DATA ENGINEERING TASK

This presentation introduces an automated Extract, Transform, Load (ETL) pipeline empowered by cloud technology, designed to streamline data processing and enable efficient insights generation.

Fredy Davis

CONTENTS

- Objective
- Recommendation For Additional Feature
- Cloud Infrastructure
- Cloud Architecture
- Data Model
- Business Logic
- ETL Script
- Infrastructure As Code
- Conclusion

OBJECTIVE

- Data-driven approach for effective user clustering.
- Involves selecting key features and building clustering models.
- Empowered by cloud technology for streamlined data processing.
- Automates Extract, Transform, Load (ETL) procedures.
- Design a data pipeline, which provides a data mart with the next features for each customer:
    - age
    - country
    - state
    - nearest distribution center
    - product return rate in the last year
    - customer profit level in the last year
        - Level 1 if the customer has bought products for 50$ or less,
        - Level 2 - more than 50 but less than 150$,
        - Level 3 - more than 150$.

RECOMMENDATION FOR ADDITIONAL FEATURE

Most Traffic Source

- **Most traffic source**: Incorporate metrics related to customer engagement, such as social media platform, email etc for better marketing campaigns.
- Incorporating these features deepens insights into user behaviour.
- Enables personalized and effective marketing strategies.

CLOUD INFRASTRUCTURE

Amazon Web Services (AWS)

AWS S3

- Scalable object storage for data.

AWS GLUE

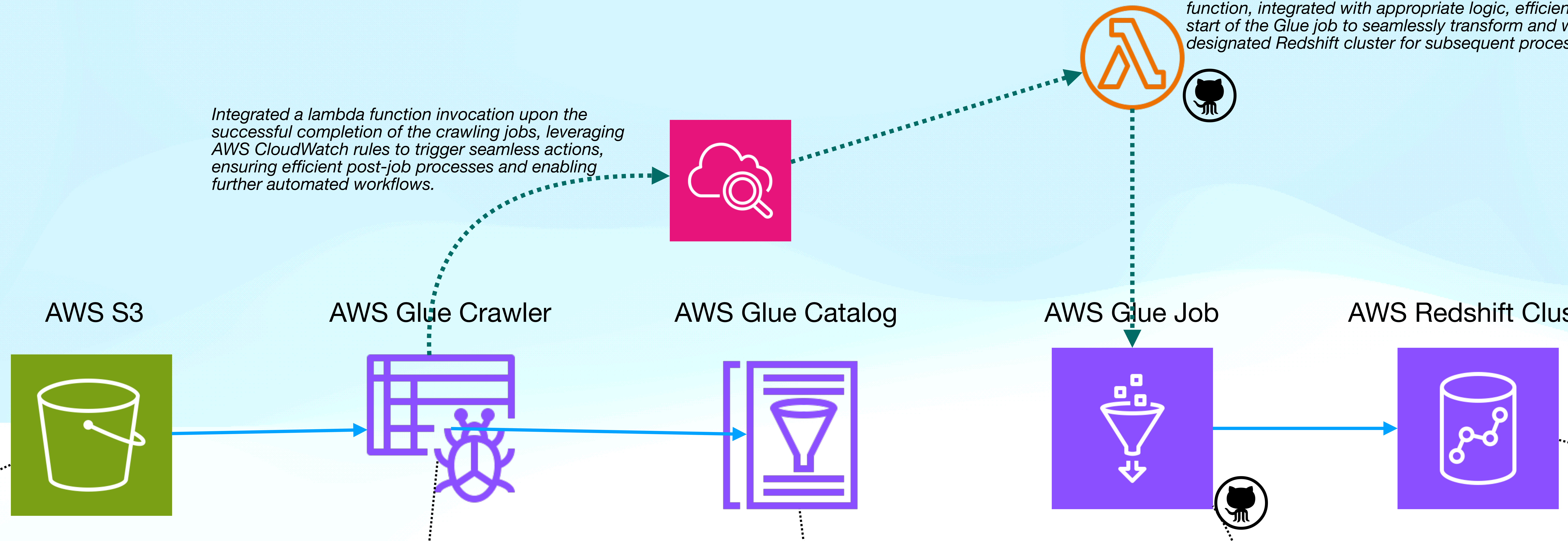
- ETL (Extract, Transform, Load) service for data preparation.
- Glue Catalog: Metadata repository for organizing data.
- Glue Crawler: Automatically discovers data and creates metadata.
- Glue Job: Executes ETL jobs to process and transform data.

