

AWS Take Home Assignment

Version:1.0

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

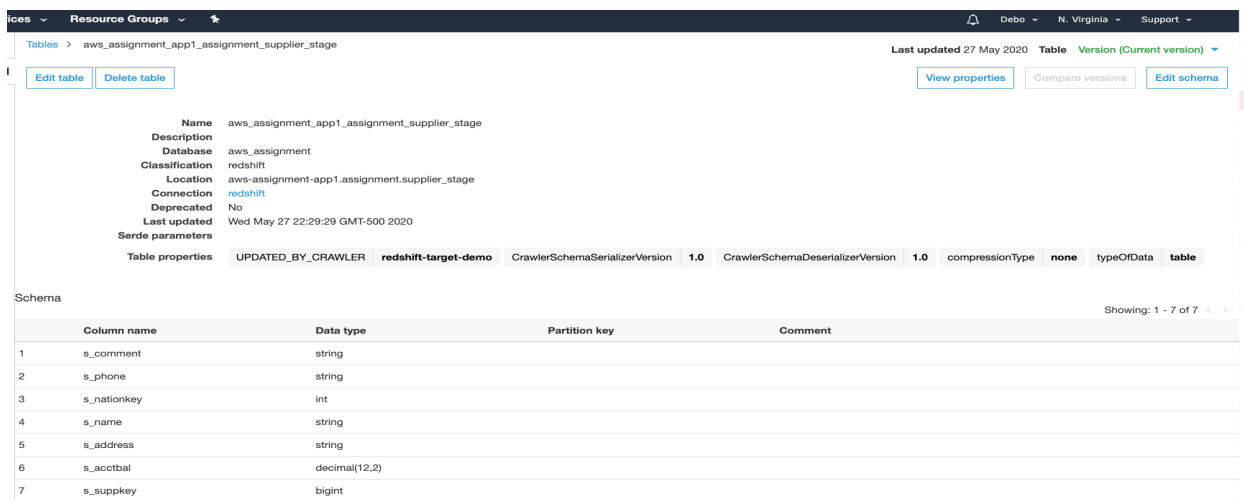
Lets Buy LLC provides SaaS services to retailers worldwide.

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:



The screenshot displays the AWS Glue console interface. At the top, there's a navigation bar with 'Tables' selected, showing the path 'aws_assignment_app1_assignment_supplier_stage'. On the right, it indicates 'Last updated 27 May 2020' and 'Table Version (Current version)'. Below this, there are buttons for 'Edit table', 'Delete table', 'View properties', 'Compare versions', and 'Edit schema'. The main content area shows the table's metadata:

- Name:** aws_assignment_app1_assignment_supplier_stage
- Description:**
- Database:** aws_assignment
- Classification:** redshift
- Location:** aws-assignment-app1.assignment.supplier_stage
- Connection:** redshift
- Deprecated:** No
- Last updated:** Wed May 27 22:29:29 GMT-500 2020
- Serde parameters:**

Below the metadata, there's a section for 'Table properties' with a list of properties:

Property Name	Value
UPDATED_BY_CRAWLER	redshift-target-demo
CrawlerSchemaSerializerVersion	1.0
CrawlerSchemaDeserializerVersion	1.0
compressionType	none
typeOfData	table

At the bottom, the 'Schema' section shows a table with 7 columns:

Column name	Data type	Partition key	Comment
1 s_comment	string		
2 s_phone	string		
3 s_nationkey	int		
4 s_name	string		
5 s_address	string		
6 s_acctbal	decimal(12,2)		
7 s_suppkey	bigint		

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 were created in it initially within schema – assignment.

```
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 changed to:

```
create table if not exists assignment.supplier_stage(  
    s_suppkey bigint encode MOSTLY32,  
    s_name varchar(100) encode zstd,  
    s_address varchar(100) encode zstd,  
    s_nationkey integer encode az64,  
    s_phone varchar(40) encode zstd,  
    s_acctbal decimal(12,2) encode az64,  
    s_comment varchar(max) encode zstd  
)  
    distkey(s_suppkey);  
  
