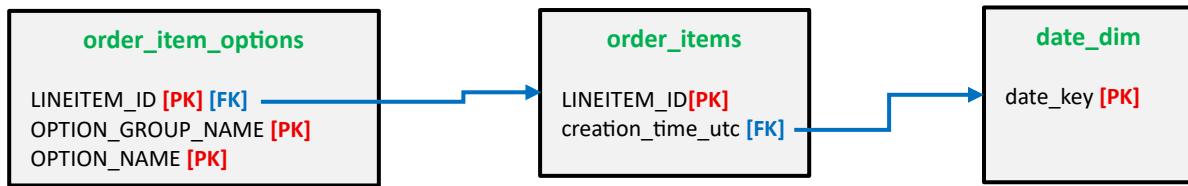


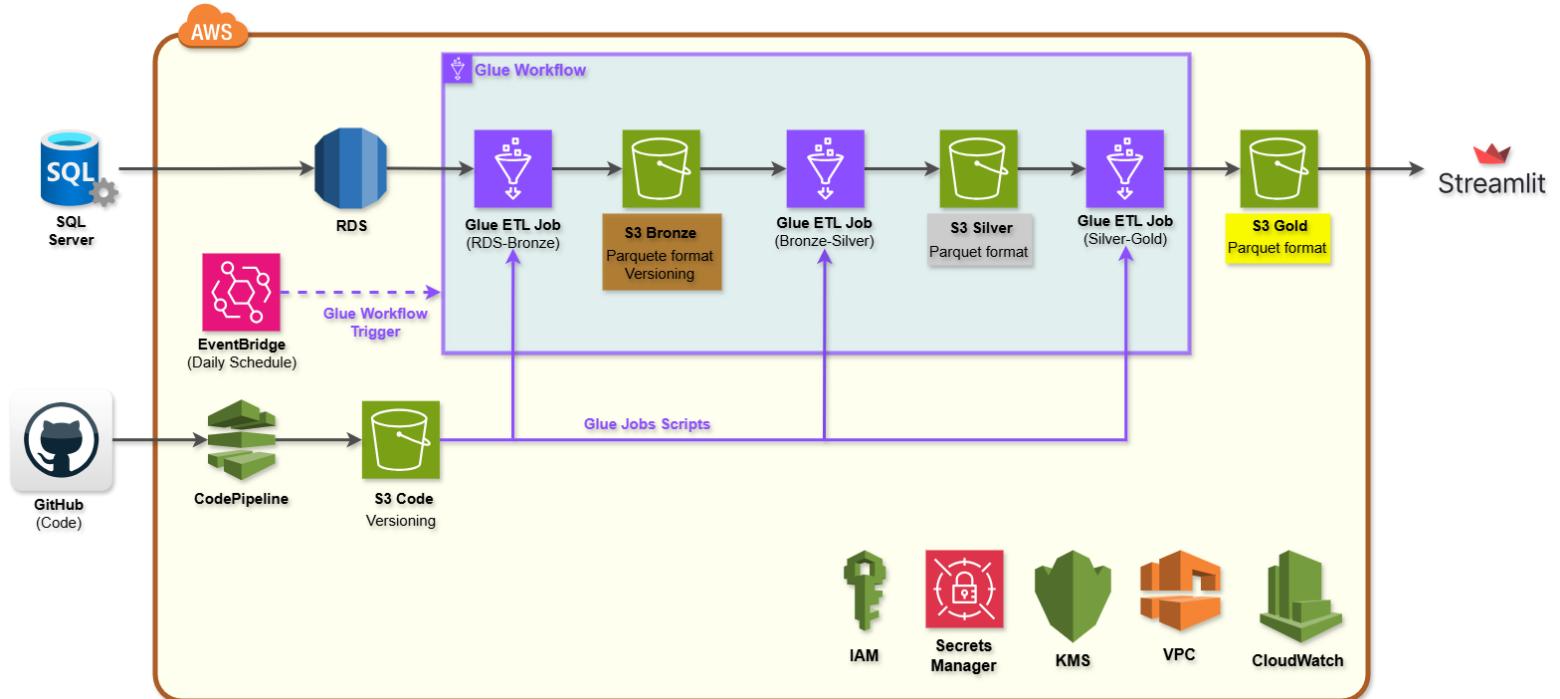
Project 3: Business Insights Assessment

Overview:

After the exploratory analysis of the three source data files (`order_items.csv`, `order_item_options.csv`, and `date_dim.csv`), I have prepared the proposal and got the approval from Jay. Now, here is the data model pipeline, CI/CD operation and orchestration from data ingestion, doing ELT jobs, until visualization to support business insights, customer analytics, and reporting requirements.



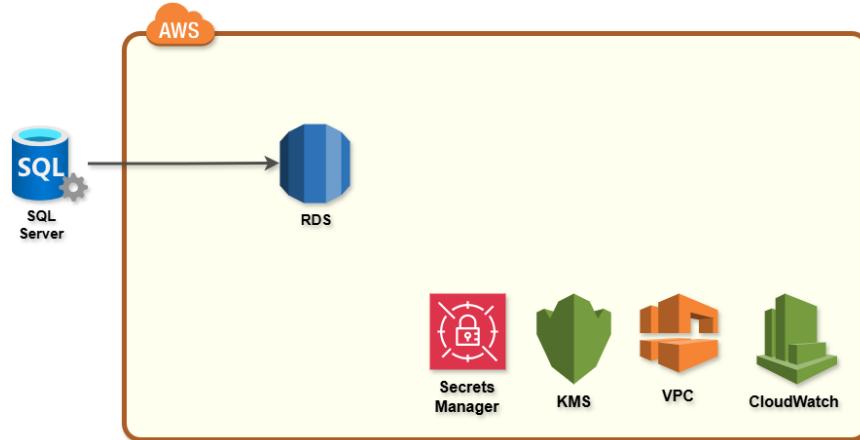
Here is the whole pipeline, and in the following, I am going to state each section in detail.



Step 1 – Data Ingestion (SQL Server to RDS)

1-1- Objective

The objective of this step is to ingest raw transactional and dimensional data from my local SQL Server database into an Amazon RDS database. This is best solution, as S3 can connect to RDS easily using VPC inside the AWS. We need S3 for Bronze, Silver, and Gold layer as DataLake, and S3 cannot directly get data from my Local SQL Server easily or privately.



1-2- Connectivity

The **Amazon VPC** provides secure network connectivity between AWS RDS and the SQL Server database, ensuring that data transfer occurs within a controlled private network. Inside AWS, I have created a private VPC with Ipv4 CIDR of **172.31.0.0/16**. I don't want to use the default VPC.

The screenshot shows the AWS VPC dashboard with the following details:

Name	VPC ID	State	Encryption control	Block Public Access	IPv4 CIDR
-	vpc-0ebfed9dc98bbc58a	Available	-	Off	172.31.0.0/16

Below the table, the details for the selected VPC (vpc-0ebfed9dc98bbc58a) are shown:

Details	Resource map	CIDRs	Flow logs	Tags	Integrations
Details					
VPC ID vpc-0ebfed9dc98bbc58a	State Available	Block Public Access Off	DNS hostnames Enabled		
DNS resolution Enabled	Tenancy default	DHCP option set dopt-079c0a96596a5003a	Main route table rtb-0f7a2fb0e0x491f65		
Main network ACL acl-0051ad57be7899186	Default VPC No	IPV4 CIDR 172.31.0.0/16	IPv6 pool -		

Then, I have created two public (**project3-public-subnet**) and private (**project3-private-subnet**) subnets for the VPC, in different zones of **us-east-1a** and **us-east-1b**. I will use the public subnet for NAT and SMSS networking, and the private for RDS and Glue JDBC job.

