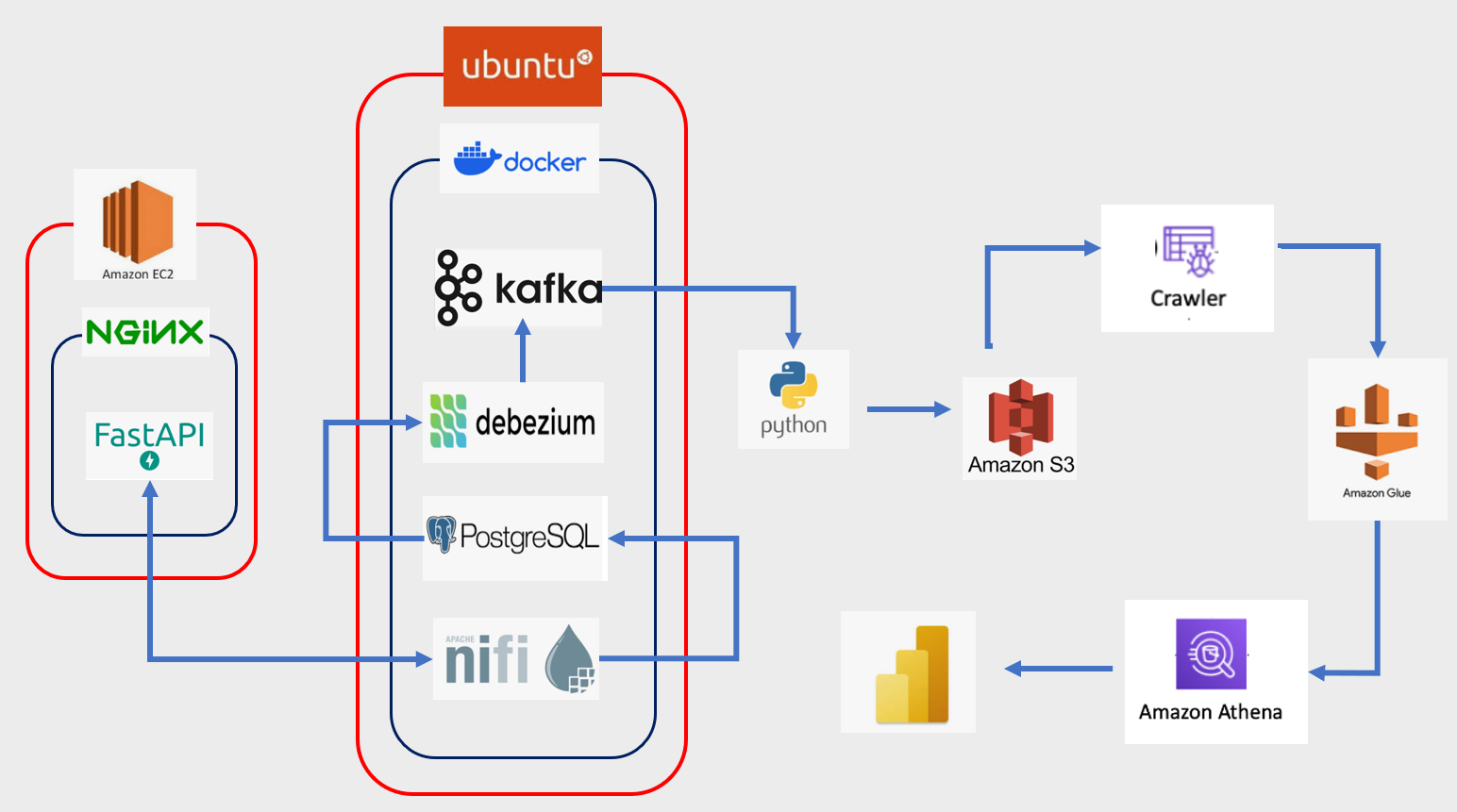
**Projet de streaming de données**

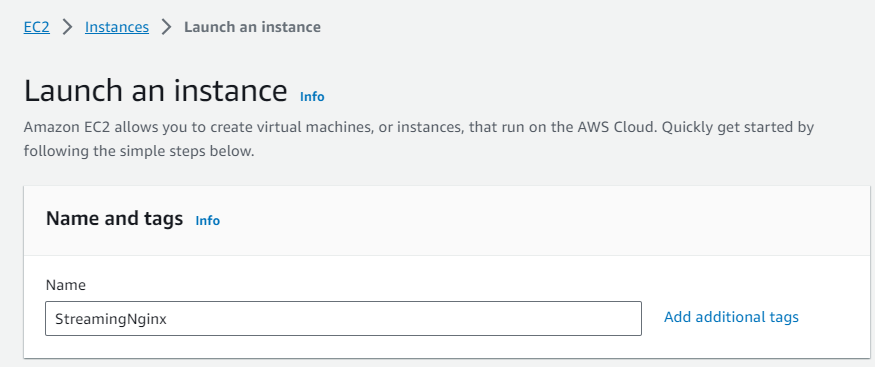
****

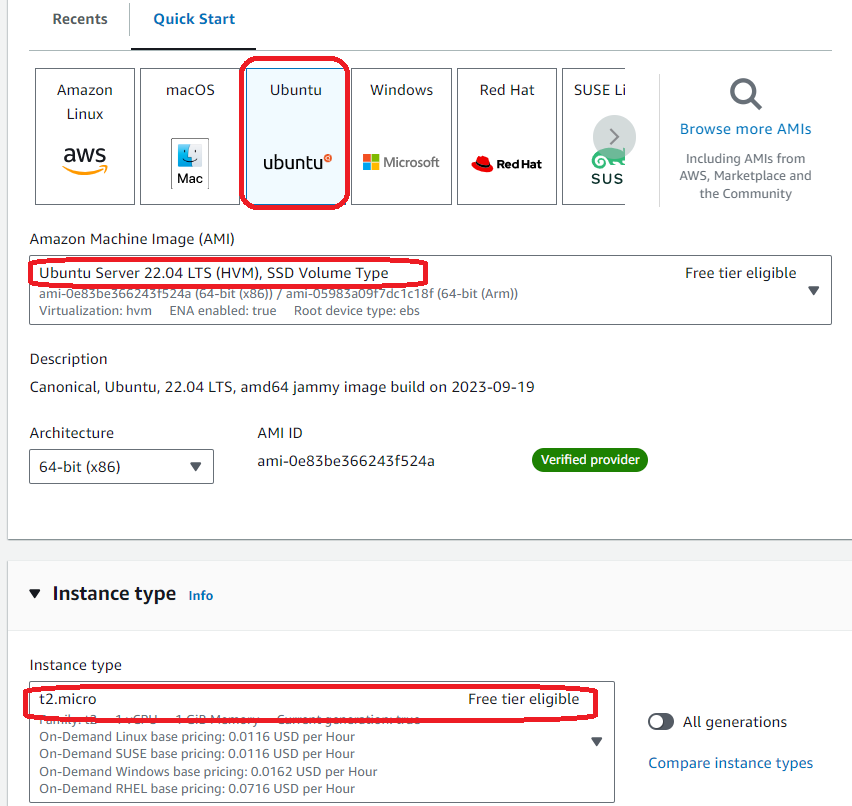
Debezium est une plateforme open source de capture de données de changement (CDC) en temps réel, permettant de surveiller et de capturer les modifications qui se produisent dans une base de données. Elle permet de diffuser en continu des événements de base de données pour être traités par d'autres systèmes.

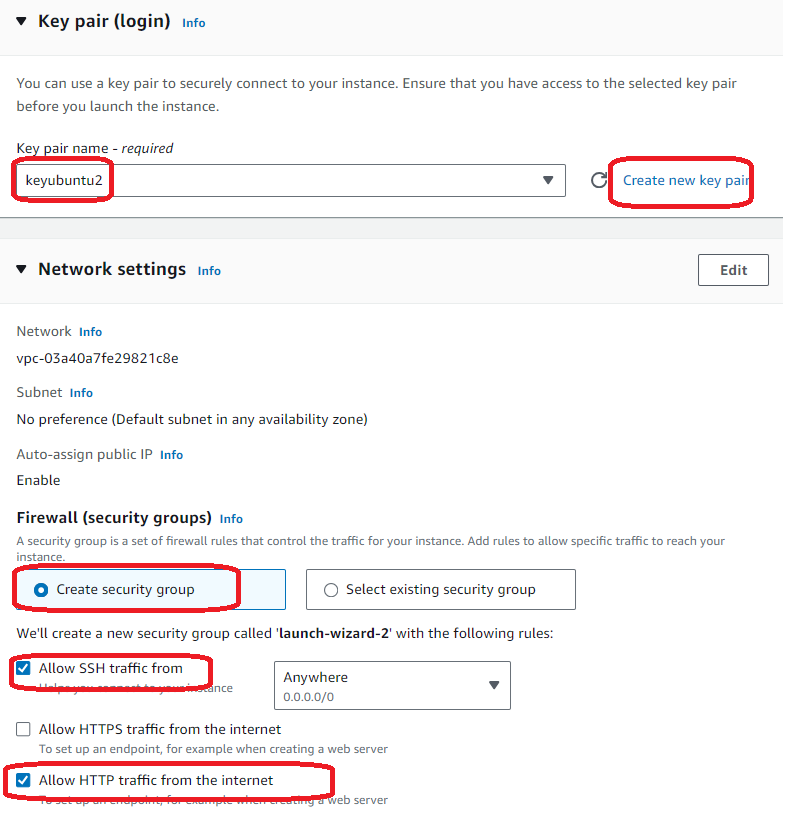
FastAPI est un framework Web open source en Python, reconnu pour sa rapidité, sa génération automatique de documentation, sa validation des données et son support des websockets. Il est utilisé pour créer des API performantes en Python.

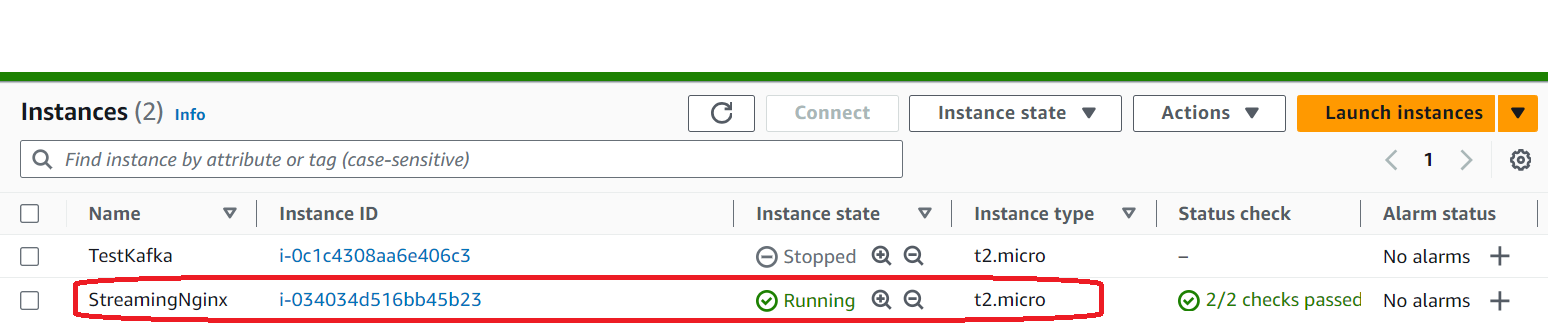
**Installation de FASTPI sur NGINX**

On crée une instance sur EC2 avec Ubuntu pour installer NGINX



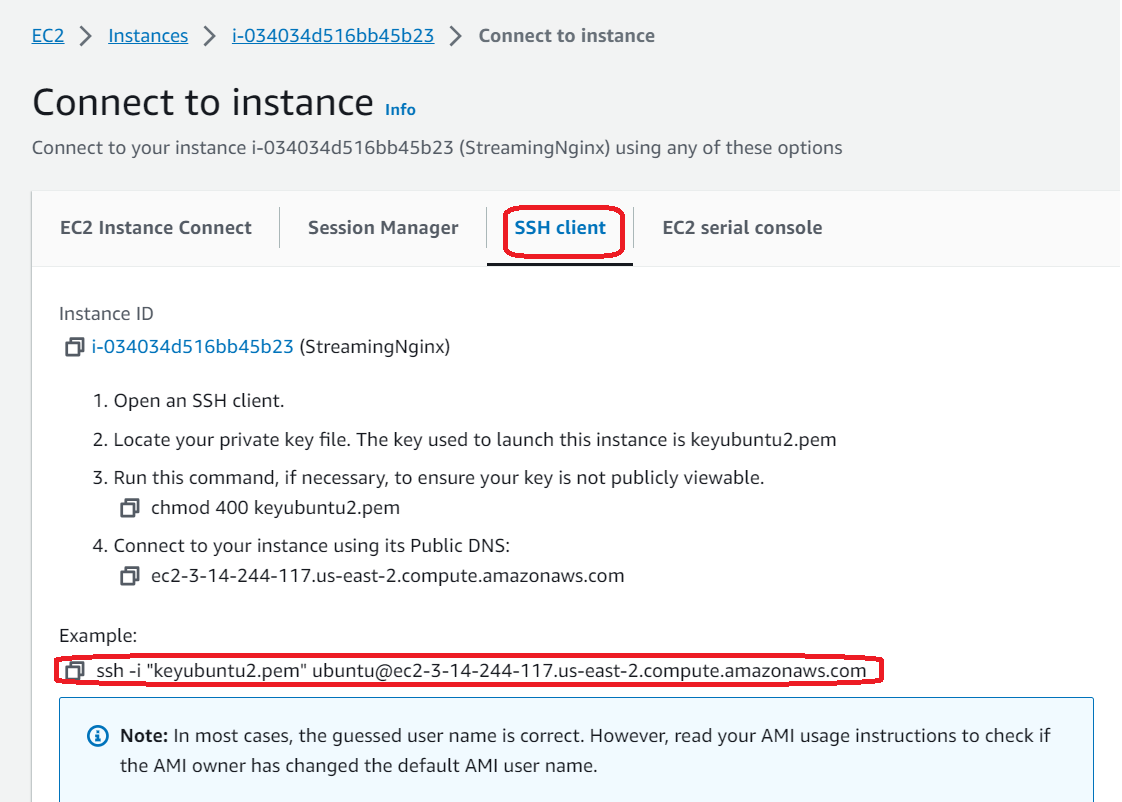


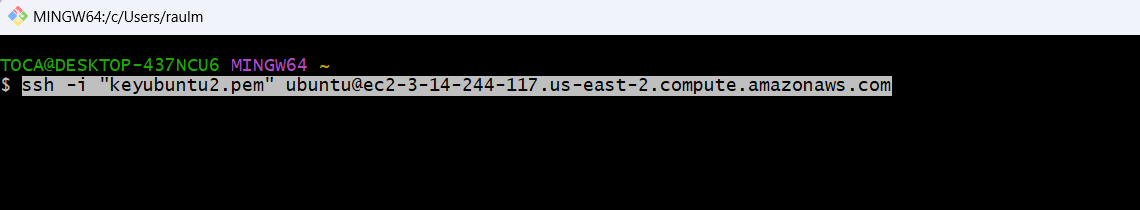




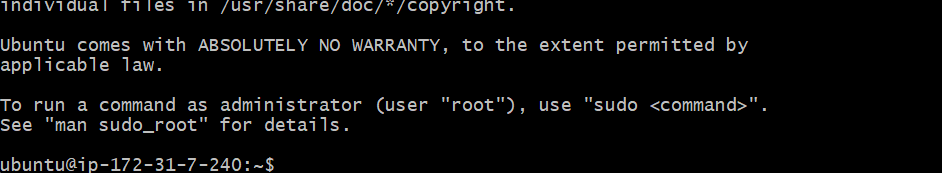
Nous nous connectons à l'instance créée à l'aide d'un terminal.

