

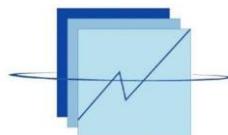
RÉPUBLIQUE DU SÉNÉGAL

Un Peuple - Un But - Une Foi



Ministère de l'Économie, du Plan et de la Coopération

Agence nationale de la Statistique et de la Démographie (ANSD)



Ecole nationale de la Statistique et de L'Analyse économique Pierre NDIAYE (ENSAE)



BIGDATA ET CLOUD COMPUTING:

DOCUMENTATION SUR LA CREATION D'UN DATA WAREHOUSE POUR L'ANALYSE DES VENTES

Rédigé par :

Judicaël Oscar Gandwende KAFANDO

Ramatoulaye NDEYE NDOYE FALL

Djerakei MISTALENGAR

Ben Idriss Diloma SOMA

Sous la supervision de :

Mme Mously DIAW

Table des matières

| | | |
|------|---|----|
| I. | Création d'un data lake avec amazon S3 | 2 |
| I.1 | Création d'un groupe et de plusieurs IAM ID (5 au total) | 4 |
| I.2 | Création d'un bucket sur AWS Glue | 5 |
| I.3 | Transformation des tables CSV en parquet | 8 |
| II. | PARTIE TEMPS REELLE DU PROJET..... | 12 |
| II.1 | 1. Le moteur de simulation et la gestion des stocks (DynamoDB)..... | 12 |
| II.2 | Génération de commandes via AWS Lambda | 14 |
| II.3 | Ingestion, Buffering et Stockage..... | 15 |
| II.4 | Intégration dans le Data Warehouse et Transformation (Redshift & dbt) | 16 |
| II.5 | Visualisation (Power BI)..... | 17 |
| II.6 | Architecture Streaming (Temps Réel Pur) et Perspectives | 19 |

I. Création d'un data lake avec amazon S3

On part sur AWS , ensuite on recherche le service S3 qui se présente comme ça :

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Buckets', 'Access management and security', and 'Storage management and insights'. The main area is titled 'General purpose buckets' and shows a list of three buckets: 'aws-glue-assets-355142185625-eu-west-3', 'ensae-student-data', and 'project-olist-ensae'. Each bucket entry includes its name, AWS Region (Europe (Paris) eu-west-3), and creation date. Below the table are two buttons: 'Account snapshot' and 'External access summary - new'. The top right corner shows account information: Account ID: 3551-4218-5625, Europe (Paris), and BenSoma.

On crée un compartiment en créant sur le bouton **create bucket**

This screenshot is identical to the one above, showing the AWS S3 console with the 'General purpose buckets' list. However, a red box and a red arrow point to the 'Create bucket' button located at the top right of the table header. This visual cue indicates where the user should click to start creating a new bucket.

Après la création du bucket, les tables de la base de données sont chargées dans celui-ci.

Le bucket, déjà créé, a été nommé **projet-olist-ensae**. En cliquant sur **projet-olist-ensae**, on accède à cette section où la base de données est visible, les différentes tables se trouvant dans le dossier **bronze**.

Amazon S3 > Buckets > project-olist-ensae

project-olist-ensae [Info](#)

Objects [Metadata](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Actions [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

| <input type="checkbox"/> | Name | Type | Last modified | Size | Storage class |
|-------------------------------------|---------------------------------|--------|---------------|------|---------------|
| <input type="checkbox"/> | bronze-parquet/ | Folder | - | - | - |
| <input checked="" type="checkbox"/> | bronze/ | Folder | - | - | - |

Voici comment les différentes tables se présentent :

Amazon S3 > Buckets > project-olist-ensae > bronze/

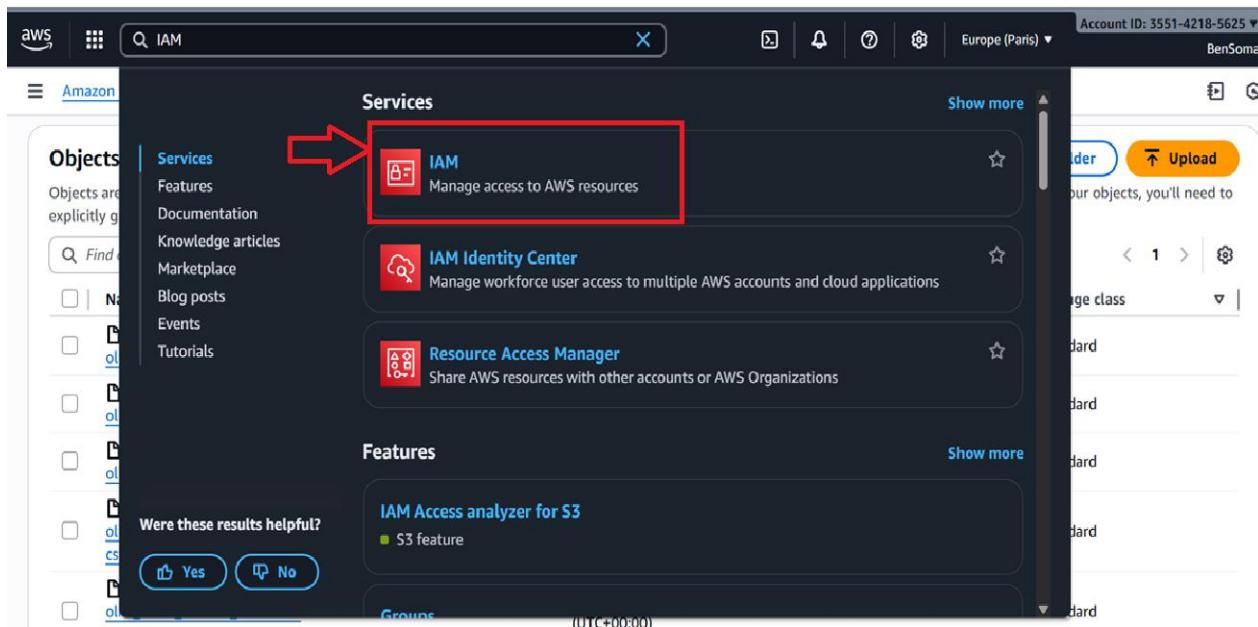
Objects (9) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

