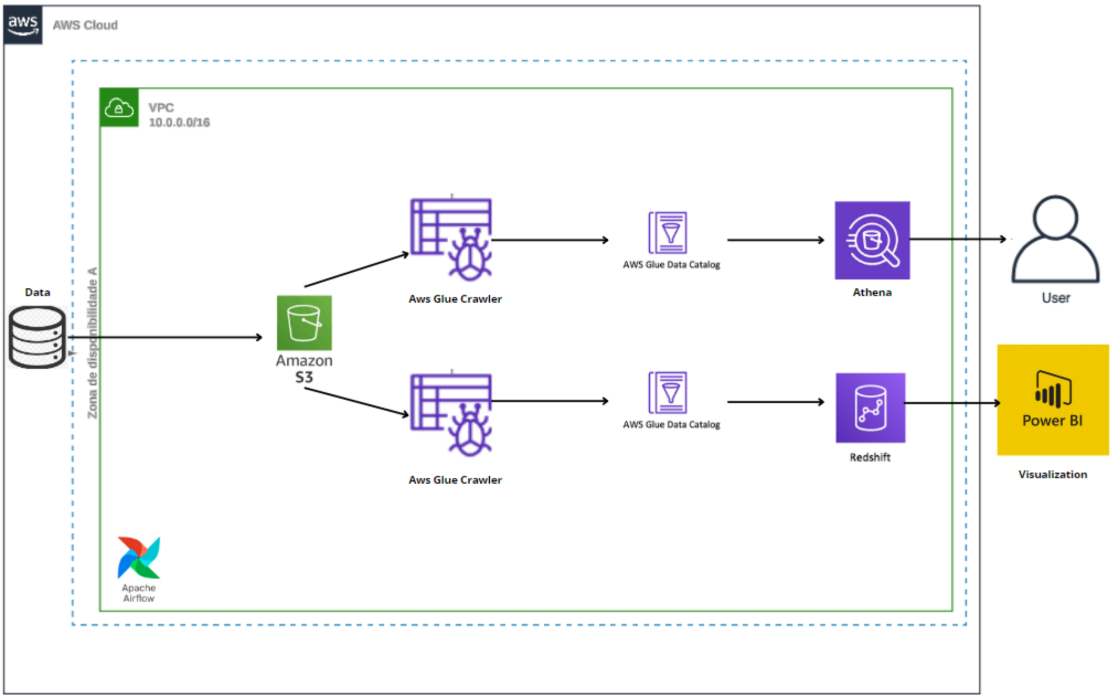
Customer Churn Analysis through a Data Pipeline using Apache Airflow, AWS Glue, S3, Amazon Redshift and Power BI

Summary

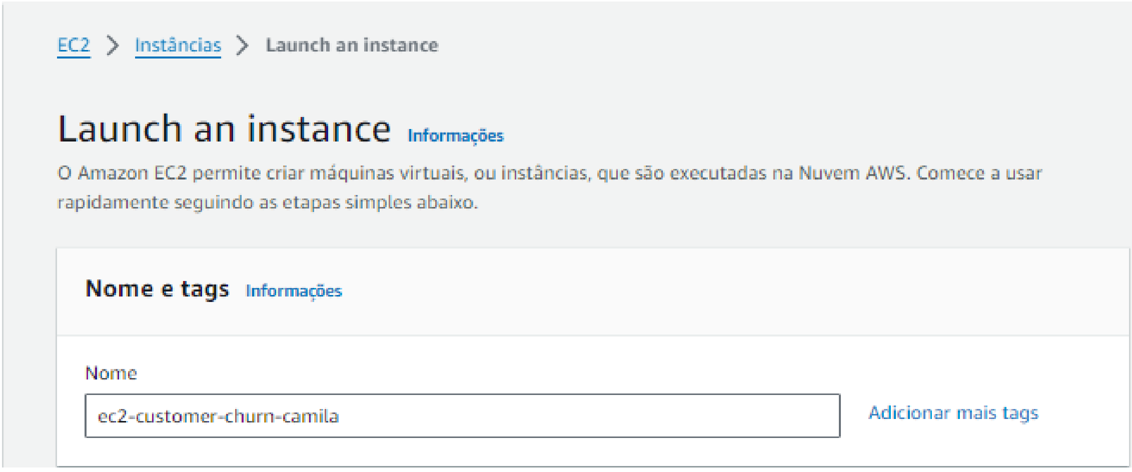
In this data engineering project, I performed a customer churn analysis, built, and automated an ETL pipeline using AWS Glue to load data from the AWS S3 bucket into an Amazon Redshift data warehouse and then connect Power BI to the Redshift cluster for end user view. AWS Glue served as a data crawler to infer the database schema after the data was made available in AWS Athena to extract the data through SQL queries. AWS Glue also served to upload the tracked and processed data to the Redshift cluster. Apache Airflow was used to orchestrate and automate this entire process that was previously manual.

AWS cloud architecture diagram:

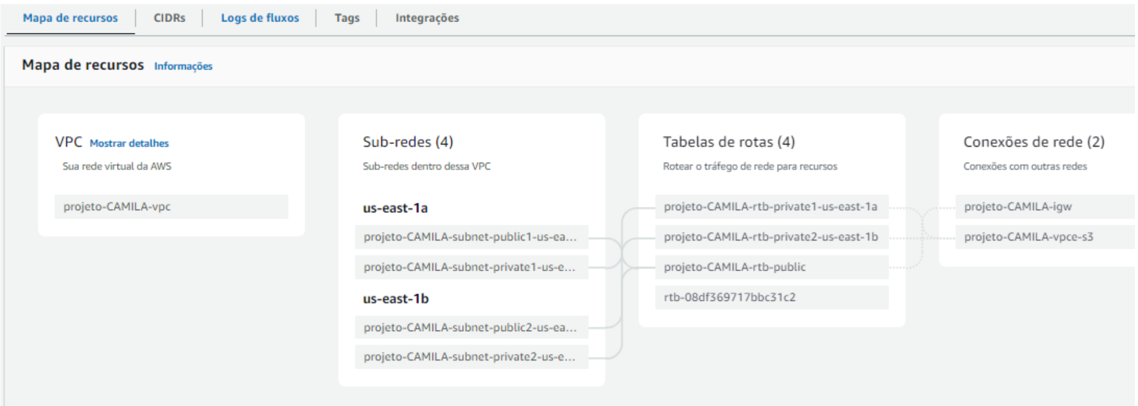


The entire project was developed from scratch in the AWS cloud, a database used by Kaggle, called "Telecommunications customer turnover: IBM dataset", below are the steps.

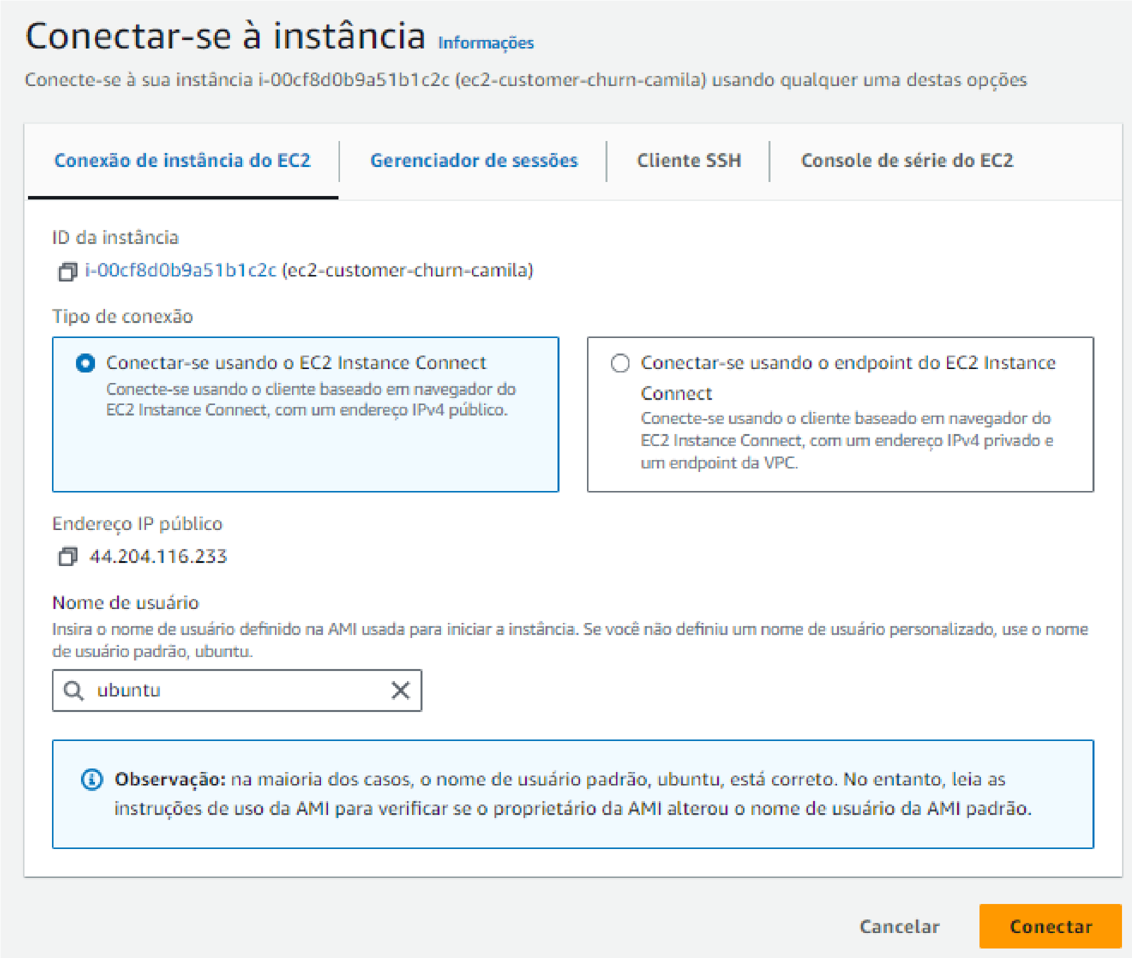
First, I created an EC2, for process configurations and orchestration. In the AWS console, search for EC2, then "Run EC2". The AMI chosen was Ubuntu, the initial configurations will be made in Linux, via the EC2 terminal. I enabled the 3 incoming connection rules SSH, HTTPS and HTTP, so it will be possible to connect via command line and browser, as the initial load will be low, choosing the t2.medium instance type, which has 4Gib and 2 CPUs, this guarantees that EC2 will not freeze during the process.



I also generated a pair key, downloaded the pair key in csv, for security reasons and to be able to connect via SSH, in addition, a VPC was created to protect EC2.



When the instance is in “Running” status, select it and click connect. At first, I connected via the connect instance to start the settings. Click “Connect” again.



A new terminal will open, I configured it with the following Linux commands:

sudo proper update (Updates a list of packages available for installation on the operating system using the APT (Advanced Package Tool) package manager)

sudo apt install python3-pip (Install the python3-pip package, which is the Python package manager for version 3.x, used to install and manage Python libraries and dependencies.)

sudo apt install python3.10-venv (Install the python3.10-venv package, which is required to create Python 3.10 virtual environments, allowing you to isolate Python development environments for specific projects.)

python3 -m venv client_churn_camila_venv (Create a virtual environment called customer_churn_camila_venv using the Python 3 venv module. This virtual environment is where the project's dependencies will be installed and isolated from the global system.)

source customer_churn_camila_venv /bin/ activate (Activates the previously created virtual environment, ensuring that subsequent installations and executions occur within this isolated environment.)

sudo pip install apache-airflow (Install Apache Airflow, a platform for scheduling, monitoring, and managing data workflows. Here, we are using pip (Python Package Installer) to install Airflow.)

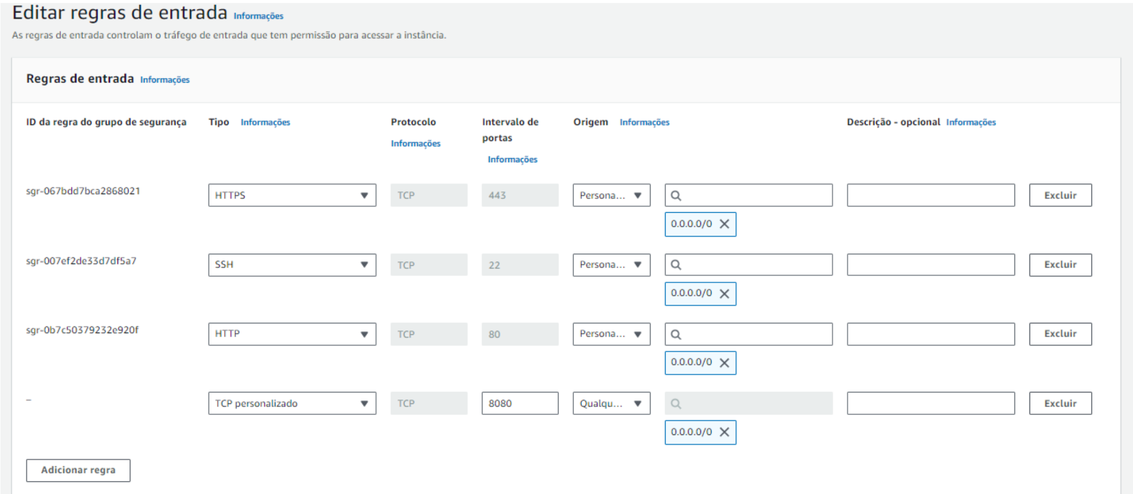
pip install apache- airflow - providers-amazon (Install a specific Apache Airflow provider for integration with Amazon Web Services (AWS) services such as S3, EC2, etc. This provides additional functionality for working with AWS services within Apache Airflow.)

air flow be alone (Start Apache Airflow in standalone mode, which means it will run on a single node without using an external database such as MySQL or PostgreSQL. This mode is useful for development or testing setups.)