Nous copions la commande de connexion



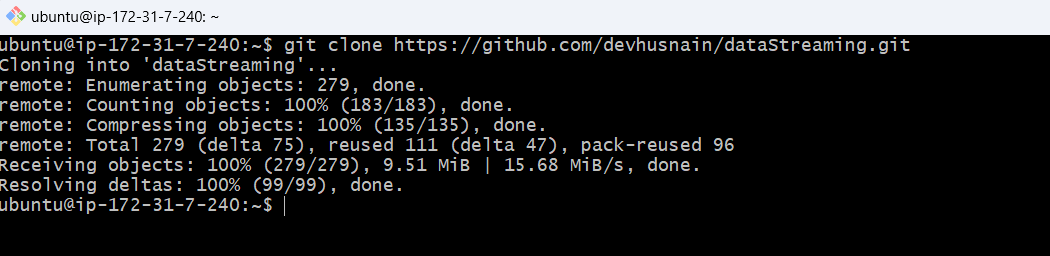


Maintenant nous sommes connectés



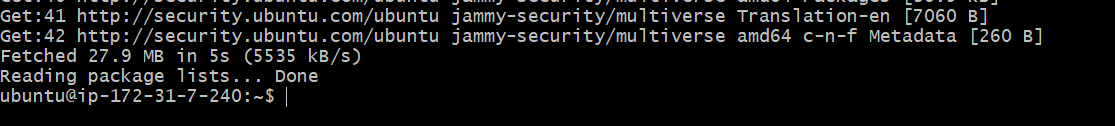
Clonons un dépôt github

git clone <https://github.com/devhusnain/dataStreaming.git>

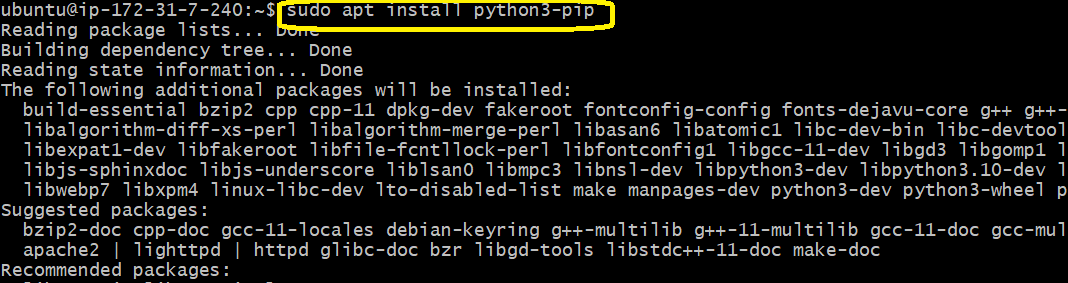


*utilisez ces commandes pour installer python-pip et la configuration requise pour fastapi*

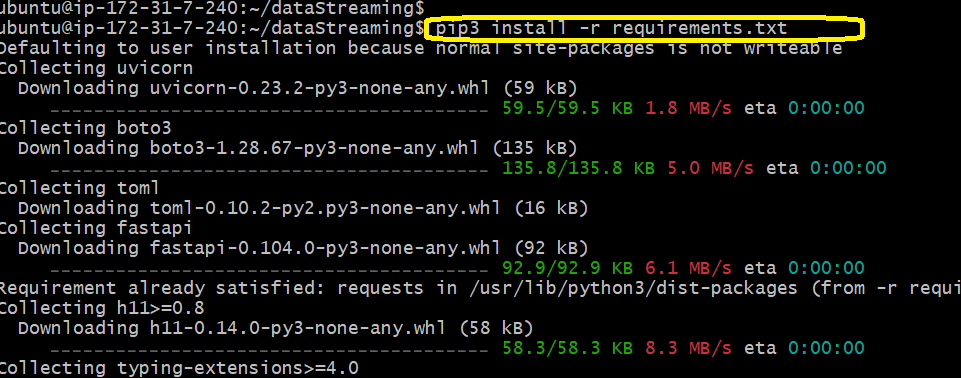
sudo apt-get update



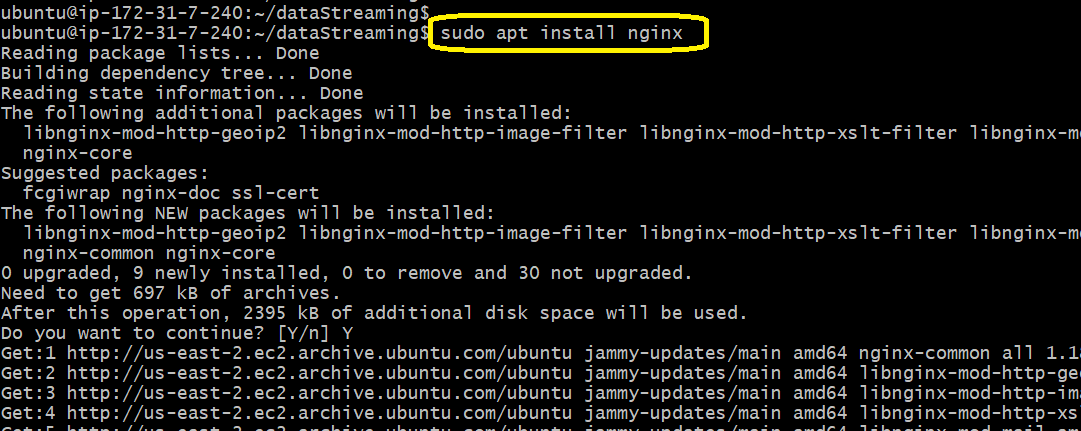
sudo apt install python3-pip



pip3 install -r requirements.txt

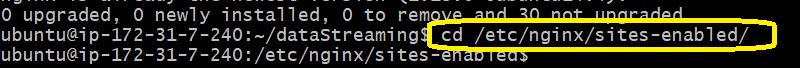


sudo apt install nginx



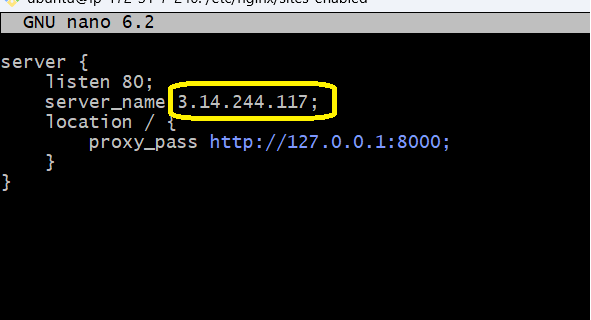
On change de dossier avec la commande

cd /etc/nginx/sites-enabled/



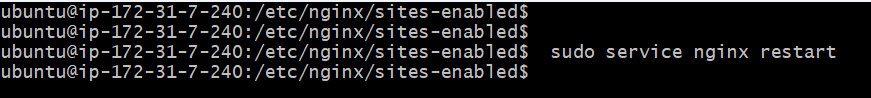
Ensuite on crée le fichier fastapi\_nginx avec l'éditeur nano avec la commande suivante

sudo nano fastapi\_nginx



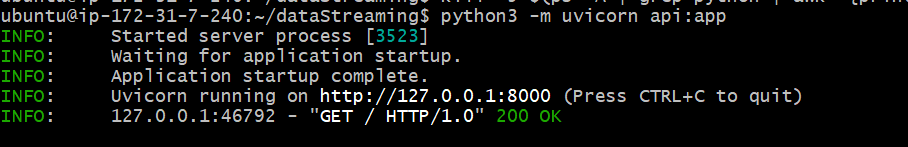
Nous exécutons ce qui suit pour redémarrer le service nginx

sudo service nginx restart

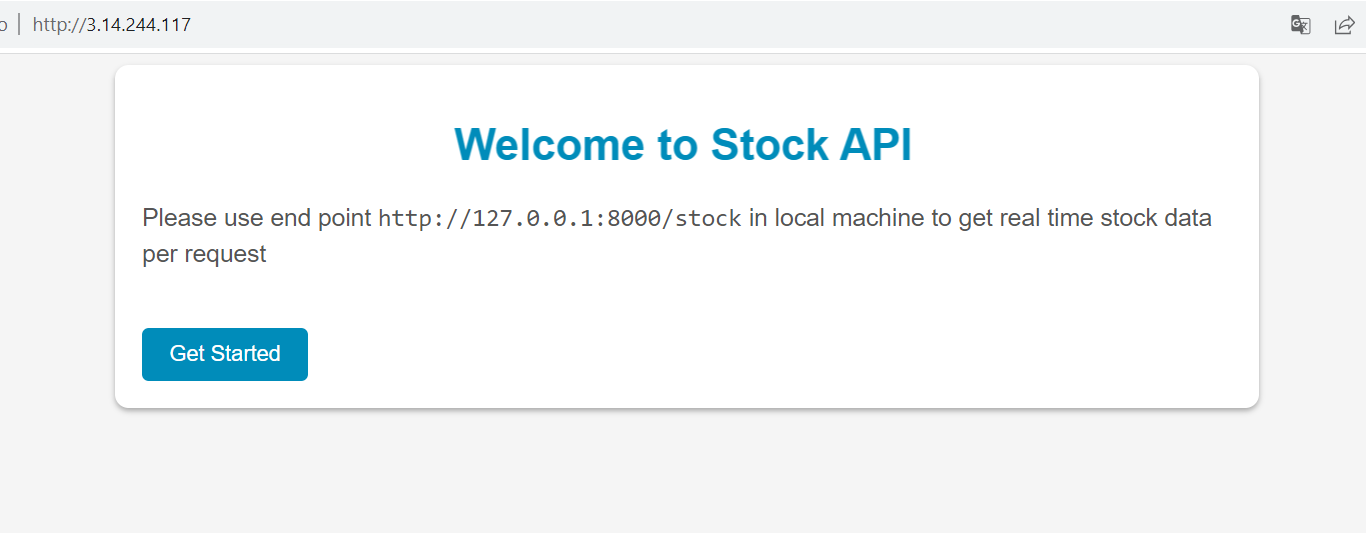


dossier dataStreaming que nous avons cloné depuis github et exécutons le fastapi créé

python3 -m uvicorn api:app



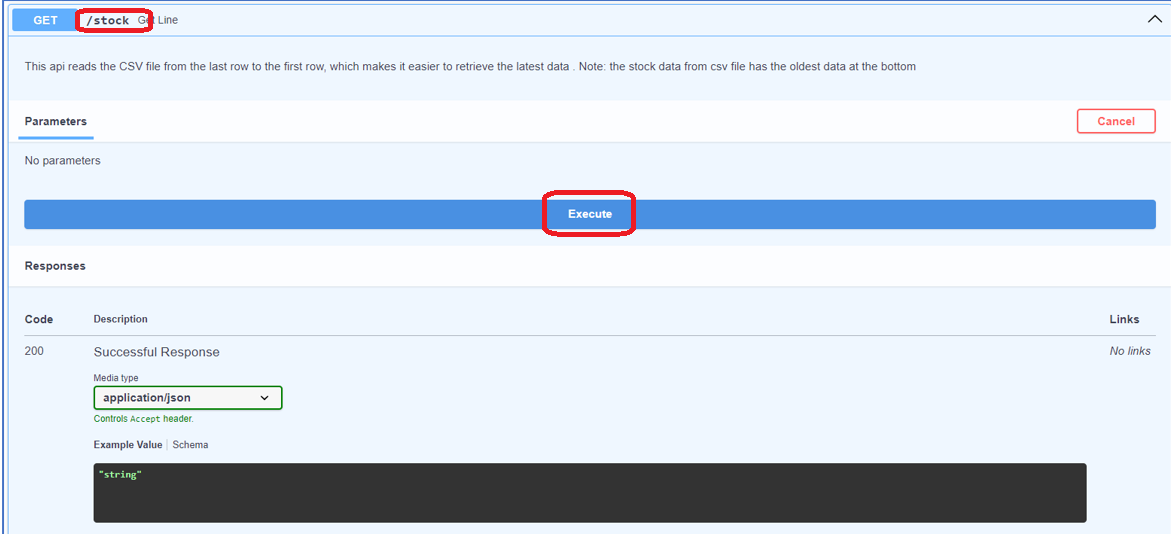
Le service est actif et nous ouvrons l'IP publique de l'instance EC2



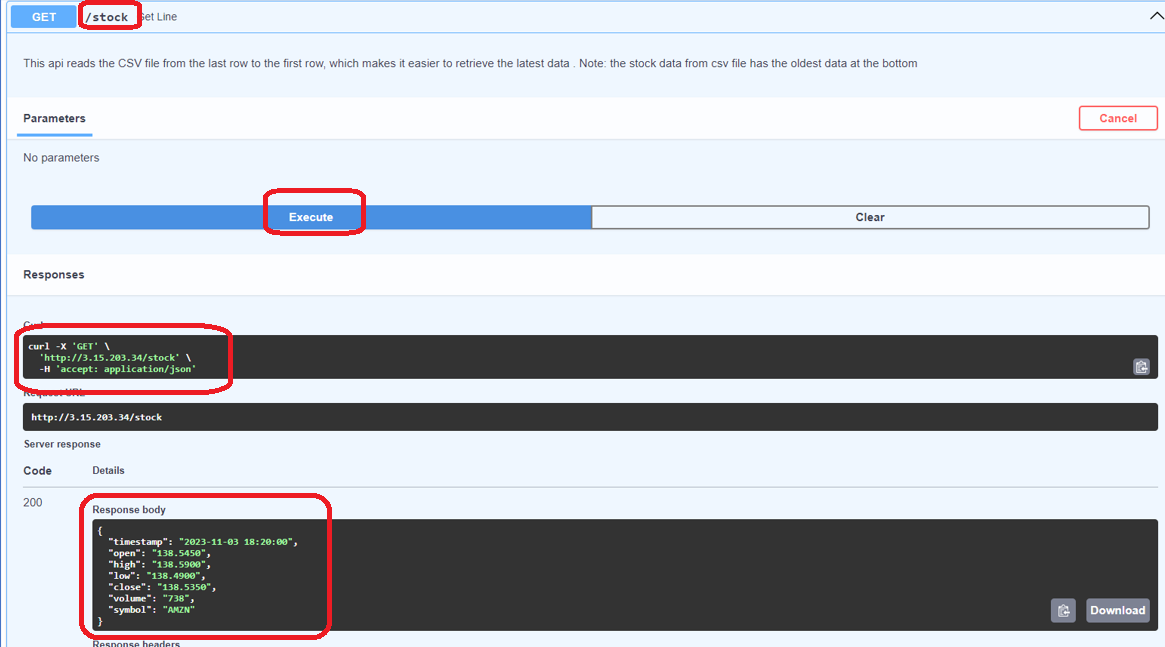
Nous testons les processus FASTAPI



Nous sélectionnons STOCK et exécutons



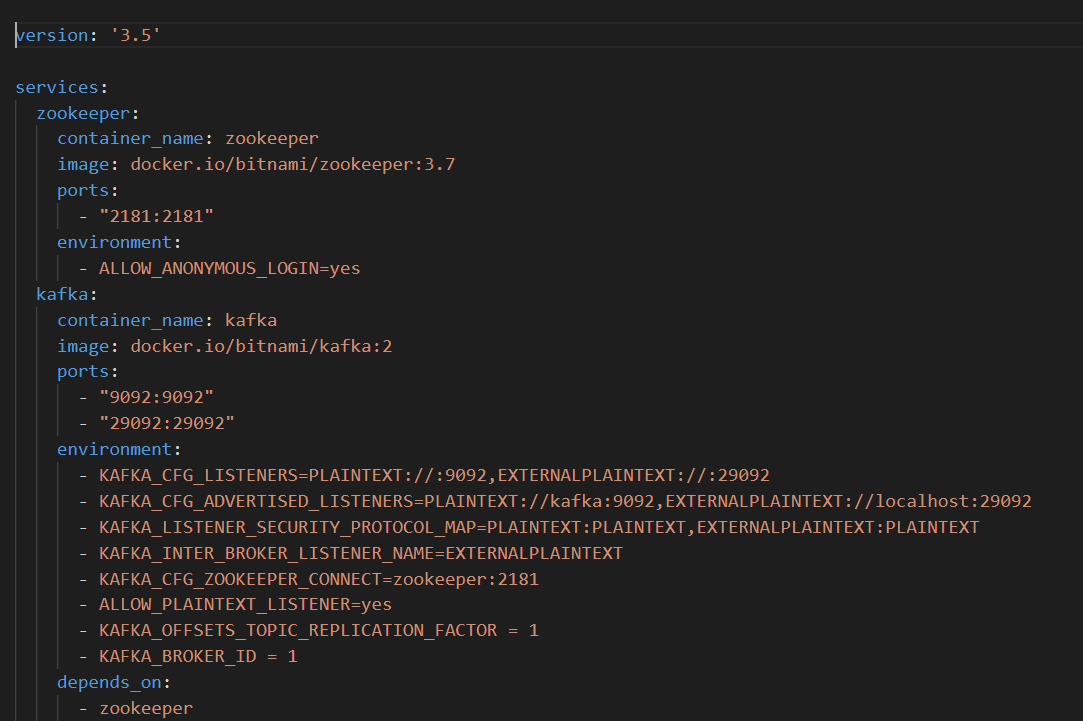
L'API nous montre le résultat au format json

