

Group Project - AWS Data Analytic Platform for The City of Vancouver

Gyanvi Bhardwaj (2320329)

Peng Wang (2306811)

Muhammad Zulqarnain Shahzad (2306553)

Haoran Xu (2317218)

University Canada West

BUSI653: Cloud Computing Technologies (HBD-WINTER25-03)

Dr Mahmood Mortazavi Dehkordi

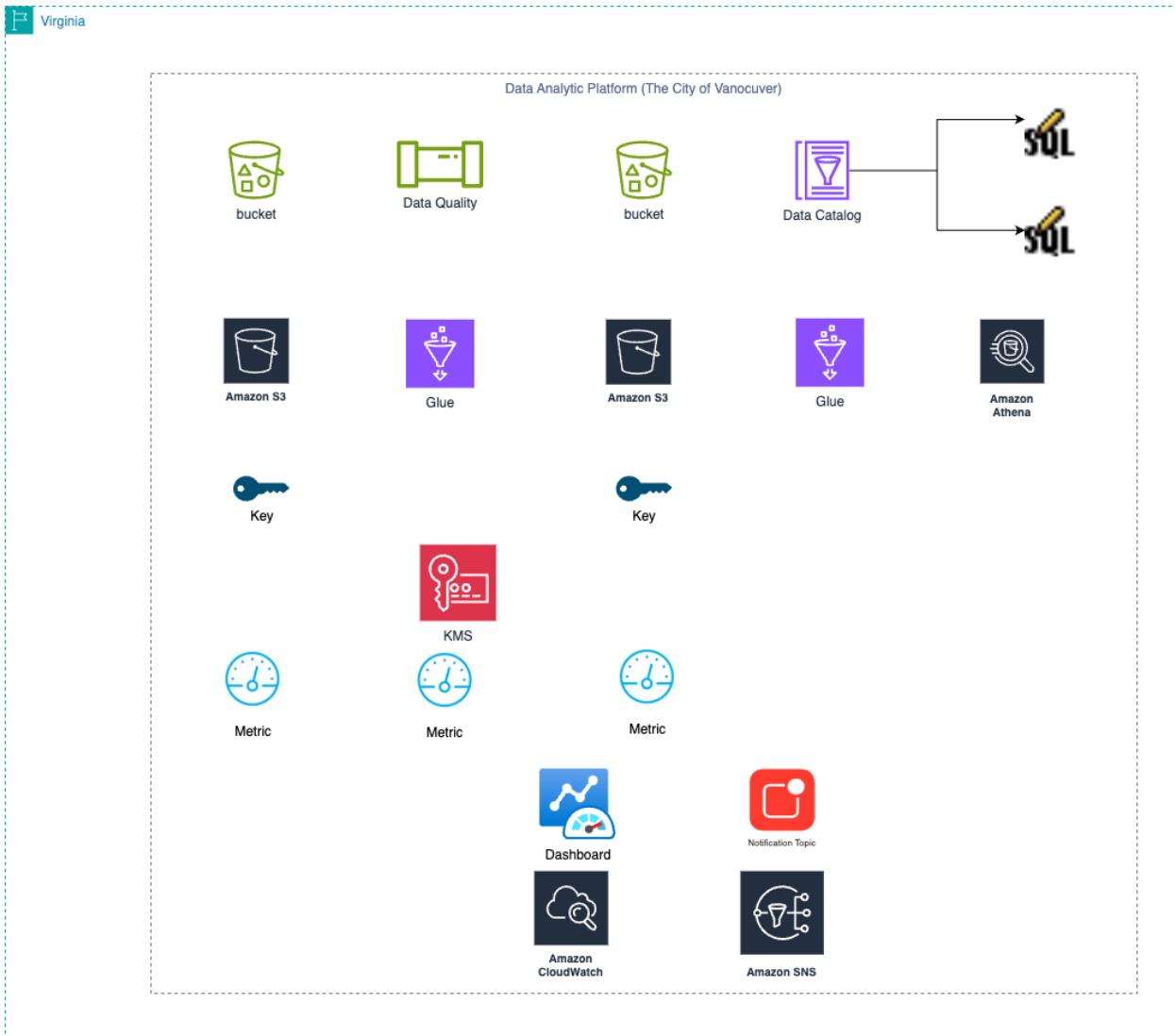
March 23, 2025

Table of Contents

| | |
|---------------------------------------------------------------------------------|-----------|
| DAP Implementation (Individual work - Gyanvi Bhardwaj) | 4 |
| Step 5: Data Analysis | 4 |
| Step 6: Data Security | 6 |
| Step 7: Data Governance..... | 9 |
| Step 8: Data Monitoring..... | 13 |
| DAP Implementation (Individual work – Peng Wang) | 16 |
| Step 5: Data Analysis | 16 |
| Step 6: Data Security | 18 |
| Step 7: Data Governance..... | 22 |
| Step 8: Data Monitoring..... | 24 |
| DAP Implementation (Individual work - Muhammad Zulqarnain Shahzad) | 26 |
| Step 5: Data Analysis | 26 |
| Step 6: Data Security | 30 |
| Step 7: Data Governance..... | 33 |
| Step 8: Data Monitoring..... | 36 |
| DAP Implementation (Individual work – Haoran Xu)..... | 41 |
| Step 5: Data Analysis | 41 |
| Step 6: Data Security | 43 |
| Step 7: Data Governance..... | 46 |
| Step 8: Data Monitoring..... | 51 |
| DAP Architecture Analysis (Teamwork) | 54 |
| Operational Excellence | 54 |
| Security | 55 |
| Reliability | 59 |
| Performance Efficiency | 61 |
| Cost Optimization | 65 |

Figure 1

DAP requirements for the City of Vancouver



Note. This picture shows the procedure our team needs to implement the DAP to support the requirements of the City of Vancouver (*Draw.io, self-work*).

DAP Implementation (Individual work - Gyanvi Bhardwaj)

Step 5: Data Analysis

The data analysis step was carried out using AWS Athena which employs SQL queries to analyze data. A new folder was created in the raw zone of the AWS S3 bucket and named ‘animal-control-semi-ds’. This contained the cleaned dataset. Using Glue Databrew, the columns that were not required to answer the two business questions were dropped. This also ensures that the cost associated with scanning the data is lowered as only the required data is available in the dataset provided for analysis to Athena. Now, the dataset is structured and ready for analysis using Athena.

Figure 2: Cleaned dataset ready for analysis

The screenshot shows the AWS Glue DataBrew interface. At the top, a green header bar indicates 'Created project "animal-control-semi-prj"'. Below the header, the main workspace displays a 'SAMPLE' view of three tables: 'ABC Breed', 'Datelmpounded', and 'ABC AgeCategory'. The 'ABC Breed' table has 500 rows and includes columns for Distinct (209), Unique (132), Total (500), and various breeds like Pit Bull, Chihuahua, Husky, and Bichon Frise. The 'Datelmpounded' table has 500 rows and includes columns for Distinct (255), Unique (110), Total (500), and dates from 2024-06-14 to 2024-01-30. The 'ABC AgeCategory' table has 500 rows and includes columns for Distinct (5), Unique (0), Total (500), and categories like Adult, Young Adult, Senior, and Puppy. To the right of the sample view, there is a 'RECIPE' section titled 'Recipe (2)' which lists applied steps: 1. Delete column AnimalID, Sex, ReceiptNumber, Name, KennelNumber, DispositionDate, Color, Source, Status, ACO, EnteredBy and 2. Change format of Datelmpounded to yyyy-mm-dd. The bottom of the interface shows standard AWS navigation links for CloudShell, Feedback, and copyright information from 2025.

Note. The cleaned dataset is further formatted to drop unnecessary columns. (Source: AWS Glue, self-work)

AWS Athena query editor takes our Data catalog as the data source and answers the two business questions. For our analysis, we need only 3 columns- date impounded, breed and age category.

The SQL query for business question 1 selects the maximum date of impoundment and creates an alias as ‘Impound_Date’ and groups it by the age category. The result gives the recent impound date for each animal age category.

Figure 3: AWS Athena Business Question 1 SQL query

The screenshot shows the AWS Athena Query Editor interface. On the left, the sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (animal-control-catalog). Under Tables and views, there are two tables: 'animal-control-metrics' and 'animal_control_trfsystem'. The main panel shows the SQL query:

```

Metric1 : X Metric2 : X
1 select max(dateimpounded) as Impound_Date, agecategory
2 from "animal_control_trfsystem"
3 group by agecategory;
4

```

The 'Query results' tab shows the output of the query:

| # | Impound_Date | agecategory |
|---|--------------|--------------|
| 1 | 2024-12-25 | unidentified |
| 2 | 2024-12-31 | Young Adult |
| 3 | 2024-12-20 | Puppy |
| 4 | 2024-12-31 | Senior |
| 5 | 2024-12-31 | Adult |

Note. AWS Athena SQL query for business question 1. (Source: AWS Athena, self-work)

The SQL query for business question 2 selects the number of breeds and creates an alias as ‘Distinct_Breed’ and groups it by the age category. The result gives the number of distinct breeds for each animal age category.

Figure 4: AWS Athena Business Question 2 SQL query

The screenshot shows the AWS Athena Query editor interface. On the left, the 'Data' sidebar is open, displaying the 'Tables and views' section with two tables listed: 'animal-control-metrics' and 'animal_control_trfystem'. The main area shows a query named 'Metric1' with the following SQL code:

```

1 select count(breed) as Distinct_Breed, agecategory
2 from "animal_control_trfystem"
3 group by agecategory;

```

The results section shows 5 rows of data:

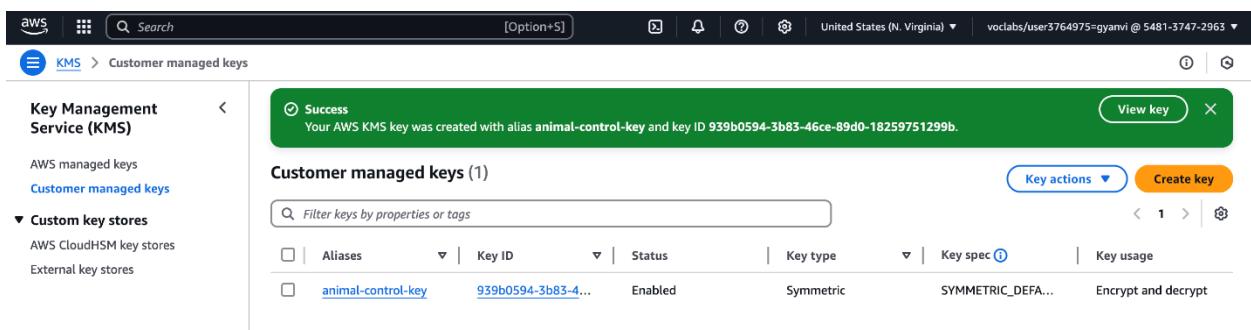
| # | Distinct_Breed | agecategory |
|---|----------------|--------------|
| 1 | 65 | Senior |
| 2 | 296 | Adult |
| 3 | 47 | Puppy |
| 4 | 86 | Young Adult |
| 5 | 16 | unidentified |

Note. AWS Athena SQL query for business question 2. (Source: AWS Athena, *self-work*)

Step 6: Data Security

Data security deals with creating and managing keys while controlling encryption activities to protect our data. Amazon Key Management System has been used to create a key called ‘animal-control-key’. LabRole is the key administrator and the key user. Encryption has been enabled for this key.

Figure 5: Key created for animal control inventory data



Note. Key created for encryption to be used in animal control inventory dataset. (Source: AWS KMS, self-work)

Next, the raw and transformed buckets were managed to provide encryption and replication for enhancing data security. The key ‘animal-control-key’ is used for both the raw and transformed buckets to enable encryption.

For bucket versioning, we have chosen this option to keep our data safe by creating multiple copies of it for any unforeseen event. By creating a new S3 bucket called ‘animal-control-inventory-backup’ for replication rules, the replication rule named ‘animal-control-inventory-replication’ is stored in the new bucket.

Similarly, for creating the replication rule for the transformed bucket, we have created another S3 bucket called ‘animal-control-trf-backup’. The replication rule called ‘animal-control-trf-replication’ is stored in this new bucket created.

Figure 6: Bucket versioning and encryption for raw bucket

The screenshot shows the AWS S3 Bucket Properties page for the bucket 'animal-control-inventory-raw'. The left sidebar lists various S3 features like General purpose buckets, Storage Lens, and Feature spotlight. The main content area has three tabs: Bucket Versioning, Tags (0), and Default encryption. The Bucket Versioning tab shows that versioning is enabled. The Tags tab shows no tags associated with the resource. The Default encryption tab shows that server-side encryption is applied using AWS Key Management Service keys (SSE-KMS), with the ARN arn:aws:kms:us-east-1:548137472963:key/939b0594-3b83-46ce-89d0-18259751299b listed.

Note. Bucket versioning and encryption for the animal-control-raw bucket. (AWS S3, self-work).

Figure 7: Bucket versioning and encryption for transformed bucket

The screenshot shows the AWS S3 Bucket Details page for the bucket 'animal-control-inventory-trf'. The left sidebar includes sections for General purpose buckets, Storage Lens, Feature spotlight, and AWS Marketplace for S3. The main content area displays the following settings:

- Bucket Versioning:** Enabled
- Multi-factor authentication (MFA delete):** Disabled
- Tags (0):** No tags associated with this resource.
- Default encryption:** Server-side encryption is automatically applied to new objects stored in this bucket. Encryption type: SSE-KMS. Encryption key ARN: arn:aws:kms:us-east-1:548137472963:key/939b0594-3b83-46ce-89d0-18259751299b
- Bucket Key:** When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. Bucket key status: Enabled

Note. Bucket versioning and encryption for the animal-control-trf bucket. (AWS S3, self-work).

Figure 8: Replication rule created for raw bucket

The screenshot shows the AWS S3 Replication rules page for the bucket 'animal-control-inventory-raw'. The page displays a single replication rule:

| Replication rule name | Status | Destination bucket | Destination Region | Priority | Scope | Storage class | Replica owner | Replication Time Control | KMS-encrypted objects (SSE-KMS or DSSE-KMS) | Replica modification sync |
|--------------------------------------|---------|--------------------------------------|---------------------------------|----------|---------------|----------------|----------------|--------------------------|---------------------------------------------|---------------------------|
| animal-control-inventory-replication | Enabled | s3://animal-control-inventory-backup | US East (N. Virginia) us-east-1 | 0 | Entire bucket | Same as source | Same as source | Disabled | Replicate | Disabled |

Note. A replication rule was created for the raw zone in the new raw-backup bucket. (AWS S3, self-work).

Figure 9: Replication rule created for transformed bucket

| Replication rule name | Status | Destination bucket | Destination Region | Priority | Scope | Storage class | Replica owner | Replication Time Control | KMS-encrypted objects (SSE-KMS or DSSE-KMS) | Replica modification sync |
|--------------------------------|---------|--------------------------------|---------------------------------|----------|---------------|----------------|----------------|--------------------------|---------------------------------------------|---------------------------|
| animal-control-trf-replication | Enabled | s3://animal-control-trf-backup | US East (N. Virginia) us-east-1 | 0 | Entire bucket | Same as source | Same as source | Disabled | Disabled | Disabled |

Note. A replication rule was created for the transformed zone in the new trf-backup bucket.

(AWS S3, self-work).