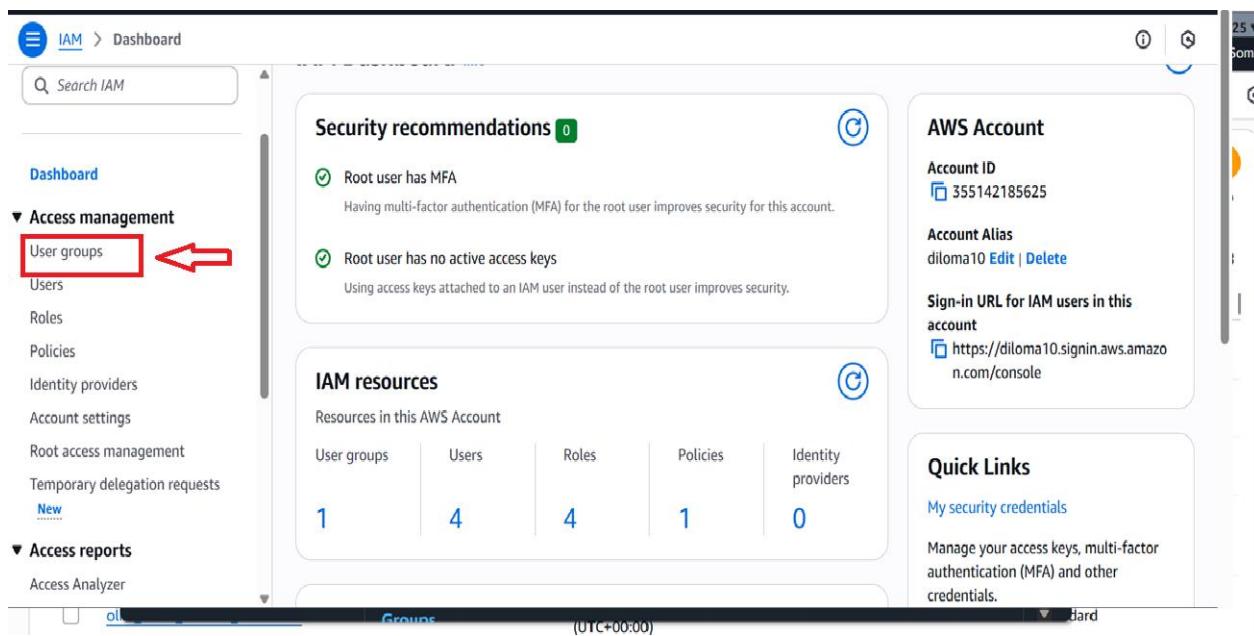
| <input type="checkbox"/> | Name | Type | Last modified | Size | Storage class |
|--------------------------|--|------|---|---------|---------------|
| <input type="checkbox"/> | olist_customers_dataset.csv | csv | December 11, 2025, 18:59:41 (UTC+00:00) | 8.6 MB | Standard |
| <input type="checkbox"/> | olist_geolocation_dataset.csv | csv | December 11, 2025, 18:59:41 (UTC+00:00) | 58.4 MB | Standard |
| <input type="checkbox"/> | olist_order_items_dataset.csv | csv | December 11, 2025, 18:59:41 (UTC+00:00) | 14.7 MB | Standard |
| <input type="checkbox"/> | olist_order_payments_dataset.csv | csv | December 11, 2025, 18:59:41 (UTC+00:00) | 5.5 MB | Standard |
| <input type="checkbox"/> | olist_order_reviews_dataset.csv | csv | December 11, 2025, 18:59:41 (UTC+00:00) | 13.8 MB | Standard |

I.1 Cr ation d'un groupe et de plusieurs IAM ID (5 au total)

Partir au niveau de la barre de recherche de la console et rechercher **IAM**



On clique sur **IAM**

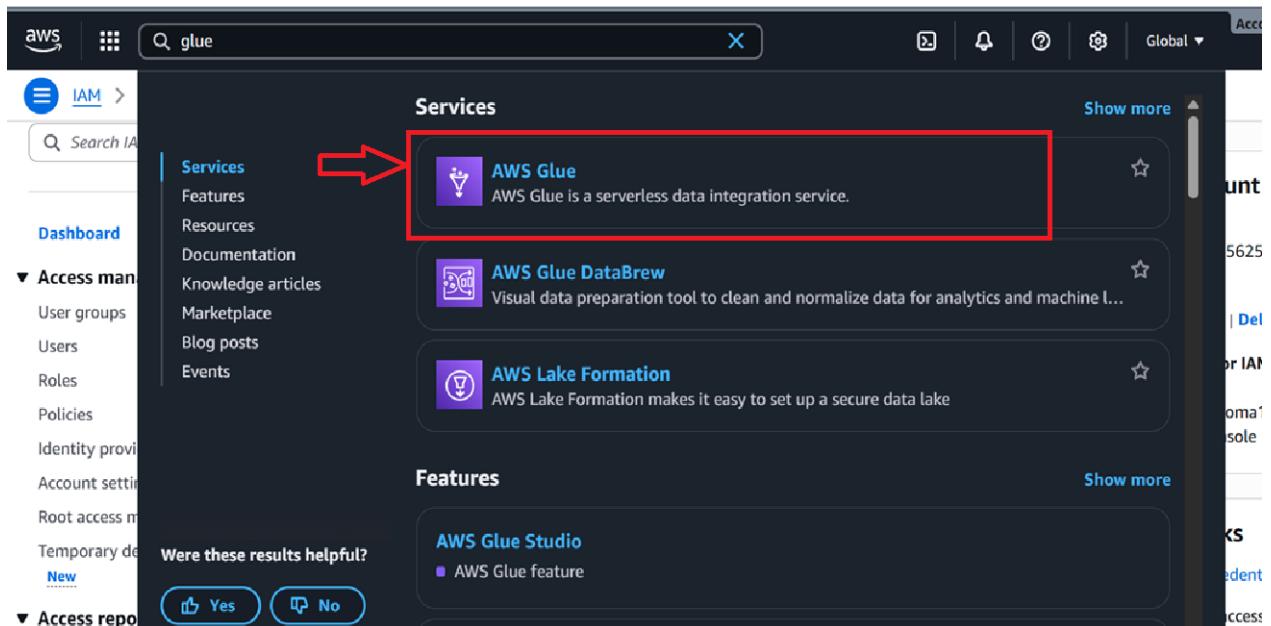


À présent, il convient de cliquer sur **User_groups** afin de créer les différents utilisateurs **IAM** disposant d'un accès complet aux ressources.

