**ILO (INTERNATIONAL LABOUR ORGANIZATION) -FlashApp -SYSTEM DESIGN (back-end)**

FlashApp is a clown of one production application that I’ve designed in the past. One ILO’s branch (International Labour Organization) needed an application in order to provide some insights. For example : the link between time and business creation, also the link between countries and number of employees.

They needed the application stored in the cloud. I ‘m going to design this app that I called « FlashApp ».

**Functional requirements:**

* Initial Data storage (organization Id, company creation date, company’s name, website, number of employees …).Data are initially strored in a csv file.
* ETL (Extraction-Transform-Load) job along with data analytics should be implemented. The system must:
* extracts the data from the above-mentioned storage.
* transforms/converts the data
* finally load data in a reliable datawarehouse.
* Data analytics.
* A visualization system should provide 2 insights:

1. The companies’s creation over time in a line chart graph
2. The link between countries and employees number in a histogram.

**Non-functional requirements:**

* Daily active users: 10.000
* International users support
* Minimum number of rows in the raw data file: 500.000.
* Scalability. The system should be scalable and able to support a huge amount of requests during the pic period.
* High-Availibilty of the system. It should always be available.
* Data durability.
* Reliability of the system.
* Security (Data encryption both at rest than in transit. Security at database connection level and user’s session. We won’t discuss the session connection process in this article).

We humbly acknowledge that there could be other requirements, but we think the above-mentioned are the main ones.

**First step**: Uploading files from a local computer.

Initially, data are stored in a .csv file in a local server/computer. The client will hit a button in his browser in order to upload the local file. The upload button is linked with a fleet of API servers placed after an Application Load Balancer. A REST API is connected with a presigned URL, that will automatically upload the file from the local computer to a Cloud storage.

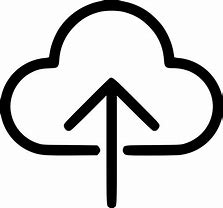
We prefer a REST API because with REST APIs client and server are independent, the data storage along with the UI are independent, they are more scalable and flexible.

We could also use a serverless Faas (Function As A service) like AWS Lambda in order to start the file uploading. Since the previous system requires less efforts and time we choose it.

Client Browser

Load Balancer

 Fleet of API servers (serverless and a throttling limit of 20)



Cloud storage service (scalable, durable, secure and low cost).

## **Fig.1. uploading file system design.**

We will use AWS services like:

* Application load Balancer well known as ALB.
* Amazon API Gateway (as a fleet of API servers), because it is serverless and the application need to scale up automatically, depending with the user’s requests. We will set a throttling limit of 20, so that our application could not struggle due to a huge amount of API requests.
* Finally we will use Amazon S3 as cloud storage for his scalability, durability, security, low costs and easy manipulation.