Step 7: Data Governance

The data governance step was carried out in AWS Glue where the visual ETL pipeline was created. This pipeline involves checking data for its quality using three parameters namely, completeness, uniqueness and freshness. The pipeline is named ‘Animal-control-inventory-QC’ and takes the data from the animal-control-inventory raw bucket. Next, the ruleset editor creates the following three rules:

1. Completeness of the ‘breed’ column should be greater than or equal to 0.95

2. Uniqueness concerns the primary key column called ‘*animalid*’ which should be greater than 0.99

3. Freshness has a date column called ‘*dateimpounded*’ less than 90 days

Figure 10: Rules created for animal-control-inventory-QC

The screenshot shows the AWS Glue Data Quality interface for the job 'Animal-control-inventory-QC'. A green success message at the top states: 'Successfully created job Animal-control-inventory-QC. To run the job choose the Run Job button.' The main interface displays three successfully created rules:

- 1. Completeness "breed" >= 0.95
- 2. Uniqueness "animalid" > 0.99
- 3. DataFreshness "dateimpounded" < 90 days

The 'Data preview' section shows the results for these rules:

| Rule | Outcome | FailureReason | EvaluatedMetrics.string | Evaluated |
|-----------------------------------------|---------|-------------------------------------|-----------------------------------------|-----------|
| Completeness "breed" >= 0.95 | Passed | null | Completeness 0.95 | 0.95 |
| Uniqueness "animalid" > 0.99 | Passed | null | Uniqueness 0.99 | 0.99 |
| DataFreshness "dateimpounded" < 90 days | Failed | 0.50 % of rows passed the threshold | DataFreshness "dateimpounded" < 90 days | 0.50 |

The 'Ruleset editor' pane shows the rule definitions:

```

1 ▼ Rules = [Completeness "breed" >= 0.95,
2 Uniqueness "animalid" > 0.99,
3 DataFreshness "dateimpounded" < 90 days
4 ]

```

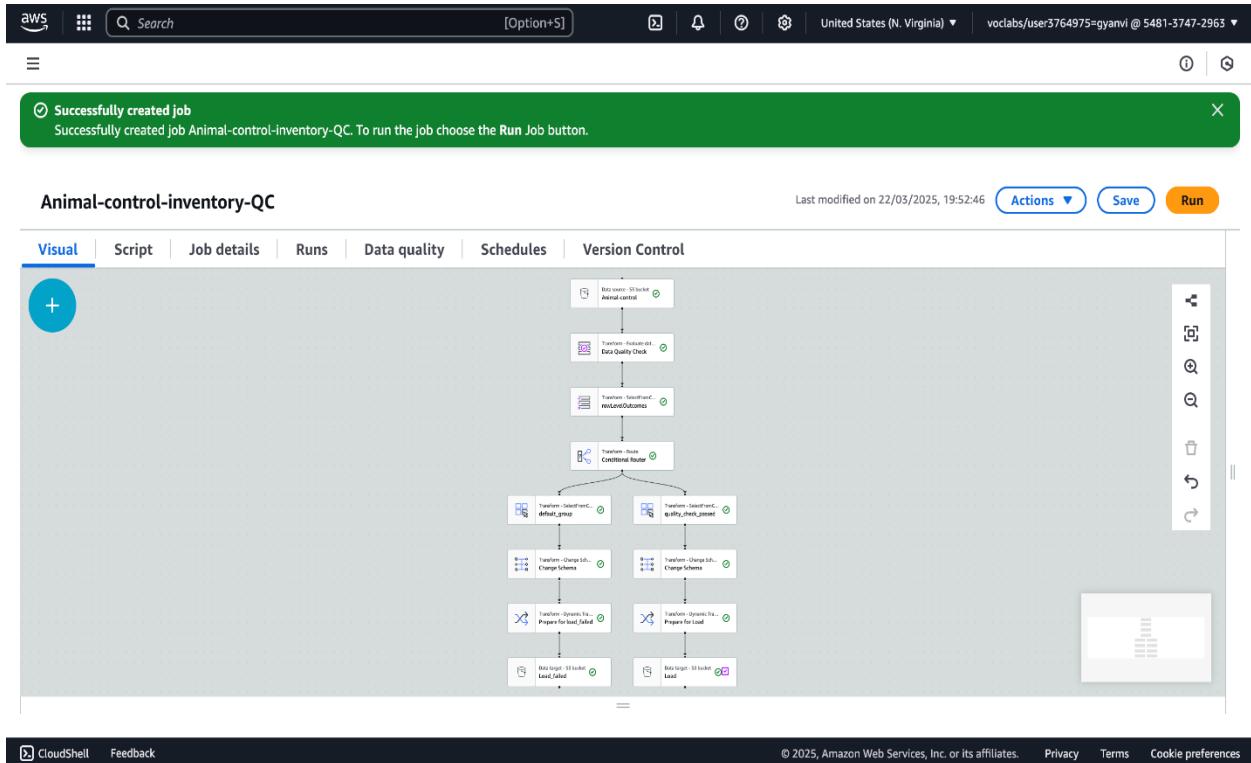
Note. Evaluate data quality under the transform section in the ETL pipeline. (AWS Glue, *self-work*).

Two of the rules have passed and one has failed based on the conditions provided under the ruleset editor. Now, the following steps have used transform features like transform-route for creating two groups called, ‘*quality_check_passed*’ and the other is the default group.

The logical operator is AND and the key is the Data quality evaluation result. The operation is ‘matches’ as “Passed” is the string value. By employing the usual steps of dropping

unnecessary columns of data quality check under the Change Schema feature and then using auto-balancing for a single output, we perform the steps for both passed and failed sets.

Figure 11: ETL pipeline for animal control inventory quality check



Note. The entire ETL pipeline was created for the data governance step for quality check of animal control inventory. (AWS Glue, *self-work*).

In the transformed bucket, we created a folder called ‘Quality_check’ which has two sub-folders called ‘Passed’ and ‘Failed’. The single CSV output for passed goes into the passed folder and the same goes for the failed folder.

Figure 12: Job output for ‘Passed’ folder

Amazon S3 < passed/

Objects Properties

Objects (1)

| Name | Type | Last modified | Size | Storage class |
|--------------------------------|------|--------------------------------------|--------|---------------|
| run-1742698471897-part-r-00000 | - | March 22, 2025, 19:54:48 (UTC-07:00) | 1.5 KB | Standard |

Find objects by prefix Show versions

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Note. AWS S3 transformed bucket containing the output of the passed data quality check (AWS S3, self-work).

Figure 13: Job output for the ‘Failed’ folder

Amazon S3 < failed/

Objects Properties

Objects (1)

| Name | Type | Last modified | Size | Storage class |
|--------------------------------|------|--------------------------------------|---------|---------------|
| run-1742698487904-part-r-00000 | - | March 22, 2025, 19:54:51 (UTC-07:00) | 68.5 KB | Standard |

Find objects by prefix Show versions

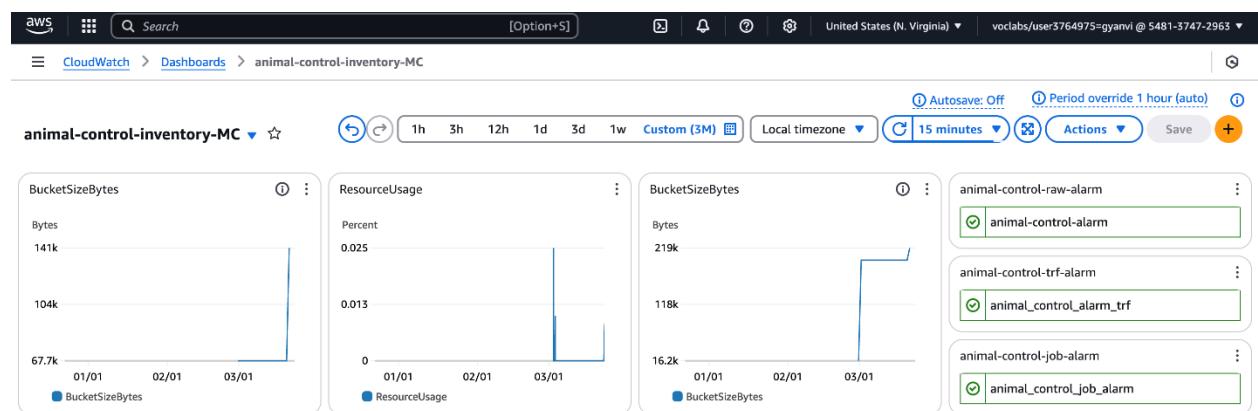
CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Note. AWS S3 transformed bucket containing the output of the failed data quality check (AWS S3, self-work).

Step 8: Data Monitoring

For data monitoring, we have used the AWS CloudWatch feature that monitors resource usage which is essential in cost and storage optimization. The dashboard created is named ‘animal-control-inventory-MC’ which contains 3 metrics and 3 alarms for the respective metrics. A simple line widget is used for the three metrics- Raw S3 bucket, Transformed S3 bucket and Glue JobRun. The timeline is customized for 3 months.

Figure 14: Dashboard created on CloudWatch



Note. The overview of the dashboard contains the three metrics and three alarms. (AWS CloudWatch, self-work).

Next, the alarms created for the S3 service have an average as the stat and 1 day as the time period. The Glue service has a time period of 5 minutes.

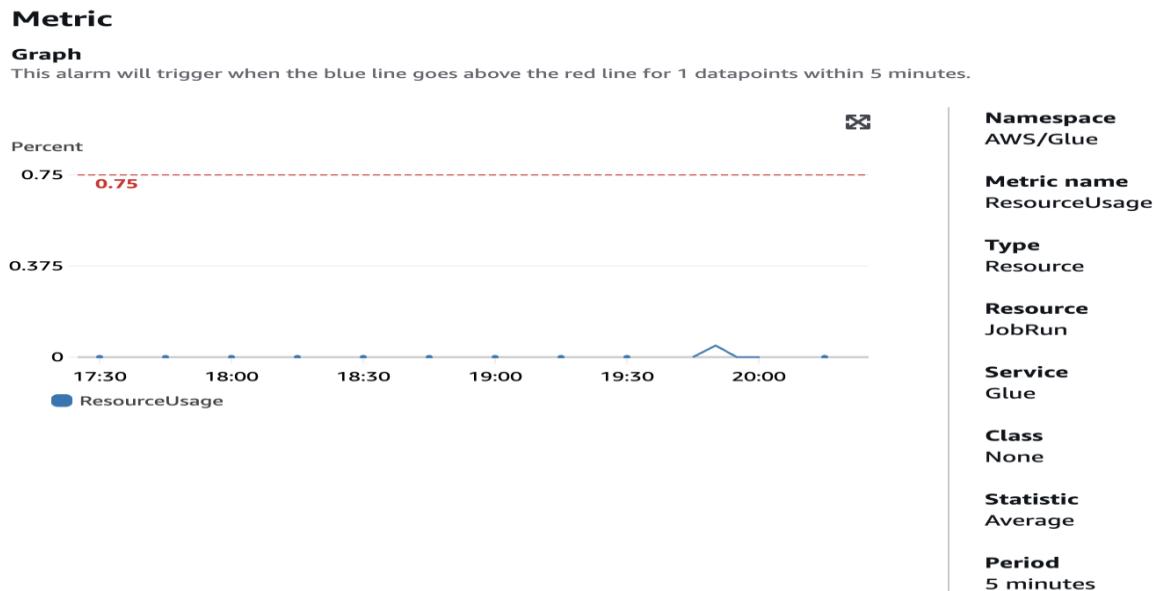
The threshold is set for the three metrics as:

1. Raw S3 bucket bucket size bytes: 180,000

2. Transformed S3 bucket size bytes: 300,000

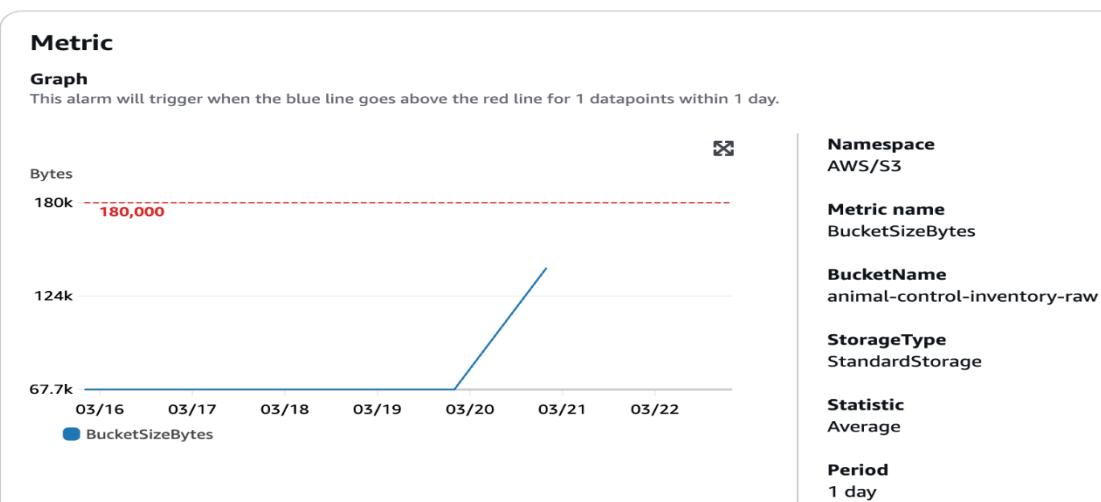
3. JobRun percent: 0.75

Figure 15: AWS Glue alarm



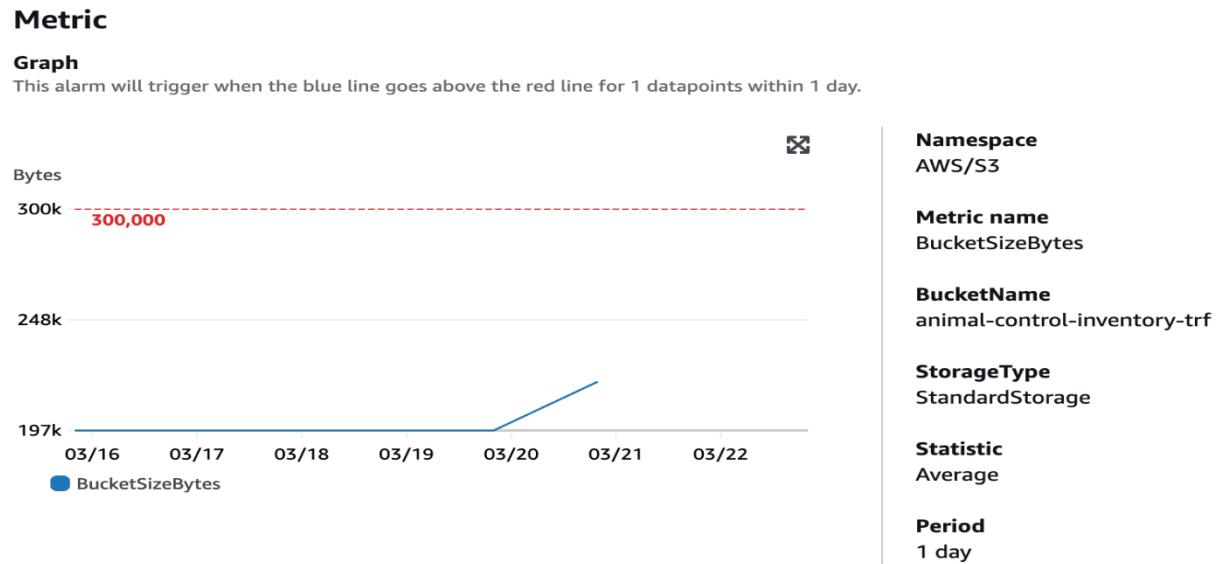
Note. Overview of the AWS Glue alarm with 0.75 percent as threshold. (AWS CloudWatch, *self-work*).

Figure 16: AWS S3 Raw bucket alarm



Note. Overview of the AWS S3 Raw bucket alarm with 180,000 bytes as threshold (AWS CloudWatch, *self-work*).

Figure 17: AWS S3 transformed bucket alarm



Note. Overview of the AWS S3 Transformed bucket alarm with 300,000 bytes as threshold (AWS CloudWatch, *self-work*).

We can use AWS Cloudtrail to monitor user activity and understand patterns of usage.

For this, we have created a trail called ‘animal-control-inventory’ which contains all steps taken during the session.

Figure 18: AWS CloudTrail used to create a trail

The screenshot shows the AWS CloudTrail Trails page. At the top, there is a banner with the text "What's new: Strengthen your data perimeter and implement better detective controls for your VPC endpoints by enabling Network activity events on your Trail or CloudTrail Lake. Learn more". Below the banner, the page title is "Trails". There is a table with the following columns: Name, Home region, Multi-region trail, Insights, Organization trail, S3 bucket, Log file prefix, CloudWatch Logs log group, and Status. The single row in the table is for a trail named "animal-control-inventory" located in "US East (N. Virginia)". The status is "Logging".

| Name | Home region | Multi-region trail | Insights | Organization trail | S3 bucket | Log file prefix | CloudWatch Logs log group | Status |
|------------------------------------------|-----------------------|--------------------|----------|--------------------|-------------------------------------------|-----------------|---------------------------|----------------------|
| animal-control-inventory | US East (N. Virginia) | Yes | Disabled | No | aws-cloudtrail-logs-548137472963-10f38656 | - | - | Logging |

Note. Trail created in AWS CloudTrail to understand user activity and usage patterns for cost and resource optimization (AWS CloudTrail, *self-work*).

DAP Implementation (Individual work – Peng Wang)

Step 5: Data Analysis

Figure 19: Semi-dataset jobs

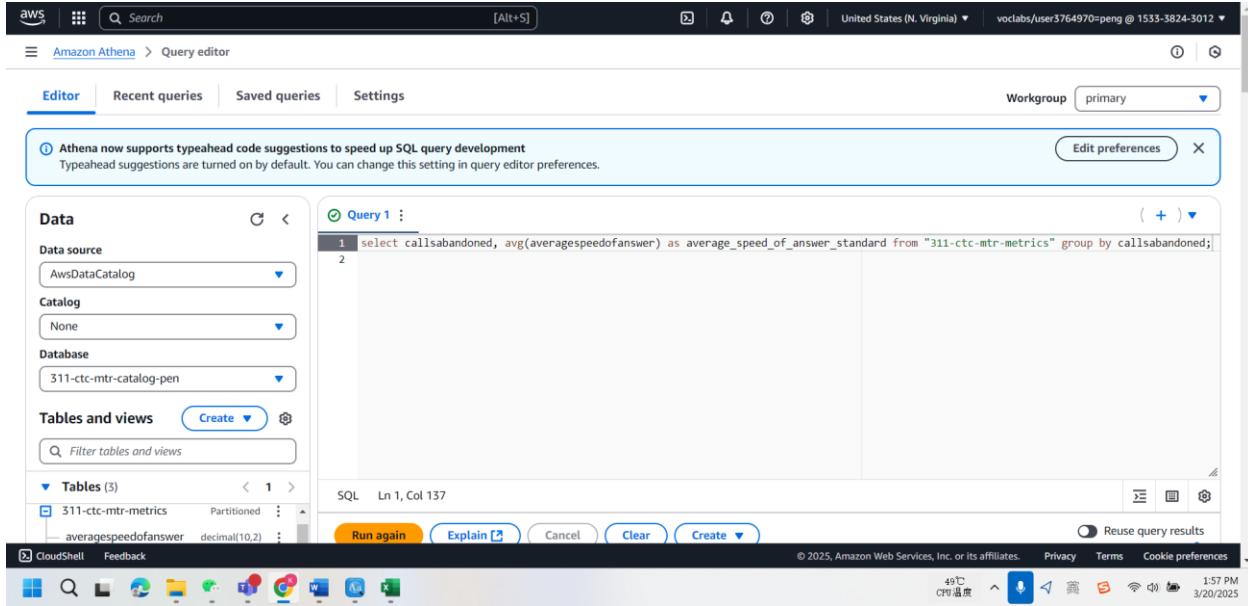
The screenshot shows the AWS DataBrew Jobs page. The left sidebar includes icons for Datasets, Projects, Recipes, and DQ Rules. The main area shows a table titled "Recipe jobs (2) Info". The table has columns: Job name, Status, Job input, Job output, Last run, Created on, Created by, and Tags. Two rows are listed:

| Job name | Status | Job input | Job output | Last run | Created on | Created by | Tags |
|--------------------------|------------------------|------------------------------------------------------------------------------|------------|------------------------------------------|------------------------------------------|------------|------|
| 311-ctc-mtr-semi-pri-pen | Succeeded | 311-ctc-mtr-... (311-ctc-mtr-... + 311-ctc-mtr-...) Project Dataset Recipe | 2 outputs | 2 minutes ago March 20, 2025, 9:43:04 am | 5 minutes ago March 20, 2025, 9:40:36 am | voclabs | - |
| 311-ctc-mtr-cln-pen | Succeeded | 311-ctc-mtr-... (311-ctc-mtr-... + 311-ctc-mtr-...) Project Dataset Recipe | 2 outputs | 17 days ago March 2, 2025, 11:10:26 pm | 17 days ago March 2, 2025, 11:07:22 pm | voclabs | - |

Note. The screenshot shows the semi-dataset jobs. Source from AWS.