La création de ces utilisateurs permettra aux membres de l'équipe d'accéder à l'ensemble des travaux et d'y apporter des modifications.

I.2 Crédation d'un bucket sur AWS Glue

Rechercher **Glue** dans la barre de recherche



Il faut cliquer sur **AWS Glue**.

Après avoir cliqué sur ce service, on se rend dans l'onglet **Databases** situé à droite.

The screenshot shows the AWS Glue Welcome page. On the left, a sidebar menu is visible with a red box around the 'Data Catalog' section. A red arrow points from this box to the 'Databases' link under 'Data Catalog'. The main content area contains several cards: 'Prepare your account for AWS Glue', 'Catalog and search for datasets', 'Move and transform data', 'Resources and tutorials', and 'Data integration and management'.

The screenshot shows the AWS Glue Databases page. It displays a table with one database entry: 'olist-database'. The table has columns for Name, Description, Location URI, and Created on (UTC). At the top right, there are 'Edit' and 'Delete' buttons, and a prominent 'Add database' button highlighted with a red box and an arrow. The sidebar on the left is identical to the one in the first screenshot.

Il convient ensuite de cliquer sur « **Add database** » afin de créer la base de données sur AWS Glue, ce qui permettra de charger les tables depuis S3.

On crée un CRAWLER en cliquant sur **CRAWLER** dans la barre latérale

The screenshot shows the AWS Glue Crawler interface. On the left, there's a navigation sidebar with sections like 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Data Catalog' (expanded), 'Databases', 'Tables', 'Stream schema registries', 'Schemas', 'Connections', 'Crawlers' (expanded), 'Classifiers', and 'Catalog settings'. Below these are sections for 'Data Integration and ETL' and 'Legacy pages'. A 'What's New' link and a 'Documentation' link are also present. The main content area is titled 'Crawlers' and contains a table with one row. The table has columns for 'Name' (with 'olist' selected), 'State' (marked as 'Ready'), 'Last run' (showing success), 'Last run ...' (December 1...), 'Log' (View log), and 'Table cha...' (9 created). At the top right of the table are 'Action', 'Run', and a yellow 'Create crawler' button. A red arrow points to the 'Create crawler' button.

Après la creation de notre crawler, cliquer sur « **run a crawler** » et apres ca nous verrons les différentes tables qui ont été importées sur **GLUE** .

La base a eté crée et se nomme olist-database on clique sur olist-database

The screenshot shows the AWS Glue Databases interface. The left sidebar is identical to the previous screenshot. The main content area is titled 'Databases (1)' and contains a table with one row. The table has columns for 'Name' (with 'olist-database' selected), 'Description', 'Location URI', and 'Created on (UTC)' (December 11, 2025 at 19:07:30). At the top right of the table are 'Edit', 'Delete', and a yellow 'Add database' button. A red arrow points to the 'Add database' button. Another red arrow points to the 'olist-database' entry in the table.

I.3 Transformation des tables CSV en parquet

Pourquoi parquet ?

- Pour pouvoir gérer de grandes bases de données
- Pour faire ceci, il faut aller au niveau de **GLUE** dans la barre latérale à gauche nous cliquons sur « **ETL JOB** »

The screenshot shows the AWS Glue interface with the 'Crawlers' section selected. On the left sidebar, 'ETL jobs' is highlighted with a red box and an arrow pointing to it from the top-left. The main area displays a table of crawlers, with one entry named 'olist' shown as 'Ready' with a green checkmark, 'Succeeded', and 'December 1...'. A red box highlights the 'olist' row.

Dans **ETL Jobs** on clique sur script Editor

The screenshot shows the AWS Glue Studio interface with the 'Create job' section selected. On the left sidebar, 'ETL jobs' is highlighted with a red box and an arrow pointing to it from the top-left. The main area shows three options for creating a job: 'Visual ETL' (selected), 'Notebook', and 'Script editor'. 'Script editor' is highlighted with a red box and an arrow pointing down to it from the top-right. The 'Your jobs' section shows one job named 'csv_to_parquet'.

On choisit l'option spark pour le code

Un code a été utilisé pour transformer les fichiers CSV en parquet : il se trouve en annexes.

Lorsque l'on se rend dans S3, il est possible de constater qu'un nouveau dossier a été créé et qu'il contient les fichiers parquet. Pourquoi des tables sous format parquet ?

Parce que le format Parquet est plus performant que le CSV : il est orienté colonnes, permet une **compression efficace**, réduit l'espace de stockage sur S3, **accélère les lectures avec Spark**, conserve le **schéma des données** et est mieux adapté aux **analyses Big Data**.

Tableau 1: Résumé des services utilisés et leur rôle jusqu'à présent

| Service AWS | Utilité dans le projet |
|-------------------|--|
| Amazon S3 | Stockage scalable et sécurisé des données brutes et transformées. |
| IAM | Gestion des accès et des permissions pour les utilisateurs et services. |
| AWS Glue | Catalogage des données, crawling automatique, et transformation ETL serverless. |
| Glue Data Catalog | Catalogue centralisé des métadonnées pour interroger les données avec Athena/Redshift. |
| Glue Crawler | Découverte automatique des schémas et mise à jour du catalogue. |
| Glue ETL Job | Exécution de scripts PySpark pour transformer les données (CSV → Parquet). |
| Parquet | Format de stockage optimisé pour l'analytique Big Data. |

Source : calculs des auteurs

Après ces différentes étapes, l'accès à Amazon Redshift est effectué.

The screenshot shows the AWS CloudSearch interface. The search bar at the top contains the query 'redshift Serverless'. Below the search bar, there's a sidebar with sections for 'Services' (4), 'Fonctionnalités' (15), 'Ressources' (7), 'Publications de blog' (3), 'Documentation' (2075), 'Articles de connaissances' (107), 'Tutoriels' (2), and 'Marketplace' (863). A large red arrow points down to the 'Amazon Redshift' result card. The card includes the service icon, the name 'Amazon Redshift', a description ('Entreposage de données rapide, simple et rentable'), and a star icon.