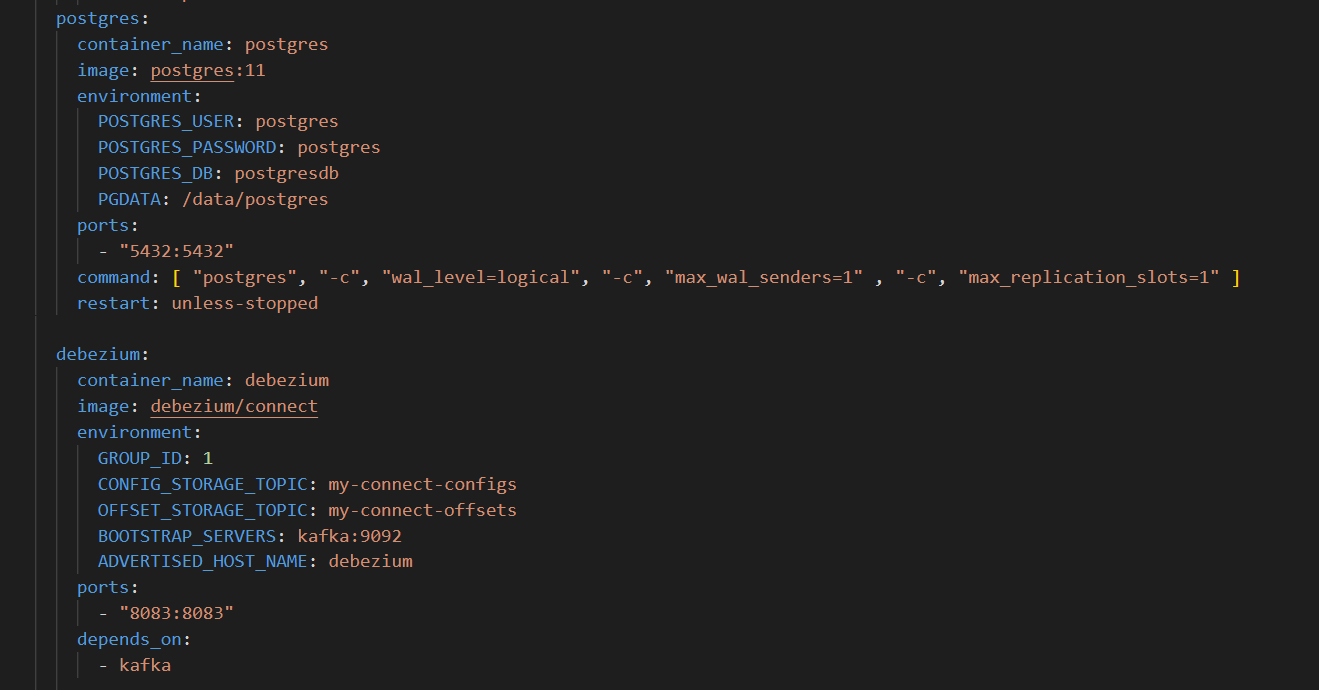


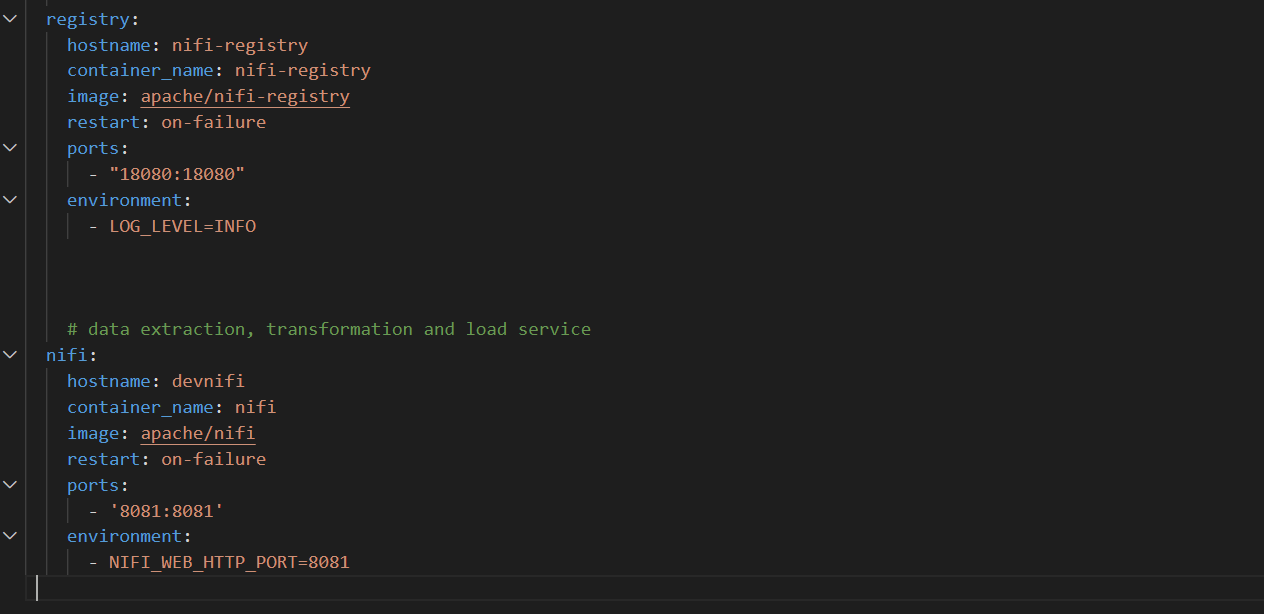


**Installation de NIFI, Debezium , Postgres et Kafka avec Dockers sous Ubuntu dans une machine virtuelle**

Nous allons installer les produits en utilisant Docker compose et pour cela nous avons un fichier Docker- compose.yml

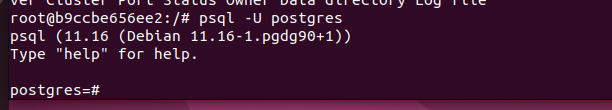




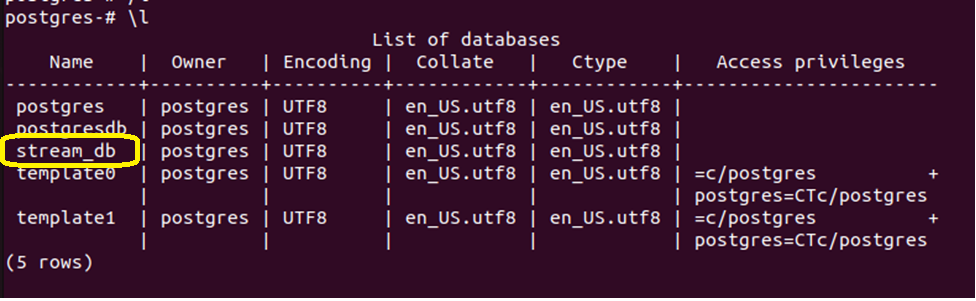


Nous exécutons Docker compose -up et nous aurons Kafka, Debezium , Postgres et Nifi en cours d'exécution

Créons la table stock sur Postgres



Nous listons la base de données

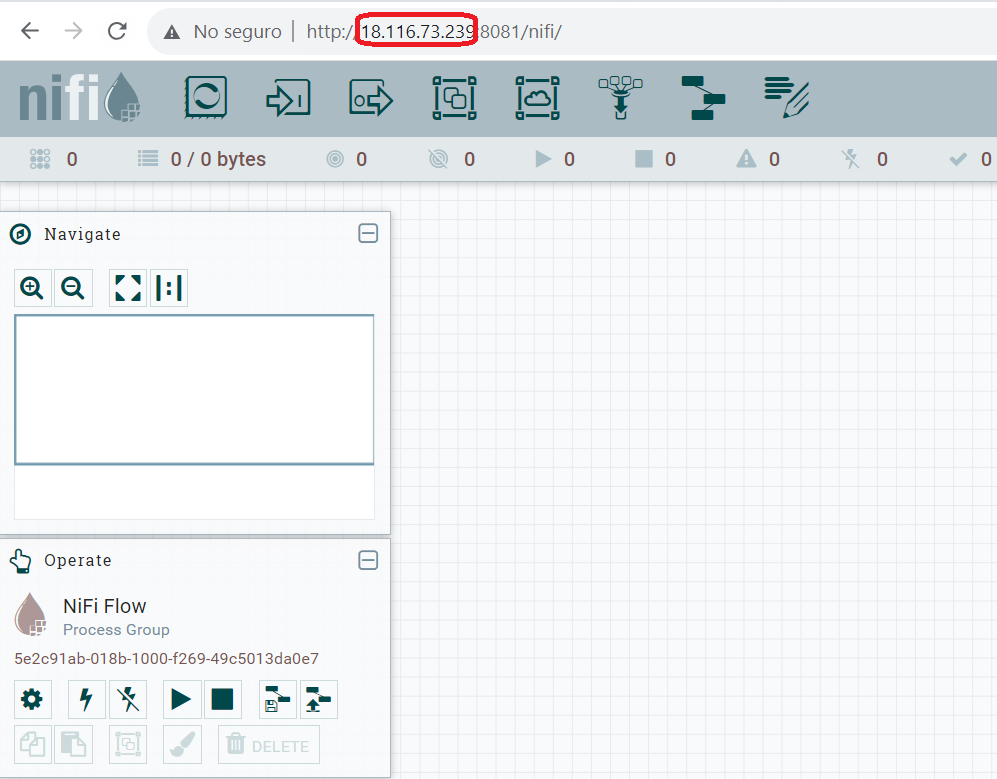


Nous créons la table des stocks



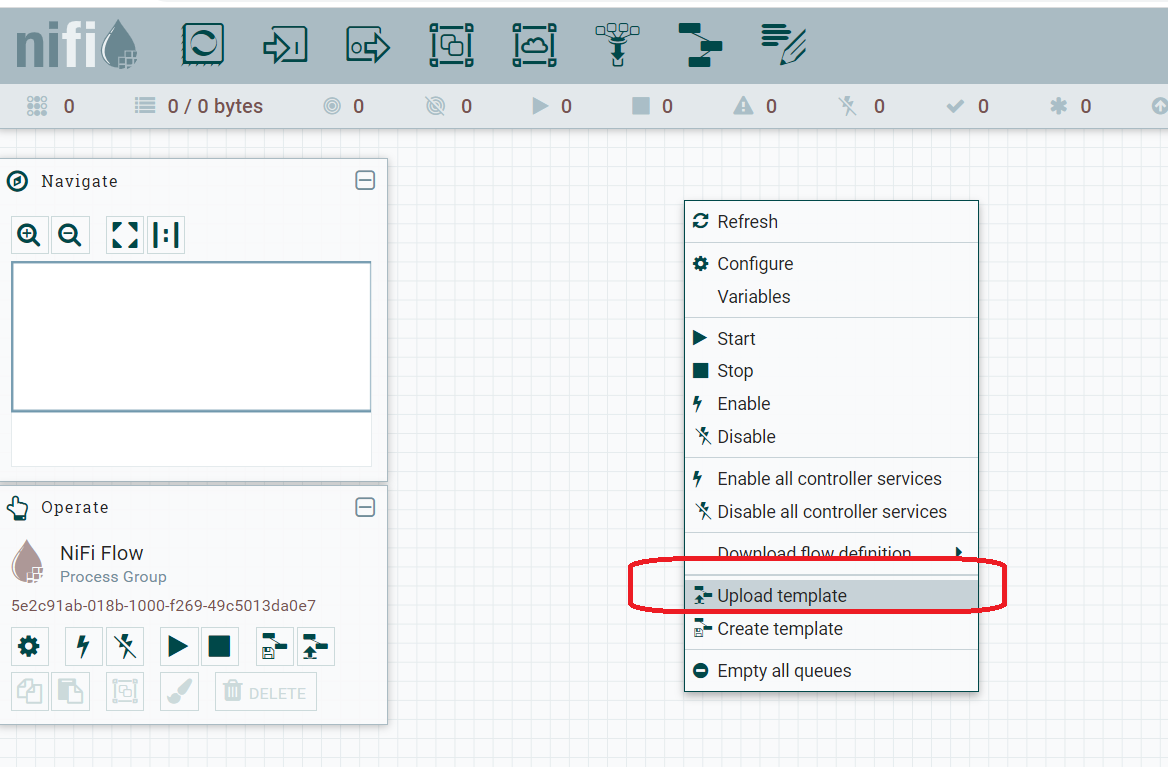
**Essayons Apache NIFI.**

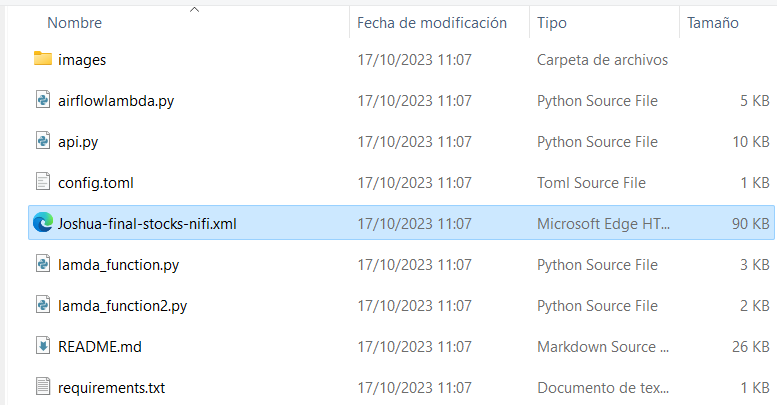
Apache NIFi se connecte à FASTAPI installé sur une instance EC2 avec NGINX