AWS REDSHIFT

- Fully-managed data warehouse for analytics.

**To access the relevant code files within the repository, please click on the corresponding GitHub icon.*

CLOUD ARCHITECTURE



Integrated a lambda function invocation upon the successful completion of the crawling jobs, leveraging AWS CloudWatch rules to trigger seamless actions, ensuring efficient post-job processes and enabling further automated workflows.

Implemented a trigger mechanism employing a Lambda function using python, to initialise the AWS Glue ETL job. This Lambda function, integrated with appropriate logic, efficiently triggers the start of the Glue job to seamlessly transform and write data into the designated Redshift cluster for subsequent processing.

Firstly, we'll establish an S3 bucket on Amazon Web Services (AWS). Within this bucket, we'll create folders to structure the data appropriately. Following this organization, we'll proceed to upload your shopping data, formatted as a CSV file, into the designated folder within the S3 bucket.

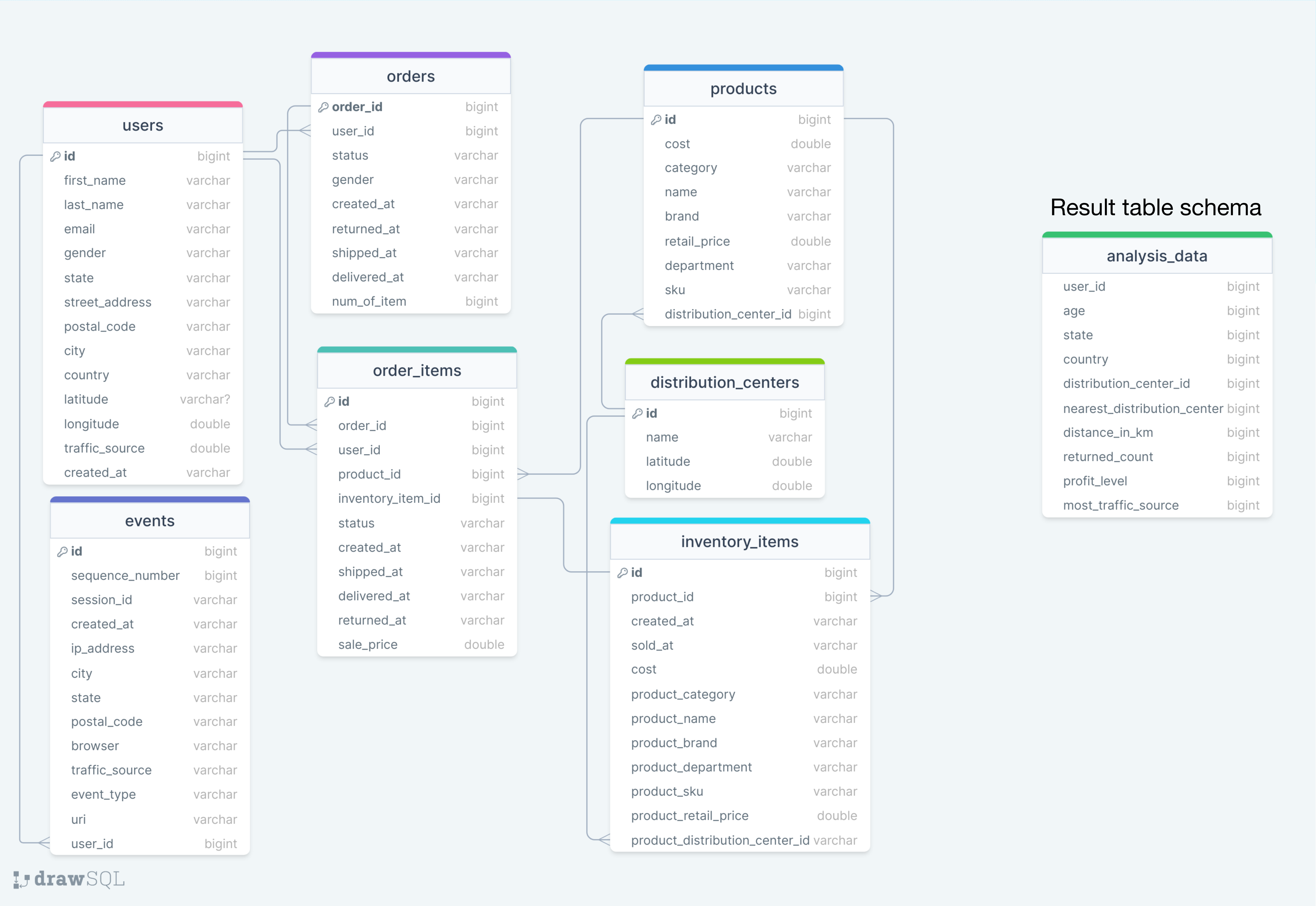
Set up a crawler to fetch data from an S3 bucket. Choose the IAM role that grants permissions to the crawler to access the S3 bucket. Schedule the crawler to run weekly.

