Main Image Selection System Design

- **Date**: July 22, 2024
- **Author**: Prashant Dixit
- **Version**: 1.0

Table of Contents

This document outlines the design and implementation of a robust image processing pipeline for a hotel meta-search platform. It delves into the various stages of data management, from storage and ingestion to processing and change data capture, while emphasizing scalability, fault tolerance, and metrics calculation.

The document is structured to provide a comprehensive overview, detailed design, key design decisions, testing strategies, and future considerations, ensuring a thorough understanding of the system's architecture and functionalities.

1. Introduction

Purpose:

The purpose of this document is to detail the architecture, design, and implementation of an image processing pipeline aimed at selecting and managing the best main images for hotels listed on a meta-search platform.

This system is designed to handle large volumes of image data, ensuring high-quality images are selected and updated efficiently.

The document serves as a guide for developers, architects, and stakeholders involved in the project, providing insights into the rationale behind design choices and the methodologies used to achieve the desired outcomes.

Scope:

This document covers the following aspects of the Main Image Selection System:

- Overview of the system architecture and its components
- Detailed design and processing logic.
- Implementation of data ingestion, storage, and processing
- Mechanisms for change data capture (CDC)
- Metrics calculation for reporting purposes
- Scalability and fault tolerance considerations
- Testing and validation strategies

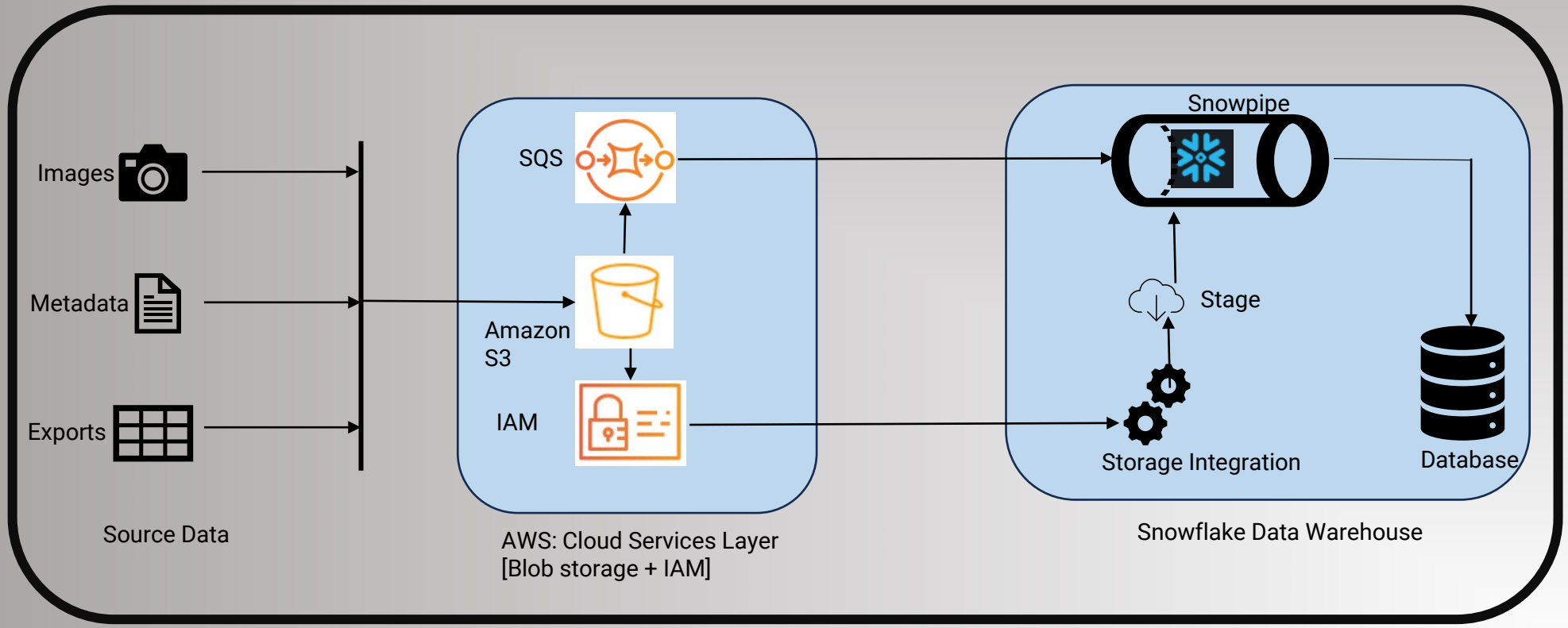
2. System Overview

Architecture Overview:

Our Main Image Selection Pipeline leverages **Snowflake** as the central data warehouse, utilizing **Snowpipe** for real-time data ingestion of image metadata into structured tables.

- Amazon S3 for image metadata storage and image blob storage.
- We use stored procedures to process images based on resolution, aspect ratio, freshness, and tag priority, determining the optimal main image for each hotel.
- Change Data Capture (CDC) ensures any updates or deletions are promptly addressed, maintaining accuracy.
- Snowflake **Tasks** and **Streams** combined bring a powerful mechanism for serverless triggered orchestration.
- The pipeline produces two key outputs: Change Data for immediate updates and a Snapshot of the current main images.

This architecture ensures infinite scalability, robust fault tolerance, and future extensibility.



Components:

AWS Components

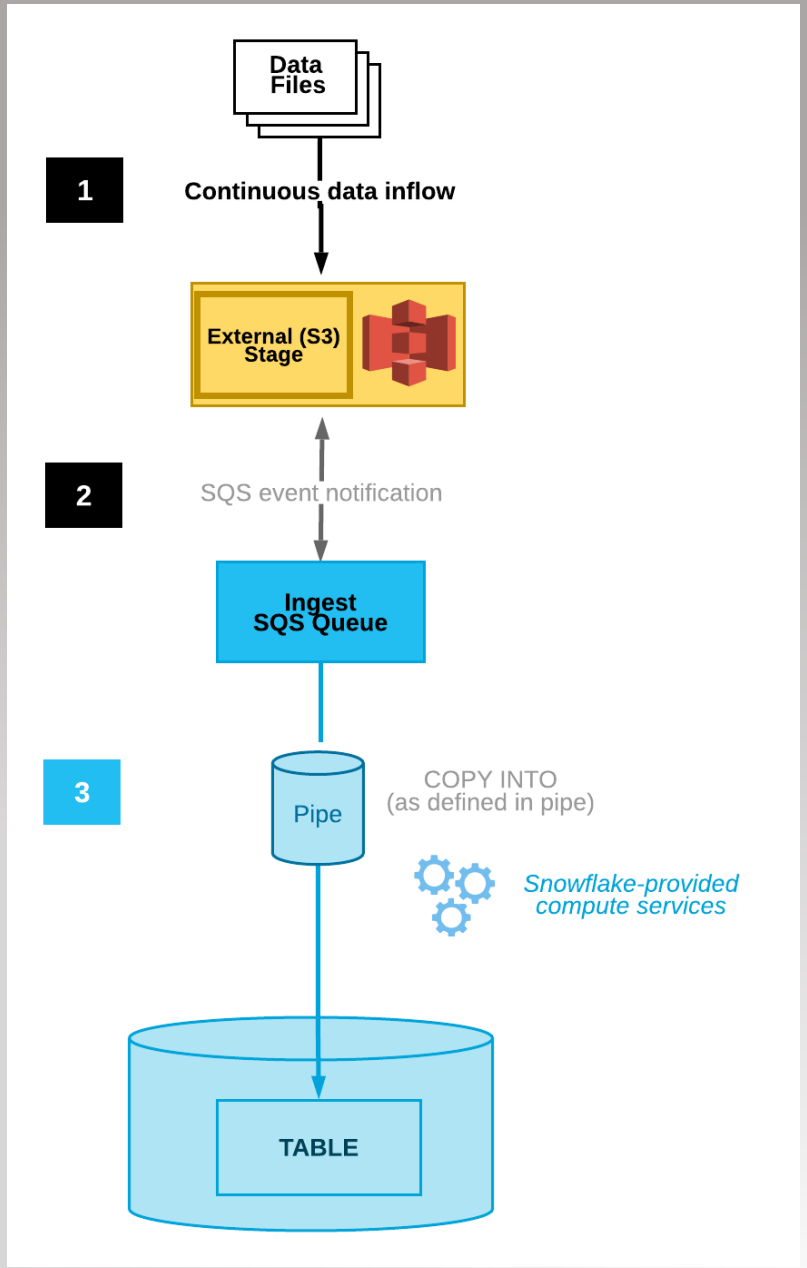
- **Amazon S3**: Stores image metadata and tags as JSONL files
.
- **SQL Queue**: Triggers Snowpipe for real-time data ingestion.
- **AWS Identity and IAM**
  authorize through storage integration

