Project 3 - Data Warehouse: Creating Redshift Cluster using the AWS python SDK

Infrastructure-as-code

In [1]:
```
import pandas as pd
import boto3
import json
```

STEP 0: Make sure you have an AWS secret and access key

- Create a new IAM user in your AWS account
- Give it `AdministratorAccess` , From `Attach existing policies directly` Tab
- Take note of the access key and secret
- Edit the file `dwh.cfg` in the same folder as this notebook and fill

  [AWS]
  KEY= YOUR_AWS_KEY
  SECRET= YOUR_AWS_SECRET

Load DWH Params from a file

In [2]:
```
import configparser
config = configparser.ConfigParser()
config.read_file(open('dwh.cfg'))

KEY                    = config.get('AWS','KEY')
SECRET                 = config.get('AWS','SECRET')

DWH_CLUSTER_TYPE       = config.get("DWH","DWH_CLUSTER_TYPE")
DWH_NUM_NODES          = config.get("DWH","DWH_NUM_NODES")
DWH_NODE_TYPE          = config.get("DWH","DWH_NODE_TYPE")

DWH_CLUSTER_IDENTIFIER = config.get("DWH","DWH_CLUSTER_IDENTIFIER")
DWH_DB                 = config.get("DWH","DWH_DB")
DWH_DB_USER            = config.get("DWH","DWH_DB_USER")
DWH_DB_PASSWORD        = config.get("DWH","DWH_DB_PASSWORD")
DWH_PORT               = config.get("DWH","DWH_PORT")

DWH_IAM_ROLE_NAME      = config.get("DWH", "DWH_IAM_ROLE_NAME")

(DWH_DB_USER, DWH_DB_PASSWORD, DWH_DB)

pd.DataFrame({"Param":
                  ["DWH_CLUSTER_TYPE", "DWH_NUM_NODES", "DWH_NODE_TYPE", "DWH_CLUSTER_IDENTIFIER", "DWH_DB",
"DWH_DB_USER", "DWH_DB_PASSWORD", "DWH_PORT", "DWH_IAM_ROLE_NAME"],
              "Value":
                  [DWH_CLUSTER_TYPE, DWH_NUM_NODES, DWH_NODE_TYPE, DWH_CLUSTER_IDENTIFIER, DWH_DB, DWH_DB_USE
R, DWH_DB_PASSWORD, DWH_PORT, DWH_IAM_ROLE_NAME]
             })
```

Out[2]:

	Param	Value
0	DWH_CLUSTER_TYPE	multi-node
1	DWH_NUM_NODES	4
2	DWH_NODE_TYPE	dc2.large
3	DWH_CLUSTER_IDENTIFIER	dwhCluster
4	DWH_DB	dwh
5	DWH_DB_USER	dwhuser
6	DWH_DB_PASSWORD	PasswordXXX
7	DWH_PORT	5439
8	DWH_IAM_ROLE_NAME	myRedshiftRole

Create clients for IAM, EC2, S3 and Redshift

In [3]:
```
import boto3

ec2 = boto3.resource('ec2',
                       region_name="us-west-2",
                       aws_access_key_id=KEY,
                       aws_secret_access_key=SECRET
                    )

s3 = boto3.resource('s3',
                       region_name="us-west-2",
                       aws_access_key_id=KEY,
                       aws_secret_access_key=SECRET
                    )

iam = boto3.client('iam',aws_access_key_id=KEY,
                     aws_secret_access_key=SECRET,
                     region_name='us-west-2'
                  )

redshift = boto3.client('redshift',
                       region_name="us-west-2",
                       aws_access_key_id=KEY,
                       aws_secret_access_key=SECRET
                       )
```

Check out the sample data sources on S3

In [4]:
```
sampleDbBucket =  s3.Bucket("awssampledbuswest2")
for obj in sampleDbBucket.objects.filter(Prefix="ssbgz"):
    print(obj)
# for obj in sampleDbBucket.objects.all():
#     print(obj)
```

```
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/customer0002_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/dwdate.tbl.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0000_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0001_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0002_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0003_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0004_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0005_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0006_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/lineorder0007_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/part0000_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/part0001_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/part0002_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/part0003_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/supplier.tbl_0000_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/supplier0001_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/supplier0002_part_00.gz')
s3.ObjectSummary(bucket_name='awssampledbuswest2', key='ssbgz/supplier0003_part_00.gz')
```