The screenshot shows the 'Amazon Redshift sans serveur' dashboard. At the top, there's a header with the title 'Amazon Redshift sans serveur' and a 'Tableau de bord sans serveur' link. Below the header, there's a section titled 'Présentation de l'espace de noms' with a sub-section 'informations'. On the right, there's a 'Filtrer l'espace de noms' dropdown set to 'All namespaces'. A prominent orange button labeled 'Créer un groupe de travail' is highlighted with a red box and a red arrow pointing towards it. Below this button, there's a table with various metrics:

| Nombre total d'instantanés | Unités de partage des données dans mon compte | Unités de partage des données nécessitant une autorisation | Unités de partage des données d'autres comptes | Unités de partage des données nécessitant une association |
|----------------------------|---|--|--|---|
| 0 | 2 | 0 | 0 | 1 |

Après la création du groupe il faut cliquer sur le nom dans l'espace de groupes

The screenshot shows the AWS Glue Data Catalog interface. At the top, there is a blue banner with a message about registering Redshift table definitions to AWS Glue Data Catalog. Below the banner, the page title is "olistspace Informations". In the top right corner, there is a red circle around the "Actions" dropdown menu, and a red arrow points to the "Données de requête" (Request Data) button, which is also circled.

Ensute, il faut se rendre dans **Query Editor** afin d'ajouter des scripts, d'importer des tables et d'effectuer les transformations nécessaires.

Dans cette étape, de nombreuses transformations sont réalisées sur la base de données.

Avant toute transformation, il est nécessaire d'importer les données de S3 vers Redshift. Les codes correspondant à cette tâche figurent en annexe.

Une fois l'importation terminée, les données sont transformées, puis stockées dans un dossier **silver**. Les codes utilisés pour ces transformations sont également disponibles en annexe.

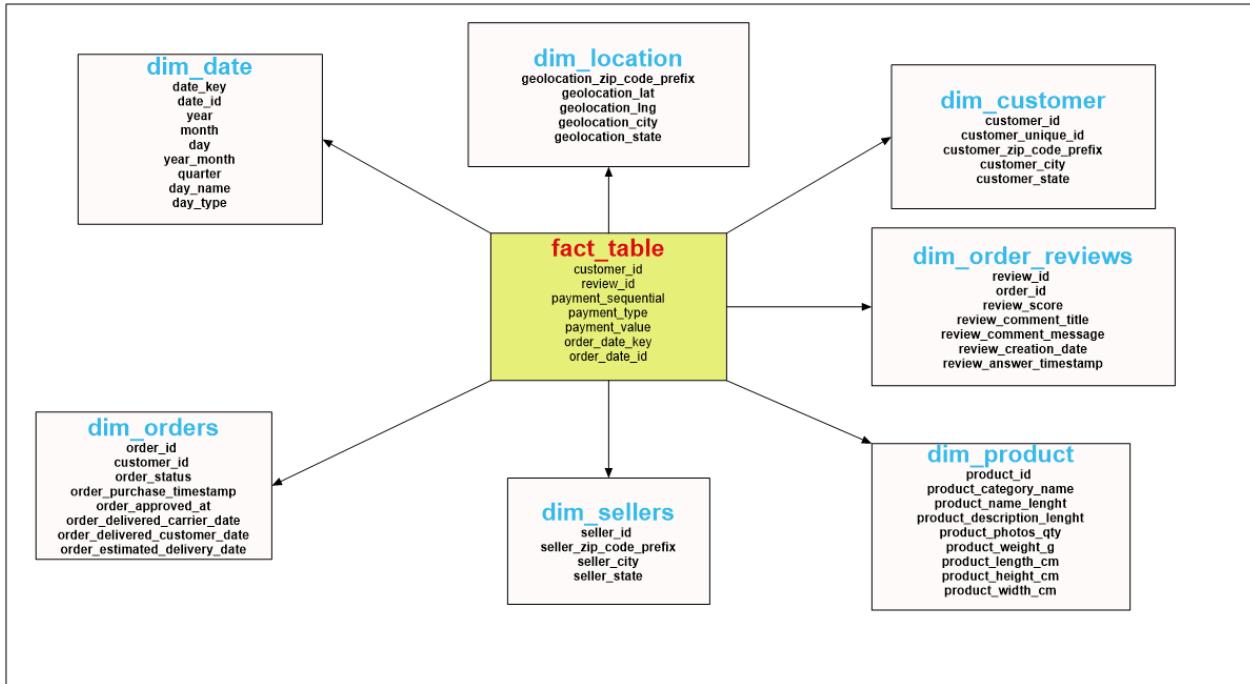
Le schéma du data warehouse est ensuite défini.

Après cela, un dossier **dw** est créé pour stocker les tables de dimensions et de faits, selon une modélisation en **étoile**.

Pourquoi le modèle étoile ?

Nous avons utilisé le **modèle en étoile** car il permet une **modélisation simple et lisible**, optimise les **performances des requêtes analytiques**, facilite les **jointures entre tables de faits et dimensions**, et est particulièrement adapté aux **outils de BI** comme Power BI pour l'analyse et la visualisation des données.

Figure 1: Modèle en étoile



Source : auteurs

II. PARTIE TEMPS REELLE DU PROJET

Dans le cadre de la construction de notre Data Warehouse basé sur le dataset Olist, nous avons mis en place une architecture capable de gérer les données en temps réel. Il est important de préciser que, dans notre contexte, nous distinguons deux approches du "temps réel". La première est une approche "**"Batch rapide"**", où les données arrivent par intervalles de quelques minutes, et la seconde est une version purement streaming, de l'ordre de la seconde ou de la milliseconde. Cette section de la documentation se concentre spécifiquement sur l'implémentation de la partie Batch, simulant un flux de commandes continu qui alimente notre entrepôt de données.

II.1 1. Le moteur de simulation et la gestion des stocks (DynamoDB)

Pour simuler un environnement de production réaliste, nous ne pouvions pas simplement injecter des données statiques. Nous avons donc créé un moteur de simulation basé sur Amazon

DynamoDB. La première composante est la table Sim_Config. Cette table agit comme le chef d'orchestre temporel de notre simulation. Elle contient une clé globale qui stocke l'heure virtuelle (simulated_time) et un facteur d'accélération (speed_factor). Cela nous permet de faire avancer le temps beaucoup plus vite que la réalité (par exemple, une minute réelle équivaut à une heure simulée), testant ainsi la robustesse du pipeline sur de longues périodes virtuelles en peu de temps.