The first screenshot shows the 'Subnets' page with two subnets listed:

Name	Subnet ID	State	VPC	Block Public Access	IPv4 CIDR	Availability Zone	Network border group
project3-private-subnet	subnet-0e99ace8108589d3e	Available	vpc-0ebfed9dc98bbc58a	Off	172.31.1.0/24	use1-az6 (us-east-1b)	us-east-1
project3-public-subnet	subnet-0d4ad4416daaebe37e	Available	vpc-0ebfed9dc98bbc58a	Off	172.31.0.0/24	use1-az4 (us-east-1a)	us-east-1

The second screenshot shows the details for the public subnet:

Details	Subnet ID	Subnet ARN	State	Block Public Access
Subnet ID	subnet-0d4ad4416daaebe37e	arn:aws:ec2:us-east-1:000185048470:subnet/subnet-0d4ad4416daaebe37e	Available	Off
IPv4 CIDR	172.31.0.0/24		IPv6 CIDR	—
Availability Zone	use1-az4 (us-east-1a)		VPC	vpc-0ebfed9dc98bbc58a
Network ACL	acl-0051ad55be7899186		Auto-assign public IPv4 address	Yes
			Route table	rtb-0ff40659f1460a894 project3-public-rt
			Auto-assign IPv6 address	No

The third screenshot shows the details for the private subnet:

Details	Subnet ID	Subnet ARN	State	Block Public Access
Subnet ID	subnet-0e99ace8108589d3e	arn:aws:ec2:us-east-1:000185048470:subnet/subnet-0e99ace8108589d3e	Available	Off
IPv4 CIDR	172.31.1.0/24		IPv6 CIDR	—
Availability Zone	use1-az6 (us-east-1b)		VPC	vpc-0ebfed9dc98bbc58a
Network ACL	acl-0051ad55be7899186		Auto-assign public IPv4 address	No
			Route table	rtb-0fc1d10e10217c6d2 project3-private-rt
			Auto-assign IPv6 address	No

After that, I need to connect SQL Server to AWS via VPC, then I need to create an Internet Gateway and design the Route Table based on public subnet of VPC. First, I created an Internet Gateway, named **project3-igw**.

The screenshot shows the 'Internet gateways' page with one internet gateway listed:

Details	Internet gateway ID	State	VPC ID
Internet gateway ID	igw-007436ac9f2a92a32	Attached	vpc-0ebfed9dc98bbc58a

The 'Tags' section shows:

Key	Value
project	3
Name	project3-igw

Then, I have created two different Route Tables, one for private subnet (**project3-private-rt**) and the other for public subnet (**project3-public-rt**). There is a main route table by default in the AWS VPC, which we will not use it.

The screenshot shows the AWS VPC Route tables page. On the left, there's a sidebar with 'VPC dashboard' and 'Virtual private cloud' sections. The main area displays 'Route tables (3) Info' with a search bar and a table. The table has columns for Name, Route table ID, Explicit subnet assoc..., Edge associations, Main, and VPC. The rows show:

Name	Route table ID	Explicit subnet assoc...	Edge associations	Main	VPC
project3-public-rt	rtb-0ff40659f1460a894	subnet-0d4ad416daeb...	-	No	vpc-0ebfed9dc98bbc58a
project3-private-rt	rtb-0fc1d10e10217c6d2	subnet-0e99ace8108589...	-	No	vpc-0ebfed9dc98bbc58a
-	rtb-0f7a2fb0e0c491f65	-	-	Yes	vpc-0ebfed9dc98bbc58a

Below the table is a 'Select a route table' button.

If I click on each route table, I can see these information below:

This screenshot shows the details for the route table 'rtb-0ff40659f1460a894 / project3-public-rt'. It includes sections for 'Details' (Main, VPC), 'Routes' (2), and 'Subnet associations', 'Edge associations', 'Route propagation', and 'Tags' tabs.

Routes (2)

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	ipw-007436ac9f2a92a52	Active	No	Create Route
172.31.0.0/16	local	Active	No	Create Route Table

This screenshot shows the details for the route table 'rtb-0fc1d10e10217c6d2 / project3-private-rt'. It includes sections for 'Details' (Main, VPC), 'Routes' (2), and 'Subnet associations', 'Edge associations', 'Route propagation', and 'Tags' tabs.

Routes (2)

Destination	Target	Status	Propagated	Route Origin
0.0.0.0/0	nat-088845ea838d1c1d	Active	No	Create Route
172.31.0.0/16	local	Active	No	Create Route Table

They mean that I connected the public subnet to internet gateway via public route table.

This screenshot shows the details for the VPC 'vpc-0ebfed9dc98bbc58a'. It includes sections for 'Enabled', 'Main network ACL', 'IPv6 CIDR', 'Encryption control ID', and 'Resource map'. The 'Resource map' section shows connections between the VPC, Subnets, Route tables, and Network Connections.

Resource map

- VPC:** Your AWS virtual network, vpc-0ebfed9dc98bbc58a
- Subnets (2):** us-east-1a (project3-public-subnet), us-east-1b (project3-private-subnet)
- Route tables (3):** project3-public-rt, project3-private-rt, rtb-0f7a2fb0e0c491f65
- Network Connections (2):** project3-igw, project3-private-nat

Connections are shown as orange lines: project3-public-rt connects to both subnets; project3-private-rt connects to both subnets; and project3-igw connects to project3-private-nat.

Also, as you can see from the figure above, we need to connect private subnet to NAT using private route table. Then, I created a NAT Gateway, named **project3-private-nat**.

The screenshot shows the AWS VPC dashboard with the 'NAT gateways' section selected. A table lists one NAT gateway:

Name	NAT gateway ID	Connectivity type	State	Availability...	Primary public IPv4...	Primary private IPv4...	Primary network interface
project3-private-nat	nat-0888453ea838d1c1d	Public	Available	Zonal	44.199.221.53	172.31.0.14	eni-0c731295df7285b9f

And the final and **IMPORTANT** step is creating Security Groups for networking. I created two different security groups for RDS (**project3-rds-sg**) and Glue (**project3-glue-sg**). There is also a **default** security group by default in the AWS VPC, which we will not use it.

The screenshot shows the AWS VPC Security Groups console. A table lists three security groups:

Name	Security group ID	Security group name	VPC ID	Description
-	sg-0c71759a2c5bbc74c	project3-rds-sg	vpc-0ebfed9dc98bbc58a	Allow Glue JDBC to have access to RDS ...
-	sg-0e3f9a2c682e0f61d	project3-glue-sg	vpc-0ebfed9dc98bbc58a	Allow Glue to have access to RDS
-	sg-0f3049729f5f003a7	default	vpc-0ebfed9dc98bbc58a	default VPC security group

We can access to the Security Groups via VPC (above picture) or via EC2 (below picture).

The screenshot shows the AWS EC2 Security Groups console. A table lists three security groups:

Name	Security group ID	Security group name	VPC ID	Description
-	sg-0c71759a2c5bbc74c	project3-rds-sg	vpc-0ebfed9dc98bbc58a	Allow Glue JDBC to have access to RDS ...
-	sg-0e3f9a2c682e0f61d	project3-glue-sg	vpc-0ebfed9dc98bbc58a	Allow Glue to have access to RDS
-	sg-0f3049729f5f003a7	default	vpc-0ebfed9dc98bbc58a	default VPC security group

A dedicated security group named **project3-rds-sg** was created for the Amazon RDS instance to control inbound and outbound network traffic. The security group rules were configured as follows:

- **Inbound rules:**

1. MSSQL access from the local machine (`*.*.*.*/32`) allows Microsoft SQL Server connections from the laptop's public IP address for administrative access and testing.
2. MSSQL access from the VPC subnet (`172.31.1.0/24`) allows AWS Glue jobs running within the VPC to connect to the RDS instance for data ingestion.

- **Outbound rules:** All outbound traffic is allowed (`0.0.0.0/0`), enabling the RDS instance to communicate with required AWS services such as logging, monitoring, and maintenance endpoints.

This configuration ensures that the RDS instance is accessible only from trusted sources while maintaining necessary outbound connectivity.

If I click on the **project3-rds-sg**, I can edit the **inbound** and **outbound** IPv4 for this security group.

The screenshot shows the AWS VPC Security Groups console. The main table lists three security groups: project3-rds-sg, project3-glue-sg, and default. The project3-rds-sg row is selected. Below the table, a detailed view for the selected group shows the Inbound rules tab selected. It displays two inbound rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
sgr-005aea0e53811b479	IPv4	MSSQL	TCP	1433	146.190.57.198/32	My laptop IP	
sgr-0675e5975fadb36b3	IPv4	MSSQL	TCP	1433	172.31.1.0/24	Private Subnet	

As you can see above, the Inbound rule has two IPs. One of them is my local laptop's IP address (**146.190.57.198/32**), which will change as I change my wifi location; it is like ***.*.*.*/32** always. The other IP is a private subnet's IP (**172.31.1.0/24**) and is not change by end of the project. Also, as you can see below, the Outbound rule has only one general IP, which is open for **All traffic**, **0.0.0.0/0**.

The screenshot shows the AWS VPC Security Groups console. The main table lists the same three security groups. The project3-rds-sg row is selected. Below the table, a detailed view for the selected group shows the Outbound rules tab selected. It displays one outbound rule:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Destination	Description
sgr-0ebb50f5014a5673	IPv4	All traffic	All	All	0.0.0.0/0	Allow all	

1-3- Amazon RDS

After securing the connection, now it is time to create the RDS database for data ingestion from local SQL Server. I created the **project3-rds** database and connected it to our created VPC and network with minimal configuration.