Explanation: I created the semi-dataset, but I found I have already created the cleaned dataset, which can be used for data analysis, so I think it is not necessary to create the semi-dataset.

Figure 20: SQL Syntax in Athena



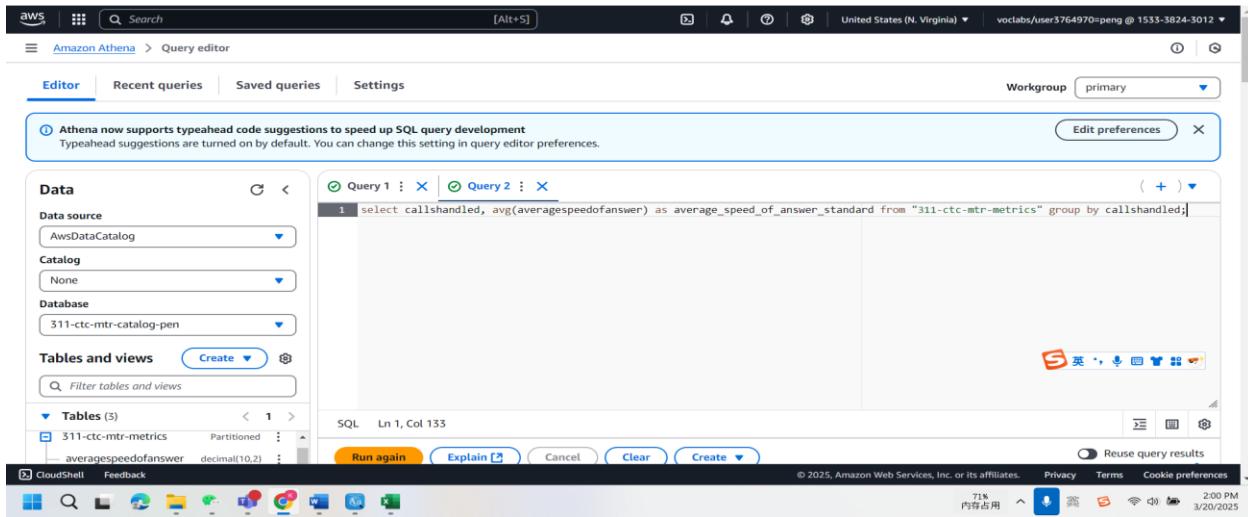
The screenshot shows the AWS Lambda interface with the Athena Query editor open. The left sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (311-ctc-mtr-catalog-pen). The Tables and views section shows a single table named '311-ctc-mtr-metrics'. The main query editor window contains the following SQL code:

```
1 select callsabandoned, avg(averagespeedofanswer) as average_speed_of_answer_standard from "311-ctc-mtr-metrics" group by callsabandoned;
```

The status bar at the bottom right indicates the date as 3/20/2025.

Note. The screenshot shows the first SQL syntax in Athena. Source from AWS.

Figure 21: SQL Syntax in Athena



The screenshot shows the AWS Lambda interface with the Athena Query editor open. The left sidebar displays the Data source (AwsDataCatalog), Catalog (None), and Database (311-ctc-mtr-catalog-pen). The Tables and views section shows a single table named '311-ctc-mtr-metrics'. The main query editor window contains the following SQL code:

```
1 select callhandled, avg(averagespeedofanswer) as average_speed_of_answer_standard from "311-ctc-mtr-metrics" group by callhandled;
```

The status bar at the bottom right indicates the date as 3/20/2025.

Note. The screenshot shows the second SQL syntax in Athena. Source from AWS.

Explanation: I analyzed through SQL syntax in Athena. That is because Athena provides the functions to proceed with business questions by SQL syntax. There are two questions that need

to be analyzed. The first one is the calls abandoned analysis, compared with an average speed of answer. The second one is the calls handled analysis, compared with an average speed of answer. Specifically, I set the standard value as the average of average speed of the answer. In this case, calls abandoned data and calls handled data can be shown whether it is more than this standard value or less than the standard value.

Step 6: Data Security

Figure 22: Key Management Service

The screenshot shows the AWS KMS (Key Management Service) console. The left sidebar navigation includes 'Key Management Service (KMS)', 'AWS managed keys', 'Customer managed keys' (which is selected), and 'Custom key stores' (which has 'AWS CloudHSM key stores' and 'External key stores' listed under it). The main content area is titled 'Customer managed keys (1)' and displays a single key named '311-ctc-mtr-key-pen'. The table columns are 'Aliases', 'Key ID', 'Status', 'Key type', 'Key spec', and 'Key usage'. The key's details are: Aliases (empty), Key ID (de487171-bf21-47ef...), Status (Enabled), Key type (Symmetric), Key spec (SYMMETRIC_DEFAULT), and Key usage (Encrypt and decrypt). There are 'Key actions' and 'Create key' buttons at the top right of the table. The bottom of the screen shows the Windows taskbar with various icons and the system tray.

Note. The screenshot shows the key management service. Source from AWS.

Figure 23: Raw Bucket Versioning and Default Encryption

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning
Enabled

Multi-factor authentication (MFA) delete
An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. [Learn more](#)

Tags (0)
You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

| Key | Value |
|----------------------------------------|-------|
| No tags associated with this resource. | |

Default encryption [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)
Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN
[arn:aws:kms:us-east-1:153338243012:key/de487171-bf21-47ef-ad92-813d277ff815](#)

Bucket Key
When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. [Learn more](#)

Enabled

Default encryption [Info](#)

Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)
Server-side encryption with AWS Key Management Service keys (SSE-KMS)

Encryption key ARN
[arn:aws:kms:us-east-1:153338243012:key/de487171-bf21-47ef-ad92-813d277ff815](#)

Bucket Key
When KMS encryption is used to encrypt new objects in this bucket, the bucket key reduces encryption costs by lowering calls to AWS KMS. [Learn more](#)

Enabled

Intelligent-Tiering Archive configurations (0)

Enable objects stored in the Intelligent-Tiering storage class to tier-down to the Archive Access tier or the Deep Archive Access tier which are optimized for objects that will be rarely accessed for long periods of time. [Learn more](#)

| Name | Status | Scope | Days until transition to Ar... | Days until transition to D... |
|------------------------------------------------------------|--------|-------|--------------------------------|-------------------------------|
| No archive configurations No configurations to display. | | | | |

[Create configuration](#)

Note. The screenshot shows the raw bucket versioning and default encryption. Source from AWS.

Figure 24: Replication Rules in Raw Bucket

Replication rules (1)

| Replication rule name | Status | Destination bucket | Destination Region | Priority | Scope | Storage class | Replica owner | KMS-encrypted objects KMS or DSSE-KI |
|-------------------------|---------|--------------------------|---------------------------------|----------|---------------|----------------|----------------|----------------------------------------|
| 311-ctc-mtr-rep-rul-pen | Enabled | s3://311-ctc-mtr-bac-pen | US East (N. Virginia) us-east-1 | 0 | Entire bucket | Same as source | Same as source | Disabled |

Inventory configurations (0)

| Name | Status | Scope | Destination | Frequency | Last export | Format |
|-------------------|------------------------------|-------|-------------|-----------|-------------|--------|
| No configurations | No configurations to display | | | | | |

Note. The screenshot shows the replication rules in the raw bucket. Source from AWS.

Figure 25: Transfer Bucket Versioning and Default Encryption

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning
Enabled

Multi-factor authentication (MFA) delete
An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. [Learn more](#)

Tags (0)
You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

Default encryption [Info](#)
Server-side encryption is automatically applied to new objects stored in this bucket.

Encryption type [Info](#)
Server-side encryption with Amazon S3 managed keys (SSE-S3)

The screenshot shows the AWS S3 console for the bucket '311-ctc-mtr-trf'. On the left, the navigation pane includes 'Amazon S3', 'General purpose buckets', 'Storage Lens', and 'Feature spotlight' (CloudShell). The main content area displays the 'Default encryption' configuration, which is set to 'Server-side encryption with Amazon S3 managed keys (SSE-S3)'. It also shows 'Bucket Key' information and a note about KMS encryption costs. Below this is the 'Intelligent-Tiering Archive configurations' section, which is currently empty. The bottom of the screen shows the Windows taskbar with various icons.

Note. The screenshot shows the transfer bucket versioning and default encryption. Source from AWS.

Figure 26: Replication Rules in Transfer Bucket

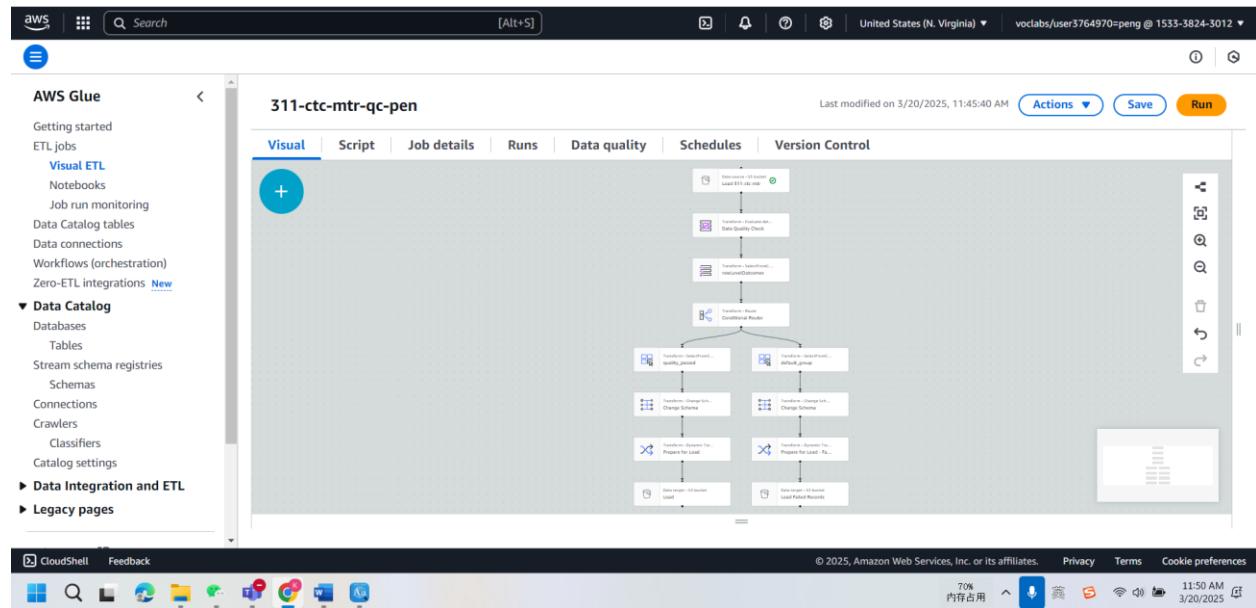
The screenshot shows the AWS S3 console for the bucket '311-ctc-mtr-trf'. The left sidebar includes 'Amazon S3', 'General purpose buckets', 'Storage Lens', and 'Feature spotlight' (CloudShell). The main area displays the 'Replication rules (1)' section, showing one rule named '311-ctc-mtr-rep-rul-pen' that replicates objects to the 'us-east-1' region. Below this is the 'Inventory configurations (0)' section, which is currently empty. The bottom of the screen shows the Windows taskbar.

Note. The screenshot shows the replication rules in the transfer bucket. Source from AWS.

Explanation: I set the key through KMS, and then, I set the bucket versioning and default encryption. Finally, it is necessary to set the replication rules in the raw bucket and transfer bucket. The reason why I set the key in KMS is that KMS keys can reduce the risk and encrypt the sensitive data, the bucket versioning is mainly to prevent data loss, and the default encryption is mainly to avoid unauthorized access to stored data. Overall, this operation can ensure data security.

Step 7: Data Governance

Figure 27: Quality Check in Visual ETL

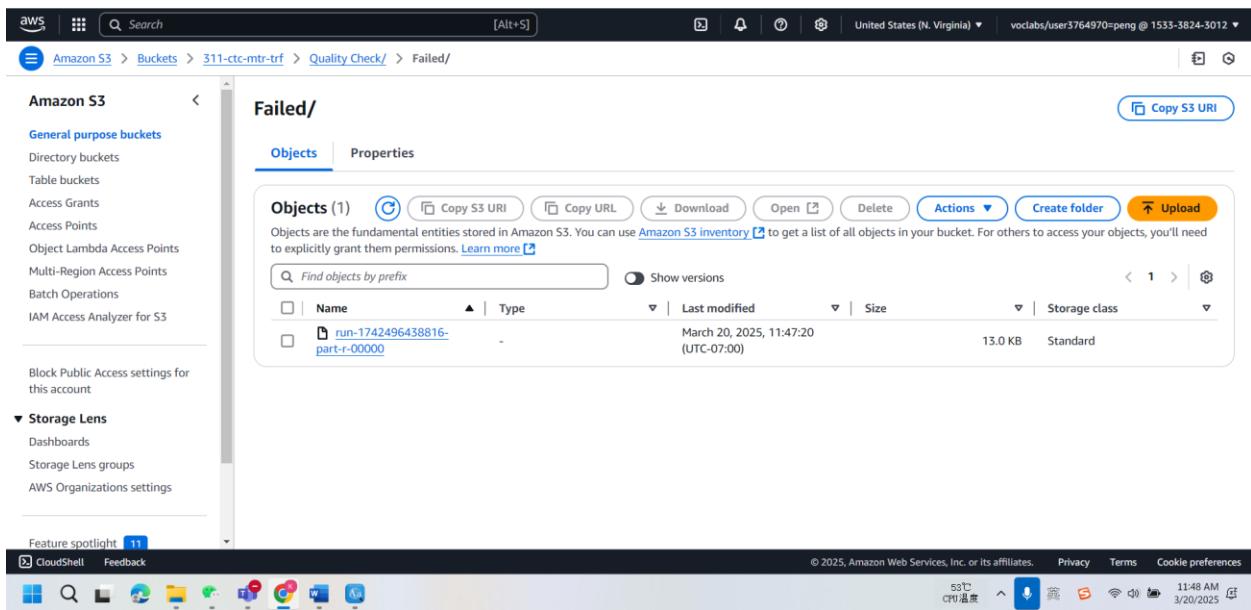


Note. The screenshot shows the visual ETL for quality check. Source from AWS.

Explanation: I did the ruleset for data quality and then set the conditional router, including passed and failed, and then changed the schema before load, including dropping the useless columns. Finally, I loaded the Passed and Failed into the corresponding folders. This part mainly focuses on passed data and failed data based on specific rules.

The whole operation is mainly to organize the data better. For example, the data in the Passed subfolder is cleaned and validated, which can be processed successfully. However, the data in the Failed subfolder is incorrect and incomplete, which cannot be processed better. In this case, better data quality control can be achieved. Moreover, visual ETL is beneficial in reducing errors and speeding up data transformation.

Figure 28: Failed Dataset in Transfer Bucket



Note. The screenshot shows the failed dataset in the transfer bucket. Source from AWS.

Figure 29: Passed Dataset in Transfer Bucket

The screenshot shows the AWS S3 console interface. The left sidebar includes sections for General purpose buckets, Storage Lens, and IAM Access Analyzer. The main content area displays a bucket named 'Passed/'. Inside the bucket, there is one object named 'run-1742496423354-part-r-00000'. The object was last modified on March 20, 2025, at 11:47:19 (UTC-07:00) and has a size of 6.6 KB in Standard storage class. The top navigation bar shows the user is in the United States (N. Virginia) region.

Note. The screenshot shows the passed dataset in the transfer bucket. Source from AWS.

Explanation: After loading the passed and failed in Visual ETL, the corresponding datasets have already been published into each sub-bucket in the transfer bucket, which can be downloaded for different uses.

Step 8: Data Monitoring

Figure 30: Dashboard and Alarms in CloudWatch

The screenshot shows the AWS CloudWatch Metrics dashboard for the bucket '311-ctc-mtr-mcr-pen'. The dashboard displays three metrics over a 3M period: 'BucketSizeBytes' (Bytes), 'ResourceUsage' (Percent), and another 'BucketSizeBytes' (Bytes). The 'ResourceUsage' chart shows a sharp spike from approximately 7e-3 to 0.014 percent around March 1st. On the right side of the dashboard, three CloudWatch Metrics alarms are listed: '311-ctc-mtr-alm-pen', '311-ctc-mtr-alm-trf-pen', and '311-ctc-mtr-alm-jrn-pen'. The bottom navigation bar shows the user is in the United States (N. Virginia) region.