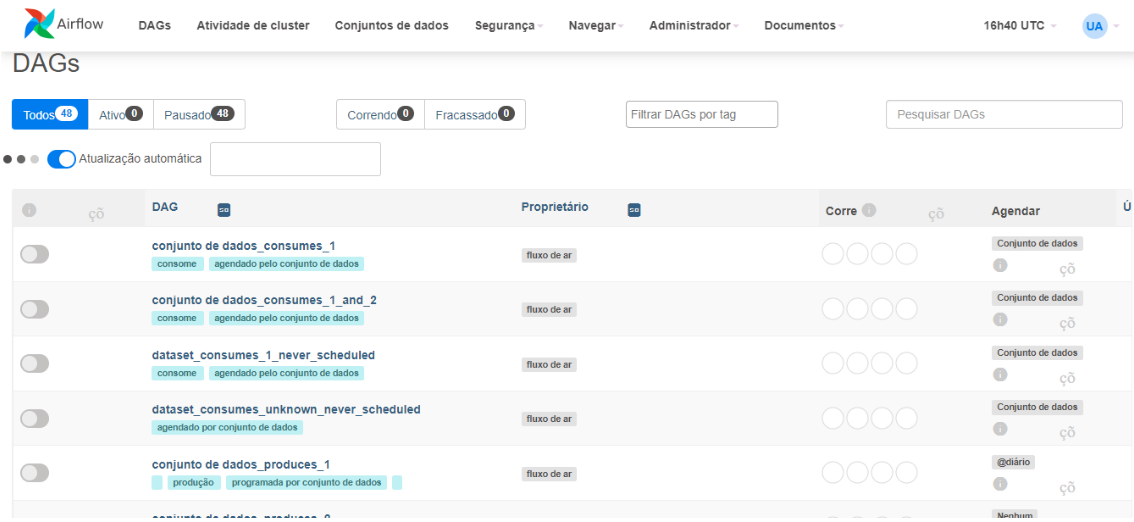
After configuring the virtual environment and dependencies, Airflow should indicate that it is ready and will provide a username and password, as follows:

My username: admin password : 9xYkerwf6v3v97qE.

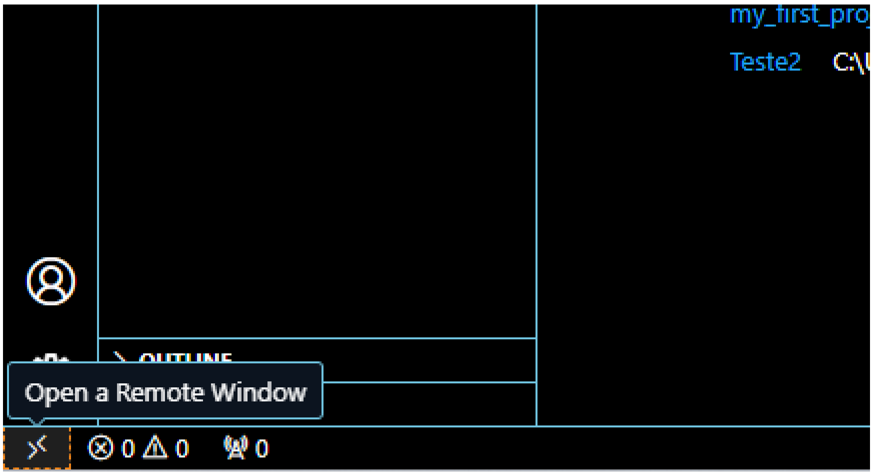
After that, go back to the created EC2, select and in the “Security” tab, click on Security groups, then on Inbound Rules and Edit Inbound Rules, add the rule to access Apache Airflow via browser, this will be important to view whether the job you created is working correctly.



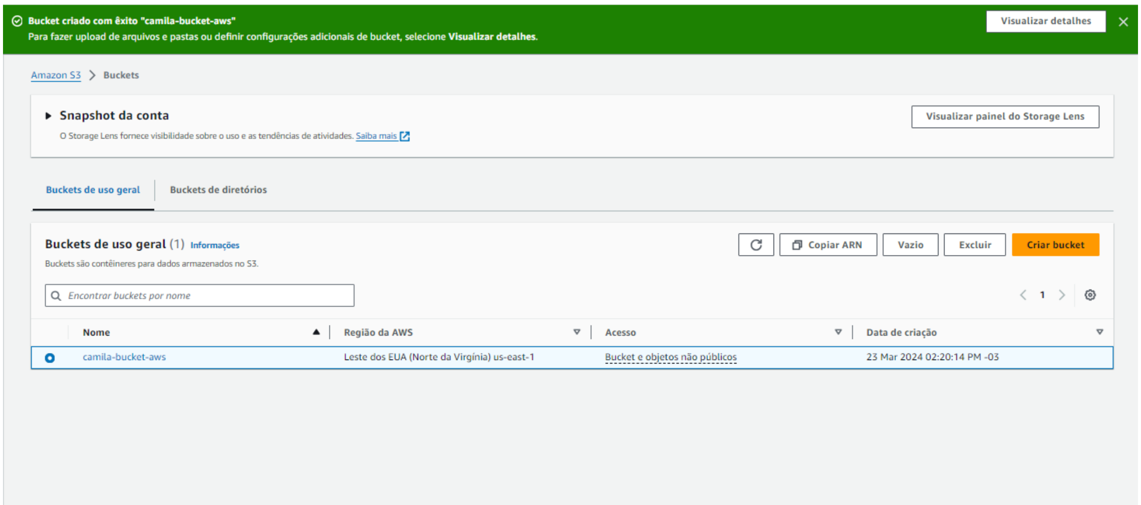
Add the rule, Custom TCP type, port range 8080, and Source Anywhere IPv4. Port 8080 is a network communication port. In many cases, it is used as the default port for local or development web servers. When you run Apache Airflow with the **airflow standalone command,** by default it uses port 8080 to provide a web interface where you can monitor and manage workflows. So, when you start Airflow with **airflow standalone,** you can access its web interface by typing **http://localhost:8080** in the browser. Our localhost is our EC2 public IP address. After logging into Airflow through the browser with the credentials he had created.



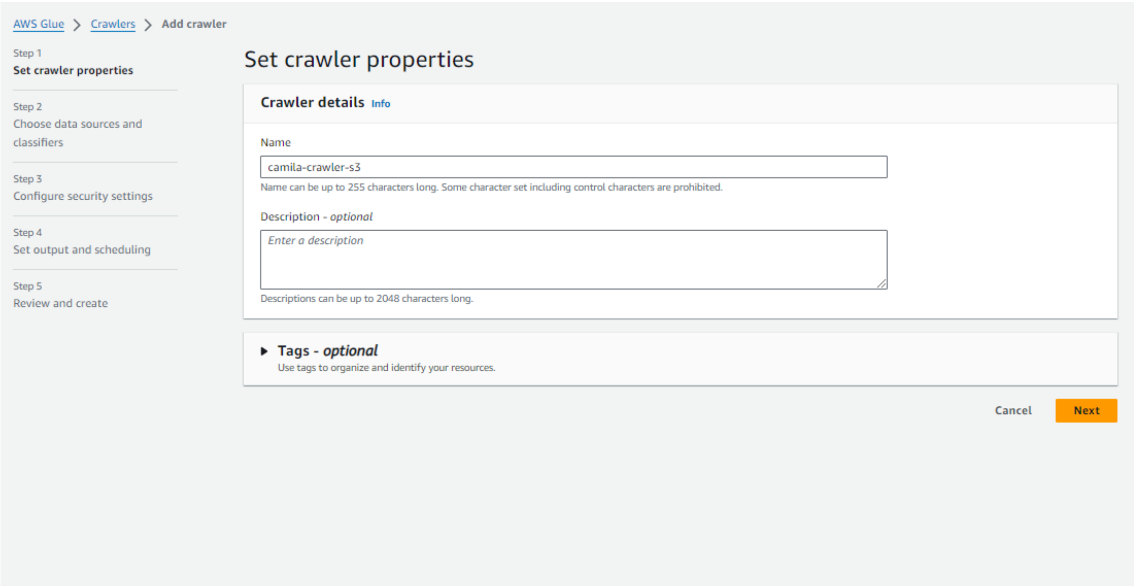
Then I set up remote access via SSH through VScode, because it's more user-friendly to use VIM or the EC2 terminal. In the footer on the left, "Open a remote window", connect to the Host, inform your EC2 and connect to the virtual environment created previously.



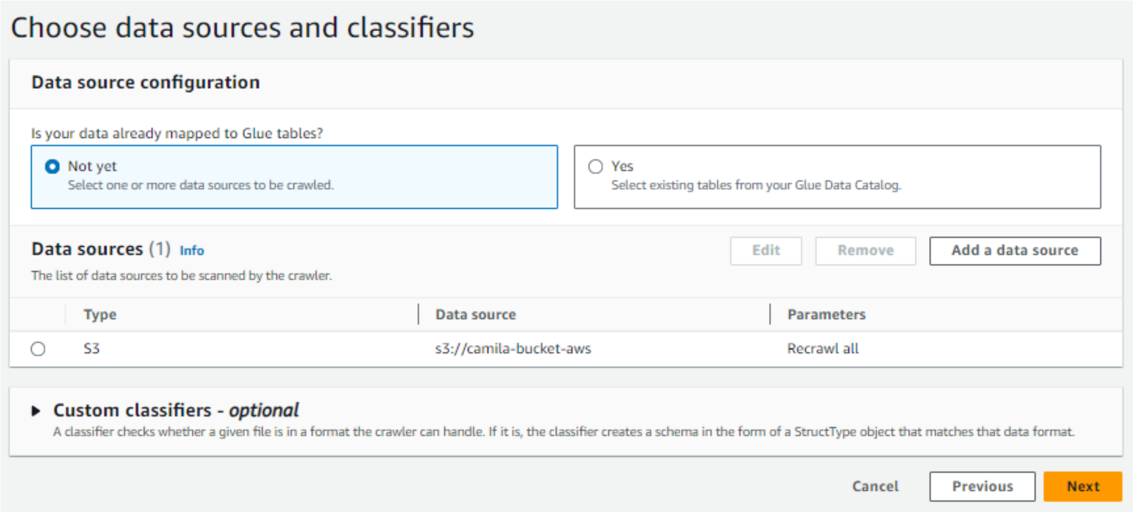
S3 bucket, it is like a cloud storage folder where you can store any type of data, such as files, documents, images, videos, database backups, among others. I stored the data in the S3 bucket.



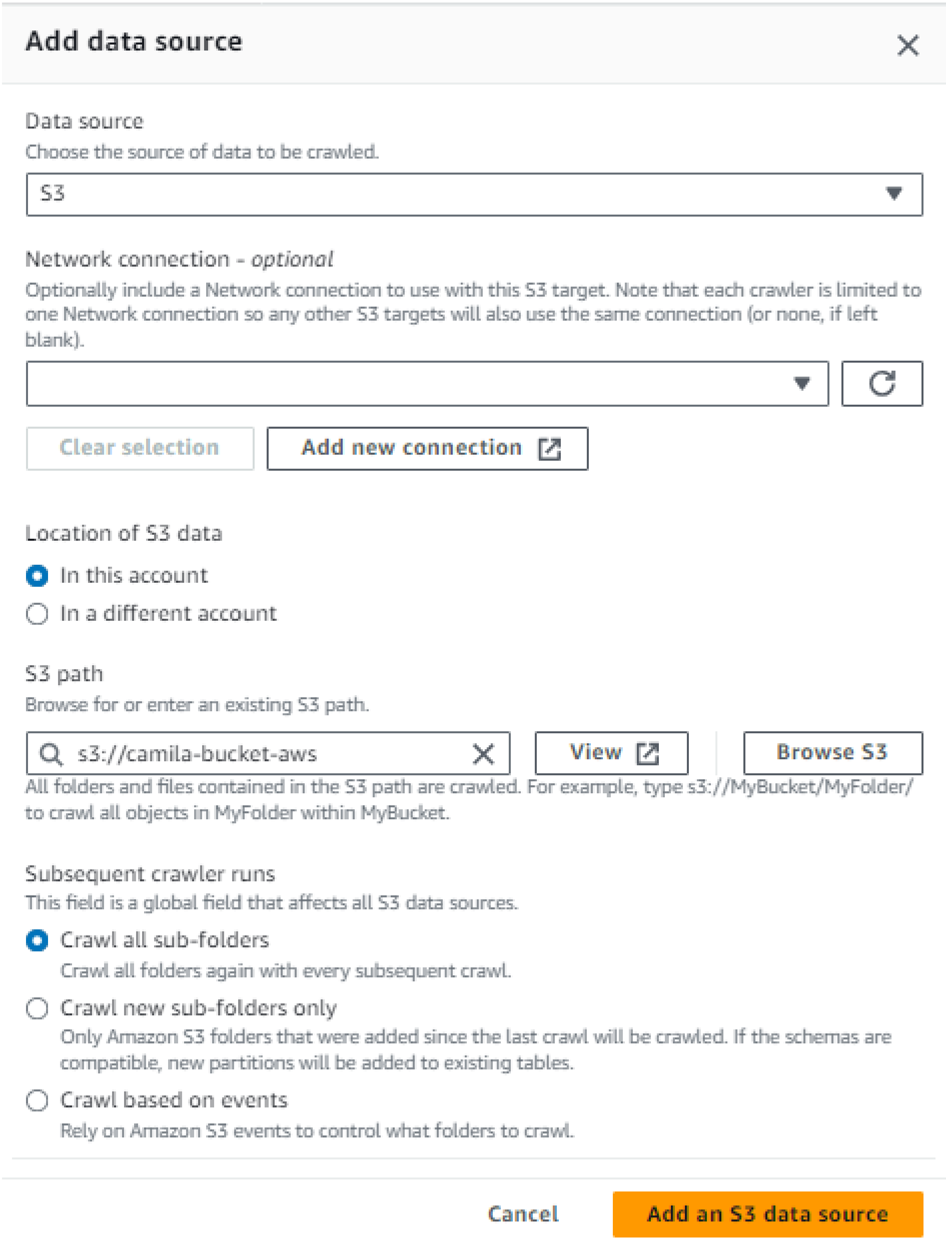
Then I created AWS Glue, which provides an easy way to create, manage, and run ETL (Extract, Transform, Load) pipelines to process and transform large volumes of data. I used the resource called AWS Glue Crawler, it automates the discovery and classification of data in various data sources, such as Amazon S3, related and non-relational databases, so it will also be used with Athena and Redshift. We investigate using AWS Glue, then in “Data catalog”, “Crowers” we will process the data that will be stored in S3.