The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with links like 'Tableau de bord', 'Tables', 'Explorer les éléments', 'Éditeur PartiQL', etc. The main area shows the 'Sim_Config' table selected from a list of two tables ('Sim_Config' and 'Sim_Inventory'). The table details page includes sections for 'Sélectionner la projection d'attributs' (choose attributes), 'Filtres - facultatif' (optional filters), and execution status ('Terminé - Éléments retournés : 1 · Éléments analysés : 1 · Efficacité : 100% · RCU consommées : 2'). Below this, a detailed view of the table's contents is shown, titled 'Table : Sim_Config – Éléments retournés (1)'. It lists one item: 'GLOBAL' with simulated_time '2018-01-03T11:0...' and speed_factor '60'. There are buttons for 'Actions' and 'Créer un élément'.

En parallèle, nous avons mis en place la table Sim_Inventory pour gérer les stocks de produits. Plutôt que de reprendre l'intégralité du dataset Olist, nous avons sélectionné un échantillon pertinent pour démontrer la faisabilité du concept. Cette table contient les identifiants produits (product_id), ainsi que leur niveau de stock actuel (stock_level), la catégorie et le prix. Nous avons utilisé un script Python, exécuté via l'AWS CLI, pour peupler initialement cette table. Ce script prépare le terrain pour que notre générateur de commandes puisse interagir avec des données concrètes.

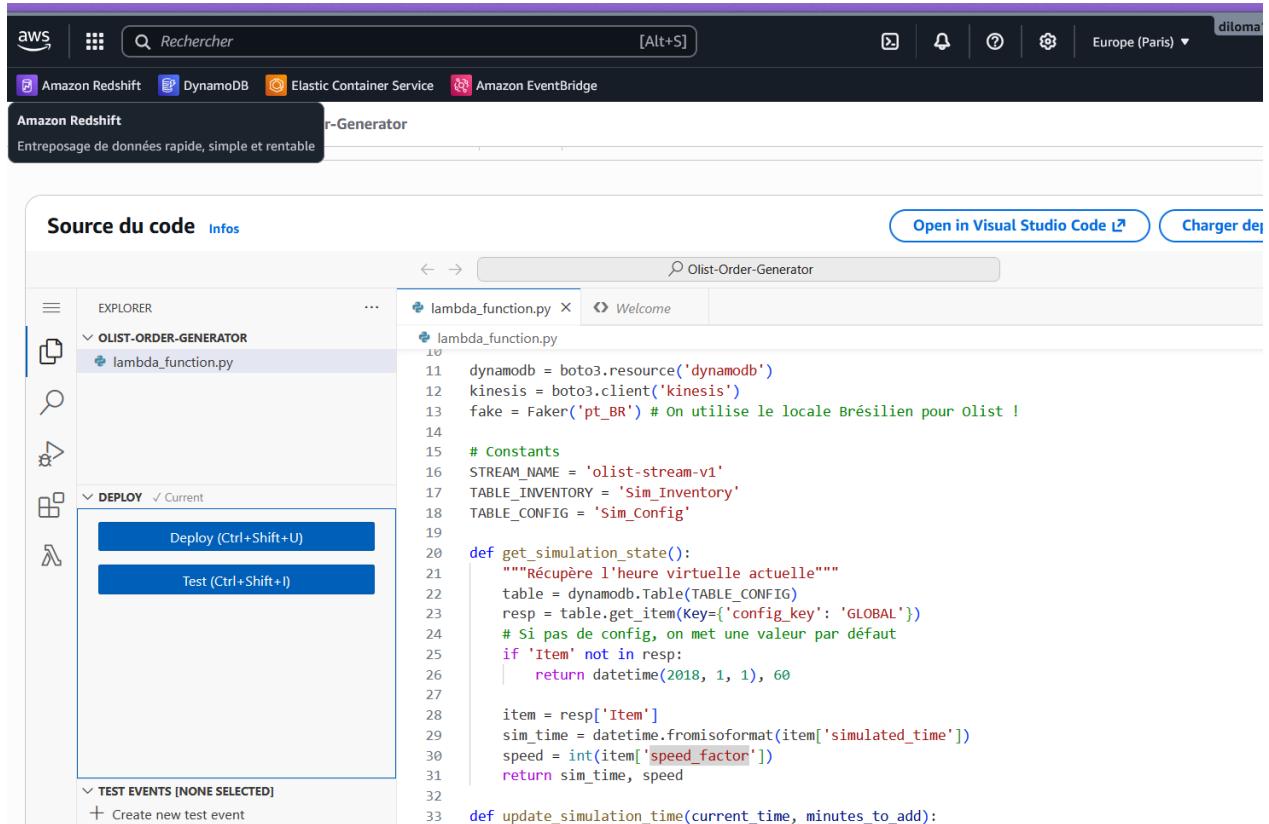
The screenshot shows the AWS DynamoDB console interface. On the left, there's a sidebar with navigation links like 'Tableau de bord', 'Tables', 'Explorer les éléments', 'Éditeur PartiQL', etc. The main area shows a list of tables: 'Sim_Config' (selected) and 'Sim_Inventory'. The 'Sim_Config' table details are shown on the right, including its schema: 'config_key (Chaîne)', 'simulated_time (Text)', and 'speed_factor (Number)'. A message at the bottom indicates the analysis was completed successfully with 100% efficiency.

II.2 Génération de commandes via AWS Lambda

Le cœur de notre ingestion de données réside dans une fonction AWS Lambda nommée "Order Generator". Cette fonction est déclenchée automatiquement toutes les minutes grâce à un planificateur Amazon EventBridge. La logique du code est conçue pour être performante et réaliste. Au démarrage, la fonction consulte la table Sim_Config pour savoir "quelle heure il est" dans la simulation. Pour éviter de surcharger la base de données et réduire les coûts, nous avons implémenté un système de mise en cache mémoire qui charge une liste limitée d'IDs de produits une seule fois par cycle, évitant ainsi de scanner toute la table d'inventaire à chaque commande.

La fonction génère ensuite un nombre aléatoire de commandes (entre 3 et 20) en utilisant la librairie Faker localisée pour le Brésil, afin de rester fidèle au contexte Olist. Point crucial de notre architecture : la gestion de la cohérence des données. Avant de valider une commande, le script effectue une transaction atomique sur DynamoDB pour décrémenter le stock. Si le stock est insuffisant, la commande est annulée, ce qui empêche de vendre des produits inexistant. Une fois la commande validée, elle est convertie en JSON et poussée vers notre flux de données. L'événement peut être activé ou désactivé, cela n'impacte pas le temps simulé et nous cela nous économise de l'argent sans qu'il y ait des troues. Firehose stocke le format parquet en format

année/mois/heure/jour/minutes en se referant au temps de la vrai vie , pour changer cela il vas faloir appeler une nouvelle function lambda et c'est couteux et comme nous ne nous servons pas directement de cette information (par contre si on voulais stocker dans ce style les data statique et les similé ensemble alors la oui) donc nous avons laissé intacte.