Note. The screenshot shows the dashboard and alarms in CloudWatch. Source from AWS.

Explanation: I established the dashboard in CloudWatch, including the raw bucket and the transfer bucket in S3, and the JobRun in Glue. Finally, I also established three alarms for the raw bucket, transfer bucket, and the jobrun, which were added to the dashboard. Regarding the alarms, it is necessary to set the threshold for each alarm, and the threshold is based on the dataset. For example, I set the threshold as 40000 for the bucket size bytes and 1 day to monitor the data. That is because this threshold is suitable for this bucket, which is originally upper. The period of 1-day monitoring data is mainly to reduce the cost. Finally, the dashboard can visualize the comprehensive view of key metrics, which can achieve the goal of monitoring the data, and alarms are mainly related to thresholds, which can monitor the issue before impacting users.

Figure 31: Trail in CloudTrail

The screenshot displays the AWS CloudTrail console interface. The top navigation bar shows the AWS logo, a search bar, and the user's email address: `voclabs/user3764970=peng @ 1533-3824-5012`. Below the navigation bar, the left sidebar is open, showing the 'CloudTrail' section with 'Trails' selected. The main content area shows the configuration for a trail named '311-ctc-mtr-tra-pen'. The 'General details' section includes fields for 'Trail logging' (set to 'Logging'), 'Trail name' ('311-ctc-mtr-tra-pen'), 'Multi-region trail' (set to 'Yes'), and 'Apply trail to my organization' (set to 'Not enabled'). The 'CloudWatch Logs' section indicates 'No CloudWatch Logs log groups' and 'CloudWatch Logs is not configured for this trail'. On the far right, there are 'Delete' and 'Stop logging' buttons, and an 'Edit' button for the general details. The bottom of the screen shows the Windows taskbar with various pinned icons and the system tray.

Note. The screenshot shows the trail in CloudTrail. Source from AWS.

Explanation: I established the trail in CloudTrail. That is because I think this operation is essential to do the data monitoring by detecting threats and unauthorized access.

DAP Implementation (Individual work - Muhammad Zulqarnain Shahzad)

Step 5: Data Analysis

Figure 32: Data Brew Projects – Semi Dataset Setup

| Project name | Associated dataset | Attached recipe | Jobs | Create date | Created by | In use by | Tags |
|-----------------------|--------------------------------------------------------|------------------------------|-------------------------|----------------------------------------------|------------|-----------------------------|------|
| bl-cms-semi-prj-zul | Business Licences - Consulting and Management Services | bl-cms-semi-prf-recipe | bl-cms-cln-semi-prj-zul | 15 minutes ago March 22, 2025, 4:29:41 pm | vodabs | vodabs/user3764972=zayerayz | - |
| bl-cms-data-profiling | Business Licences - Consulting and Management Services | bl-cms-data-profiling-recipe | bl-cms-cln-zul | 20 days ago March 2, 2025, 5:48:01 pm | vodabs | vodabs/user3764972=zayerayz | - |

Note. This screenshot shows that I created a new Glue DataBrew project named bl-cms-semi-prj-zul to work on my semi dataset based on the raw version of Business Licences – Consulting and Management Services. Source: AWS Glue DataBrew, *self-work*

Explanation

I just copy my old project from part 1 and made new one for part 2 to show I can handle semi dataset too. It's same data but just to make it look structured like everyone. Professor said

we don't need new data, but structure is important. So, I use same dataset but made new project with new name.

Figure 33: Data Brew Recipe Jobs – Semi Cleaning Job Execution

| Job name | Status | Job input | Job output | Last run | Created on | Created by | Tags |
|-------------------------|-----------|------------------------------------------------------------------------|------------|----------------------------------------------|---------------------------------------------|------------|------|
| bl-cms-cln-semi-prj-zul | Succeeded | bl-cms-semi... (Business Lic... + bl-cms-semi...) Project Dataset | 2 outputs | 3 minutes ago March 22, 2025, 4:44:41 pm | 4 minutes ago March 22, 2025, 4:42:50 pm | voclabs | - |
| bl-cms-cln-zul | Succeeded | bl-cms-data... (Business Lic... + bl-cms-data...) Project Dataset | 2 outputs | 20 minutes ago March 22, 2025, 4:27:12 pm | 20 days ago March 2, 2025, 6:18:36 pm | voclabs | - |

Note. This screenshot shows the job bl-cms-cln-semi-prj-zul was created and successfully ran to process the semi dataset and store the output in S3 semi/user and semi/system. Source: AWS Glue DataBrew, *self-work*

Explanation

This is the job I made from the semi project to do a basic run on the dataset. I didn't need to do much cleaning again because it was already clean from before. But I still ran the job just to complete the step and make sure I can track how the data flows from raw to semi. It saves the output into the right folders in S3 and shows that I understand how to structure things with different stages like raw, semi, and transform. So, this is more about keeping a clear data process, even if the content didn't change much.

Figure 34: Business Licenses Issued by Year – Athena SQL Query

The screenshot shows the AWS Athena Query Editor interface. On the left, the Data sidebar displays the 'cityofvancouver_b1_cms_trf_system' table with columns: folderyear, licensern, licencenumber, licenserevisionnumber, businessname, businesstradename, status, issueddate, expireddate, and businessstype. The main area shows a query in 'Query 1':

```
1 SELECT year(issue_date) AS issue_year, COUNT(*) AS total_licenses FROM cityofvancouver_b1_cms_trf_system GROUP BY year(issue_date) ORDER BY issue_year ASC;
```

The results table shows two rows of data:

| # | issue_year | total_licenses |
|---|------------|----------------|
| 1 | 2024 | 550 |
| 2 | 2025 | 16 |

Note. This query shows the number of consulting licenses issued each year using the issued date field. The result shows 550 licenses in 2024. Source: Self-work, AWS Athena Query Editor

Explanation

I wanted to see like how many licenses are given each year, so I ran this query. It's just counting how many new consulting businesses got licenses by year. 2024 is big, 2025 is just started. This helps to see the trend and business activity, if it's going up or down.

Figure 35: Top 10 Business Trade Names by License Count – Athena SQL Query

The screenshot shows the AWS Athena Query Editor interface. At the top, there's a banner for 'Introducing a new tutorial' and a note about typeahead code suggestions. The main area has tabs for 'Editor', 'Recent queries', 'Saved queries', and 'Settings'. A 'Workgroup' dropdown is set to 'primary'. The 'Data' sidebar shows a 'Tables (2)' section with the 'cityofvancouver_bl_cms_trf_system' table selected. This table has columns: folderyear, licensrn, licensenumber, licenserevisionnumber, businessname, businesstradename, status, issuedate, expiredate, and lastupdate. The 'Query 1' tab displays a SQL query: 'SELECT businesstradename, COUNT(*) AS license_count FROM cityofvancouver_bl_cms_trf_system GROUP BY businesstradename ORDER BY license_count DESC LIMIT 10;'. Below the query is a 'Run again' button and an 'Explain' button. The 'Query results' section shows a table with one row: businesstradename (1) and license_count (324). Other buttons include 'Copy' and 'Download results CSV'. The bottom of the screen includes standard AWS footer links.

Note. This query returns the most frequent business trade names from the consulting dataset, ordered by license count in descending order. Source: Self-work, AWS Athena Query Editor

Explanation

This one show which business names come most often in the consulting list. I used COUNT to count how many times each name comes. Then I sorted from biggest to smallest. The top one has 324, that's a lot. It helps to know who is most active or popular in this business.

Figure 36: License Status Breakdown – Active vs Inactive

The screenshot shows the AWS Athena Query Editor interface. On the left, there's a sidebar with 'Data' settings, including 'Data source' set to 'AwsDataCatalog', 'Catalogue' set to 'None', and 'Database' set to 'cityofvancouver-bl-cms-data-catalog-zul'. Below this is a 'Tables and views' section with a 'Create' button and a search bar. A table listing is shown with 2 rows, one of which is expanded to show columns like 'status', 'count', and 'total'. The main area contains a query editor tab titled 'Query 1' with the SQL command: 'SELECT status, COUNT(*) AS total FROM cityofvancouver_bl_cms_trf_system GROUP BY status;'. Below the editor is a 'Results' section showing the output of the query:

| # | status | total |
|---|----------------------|-------|
| 1 | Inactive | 12 |
| 2 | Issued | 552 |
| 3 | Gone Out of Business | 2 |

Note. This query shows the distribution of license statuses, including issued, inactive, and gone out of business. Source: Self-work, AWS Athena Query Editor

Explanation

Here I checked how many licenses are active and how many are not. So, I just grouped by status and counted. Most of them are issued (like active), but some are inactive or gone out of business. It's useful to know which ones are still running and which stopped.

Step 6: Data Security

Figure 37: AWS KMS Key Creation – bl-cms-key-zul

The screenshot shows the AWS KMS console under the 'Customer-managed keys' section. A success message at the top states: 'Your AWS KMS key was created with alias bl-cms-key-zul and key ID 0cda23a8-3e4a-4cdf-ba0c-25f79a6f09fa.' Below this, a table lists the key details:

| Aliases | Key ID | Status | Key type | Key spec | Key usage |
|--------------------------------|--------------------------------------------------|---------|-----------|-------------------|---------------------|
| bl-cms-key-zul | 0cda23a8-3e4a-4cdf-ba0c-25f79... | Enabled | Symmetric | SYMMETRIC_DEFAULT | Encrypt and decrypt |

At the bottom of the page, there are links for CloudShell, Feedback, and copyright information: © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences.

Note. This screenshot shows the KMS key I created in AWS for encrypting S3 data. The key is symmetric and will be used for both encryption and decryption of business license files. Source: Self-work, AWS Key Management Service

Explanation

This is my own KMS key. I make this to lock my files in S3 buckets. I named it bl-cms-key-zul to match my consulting dataset project. It's symmetric so I can use it for both encrypt and decrypt. I will use this in S3 buckets in next steps to make the data secure.

Figure 38: S3 Bucket Encryption and Versioning – Raw Zone

Note. This screenshot shows the settings of my raw S3 bucket bl-cms-dec-2024-raw-zul. I enabled versioning and set default encryption using the KMS key I created earlier. Source: Self-work, AWS S3 Bucket Settings

Explanation

Here I turn on versioning and encryption for my raw bucket. Versioning is for saving old copies if something changes or delete. Encryption is to protect my raw business license files. I select my KMS key to lock them. Now every file I upload here will be safe and can't be read by others.

Figure 39: S3 Replication and Lifecycle Rules for Raw Bucket

The screenshot shows the AWS S3 Management console for the bucket 'bl-cms-dec-2024-raw-zul'. The left sidebar includes links for General purpose buckets, Storage Lens, and AWS Marketplace for S3. The main content area has tabs for Objects, Metadata, Properties, Permissions, Metrics, Management, and Access Points. The Management tab is selected.

Lifecycle configuration: A note says 'To manage your objects so that they are stored cost effectively throughout their lifecycle, configure their lifecycle. A lifecycle configuration is a set of rules that define actions that Amazon S3 applies to a group of objects. Lifecycle rules run once per day.' It shows a 'Default minimum object size for transitions' of 'All storage classes 128K'. There is one rule named 'bl-cms-dec-2024-lr-zul' which is Enabled and set to Transition to Glacier Instant Ret.

Lifecycle rules (1): A table with columns: Lifecycle rule name, Status, Scope, Current version actions, Noncurrent versions actions, Expired object delete mark..., and Incomplete multipart up... The rule 'bl-cms-dec-2024-lr-zul' is listed as Enabled, Filtered, and Transition to Glacier Instant Ret.

Replication rules (1): A table with columns: Replication rule name, Status, Destination bucket, Destination Region, Priority, Scope, Storage class, Replica owner, Replication Time Control, KMS-encrypted objects (SSE-KMS or DSSE-KMS), and Replica modification sync. The rule 'bl-cms-rep-rul-zul' is listed as Enabled, replicating to 's3://bl-cms-backup-zul' in US East (N. Virginia) us-east-1, with an entire bucket scope and same source replica.

Inventory configurations (0): A table with columns: Name, Status, Scope, Destination, Frequency, Last export, and Format. No configurations are listed.

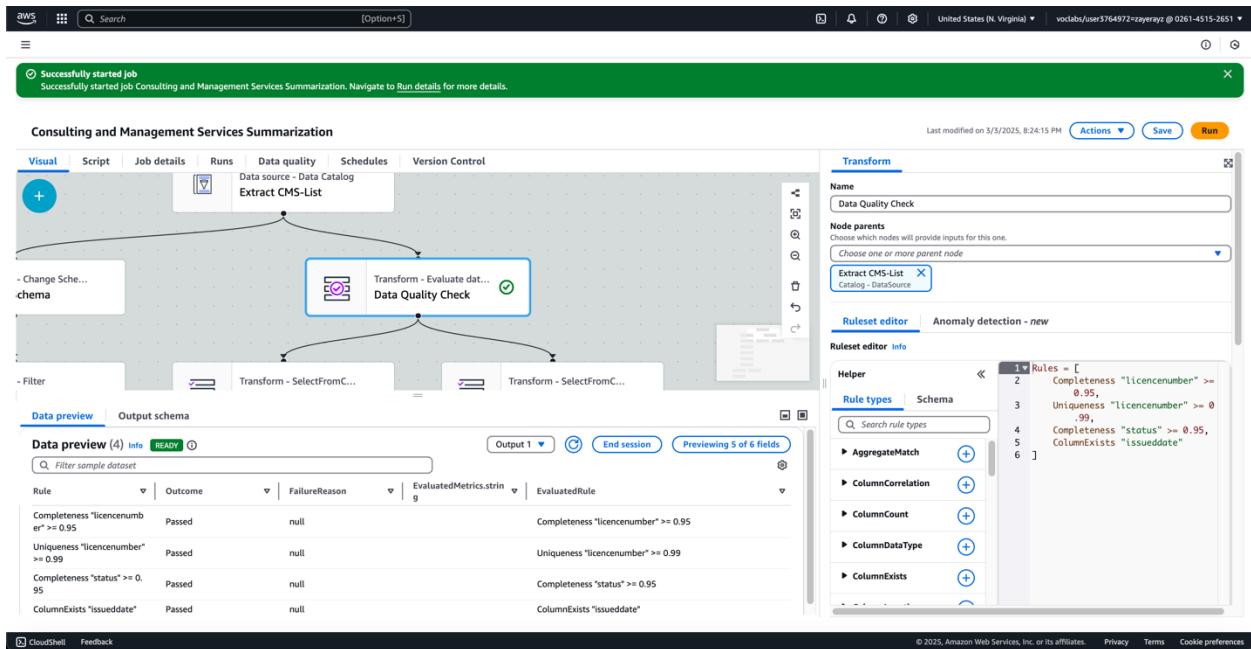
Note. This screenshot shows the replication and lifecycle settings applied to my raw zone bucket. The replication rule sends data to a backup bucket using the same KMS key. The lifecycle rule is used to manage long-term storage. Source: Self-work, AWS S3 Management Tab

Explanation

In this I setup replication from my raw bucket to another backup one. So, my files don't get lost if main bucket got error. I also made lifecycle rule to move old files to glacier storage after some time. It helps to save cost and make system more professional and safer. Replication is using same KMS key I made before.

Step 7: Data Governance

Figure 40: Data Quality Rules, Execution



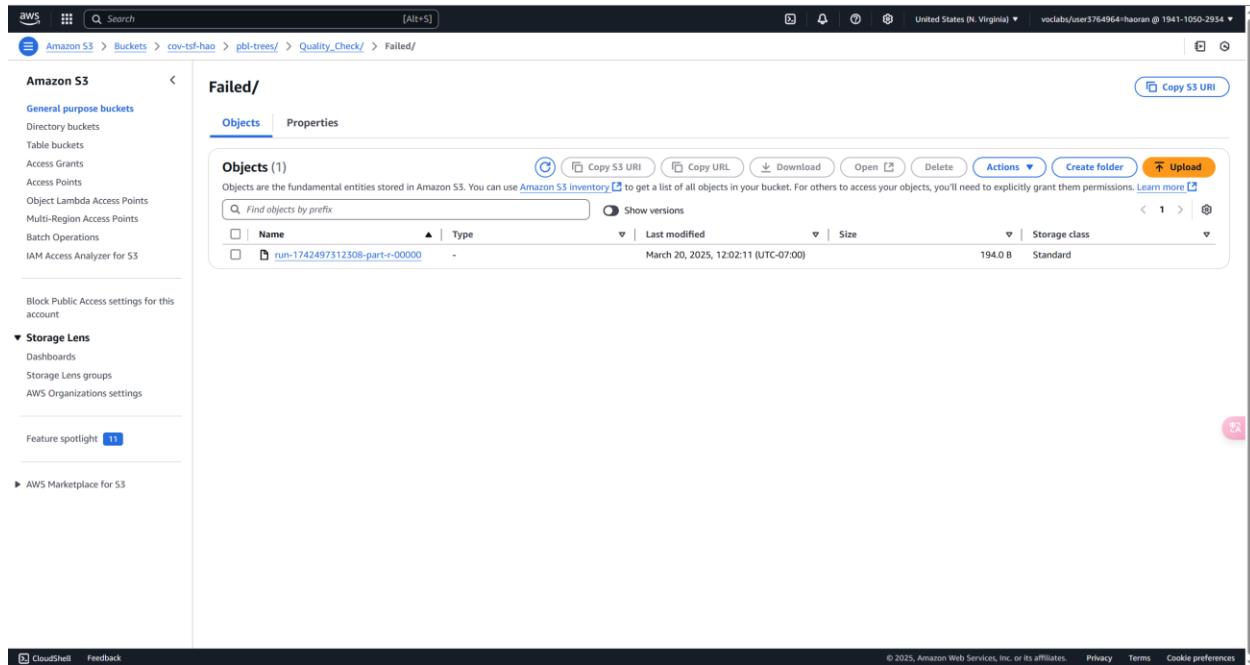
Note. This screenshot shows the main data quality rules I created and executed in AWS Glue Studio. It includes completeness check on “licencenumber” and “status” columns ($\geq 95\%$), uniqueness for “licencenumber” ($\geq 99\%$), and existence check for the “issueddate” column.