As my table schema was not created to do zero.

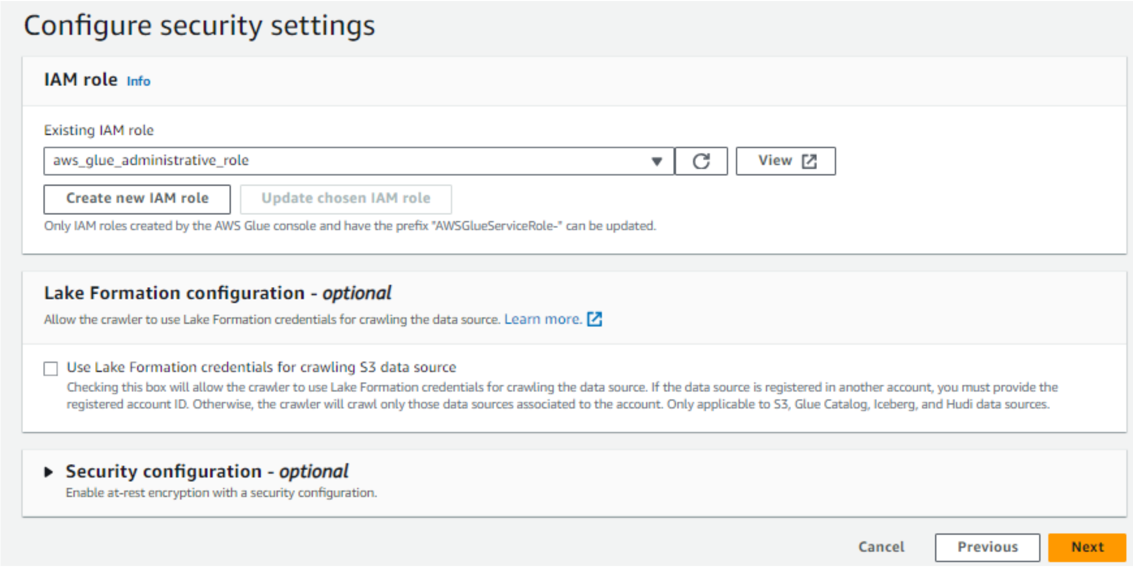


I added the data source, which will be the contents of the S3 bucket.

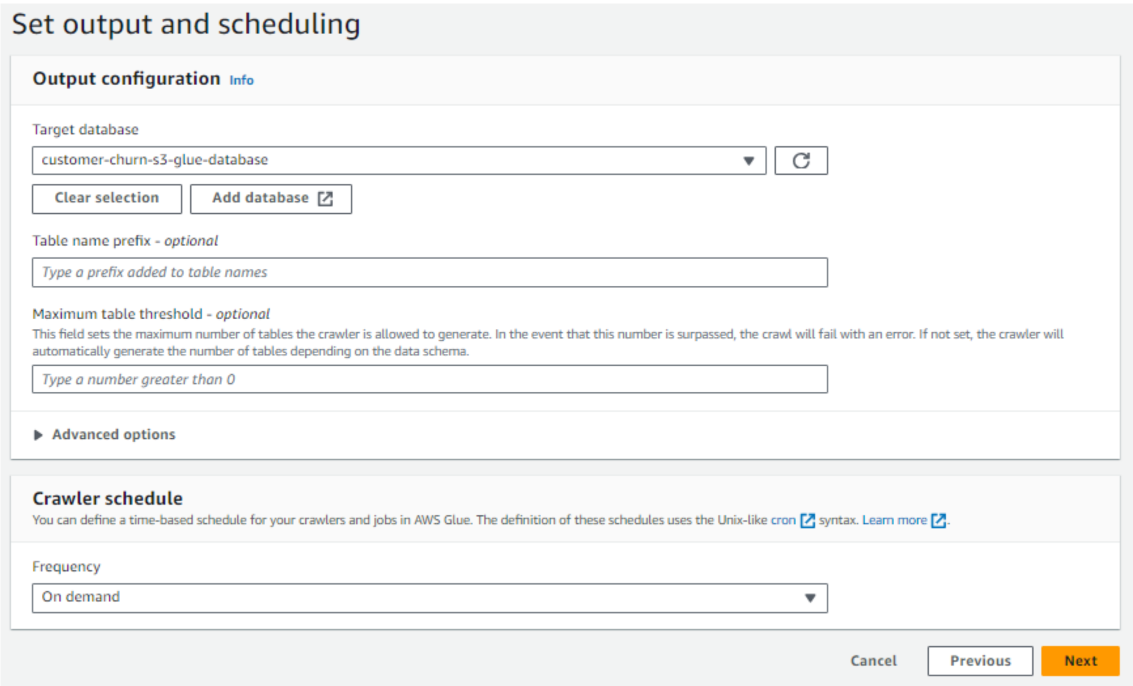


It will be necessary to inform the S3 path and I set it to “Crawl all subfolders” because I want the bucket data to be updated with each new load.

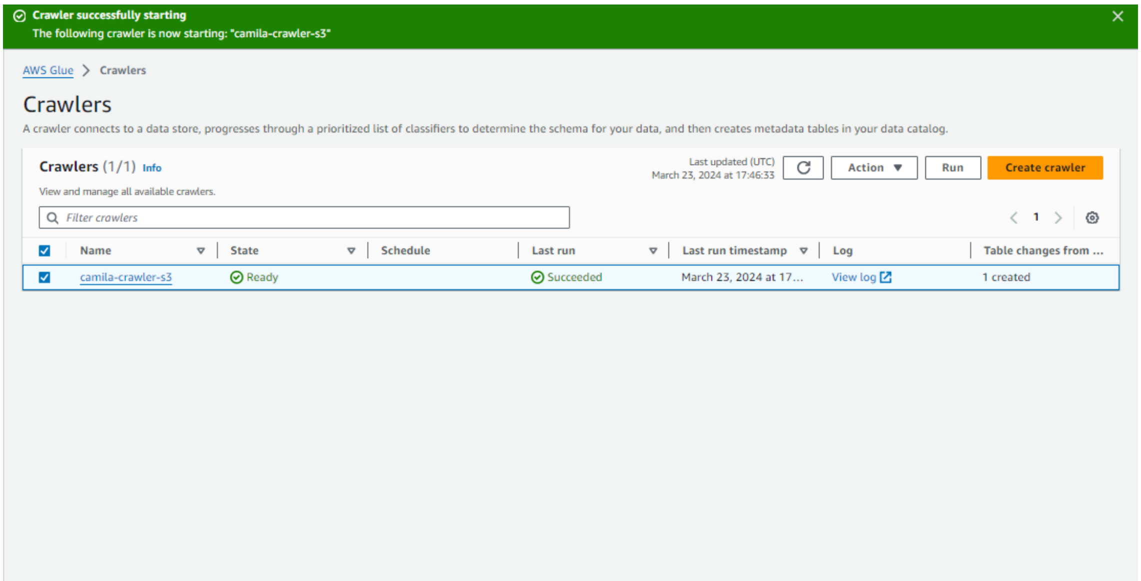
The AWS Glue Crawler determines that the IAM role must already be created to operate the process, so it adds the administrator user permission to manage Glue.



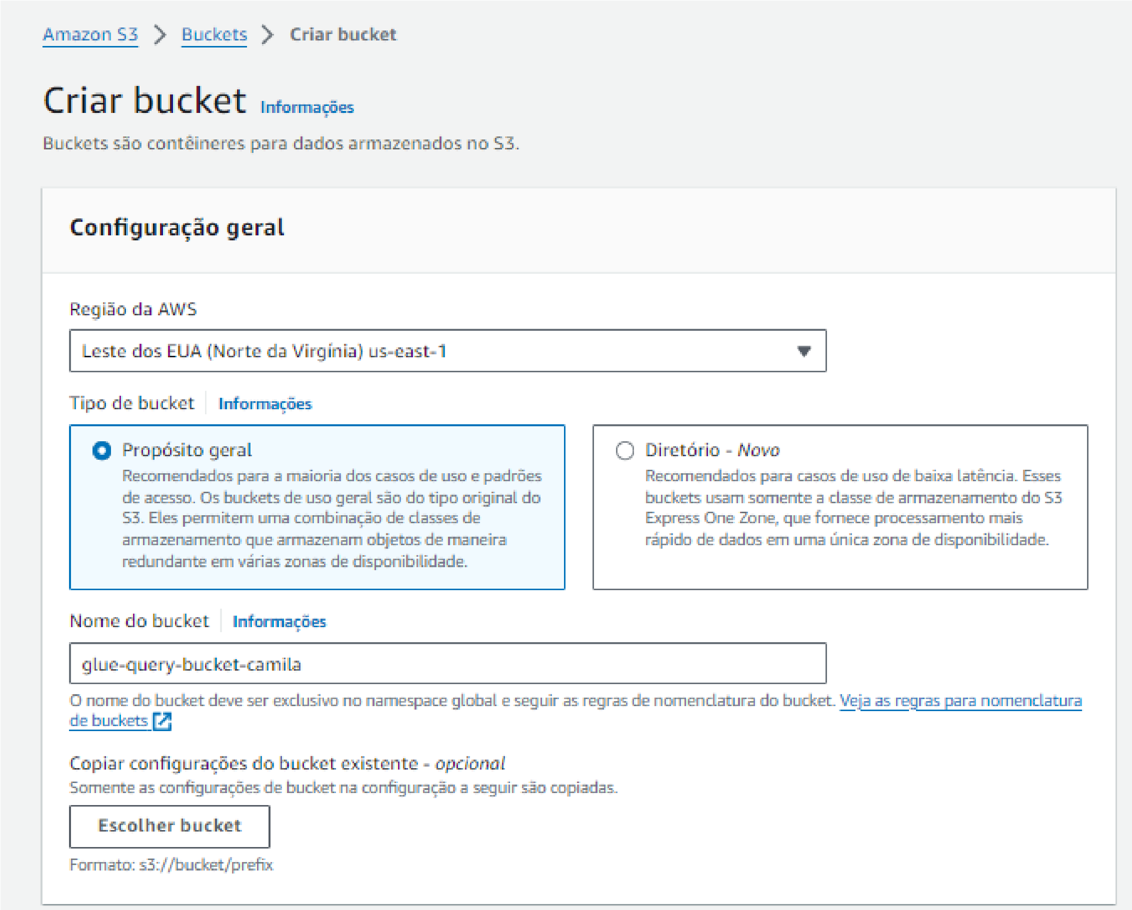
I added a database that I had already added to the bucket.



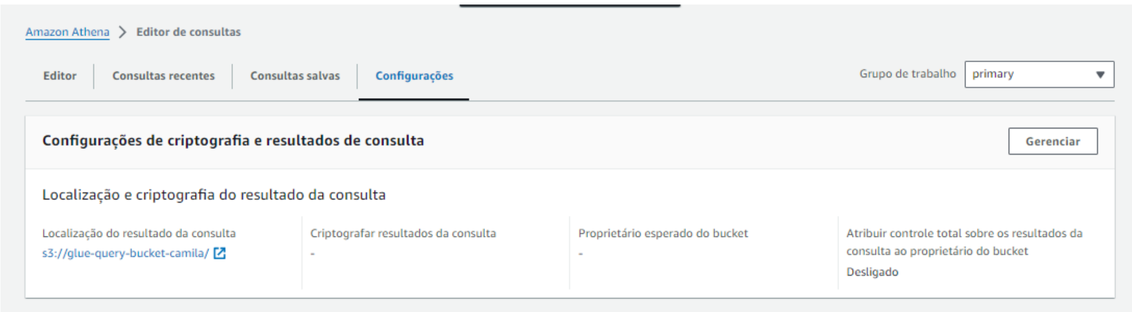
I kept the other default settings and reviewed and then created, then press “Run” and wait until it reaches “Ready” status.