Client browser

Instances health check



REST API

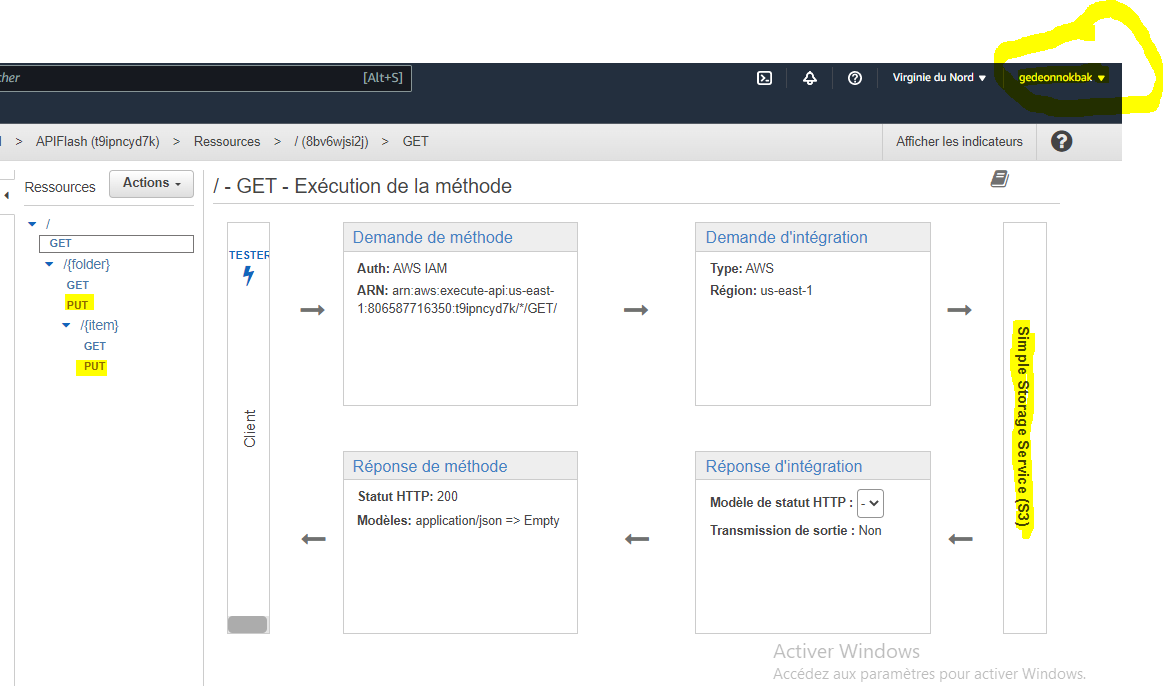


File Upload by presigned URL

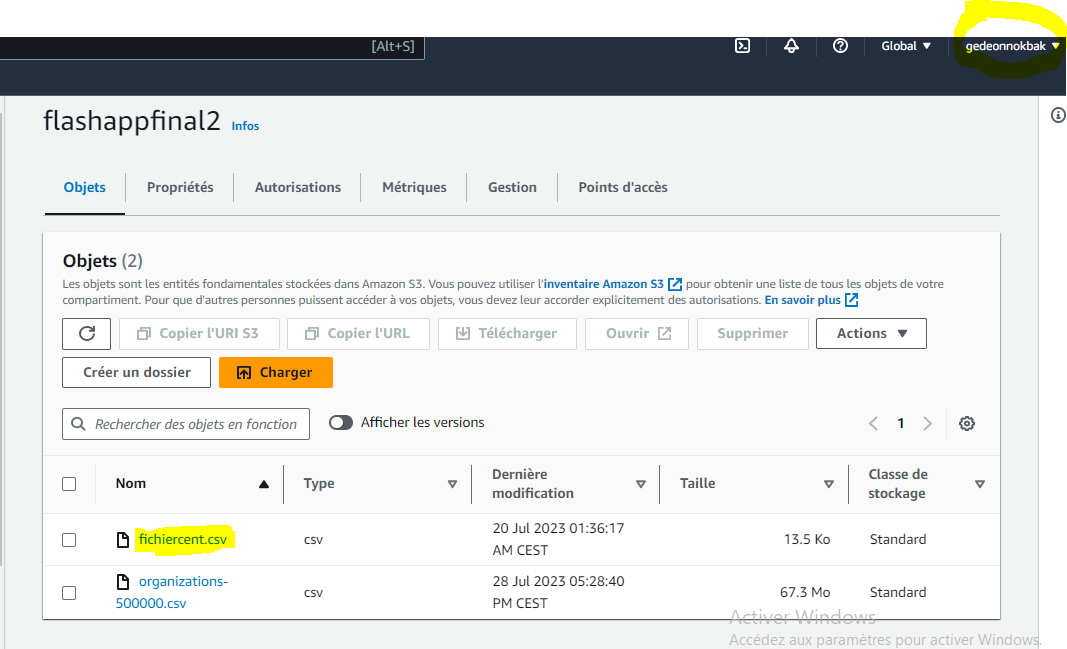


# **Fig.2. Uploading system design with AWS services.**

Following are some captions related with the previous system.



**Fig.3. REST API conception in API Gateway**

****

**Fig.4. The S3 bucket in which the file is uploaded.**

**Step 2 : Extraction – Transform - Load Job (from Cloud Storage to Datawarehouse)**

Now that we have our file stored in a cloud storage, let’s proceed we the ETL job in 3 distinct steps. We will use AWS GLUE as the ETL service performer because It’s serverless, low cost, easy to use and easy to connect with S3.

1. **Extraction**

The following schema could easily explain what the extraction process is.