Create a database within the data catalog to store the data fetched by the crawler.

Compose a Glue job to execute a business logic SQL query on data stored within catalog tables, enabling data transformation.

Save the transformed data into a Redshift cluster for future analysis. Enable AWS Glue with the required permissions to write data into the Redshift cluster.

DATA MODEL



users	
id	bigint
first_name	varchar
last_name	varchar
email	varchar
gender	varchar
state	varchar
street_address	varchar
postal_code	varchar
city	varchar
country	varchar
latitude	varchar?
longitude	double
traffic_source	double
created_at	varchar

events	
id	bigint
sequence_number	bigint
session_id	varchar
created_at	varchar
ip_address	varchar
city	varchar
state	varchar
postal_code	varchar
browser	varchar
traffic_source	varchar
event_type	varchar
uri	varchar
user_id	bigint

orders	
order_id	bigint
user_id	bigint
status	varchar
gender	varchar
created_at	varchar
returned_at	varchar
shipped_at	varchar
delivered_at	varchar
num_of_item	bigint

order_items	
id	bigint
order_id	bigint
user_id	bigint
product_id	bigint
inventory_item_id	bigint
status	varchar
created_at	varchar
shipped_at	varchar
delivered_at	varchar
returned_at	varchar
sale_price	double

products	
id	bigint
cost	double
category	varchar
name	varchar
brand	varchar
retail_price	double
department	varchar
sku	varchar
distribution_center_id	bigint

distribution_centers	
id	bigint
name	varchar
latitude	double
longitude	double

inventory_items	
id	bigint
product_id	bigint
created_at	varchar
sold_at	varchar
cost	double
product_category	varchar
product_name	varchar
product_brand	varchar
product_department	varchar
product_sku	varchar
product_retail_price	double
product_distribution_center_id	varchar

Result table schema

analysis_data	
user_id	bigint
age	bigint
state	bigint
country	bigint
distribution_center_id	bigint
nearest_distribution_center	bigint
distance_in_km	bigint
returned_count	bigint
profit_level	bigint
most_traffic_source	bigint