Now we will use the Crawler created to connect with AWS Athena. AWS Athena is an interactive data query service provided by AWS that allows you to analyze directly in Amazon S3 using standard SQL. When loading data into a database or traditional data store, Athena allows you to run SQL queries directly against files stored in S3, so you don't need to set up and manage a database infrastructure. The queries I made to create a report for the end user must be stored in another S3 Bucket, which is why I created another bucket. After creating the S3, I can adjust the Athena settings.



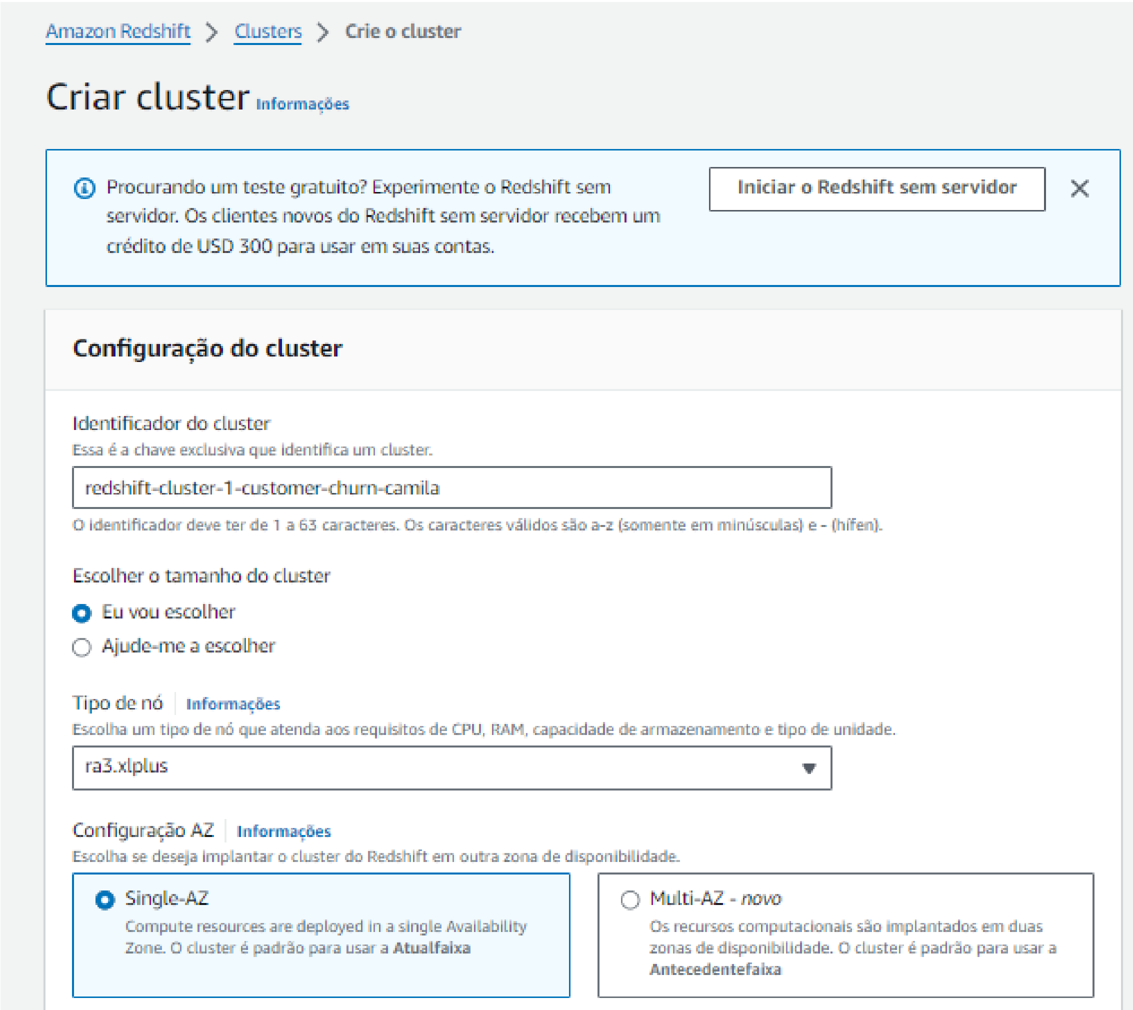
After "Settings", in Athena I selected this created bucket.



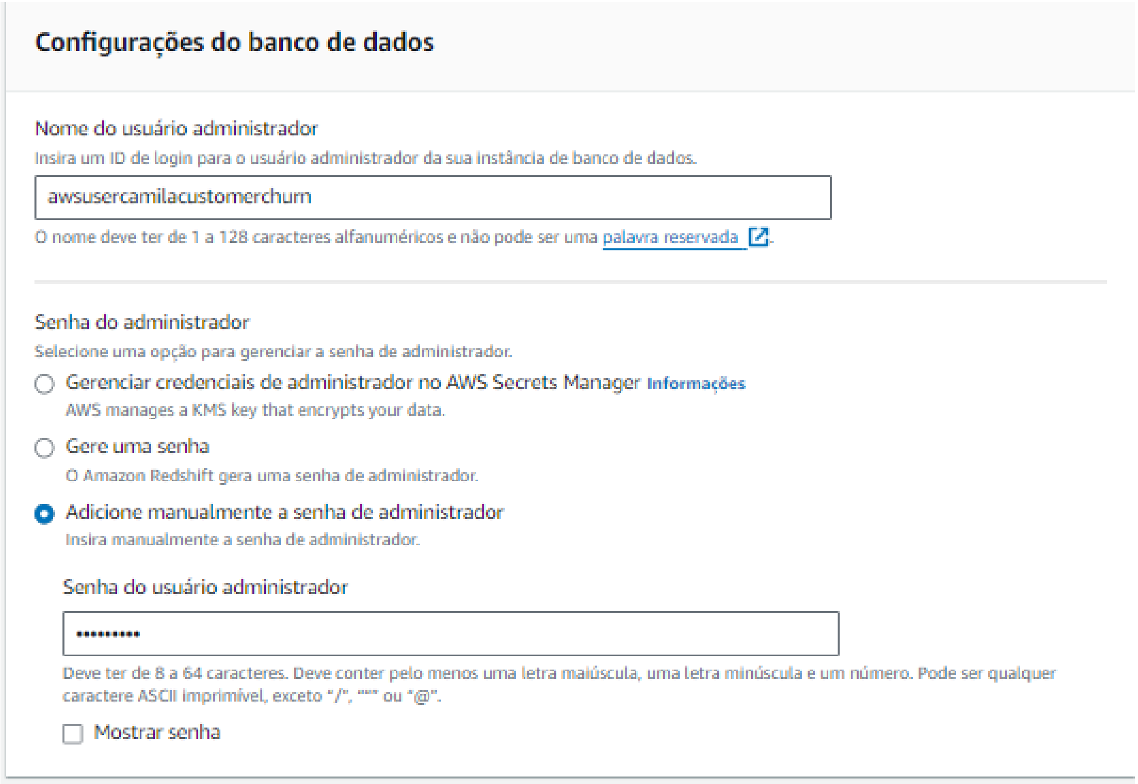
And in the "Editor" tab we can create and execute our queries. It looks like PostgresSQL or MySql, in this case would be for users to perform queries on a structured database, so Athena was connected via the AWS Glue Crawler.

After the analysis and the end-user dashboard, I used AWS Redshift. It is designed to process large volumes of data and perform complex analysis in real time. Redshift is based on a columnar relational database model and has been optimized to provide high performance in analytical queries, aggregations, and processing large data sets. Redshift easily integrates with other AWS tools and services, such as AWS Glue for data ETL (Extract, Transform, Load), AWS Data Pipeline for orchestrating data workflows, and Amazon S3 for data storage. Therefore, Crawler will also be integrated with Redshift.

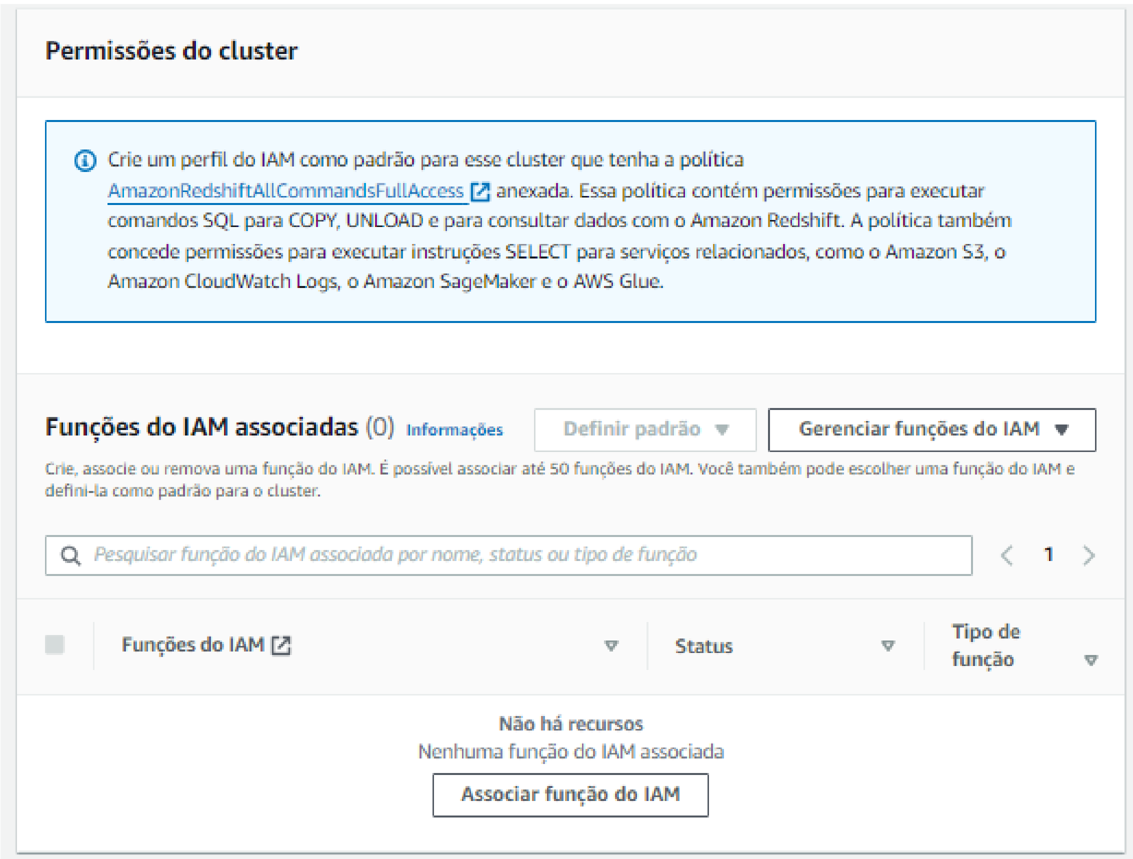
In the console, search for Amazon Redshift, then Clusters and finally create the cluster. Due to load size and cost optimization, I decided to choose the ra3.xlplus node type and an availability zone.



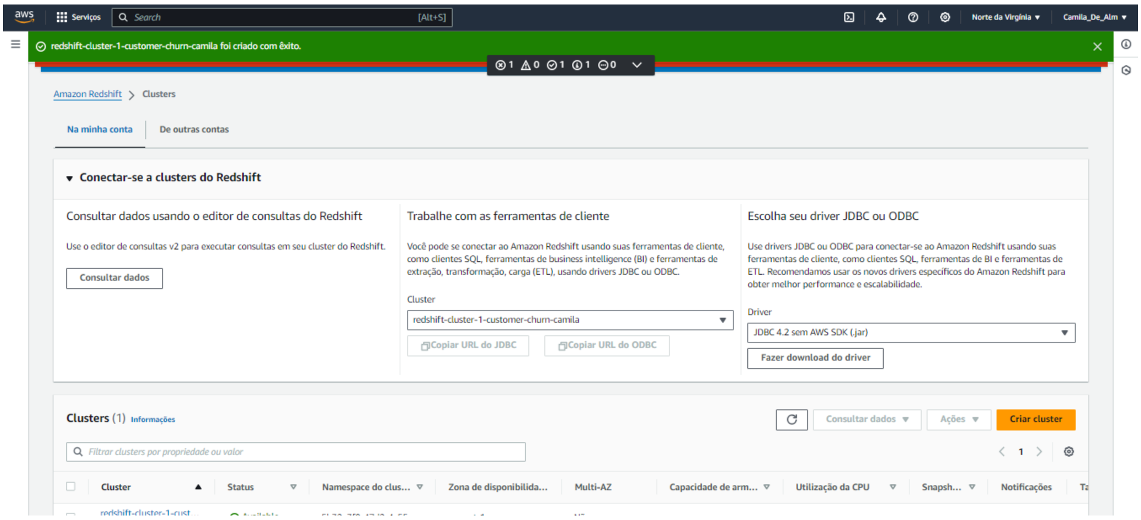
About database settings, choose admin username and set password.