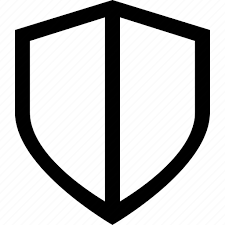
**    **

**S3 endpoint first Glue crawler intermediate Glue database Glue data catalog**

****

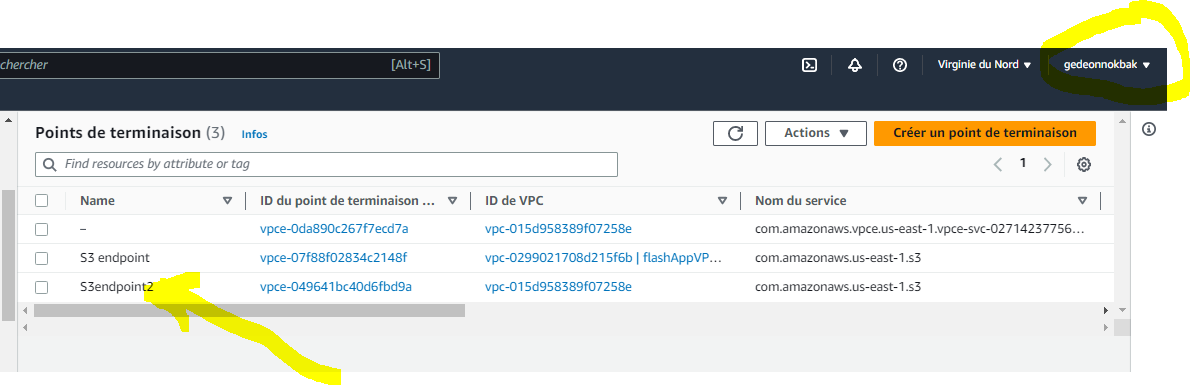
**VPC**

****

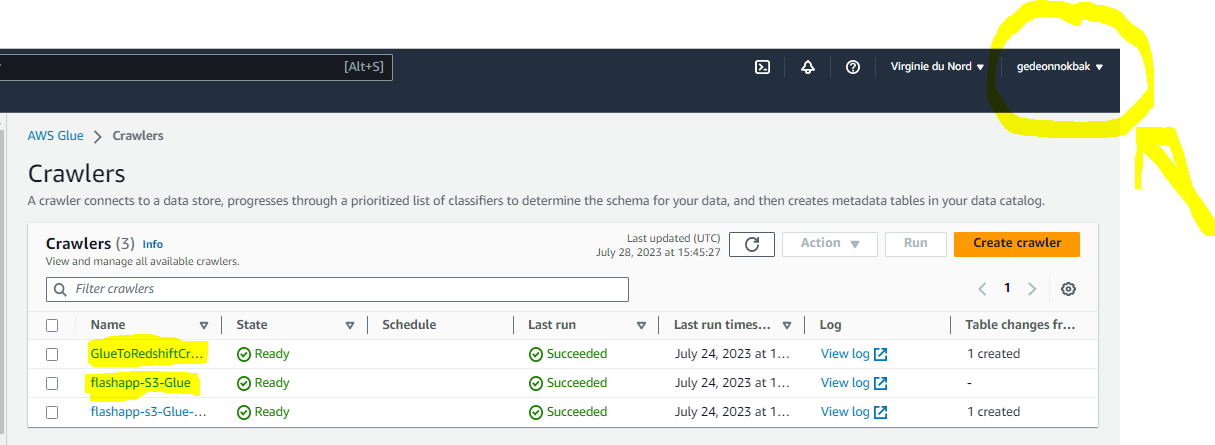
**  **

**Redshift cluster Security group**

As we could see, first, we should crawl data from our S3 bucket, the goal is to capture the data source schema in order to reproduce it in a glue catalog table. The table itself is stored in a temporary glue database. It is very important to set an S3 endpoint so that the process should be secure. Some times, the crawler process can fail if there is no endpoint between the S3 bucket and the data catalog. Here is the process in the AWS console.

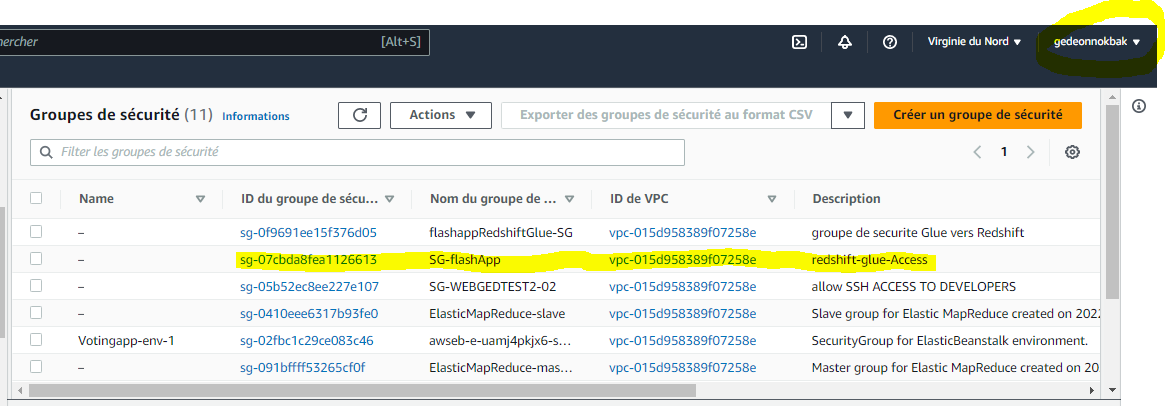


**Fig.5. S3 endpoint created**

****

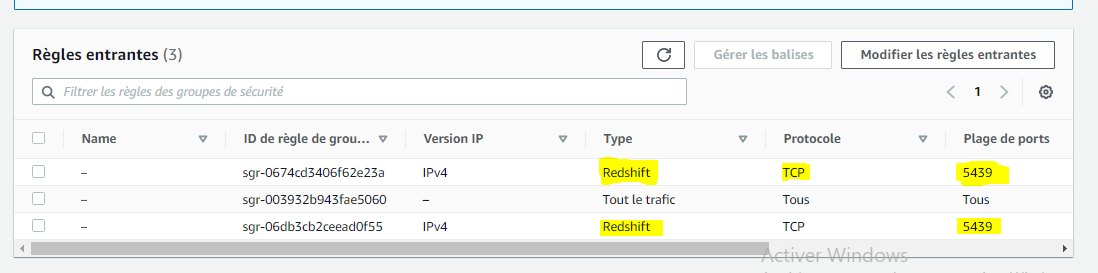
**Fig.6. first crawler created**.