Snowflake Components

Ingestion
- Snowpipe: Loads data from S3 into Snowflake's raw tables.
- Raw Tables: Stores raw image metadata and tags. (Truncate-Load)

CDC and Processing
- Main Tables: Stores processed main image data (history table)
- Views: Provides ranked images based on selection criteria.
- Stored Procedures: Contains the logic for main image selection and updates.

Automation
- Streams: Tracks changes in the raw tables for incremental processing.
- Tasks: Schedules the execution of stored procedures based on changes or time intervals.

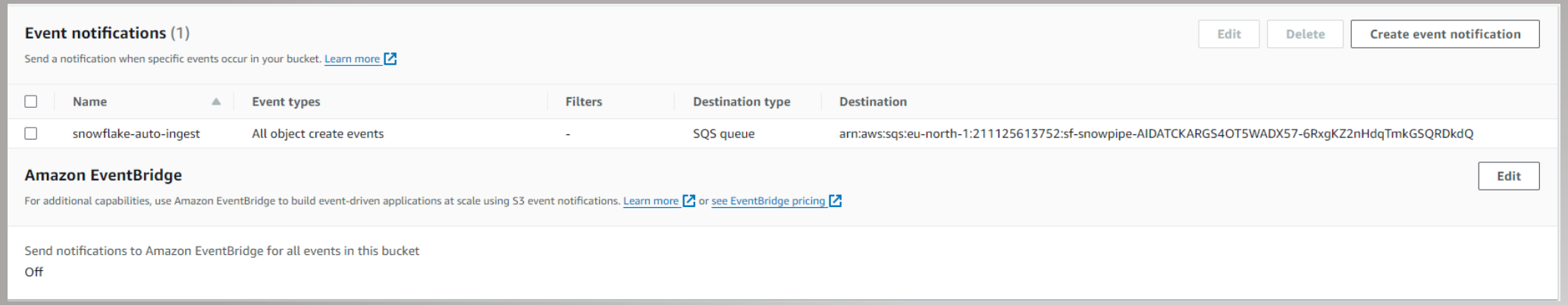


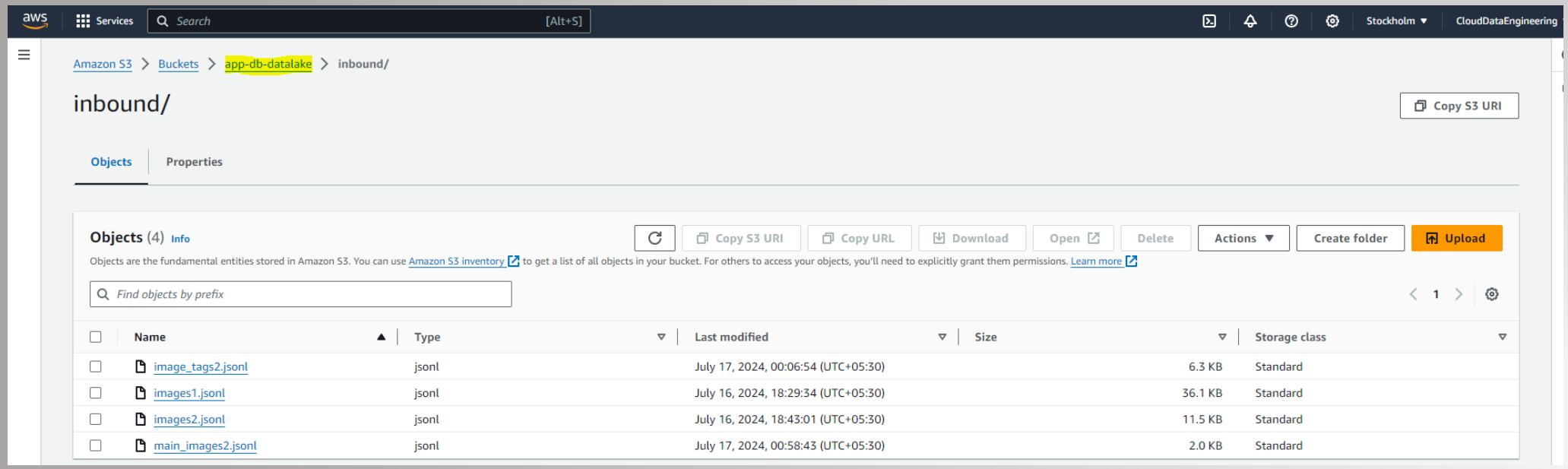
3. Detailed Design

Data Storage on AWS S3:

Amazon S3 is used as the primary storage solution for image metadata and tags, providing a reliable and scalable environment for our pipeline.

- S3 Bucket & SQS Integration: Image metadata & tags are stored in an S3 bucket. Storage events are captured using S3 Event Notifications, which are sent to SQS queue.
- Snowpipe Integration: Snowpipe uses the SQS notifications to trigger data loading into Snowflake. The ARN of the Snowpipe is referenced for automated ingestion.





Data Ingestion using Snowpipe

Snowpipe is leveraged for continuous and automated data ingestion into Snowflake, ensuring timely and accurate data processing.

- Event Message from SQS Queue Snowpipe listens for event messages from the SQS queue, triggered by S3 storage events, to initiate data loading
  .
- Automated Data Loading Snowpipe uses the S3 bucket integration to continuously load new files into Snowflake tables using the `COPY INTO` command defined in the Snowpipe configuration.