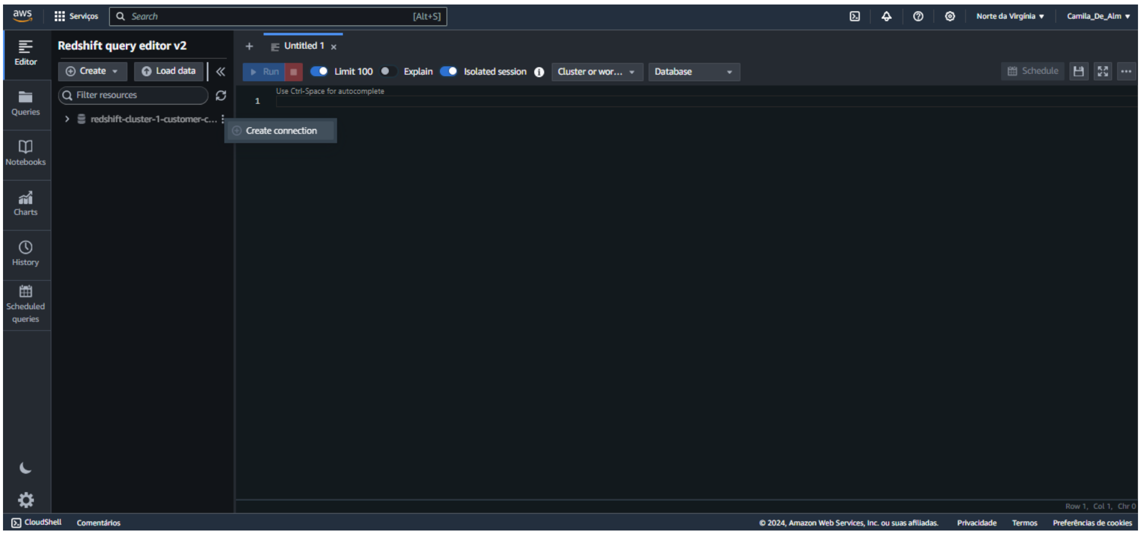
Redshift has already created an IAM profile with full access, as shown below.



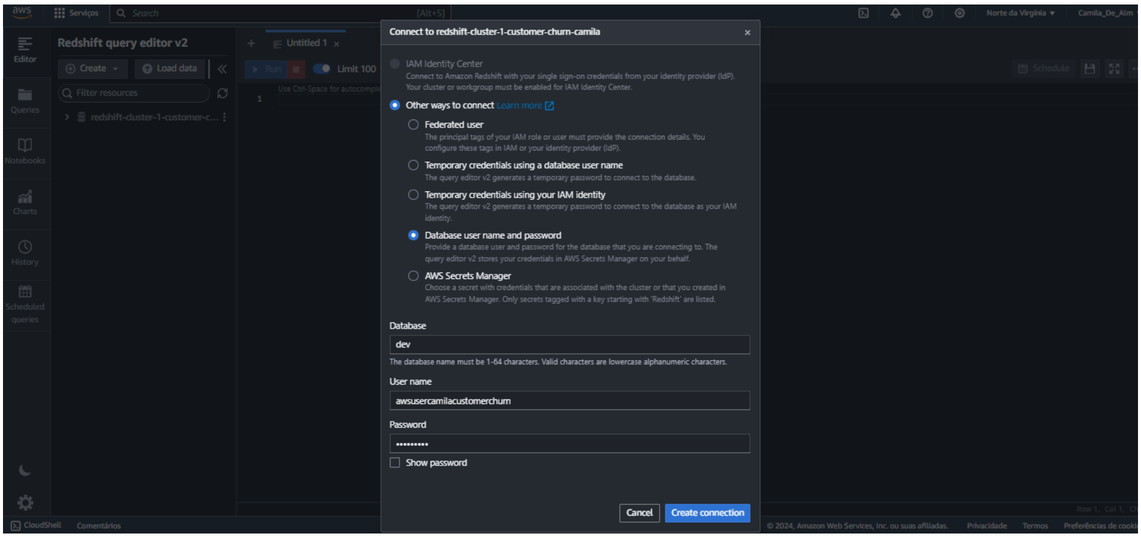
The other settings remain the same, as I had already created a VPC, I just had to select it.



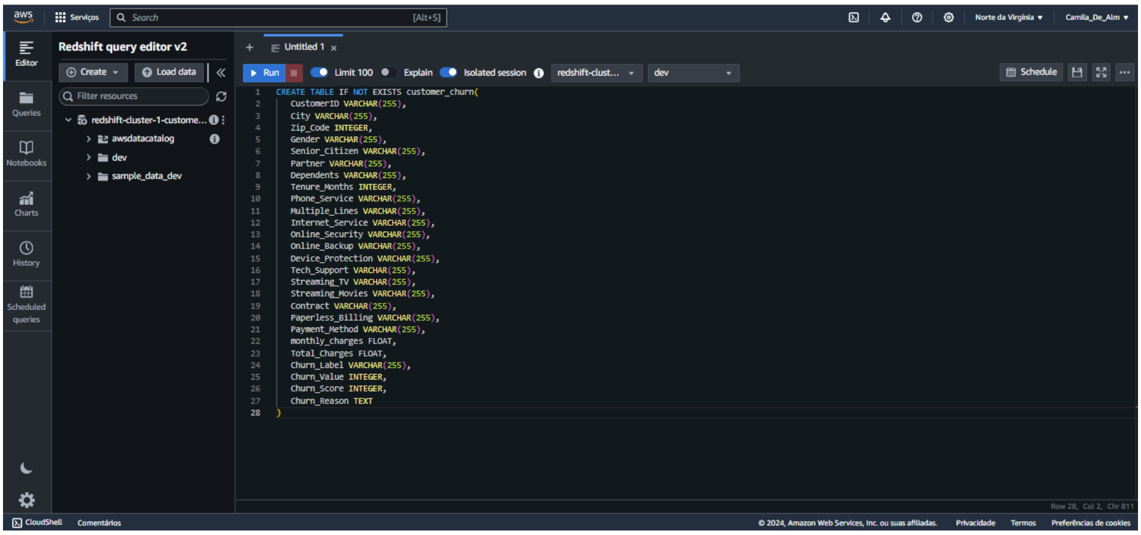
Then on the right click on Query Editor V2, open the editor below, in the three dots click on “Create connection.”



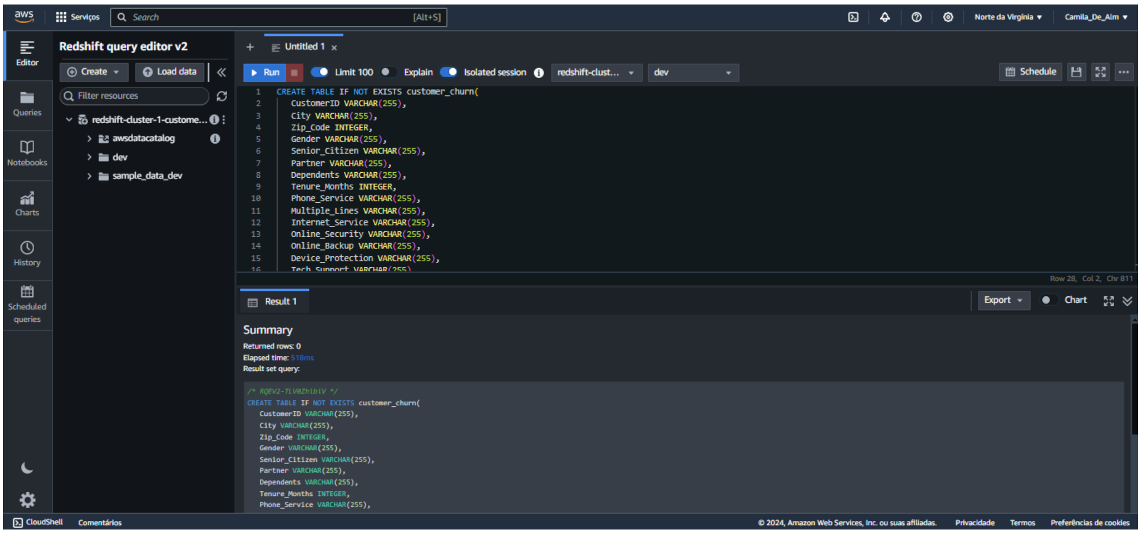
Now you must select the Database username and password option and enter the username and password that we defined when creating the Cluster. The database name was defined by the Redshift editor itself but can be changed.



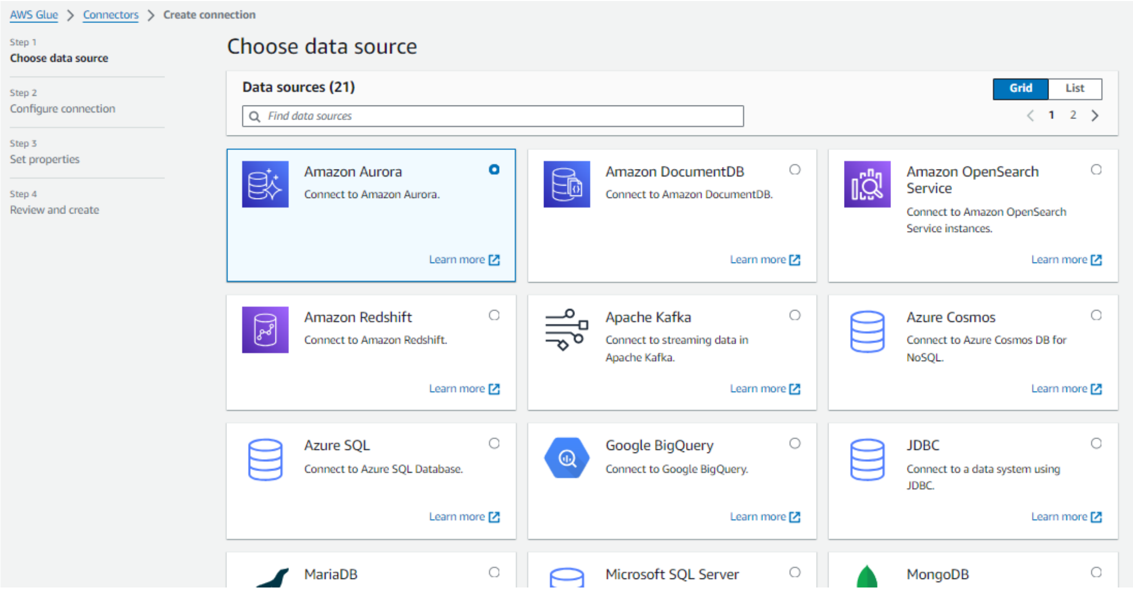
After creating the Schema of our table, as I already knew the Kaggle database, it was enough to format it in SQL format, as shown below.



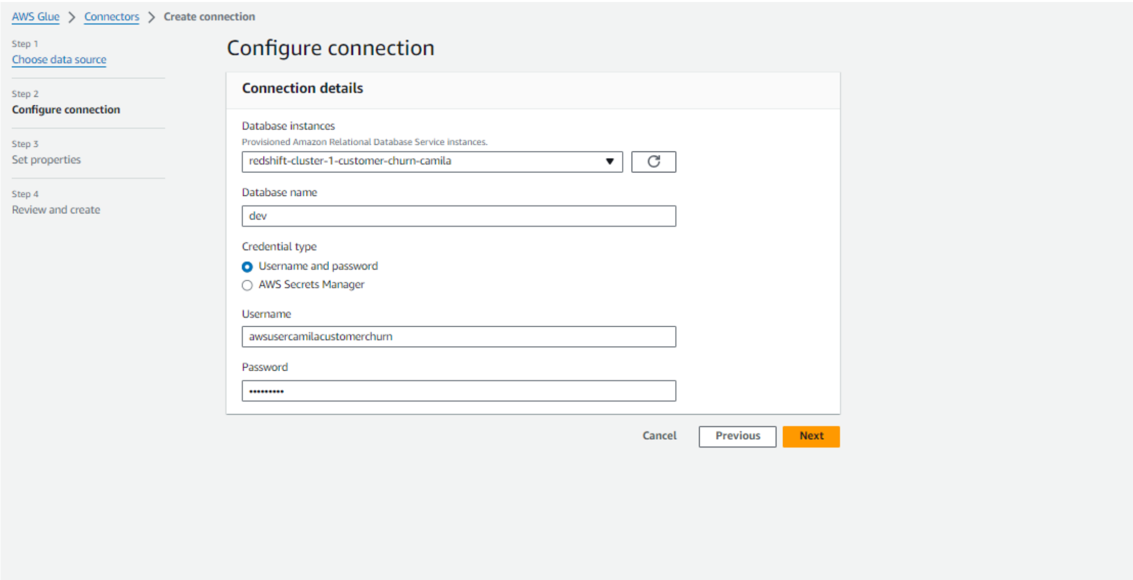
I executed the command.



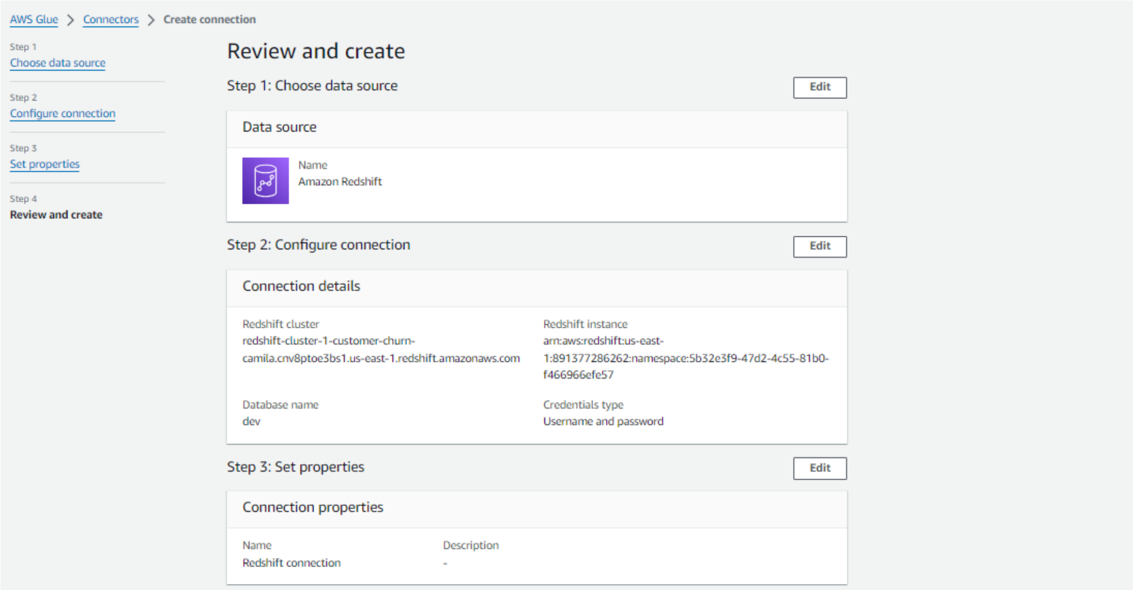
Now we need to connect Redshift to Glue. Then go back to Paste in Data Connections on the right, click on "Create connection" we can connect with different data sources, but I chose Redshift.