The screenshot shows the Aurora and RDS console. The left sidebar shows the Databases section. The main area displays the details for the **project3-rds** database. The Summary tab is selected, showing the following information:

- DB identifier:** project3-rds
- Status:** Available
- Role:** Instance
- Engine:** SQL Server Express Edition
- Region & AZ:** us-east-1a
- Recommendations:** 1 informational

The Connectivity & security tab is selected, showing the following details:

- Endpoint:** project3-rds.cs52yq4ic7j.us-east-1.rds.amazonaws.com
- Port:** 1433
- Availability Zone:** us-east-1a
- VPC:** vpc-0ebfed9dc98bbc58a
- Subnet group:** Subnet_group_a
- Security:** VPC security groups: project3-rds-sg (sg-0c71759a2c5bce74c) (Active)
- Publicly accessible:** Yes

To have access to this database, we need to have username and password. I have created a secret inside the Secret Manager and save the RDS credentials in it. Then, I connected SQL Server to RDS using **Endpoint** information and **Port 1433** and with these information.

1-4- Secrets Manager

As I mentioned, I created the secret, named **project3-rds-secret**, and put all required information for RDS connection in it. You can see them in the format of JSON format. I do not use the Key/Value format as Glue JDBC can only read them in JSON format. To have strong encryption, I have created a KMS encryption key using **AWS KMS** service and encrypt our secret with that key.

The screenshot shows the AWS Secrets Manager interface. The top navigation bar includes 'AWS Secrets Manager > Secrets > project3-rds-secret'. The main title is 'project3-rds-secret'. Under 'Secret details', there are three fields: 'Encryption key' (set to 'project3-s3-kms'), 'Secret name' (set to 'project3-rds-secret'), and 'Secret ARN' (set to 'arn:aws:secretsmanager:us-east-1:000185048470:secret:project3-rds-secret-Dppcq'). Below this is a navigation bar with tabs: Overview (selected), Rotation, Versions, Replication, and Tags. The 'Secret value' section contains a 'Info' link and a note: 'Retrieve and view the secret value.' It has two tabs: 'Key/value' (disabled) and 'Plaintext' (selected). The secret value is shown as a JSON object:

```
[{"username": "admin", "password": "adminpass", "host": "project3-rds.ce52yg4ic7jk.us-east-1.rds.amazonaws.com", "port": 1433, "dbname": "project3_db"}]
```

1-5- AWS KMS

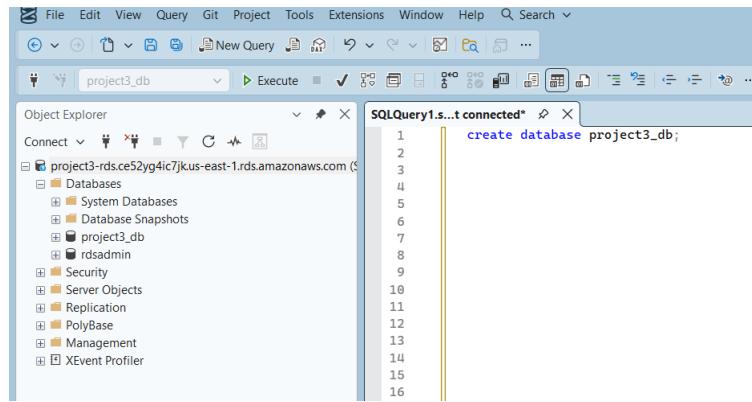
As I mentioned, I create an encryption key using Key Management Service (KMS), named **project3-s3-kms**. I used this key for whole project, specifically in DataLake S3 buckets.

The screenshot shows the AWS KMS interface. The top navigation bar includes 'KMS > Customer managed keys'. The left sidebar shows 'Key Management Service (KMS)' with sections for 'AWS managed keys' and 'Customer managed keys'. Under 'Customer managed keys', there is one item: 'project3-s3-kms'. The main area is titled 'Customer managed keys (1)' and lists the key with its details: 'Aliases' (empty), 'Key ID' (cd88ca6d-a393-4430-987...), 'Status' (Enabled), 'Key type' (Symmetric), 'Key spec' (SYMMETRIC_DEFAULT), and 'Key usage' (Encrypt and decrypt). There are also 'Key actions' and 'Create key' buttons at the top right.

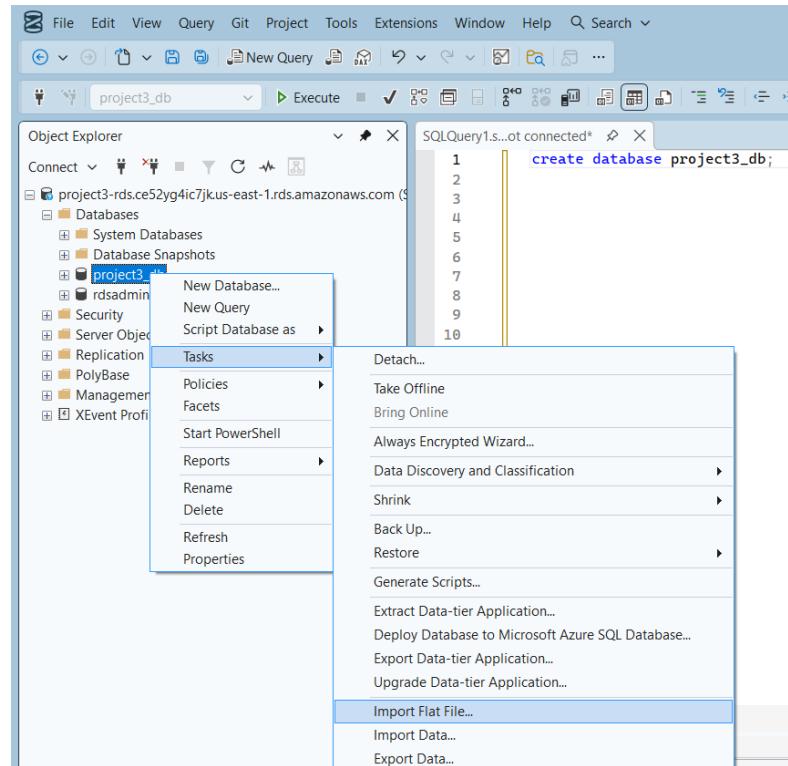
1-6- SQL Server

As requested in the project, our raw data is in local SQL Server. Therefore, I installed the Microsoft SQL Server 2022 (Evaluation Edition) and SQL Server Management Studio (SSMS) (v. 22.1.0) on my local computer. After opening the SSMS, I have connected it to the RDS using the credential and Endpoint and Portal information above. After making secure connection, I have created a database, named **project3_db**, and tried to upload three raw tables inside that database (inside schema **dbo**).

```
create database project3_db;
```



To upload the csv files, I used to import flat files directly into the **project3_db** database.



The file of **date_dim.csv** is uploaded easily, but for the other two, I need to change some data format during uploading:

Changes for **order_item_options.csv**:

Modify Columns				
This operation generated the following table schema. Please verify if schema is a please make any changes.				
Column Name	Data Type	Primary Key	<input type="checkbox"/> Allow Nulls	
ORDER_ID	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
LINEITEM_ID	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
OPTION_GROUP_NAME	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
OPTION_NAME	nvarchar(500)	<input type="checkbox"/>	<input type="checkbox"/>	
OPTION_PRICE	float	<input type="checkbox"/>	<input type="checkbox"/>	
OPTION_QUANTITY	tinyint	<input type="checkbox"/>	<input type="checkbox"/>	

and for **order_items.csv**:

Modify Columns				
This operation generated the following table schema. Please verify if schema is a please make any changes.				
Column Name	Data Type	Primary Key	<input type="checkbox"/> Allow Nulls	
APP_NAME	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
RESTAURANT_ID	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
CREATION_TIME_UTC	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
ORDER_ID	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
USER_ID	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
PRINTED_CARD_NUMBER	bigint	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
IS_LOYALTY	bit	<input type="checkbox"/>	<input type="checkbox"/>	
CURRENCY	nvarchar(50)	<input type="checkbox"/>	<input type="checkbox"/>	
LINEITEM_ID	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ITEM_CATEGORY	nvarchar(500)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ITEM_NAME	nvarchar(50)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
ITEM_PRICE	float	<input type="checkbox"/>	<input type="checkbox"/>	
ITEM_QUANTITY	int	<input type="checkbox"/>	<input type="checkbox"/>	

To validate all three tables are uploaded inside our local SQL Server, I quickly did this query:

```
SELECT COUNT(*) FROM project3_db.dbo.date_dim;
SELECT COUNT(*) FROM project3_db.dbo.order_items;
SELECT COUNT(*) FROM project3_db.dbo.order_item_options;

SELECT TOP 5 * FROM project3_db.dbo.date_dim;
SELECT TOP 5 * FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.order_items;
```

And the results show that everything is OK!

File Edit View Query Git Project Tools Window Help Search ...