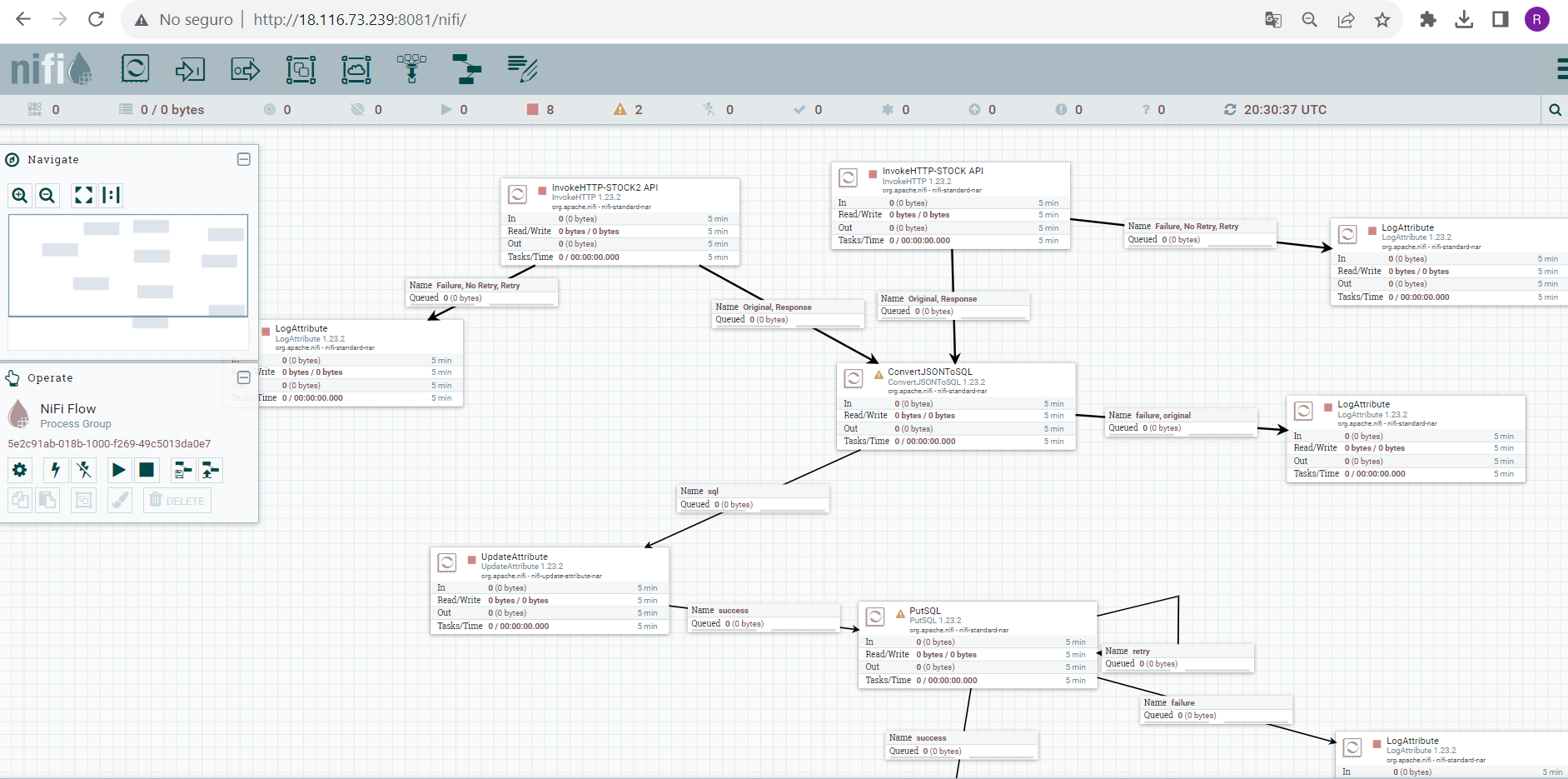




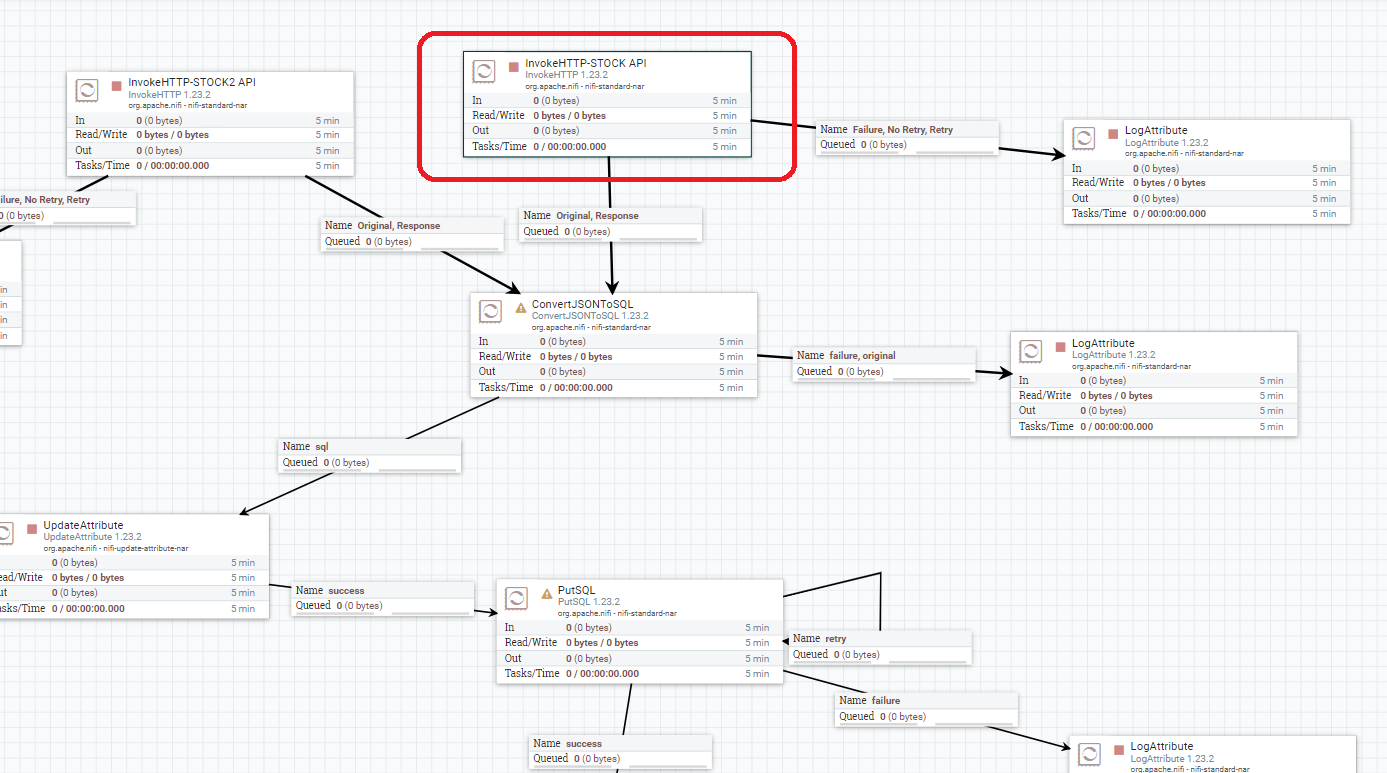
Nous chargeons le modèle

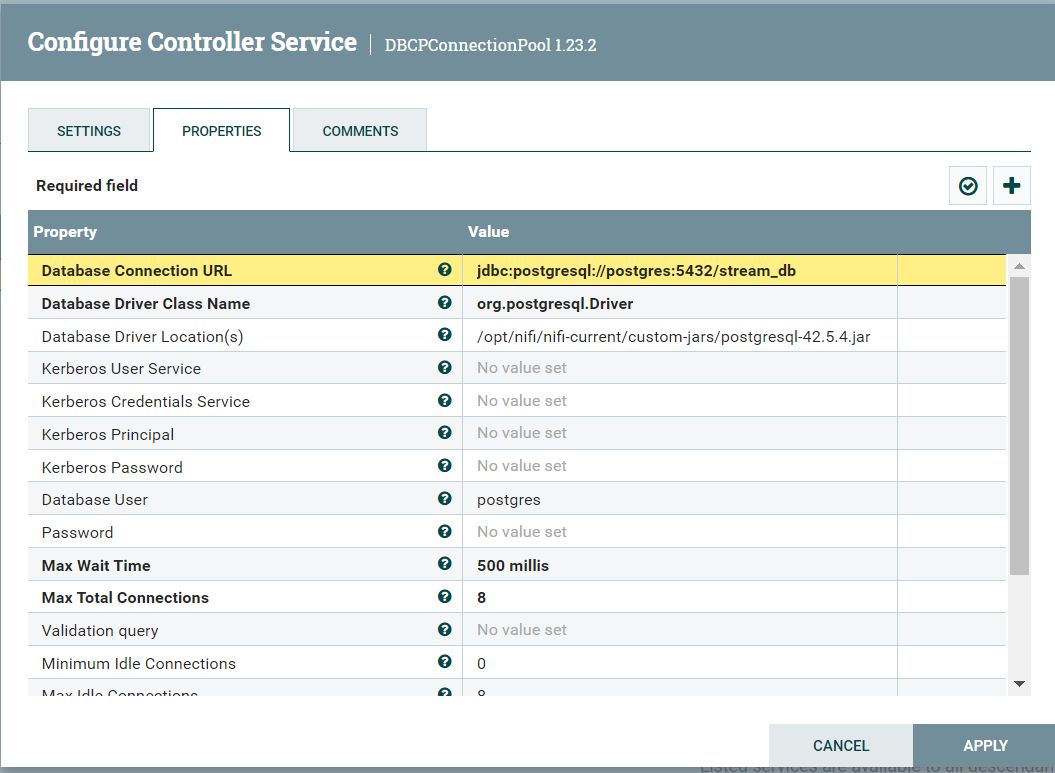


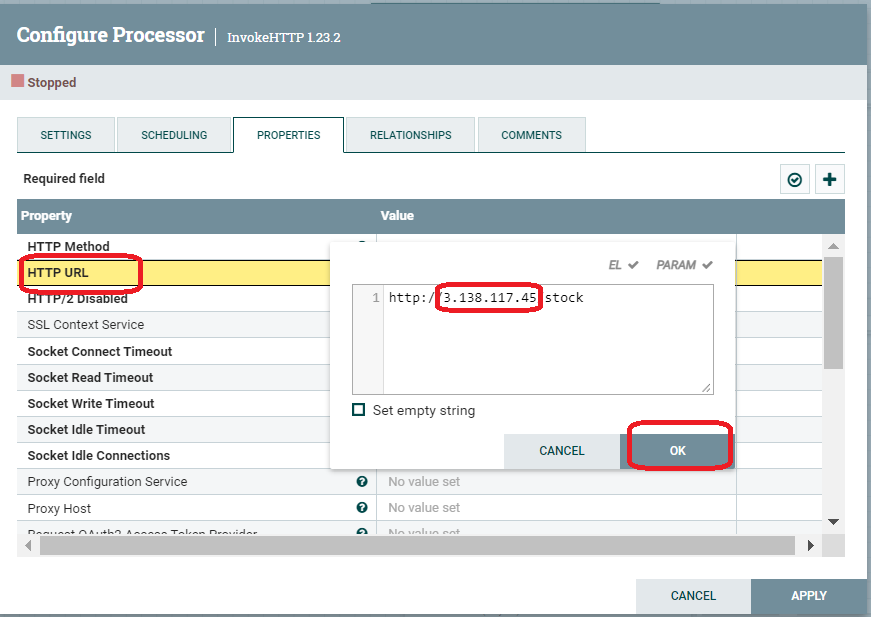




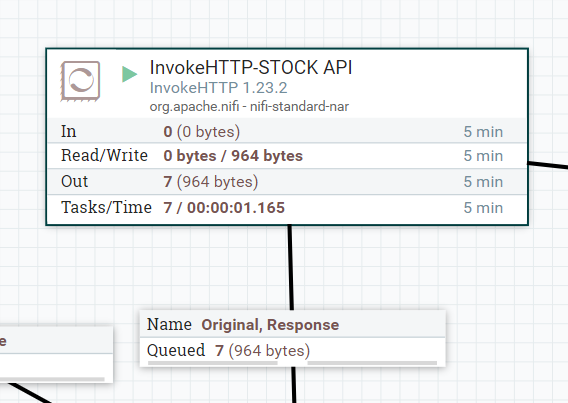
Nous allons changer l'IP du processus qui obtient les informations de l'API NGINX



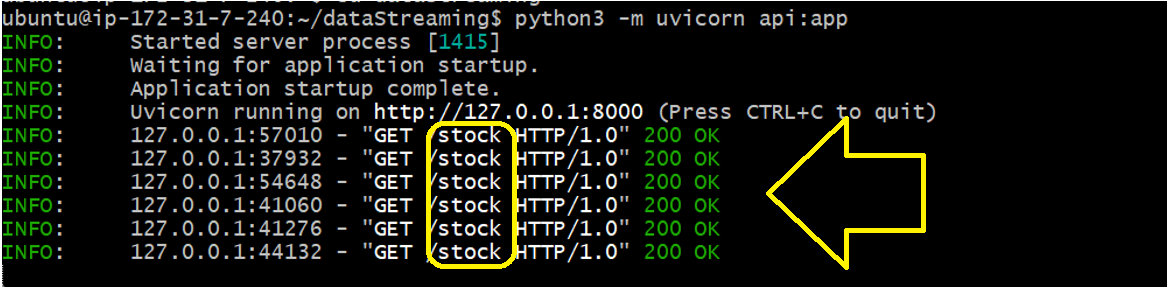




Nous allons maintenant exécuter le processus API InvokeHTTP -STOCK. Ce processus appelle l'API Stock pour obtenir les données boursières des sociétés Amazon et Apple. L'API obtient les données d'un bucket dans S3



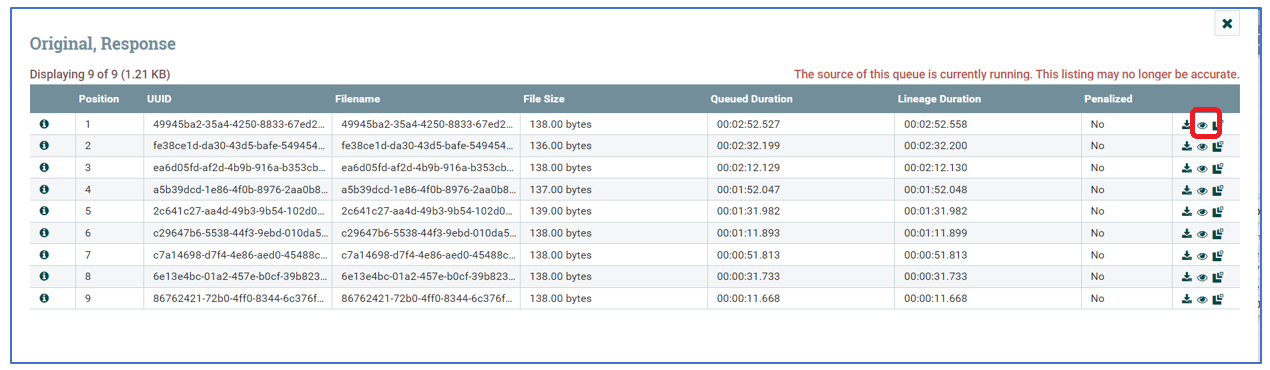
Dans l'écran suivant, nous voyons comment pour chaque accès à l'API dans la console dans NGINX une exécution est ajoutée



Nous allons maintenant voir la file d'attente d'inscription dans NIFI



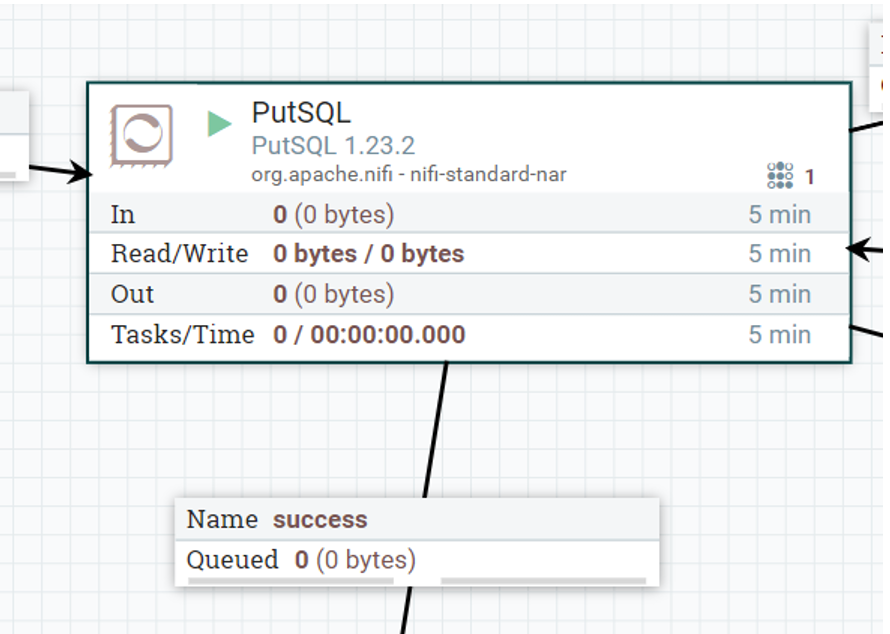
Nous verrons les enregistrements obtenus à partir de l'API



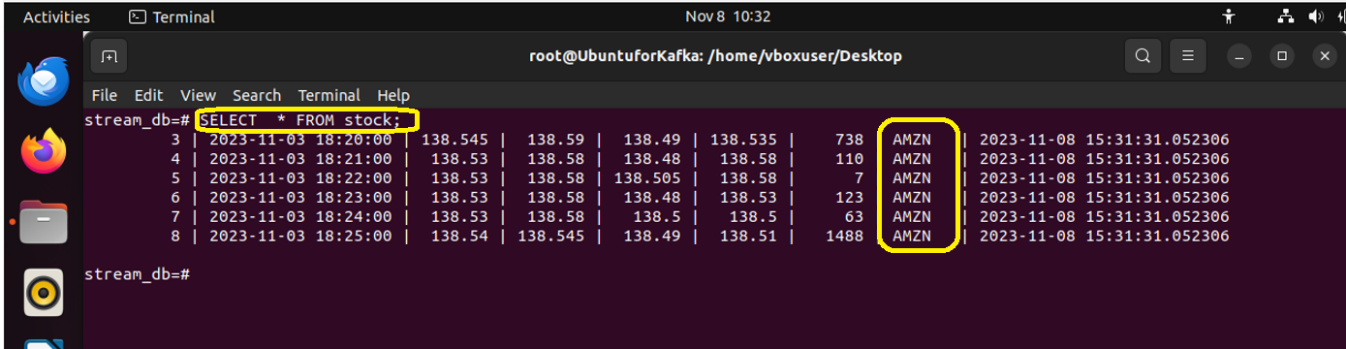
Nous allons maintenant voir les détails de chaque enregistrement. Ce sont les données qui seront insérées dans la base de données Postgres .



base de données stream\_db hébergée dans Postgres



Nous vérifions si les données ont été insérées dans la base de données



**connecteur Debezium avec Postgres**

curl -X POST -H " Accepter : application / json " -H " Type de contenu : application / json " localhost : 8083/connectors/ -d '

{

"name": "test-connecteur",

"configuration": {

" connecteur.class ": " io.debezium.connector.postgresql.PostgresConnector ",

" base de données.nom d'hôte " : " postgres ",

" base de données.port " : " 5432 ",

" base de données.user " : " postgres ",

" base de données.password " : " postgres ",

" base de données.nomdb " : " stream\_db ",

"base de données.serveur.name": "serveurbd1",

" topic.prefix " : "dbserver1",

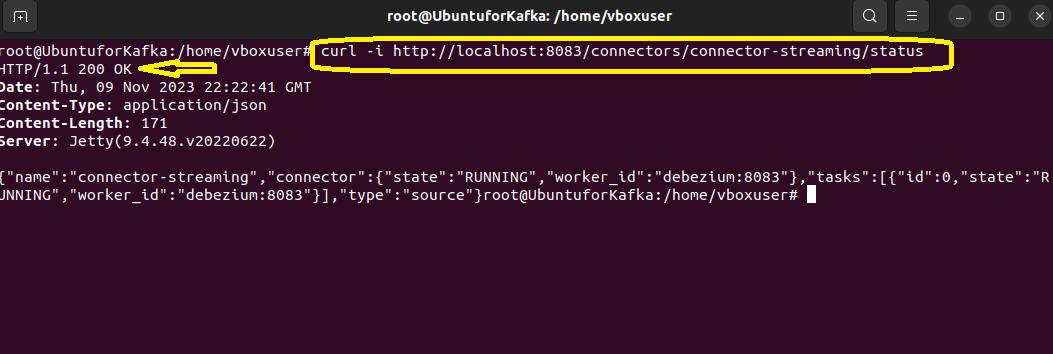
"plugin.name": " pgoutput "

}

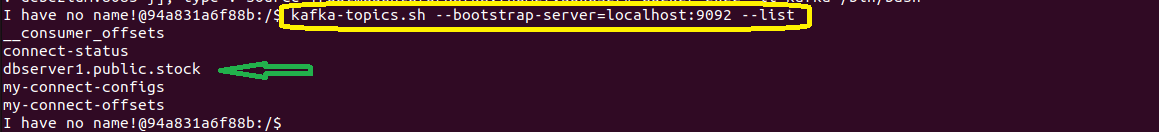
}'



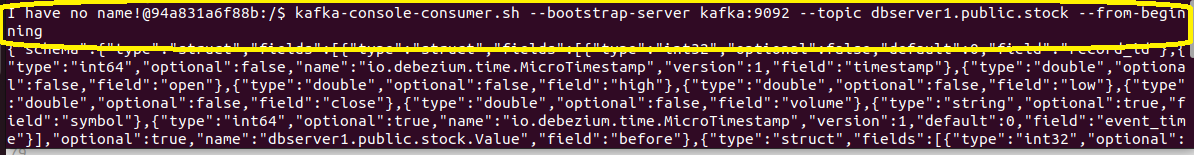
On vérifie que le connecteur est actif



conteneur Kafka et listons les sujets créés . Le sujet indiqué par la flèche verte a été créé automatiquement

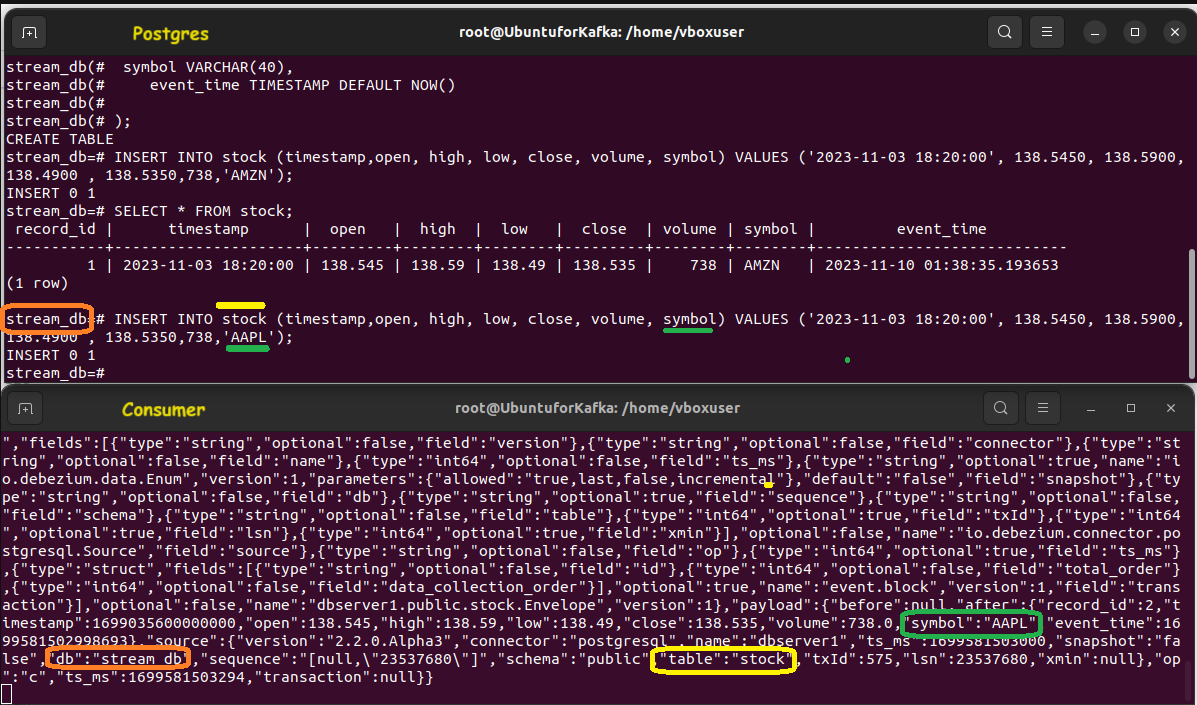


Nous allons maintenant démarrer le consommateur Kafka avec le sujet créé



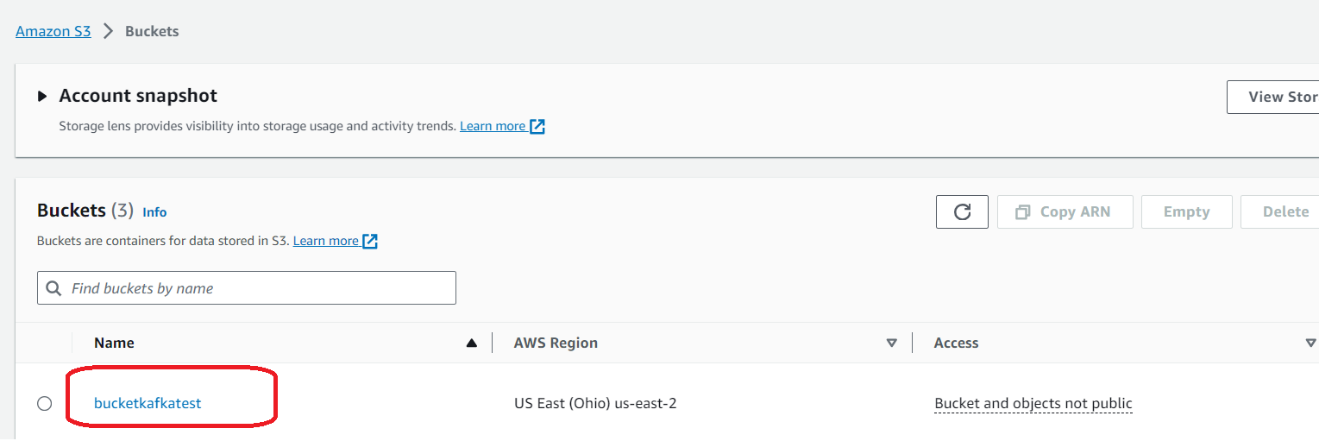
Maintenant, nous testons le connecteur.