BUSINESS LOGIC

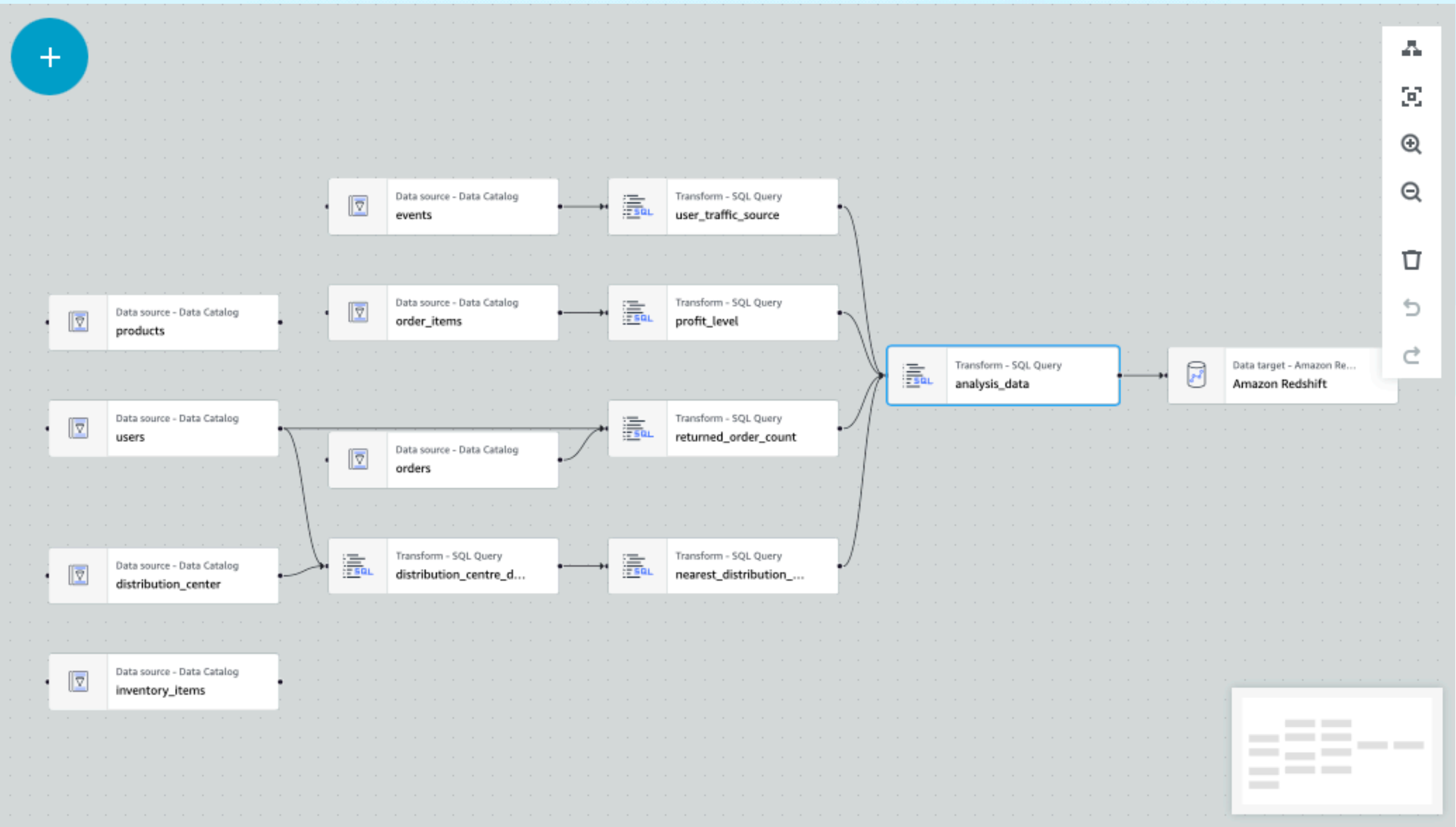
```sql
WITH NearestDistribution AS (
    -- Query 1: Find Nearest Distribution Center for Each User
    SELECT
        user_id,
        age,
        state,
        country,
        dc_id,
        nearest_distribution_center,
        distance_in_km
    FROM (
        SELECT
        u.id as user_id,
        u.age,
        u.state,
        u.country,
        dc.id as dc_id,
        dc.name as nearest_distribution_center,
        u.latitude as user_lat,
        u.longitude as user_lon,
        dc.latitude as dc_lat,
        dc.longitude as dc_lon,
        ROUND(
            6371 * 2 * ASIN(
                SQRT(
                    POWER(SIN(RADIANS(dc.latitude - u.latitude) / 2), 2) +
                    COS(RADIANS(u.latitude)) * COS(RADIANS(dc.latitude)) *
                    POWER(SIN(RADIANS(dc.longitude - u.longitude) / 2), 2)
                )
            ),
            2
        ) AS distance_in_km,
        ROW_NUMBER() OVER(PARTITION BY u.id ORDER BY distance_in_km) as rn
    FROM
        "awsdatacatalog"."shopping"."users" as u
    CROSS JOIN
        "awsdatacatalog"."shopping"."distribution_centers" as dc
    ORDER BY
        distance_in_km
    ) subquery
    WHERE rn = 1
),
ReturnedOrdersCount AS (
    -- Query 2: Count of Returned Orders in 2022 by User
    SELECT u.id AS user_id, COUNT(*) AS returned_count
    FROM "awsdatacatalog"."shopping"."orders" o
    JOIN "awsdatacatalog"."shopping"."users" u ON o.user_id = u.id
    WHERE o.status = 'Returned'
      AND EXTRACT(YEAR FROM TO_TIMESTAMP(o.returned_at, 'YYYY-MM-DD HH24:MI:SS')) = 2022
    GROUP BY u.id
),
ProfitLevel AS (
    -- Query 3: Calculate profit level for each user in 2022
    SELECT
        user_id AS user_id,
        CASE
            WHEN total_purchase <= 50 THEN 1
            WHEN total_purchase > 50 AND total_purchase < 150 THEN 2
            ELSE 3
        END AS profit_level
    FROM (
        SELECT
            oi.user_id,
            SUM(sale_price) AS total_purchase
        FROM
            "awsdatacatalog"."shopping"."order_items" AS oi
        WHERE
            EXTRACT(YEAR FROM TO_TIMESTAMP(oi.created_at, 'YYYY-MM-DD HH24:MI:SS')) = 2022 --
 Filter orders for the year 2022
            AND oi.status = 'Complete'
        GROUP BY