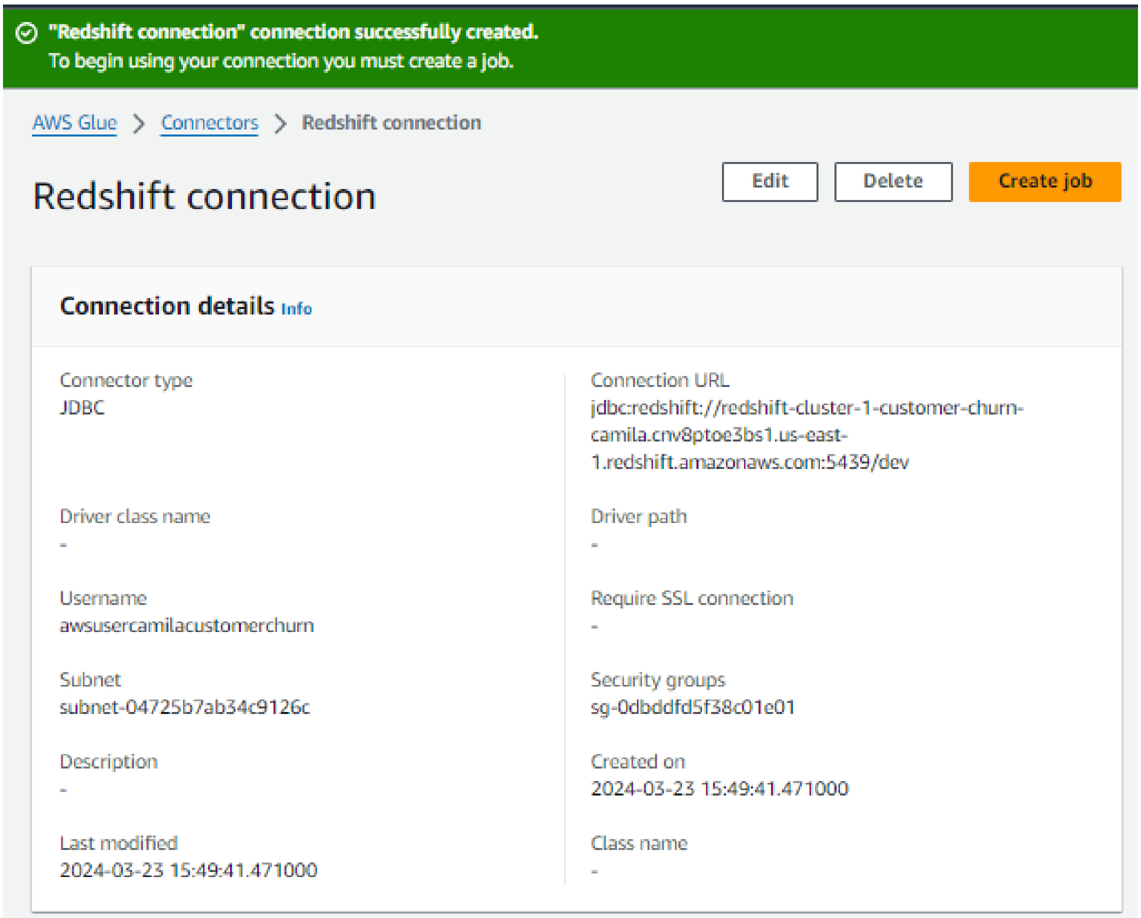
I provided the cluster created and the credentials.



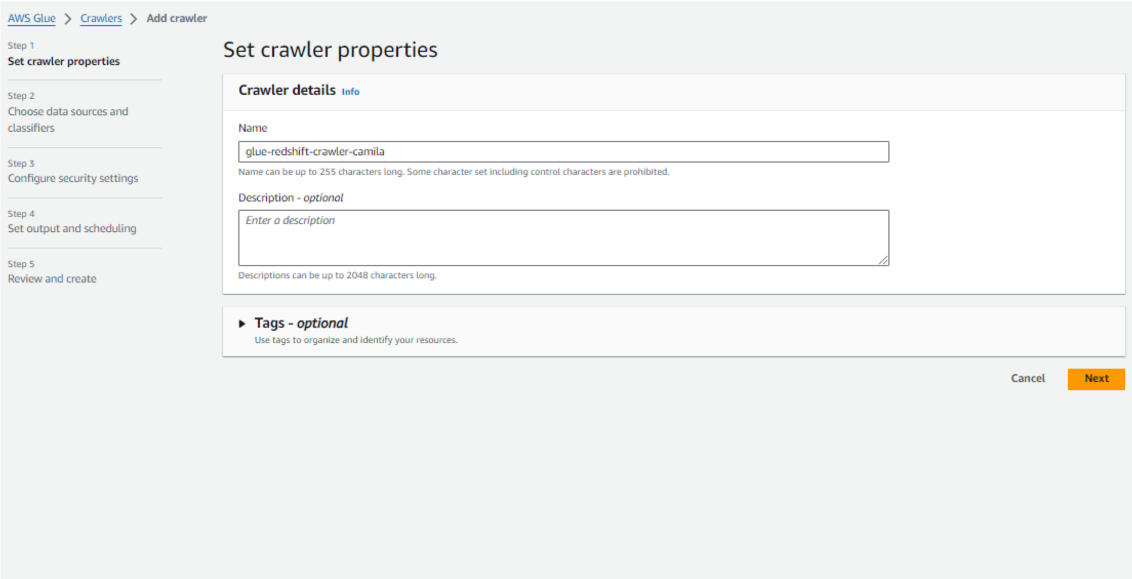
I revised and created.



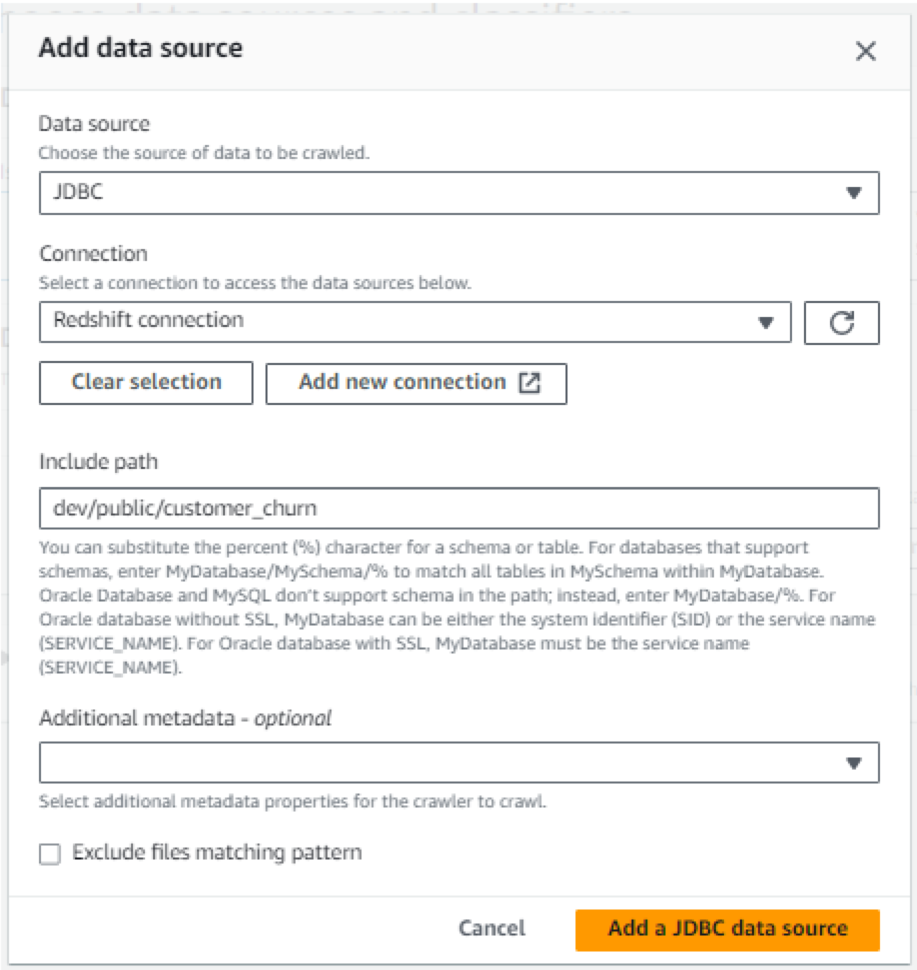
Connector type is JDBC which acts as a bridge between the application and the database, provides the means to establish a connection to the database, send SQL queries, receive results, and manipulate data, and is the default connector for Redshift.



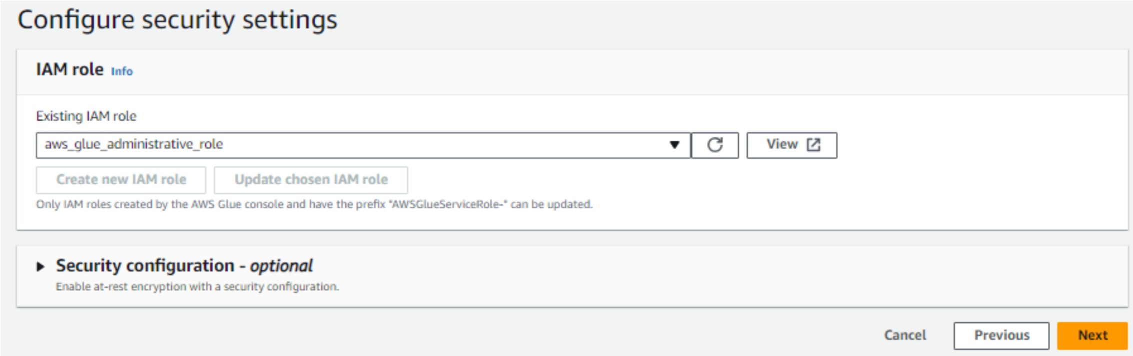
Now we need to connect Crawler (from AWS Glue) to Redshift, go to Crawler and add a new one for Redshift, as below.



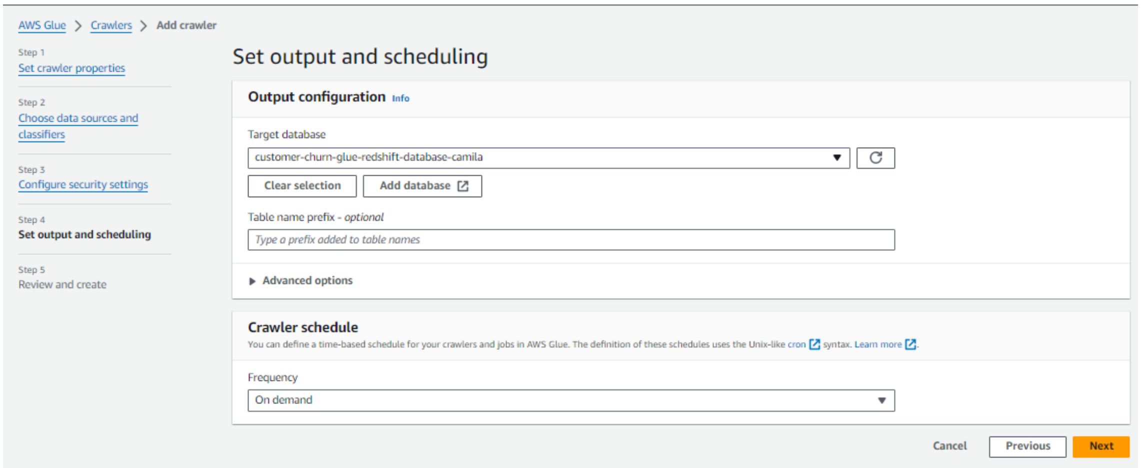
I clicked on "Next", then added a new database and the JDBC connection, which is Redshift, and entered the path.