Nous ferons une insertion dans la table STOCK de la base de données STREAM\_DB et la modification de la table sera automatiquement affichée sur l'écran du consommateur Kafka .



fichiers json générés par le connecteur Debezium vers un bucket S3 à l'aide d'un programme Python

Nous créons un bucket dans AWS S3

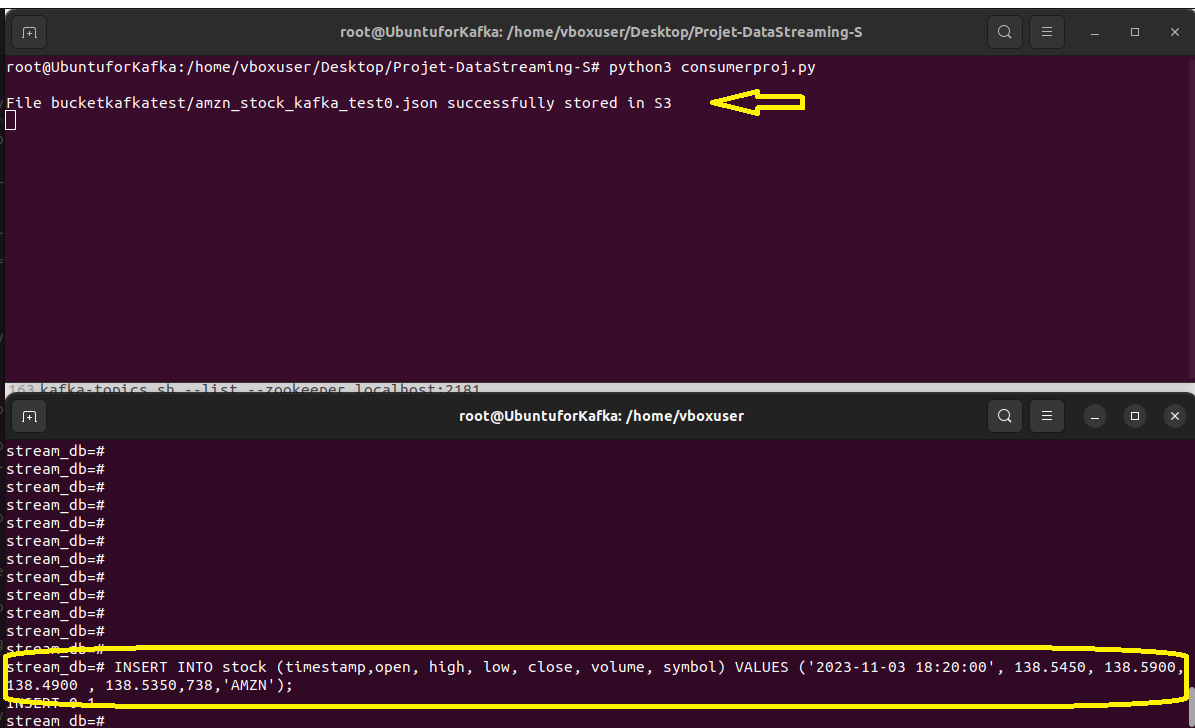


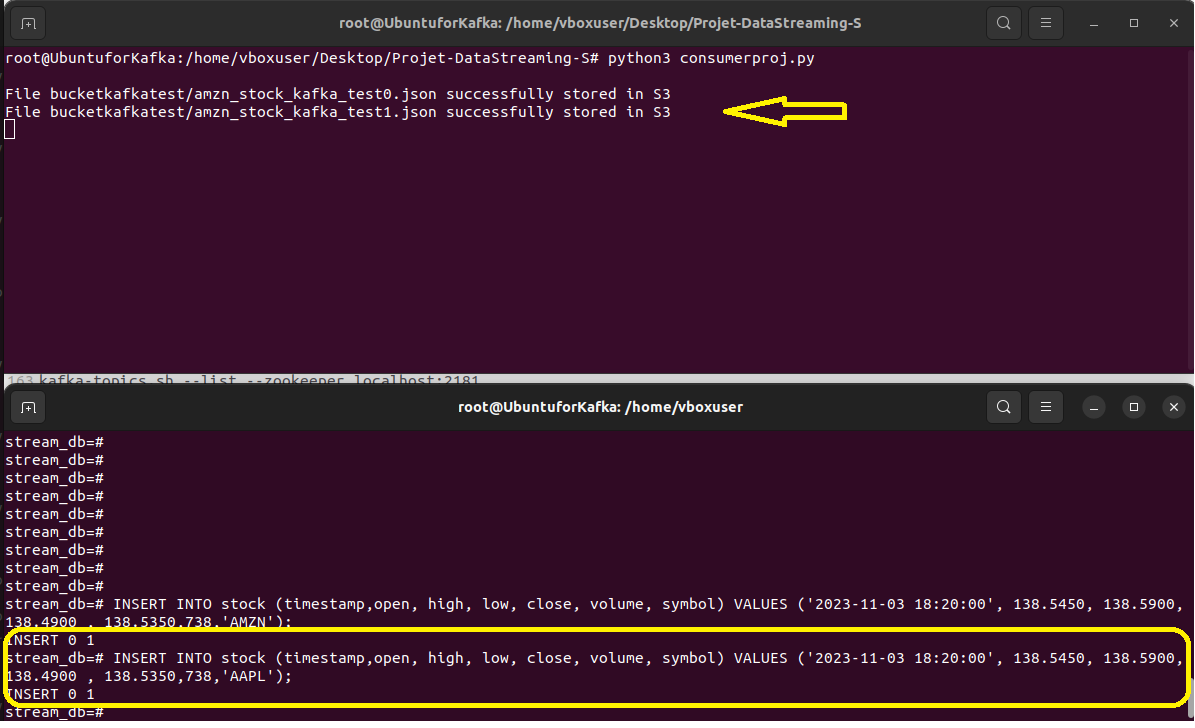
Ensuite, nous exécutons un programme Python que nous avons créé appelé consumerproj.py

Sur l'écran suivant, nous insérerons un enregistrement dans la table des stocks et le programme Python l'enverra au bucket S3



Nous appuyons sur Entrée dans l' éditeur Postgres SQL

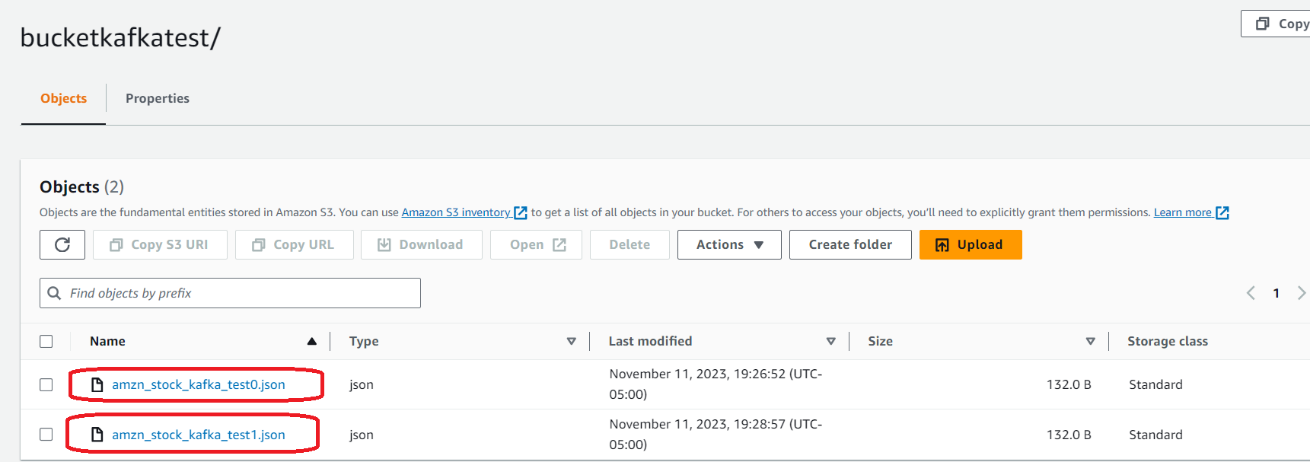




Le programme nous dit qu'il a envoyé les fichiers json à S3

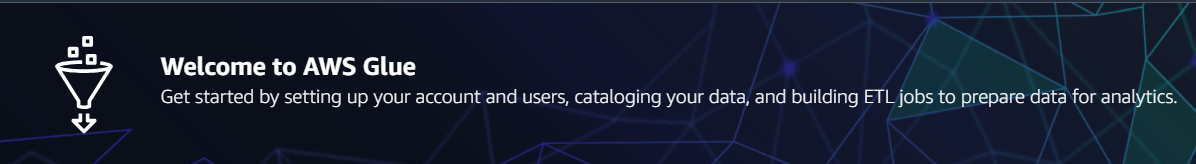
Nous allons au seau et vérifions si les fichiers sont arrivés.