The screenshot shows the AWS Lambda function editor interface. The top navigation bar includes the AWS logo, a search bar, and links for Amazon Redshift, DynamoDB, Elastic Container Service, and Amazon EventBridge. The region is set to Europe (Paris). The main area is titled "Source du code" with an "Infos" tab selected. The code editor displays the following Python script:

```

EXPLORER
OLIST-ORDER-GENERATOR
lambda_function.py

lambda_function.py x Welcome
lambda_function.py

11 dynamodb = boto3.resource('dynamodb')
12 kinesis = boto3.client('kinesis')
13 fake = Faker('pt_BR') # On utilise le locale Brésilien pour olist !
14
15 # Constants
16 STREAM_NAME = 'olist-stream-v1'
17 TABLE_INVENTORY = 'Sim_Inventory'
18 TABLE_CONFIG = 'Sim_Config'
19
20 def get_simulation_state():
21     """Récupère l'heure virtuelle actuelle"""
22     table = dynamodb.Table(TABLE_CONFIG)
23     resp = table.get_item(Key={'config_key': 'GLOBAL'})
24     # Si pas de config, on met une valeur par défaut
25     if 'Item' not in resp:
26         return datetime(2018, 1, 1), 60
27
28     item = resp['Item']
29     sim_time = datetime.fromisoformat(item['simulated_time'])
30     speed = int(item['speed_factor'])
31     return sim_time, speed
32
33 def update_simulation_time(current_time, minutes_to_add):

```

The left sidebar shows the project structure under "EXPLORER" and deployment options like "Deploy (Ctrl+Shift+U)" and "Test (Ctrl+Shift+I)".

II.3 Ingestion, Buffering et Stockage

Une fois les commandes générées, elles entrent dans le pipeline d'ingestion via Amazon Kinesis Data Stream. Ce flux continu est ensuite consommé par Kinesis Firehose. Nous avons configuré Firehose avec des paramètres de tampon spécifiques : il accumule les données soit pendant 3 minutes, soit jusqu'à atteindre 5 Mo. Ce mécanisme de "micro-batch" est essentiel pour éviter de créer une multitude de trop petits fichiers, ce qui nuirait aux performances ultérieures.

Durant ce transit dans Firehose, une transformation majeure a lieu. Grâce à son intégration avec AWS Glue Data Catalog, Firehose convertit automatiquement les données JSON brutes au format Parquet avant de les écrire moyenant la création d'un data catalogue conforme au format json

attendu. Le format Parquet, étant en colonnes et compressé, est beaucoup plus performant pour l'analytique. Les fichiers résultats sont déposés dans notre bucket S3, dans un dossier désigné comme notre couche "Bronze" (sim-bronze).

The screenshot shows the 'Flux Firehose (1) infos' page. It displays a table with one row for the flux 'olist-firehose'. The columns include Nom (olist-firehose), Statut (Activ), Heure de ... (14 décembre...), Source (olist-stream...), Transform... (Non activé), Type de d... (Amazon S3), and URL de ... (project-oli...). There are buttons for Supprimer (Delete) and Crée un flux Firehose (Create a new Firehose stream).

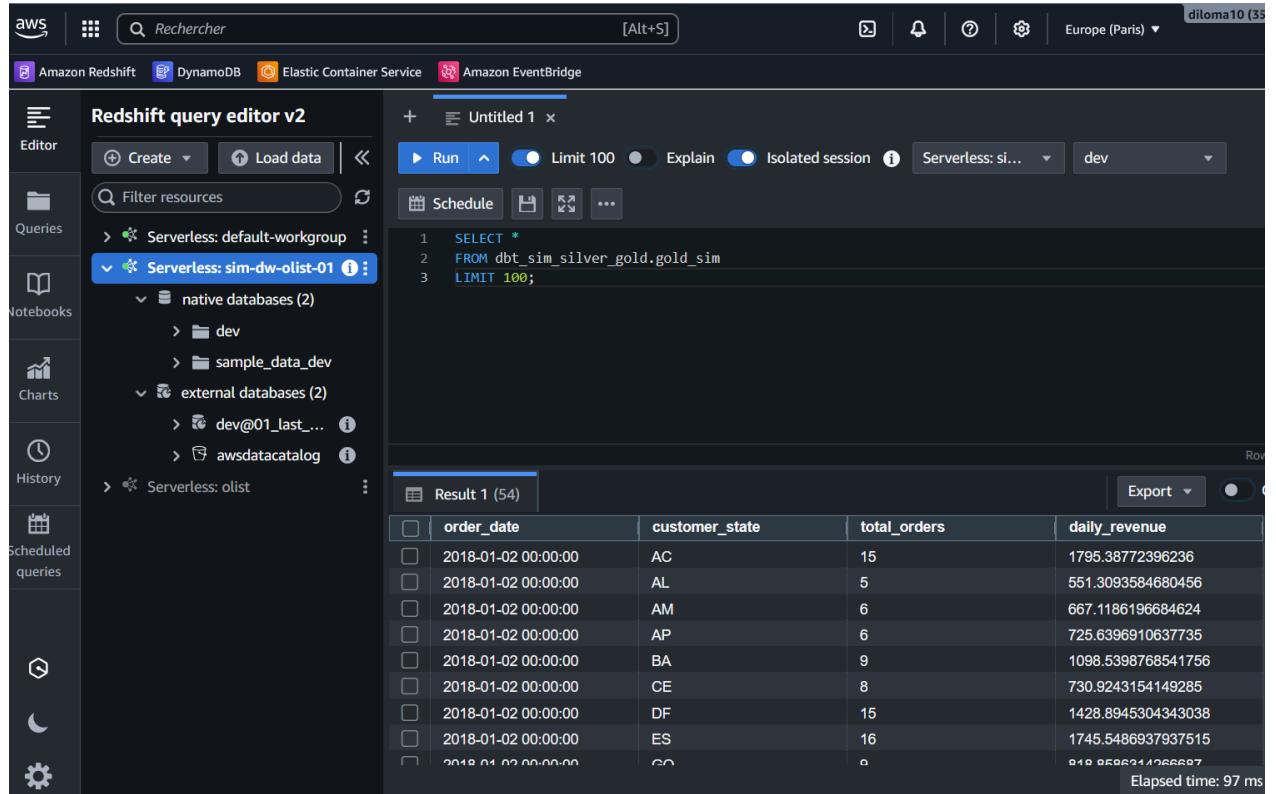
II.4 Intégration dans le Data Warehouse et Transformation (Redshift & dbt)