As we can see in the caption the first crawler flashapp-S3-glue is successfully created. Now let’s create the second crawler, the more difficult one. It’s mandatory to create a VPC endpoint and to modify the security group’s inbound rules so that the connection with redshift be secure. The following captions show how to manage this.



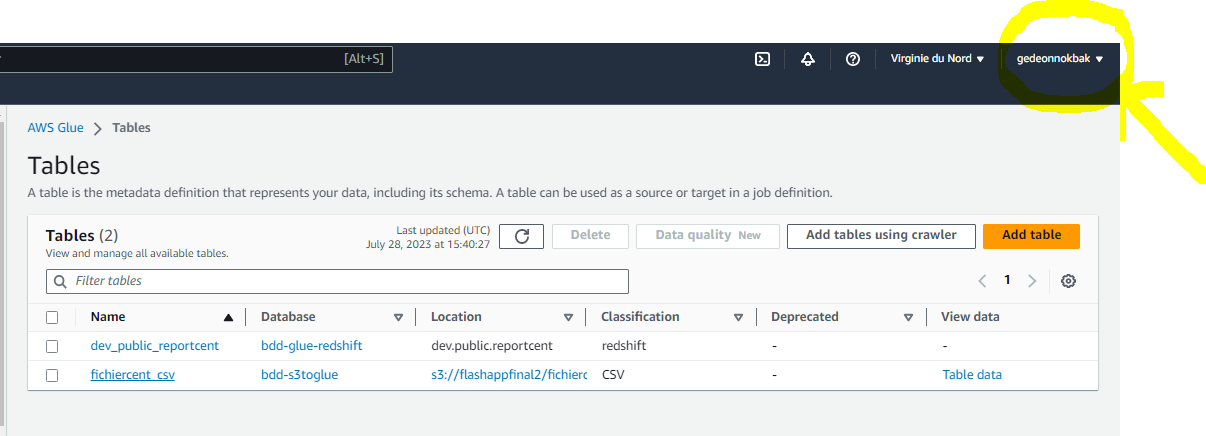
**Fig.6. security group SG-flashApp in the VPC**

Let’s define our inbound rules :



**Fig.7. security group SG-flashApp’s inbound rules modification**

Now we can easily crawl our redshift database table created before. The redshift table schema is reproduced in the Glue data catalog. So now we have 2 different tables in our data catalog. Following is the caption :