The command creates a Snowpipe named IMAGES_PIPE that automatically ingests data from an S3 stage APP_DB_STAGE into the IMAGES_RAW table in Snowflake, using the specified JSON file format and a pattern to match the file names.

AUTO_INGEST = TRUE: Enables automatic ingestion for the Snowpipe.

PATTERN = 'images[0-9]+\.jsonl$': Uses a regex pattern to match the files to be ingested.

APP_DB.APP_SCHEMA Settings Code Versions

```
41
42  create or replace pipe APP_DB.APP_SCHEMA.IMAGES_PIPE
43      auto_ingest=true
44      as
            copy into APP_DB.APP_SCHEMA.IMAGES_RAW
            from @APP_DB_STAGE
47              PATTERN='images[0-9]+\\.jsonl$'
48              FILE_FORMAT = (FORMAT_NAME = 'APP_DB.APP_SCHEMA.json_format');
49
50  desc pipe APP_DB.APP_SCHEMA.IMAGES_PIPE;
51
```

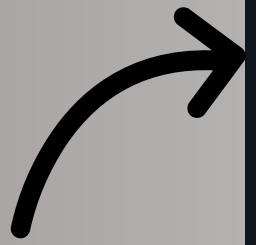
Results Chart

	property	property_type	property_value	property_default
1	ENABLED	Boolean	true	false
2	STORAGE_PROVIDER	String	S3	
3	STORAGE_ALLOWED_LOCATIONS	List	*	[]
4	STORAGE_BLOCKED_LOCATIONS	List		[]
5	STORAGE_AWS_IAM_USER_ARN	String	arn:aws:iam::211125613752:user/externalstages/cifax600(	
6	STORAGE_AWS_ROLE_ARN	String	arn:aws:iam::339712703858:role/snow_role	
7	STORAGE_AWS_EXTERNAL_ID	String	MF36329_SFCRole=2_Cpo3KPmMymlvK3JrTyvakY2DSfU=	
8	COMMENT	String		

Processing Logic in Snowflake views, SP

Business Logic in View : Views are used to filter, transform, and rank the images based on predefined criteria such as resolution, aspect ratio, and freshness. These views help in dynamically updating the main image for each hotel.