Pour rendre ces données accessibles à l'analyse, nous utilisons Amazon Redshift Serverless couplé à Redshift Spectrum. Grâce au schéma externe défini dans AWS Glue, Redshift peut lire directement les fichiers Parquet stockés dans S3 sans avoir besoin de les charger physiquement dans son propre stockage interne. Cela nous offre une flexibilité énorme et sépare les coûts de stockage des coûts de calcul.

The screenshot shows the 'Référentiels privés (1)' page. A blue banner at the top says 'La signature générée est désormais disponible' and 'Signez automatiquement vos images de conteneurs lors de leur envoi pour en vérifier l'authenticité et sécuriser votre chaîne d'approvisionnement.' Below is a table with one row for the repository 'olist_dw_dbt_run_sim'. The columns are Nom du référentiel (olist_dw_dbt_run_sim), URI (355142185625.dkr.ecr.eu-west-3.amazonaws.com/olist_dw_dbt_ru_n_sim), Créé à (16 décembre 2025, 00:11:41 (UTC-00)), Immuabilité des identifications (Mutable), and Type de chiffrement (AES-256). Buttons include Afficher les commandes Push, Supprimer, Actions, and Crée un réfé.

La transformation de la donnée brute (Bronze) vers des modèles nettoyés (Silver) et agrégés (Gold) est assurée par dbt (data build tool). Pour automatiser cette étape, nous avons conteneurisé notre projet dbt et stocké l'image dans Amazon ECR. L'orchestration est gérée par AWS Step Functions. Le processus est entièrement événementiel : dès qu'EventBridge détecte qu'un nouveau fichier a

été déposé dans le dossier S3 Bronze, il déclenche la Step Function. Celle-ci lance alors le conteneur ECR qui exécute les transformations dbt. Pour sécuriser l'accès à la base de données, le conteneur récupère les identifiants de connexion de manière sécurisée via AWS Secrets Manager.



The screenshot shows the AWS Redshift query editor interface. On the left, there's a sidebar with icons for Editor, Queries, Notebooks, Charts, History, and Scheduled queries. The main area has tabs for 'Untitled 1' and 'Serverless: sim-dw-olist-01'. The 'Serverless: sim-dw-olist-01' tab is active, showing a query in the editor pane:

```

1  SELECT *
2  FROM dbt_sim_silver_gold.gold_sim
3  LIMIT 100;

```

The results pane below shows a table titled 'Result 1 (54)' with the following data:

| | order_date | customer_state | total_orders | daily_revenue |
|---|---------------------|----------------|--------------|--------------------|
| 1 | 2018-01-02 00:00:00 | AC | 15 | 1795.38772396236 |
| 2 | 2018-01-02 00:00:00 | AL | 5 | 551.3093584680456 |
| 3 | 2018-01-02 00:00:00 | AM | 6 | 667.1186196684624 |
| 4 | 2018-01-02 00:00:00 | AP | 6 | 725.6396910637735 |
| 5 | 2018-01-02 00:00:00 | BA | 9 | 1098.5398768541756 |
| 6 | 2018-01-02 00:00:00 | CE | 8 | 730.9243154149285 |
| 7 | 2018-01-02 00:00:00 | DF | 15 | 1428.8945304343038 |
| 8 | 2018-01-02 00:00:00 | ES | 16 | 1745.5486937937515 |
| 9 | 2018-01-02 00:00:00 | GO | 0 | 0.000000000000000 |

At the bottom right of the results pane, it says 'Elapsed time: 97 ms'.

II.5 Visualisation (Power BI)

La finalité de ce pipeline est la visualisation des données. Nous avons connecté Power BI directement à notre cluster Redshift. Pour permettre cette connexion, nous avons dû configurer un utilisateur dédié sur Redshift et ajuster les paramètres du VPC (Virtual Private Cloud) et des groupes de sécurité pour autoriser le trafic entrant depuis Power BI. Cela permet aux parties prenantes de suivre l'évolution des commandes et des stocks avec une latence de seulement quelques minutes par rapport à la génération des données.

Les transformation pour avoir le silver et le gold data sont dans le dossier models/ sur <https://github.com/Ben10-som/Data-warehouse-for-sales-analysis->

Dans la prochaine version nous passerons par Cloudformation ou terraform pour refaire le projet et y inclure la partie temps réelle.

The screenshot shows the Power BI Desktop interface with the 'Navigator' pane open. The table selected is 'silver_sim'. The first few rows of the table are:

| | order_purchase_timestamp | product_id |
|---------------------------|--------------------------|-----------------------------|
| d-4c3e-bc35-71ddfbaff9c | 02/01/2018 14:00:00 | eaa99c9c7061d17bbfe7e85d842 |
| 34-45ca-b409-0659649134e | 02/01/2018 14:00:00 | f9b3ae63a74b9f948a66d00be |
| b-439a-b01e-40d5628e0bd3 | 02/01/2018 14:00:00 | 66a5f1151e8e4215b3c3e659 |
| 1-4011-aa40-bd57237774b4 | 02/01/2018 15:00:00 | 06b55f1f3e305bd276cabbb4d |
| 9-497a-b03e-9b44871d786 | 02/01/2018 15:00:00 | 5fd2295eca1c3821da53a52da |
| t-4c01-998f-fa7417f636 | 02/01/2018 16:00:00 | 2c7dcb2b1eccc3d8e0ff2d0e |
| 2-4760-9eb0-a5aa5d5d7d31 | 02/01/2018 16:00:00 | fef526cd8279b98e8e3d3e8b |
| 6-4f82-80f9-b2af9b38c701 | 02/01/2018 16:00:00 | 13c7f503c7a0c12e243d2e8bb |
| 3-4907-aecb-541608942d8 | 02/01/2018 16:00:00 | 244811270c7c86bcece6e9340f |
| i-4bd7-88b6-62818d5c7996 | 02/01/2018 18:00:00 | 07d8e1a14d008274672847d88c |
| b-432e-bc14-59e7ca09dc | 02/01/2018 18:00:00 | a32a25c59034c4b0c3a031798e |
| 7-45fc-8300-65519a15a76c | 02/01/2018 18:00:00 | 9750263daea3988ed39eeb271 |
| c-4c9d-9209-dec3f1f63baa | 02/01/2018 19:00:00 | 3b91132a7eaba17b615e8b8d |
| 14-4f57-b4fb-bd1d1f19234 | 02/01/2018 19:00:00 | dfdd4a0d709e508749a28d31b |
| 1-492a-8c02-3f0a0d296f64 | 02/01/2018 19:00:00 | ce040d97aa96a49494933f3d4a' |
| 1b-4fe3-8210-acd97fe59040 | 02/01/2018 19:00:00 | 1177a4988372a9eff08364c4d7 |
| 77-455e-9521-df186a789ac | 02/01/2018 20:00:00 | 7d2e1a2c7096151cd14274d42 |
| i-417f-a07b-5fca184395be | 02/01/2018 20:00:00 | 1823142d83a24b619eb0b0088 |
| 30-4e04-a6ad-05d59d9dacf | 02/01/2018 20:00:00 | fbad08ee4c3aa79a86833c838 |
| 1-41fb-9c24-8af423a2bf9 | 03/01/2018 03:00:00 | 2942e500c8ae1e055a5d893 |
| ib-4ce8-a786-41a726793a7d | 03/01/2018 03:00:00 | 33c59c2b23093baae8f760a49 |
| z-4e70-9ee4-24bf273583c3 | 03/01/2018 03:00:00 | 08ff026aae73da385cf1342f |
| id-4eac-bddc-2600e18e79b | 03/01/2018 03:00:00 | fe4dece0c663ba1f9f2a0d4d3b |

The screenshot shows the Power BI Desktop interface with the 'Navigator' pane open. The table selected is 'gold_sim'. The first few rows of the table are:

| order_date | customer_state | total_orders | daily_revenue |
|---------------------|----------------|--------------|---------------|
| 02/01/2018 00:00:00 | "AP" | 6 | 725,6396911 |
| 02/01/2018 00:00:00 | "PA" | 7 | 471,8314998 |
| 02/01/2018 00:00:00 | "RO" | 7 | 896,2623351 |
| 02/01/2018 00:00:00 | "AL" | 5 | 551,3093585 |
| 02/01/2018 00:00:00 | "RN" | 11 | 960,5998842 |
| 02/01/2018 00:00:00 | "GO" | 9 | 818,8586314 |
| 02/01/2018 00:00:00 | "MA" | 6 | 633,9273882 |
| 02/01/2018 00:00:00 | "AM" | 6 | 667,1186197 |
| 02/01/2018 00:00:00 | "SP" | 17 | 183,071607 |
| 02/01/2018 00:00:00 | "BA" | 9 | 1086,539877 |
| 02/01/2018 00:00:00 | "TO" | 10 | 1099,36265 |
| 02/01/2018 00:00:00 | "MG" | 12 | 1140,44655 |
| 02/01/2018 00:00:00 | "RR" | 10 | 952,1066565 |
| 02/01/2018 00:00:00 | "PI" | 8 | 874,3274784 |
| 02/01/2018 00:00:00 | "ES" | 16 | 1745,548694 |
| 02/01/2018 00:00:00 | "SE" | 14 | 1468,197146 |
| 02/01/2018 00:00:00 | "RJ" | 13 | 1553,920756 |
| 02/01/2018 00:00:00 | "DF" | 15 | 1428,89453 |
| 02/01/2018 00:00:00 | "MS" | 8 | 612,9592916 |
| 02/01/2018 00:00:00 | "PB" | 9 | 907,9103784 |
| 02/01/2018 00:00:00 | "PR" | 13 | 1294,728624 |
| 02/01/2018 00:00:00 | "RS" | 18 | 1743,972284 |
| 02/01/2018 00:00:00 | "CE" | 8 | 730,9243154 |
| 02/01/2018 00:00:00 | "MT" | 13 | 1170,913866 |

II.6 Architecture Streaming (Temps Réel Pur) et Perspectives

Comme évoqué dans l'introduction, notre analyse distinguait deux niveaux de temporalité : le micro-batch (actuellement en place) et le streaming pur (latence de l'ordre de la seconde). Pour cette itération du projet, nous avons fait le choix conscient de ne pas implémenter la branche de streaming pur. Cette décision a été dictée par des contraintes pragmatiques de temps de développement et de budget (optimisation des coûts des services Cloud), l'architecture actuelle suffisant à démontrer la robustesse de l'ingestion de données.

Toutefois, l'architecture cible pour ce besoin spécifique est déjà théorisée. L'objectif serait de réduire drastiquement la latence en éliminant l'étape de stockage intermédiaire sur Amazon S3 ("Sim-Bronze"). Dans cette configuration, nous utiliserions l'intégration native Streaming Ingestion d'Amazon Redshift. Concrètement, le flux Kinesis Firehose ne déposerait plus de fichiers Parquet, mais injecterait les données directement dans les tables Redshift en temps réel. Cela permettrait de contourner les délais liés à la bufferisation des fichiers et aux triggers événementiels, offrant ainsi une vision quasi instantanée des transactions pour des cas d'usage critiques.

Annexe: Code pour transforme des bases csv en format parquet

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

# Initialisation du job Glue

args = getResolvedOptions(sys.argv, ["JOB_NAME"])
sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args["JOB_NAME"], args)

#Emplacements S3
input_path = "s3://your-bucket-name/bronze/"      # tes CSV Olist
output_path = "s3://your-bucket-name/bronze-parquet/" # dossier Parquet à créer

# Lecture des CSV

df_csv = (
    spark.read
        .option("header", "true")
        .option("inferSchema", "true") # (optionnel)
        .option("sep", ",")
        .csv(input_path)
)

print("Nombre de lignes chargées :", df_csv.count())
print("Schéma détecté :")
df_csv.printSchema()

# Écriture en Parquet
(
    df_csv.write
        .mode("overwrite")      # écrase si le dossier existe
        .option("compression", "snappy")
        .parquet(output_path)
)

print("Transformation CSV → Parquet terminée.")

job.commit()
```