**AWS Take Home Assignment**

**Purpose**

The purpose of this document is to produce the Architecture considerations and design for the AWS take home assignment. The documents also reveal the details of the code base and access details for the AWS account.

**Assignment Task Details**

Lets Buy LLC provides SaaS services to retailers worldwide.

One of their leading solutions uses Amazon Redshift which is an MPP database to allow their customers to evaluate and understand the effectiveness of their supply chains. It is critical that the data loaded into Amazon Redshift is accurate and the data storage is optimized so as to minimize the cost of the MPP cluster.

The Goal for this assignment is to create two different processes for loading the sample data files provided at the S3 location - s3://dory-public/tpch/1000/supplier/ into a Redshift. The data loaded into Redshift must maintain the original grain from source and must be secured during transition and storage.

**Analysis of Raw Data**

The raw data that is available at s3://dory-public/tpch/1000/supplier/ was copied to a S3 location and a one time Glue Crawler was setup to fetch metadata information from the data. The information retrieved from the crawler can be seen below:

A screenshot of a cell phone

Description automatically generated

The metadata retrieved has been used further to create the Redshift tables.

**Redshift Cluster and Database**

The following redshift cluster has been launched in the separate VPC from AWS default:

*aws-assignment-awsassignmentredshiftcluster-d8z2i899ckcw.cnowfxl3k0d2.us-east-1.redshift.amazonaws.com*

The Cluster Configuration is – dc2.larger – 1 Leader and 4 Compute nodes. The database name is aws-assignment.

For the purpose of the take home assignment a schema awsassignment has been created and the following two tables have been created under it initially

create schema if not exists assignment;  
  
create table if not exists assignment.supplier\_stage(  
 s\_suppkey bigint encode MOSTLY32,  
 s\_name varchar(100) encode LZO,  
 s\_address varchar(100) encode LZO,  
 s\_nationkey integer encode MOSTLY8,  
 s\_phone varchar(40) encode LZO,  
 s\_acctbal decimal(12,2) encode delta32k,  
 s\_comment varchar(max) encode LZO  
 )  
 distkey(s\_suppkey) compound sortkey(s\_suppkey) ;  
  
create table if not exists assignment.supplier\_data(  
 s\_suppkey bigint NOT NULL encode MOSTLY32,  
 s\_name varchar(100) encode LZO,  
 s\_address varchar(100) encode LZO,  
 s\_nationkey integer encode MOSTLY8,  
 s\_phone varchar(40) encode LZO,  
 s\_acctbal decimal(12,2) encode delta32k,  
 s\_comment varchar(max) encode LZO,  
 load\_ts timestamp NOT NULL  
 )  
 distkey(s\_suppkey) compound sortkey(load\_ts) ;

After the initial load and post running the analyze compression, the table compression has been changes to :

TBD

The document will mention in the Data Loading Processes section the reason behind creating two tables.

**GitHub Link**

The code repository used for this assignment can be found at - <https://github.com/debojyotimukherjee/aws-assignment.git>

**Data Loading Process -1 (Serverless architecture)**

**Infrastructure setup**

The UNIX script - aws\_assignment\_deploy.sh, has been created and used to setup the following infrastructure:

1. VPC
2. Subnet and Internet Gateway
3. S3 Bucket
4. Redshift Cluster
5. Glue ETL Job
6. Glue Python Job
7. Glue Trigger
8. Secret Key Manager to storing Redshift Password

The script should be executed with one argument - <environment name>. This is required as all the resources will include this name. like for the current assignment the environment name – aws-assignment has been used - ***./aws\_assignment\_deploy.sh aws-assignment***

The CloudFormation template can be found at the location:

***cloudformation/ aws\_assignment\_create\_stack.yaml***

**High Level Diagram for the design:**

![A close up of a sign

Description automatically generated]()

Fig1: ETL Approach 1 HLD

**Design Principle**

1. The data will be loaded into Redshift as a nightly batch process.
2. Source Data Assumption - For the purpose of this assignment, it has been assumed that the data file will arrive at a s3 source bucket into the following folder structure: s3://aws-assignment-data-source/<data\_source\_name – eg: supplier>/<todays date – eg: 20200604>
3. A glue trigger has been created to initiate a Glue Spark job – glue\_etl/awsassignment\_glue\_prepare\_file.py to read the data from s3 source location and load into another s3 output location. In this process the spark job reads a parameter to determine the number of files it needs to create. For the assignment the default value has been set to 4, matching the number of compute nodes in Redshift.
   1. The job been created to accept other types of data source and data file suing parameters. Snippet from the code:

def get\_read\_file\_df(spark, file\_path, file\_type, file\_header="true", file\_delimiter=None, null\_value=''):  
 try:  
 if file\_type == "delimited":  
 if file\_delimiter:  
 return spark.read.option("nullValue", null\_value).load(file\_path + "/\*", format="csv",  
 sep=file\_delimiter, inferSchema="true",  
 header=file\_header)  
 else:  
 sys.exit("Missing delimiter for delimited file : ")  
  
 elif file\_type == "parquet":  
 return spark.read.parquet(file\_path + "/\*")  
  
 elif file\_type == "orc":  
 return spark.read.orc(file\_path + "/\*")  
  
 else:  
 sys.exit("Incorrect file type defined")  
  
 except Exception as e:  
 print(f'Unhandled exception: {str(e)}')  
 sys.exit()