New Query Execute ...

Object Explorer

SQLQuery1...admin (71)*

```
--create database project3_db;
SELECT COUNT(*) FROM project3_db.dbo.date_dim;
SELECT COUNT(*) FROM project3_db.dbo.order_items;
SELECT COUNT(*) FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.date_dim;
SELECT TOP 5 * FROM project3_db.dbo.order_item_options;
SELECT TOP 5 * FROM project3_db.dbo.order_items;
```

100% No issues found

Results Messages

(No column name)							
1 365							
(No column name)							
1 203519							
(No column name)							
1 193017							
date_key	year	month	week	day_of_week	is_weekend	is_holiday	holiday_name
1 2023-01-01	2023	1	52	Sunday	1	1	New Year's Day
2 2023-01-02	2023	1	1	Monday	0	1	New Year's Day Observed
3 2023-01-03	2023	1	1	Tuesday	0	0	NULL
4 2023-01-04	2023	1	1	Wednesday	0	0	NULL
5 2023-01-05	2023	1	1	Thursday	0	0	NULL

ORDER_ID	LINEITEM_ID	OPTION_GROUP_NAME	OPTION_NAME	OPTION_PRICE	OPTION_QUANTITY
1 6278eb55dfdfab1e053fe172	6278eb63dfdfab1e053fe18c	Cheese	American Cheese	0	1
2 6278eb55dfdfab1e053fe172	6278eb63dfdfab1e053fe18c	Veggies	Lettuce	0.5	1
3 6278eb55dfdfab1e053fe172	6278eb63dfdfab1e053fe18c	Veggies	Tomatoes	0	1
4 6278eb55dfdfab1e053fe172	6278eb63dfdfab1e053fe18c	Veggies	Onion	0	1
5 6278eb55dfdfab1e053fe172	6278eb63dfdfab1e053fe18c	Veggies	Green Peppers	0	1

APP_NAME	RESTAURANT_ID	CREATION_TIME_UTC	ORDER_ID	USER_ID	PRINTED_CARD_NUMBER	IS_LOYALTY	CURRENCY	LINEITEM_ID	ITEM_CATEG
1 Alltown Fresh	630c98a7519adc105105990	2023-03-08T11:03:32.223Z	640886b8463905b310002079	63ee144950286a8367041911	NULL	0	USD	640886b84d5fb8718a0717e6	Breakfast
2 Alltown Fresh	5f6a6c1537ab46bd38e9d75	2023-05-13T12:45:00.475Z	645f864dc1cb03121f0ca79	NULL	NULL	0	USD	645f8649568eb8e534e099ea7	Breakfast
3 Alltown Fresh	5f6e4a4ad0d51509c36f22	2023-02-03T07:15:28.519Z	63dcba90065d80cb6be09ab...	5ece7e902ad50133/b29d	NULL	0	USD	63dcba90065d80cb6be09ab...	Breakfast
4 Alltown Fresh	5f6a6c1537ab46bd38e9d75	2023-08-17T16:18:02.231Z	64de483a7028adc2520d8f0	NULL	NULL	0	USD	64de484eb7b053d89d0041...	Salads
5 Alltown Fresh	622107b40ac81503e0369ca6	2023-08-20T16:02.12.518Z	64e23904baef31a741016be9	NULL	NULL	0	USD	64e2395465826c9b650c137	Breakfast

Query executed successfully.

project3-rds.ce52yg4ic7jku... admin (71) master 00:00:00 Row: 1, Col: 1 18 rows

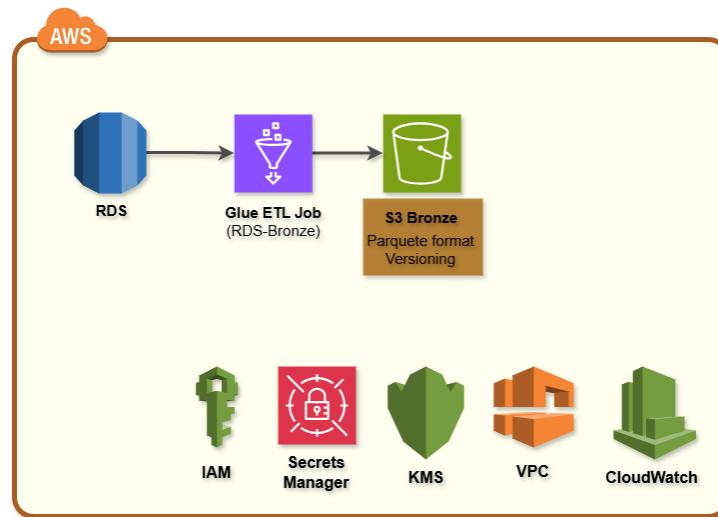
Step 2 – Data Landing (RDS to S3 Bronze)

2-1- Objective

The objective of this step is to land the ingested raw data from RDS database into an Amazon S3-based data lake, forming the Bronze layer. To ensure **data security at rest**, all Amazon S3 buckets (Bronze, Silver, and Gold) are protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**. This approach enables centralized key management and fine-grained access control through **IAM policies** to give permission for Glue ETL job to have access to S3 and RDS services.

Implementation Approach:

- AWS Glue ETL jobs will connect to SQL Server using JDBC.
- Source tables (`order_items`, `order_item_options`, and `date_dim`) will be extracted in batch mode.
- Data will be written to Amazon S3 in **Parquet** format to optimize storage efficiency and downstream processing.
- Enable S3 bucket versioning to retain all versions of the raw Parquet data.



I will give more details in the next sections.

2-2- Landing Storage (Bronze Layer)

First of all, I prefer to create the landing storage as a datalake. I used Amazon S3 service and create a bucket with name of `s3://project3-bronze-bucket/`. I have enabled its **versioning** option to store raw and immutable data to support auditability, reprocessing, and downstream transformations. After that, I created a folder inside that bucket with the name of `raw-data`. Then, all raw data will be written separately into the following S3 bucket: `s3://project3-bronze-bucket/raw-data/`

2-3- IAM Role

Before starting data ingestion using AWS Glue ETL jobs, I configured an IAM role to enable AWS Glue to securely access the required AWS services during job execution.

The following permissions were configured:

- **AWSGlueConsoleFullAccess:** Attached to my IAM user to allow me to create and manage AWS Glue resources, such as ETL jobs, workflows, and connections, through the AWS Management Console.
- **AWSGlueServiceRole:** Attached to the Glue job runtime role, allowing AWS Glue to assume the role at execution time and access required services, including Amazon S3, AWS Secrets Manager, Amazon CloudWatch Logs, and VPC networking resources.
- **AmazonS3FullAccess:** Granted to allow AWS Glue to read from and write to the Bronze, Silver, and Gold S3 buckets.
- **CloudWatchLogsFullAccess:** Enabled logging and monitoring of Glue job execution in Amazon CloudWatch.
- **SecretsManagerReadWrite:** Provided access to retrieve and manage secrets, including the RDS credentials.
- **AmazonRDSDataFullAccess:** Allowed AWS Glue to read raw source data from Amazon RDS using JDBC connectivity.
- **project3-KMS-key (Inline Policy):** A custom inline policy was created to allow encryption and decryption of sensitive data, including S3 bucket objects and RDS credentials stored in AWS Secrets Manager. The policy grants permissions to retrieve secret values and perform cryptographic operations using a customer-managed KMS key, as shown below:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:us-east-1:000185048470:secret:project3-rds-secret-DppcQK"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:GenerateDataKey",
        "kms:Encrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:000185048470:key/cd88ca6d-a393-4430-987a-f47fd77c4124"
    }
  ]
}
```

The IAM role was named **project3-glue-role** and tagged with **project: 3**, and it was attached to the AWS Glue ETL jobs to ensure secure and authorized access during execution.

2-4- VPC Security Group

As internal VPC networking is used between Amazon RDS and AWS Glue jobs, a separate security group was created specifically for AWS Glue and named **project3-glue-sg**.

Within the **project3-glue-sg**, inbound and outbound IPv4 rules were configured as follows:

- Inbound rules:** The inbound traffic is allowed from **project3-glue-sg** itself, enabling Glue components within the same security group to communicate with each other.
- Outbound rules:** Outbound traffic is allowed to **project3-rds-sg**, permitting the AWS Glue job to connect to the Amazon RDS instance for data ingestion.

This configuration ensures secure, controlled communication between AWS Glue and Amazon RDS within the VPC while restricting unnecessary network access.

If I click on the **project3-glue-sg**, I can edit the **inbound** and **outbound** IPv4 for this security group.

As you can see above, the inbound rule has one source, **project3-glue-sg**.

The screenshot shows the AWS VPC Security Groups console. In the top navigation bar, there are tabs for Actions, Export security groups to CSV, and Create security group. Below the navigation is a search bar labeled "Find security groups by attribute or tag". A table lists three security groups: project3-rds-sg, project3-glue-sg, and default. The project3-glue-sg row is selected and expanded, showing its details. The "Outbound rules" tab is active, displaying a table with two entries. The first entry is for port range 0-65535, destination sg-0c71759a2c5bce74c / project3-rds-sg, and type All TCP. The second entry is for port range 0.0.0.0/0, destination All AWS Service, and type All traffic.

There are two outbound rules. One of them is the source of **project3-rds-sg**, and the other is general IP, which is open for **All AWS Services, 0.0.0.0/0**.

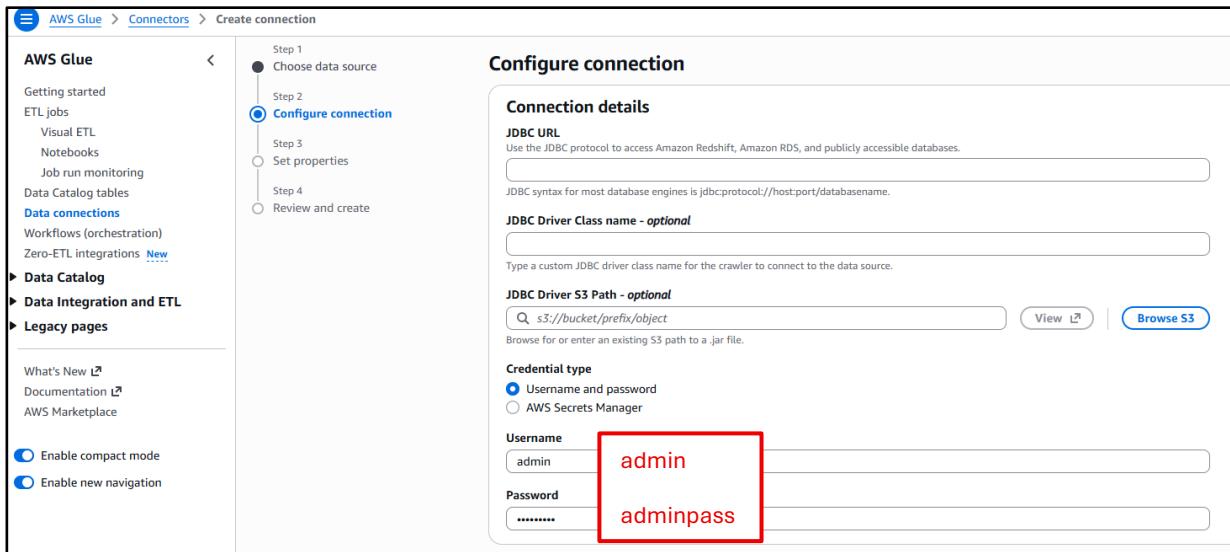
2-5- Glue JDBC Connection

To comply with the project requirement that all transformation logic be implemented using Spark, AWS Glue ETL job with JDBC connectivity will be used for data ingestion instead of AWS Lambda. This approach ensures that the ingestion process is both scalable and aligned with Spark-based processing standards. AWS Glue provides a distributed Spark environment that is well suited for:

- Reading relational data using JDBC
- Handling incremental and batch ingestion
- Scaling efficiently for future data growth
- Maintaining consistency across ingestion and transformation layers

In the section of AWS Glue → Data Connections, I created two connectors. The first one, named **project3-jdbc-connection**, which connects Glue to the RDS and I put the RDS credentials in the configuration of this connection, which is not secure (only for test). Then, I created the second connection, **project3-jdbc-connection-secret**, which all credentials are inside out secret in the secrets manager service. To create both connectons, I used JDBC data source.

The screenshot shows the AWS Glue Connectors console. On the left, there is a sidebar with links for AWS Glue, Getting started, ETL jobs, Visual ETL, Notebooks, Job run monitoring, Data Catalog tables, Data connections, Data Catalog, Data Integration and ETL, and Legacy pages. There are also checkboxes for Enable compact mode and Enable new navigation. The main area has tabs for Connectors, Marketplace connectors, and Custom connectors. The Connectors tab is active, showing a table with two entries: project3-jdbc-connection-secret and project3-jdbc-connection. Both entries are listed as "Ready" and of type "JDBC".



The important point is that the secret should be in the plaintext, not key/value, format. In that case Glue can read the secret successfully.

2-6- Glue ETL Job (RDS - Bronze)

Finally, after securing the connection between RDS and Glue, now, I am going to create the first AWS Glue PySpark ETL job, named **project3-rds-bronze**, to perform raw data ingestion from RDS database into S3 Bronze bucket and convert the file format from CSV into Parquet. For test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put all Glue ETL Jobs' script in another S3 code bucket. The script file name in the PySpark language is **project3-rds-bronze.py**.

```

import sys
import json
import logging
import boto3
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.utils import getResolvedOptions
from pyspark.sql import SparkSession
# -----
# Initialize Glue Context

```

After running the Glue ETL job, all three files landed from RDS database into S3 Bronze bucket successfully.

The screenshot shows the Amazon S3 console interface. The left sidebar navigation includes 'Amazon S3', 'Buckets' (with 'General purpose buckets' expanded), 'Access management and security', and 'Storage management and insights'. The main content area displays the 'raw-data/' folder under 'project3-bronze-bucket'. The 'Objects' tab is selected, showing 'Objects (3)'. A table lists three objects: 'dbo_date_dim/' (Folder), 'dbo_order_item_options/' (Folder), and 'dbo_order_items/' (Folder). Each object has a checkbox next to its name.

This screenshot shows the contents of the 'dbo_date_dim/' folder within the 'raw-data/' folder. The left sidebar is identical to the first screenshot. The main content area shows 'Objects (1/1)'. A table lists one object: 'part-00000-ebcc4a67-cfdb-4320-b693-b657b620d1ba-c000.snappy.parquet' (parquet). Both the folder and the file have checkboxes next to their names.

As you can see, there are three folders, with schema name and table name, in **s3://project3-bronze-bucket/raw-data/**, and inside each folder, there is a Parquet file.

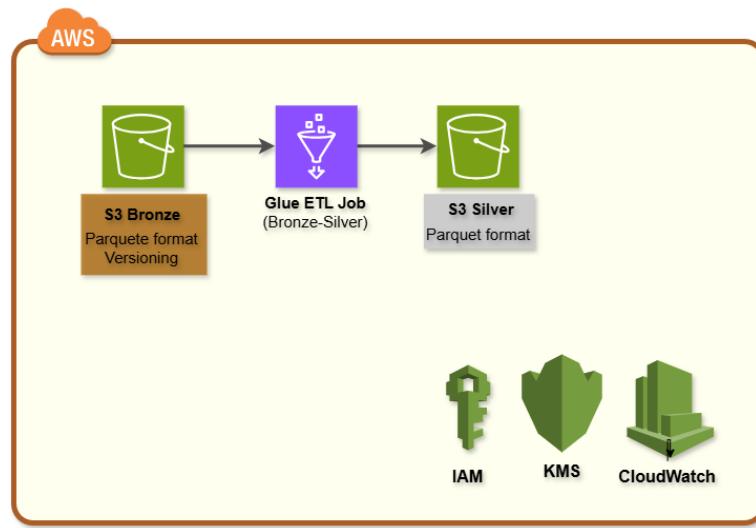
Step 3 – Transformation (S3 Bronze to S3 Silver)

3-1- Objective

The objective of the Silver layer is to transform raw ingested data into **clean, standardized, and analytically usable datasets**. This step focuses on improving data quality by resolving **missing values, removing duplicates, enforcing schemas, and preparing fact and dimension tables** for downstream analytics and metric computation.

Implementation Approach:

- AWS Glue ETL jobs will transform and clean up the raw data.
- Two fact tables of **fact_orders** and **fact_options**, and one dime table of **dim_date** will be created.
- Data will be written to Amazon S3 in **Parquet** format to optimize storage efficiency and downstream processing.
- Amazon S3 Silver bucket is protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**.



3-2- Transformed Storage (Silver Layer)

Data in this layer will be structured into fact and dimension-oriented datasets, enabling efficient joins and aggregations in subsequent processing stages. All three transformed data will be stored in a dedicated S3 bucket: <s3://project3-silver-bucket/>. I removed schema name from raw raw folders and classified them into dim and fact tables. Then, all transformed data will be written separately into their related folders. I have enabled the **SSE-KMS Encryption** for S3 Silver bucket.

3-3- Glue ETL Job (Bronze - Silver)

An AWS Glue PySpark ETL job, named **project3-bronze-silver**, will be executed to perform the following transformations:

- **Schema Enforcement**
 - Explicitly define schemas for fact and dimension tables to prevent schema drift.
 - Ensure consistent column ordering and naming conventions.
- **Data Type Standardization**
 - Cast columns to appropriate data types (e.g., timestamps, numeric fields, boolean flags).
 - Ensure consistency across datasets for join keys and metrics.
- **Duplicate Handling**
 - Identify and remove duplicate records based on defined primary or composite keys.
 - Preserve only the most recent or valid records where applicable.
- **Missing Value Management**
 - Remove records with critical missing identifiers (e.g., primary keys).
 - Retain non-critical nulls where they are analytically meaningful (e.g., holiday names).
- **Column Normalization**
 - Standardize column names (e.g., lowercase, snake_case) for consistency and usability.
 - Align naming conventions across all datasets.
- **Optimized Storage Format**
 - Persist transformed datasets in **Parquet format** to improve query performance and reduce storage costs.

Again, for test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put all Glue ETL Jobs' script in another S3 code bucket. The script file name in the PySpark language is **project3-bronze-silver.py**.

The screenshot shows the AWS Glue job configuration interface. On the left, there's a sidebar with navigation links like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Job run monitoring', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Zero-ETL integrations', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. Below that are 'What's New', 'Documentation', 'AWS Marketplace', and a 'Enable compact mode' checkbox. The main area has a header 'project3-bronze-silver' with tabs for 'Script' (which is selected), 'Info', 'Job details', 'Runs', 'Data quality', 'Schedules', 'Version Control', and 'Upgrade analysis - preview'. The 'Script' tab shows the following Python code:

```
1 import sys
2 import logging
3 from pyspark.context import SparkContext
4 from awsglue.context import GlueContext
5 from awsglue.utils import getResolvedOptions
6 from pyspark.sql.functions import (
7     col, regexp_replace, trim,
8     to_timestamp, to_date, date_format
9 )
10
11 # -----
12 # Initialize Glue Context
13 # -----
14 args = getResolvedOptions(sys.argv, ["JOB_NAME"])
15 sc = SparkContext()
16 glueContext = GlueContext(sc)
17 spark = glueContext.spark_session
```

After running the Glue ETL job, all three dim and fact folders with Parquet files inside them is created in the S3 Silver bucket successfully.

The image contains two screenshots of the Amazon S3 console interface.

Screenshot 1: project3-silver-bucket

This screenshot shows the main S3 bucket page for 'project3-silver-bucket'. The left sidebar includes sections for Buckets, Access management and security, and Storage management and insights. The right panel displays the 'Objects' tab, which lists three items under 'Objects (3)': 'dim_date/' (Folder), 'fact_options/' (Folder), and 'fact_orders/' (Folder). A 'Copy S3 URL' button is visible above the object list.

Name	Type
dim_date/	Folder
fact_options/	Folder
fact_orders/	Folder

Screenshot 2: dim_date/

This screenshot shows the contents of the 'dim_date/' folder. The left sidebar is identical to the first screenshot. The right panel displays the 'Objects' tab, which lists five Parquet files under 'Objects (5)'. The files are named: 'part-00000-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet', 'part-00001-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet', 'part-00002-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet', 'part-00003-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet', and 'part-00004-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet'.

Name	Type
part-00000-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet	parquet
part-00001-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet	parquet
part-00002-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet	parquet
part-00003-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet	parquet
part-00004-fd649079-021d-4a5f-a60f-6ff8eaae594-c000.snappy.parquet	parquet

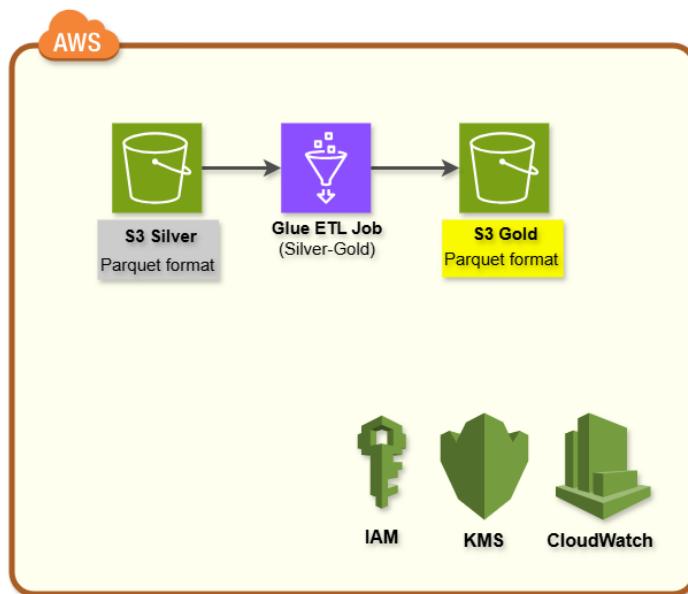
Step 4 – Analytics & Aggregation (S3 Silver to S3 Gold)

4-1- Objective

The objective of this step is to produce analytics-ready, aggregated datasets by joining and summarizing the dimensional and fact tables created in the Silver layer. The Gold layer serves as the single source of truth for business metrics and is optimized for direct consumption by visualization and reporting tools.

Notes:

- The Gold layer is explicitly optimized for **Streamlit dashboards**, enabling fast data access, simplified querying, and seamless visualization without additional transformation logic.
- The transformed datasets will be persisted in **Parquet format** to improve query performance and reduce storage costs.
- Amazon S3 Gold bucket is protected using **Server-Side Encryption with AWS Key Management Service (SSE-KMS)**.



4-2- Gold Layer Storage

Data in this layer will be structured into two different fact tables and stored in a dedicated S3 bucket: <s3://project3-gold-bucket/>. Again, I have enabled the **SSE-KMS Encryption** for S3 Gold bucket. All fact tables are stored in the Parquet format.

4-3- Glue ETL Job (Silver - Gold)

An AWS Glue PySpark ETL job, named **project3-silver-gold**, will generate curated datasets and metrics aligned with the business analytics questions, including but not limited to:

- Customer Lifetime Value (CLV) evolution (daily)
- RFM (Recency, Frequency, Monetary) snapshot
- Sales trends over time
- Location-level performance metrics
- Customer Churn indicators
- Other relevant KPI metrics

Again, for test, the script of this Glue ETL job is loaded inline script, but later in the CI/CD production level, I will put the script, **project3-silver-gold.py**, in another S3 code bucket.

The screenshot shows the AWS Glue console interface. On the left, there's a sidebar with navigation links like 'Getting started', 'ETL jobs', 'Visual ETL', 'Notebooks', 'Data Catalog tables', 'Data connections', 'Workflows (orchestration)', 'Zero-ETL integrations', 'Data Catalog', 'Data Integration and ETL', and 'Legacy pages'. The main area is titled 'project3-silver-gold'. At the top, there are tabs for 'Script' (which is selected), 'Job details', 'Runs', 'Data quality', 'Schedules', 'Version Control', and 'Upgrade analysis - preview'. Below the tabs, there's a timestamp 'Last modified on 12/31/2025, 10:19:54 PM' and buttons for 'Actions', 'Save', and 'Run'. The 'Script' tab contains the following Python code:

```
1 import sys
2 import logging
3 from pyspark.context import SparkContext
4 from awsglue.context import GlueContext
5 from pyspark.sql import functions as F
6 from pyspark.sql.window import Window
7 from pyspark.sql.types import IntegerType, BooleanType
8
9 # =====#
10 # LOGGING
11 # =====#
```

After running the Glue ETL job, two fact folders with Parquet files inside them (**fact_customer** and **fact_orders_location**) are created in the S3 Gold bucket successfully.

The screenshot shows the AWS S3 console. On the left, there's a sidebar with 'Amazon S3' and sections for 'Buckets', 'General purpose buckets', 'Access management and security', 'Access Points', 'Access Points for FSx', 'Access Grants', and 'IAM Access Analyzer'. The main area shows the contents of the 'project3-gold-bucket'. There are tabs for 'Objects', 'Metadata', 'Properties', and 'Permissions'. Under the 'Objects' tab, it says '(2)' and shows two entries: 'fact_customer/' and 'fact_orders_location/'. The 'fact_customer/' folder is expanded, showing its contents. On the right, there's a detailed view of the 'fact_customer/' folder, which also has tabs for 'Objects' and 'Properties'. It says '(4)' and shows four objects: 'part-00000-f8a09255-a3db-4e41-bc1c-526d9fd88a09_c000.snappy.parquet', 'part-00001-f8a09255-a3db-4e41-bc1c-526d9fd88a09_c000.snappy.parquet', 'part-00002-f8a09255-a3db-4e41-bc1c-526d9fd88a09_c000.snappy.parquet', and 'part-00003-f8a09255-a3db-4e41-bc1c-526d9fd88a09_c000.snappy.parquet'. All objects are of type 'parquet'.

Step 5 – Visualization (Streamlit)

5-1- Objective

In the final step, Streamlit dashboards will query data directly from the Gold S3 bucket. I have created two **key buttons** inside the dashboard for manual triggering the orchestration. I used Pandas for data handling (sufficient for moderate-sized parquet files), and built interactive dashboards for stakeholders with filters, charts, and KPI cards.

Step 6 – Jobs Orchestration and CI/CD

6-1- Objective

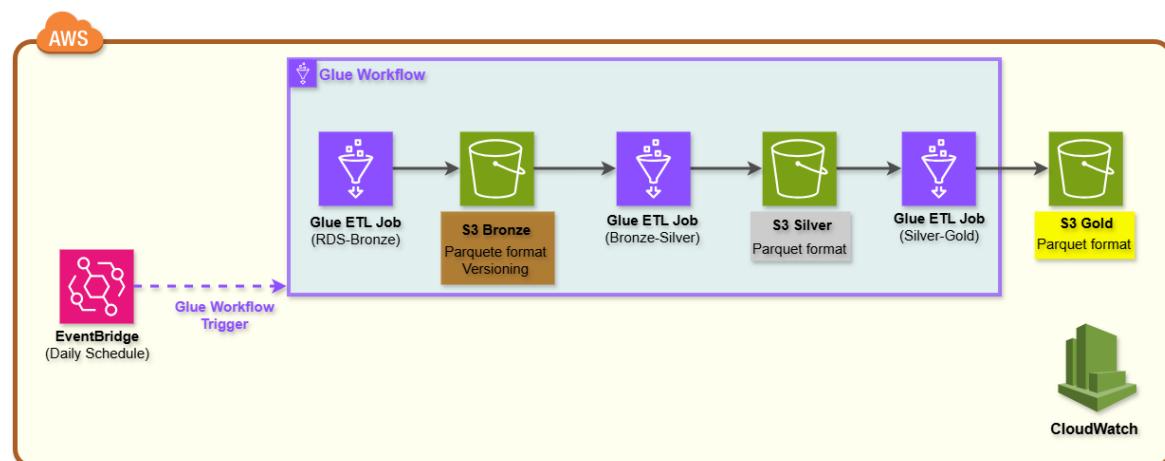
The objective of this component is to automate the end-to-end batch data pipeline on a daily schedule while ensuring reliable orchestration, clear job dependencies, and maintainable CI/CD practices. All data is ingested from SQL Server into AWS RDS, transformed through Bronze, Silver, and Gold layers, and made available for analytics and visualization without manual intervention. To support continuous development and deployment, all AWS Glue Spark scripts are version-controlled in a GitHub repository and automatically deployed to AWS S3 code bucket when changes occur.

6-2- Scheduling and Orchestration

For orchestration of Glue ETL jobs, I used Glue Workflow, named **project3-bronze-silver-gold**, which acts as the central orchestration engine for the pipeline. This Glue Workflow is triggered **manually** or using **EventBridge**. EventBridge is scheduled to trigger the Glue Workflow daily (10 minutes for test).

The Glue Workflow executes jobs in a dependency-based sequence:

Glue ETL Job (RDS - Bronze) → Glue ETL Job (Bronze - Silver) → Glue ETL Job (Silver - Gold)

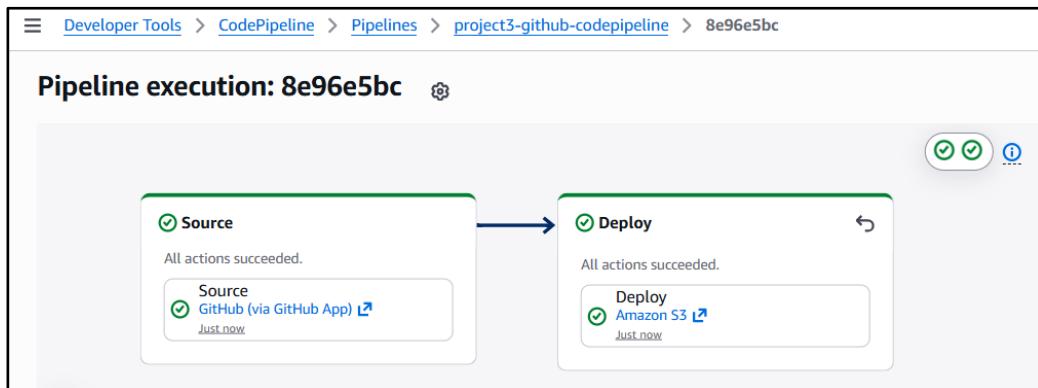


6-3- CI/CD Strategy

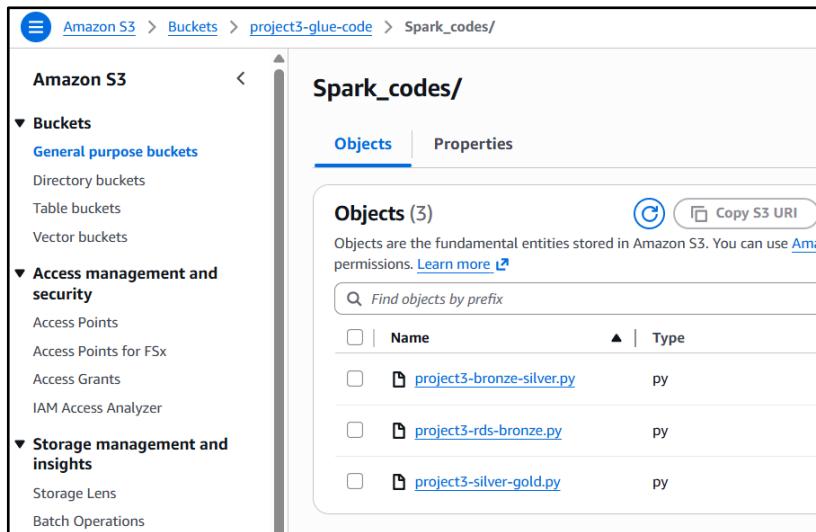
This approach enables automated deployment, traceability, and reproducibility of data pipelines. All PySpark scripts ([project3-rds-bronze.py](#), [project3-bronze-silver.py](#), [project3-silver-gold.py](#)) for three Glue ETL jobs are stored in a **GitHub** repository:

https://github.com/Hadi2468/ELT_Project3/tree/main/Spark_codes.

The CI/CD pipeline automatically detects changes or additions to Glue job scripts in the GitHub repo and immediately deploys updated scripts to another S3 (code) bucket, name [s3://project3-code-bucket/](#) using AWS CodePipeline service.

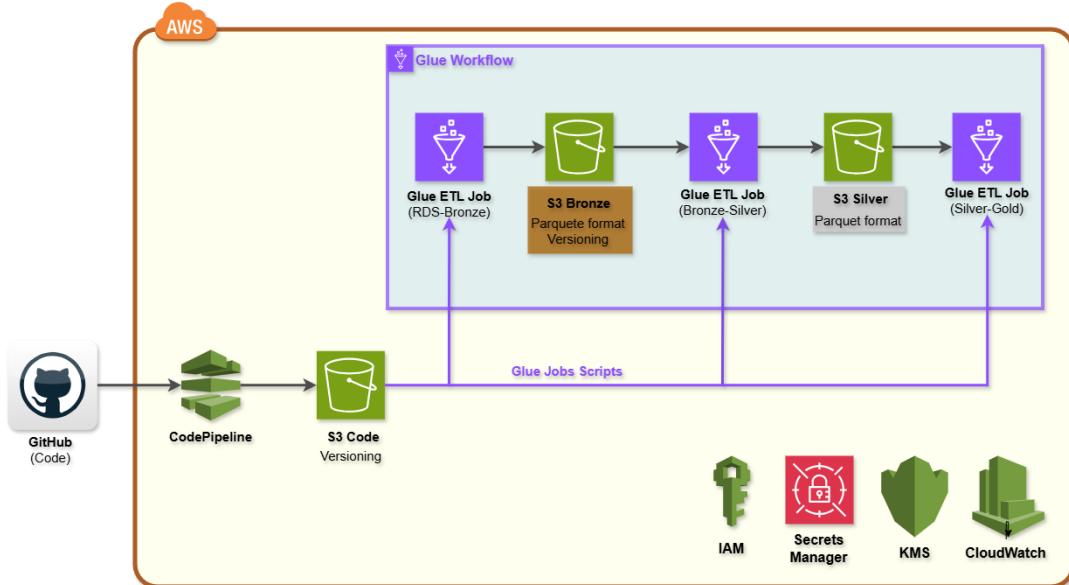


I have enabled its **versioning** option to store all versions of codes to support auditability, reprocessing, and downstream transformations. Also, I have enabled the SSE_KMS encryption in Amazon S3 Code bucket using AWS Key Management Service (KMS) for more security.



All Glue jobs reference their scripts directly from the S3 code bucket, ensuring:

- Version consistency
- Easy rollback
- No manual code uploads



6-4- IAM Role

Before starting the code transfer using AWS CodePipeline, I configured an IAM role named **project3-codepipeline-role** and tagged with **project: 3**. This role grants CodePipeline the required permissions to securely connect to the GitHub repository, retrieve the PySpark source code, and deploy the artifacts to Amazon S3. The following permissions were configured:

- **AWSCodePipelineServiceRole**: Grants CodePipeline core permissions to orchestrate pipeline stages and interact with AWS services on behalf of the pipeline.
- **CodePipeline-CodeConnections**: Allows CodePipeline to establish and manage secure connections to external source providers such as GitHub using AWS CodeStar Connections.
- **CodePipeline-S3Deploy**: Provides permission for CodePipeline to upload, read, and manage build artifacts in Amazon S3 buckets used during the deployment process.
- **project3-KMS-codepipeline (Inline Policy)**: A custom inline policy was created to allow AWS CodePipeline to use a customer-managed KMS key for encrypting and decrypting pipeline artifacts, as shown below. Enables CodePipeline to encrypt and decrypt artifacts stored in Amazon S3 using AWS Key Management Service (KMS).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:GenerateDataKeyWithoutPlaintext"
      ],
      "Resource": "arn:aws:kms:us-east-1:000185048470:key/cd88ca6d-a393-4430-987a-f47fd77c4124"
    }
  ]
}
```