Source: Self-work, AWS Glue Studio – Data Quality Rule Editor

Explanation

This was the part where I added my data quality check block to the ETL pipeline. I tried to make sure important fields are filled and unique, and check if the column even there. All passed and result came green. I was able to do this part fully, and now data looks ready for next steps.

Figure 41: S3 Output Folder – Failed Results Preview



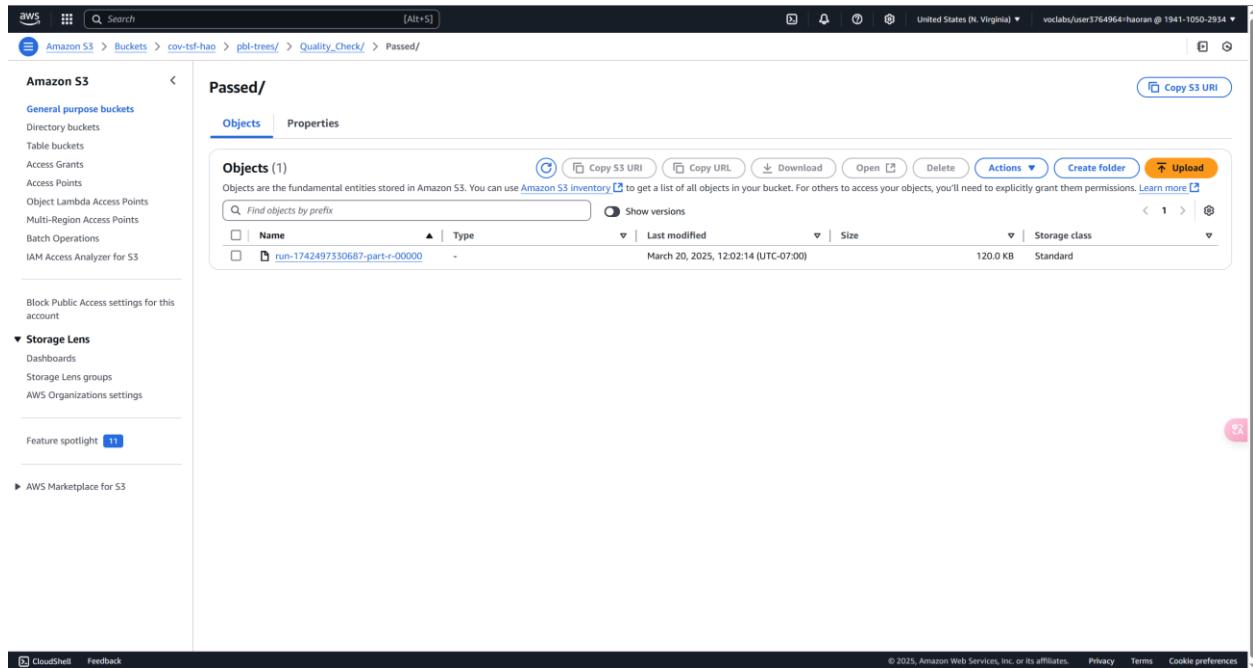
Note. This screenshot shows the Failed folder inside the S3 bucket where failed data quality records are stored. This part was done to separate any data that didn't meet the defined rules.

Source: Peer provided screenshot, AWS S3 – Quality Check Output

Explanation

This part is important to show failed and passed results but honestly, I was struggling here. I couldn't find the right video or remember the exact steps to separate output in passed and failed like this. I understand the logic and how it works, but I couldn't finish it myself so I'm just attaching to show what it looks like.

Figure 42: S3 Output Folder – Passed Results Preview



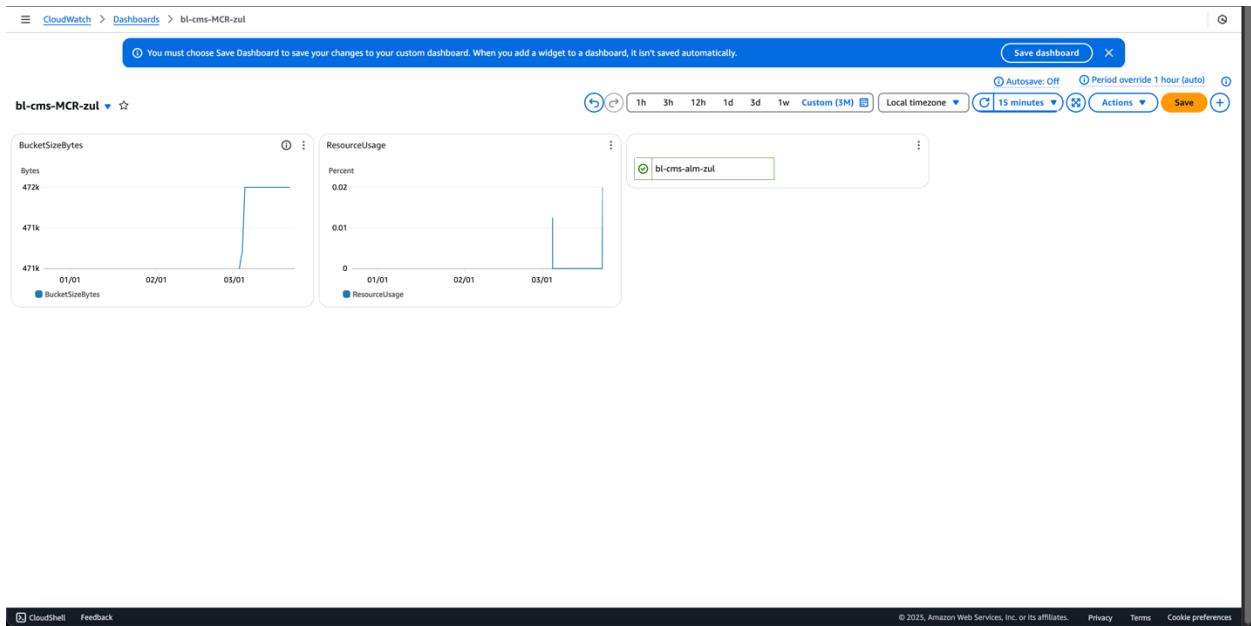
Note. This screenshot shows the Passed folder where records that met all data quality rules were saved. It's part of the validation output done automatically when job is set properly. *Source:* Peer provided screenshot, AWS S3 – Quality Check Output

Explanation

Again, same as before. Just couldn't locate full guide or setup during work session. I understand what's happening in this screenshot and I'll fix mine before final submission. Just showing this to explain I know where it should go.

Step 8: Data Monitoring

Figure 43: Custom CloudWatch Dashboard for Monitoring

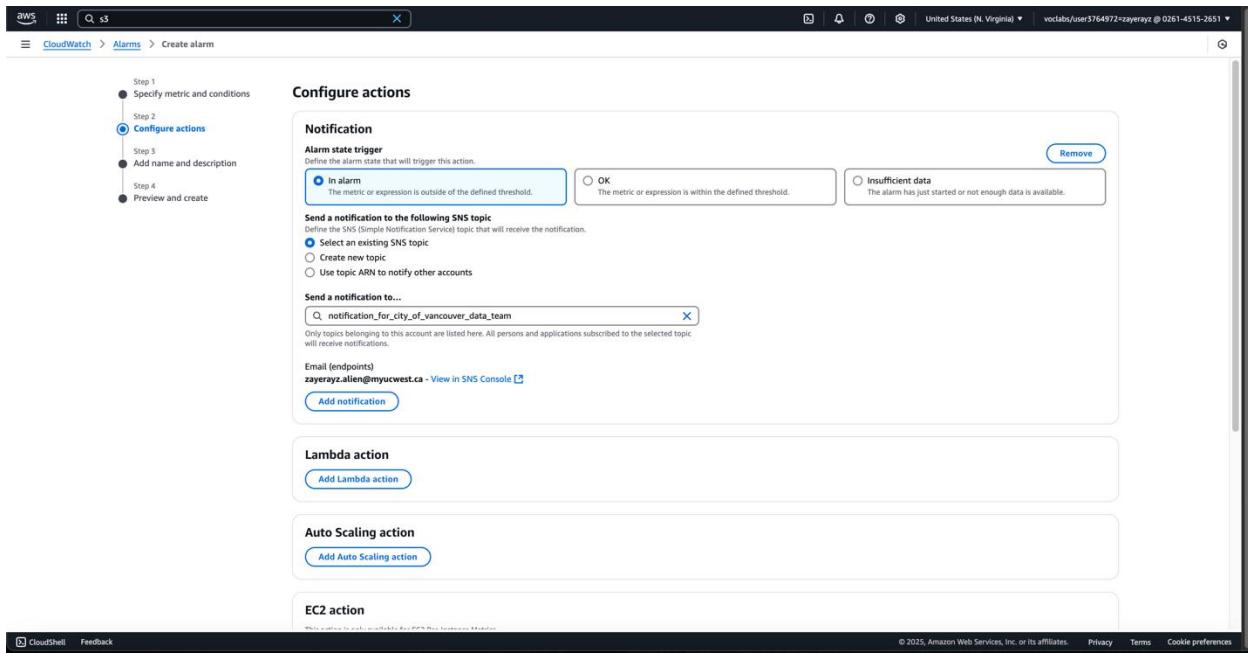


Note. This screenshot shows the custom dashboard bl-cms-MCR-zul with two widgets: one tracking bucket size and another for overall resource usage. *Source:* Self-work, AWS CloudWatch, Dashboards

Explanation

I added dashboard to keep visual on stuff like how much storage I am using and what's happening. I saw bucket size go up in March, which is when I started running my job. It helps to check fast if something crazy happen without opening full AWS pages.

Figure 44: Notification Setup for CloudWatch Alarm



Note. This screenshot shows the setup during alarm creation, with email endpoint zayerayz.alien@myucwest.ca added for receiving bucket alert notifications. *Source:* Self-work, AWS CloudWatch, Alarms, Configure Actions

Explanation

I tried to make email go to me if bucket go above limit. I use topic name like “notification_for_city_of_vancouver_data_team”. I click add notification and put my UCW email. Still waiting on confirm I think, but idea is to know if something weird happen.

Figure 45: CloudWatch Alarm Setup – Billing and Storage Thresholds

The screenshot shows the AWS CloudWatch Alarms interface. At the top, there's a message: "Some subscriptions are pending confirmation. Amazon SNS doesn't send messages to an endpoint until the subscription is confirmed". Below this, the "Alarms (2)" section is displayed. There are two alarms listed:

- bl-cms-alm-zul**: State OK, Last state update 2025-03-22 19:47:47. Condition: BucketSizeBytes > 480000 for 1 datapoints within 1 day. Actions: Actions enabled, Warning.
- Billing-Alarm-Zulqarnain**: State OK, Last state update 2025-03-06 11:27:05. Condition: EstimatedCharges > 10 for 1 datapoints within 6 hours. Actions: Actions enabled.

The left sidebar includes links for Dashboards, AI Operations Preview, Alarms, All alarms, Billing, Logs, Metrics, X-Ray traces, Events, Application Signals, Network Monitoring, Insights, Settings, Telemetry config, Getting Started, and What's new.

Note. This screenshot shows two CloudWatch alarms set up for my project: one for S3 bucket storage usage (bl-cms-alm-zul) and one for estimated billing cost (Billing-Alarm-Zulqarnain). Both are in OK state and active. *Source:* Self-work, AWS CloudWatch, All Alarms

Explanation:

Here I set one new alarm just to be safe and monitor stuff. First one is for storage, I set it to alert me if my S3 bucket goes above 480,000 bytes in one day. I use that to keep an eye on my storage cost and make sure nothing unexpected is filling up the bucket.

Figure 46: CloudTrail Setup with Logging Events

The screenshot shows the AWS CloudTrail Dashboard. On the left, there's a navigation sidebar with options like CloudTrail, Dashboard, Insights, Lake, and Trails. The main area has a banner about strengthening data perimeter controls. Below it, the 'Dashboard Info' section has a 'Query results history' panel (empty) and a 'Trails Info' panel for the trail 'bl-cms-tra-zul' (status: Logging). The 'CloudTrail Insights Info' section indicates Insights are not enabled. The 'Event history Info' section contains a table of recent events:

| Event name | Event time | Event source |
|---------------------|-----------------------------------|--------------------------|
| CreateTrail | March 22, 2025, 19:55:06 (UTC...) | cloudtrail.amazonaws.com |
| StartLogging | March 22, 2025, 19:55:06 (UTC...) | cloudtrail.amazonaws.com |
| PutEventSelectors | March 22, 2025, 19:55:06 (UTC...) | cloudtrail.amazonaws.com |
| PutBucketEncryption | March 22, 2025, 19:55:05 (UTC...) | s3.amazonaws.com |
| PutBucketPolicy | March 22, 2025, 19:55:05 (UTC...) | s3.amazonaws.com |

[View full Event history](#)

Note. This screenshot shows the bl-cms-tra-zul CloudTrail trail created with logging enabled.

Recent logged events include trail creation and S3 policy updates. *Source:* Self-work, AWS CloudTrail, Dashboard

Explanation

Professor, so this is where I set up CloudTrail to track stuff I do in AWS. It shows logs like when I created this trail, turned logging on, added bucket policy, encryption, all that. According to Chat GPT It's not really for querying right now but it's super helpful to see if something weird happen like if someone touch the config or bucket without me knowing. Just keeping track of my own work and for security too. If anything goes wrong, this history helps us to trace back what we did or what someone else did.

DAP Implementation (Individual work – Haoran Xu)

Step 5: Data Analysis

Figure 47: Project Screenshot

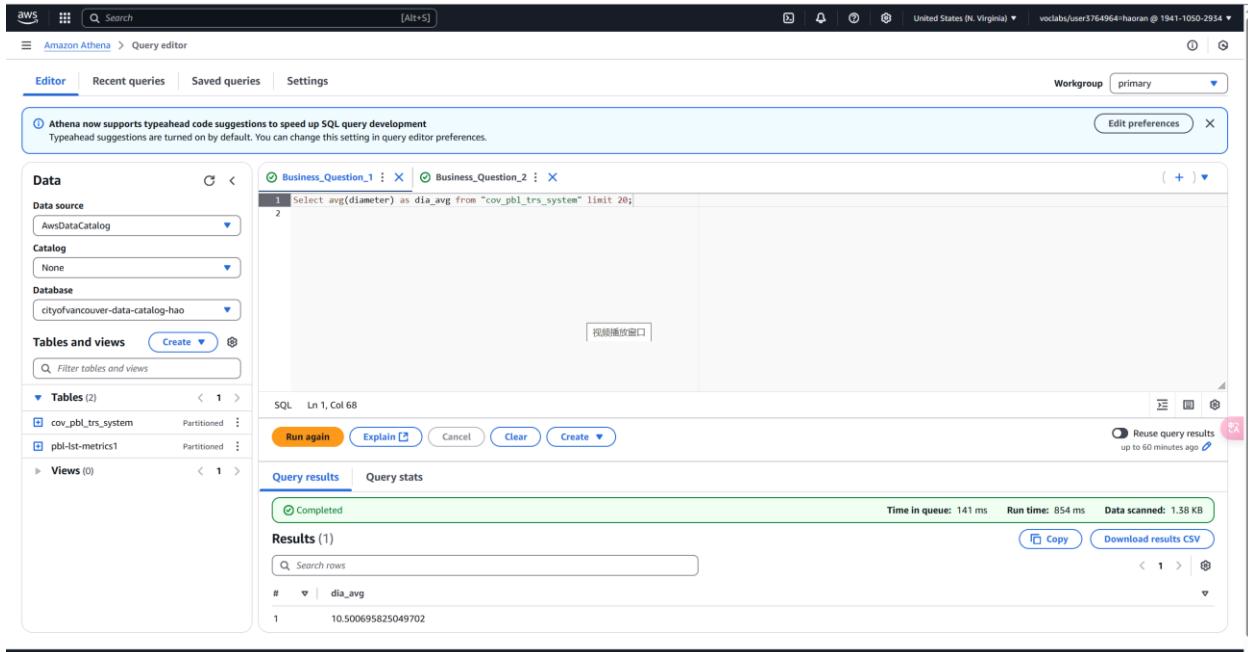
| Project name | Associated dataset | Attached recipe | Jobs | Create date | Created by | In use by | Tags |
|----------------------|------------------------------------------------------------------------------------|-----------------------------|---------------------|---------------------------------------------|------------|----------------------------|------|
| cov-pbl-semi-prj-hao | cov-pbl-tre-ds-hao-50b5fe62d94a5bb84eeae0313ebc11a698435ef098f01dbc5306729d7b9c82a | cov-pbl-semi-prj-hao-recipe | - | 5 minutes ago March 20, 2025, 9:59:33 am | voclabs | voclabs/user3764964=haoran | - |
| cov-pbl-tre-prj-hao | cov-pbl-tre-ds-hao | cov-pbl-tre-prj-hao-recipe | cov-pbl-tre-cln-hao | 19 days ago March 1, 2025, 4:19:43 pm | voclabs | - | - |

Note. This figure shows the screenshot of the project in AWS Glue Data Brew. Source: AWS Glue

I created the project to check if my dataset has a semi-structured issue. But I found that my dataset is cleaned and can be directly used for analysis.