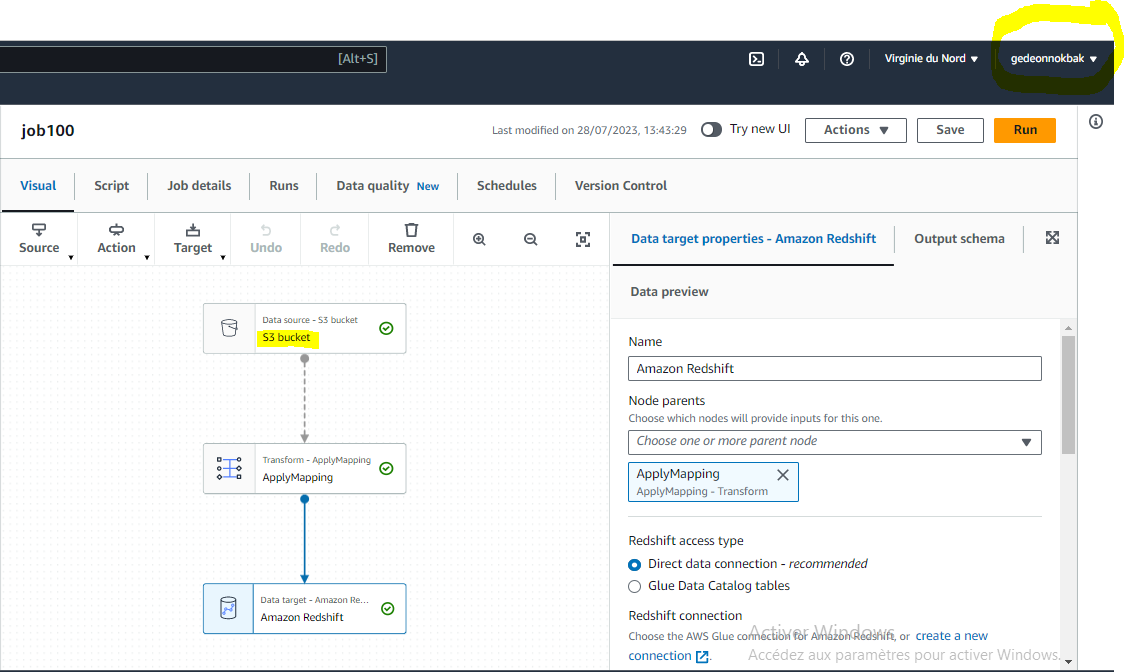
**Fig.8. Tables created**

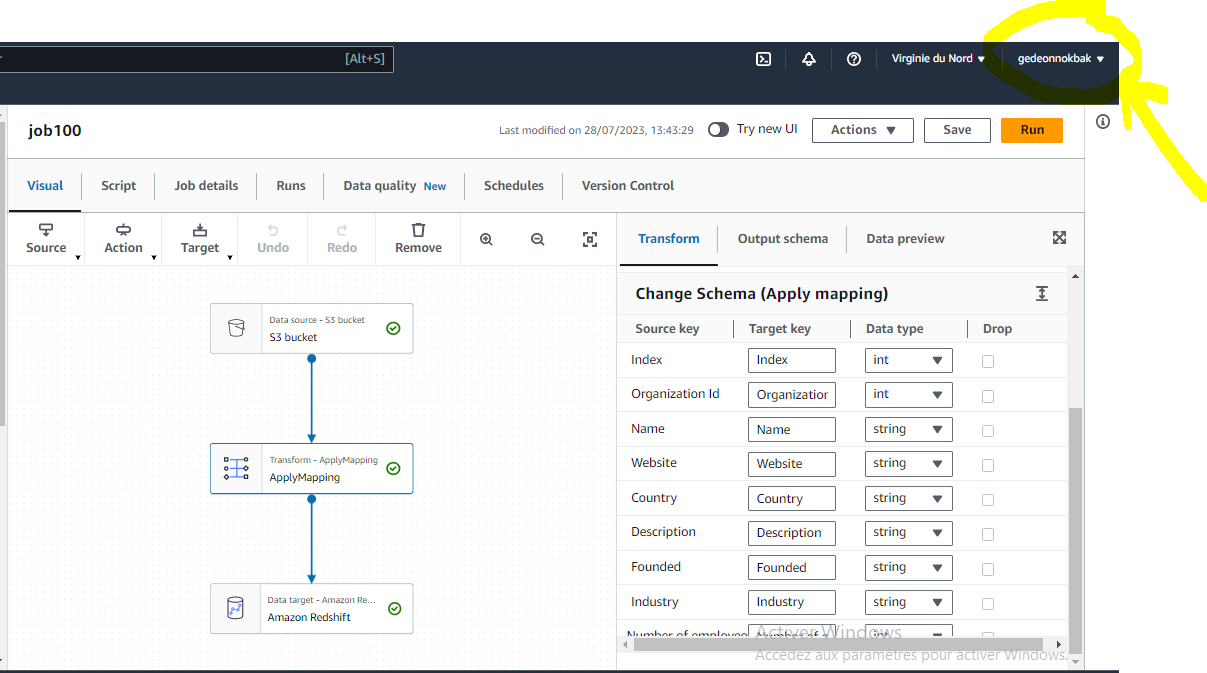
1. **Transformation module**

Right now, we can process with the transformation module. In order to perform this task, let’s create a job, which will :

1. Take the data in the source.
2. Convert the file in PARQUET file
3. Map the data column from the source to the redshift target.
4. Load data in a redshift table.