Ici nous avons les fichiers créés

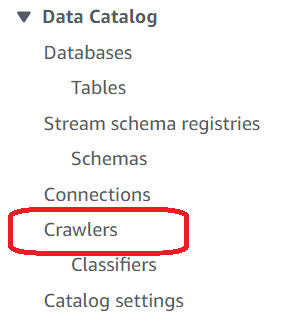


Travaillons maintenant avec les informations du bucket

Nous allons à AWS Glue

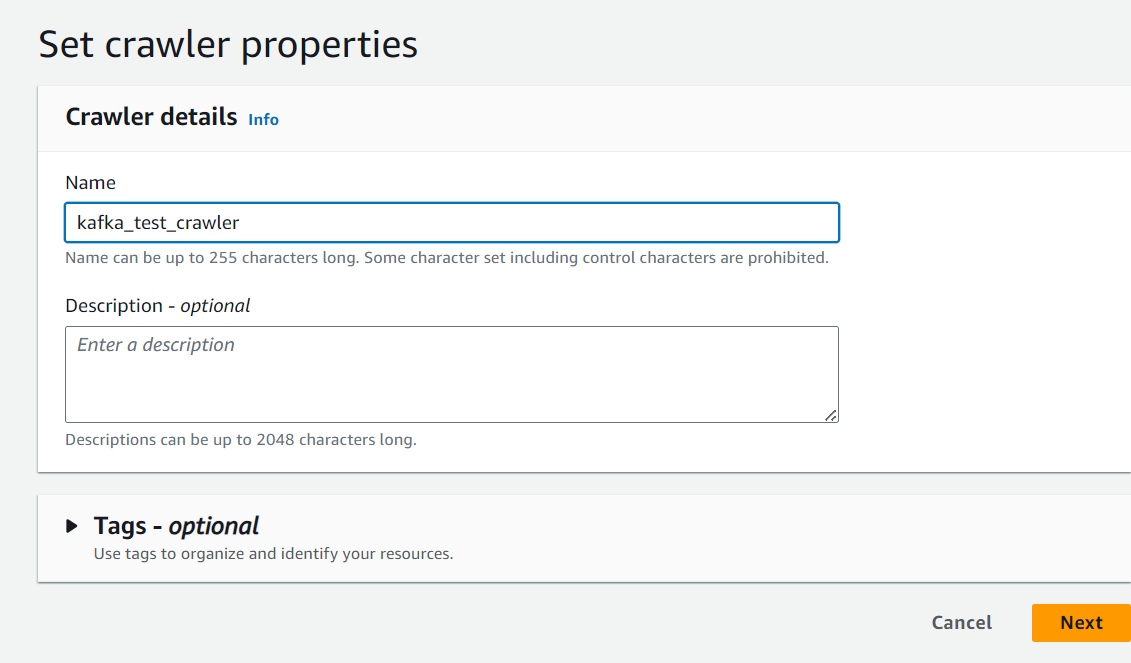


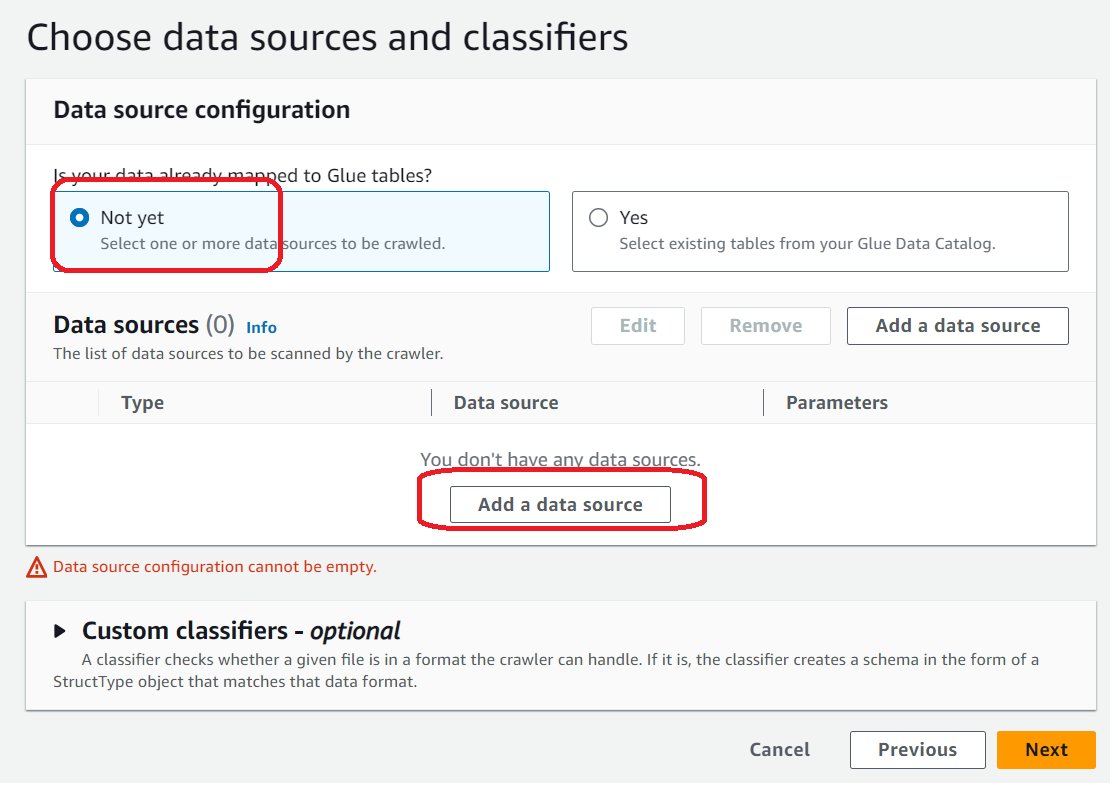
Et dans le menu de gauche on choisit

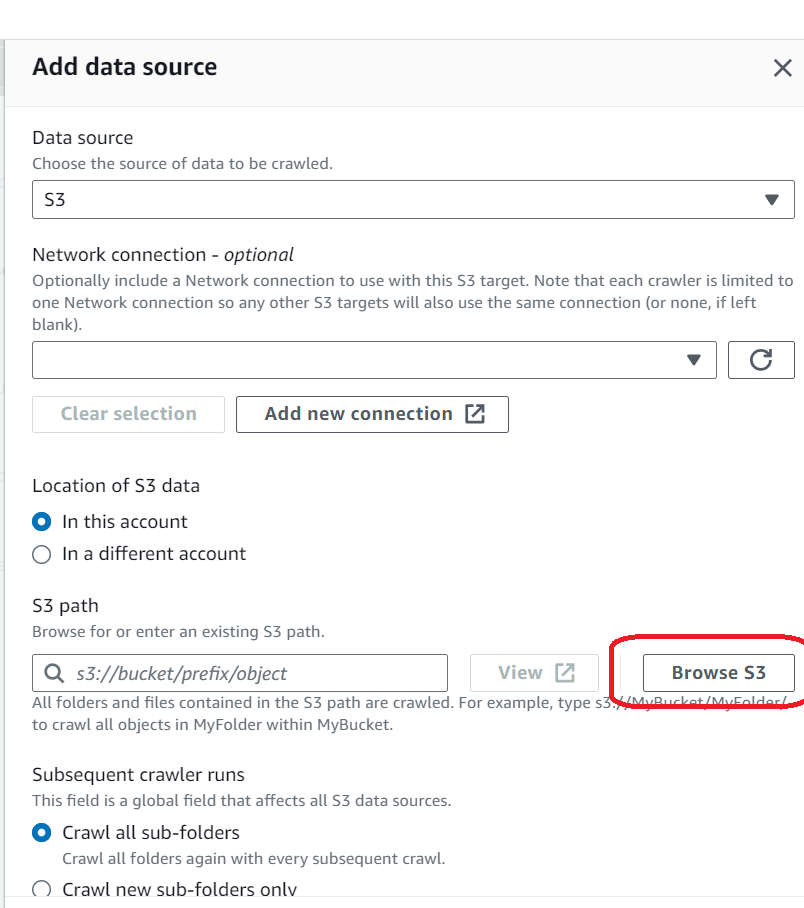


Créer Robots d'exploration

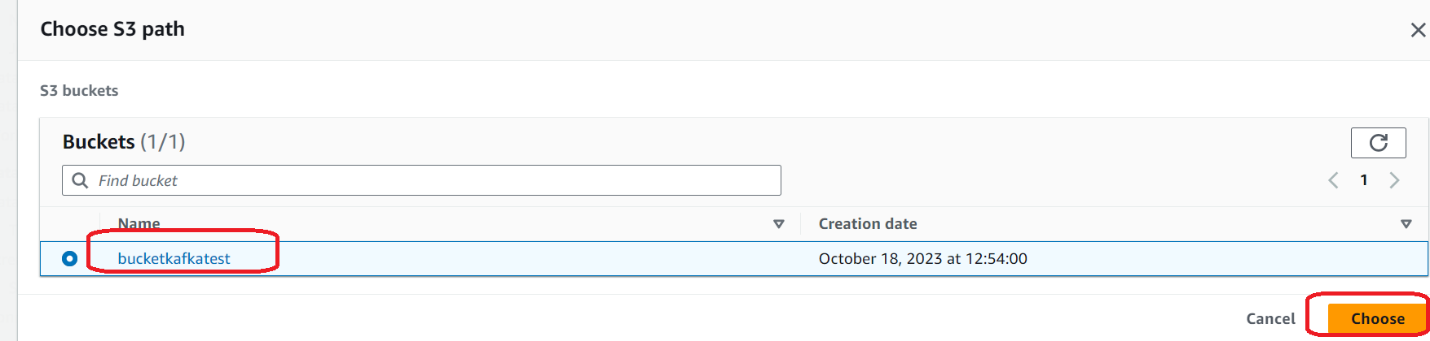


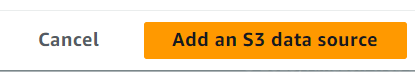


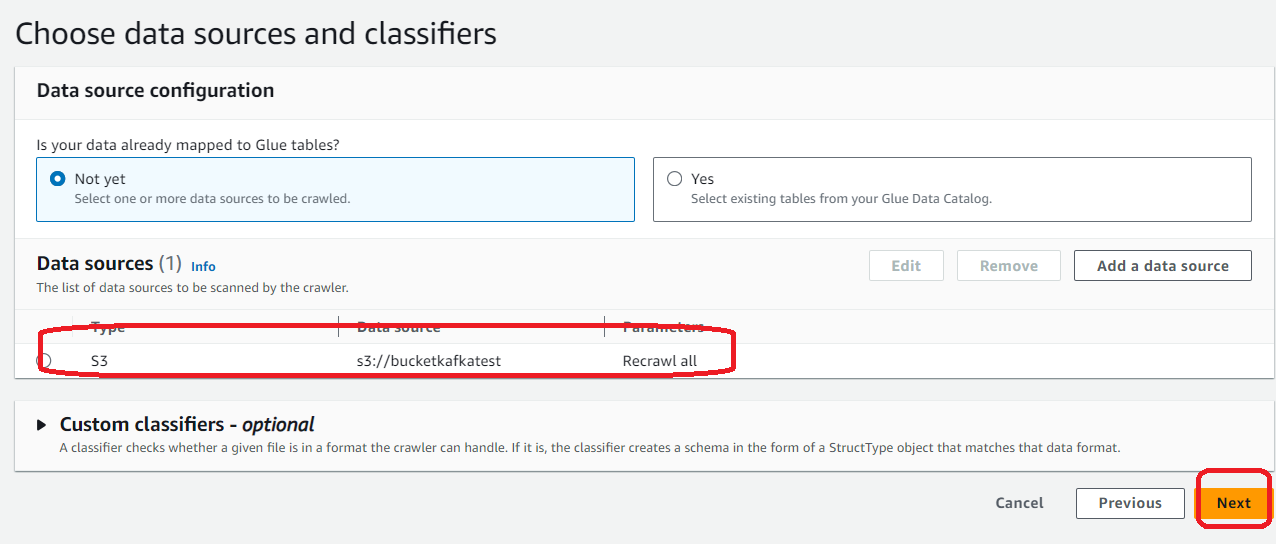




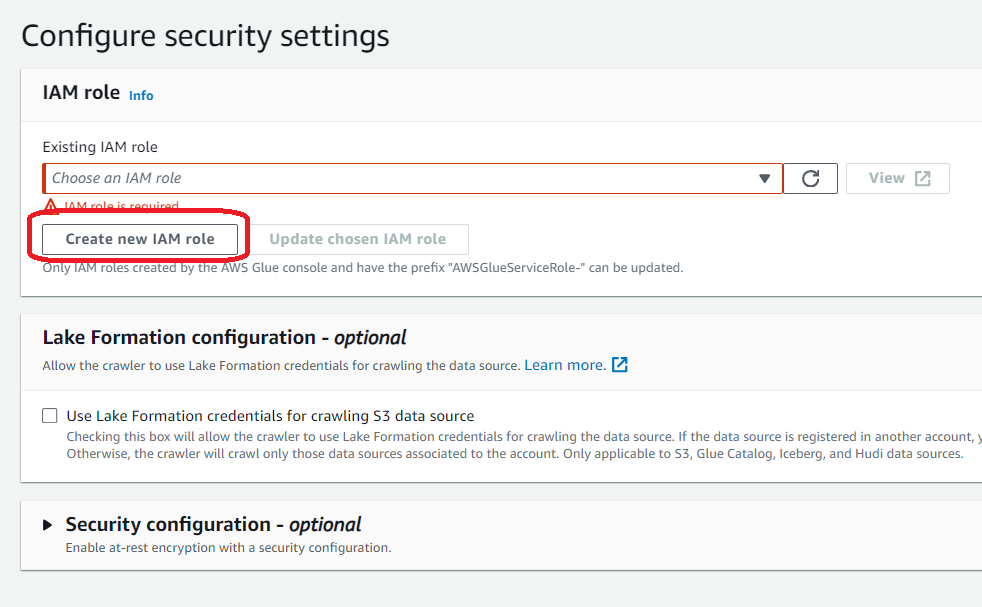
Choisissez le seau

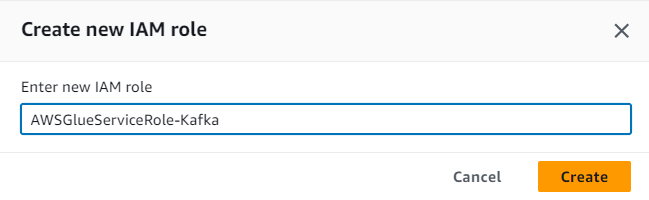


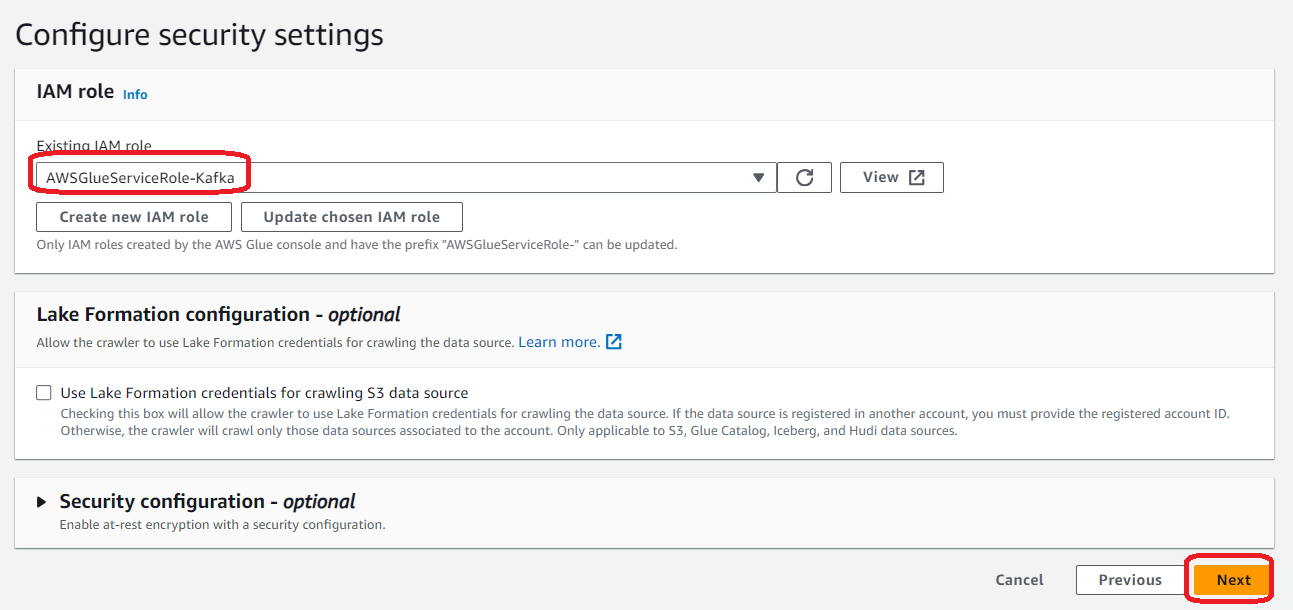




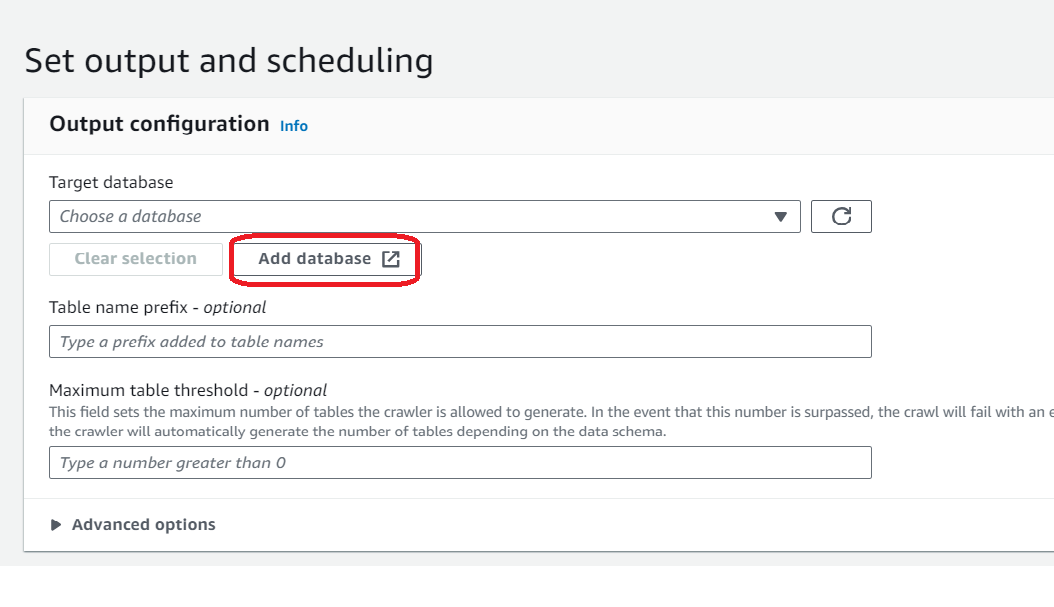
Créer un nouveau rôle



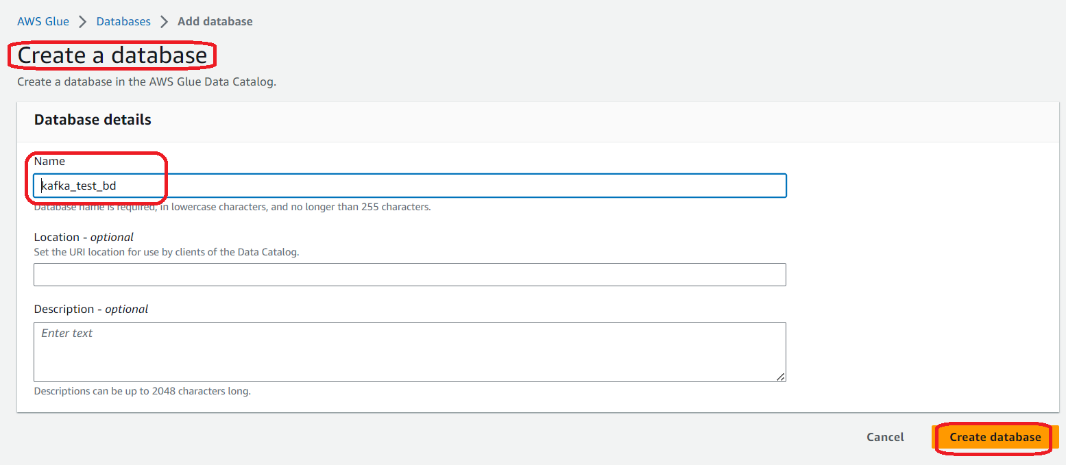


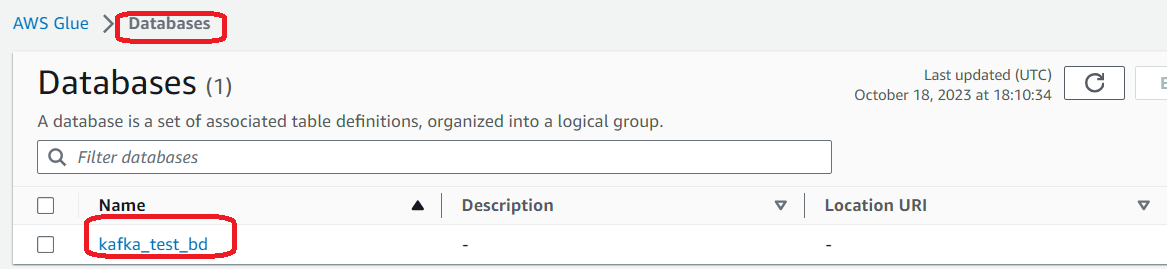


Créer une base de données

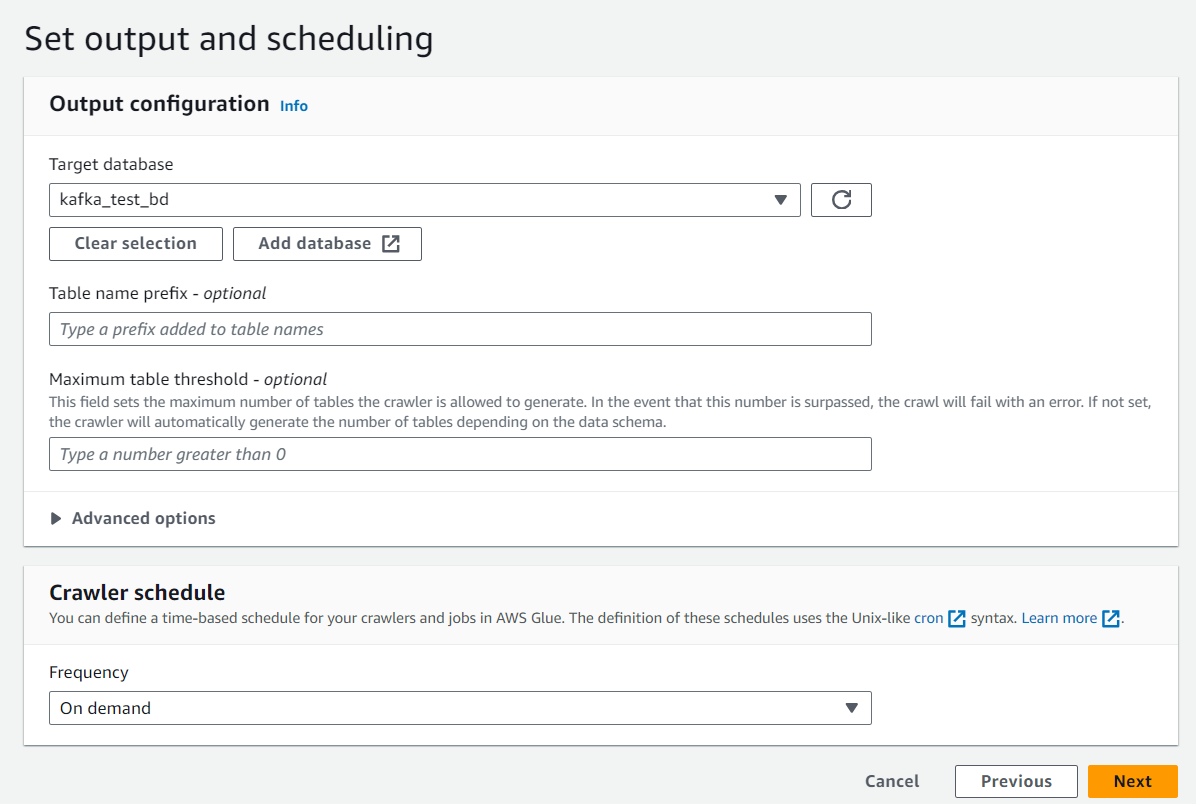


Entrez le nom et créez la base de données

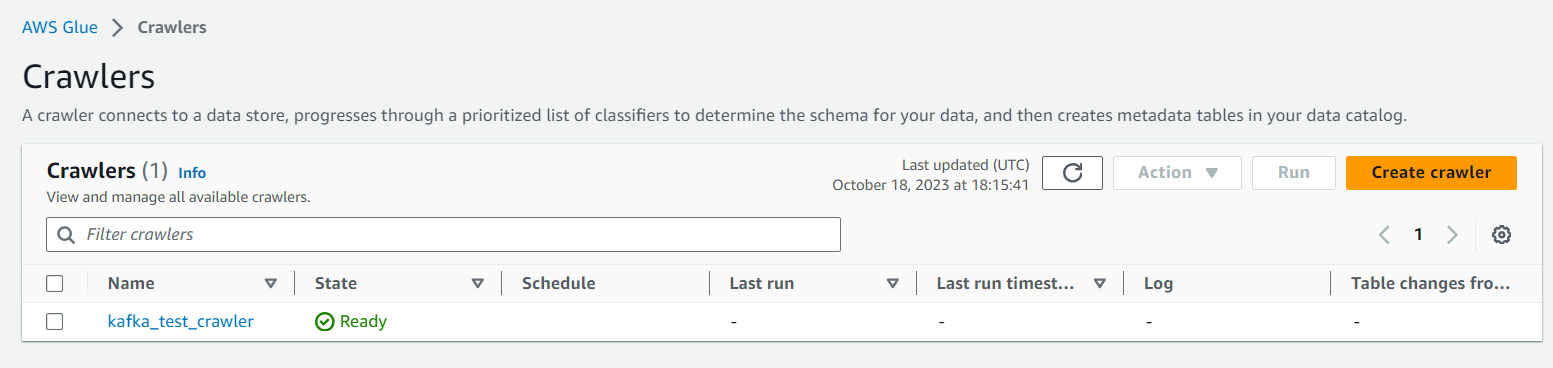




Nous revenons aux robots