* 1. The following arguments in the Job determine which while to read from the s3 location:

try:  
 default\_args = getResolvedOptions(sys.argv, ['AWS\_REGION', 'DATA\_SOURCE\_BUCKET\_NAME',  
 'DATA\_OUTPUT\_BUCKET\_NAME', 'ENVIRONMENT',  
 'DATA\_SOURCE\_NAME', 'SOURCE\_FILE\_TYPE',  
 'FILE\_HEADER', 'FILE\_DELIMITER',  
 'FILE\_NULL\_VALUE', 'OUTPUT\_FILE\_PARTITIONS',  
 'OUTPUT\_FILE\_DELIMITER'])  
  
 aws\_region = default\_args['AWS\_REGION']  
 environment = default\_args['ENVIRONMENT']  
 target\_bucket = f'{environment}-data-output'  
 source\_bucket = f'{environment}-data-source'  
 data\_source\_name = default\_args['DATA\_SOURCE\_NAME']  
 source\_file\_type = default\_args['SOURCE\_FILE\_TYPE']  
 file\_header = default\_args['FILE\_HEADER']  
 file\_delimiter = default\_args['FILE\_DELIMITER']  
 null\_value = default\_args['FILE\_NULL\_VALUE']  
 output\_file\_partitions = int(default\_args['OUTPUT\_FILE\_PARTITIONS'])  
 output\_file\_delimiter = default\_args['OUTPUT\_FILE\_DELIMITER']  
  
 source\_folder\_date = datetime.now().strftime("%Y%m%d")

* 1. The Output file created will always be a gzip compressed delimited file. The delimiter can be passes as a parameter to the job.

1. Post completion of the spark job a python shell Glue job is triggered to load the data into Redshift. The python job basically triggers sql’s from the the SQL script sql/supplier\_load\_data.sql using the postgres – pgdb module.
   1. The Job fetches the password for etl\_user from the AWS Secret Manager:

def get\_secret(rs\_etl\_password\_secret, aws\_region):  
 secret\_name = rs\_etl\_password\_secret  
 region\_name = aws\_region  
  
 # Create a Secrets Manager client  
 session = boto3.session.Session()  
 client = session.client(  
 service\_name='secretsmanager',  
 region\_name=region\_name  
 )  
  
 get\_secret\_value\_response = client.get\_secret\_value(  
 SecretId=secret\_name  
 )  
  
 return json.loads(get\_secret\_value\_response['SecretString'])['password']

* 1. Gets the Query from the s3 location. Replaces some of the keywords within the query to maintain the generic nature of the script and executes it. Post successful completion of the query the transaction is committed.

def get\_sql\_body(environment, data\_source\_name, rs\_schema, aws\_account\_id):  
 s3 = boto3.resource('s3')  
 sql\_file\_obj = s3.Object(f'{environment}-functions', f'sql/{data\_source\_name}\_load\_data.sql')  
 sql\_body = sql\_file\_obj.get()['Body'].read()  
  
 return sql\_body.decode('utf-8').replace("schema\_name", rs\_schema). \  
 replace("aws\_account\_id", aws\_account\_id).replace("environment", environment). \  
 replace("data\_source\_name", data\_source\_name)  
  
  
def execute\_query(sql\_query\_body):  
 con = connect(host=rs\_host + ':' + rs\_port, database=environment, user=rs\_etl\_user, password=rs\_etl\_password)  
 cursor = con.cursor()  
 cursor.execute(sql\_query\_body)  
 con.commit()  
 cursor.close()  
 con.close()

* 1. The SQL query used in the script uses s3 to Redshift copy command to load the data into staging table after truncating it and then from the staging table inserts the data to the main table.

truncate table schema\_name.supplier\_stage;  
  
copy schema\_name.supplier\_stage  
from 's3://environment-data-output/data\_source\_name/output/'  
iam\_role 'arn:aws:iam::aws\_account\_id:role/environment-redshift-s3-access-role'  
delimiter '|' gzip;  
  
  
insert into schema\_name.supplier\_data  
(  
 select  
 s\_suppkey ,  
 s\_name ,  
 s\_address ,  
 s\_nationkey ,  
 s\_phone ,  
 s\_acctbal ,  
 s\_comment,  
 current\_timestamp as load\_ts  
 from schema\_name.supplier\_stage  
);

**Architectural Benefits**

1. The entire ETL architecture is Server less. AWS will only bill for the number of times the Glue job is executed.
2. The Glue Spark job can be scaled by adding more DPU’s or changing the worker type.
3. Using Redshift Copy command from s3 takes full advantage of fast data loading to the staging table.

**Data Loading Process -2 (Persistent EMR)**