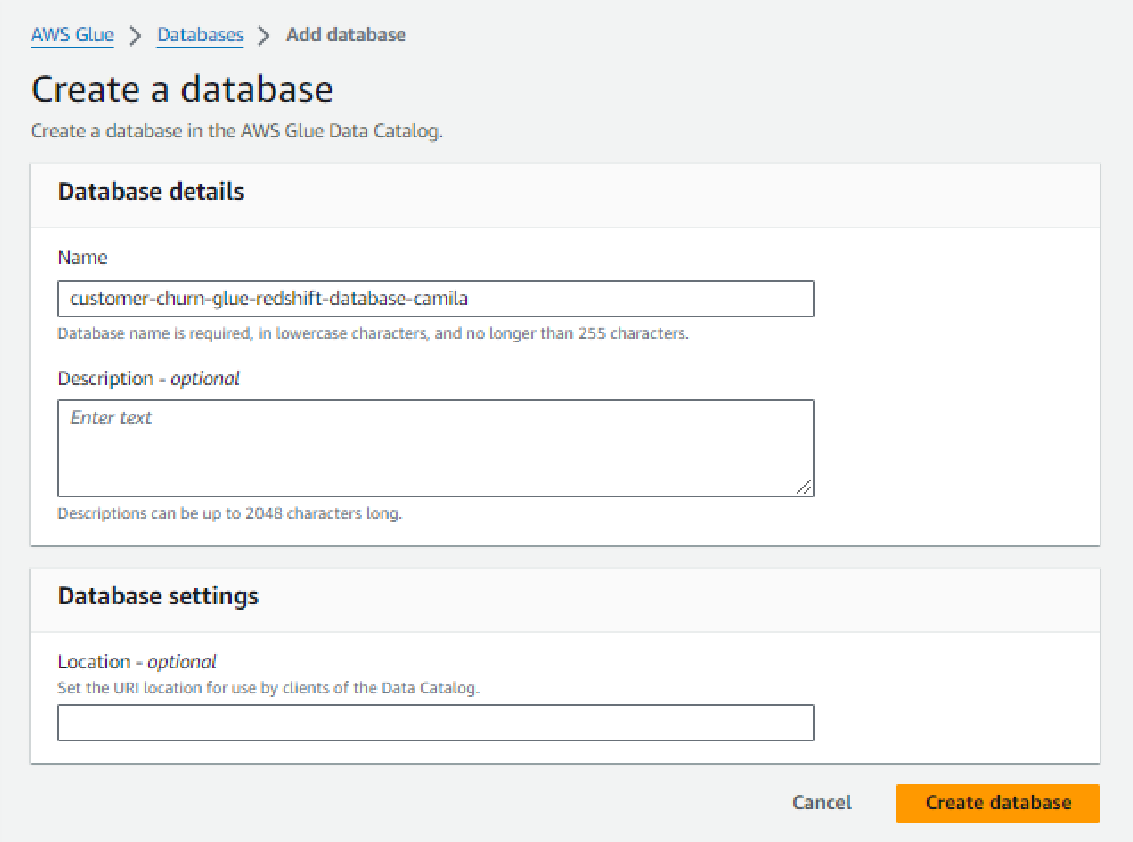


The IAM role could be the same as the one created for Athena.

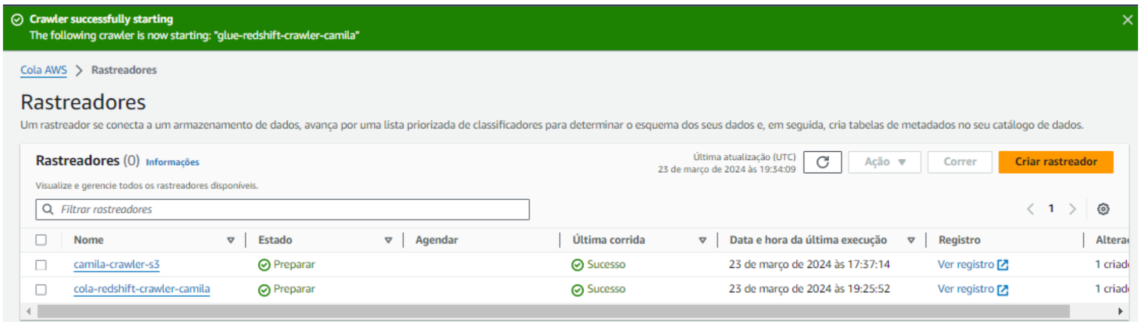


In “Target database” we will have to create one for Redshift. I created it in add database and then selected it.



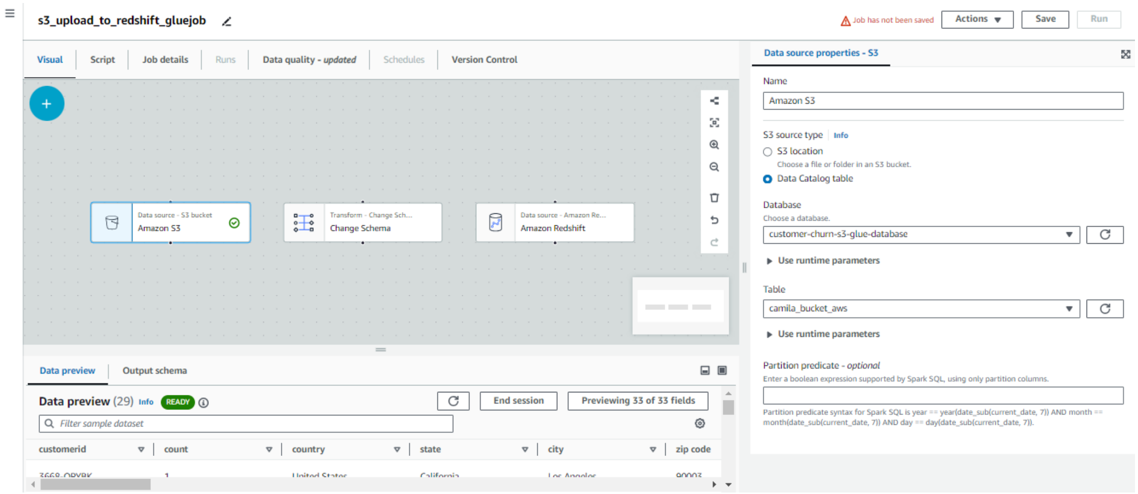


We now have two Crawlers on AWS Glue, one to integrate with Redshift and the other with Athena.

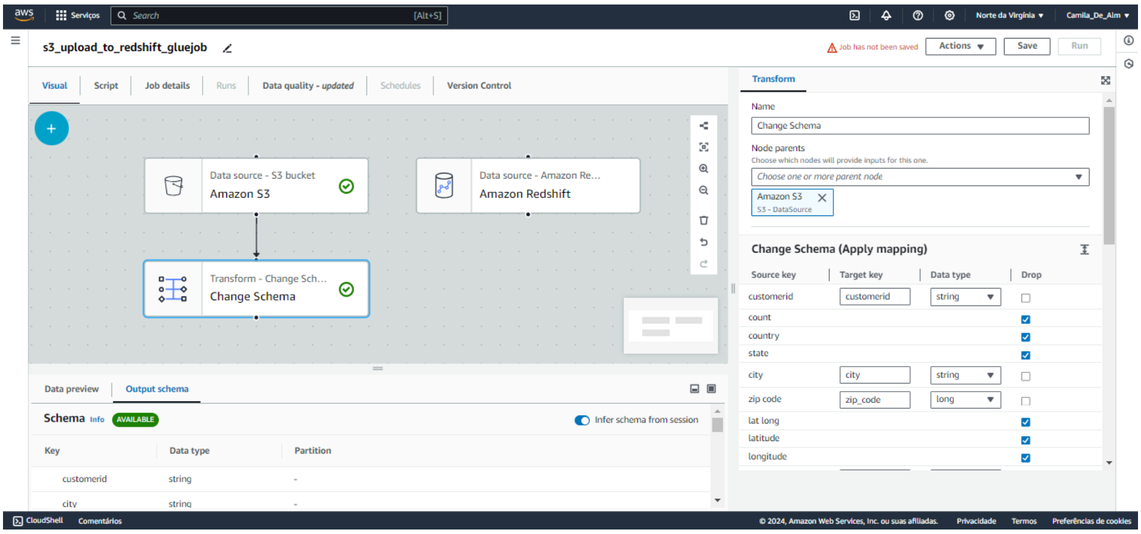


Still in AWS Glue, we need to configure the ETL job, in AWS Glue, on the right under ETL Jobs. Open the screen below, I named the job and in Visual, I configured each part of the ETL, just click on each item and configure the settings on the right.

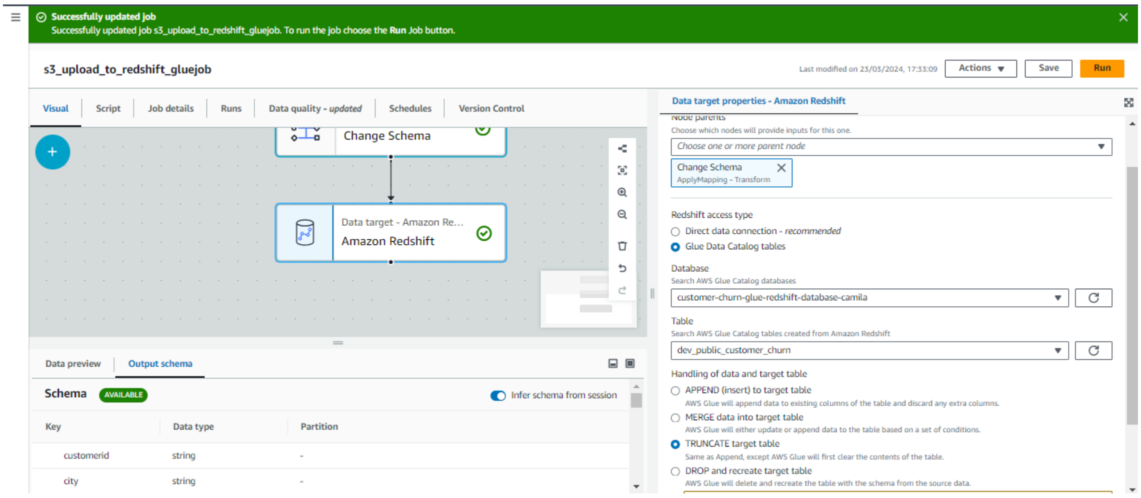
In the S3 bucket settings, they are on the right. I named it and could indicate the location of S3, but as I had created a "Data Catalog table" I directed it to this table.



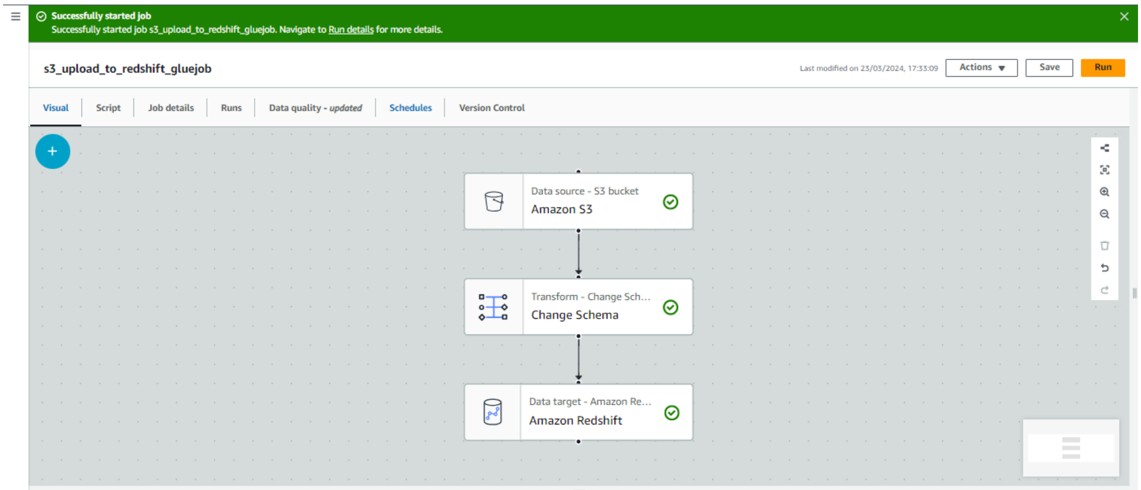
Database transformation, the "Change Schema", displays the settings on the right. Here we made adjustments to the raw data that was in S3, dropped and adjusted the column names and types. Redshift doesn't have a float, it has double , pay attention to this difference.



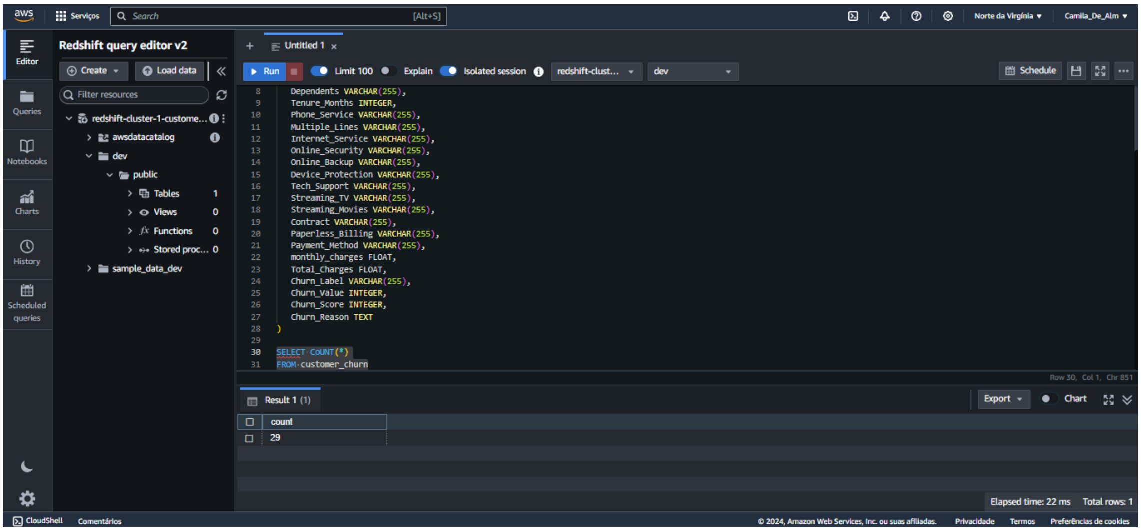
Loading in Redshift, below job settings. I could have made a direct connection, but as I had already created the table, I just directed it to it. I defined that the table would truncate with each new data load, so the data would always be updated with the latest available data, but it would also be possible to just use Append , merge or even recreate the table from scratch.