The screenshot shows the AWS IAM Roles page. The left sidebar has a search bar and navigation links for Dashboard, Access management (User groups, Users, Roles), and IAM. The main area shows a message: "You can attach up to 10 managed policies." A search bar and filter buttons ("Filter by Type" and "All types") are present. A table lists four policies:

Policy name	Type
AWSCodePipelineServiceRole-us-east-1-project3-github-codepipeline	Customer managed
CodePipeline-CodeConnections-us-east-1-project3-github-codepipeline	Customer managed
CodePipeline-S3Deploy-us-east-1-project3-github-codepipeline	Customer managed
project3-KMS-codepipeline	Customer inline

The benefits of this orchestration strategy are that this design ensures:

- Clear and deterministic job dependencies
- Controlled execution order
- Parallel processing where appropriate
- Minimal operational complexity
- Scalable and production-ready CI/CD practices
- Failure isolation and observability using **Amazon CloudWatch**

Step 7 – Demonstration

Now, we want to demonstrate the project and run the EventBridge to trigger the Glue Workflow function to start getting data from local SQL Server, run Glue ETL jobs for data ingestion, transformation, and aggregation analysis in the approach of Bronze-Silver-Gold. The first Glue job does some data clean-up stuff on the raw CSV files and convert them into parquet files and saves dim and fact tables inside Silver S3 bucket. The second and third Glue jobs will be run after the previous Glue job finished, and all final BI-related dim and fact tables will be created in the Gold S3 bucket.

7-1- CI/CD approach

As soon as we click on **GitHub Deployment via CodePipeline** button in the Streamlit dashboard (manually), or any small change in the GitHub repository, the CodePipeline automatically gets the update from GitHub repo and shares it inside the S3 code bucket. Then, the PySpark scripts are available for Glue ETL jobs immediately.



7-2- Trigger Glue Workflow

As soon as we click on **Trigger Glue Job Manually** button in the Streamlit dashboard (manually), or the scheduled time on the EventBridge is approached, the EventBridge will trigger the Glue Workflow and all three Glue ETL jobs get their PySpark scripts from the S3 code bucket and start one after one successfully.



As you can see in the [project3-rds-bronze.py](#) file, the first Glue ETL job ingests all CSV files from RDS database and put them in the S3 Bronze bucket with Parquet format. After succeeding in this process, the Glue Workflow triggers the second Glue ETL job, and the orchestration keep ing on.

As shown in the [project3-rds-bronze.py](#) file, the first AWS Glue ETL job ingests all CSV files from the RDS database and stores them in the Amazon S3 Bronze bucket in Parquet format. Once this process is completed successfully, the Glue Workflow automatically triggers the second Glue ETL job, and the orchestration continues sequentially.

To validate, I have downloaded all Parquet files from AWS Bronze, Silver, and Gold buckets into my local machine, and checked them carefully.

```
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_date_dim/ ./data/s3/date_dim --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_order_items/ ./data/s3/order_items --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-bronze-bucket/raw-data/dbo_order_item_options/ ./data/s3/order_item_options --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/fact_options/ ./data/s3/fact_options --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/fact_orders/ ./data/s3/fact_orders --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-silver-bucket/dim_date/ ./data/s3/dim_date --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-gold-bucket/fact_customer/ ./data/s3/fact_customer/ --recursive --exclude "*" --include "*.parquet"
aws s3 cp s3://project3-gold-bucket/fact_orders_location/ ./data/s3/fact_orders_location/ --recursive --exclude "*" --include "*.parquet"
```

After downloading Parquet files for both dim and fact tables, now in Jupyter Notebook and using Pandas library, explored on those tables for validation. In the Bronze layer, everything is OK. Also, in the Silver layer, we can see, all the cleaning actions are done successfully.

date_dim table:

- Rename table **date_dim** into **dim_date**

order_items table:

- Rename table **order_items** into **fact_orders**
- Normalizes column names (lower_case and snake_case)
- Drops one record with **NULL 'LINEITEM_ID'**
- Splits '**CREATION_TIME_UTC**' into three columns of '**creation_date**' (DATE), '**creation_time**' (TIME), and removes T, Z, and milliseconds in '**creation_ts_utc**'
- Writes clean Parquet to Silver bucket

order_item_options table:

- Rename table **order_items** into **fact_options**
- Normalizes column names (lower_case and snake_case)
- Deduplicates correctly 2299 rows from **fact_options**

Here are the results of transformation in the S3 Gold layer:

fact_customer_daily_metrics	
Columns	Type
user_id [PK]	object
creation_d	datetime64[ns]
orders_daily	int64
revenue_daily	float64
loyalty_revenue	float64
loyalty_orders	int64
recency_days	Int32
churn_risk_flag	object
clv_cumulative	float64
frequency_7d	float64
monetary_7d	float64
frequency_30d	float64
monetary_30d	float64
frequency_90d	float64
monetary_90d	float64

fact_orders_location_performance_daily	
Columns	Type
creation_date	datetime64[ns]
restaurant_id [PK]	object
total_orders	int64
total_revenue	float64
avg_order_value	float64
loyalty_revenue	float64
loyalty_orders	int64
non_loyalty_revenue	float64
non_loyalty_orders	int64
period_day	Int32
period_month	Int32
period_week	Int32
year	Int32
revenue_rank_day	Int32
revenue_rank_week	Int32
revenue_rank_month	Int32

Conclusion

In this project, an end-to-end, production-oriented data engineering pipeline was designed and implemented to transform raw transactional data into analytics-ready business insights. Starting from exploratory data analysis on local CSV files, the project systematically addressed data quality issues, schema design, and primary key definition before moving into a fully automated cloud-based architecture on AWS.

The pipeline follows a **Bronze–Silver–Gold** data lake architecture using Amazon S3, ensuring clear separation of raw, cleaned, and curated datasets. Data ingestion was securely performed from a local SQL Server into Amazon RDS and subsequently landed into the S3 Bronze layer using AWS Glue PySpark jobs with JDBC connectivity. Strong security practices were applied throughout the system, including private VPC networking, dedicated security groups, IAM least-privilege roles, AWS Secrets Manager for credential management, and SSE-KMS encryption across all S3 buckets.

Data transformation logic implemented in the Silver layer successfully enforced schemas, removed duplicates, handled missing values, normalized column naming, and produced well-structured fact and dimension tables. The Gold layer aggregated these cleaned datasets into analytics-ready fact tables optimized for business intelligence use cases such as customer lifetime value (CLV), RFM analysis, churn indicators, sales trends, and location-level performance metrics.

Automation and reliability were achieved through **AWS Glue Workflows** for orchestration and **Amazon EventBridge** for scheduling, enabling deterministic, dependency-driven execution of all ETL jobs. Additionally, a robust **CI/CD pipeline** using **AWS CodePipeline** and **GitHub** was implemented to manage Glue job scripts, ensuring version control, reproducibility, easy rollback, and zero manual deployment effort.

Finally, the curated Gold datasets were consumed directly by **Streamlit dashboards**, demonstrating how the pipeline supports real-time business insights and decision-making. Overall, this project demonstrates a scalable, secure, and maintainable data platform that closely mirrors real-world production data engineering systems and effectively bridges raw data ingestion with actionable analytics.