Now it's time to present the business questions in Athena using SQL.

Figure 48: Athena Business Question 1



The screenshot shows the AWS Athena Query Editor interface. On the left, the 'Data' sidebar is open, showing the Data source (AwsDataCatalog), Catalog (None), and Database (cityofvancouver-data-catalog-hao). Under 'Tables and views', there are two tables listed: cov_pbl_trs_system and pbl-lst-metrics1. The main query editor window contains the following SQL code:

```
1 Select avg(diameter) as dia_avg from "cov_pbl_trs_system" limit 20;
```

Below the query, the status bar indicates: SQL, Ln 1, Col 68. The 'Run again' button is highlighted in orange. The results section shows one row of data:

| # | dia_avg |
|---|--------------------|
| 1 | 10.500695825049702 |

At the bottom, the results summary shows: Completed, Time in queue: 141 ms, Run time: 854 ms, Data scanned: 1.38 KB. There are 'Copy' and 'Download results CSV' buttons.

Note. This figure shows the screenshot of the business question 1 in Athena. Source: Athena

The first business question is simple, it will ask for calculating the average of diameter for the first 20 rows data. And make the output column named “dia_avg”.

Figure 49: Athena Business Question 2

```

Business_Question_1 : Business_Question_1
1. select heightrange, avg(diameter) as dia_avg from "cov_pbl_trs_system" group by heightrange limit 20;

SQL Ln 1, Col 102
Run again Explain Cancel Clear Create
Completed
Results (7)
Search rows
# | heightrange | dia_avg
1 | 70-80 | 19.45
2 | 50-60 | 16.142
3 | 60-70 | 18.807692307692307
4 | 10-20 | 4.816901408450704
5 | 40-50 | 13.053571428571429
6 | 30-40 | 10.067901254567902
7 | 20-30 | 7.1151079136669065

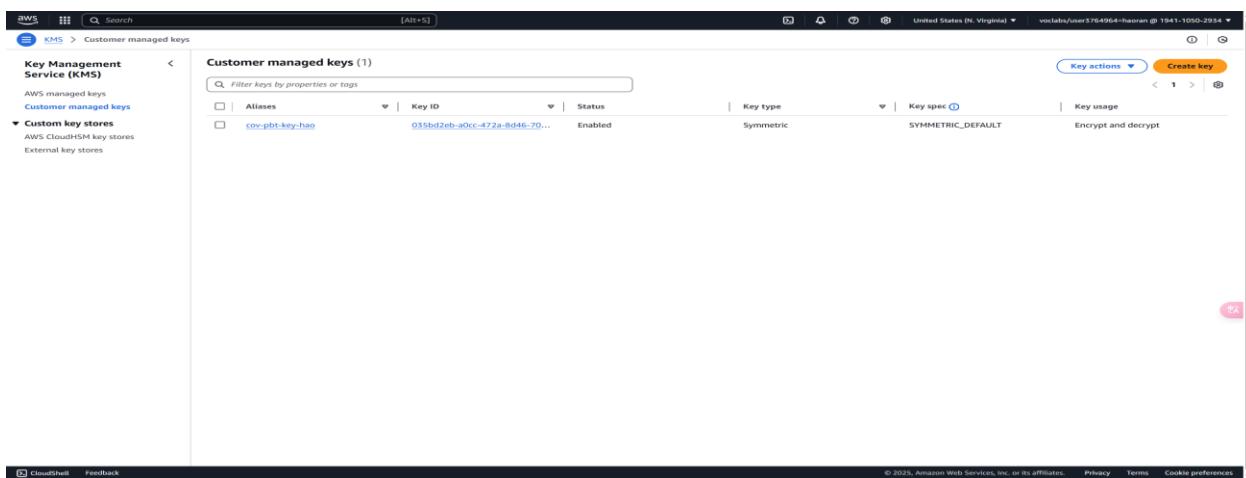
```

Note. This figure shows the screenshot of the business question 2 in Athena. Source: Athena

The second business question will ask whether the average diameter is directly related to height range under different height_range conditions, and group the output according to height_range. We can conclude from the data that average diameter is indeed correlated with height range and is positively correlated.

Step 6: Data Security

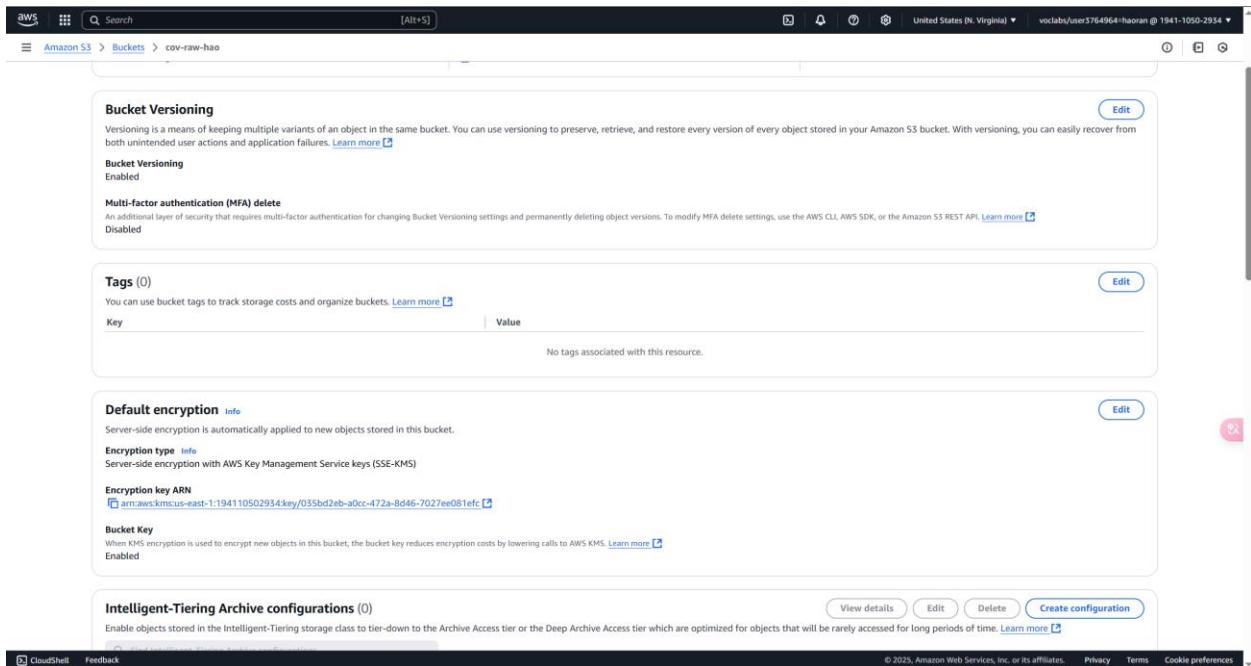
Figure 50: Create KMS Key



Note. This figure shows the screenshot of the key created in AWS KMS. Source: AWS KMS.

Here, I create a key, it's for sharing with Raw bucket and Transfer Bucket.

Figure 51: Raw-bucket-Versioning and Encryption



Note. This figure shows the screenshot of the versioning and encryption of raw bucket created in S3. Source: AWS S3.

Figure 52: Raw-bucket-Replication Rules

The screenshot shows the AWS S3 Management tab for the 'cov-raw-hao' bucket. Under the 'Replication rules' section, there is one rule named 'cov-pbt-rep-rul-hao'. This rule is enabled and replicates objects from the 'cov-raw-bac-hao' bucket in the 'us-east-1' region to the same bucket in the 'us-east-1' region. The rule has a priority of 0, applies to the entire bucket, and uses the same source and destination.

| Replication rule name | Status | Destination bucket | Destination Region | Priority | Scope | Storage class | Replica owner | Replication Time Control | KMS-encrypted objects (SSE-KMS or DSS-E-KMS) | Replica modification sync |
|-----------------------|---------|----------------------|---------------------------------|----------|---------------|----------------|----------------|--------------------------|----------------------------------------------|---------------------------|
| cov-pbt-rep-rul-hao | Enabled | s3://cov-raw-bac-hao | US East (N. Virginia) us-east-1 | 0 | Entire bucket | Same as source | Same as source | Disabled | Replicate | Disabled |

Note. This figure shows the replication rules of raw bucket created in S3. Source: AWS S3.

Before this step, I also created the bac-folder for raw bucket in S3. And this is the destination of output.

Figure 53: Transfer-bucket-Versioning and Encryption

The screenshot shows the AWS S3 Management tab for the 'cov-tsf-hao' bucket. Under 'Bucket Versioning', it is enabled. Under 'Default encryption', the 'Encryption type' is set to 'Server-side encryption with AWS Key Management Service keys (SSE-KMS)' using the ARN 'arn:aws:kms:us-east-1:194110502954:key/035bd2eb-a0cc-472a-8d46-7027ee081efc'. Under 'Bucket Key', it is noted that KMS encryption is used to encrypt new objects. Under 'Intelligent-Tiering Archive configurations', there are no configurations listed.

Note. This figure shows the screenshot of the versioning and encryption of transfer bucket created in S3. Source: AWS S3.

Figure 54: Transfer-bucket-Replication Rules

The screenshot shows the AWS S3 console interface for the 'cov-tsf-hao' bucket. It displays two main sections: 'Lifecycle configuration' and 'Replication rules'.

Lifecycle configuration: A section titled 'Lifecycle rules' shows a table with one row. The row details a rule named 'cov-pbt-rep-rul-hao', which is enabled and points to the destination bucket 's3://cov-tsf-bac-hao'. The rule is set to US East (Virginia) us-east-1, has a priority of 0, applies to the entire bucket, and uses the same source and destination.

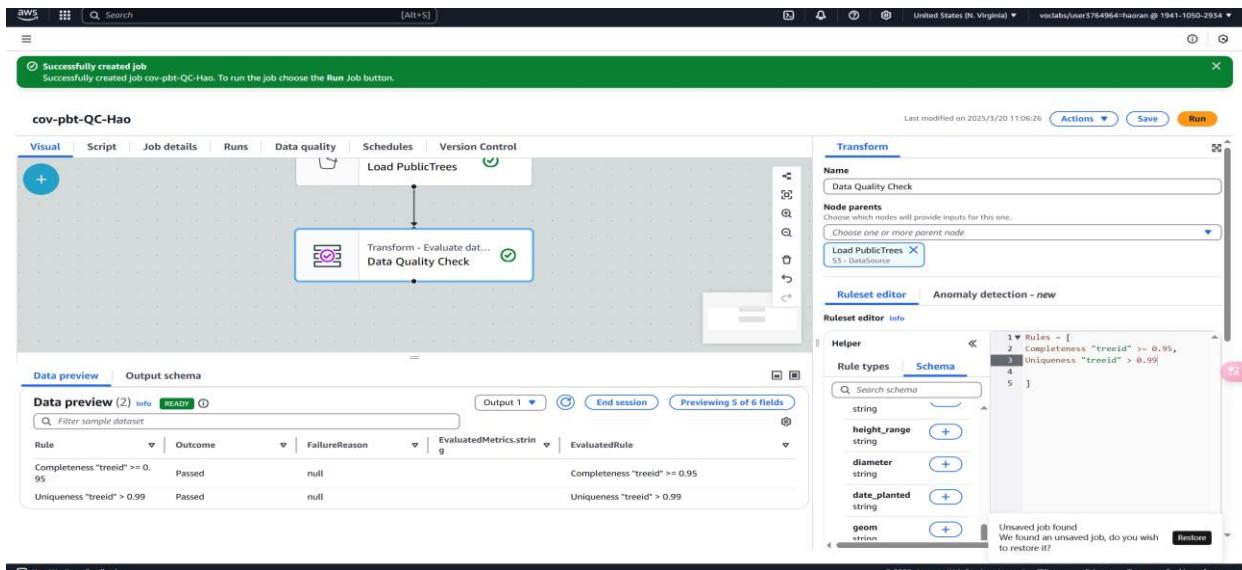
Replication rules (1): A table showing a single replication rule. The rule is named 'cov-pbt-rep-rul-hao', is enabled, and replicates from 's3://cov-tsf-bac-hao' to 'US East (Virginia) us-east-1'. The replication scope is 'Entire bucket', and it uses 'Same as source' for both storage class and replica owner. The replication time control is disabled, and replication is disabled.

Note. This figure shows the replication rules of transfer bucket created in S3. Source: AWS S3.

Before this step, I also created the bac-folder for transfer bucket in S3. And this is the destination of output.

Step 7: Data Governance

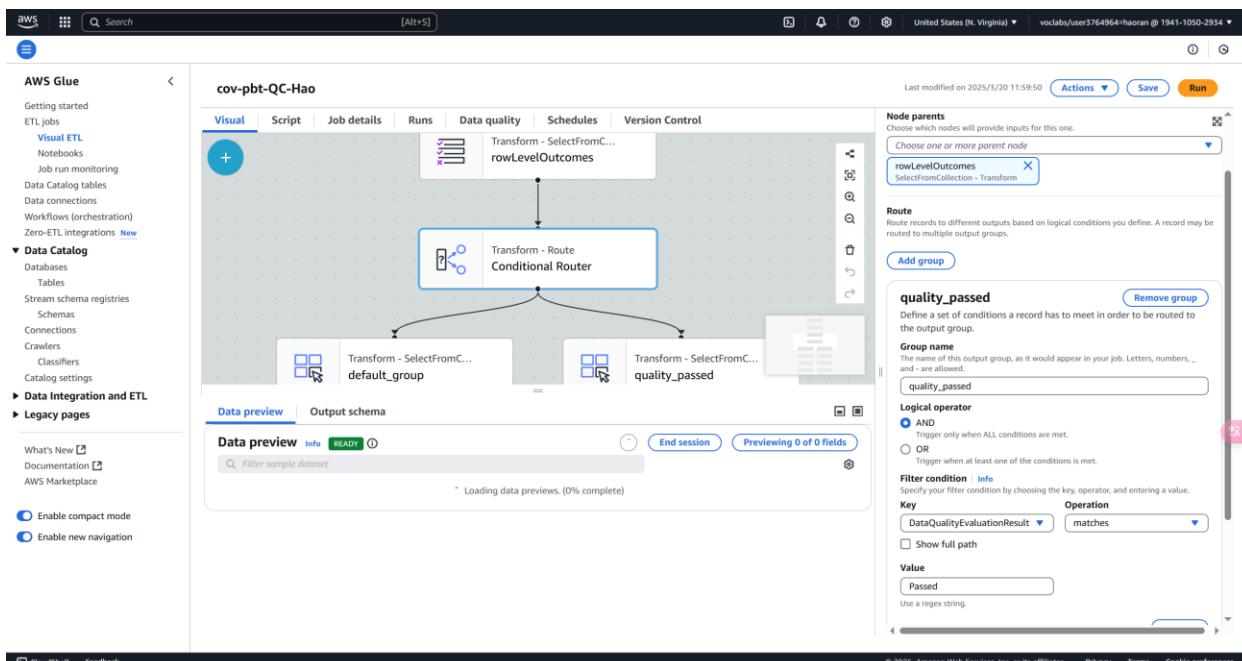
Figure 55: Data Quality Check Rules



Note. This figure shows the screenshot of the data quality check step in visual ETL created in Glue. Source: AWS Glue

In this step, I checked the completeness and uniqueness for “treeid” column. To see if the completeness is equal or greater than 95% and if Uniqueness is greater than 99%.

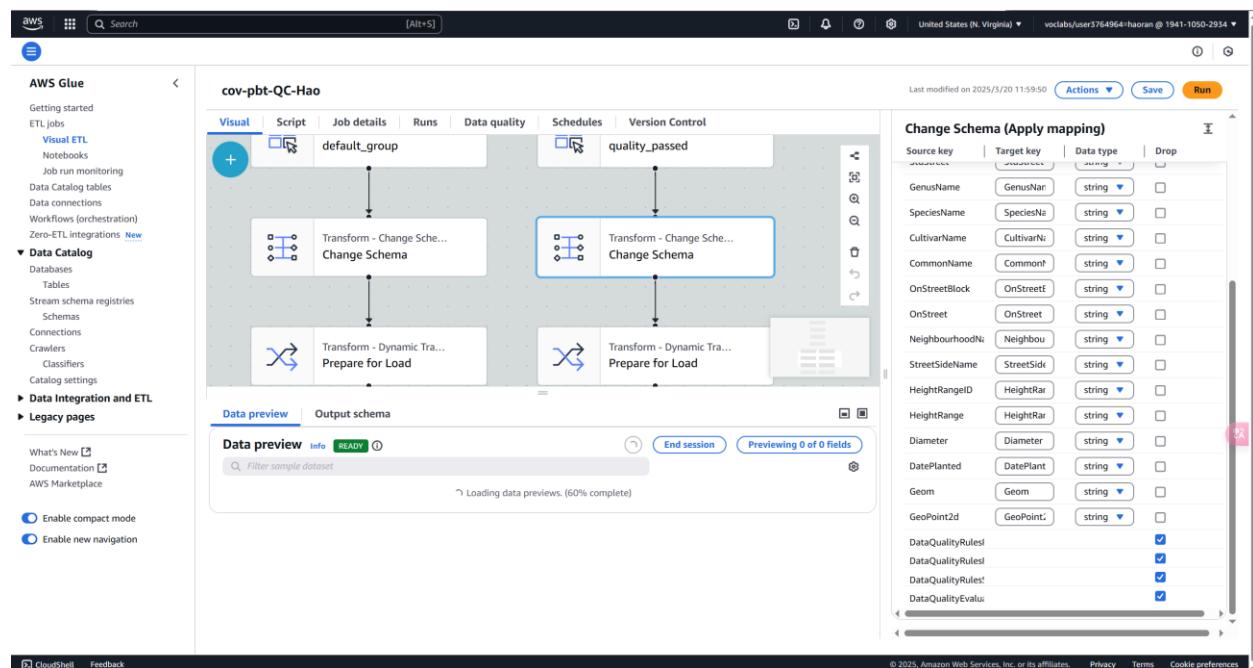
Figure 56: Conditional Router



Note. This figure shows the screenshot of the conditional router step in visual ETL created in Glue. Source: AWS Glue

This step is to separate the quality check output as two files: passed and failed into two different prepared folders.

