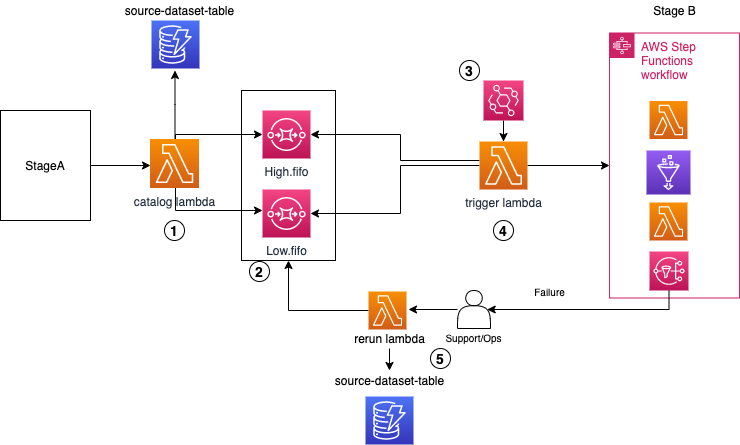
# Use Cases

In Data Lake projects, there are various different stages ranging from ingestion, foundations and consumption. During the Ingestion phases, a lot of data is ingested to the data lake on s3 from various sources. Each source can have multiple datasets that will be ingested concurrently onto the platform. During one of our customer implementations, we faced couple of issues with large number of simultaneous executions. 1) The glue job failed with max concurrency which was resolved by adding wait states and increasing the limits 2) Associate KMS key when using KMS key in the glue security configuration where we hit a hard limit on associatekmskey action on CloudWatch log groups. 3) Glue crawler api throttle on start crawler. Furthermore, it become a concern when sources with large number of datasets started taking control of compute resource for a long time and clogging the pipeline. The workload management solution aims to control the flow of step function executions based on priority of source-dataset which will unsure all high priority datasets irrespective of the source are processed.

## Architecture

Below diagram represents the high-level architecture of this solution.

**High Level Architecture**:



In this architecture we will consider a specific pattern for which the code and IaaC are developed

**StageA**

This is the compute resource which connects to a source database to ingest data to s3 bucket converted into parquet files. At the end the glue job writes an empty trigger file. The s3 notification trigger is configured on the suffix “trigger”

**Workload Management**

* 1. The catalog lambda uses the s3 event sent to lambda and creates a list of all datasets with all respective parquet files.
  2. It uses various attributes and creates the step function event required for stage B processing
  3. Based on source-dataset attributes it will check the DDB table and get the priority of that particular source-dataset and send the SFN event to the relevant SQS queue.
  4. There are currently 2 SQS based on high and low priority level.
  5. The EventBridge rule can be configured based on customer usecase, currently it is set to run every 5 minutes
  6. The trigger lambda is where actual workload management logic sits.
  7. It checks the number of running step function executions. If the upper limit defined by us is reached, it will exit out
  8. If there are execution slots left, it will pull the exact number of messages as the open execution slots out of the SQS queues based on priority. It can be from only high or only low or combination of both.
  9. Once we have all the messages ready, it will start the step function for each message.
  10. If there is a step function failure, a SNS message will be sent out to OPS/Support team to resolve the error.
  11. Once error is resolved, the message can be put back into the relevant SQS queue by executing the test event in the rerun lambda.
  12. This will put the message back to the SQS based on priority defined for that source-dataset in DDB.

**Stage B**

Stage B is a step function with a series of steps like lambda and glue. It can contain any customer specific pipeline logic. For demo purpose it will be a simple step function example which will either succeed or fail based on a random parameter passed as part of step function event.

**Extensibility**

The Architecture is extensible by adding more SQS queues and changing the trigger lambda logic by adding boto3 definitions for new SQS queues.