STEP 1: IAM ROLE

- Create an IAM Role that makes Redshift able to access S3 bucket (ReadOnly)

In [5]:
```
from botocore.exceptions import ClientError

#1.1 Create the role,
try:
    print("1.1 Creating a new IAM Role")
    dwhRole = iam.create_role(
        Path='/',
        RoleName=DWH_IAM_ROLE_NAME,
        Description = "Allows Redshift clusters to call AWS services on your behalf.",
        AssumeRolePolicyDocument=json.dumps({
            "Version" : "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                       "Service": "redshift.amazonaws.com"
                     },
                    "Action": "sts:AssumeRole"
                }
              ]
        }),
    )
except Exception as e:
    print(e)


print("1.2 Attaching Policy")

iam.attach_role_policy(RoleName=DWH_IAM_ROLE_NAME,
                       PolicyArn="arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
                      )['ResponseMetadata']['HTTPStatusCode']

print("1.3 Get the IAM role ARN")
roleArn = iam.get_role(RoleName=DWH_IAM_ROLE_NAME)['Role']['Arn']

print(roleArn)
```

```
1.1 Creating a new IAM Role
1.2 Attaching Policy
1.3 Get the IAM role ARN
arn:aws:iam::XXXXXXXXXXXX:role/myRedshiftRole
```

STEP 2: Redshift Cluster

- Create a RedShift Cluster
- For complete arguments to `create_cluster` , see docs (https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/redshift.html#Redshift.Client.create_cluster)

In [6]:
```
try:
    response = redshift.create_cluster(
        #HW
        ClusterType=DWH_CLUSTER_TYPE,
        NodeType=DWH_NODE_TYPE,
        NumberOfNodes=int(DWH_NUM_NODES),

        #Identifiers & Credentials
        DBName=DWH_DB,
        ClusterIdentifier=DWH_CLUSTER_IDENTIFIER,
        MasterUsername=DWH_DB_USER,
        MasterUserPassword=DWH_DB_PASSWORD,

        #Roles (for s3 access)
        IamRoles=[roleArn]
    )
except Exception as e:
    print(e)
```

2.1 *Describe* the cluster to see its status

- run this block several times until the cluster status becomes `Available`

In [8]:
```
def prettyRedshiftProps(props):
    pd.set_option('display.max_colwidth', -1)
    keysToShow = ["ClusterIdentifier", "NodeType", "ClusterStatus", "MasterUsername", "DBName", "Endpoint",
"NumberOfNodes", 'VpcId']
    x = [(k, v) for k,v in props.items() if k in keysToShow]
    return pd.DataFrame(data=x, columns=["Key", "Value"])

myClusterProps = redshift.describe_clusters(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)['Clusters'][0]
prettyRedshiftProps(myClusterProps)
```

Out[8]:

	Key	Value
0	ClusterIdentifier	dwhcluster
1	NodeType	dc2.large
2	ClusterStatus	available
3	MasterUsername	dwhuser
4	DBName	dwh
5	Endpoint	{'Address': 'dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com', 'Port': 5439}
6	VpcId	vpc-fXXXXXXXX
7	NumberOfNodes	4

2.2 Take note of the cluster endpoint and role ARN

DO NOT RUN THIS unless the cluster status becomes "Available"

In [9]:
```
DWH_ENDPOINT = myClusterProps['Endpoint']['Address']
DWH_ROLE_ARN = myClusterProps['IamRoles'][0]['IamRoleArn']
print("DWH_ENDPOINT :: ", DWH_ENDPOINT)
print("DWH_ROLE_ARN :: ", roleArn)
```

```
DWH_ENDPOINT ::  dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com
DWH_ROLE_ARN ::  arn:aws:iam::XXXXXXXXXXXX:role/myRedshiftRole
```

