Migration to AWS RedShift

October 10, 2018

1 Objective:

Make a easy guideline for Python users who want to transfer its own database to the cloud.

2 Tools:

The tools necessary to make the migration are the following: - Python 3.x - Local Database (In this example, we will use PostgreSQL) - An account in AWS (We will use S3, EC2 and RedShift) - SQL Workbench/J (To make the queries in the clustered database in AWS)

Below, there are different websites to make the installation of the tools: - Python 3.x: https://www.python.org/downloads/ - AWS: https://aws.amazon.com/ - SQL Workbench/J: http://www.sql-workbench.net/manual/install.html Python libraries to develop this application: - psycopg2: make the connection to my local database and my clustered database - boto3: make the connection to AWS - csv: use csv files - time: suspend the execution of my program for few minutes

3 Steps:

The website https://docs.aws.amazon.com/redshift/latest/gsg/getting-started.html will help us to make the migration. Below, I will make an example how to use those steps to make the transference.

- Create IAM role:

To create IAM role, please follow the instructions in this website: https://docs.aws.amazon.com/redshift/latest/gsg/rs-gsg-create-an-iam-role.html

Be sure the role created has as a policy AmazonS3ReadOnlyAccess:

- Create a cluster (Optional):

To create a cluster in RedShift, follow the instructions in this website: https://docs.aws.amazon.com/redshift/latest/gsg/rs-gsg-launch-sample-cluster.html or run the application in Python. The app will create an example of a cluster with a default database (The function creating_cluster will do it). Below, there is a picture of our cluster.

- Configure VPC Security Group:

The following website will authorize access to the cluster https://docs.aws.amazon.com/redshift/latest/gsg/rs-gsg-authorize-cluster-access.html . Make sure to configure the security group related to the cluster to authorize access:

- Connect to the cluster:

Using SQL Workbench/J, we will make the connection to the cluster. First, we need the driver to connect SQL Workbench to AWS cluster. In this website, we will do it https://docs.aws.amazon.com/redshift/latest/mgmt/configure-jdbc-connection.html. Then, we have to configure Workbench using the following website https://docs.aws.amazon.com/redshift/latest/gsg/rs-gsg-connect-to-cluster.html.

- Create a bucket (Optional):

To create a bucket in S3, follow the instructions in this website: https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html or run the application in Python. The app will create an example of a bucket (The function creating_bucket will do it)

- Creating local files from our local database:

We will do it using the application in Python. The following picture will show our database tables in PostgreSQL.

Running the application will convert to local files (the function convert_to_local_files will make this). Below, we will see a picture where converts to a local files (csv files)

- Get access to AWS intance from local:

This is a important step to achieve the access to Linux instance created in AWS. To do this, I prefer to use Putty to get access using fingerprints (you can consult this website to do it: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html) or you are free to use another method (this website can help you with that https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html)

- Transfer local files to my bucket:

Also, we will make use of our application. The function put_files_bucket will be in charge of doing this. Below, we will see some pictures which demonstrate how will look our bucket.

- Copy the files from my bucket to my clustered database:

At the end, we will do this using the application as well. The function copy_from_s3_to_redshift will make this. Below, we will see a picture of our

4 Conclusion:

This steps allowed us to make the migration from a local database to the cluster, in this case AWS. This