            oi.user_id
    ) AS purchase_summary
),
UserTrafficSource AS (
    -- Query 4: Determine most frequent traffic source for each user
    SELECT
        user_id,
        traffic_source AS most_traffic_source
    FROM (
        SELECT
            e.user_id,
            e.traffic_source,
            ROW_NUMBER() OVER(PARTITION BY user_id ORDER BY COUNT(*) DESC) AS source_rank
        FROM
            "awsdatacatalog"."shopping"."events" e
        GROUP BY
            user_id, traffic_source
    ) AS source_ranking
    WHERE source_rank = 1
)
-- Joining the results of all queries based on user_id
SELECT
    nd.user_id,
    nd.age,
    nd.state,
    nd.country,
    nd.dc_id,
    nd.nearest_distribution_center,
    nd.distance_in_km,
    roc.returned_count,
    pl.profit_level,
    uts.most_traffic_source
FROM NearestDistribution nd
LEFT JOIN ReturnedOrdersCount roc ON nd.user_id = roc.user_id
LEFT JOIN ProfitLevel pl ON nd.user_id = pl.user_id
LEFT JOIN UserTrafficSource uts ON nd.user_id = uts.user_id
ORDER BY nd.user_id;
```

ETL SCRIPT

- ETL pipeline script designed specifically for loading data from an S3 bucket

- Transformation executed through SQL queries for data processing

- Resultant data stored in a Redshift target database



AWS Glue Studio's Visual Editor orchestrating the ETL pipeline from source through transformation to target

INFRASTRUCTURE AS CODE

Terraform

- Terraform allows defining and managing infrastructure in a declarative way using code, enhancing repeatability and consistency.
- Terraform used to create infrastructure components such as S3 buckets, Glue resources, and Redshift clusters.
- IAM policies managed and configured via Terraform to ensure secure access and permissions for these created resources.

CONCLUSION

- Accomplished the establishment of an efficient ETL pipeline on AWS, ensuring seamless data flow.
- Leveraged Terraform for the development of robust and scalable infrastructure.
- Introduced the innovative 'Most Traffic Source' feature, enabling tailored and impactful marketing approaches.
- Implemented automation via AWS CloudWatch event and Lambda function, streamlining the pipeline's execution to occur weekly for enhanced efficiency.

RESOURCES

- GitHub Repository Link
- https://github.com/fredythekkekkara/etl-task