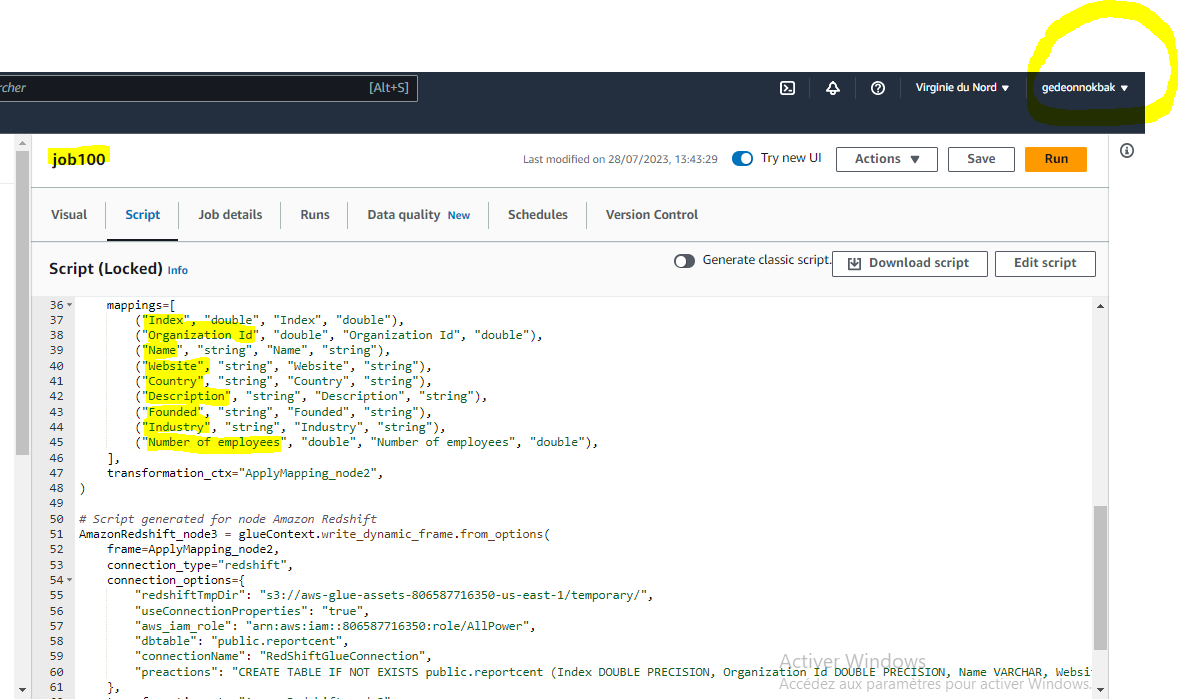
The following caption shows all the process :



****

**Fig.9. Job creation and Mapping process.**

We can see below the job program in python 3.9 :