- active_images: Filters images that are active.
  Image_with_tags: Joins active images with their tags
- ranked_images: Ranks images for each hotel based on the number of tags and resolution.



ranked_images
main view to provide ranked images from history tables based on images and image_tags history data.

```sql
create or replace view APP_DB.APP_SCHEMA.RANKED_IMAGES(
    HOTEL_ID,
    IMAGE_ID,
    CDN_URL
) as
WITH active_images AS (
    SELECT * FROM images WHERE is_active = TRUE
),
image_with_tags AS (
    SELECT i.*, it.tags
    FROM active_images i
    LEFT JOIN image_tags it ON i.image_id = it.image_id
),
ranked_images AS (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY hotel_id ORDER BY ARRAY_SIZE(tags) DESC, (height * width) DESC) AS rank
    FROM image_with_tags
)
SELECT hotel_id, image_id, cdn_url
FROM ranked_images
WHERE rank = 1;
```

Processing Logic in Snowflake views, SP:

create or replace TABLE APP_DB.APP_SCHEMA.IMAGES_RAW (SRC **VARIANT**);

** Entire JSON record gets loaded as a row using Variant column data type.

Upon receiving an event message from SQS , Snowpipe loads Data using predefined COPY INTO raw tables.

IMAGES_RAW
IMAGE_TAGS_RAW
MAIN_IMAGES_RAW

Truncate-Load Tables

Procedures
- LOAD_AND_TRUNCATE_IMAGES()
- LOAD_AND_TRUNCATE_IMAGE_TAGS()
- LOAD_AND_TRUNCATE_MAIN_IMAGES()

IMAGES
IMAGE_TAGS
MAIN_IMAGES

Stored procedures flatten the Json record and insert data into main tables as per columns' ordinal position

RANKED_IMAGES view ranks the images with highest tags and resolution

UPDATE_MAIN_IMAGES()

SP uses the RANKED_IMAGES view to update main_images table

```
160  SELECT * FROM APP_DB.APP_SCHEMA.IMAGES_RAW;
```

Results | Chart

	SRC
1	{ "cdn_url": "https://imgcy.trivago.com/partner-images/74/5a/3cc634b9b1f5c461c8f9a6b510332c9c84aadb3d0624d8d67874424e2001", "created_at": "2024-03-27", "hash": "fedeff0400808080", "height": 4300, "hotel_id": 7009,
2	{ "cdn_url": "https://imgcy.trivago.com/partner-images/40/89/bc031e667426b894bd6529619cdb6eb2daeec4e9a6fcdbe514760298d167", "created_at": "2024-03-27", "hash": "ff0300c8e0f0008f", "height": 4300, "hotel_id": 7009,
3	{ "cdn_url": "https://imgcy.trivago.com/partner-images/14/29/34af6f5784bfb850b71a85b373593d36f8c8e58203f539a3cbd5ffcacd9e", "created_at": "2023-09-26", "hash": "fffcfc60c1fb8100", "height": 1969, "hotel_id": 7009, "ima
4	{ "cdn_url": "https://imgcy.trivago.com/partner-images/49/56/23067f400b674ef3ee0748d9580eb5ccb36001f61af1235e07a47275eba6", "created_at": "2023-08-30", "hash": "1858feff0000e3fb", "height": 2362, "hotel_id": 7009, "