STEP 3: Open an incoming TCP port to access the cluster endpoint

In [10]:
```
try:
    vpc = ec2.Vpc(id=myClusterProps['VpcId'])
    defaultSg = list(vpc.security_groups.all())[0]
    print(defaultSg)
    defaultSg.authorize_ingress(
        GroupName='default',
        CidrIp='0.0.0.0/0',
        IpProtocol='TCP',
        FromPort=int(DWH_PORT),
        ToPort=int(DWH_PORT)
    )
except Exception as e:
    print(e)
```

```
ec2.SecurityGroup(id='sg-XXXXXXXXXXXXXXXX')
An error occurred (InvalidPermission.Duplicate) when calling the AuthorizeSecurityGroupIngress operation: the
specified rule "peer: 0.0.0.0/0, TCP, from port: 5439, to port: 5439, ALLOW" already exists
```

STEP 4: Make sure you can connect to the cluster

In [11]:
```
%load_ext sql
```

In [12]:
```
conn_string="postgresql://{}:{}@{}:{}/{}".format(DWH_DB_USER, DWH_DB_PASSWORD, DWH_ENDPOINT, DWH_PORT,DWH_DB)
print(conn_string)
%sql $conn_string
```

```
postgresql://dwhuser:PasswordXXX@dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com:5439/dwh
```

Out[12]:
```
'Connected: dwhuser@dwh'
```

In [13]:
```
print ('DWH_DB_USER : ' + DWH_DB_USER)
print ('DWH_DB_PASSWORD : ' + DWH_DB_PASSWORD)
print ('DWH_ENDPOINT : ' + DWH_ENDPOINT)
print ('DWH_PORT : ' + DWH_PORT)
print ('DWH_DB : ' + DWH_DB)
```

```
DWH_DB_USER : dwhuser
DWH_DB_PASSWORD : PasswordXXX
DWH_ENDPOINT : dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com
DWH_PORT : 5439
DWH_DB : dwh
```

STEP 5 : Run Python script create_tables.py to

Create Staging tables in S3 and fact and dimension tables in redshift

In [14]:
```
%run -i create_tables.py
```

```
db_host :  dwhcluster.XXXXXXXXXXX.us-west-2.redshift.amazonaws.com  db_name :  dwh db_username :  dwhuser db
_password :  PasswordXXX db-port :  5439
Create_Tables :Connected to Database
Drop info by DROP TABLE IF EXISTS staging_events
Drop info by DROP TABLE IF EXISTS staging_songs
Drop info by DROP TABLE IF EXISTS songplays
Drop info by DROP TABLE IF EXISTS users
Drop info by DROP TABLE IF EXISTS songs
Drop info by DROP TABLE IF EXISTS artists
Drop info by DROP TABLE IF EXISTS time
Created :
CREATE TABLE IF NOT EXISTS staging_events
(
        artist            VARCHAR       NULL,
        auth              VARCHAR       NULL,
        firstName         VARCHAR       NULL,
        gender            VARCHAR       NULL,
        itemInSession     INTEGER       NULL,
        lastName          VARCHAR       NULL,
        length            FLOAT         NULL,
        level             VARCHAR       NULL,
        location          VARCHAR       NULL,
        method            VARCHAR       NULL,
        page              VARCHAR       NULL,
        registration      FLOAT         NULL,
        sessionId         INTEGER       NOT NULL,
        song              VARCHAR       NULL,
        status            INTEGER       NULL,
        ts                BIGINT        NOT NULL,
        userAgent         VARCHAR       NULL,
        userId            INTEGER       NULL
)

Created :
CREATE TABLE IF NOT EXISTS staging_songs
(
       song_id            VARCHAR         NOT NULL,
       artist_id          VARCHAR         NOT NULL,
       artist_latitude    FLOAT           NULL,
       artist_longitude   FLOAT           NULL,
       artist_location    VARCHAR         NULL,
       artist_name        VARCHAR         NULL,
       duration           FLOAT           NULL,
```

```
	num_songs		INTEGER		NULL,
	title			VARCHAR		NULL,
	year			INTEGER		NULL
)

Created :
CREATE TABLE IF NOT EXISTS songplays
(
	 songplay_id	INTEGER IDENTITY(0,1)	NOT NULL SORTKEY,
	 start_time	TIMESTAMP		NOT NULL,
	 user_id	INTEGER			NOT NULL,
	 level		VARCHAR			NOT NULL,
	 song_id	VARCHAR			NOT NULL,
	 artist_id	VARCHAR			NOT NULL DISTKEY,
	 session_id	INTEGER			NOT NULL,
	 location	VARCHAR			NULL,
	 user_agent	VARCHAR			NULL
)

Created :
CREATE TABLE IF NOT EXISTS users
(
	user_id		INTEGER		NOT NULL SORTKEY,
	first_name	VARCHAR		NULL,
	last_name	VARCHAR		NULL,
	gender		VARCHAR		NULL,
	level		VARCHAR		NULL
)
diststyle ALL;

Created :
CREATE TABLE IF NOT EXISTS songs
(
	song_id		VARCHAR		NOT NULL SORTKEY,
	title		VARCHAR		NOT NULL,
	artist_id	VARCHAR		NOT NULL,
	year		INTEGER		NOT NULL,
	duration	FLOAT		NOT NULL
)

Created :
CREATE TABLE IF NOT EXISTS artists
(
```

```
        artist_id     VARCHAR        NOT NULL SORTKEY,
        name          VARCHAR        NULL,
        location      VARCHAR        NULL,
        latitude      FLOAT          NULL,
        longitude     FLOAT          NULL
)
diststyle ALL;

Created :
CREATE TABLE IF NOT EXISTS time
(
        start_time TIMESTAMP    NOT NULL SORTKEY,
        hour       INTEGER      NULL,
        day        INTEGER      NULL,
        week       INTEGER      NULL,
        month      INTEGER      NULL,
        year       INTEGER      NULL,
        weekday    INTEGER      NULL
)

Create_Tables : Tables Created
Create_Tables.py - Connection Closed
```