Figure 57: Change Schema Before Load

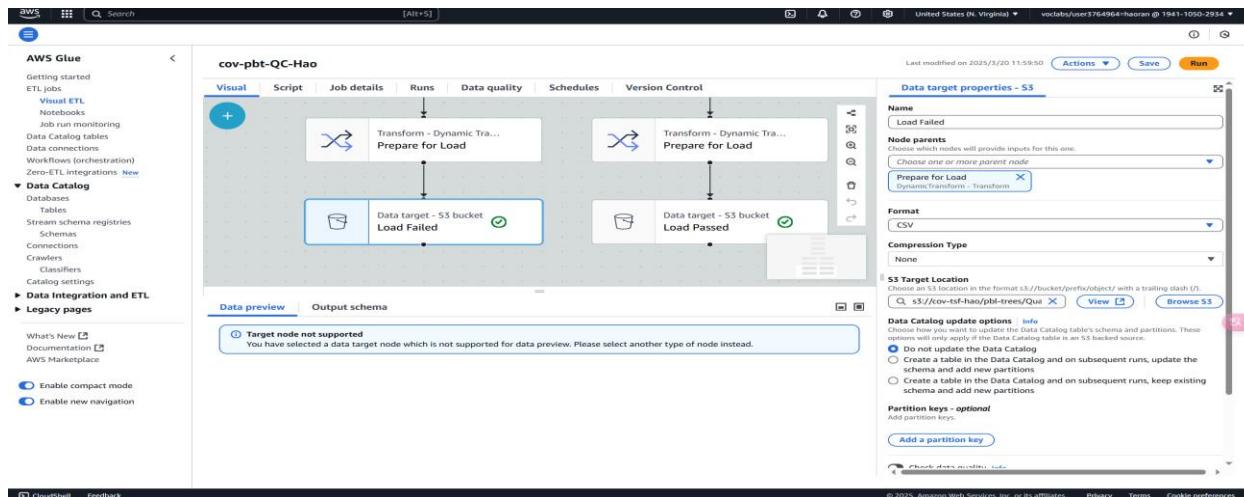


Note. This figure shows the screenshot of the change schema step in visual ETL created in Glue.

Source: AWS Glue

The meaning of this step is to drop the useless columns which are for data quality checking for both passed and failed output files. So it reduces our costs and make the output file looks cleaner.

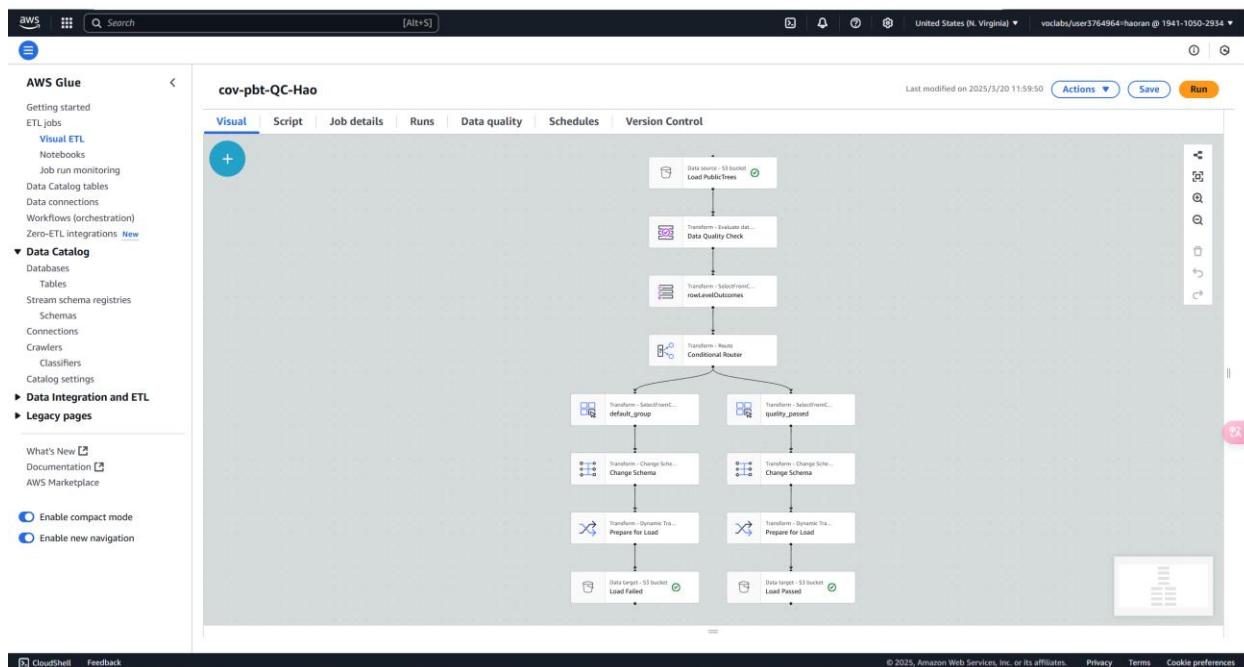
Figure 58: Load Both Passed and Failed into Corresponding Folders



Note. This figure shows the screenshot of the Load Failed/Load Passed step in visual ETL created in Glue. Source: AWS Glue

These two steps are the last step before running the visual ETL. We select the output file format as csv and choose the specified location to output.

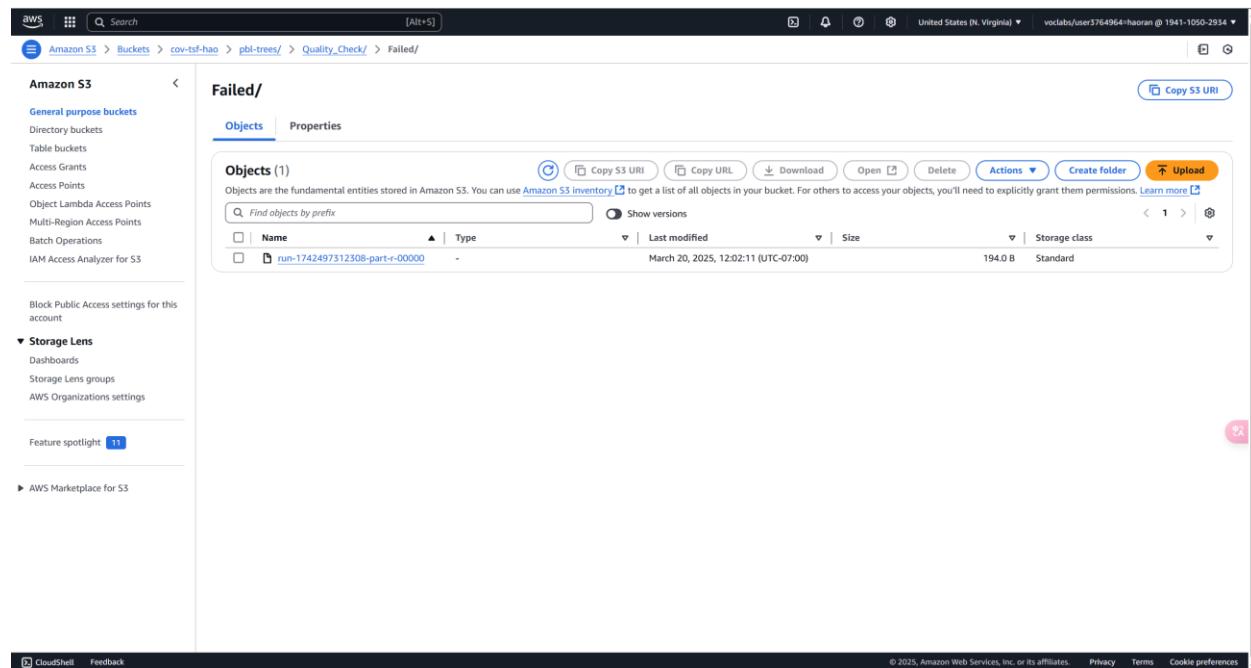
Figure 59: Overall Look of Quality Check Visual ETL



Note. This figure shows the screenshot of the overall look of Quality Check Pipeline in visual ETL created in Glue. Source: AWS Glue

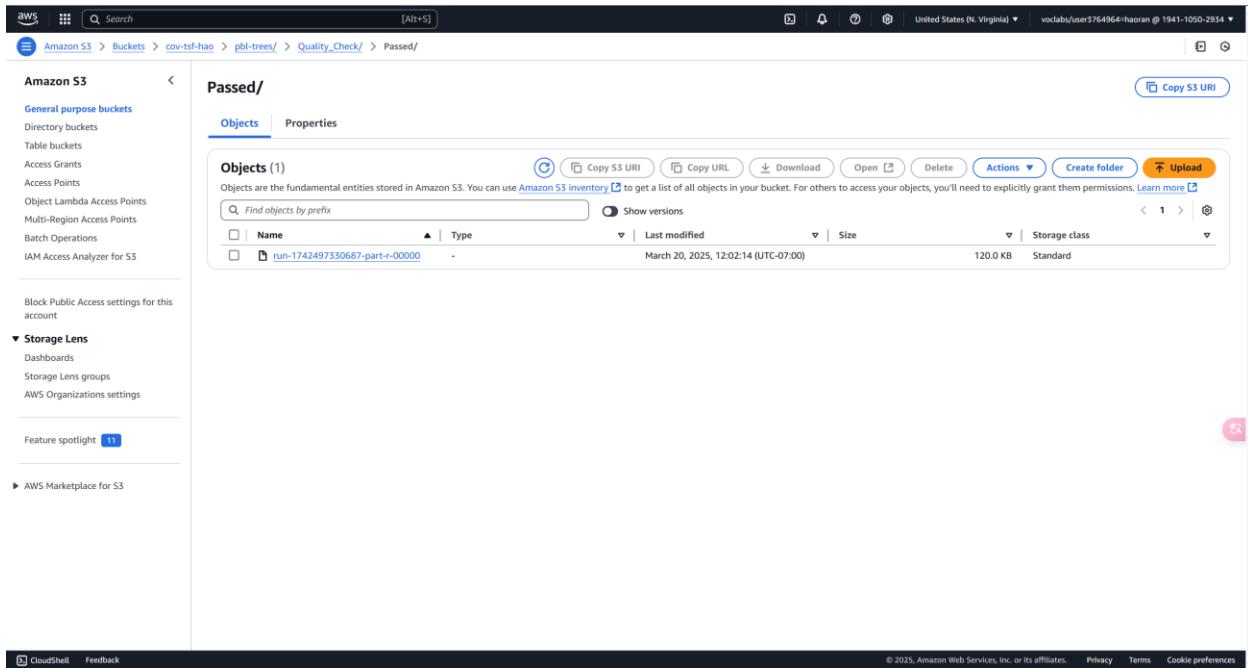
This is the overall look of the Quality Check Visual Pipeline. We can see which part of dataset passed/failed the quality check in the S3 bucket below.

Figure 60: Quality Check Failed Output



Note. This figure shows the screenshot of the Failed output in Transfer Bucket of S3. Source: AWS S3

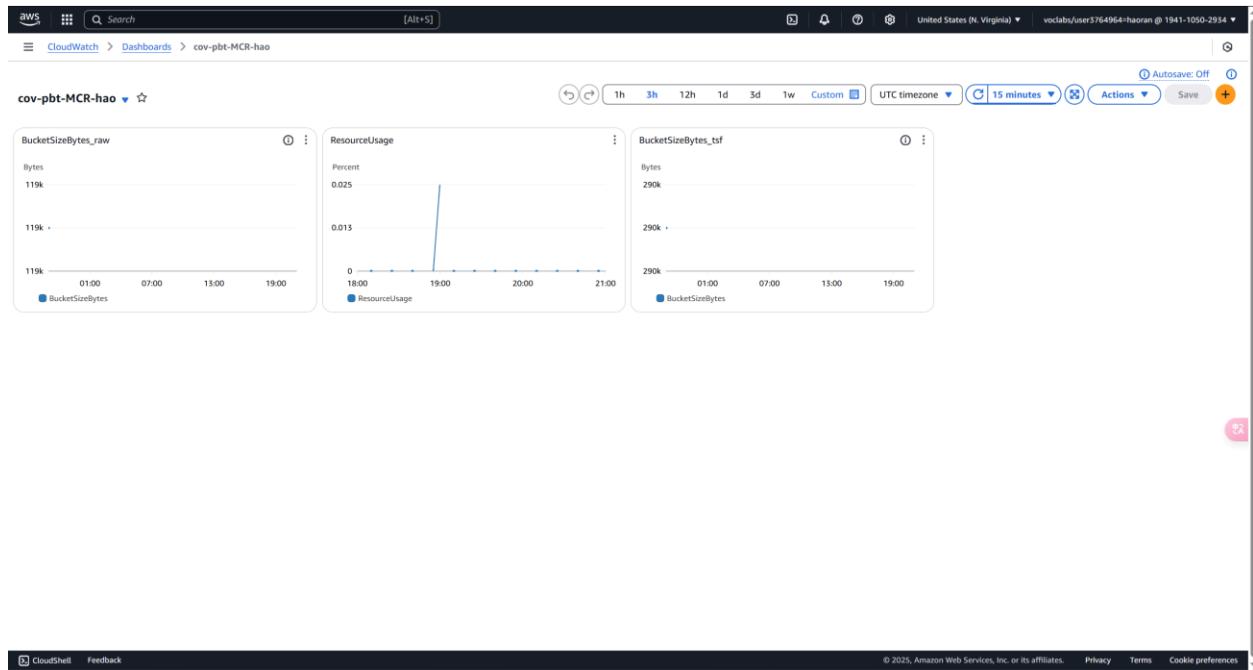
Figure 61: Quality Check Passed Output



Note. This figure shows the screenshot of the Passed output in Transfer Bucket of S3. Source: AWS S3

Step 8: Data Monitoring

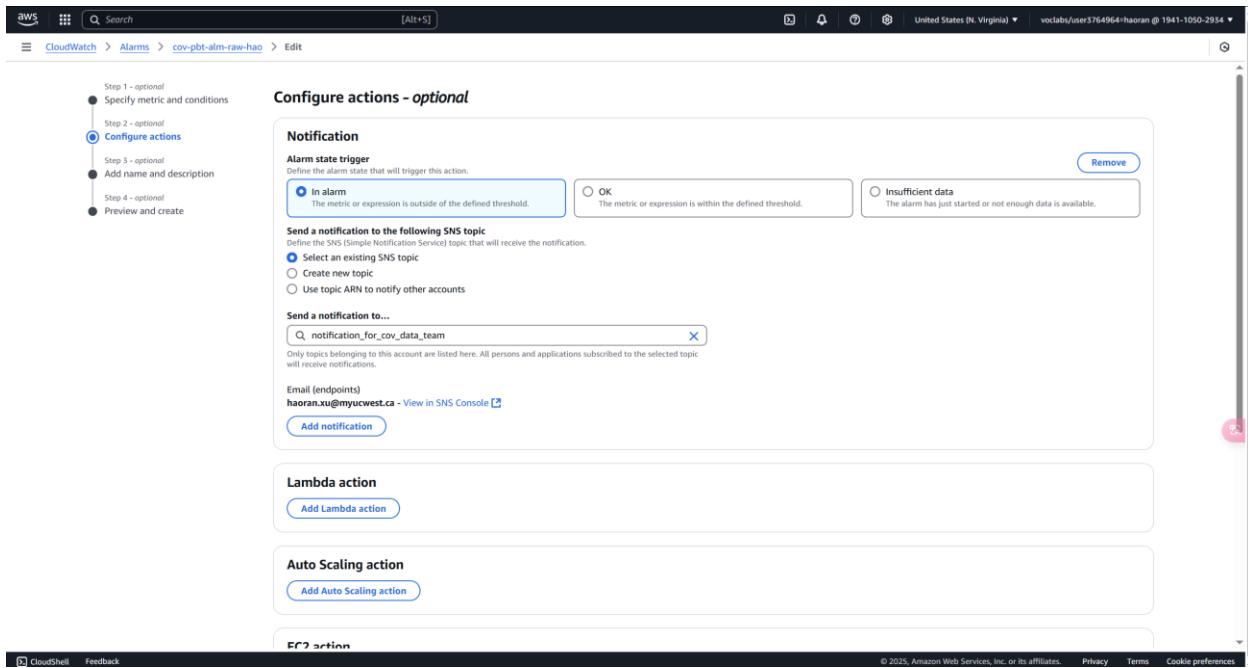
Figure 62: Dashboard of CloudWatch



Note. This figure shows the screenshot of the dashboard created in CloudWatch. Source: AWS CloudWatch

It's for monitoring the usage of raw bucket size, job run resource usage and usage of transfer bucket size.