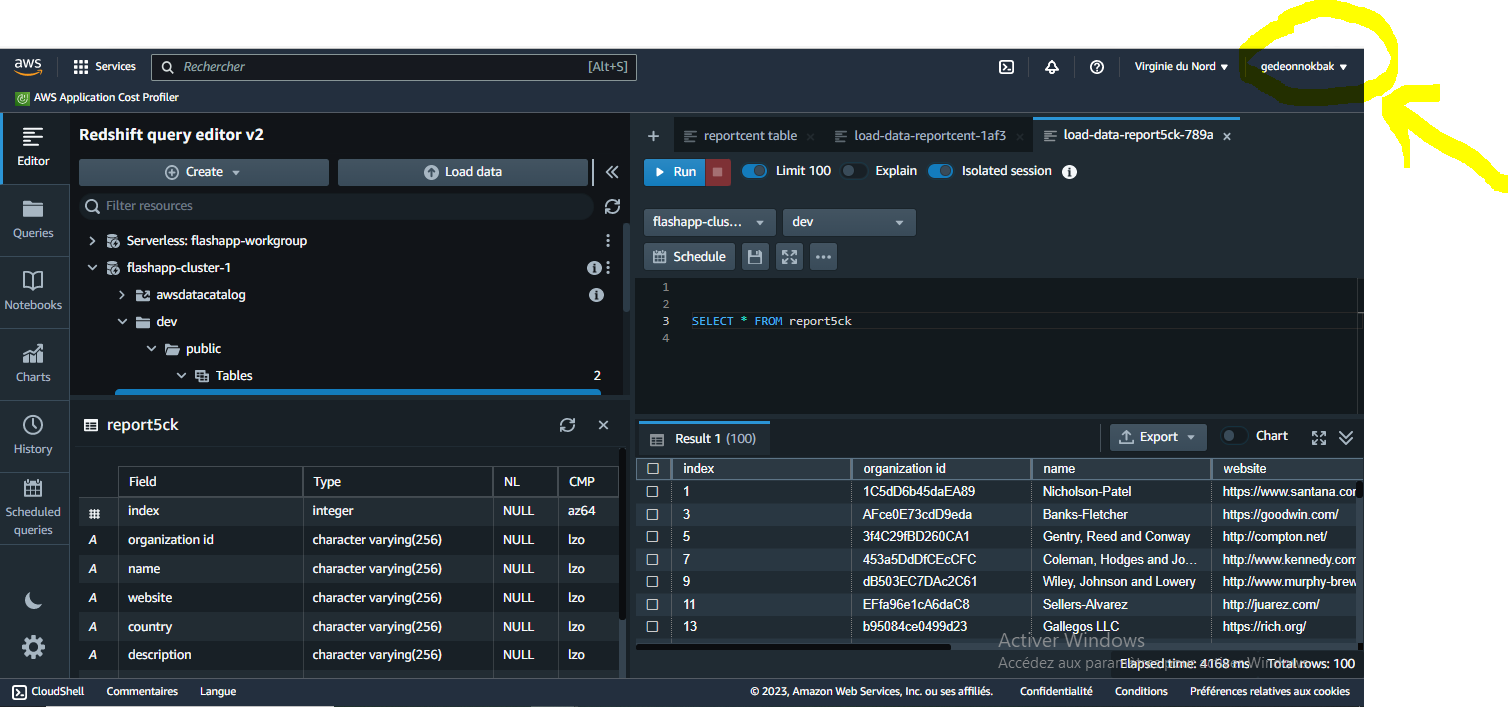
**Fig.10. ETL job program**

1. **Data downloaded in Redshift**

Once the job is succeeded, the download process will start and will download 500.000 rows of data in a redshift table . We’ve created a cluster(flashApp-cluster-1), database (dev) and table. All the data are downloaded in the public schema of the DB.

Why do we use Redshift to perform this task ? For his high speed processing time for queries, high performance, horizontal scalability, massive storage capacity.

Below is a caption showing the redshift cluster, the DB and table :



**Fig.11. Data downloaded into Redshift**

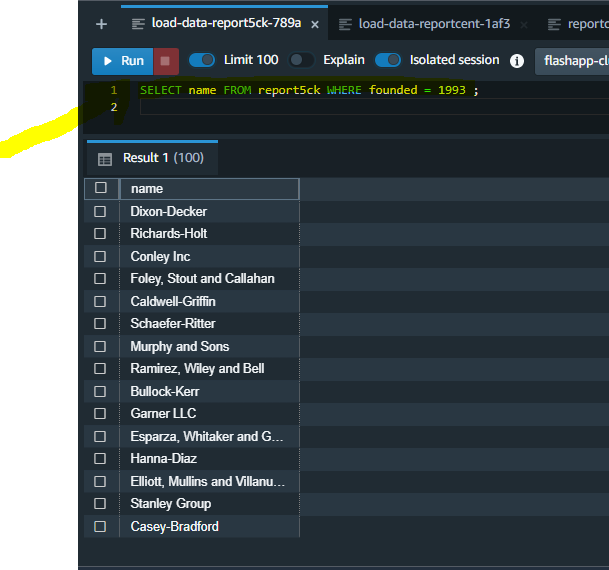
**Step3 : data Analytics in Redshift**

You can see below some operations that I made in the database.

* Let’s find all the companies founded in 1993. The SQL query is :

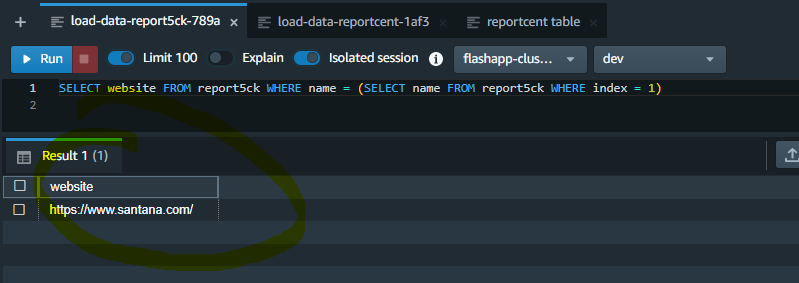
SELECT name FROM report5ck WHERE founded = 1993 ;

Below the result :



* Let’s find the website of the first company recorded. The SQL query is :

SELECT website FROM report5ck WHERE name = (SELECT name FROM report5ck WHERE index = 1 ) ;

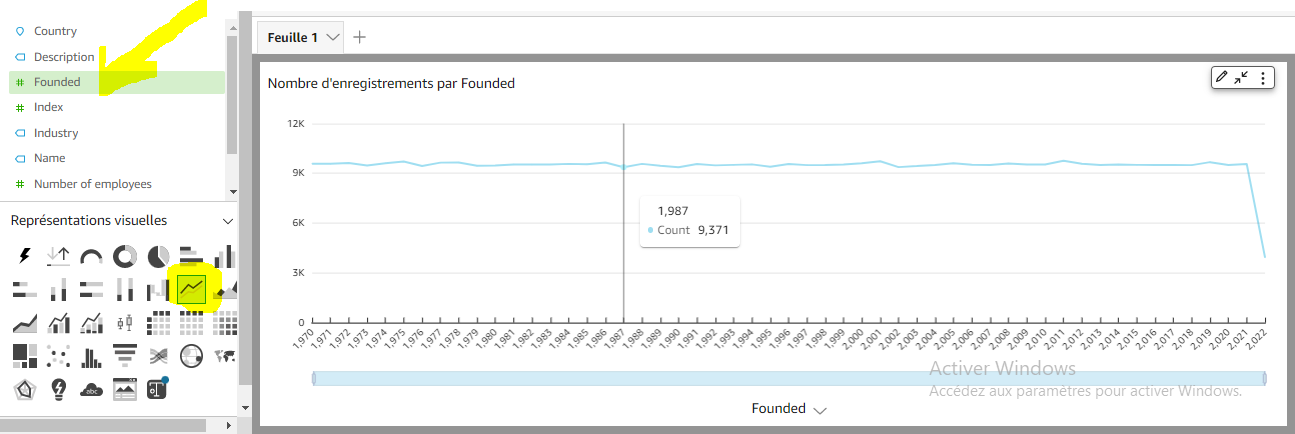


Now we are able to analyse our data as needed, let’s move to the visualization module.