## 

## Pre-requisites & Demo execution details

We need to set up account profiles in ~/.aws/credentials

Run the below command on your local computer terminal to open the help message to see all the execution options

./deploy.sh -h

-h -- Opens up this help message

-p -- Name of the AWS profile to use for the Account

-r -- AWS Region to deploy to (e.g. eu-west-1)

-s -- S3 bucket to store artifact

We start by executing the below command. For demo purpose don’t provide -s and let deploy script create the bucket

./deploy.sh -p <aws-profile:default:default> -r <region:default:us-east-1>

This will deploy all the nested cloudformation stacks required for the workload management solutions and also fill the DynamoDB with required information for demo purpose

Once all the stacks are deployed successfully, check the DDB to confirm if there are 4 records in the table. To start the demo, execute the below command to see the attributes

./bootstrap.sh -h

-h -- Opens up this help message

-p -- Name of the AWS profile to use for the Account

-r -- AWS Region to deploy to (e.g. eu-west-1)

-s -- S3 bucket to store artifact

We start by demo with below command. For demo purpose don’t provide -s and let bootstrap script use the bucket

./bootstrap.sh -p <aws-profile:default:default> -r <region:default:us-east-1>

Once the demo is done, run the clean-up script to delete all data, s3 bucket, DDB and all relevant cloudformation stacks.

./cleanup.sh -h

-h -- Opens up this help message

-p -- Name of the AWS profile to use for the Account

-r -- AWS Region to deploy to (e.g. eu-west-1)

-s -- S3 bucket to store artifact

We start the clean-up with below command. For demo purpose don’t provide -s and let bootstrap script use the bucket

./cleanup.sh -p <aws-profile:default:default> -r <region:default:us-east-1>

## Workload Management Repo Structure

The workload management solution can be found in the path below.

**CODE:** <https://code.amazon.com/packages/GenericRedshiftUnloadScript/blobs/mainline/--/exports3toloc.sh>

**Repository Structure**

. .

├── .DS\_Store

├── deploy.sh #Deploys the infrastructure and fill items into DDB

├── lambda

│   ├── rerun #Lambda to put events back into SQS after failure

│   │   └── src

│   │   └── lambda\_function.py

│   ├── send-events-sqs #catalog lambda – gets all the datasets, create SFN event, send event to SQS

│   │   └── src

│   │   └── lambda\_function.py

│   ├── step1 #An example step inside SFN

│   │   └── src

│   │   └── lambda\_function.py

│   └── workload-management #trigger wlm lambda that manages the workload based on priority

│   └── src

│   └── lambda\_function.py

├── nested-stack

│   ├── template-ddb.yaml #CFN stack that contains definition for DDB

│   ├── template-kms.yaml #CFN stack that contains definition for KMS

│   └── template-workload-management.yaml #CFN stack that contains all definition for WLM resources

├── template.yaml #CFN stack that deploys nested stacks

└── utils #Supplimentaty helper scripts

├── bootstrap.sh #starts the demo executions

├── cleanup.sh #cleans up all Iaac and other resources

├── data #Example data for demo

│   └── LEGISLATORS

│   └── LEGISLATORS

│   └── 2021-03-21:00:00:00

│   ├── MEMBERSHIPS

│   │   └── part-00000-d4e08da7-d783-4b12-9339-c527402b7908-c000.snappy.parquet

│   ├── ORGANIZATIONS

│   │   └── part-00000-d4e08da7-d783-4b12-9339-c527402b7908-c000.snappy.parquet

│   ├── PERSONS

│   │   └── part-00000-d4e08da7-d783-4b12-9339-c527402b7908-c000.snappy.parquet

│   └── REGIONS

│   └── part-00000-d4e08da7-d783-4b12-9339-c527402b7908-c000.snappy.parquet

├── ddb\_items #source-dataset-priority items for DDB

│   └── workload\_management\_ddb.json

├── fill\_ddb.py #python file to put items into DDB table from json

└── trigger #Example trigger for demo

└── LEGISLATORS

└── LEGISLATORS

└── 2021-03-21:00:00:00

└── 46fcebc8-993e-4c3e-b0ed-1f5fddbe0c6c.trigger

# Conclusion

This workload management solution can be really helpful when the workload needs to be distributed across various sources based on priority levels, it will ensure that no one source takes control of all available resources and help the customer customize the solution based on their requirement enabling them to scale better and without wasting resource on throttling errors some of them would most likely be hard limits.