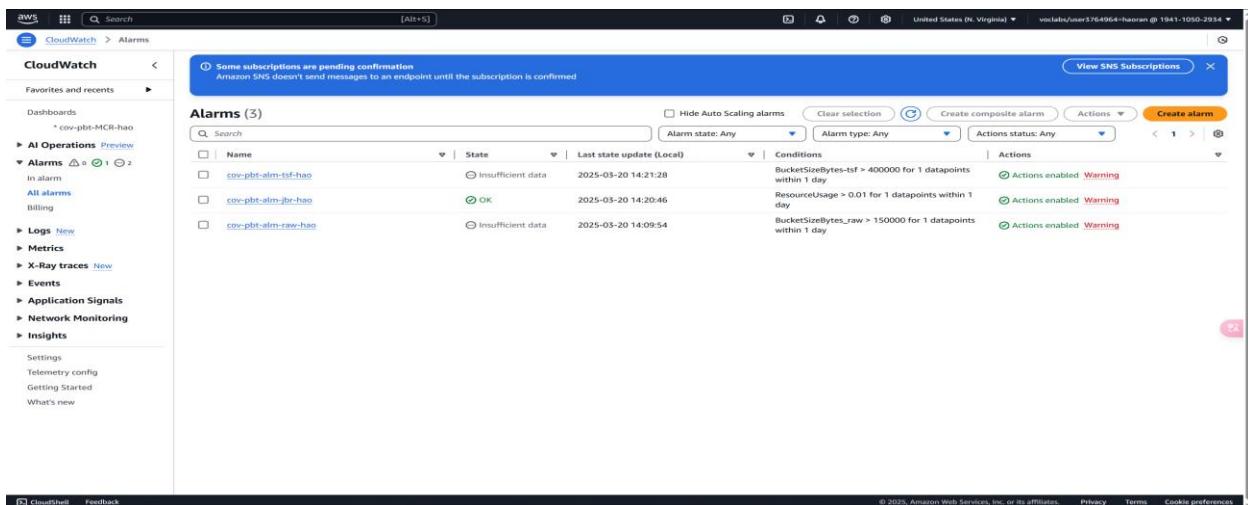
Figure 63: Alarm-Create New Topic for SNS



Note. This figure shows the screenshot of creating new SNS topic in CloudWatch. Source: AWS CloudWatch

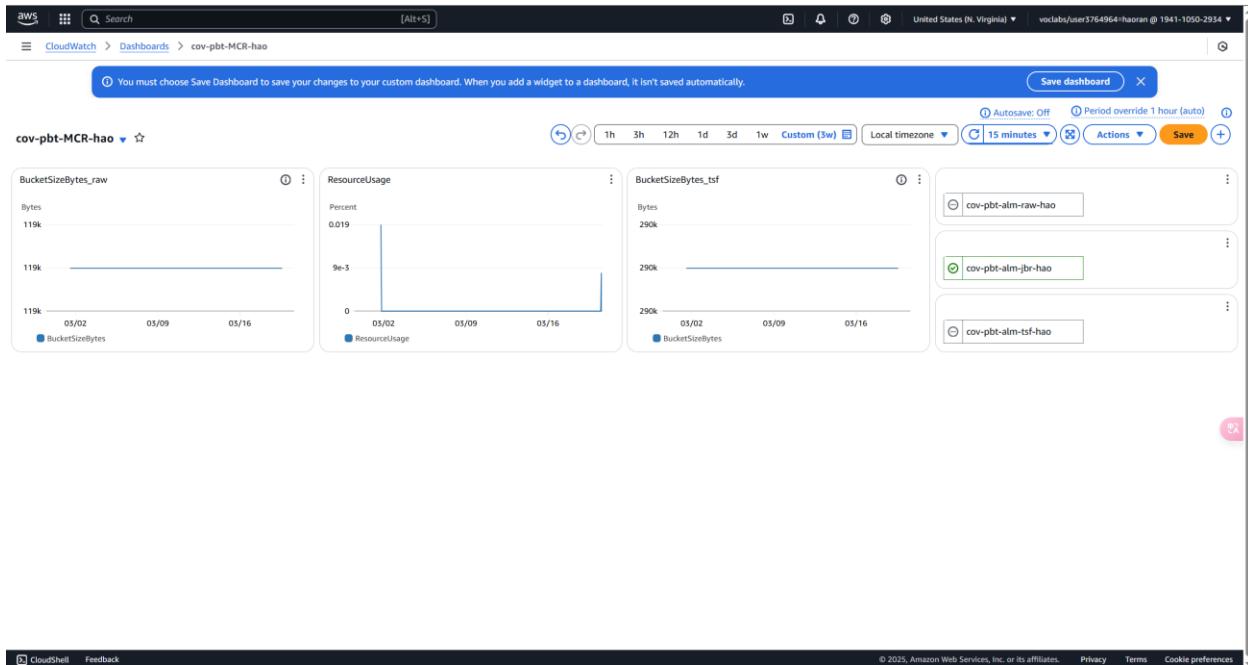
Here, I'm creating Alarm to monitoring the usage of three metrics. If the usage exceed specific amount that I set, I'll get the alarm message from AWS CloudWatch to notify me through email, so I can do some adjustment.

Figure 64: Three Alarms Screenshot



Note. This figure shows the screenshot of all alarms in CloudWatch. Source: AWS CloudWatch

Figure 65: Adding Alarms into the Dashboard



Note. This figure shows the screenshot of all alarms added into the dashboard in CloudWatch.

Source: AWS CloudWatch

DAP Architecture Analysis (Teamwork)

Operational Excellence (*Haoran*)

Operational excellence is definitely a very important part when running DAP. Effective operational excellence management can enable various projects to operate effectively, and the main responsibilities are to create, monitor and maintain the operation of the system. At the beginning of Project 1, we downloaded data from the City of Vancouver data portal to prepare raw data. Here we found and created the raw dataset. After that, we went through profiling and

cleaning steps to ensure that our data was ready to use before transfer. Here, our role in operational excellence is to ensure the integrity and cleanliness of the dataset, paving the way for subsequent use by the data team. We also need to monitor the system in daily use, which can be reflected that all our steps on AWS were recorded, which ensured that modifications were traceable. This helped us ensure system stability and data integrity in our operational excellence. At the same time, Project 2 does not require us to use cloud trail to monitor various actions. However, this is very helpful to ensure operational excellence. If we encounter problems or failures, we can find the problems from the past log history and learn from them to improve the system. We also need to adjust our operating environment and strategies at any time according to customer needs and changes in the business environment. This is also an important part of ensuring effective operational excellence.

Security (*Muhammad*)

In our group project, we all worked with our own datasets and cloud environments, but we followed the same architecture guidelines and AWS best practices. So even though we planned things together, every one of us set up and implemented security individually. This section summarizes how each team member handled security in their own setup.

- **Kyle** worked on the *Vancouver Street Trees* dataset. He created his own KMS key and encrypted his buckets, enabled versioning, used CloudTrail for auditing, and set up alarms for billing thresholds. He also ran transformation jobs in Glue and used Athena for querying.

- **Haoran** used the *Parking Tickets* dataset. His environment included encrypted buckets with KMS, CloudTrail logs, S3 replication for backup, and detailed Glue pipelines with multiple transformations. He also had a separate bucket for storing processed data and applied logging and monitoring.
- **Gyanvi** worked with the *Animal Control Inventory* dataset. She also created her own KMS key, enabled S3 versioning, and configured lifecycle rules, CloudTrail logging, and CloudWatch monitoring. She created her Glue jobs and followed the same structure.
- **Zulqarnain** used the *City of Vancouver Business Licenses* dataset. He created a custom KMS key, encrypted all S3 buckets, enabled versioning, used lifecycle rules, and set up CloudTrail for tracking. He also made CloudWatch alarms for usage and billing and enabled SNS email notifications.

All of us implemented our own identity control using IAM and CloudTrail. We didn't share IAM roles or users each person had their own environment to manage. No EC2 instances were used, so most of our security work focused on S3, Glue, CloudTrail, and CloudWatch.

Implement a Strong Identity Foundation

Did we use it? Yes

How: Everyone used CloudTrail to track activity and manage identity access in their own AWS accounts. IAM permissions were applied based on the least privilege principle. No shared accounts or public access.

Enable Traceability

Did we use it? Yes

How: All of us enabled CloudTrail and used it to track every action in AWS. We also set up CloudWatch alarms. For example, Zulqarnain and Kyle monitored billing and S3 activity with custom alarms.

Apply Security at All Layers**Did we use it? Yes**

How: We applied encryption, versioning, and blocking public access to all S3 buckets. CloudTrail monitored user actions. CloudWatch helped track S3 storage and Glue job activity.

Automate Security Best Practices**Did we use it? Yes**

How: Everyone configured alarms in CloudWatch. Zulqarnain added SNS for email alerts. Although no auto-remediation was used, our alerts helped us respond faster.

Protect Data in Transit and at Rest**Did we use it? Yes**

How: All S3 buckets were encrypted with KMS keys. CloudTrail logs were stored in encrypted buckets too. This covered both transit (data moving across services) and at rest (data in storage).

Keep People Away from Data

Did we use it? Yes

How: CloudTrail logs and datasets were encrypted and not directly accessed. Glue jobs processed the data, and logging was automatic, reducing manual exposure.

Prepare for Security Events

Did we use it? Yes

How: We didn't run a full incident simulation, but CloudTrail logs and CloudWatch alarms gave us a solid setup to catch anything unusual. If something went wrong, we would know right away.

Table 1: Security Summary Table

| Security Pillar | Did we use it? | How it was applied |
|-----------------------------------------------|----------------|--------------------------------------------------|
| Implement a Strong Identity Foundation | Yes | IAM and CloudTrail used by everyone individually |
| Enable Traceability | Yes | CloudTrail logs + CloudWatch alarms |
| Apply Security at All Layers | Yes | S3 encryption, versioning, monitoring |
| Automate Security Best Practices | Yes | Alarms + SNS email alerts |
| Protect Data in Transit and at Rest | Yes | KMS encryption + secure storage |
| Keep People Away from Data | Yes | Logs encrypted, no manual access |
| Prepare for Security Events | Yes | Alerts and logs prepared us for issues |

Note: This is the self-made table about the security summary

Reliability (*Muhammad*)

Just like with security, all of us managed our own reliability setup in our own AWS accounts. We followed the same structure, so the monitoring and fault tolerance ideas were the same just applied individually. We didn't use EC2 or advanced networking, but we did rely on S3, Glue, CloudTrail, and CloudWatch to keep things reliable and monitored.

Automatically Recover from Failure

Did we use it? Partially

How: Everyone set up CloudWatch alarms for job failures, billing, and usage. But we didn't use auto-recovery like restarting jobs or scaling resources automatically.

Test Recovery Procedures

Did we use it? No

How: We didn't simulate any failures or test recovery plans. This is something we could add in future projects.

Scale Horizontally to Increase Availability

Did we use it? No

How: We didn't use horizontal scaling or load balancing since we weren't working with EC2 or dynamic web services.

Stop Guessing Capacity

Did we use it? Partially

How: We used CloudWatch to monitor storage and costs, but we didn't scale any services automatically. Glue jobs had retried, which helped a little.

Manage Change in Automation

Did we use it? Yes

How: Everyone used CloudTrail and CloudWatch to track changes. This reduced the need for manual checking.

Table 2: Reliability Questions Overview

| <i>Reliability Area</i> | <i>Did we use it?</i> | <i>Notes</i> |
|----------------------------------------------------|-----------------------|------------------------------------------------|
| <i>Automatically Recover from Failure</i> | Partially | Alarms set up, but no automatic fixes |
| <i>Test Recovery Procedures</i> | No | No simulations or recovery drills done |
| <i>Scale Horizontally to Increase Availability</i> | No | Not applicable for this project |
| <i>Stop Guessing Capacity</i> | Partially | CloudWatch monitored usage, but no autoscaling |
| <i>Manage Change in Automation</i> | Yes | All changes logged and tracked with alerts |
| <i>Service Quotas & Network Topology</i> | No | Not included in our scope |

| | | |
|------------------------------------------------|-----------|-----------------------------------------------------------------|
| <i>Workload Architecture</i> | Partially | Each setup worked well but lacked redundancy or fault isolation |
| <i>Change Management - Monitor Resources</i> | Yes | Done using CloudWatch |
| <i>Change Management - Adapt to Demand</i> | No | No scaling features implemented |
| <i>Change Management - Implement Change</i> | Yes | Automated monitoring helped manage changes |
| <i>Failure Management - Data Backup</i> | No | No formal backup strategy set up |
| <i>Failure Management - Fault Isolation</i> | No | Everything ran in one region/account per person |
| <i>Failure Management - Withstand Failures</i> | No | No failover or load balancing done |
| <i>Failure Management - Test Reliability</i> | No | Didn't test for reliability under failure conditions |
| <i>Failure Management - Disaster Recovery</i> | No | No disaster recovery plan made |

Note: This is the self-made table about Reliability Questions Overview

Performance Efficiency (*Gyanvi*)

Table 3: Performance Efficiency Questions

| Performance efficiency questions | Implemented (Yes/No) |
|----------------------------------|----------------------|
| | |

| Selection | |
|--------------------------------------------------------------------|-----|
| How do you select the best-performing architecture? | No |
| How do you select your compute solution? | No |
| How do you select your storage solution? | Yes |
| How do you select your database solution? | No |
| How do you configure your networking solution? | No |
| Review | |
| How do you evolve your workload to take advantage of new releases? | No |
| Monitoring | |
| How do you monitor your resources to ensure they are performing? | Yes |
| Trade-offs | |
| How do you use trade-offs to improve performance? | Yes |

Note: This is the self-made table about Performance Efficiency Questions

Performance efficiency questions:

1. Selection

1.1. How do you select the best-performing architecture?

We did not implement this specifically as we used the default architecture setup for performance.

We can improve by evaluating different architecture based on our requirements and selecting the best-performing one.

1.2. How do you select your compute solution?

We did not create an EC2 instance for our project nor did we choose the auto-scaling option to automatically create EC2 instances based on demand. We can improve by selecting the EC2 instance based on our workload.

1.3. How do you select your storage solution?

We used the S3 standard option for storage in our project, however, Gyanvi implemented a lifecycle rule to switch storage class from standard to Intelligent tiering after 30 days and then to Glacier Instant Retrieval after 180 days to optimize cost. We can improve by exploring more options for storage solutions like EBS for enhancing cost and performance efficiency.

1.4. How do you select your database solution?

We did not implement a database solution in our project. We can improve by using database services like Amazon RDS to optimize performance.

1.5. How do you configure your networking solution?

We did not implement any networking solution configuration. We can improve by using VPC or content delivery networks to enhance performance.

2. Review

2.1. How do you evolve your workload to take advantage of new releases?

We did not evolve our workload to take advantage of the new releases. We can improve by implementing more features offered by AWS to improve performance.

3. Monitor

3.1. How do you monitor your resources to ensure they are performing?

We implemented resource monitoring by using the AWS CloudWatch feature. We created three metrics and three alarms in a dashboard to track billing and CPU usage in real time. We can improve by utilizing enhanced CloudWatch features to optimize performance.

4. Trade-offs

4.1. How do you use trade-offs to improve performance?

We implemented trade-offs using compression techniques like storing system files in parquet format with snappy compression type. However, we can improve by enabling caching for read-heavy workloads or compressing log data to improve performance.

Performance efficiency design principles

1. Democratize Advanced Technologies

We democratised advanced technologies like AWS CloudWatch and CloudTrail which handles monitoring and user activity logging. We did not build any custom solutions for infrastructure management.

2. Go Global in Minutes

We did not deploy our system in multiple AWS regions to reduce latency or improve the global user experience. This was not under the scope of our project.

3. Use Serverless Architectures

We have used many serverless architectures like CloudWatch, and CloudTrail for monitoring and logging user activities.

4. Experiment More Often

We did not perform comparative analysis and experimentation of various storage and computing techniques to choose the best-performing configuration.

5. Have Mechanical Sympathy

We selected the appropriate AWS services such as CloudWatch and CloudTrail for monitoring and logging. We did not deeply analyze data access patterns or optimize databases or storage.

Cost Optimization (*Peng*)

We have already adopted a consumption model for both parts of the project. That is because we paid only for the AWS services that we require based on the requirements of the City of Vancouver rather than using elaborate forecasting. For example, the city of Vancouver clearly requires us to do the data ingestion in part 1 of the project and the data monitoring in part 2 of the project. In this case, we paid only for buckets in S3 and Dashboard and Alarms in CloudWatch. Additionally, we didn't measure overall efficiency, but we operated some functions with the awareness of cost reduction. For example, in part 2 of the project, although we did not measure overall efficiency, we considered the cost and set 1 day for monitoring data for setting alarms. Moreover, we stopped spending money on data center operations in both parts of this project. That is because we mainly focus on the City of Vancouver to proceed with its business projects. Next, we analyzed and attributed expenditure in part 1 of the project but not in part 2 of the project. That is because we have already done the cost estimation for 12 months in part 1 of the project, and our team has an opportunity to optimize the resources and reduce the costs. Finally, we did not technically use managed and application-level services to reduce the cost of ownership in both parts of the project, but we considered the cost reduction when operating

services at AWS. Although we did not technically reduce the operational burden of maintaining servers for tasks such as sending emails or managing databases, we considered the cost when operating services at AWS. For example, we considered the cost and set 1 day for monitoring data for setting alarms, which not only reduced the cost but also managed databases.