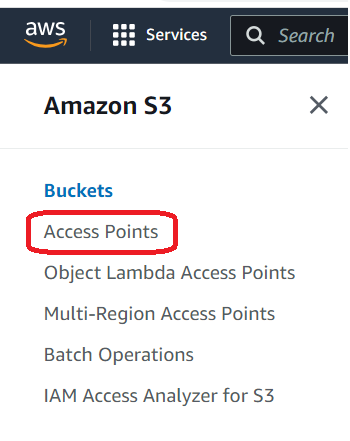


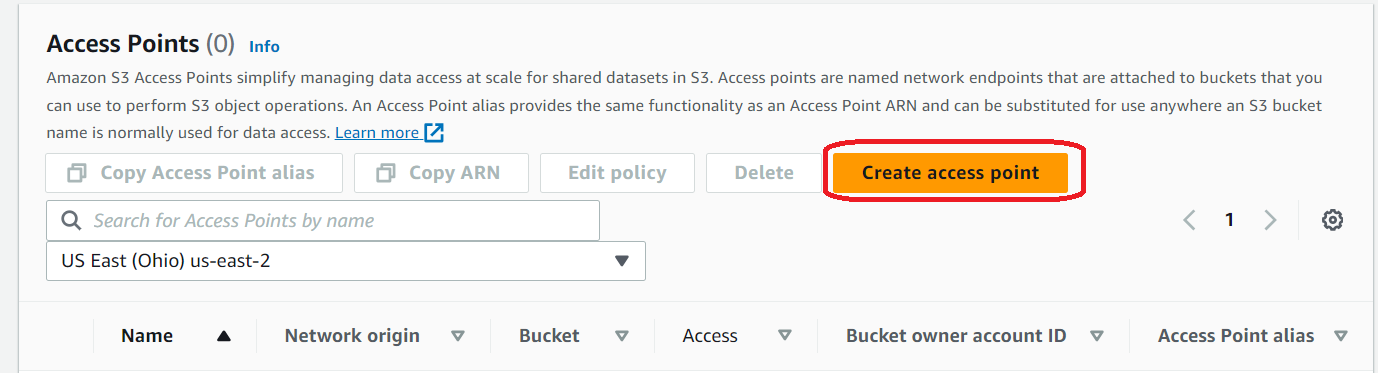


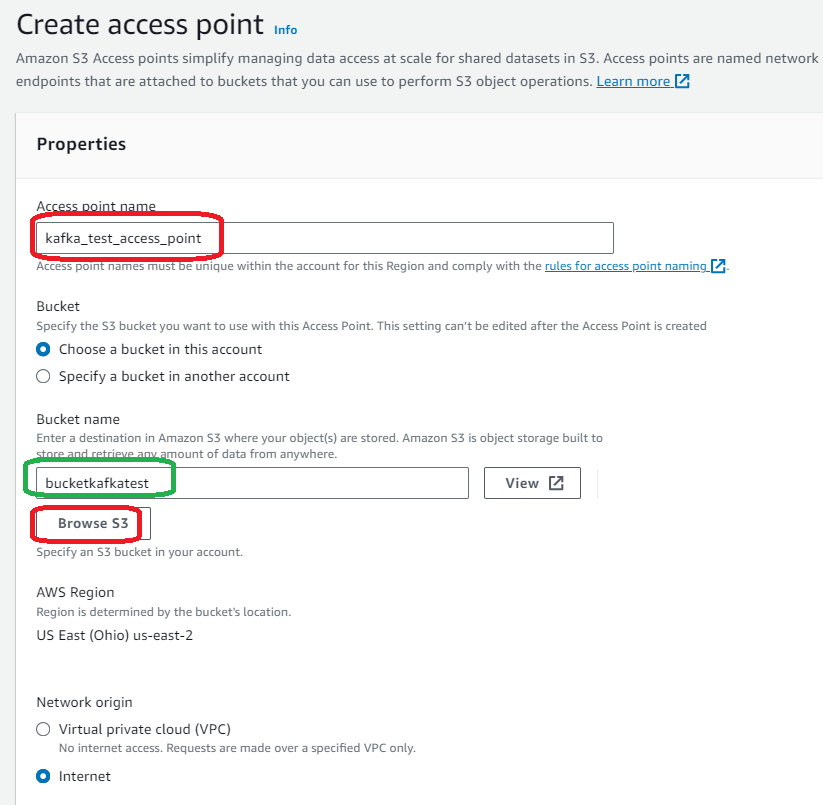


Nous créons un point d'accès

Nous allons sur S3 et choisissons l'option Points d'accès dans le menu de gauche.

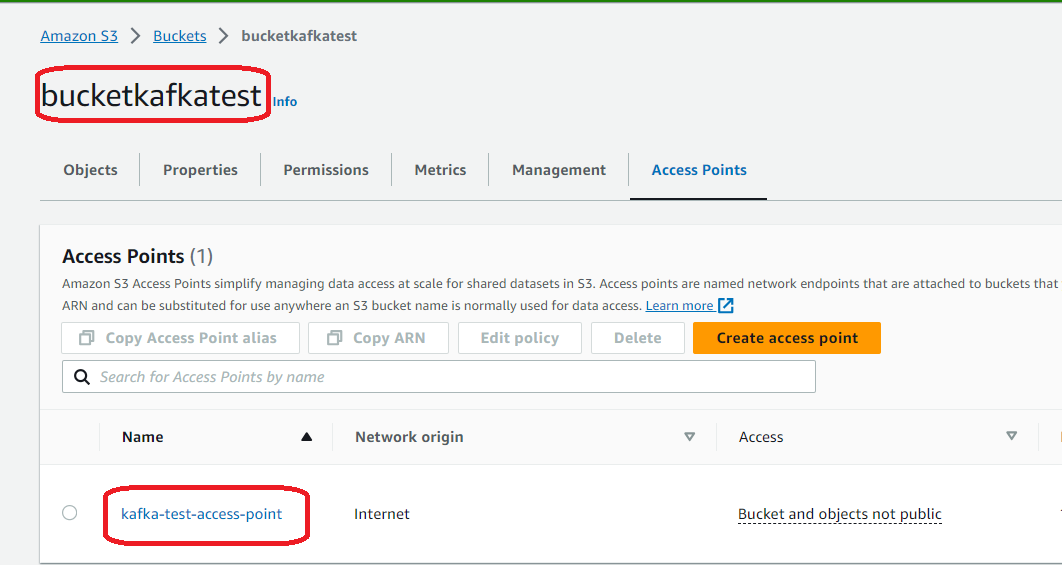








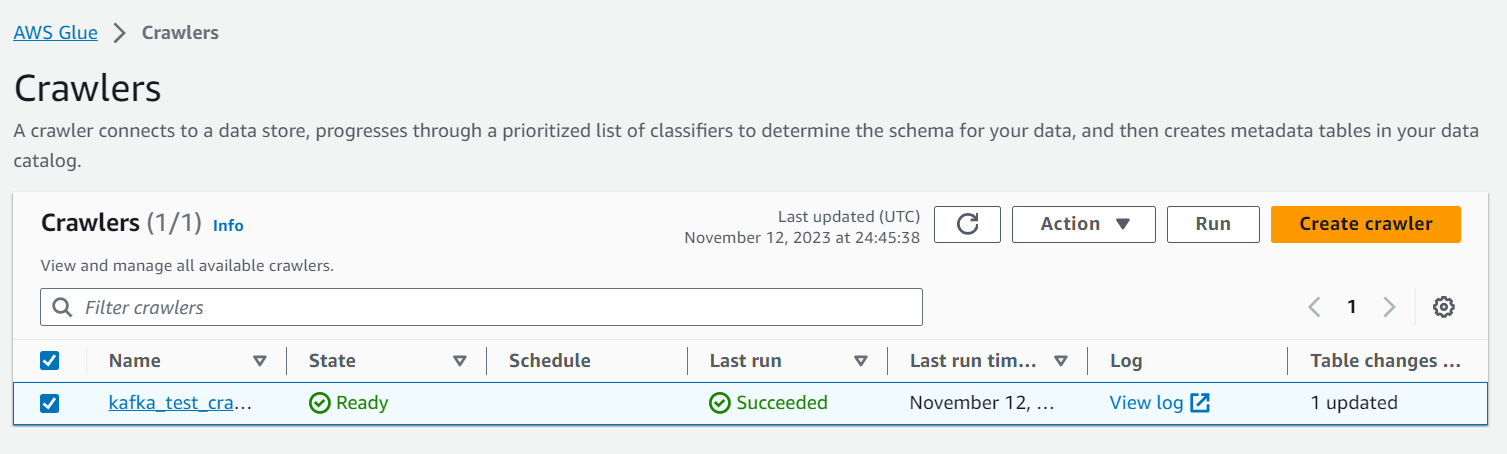
Le bucket est déjà lié au point d'accès créé .



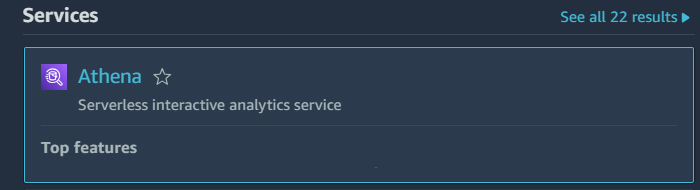
Ensuite, nous exécutons les robots d'exploration

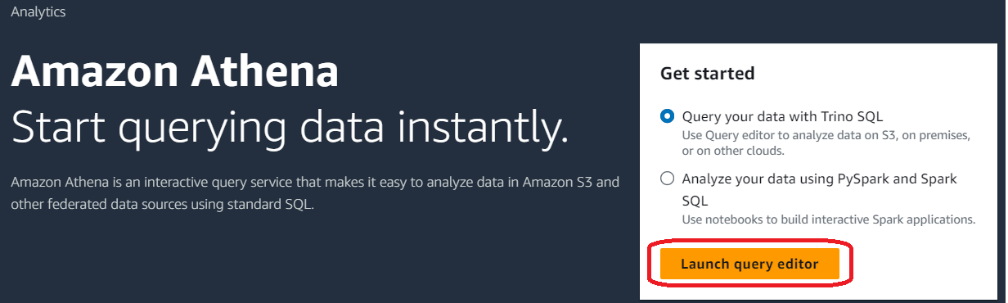


Cela nous indique qu'une table a été créée

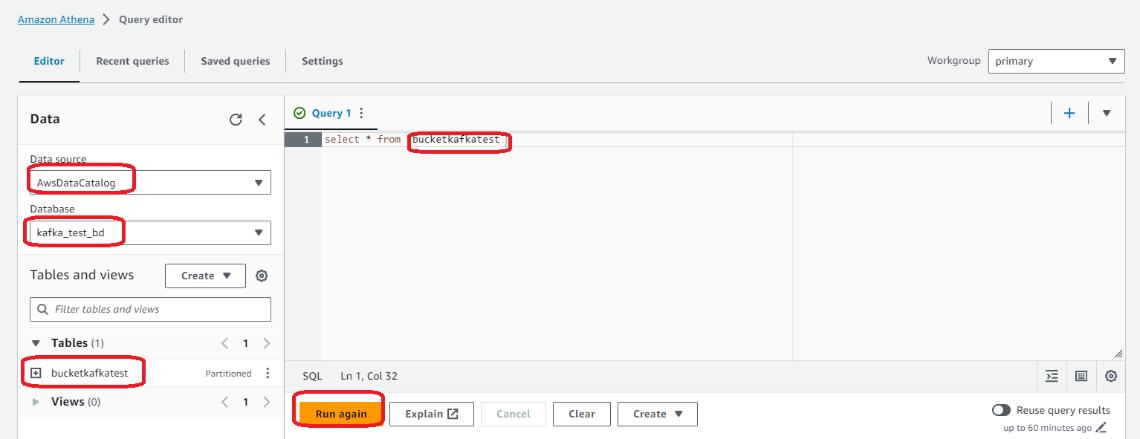


Maintenant, nous allons ouvrir Athéna

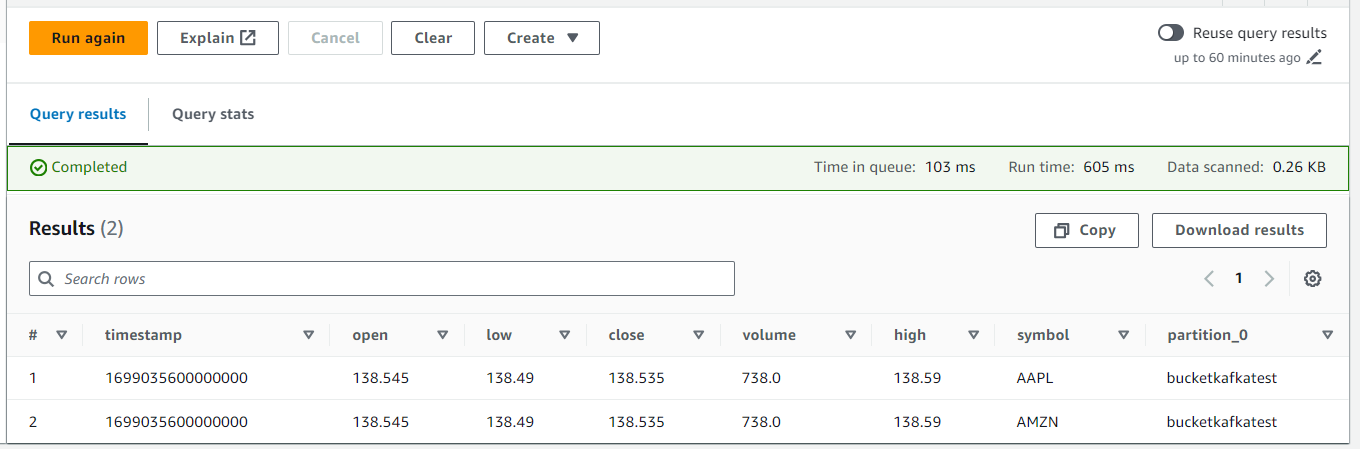




requête est saisie et exécutée



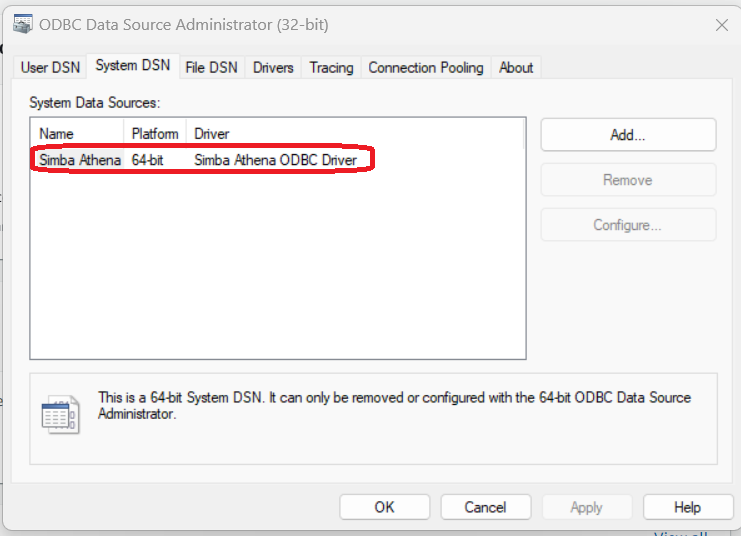
Le résultat de la requête



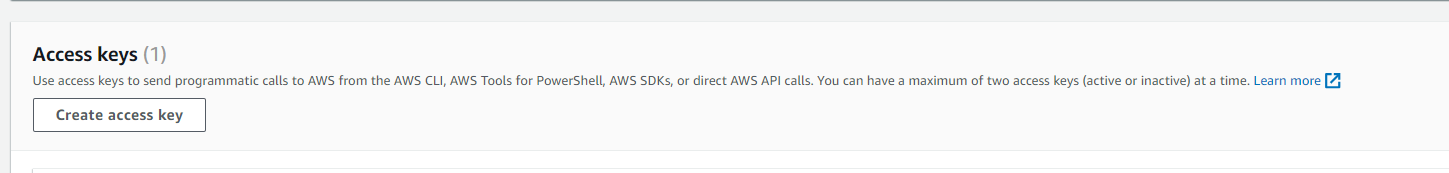
Connexion avec Power BI

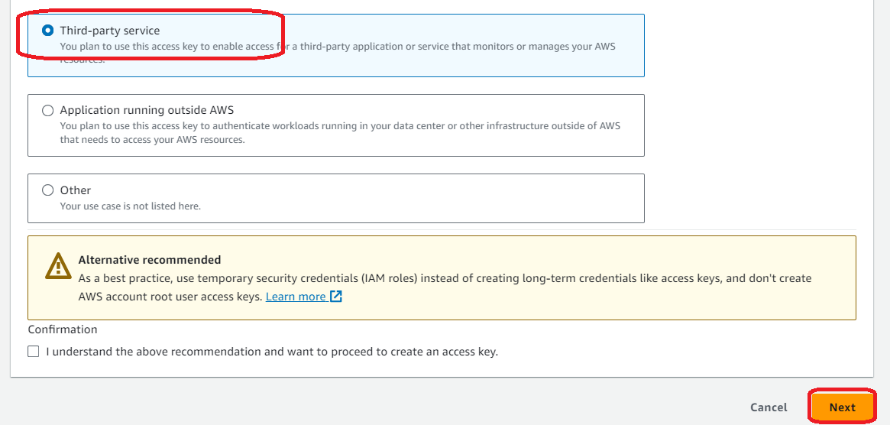
Téléchargez et installez le pilote ODBC pour Athena