STEP 6 : Run Python script etl.py

In [15]:
```
%run -i etl.py
```

```
Connect To Redshift ...
Load Staging Tables ...
Copy :
COPY staging_events
FROM 's3://udacity-dend/log_data'
CREDENTIALS 'aws_iam_role=arn:aws:iam::XXXXXXXXXXX:role/myRedshiftRole'
COMPUPDATE OFF region 'us-west-2'
FORMAT AS json 's3://udacity-dend/log_json_path.json'

Copy :
COPY staging_songs
FROM 's3://udacity-dend/song_data'
CREDENTIALS 'aws_iam_role=arn:aws:iam::XXXXXXXXXXX:role/myRedshiftRole'
COMPUPDATE OFF region 'us-west-2'
FORMAT AS json 'auto'

Transform Staging Tables
Transform data by
INSERT INTO songplays (start_time, user_id, level, song_id, artist_id, session_id, location, user_agent)
SELECT DISTINCT TIMESTAMP 'epoch' + ste.ts/1000 * INTERVAL '1 second' AS start_time,
       ste.userId     AS user_id,
       ste.level      AS level,
       sts.song_id    AS song_id,
       sts.artist_id  AS artist_id,
       ste.sessionId  AS session_id,
       ste.location   AS location,
       ste.userAgent  AS user_agent
FROM staging_events ste
JOIN staging_songs sts
ON (ste.artist = sts.artist_name)
AND ste.page = 'NextSong';

Transform data by
INSERT INTO users (user_id, first_name, last_name, gender, level)
SELECT DISTINCT ste.userId  AS user_id,
                ste.firstName  AS first_name,
                ste.lastName   AS last_name,
                ste.gender     AS gender,
                ste.level      AS level
FROM staging_events ste
WHERE ste.page = 'NextSong';

Transform data by
```

```
INSERT INTO songs(song_id, title, artist_id, year, duration)
SELECT DISTINCT sts.song_id    AS song_id,
                sts.title      AS title,
                sts.artist_id  AS artist_id,
                sts.year       AS year,
                sts.duration   AS duration
FROM staging_songs sts;

Transform data by
INSERT INTO artists(artist_id, name, location, latitude, longitude)
SELECT DISTINCT sts.artist_id         AS artist_id,
                sts.artist_name       AS name,
                sts.artist_location   AS location,
                sts.artist_latitude   AS latitude,
                sts.artist_longitude  AS longitude
FROM staging_songs sts;

Transform data by
INSERT INTO time (start_time, hour, day, week, month, year, weekday)
SELECT DISTINCT TIMESTAMP 'epoch' + ste.ts/1000 * INTERVAL '1 second' AS start_time,
                EXTRACT (hour    FROM start_time)   AS hour,
                EXTRACT (day     FROM start_time)   AS day,
                EXTRACT (week    FROM start_time)   AS week,
                EXTRACT (month   FROM start_time)   AS month,
                EXTRACT (year    FROM start_time)   AS year,
                EXTRACT (weekday FROM start_time) AS weekday
FROM staging_events ste
WHERE ste.page = 'NextSong';

Ending the ETL process
```