Result



I manually ran the job to see if it was running in the Runs tab. To be sure, go back to the Redshift editor and run the Count command, the original table was empty, it only had the schema, but now it already has some values.



We can run it manually by clicking Run, or automate it with Airflow, that's why we connected Airflow to VScode, in this connection I ran the following script:

airflow import DAG # Import the **DAG class from the airflow** module. The **DAG** class is used to define and describe a workflow in Airflow.

from datetime import timedelta , datetime

airflow.operators.python import Python Operator #Imports the PythonOperator class from the Airflow operators.python module. The PythonOperator is used to execute Python functions as tasks in a workflow.

from airflow.providers.amazon.aws.hooks.base_aws import AWSGenericHook #Imports the AwsGenericHook class from the base_aws module of the Airflow providers.amazon.aws.hooks package. This class is used to establish connections to AWS services.

import time

from airflow.providers.amazon.aws.sensors.glue import Sensor GlueJob #This class is used to wait until an AWS Glue job completes before continuing the workflow.

def cola_job_s3_redshift_transfer(job_name , ** kwargs):

session = AwsGenericHook (aws_conn_id ='aws_s3_conn')

#Set the region I was in us-west-2

boto3_session = session.get_session (region_name ='us-west-2')

client = boto3_session.client(' cola ')

client.start_job_run (

task_name = task_name,

)

get_run_id() definition:

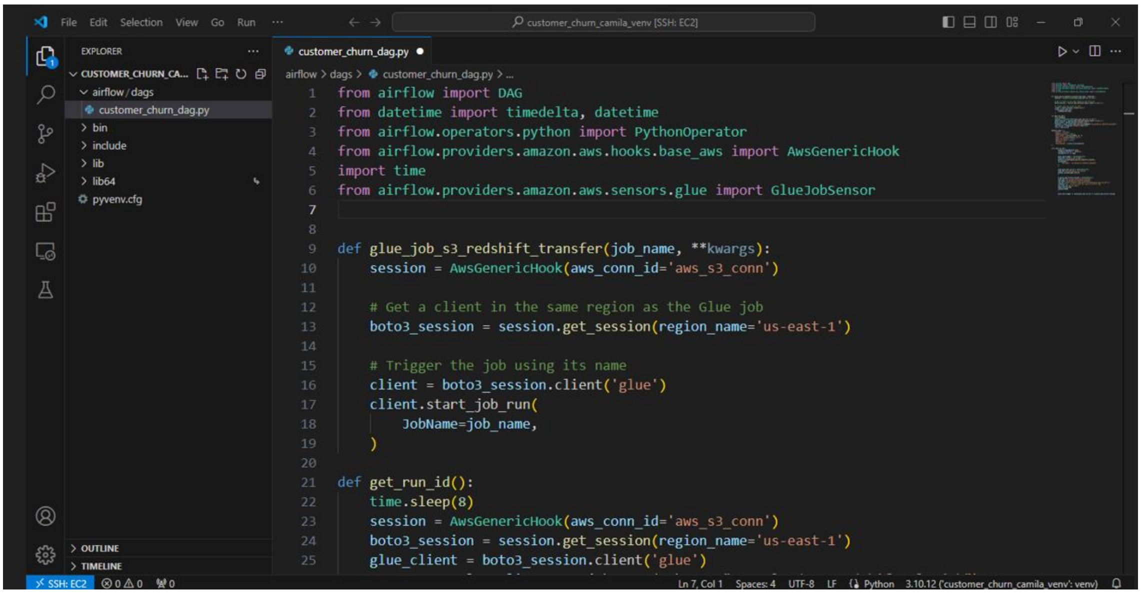
sleeptime (8)

session = AwsGenericHook (aws_conn_id ='aws_s3_conn')

boto3_session = session.get_session (region_name ='us-west-2')

```
glue_client = boto3_session.client(' glue ')
response = glue_client.get_job_runs(JobName="s3_upload_to_redshift_gluejob")
job_run_id = response[" JobRuns "][0]["Id"]
return job_run_id

#Defines a default_args dictionary that contains the default arguments for the DAG, such as the owner, start data, email settings on failure, etc.
default_args = {
'owner': 'airflow',
' depends_on_past ': False,
'start_date': datetime (2023, 8, 1),
'email': ['myemail@domain.com'],
' email_on_failure ': False,
' email_on_retry ': False,
' retries ': 2,
' retry_delay ': timedelta ( seconds =15)
}
# Starts the DAG definition with the name ' my_dag ' and the previously defined default arguments.
with DAG(' my_day ',
default_args = default_args,
scheduling_interval = '@weekly ' ,
ketchup =False) as dag :

glue_job_trigger = PythonOperator (
task_id =' tsk_glue_job_trigger ',
python_callable = cola_job_s3_redshift_transfer,
op_kwargs ={
'job_name': 's3_upload_to_redshift_gluejob'
},
)
grab_glue_job_run_id = PythonOperator (
task_id =' tsk_grab_glue_job_run_id ',
python_callable = get_run_id ,
)

is_glue_job_finish_running = GlueJobSensor (
```

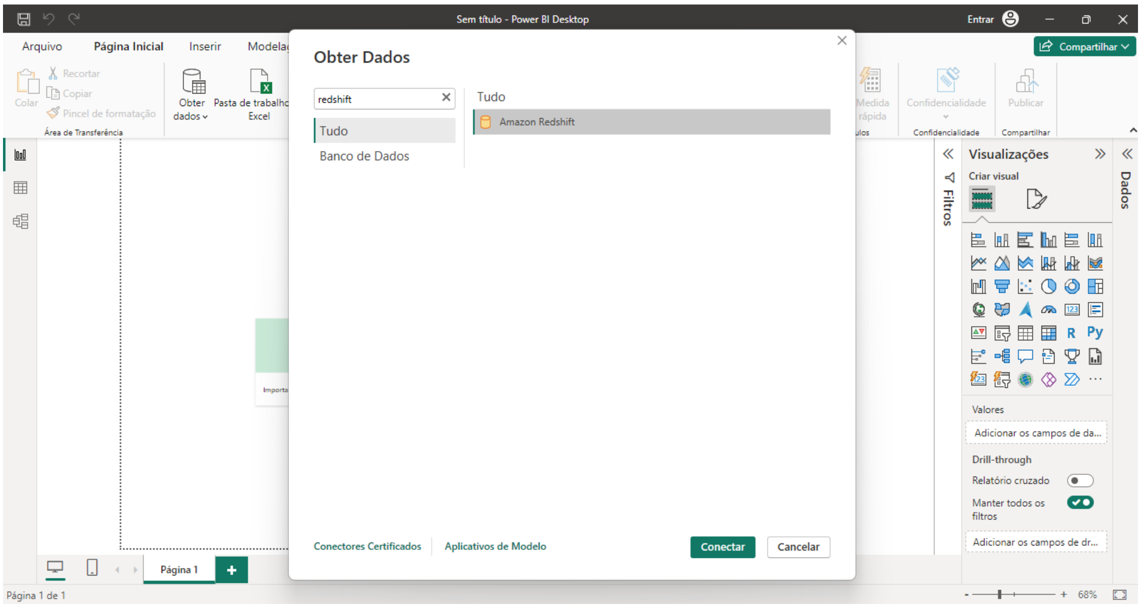
```
task_id =" tsk_is_glue_job_finish_running ",
job_name ='s3_upload_to_redshift_gluejob',
run_id='{{task_instance.xcom_pull("tsk_grab_glue_job_run_id")}}',
verbose = True , # print pastes job records into airflow records
aws_conn_id ='aws_s3_conn',
poke_interval =60,
timeout = 3600,
)

glue_job_trigger >> grab_glue_job_run_id >> is_glue_job_finish_running
```



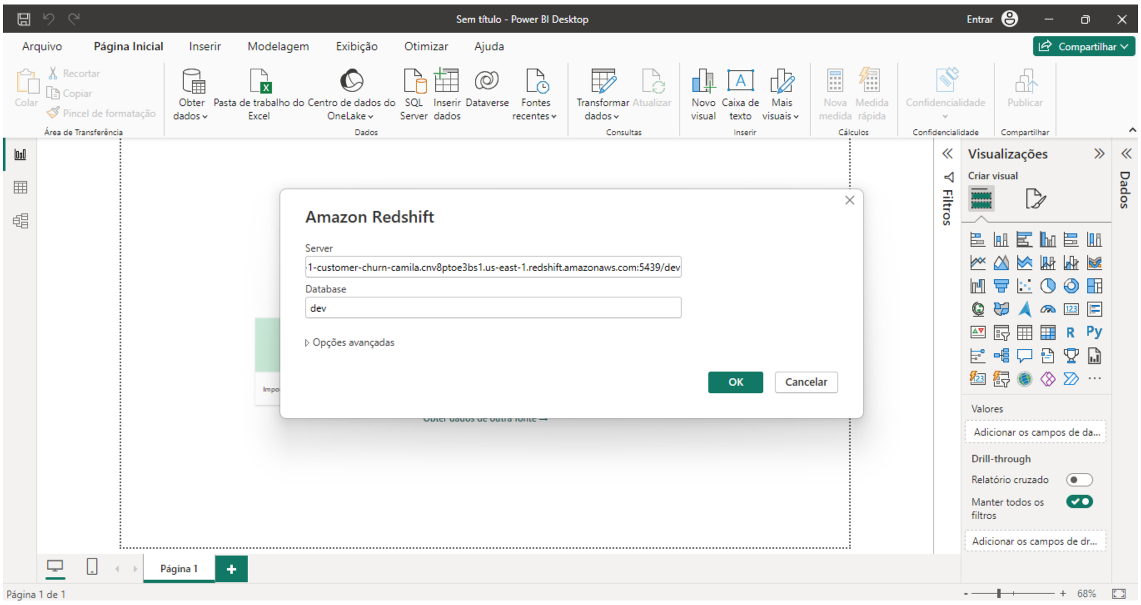
Afterwards, we must connect Airflow via the Root User's Access Keys to automate the job. This must be done via the Airflow browser for both S3 and Redshift and via the EC2 terminal, accessing the virtual environment again and entering Access Keys. After running again, the job in Crawler

After checking that the status is “Running” in Crawler and Success in Airflow,

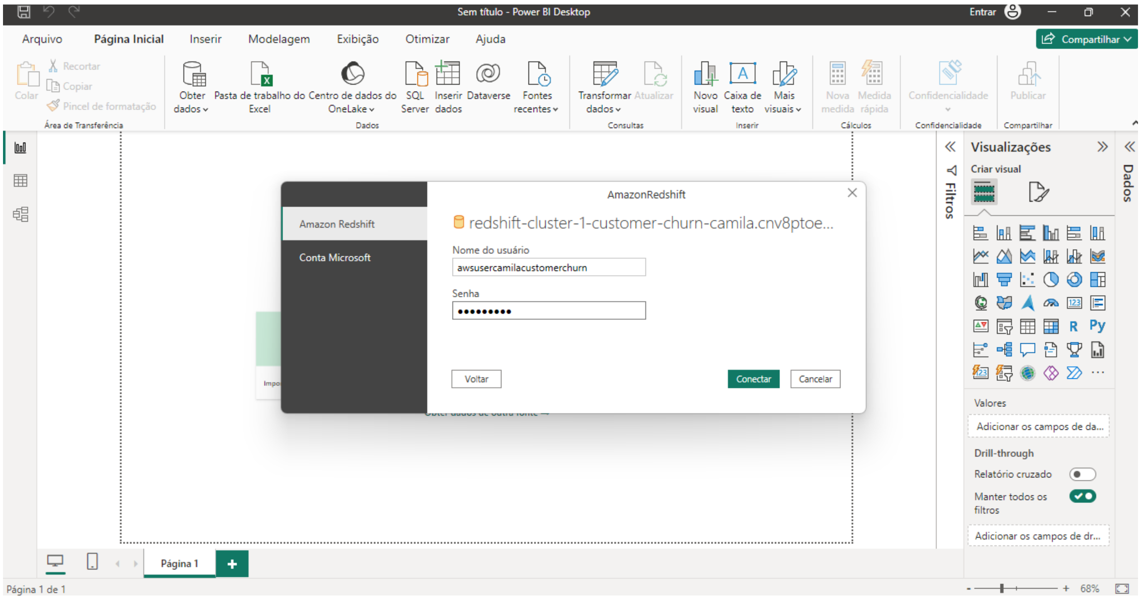
Finally, we will present data visualization in Microsoft Power BI. Connect PowerBI with Redshift don't forget to enable the cluster for connections outside the VPC.



It will ask for the server and the database, the server is the endpoint of the created cluster, and the database is dev, a name that AWS itself suggested.



Enter the cluster credentials.



Afterwards, dashboards will be created, with this table that will be updated in real time with each new load received in the S3 Bucket. Below is a visualization I created.