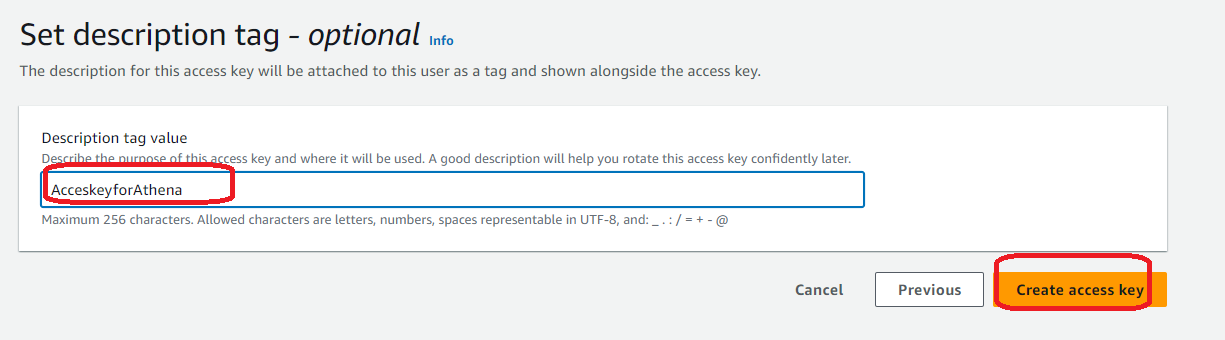




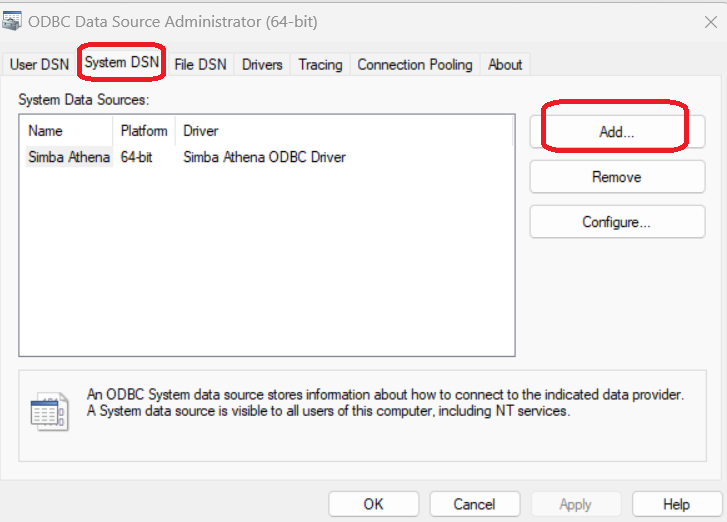
Créez ensuite une clé d'accès dans IAM



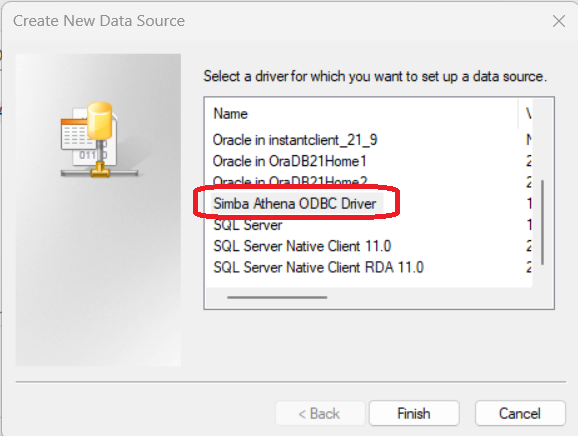




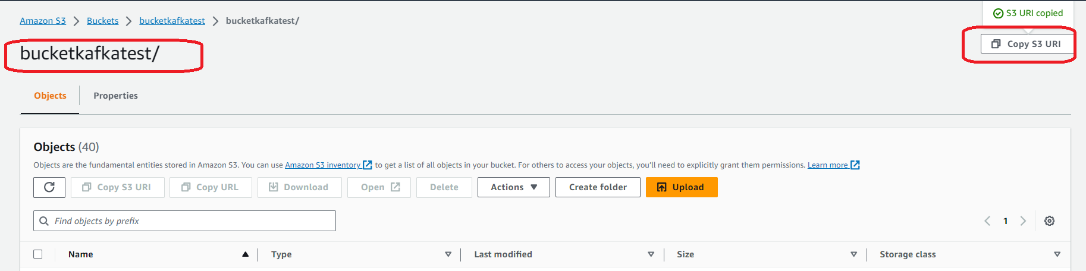
Ensuite, nous créons un nouveau DNS dans la source de données ODBC Administrateur



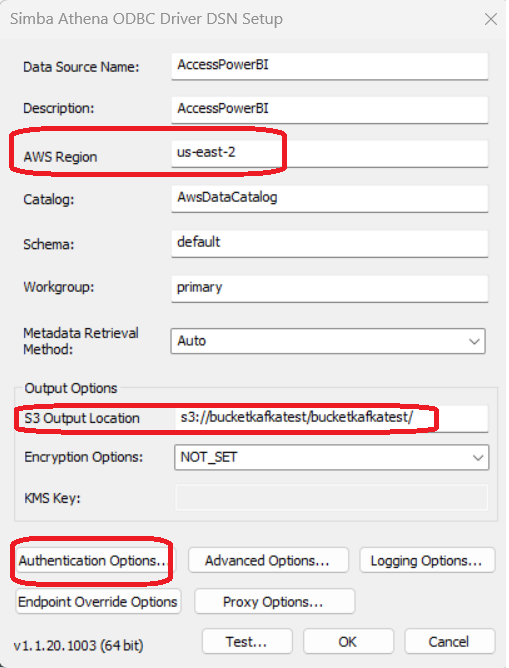
Nous sélectionnons le pilote Athena ODBC



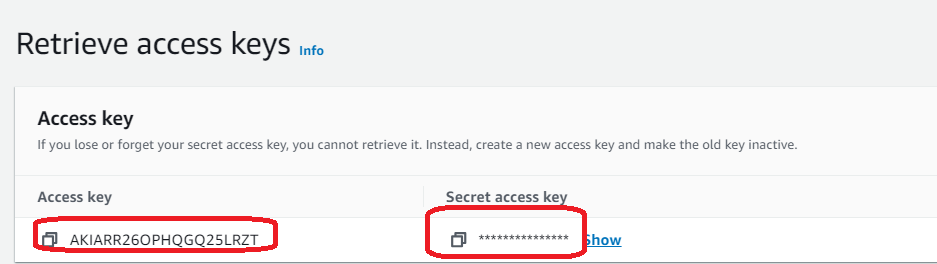
Copiez les informations de l'URL S3 , puis ajoutez-les dans la configuration ODBC.

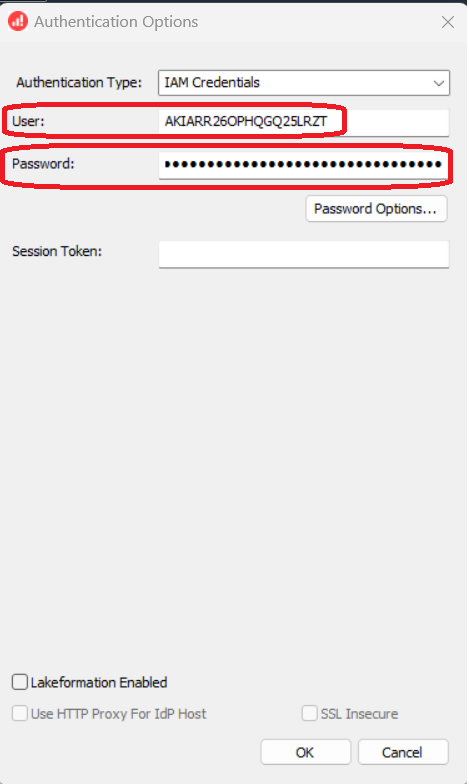


Vous collez l' URL S3 copiée , puis accédez à Authentification Possibilités

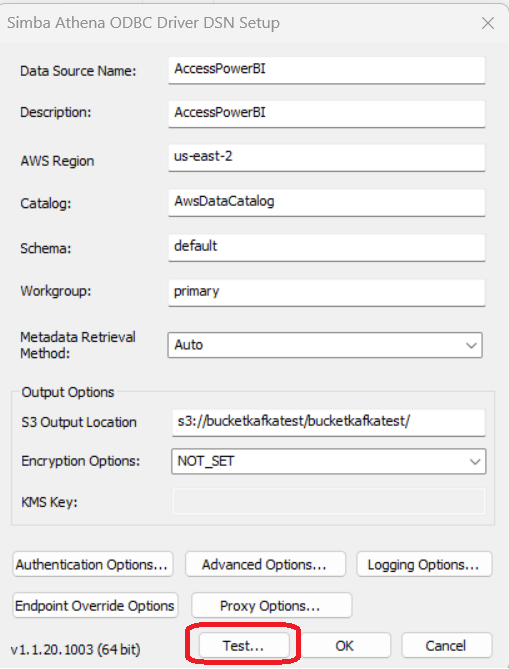


Nous allons sur IAM et copions la clé d'accès

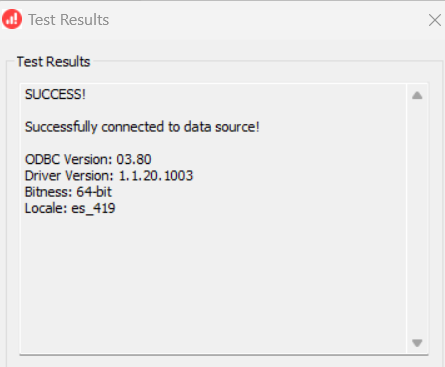




Ensuite vous effectuez le test de connexion

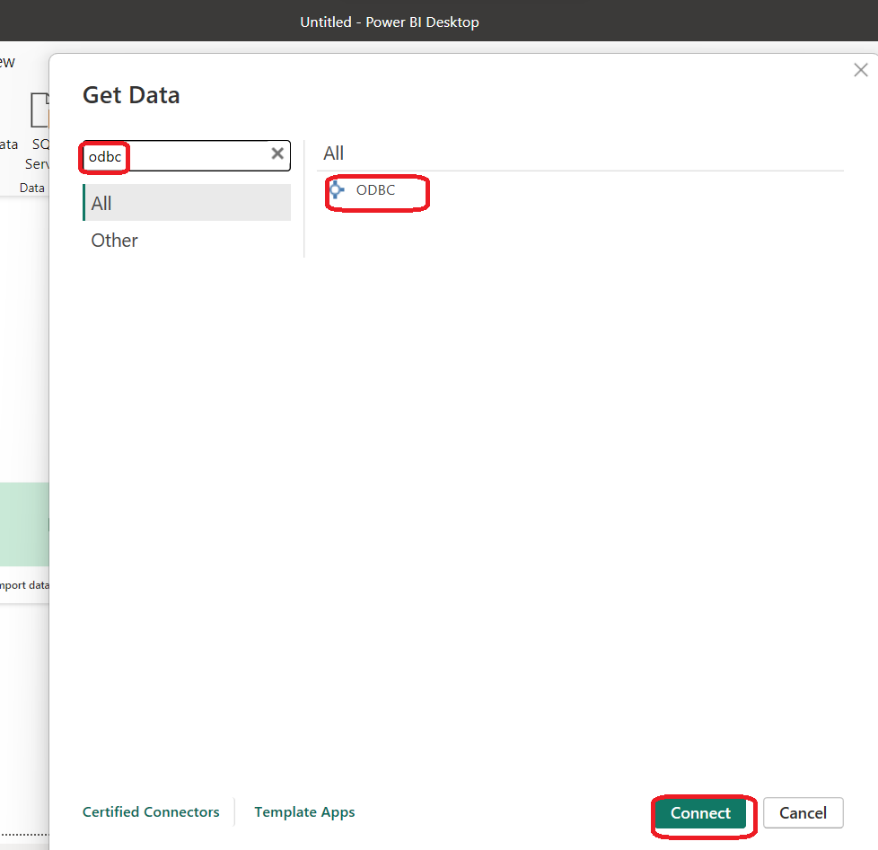


Test réussi

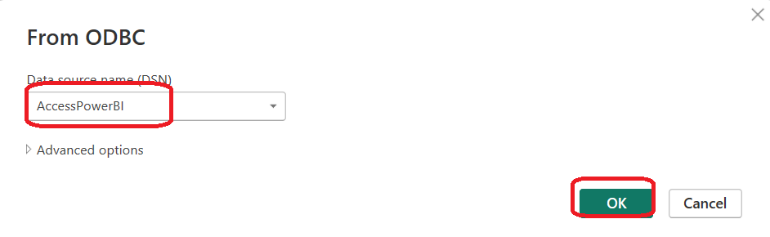


Nous ouvrons Power BI

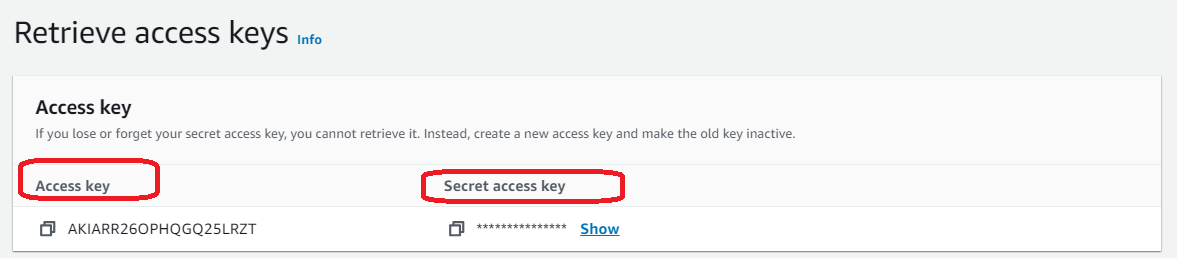
Pour la connexion de données, nous choisissons l'option ODBC

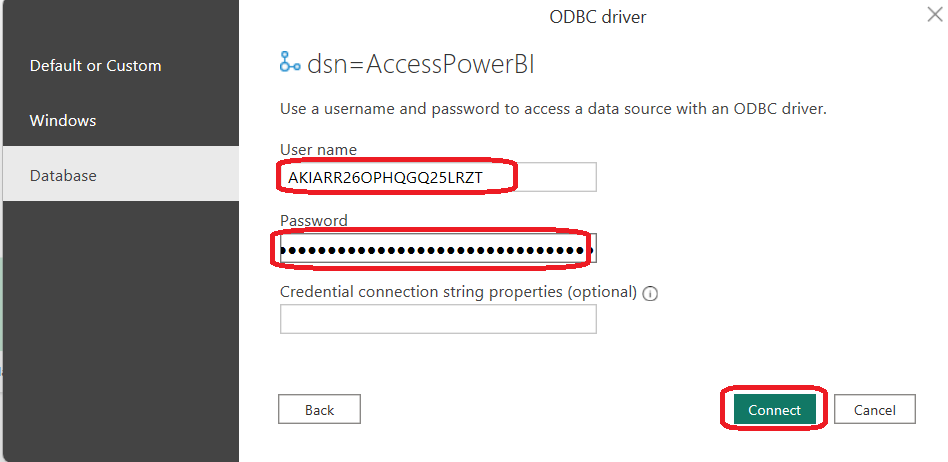


Nous choisissons l' obdc créé et cliquons sur OK



Power BI nous demande de saisir les informations d'identification de sécurité que nous devons copier depuis IAM





Nous avons pu nous connecter avec Athéna



Nous créons un graphique avec les données Athena