**Step 4 : The visualization module**

You probably remember the goal of the project. It is mentioned in the introduction paragraph : analyse the link between : time and companies creation, and also the link between countries and number of employees. We will use Amazon Quicksight in order to visualize these insights.

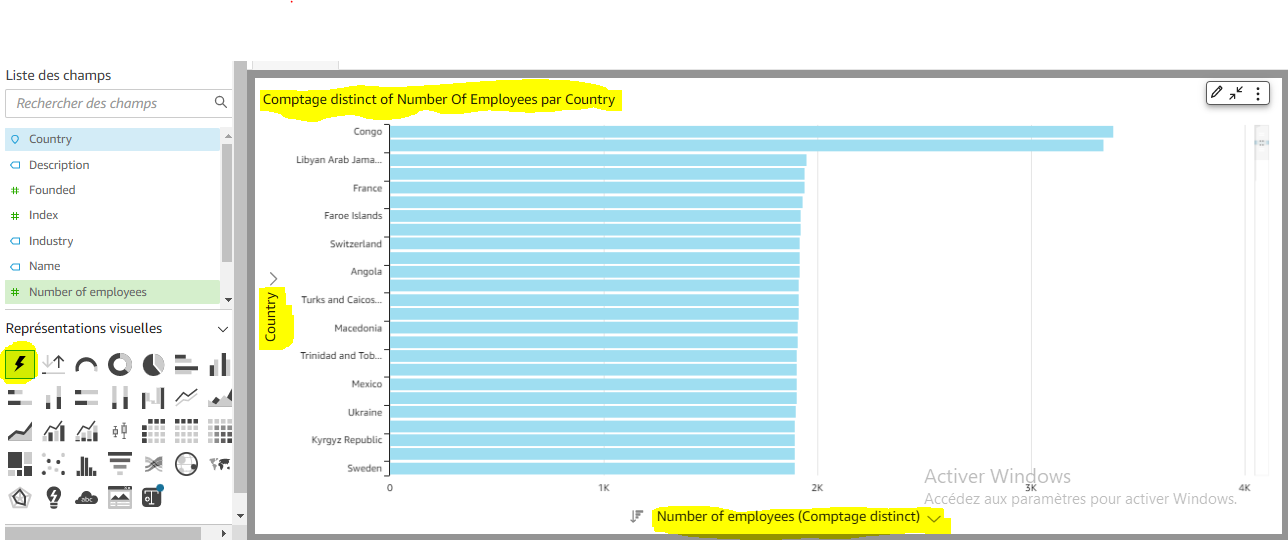
Let’s move to the Amazon Quicksight console, and establish a connection between Quicksight and redshift. The diagram choice is really mandatory. The first diagram should be a line chart, because it’s a time visualization. The column concerned is « founded ». Below, you can see the diagram provided by Quicksight :



**Fig.12. Entreprises creation over time.**

We can easily see that, the years 2001 and 2011, there was a maximum number of companies founded. Since 2020, the company number decreased, probably due to covid-19 pandemic. We’ve just achieved our first goal.

Let’s now find what is the link between countries and employees number. We use the autograph functionality of Quicksight so that the system could automatically determine what is the suitable kind of diagram in connection with this problem. Let’s design it:



**Fig.13. Distribution of the number of employees by country.**

Here we can easily see that, the country using the most important number of employees is Congo. The country using the less important number of employees is sweeden.

We’ve just achieved our second goal.

In sumary, we can provide the final system design for this app.

Quicksight

IAM role IAM role IAM role IAM role

**Fig.14. FlashApp final system design**

How can we improve our system ?

It could be interesting to set a check-job module. The goal is to check wether crawlers or glue job are effectively started and completed. In order to achieve this goal we can use Amazon cloudwatch as a watchdog. As soon the glue job is completed, Cloudwatch will trigger a lambda function which in turns will start the transformation process.

We could also connect the entry system with a streaming record tool like Amazon Kinesis Firehose which is S3-native. In that way the system will streamingly send data to a csv file strored in a S3 bucket.

The previous system, is just the back-end implementation. We can create a beautiful front-end UI compatible with Amazon Amplify for example, along with a session controler like Amazon Cognito.

We hope you enjoyed this article.