create table if not exists assignment.supplier_data(  
    s_suppkey bigint NOT NULL encode MOSTLY32,  
    s_name varchar(100) encode zstd,  
    s_address varchar(100) encode zstd,  
    s_nationkey integer encode az64,  
    s_phone varchar(40) encode zstd,  
    s_acctbal decimal(12,2) encode az64,  
    s_comment varchar(max) encode zstd,  
    load_ts timestamp NOT NULL encode AZ64  
)  
    distkey(s_suppkey) compound sortkey(load_ts) ;
```

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:

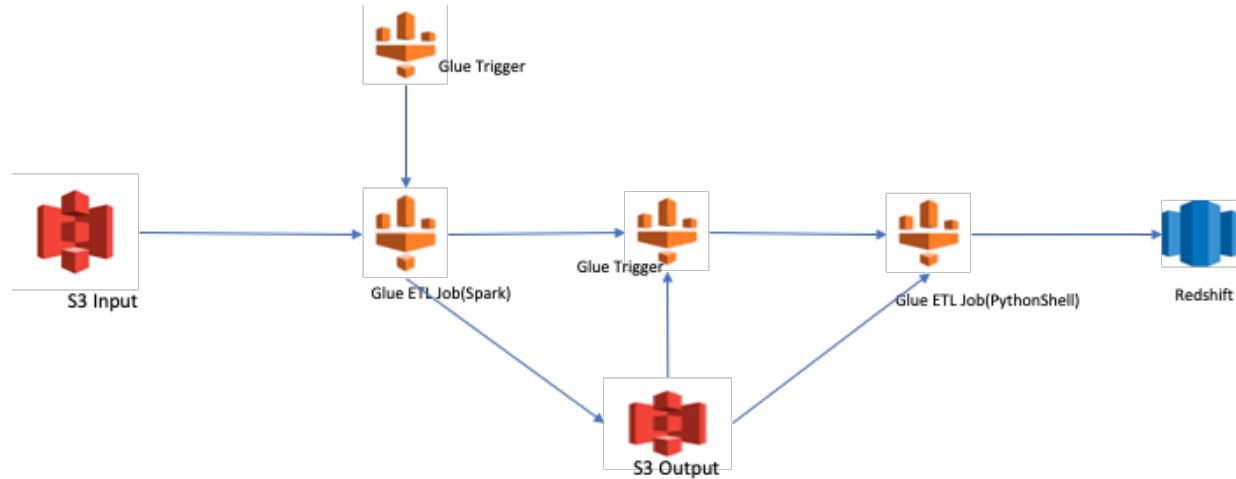


Fig1: ETL Approach 1 HLD

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>/<today's 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.
 - a. 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()

```

- b. 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")

```

- c. The Output file created will always be a gzip compressed delimited file. The delimiter can be passes as a parameter to the job.
4. 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.

- a. 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']
```

- b. 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()
```

- c. 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_supkey ,
    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.
4. Taking advantage of Redshift Copy command to move the data to the Staging table.

Data Loading Process -2 (Persistent EMR)

Infrastructure setup

On top of the stack that was created in approach 1 using the UNIX script - `aws_assignment_deploy.sh` a EMR Cluster has been spun up with the following bootstrap script – `emr_scripts/bootstrap/emr_bootstrap_setup.sh`

The script post EMR's deployment will execute the following commands:

```

#!/bin/bash

#####Script for bootstrapping the EMR
Cluster#####

if [ $# -ne 1 ]; then
echo "usage: `basename $0` <environment>" > /dev/stderr

```

```

exit 1
fi

environment=$1

#Create Log Directory
mkdir /home/hadoop/logs

#Install postgres client
sudo yum install -y postgresql

#Install git
sudo yum install -y git

#Download Code Repo
git clone https://github.com/debojyotimukherjee/aws-assignment.git

```

The name of the environment is configurable during the EMR create step. The script will install the postgres client to connect to Redshift and install git to download the code base.

High Level Diagram for the design:

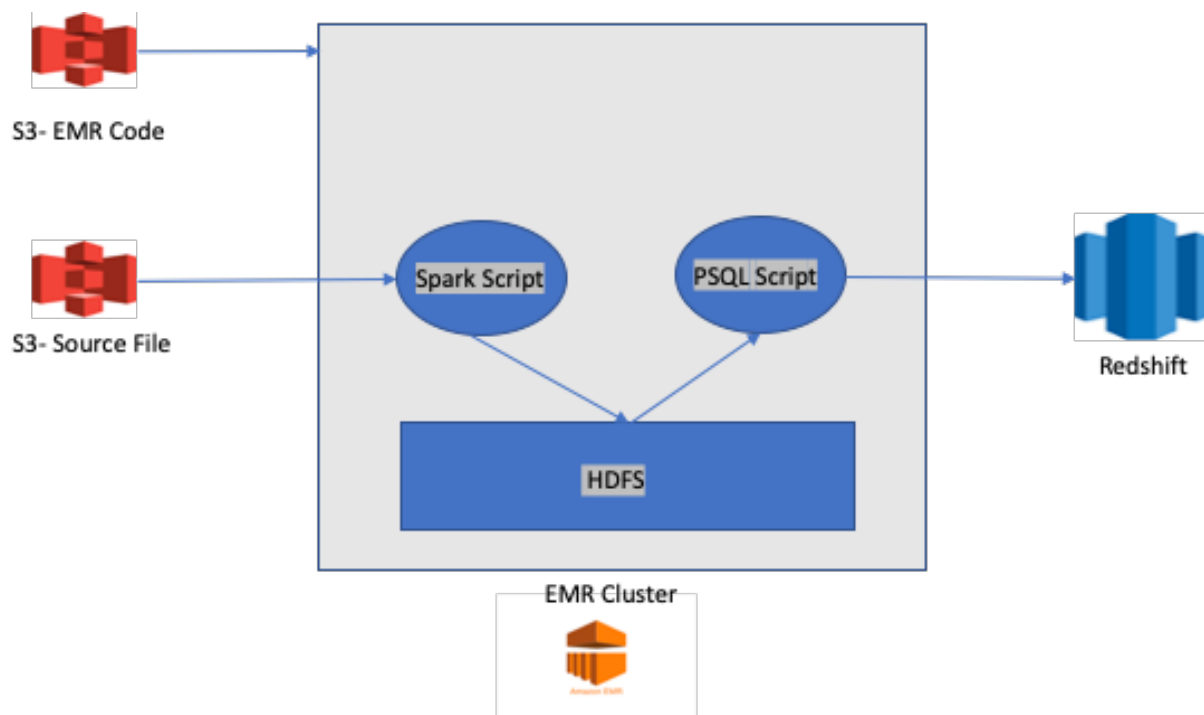


Fig2: ETL Approach 2 HLD

Design Principle

1. The data will be loaded into Redshift as a nightly batch process.
2. The entire process has been wrapped in a UNIX shell script - `emr_scripts/shell/redshift_job_trigger.sh`
3. Similar to approach-1, 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>/<today's date – eg: 20200604>`. Unlike approach 1 `configparser` package from python has been used to get the required job parameters:

```
[supplier_config]
source_file_type = parquet
file_header = None
file_delimiter = None
null_value = None
output_file_partitions = 4
output_file_delimiter = |
```

4. Once the Spark Job finishes the next step will be to trigger the psql script to load the data into redshift:

```
BEGIN transaction;

truncate table :schema_name.supplier_stage;

copy :schema_name.supplier_stage
from :emr_file_path credentials
iam_role :iam_role
delimiter '|' gzip;

insert into :schema_name.supplier_data
(
select
s_supkey ,
s_name ,
s_address ,
s_nationkey ,
s_phone ,
s_acctbal ,
s_comment,
current_timestamp as load_ts
from :schema_name.supplier_stage
);

END transaction;
```

Architectural Benefits

1. With this architecture it is slightly easier to migrate a ETL framework which might be currently running on On-Prem Hadoop.
2. Taking advantage of Redshift Copy command to move the data to the Staging table.
3. The data can be easily integrated with multiple other data sources which might only exist on Hadoop.
4. The Output of Spark Job is stored within date folder. On top this data HIVE partitioned tables can be created, which in turn can be used for analytics with in the Hadoop Ecosystem.

Future Improvement Scope

Highlighting a few improvement areas which can be achieved in future:

1. Automate a process to run the Glue Crawler more often than once. This will enable to determine the change in the data source.
2. Integrate a Data Quality Check component for the source data. A data quality check component has been added to the code base but would like to add it with the spark jobs. This will make sure that Data Quality issues are caught early in the process. The location for the module is: data_quality_check/ checkDataQuality.py
3. Based on business requirement there might be a need to change the data model. Right now, the data is not normalized and that might not be required with Redshift compression in place. But in future if there is a need a dimension table and fact table can be created with the following DDL:

```
/* Following is the DDL for Supplier Table */  
  
create schema if not exists assignment;  
  
create table if not exists assignment.supplier_stage(  
    s_suppkey bigint ,  
    s_name varchar(100) ,  
    s_address varchar(100) ,  
    s_nationkey integer ,  
    s_phone varchar(40) ,  
    s_acctbal decimal(12,2) ,
```

```

    s_comment varchar(max)
)
distkey(s_suppkey) compound sortkey(s_suppkey) ;

create table if not exists assignment.supplier_dim(
    s_suppkey bigint NOT NULL ,
    s_name varchar(100) ,
    s_address varchar(100) ,
    s_nationkey integer,
    s_phone varchar(40) ,
    eff_start_date date,
    eff_end_date date,
    active_flag char(1)
)
distkey(s_suppkey) compound sortkey(active_flag) ;

create table if not exists assignment.supplier_fact(
    s_suppkey bigint ,
    s_acctbal decimal(12,2) ,
    s_comment varchar(max) ,
    load_ts
)
distkey(s_suppkey) compound sortkey(load_ts) ;

```

4. For the Persistent EMR approach use a scheduling tool like Airflow.
5. If there is a need to ingest the data near real time into Redshift then S3 events to trigger Lambda can be setup, which can then trigger a Glue process or an ECS instance to load the data.

Access Details

AWS Console Access:

Sign-in URL: <https://048532184061.signin.aws.amazon.com/console>

Username: aws-assignment

Password: Assignment123

Redshift Access:

Database Name: aws-assignment

Username: aws_assignment

Password: Assignment123