STEP 7: Clean up your resources

DO NOT RUN THIS UNLESS YOU ARE SURE
We will be using these resources in the next exercises

In [16]:
```
#### CAREFUL!!
#-- Uncomment & run to delete the created resources
redshift.delete_cluster( ClusterIdentifier=DWH_CLUSTER_IDENTIFIER,  SkipFinalClusterSnapshot=True)
#### CAREFUL!!
```

Out[16]:

```
{'Cluster': {'ClusterIdentifier': 'dwhcluster',
  'NodeType': 'dc2.large',
  'ClusterStatus': 'deleting',
  'MasterUsername': 'dwhuser',
  'DBName': 'dwh',
  'Endpoint': {'Address': 'dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com',
   'Port': 5439},
  'ClusterCreateTime': datetime.datetime(2021, 2, 14, 7, 16, 30, 997000, tzinfo=tzlocal()),
  'AutomatedSnapshotRetentionPeriod': 1,
  'ClusterSecurityGroups': [],
  'VpcSecurityGroups': [{'VpcSecurityGroupId': 'sg-XXXXXXXX',
    'Status': 'active'}],
  'ClusterParameterGroups': [{'ParameterGroupName': 'default.redshift-1.0',
    'ParameterApplyStatus': 'in-sync'}],
  'ClusterSubnetGroupName': 'default',
  'VpcId': 'vpc-fXXXXXXXX',
  'AvailabilityZone': 'us-west-2c',
  'PreferredMaintenanceWindow': 'mon:13:00-mon:13:30',
  'PendingModifiedValues': {},
  'ClusterVersion': '1.0',
  'AllowVersionUpgrade': True,
  'NumberOfNodes': 4,
  'PubliclyAccessible': True,
  'Encrypted': False,
  'Tags': [],
  'EnhancedVpcRouting': False,
  'IamRoles': [{'IamRoleArn': 'arn:aws:iam::XXXXXXXXXXXX:role/myRedshiftRole',
    'ApplyStatus': 'in-sync'}],
  'MaintenanceTrackName': 'current'},
 'ResponseMetadata': {'RequestId': '6b1d11d2-d807-4744-88d1-b72010aa538f',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '6b1d11d2-d807-4744-88d1-b72010aa538f',
   'content-type': 'text/xml',
   'content-length': '2548',
   'vary': 'accept-encoding',
   'date': 'Sun, 14 Feb 2021 07:51:14 GMT'},
  'RetryAttempts': 0}}
```


- run this block several times until the cluster really deleted

In [17]:
```
myClusterProps = redshift.describe_clusters(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)['Clusters'][0]
prettyRedshiftProps(myClusterProps)
```

Out[17]:

	Key	Value
0	ClusterIdentifier	dwhcluster
1	NodeType	dc2.large
2	ClusterStatus	deleting
3	MasterUsername	dwhuser
4	DBName	dwh
5	Endpoint	{'Address': 'dwhcluster.XXXXXXXXXXXX.us-west-2.redshift.amazonaws.com', 'Port': 5439}
6	VpcId	vpc-fXXXXXXXX
7	NumberOfNodes	4

In [18]:
```
#### CAREFUL!!
#-- Uncomment & run to delete the created resources
iam.detach_role_policy(RoleName=DWH_IAM_ROLE_NAME, PolicyArn="arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
)
iam.delete_role(RoleName=DWH_IAM_ROLE_NAME)
#### CAREFUL!!
```

Out[18]:
```
{'ResponseMetadata': {'RequestId': '2f75079d-9328-4d60-8826-63378feb3382',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '2f75079d-9328-4d60-8826-63378feb3382',
   'content-type': 'text/xml',
   'content-length': '200',
   'date': 'Sun, 14 Feb 2021 07:51:25 GMT'},
  'RetryAttempts': 0}}
```

In [ ]:
```

```