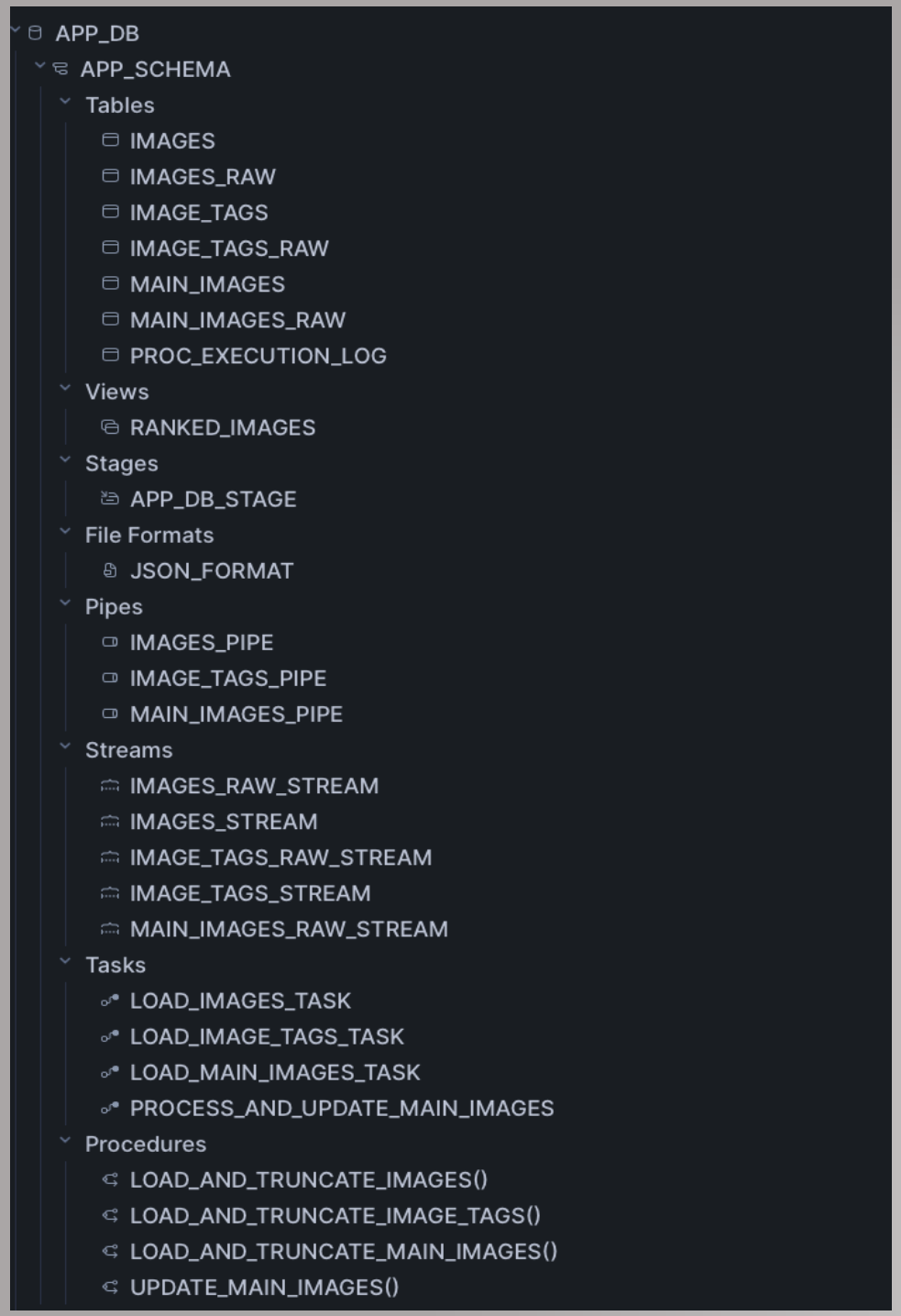
Variant record in _RAW tables

```
160  SELECT * FROM APP_DB.APP_SCHEMA.IMAGES;
```

Results | Chart

	HOTEL_ID	IMAGE_ID	CDN_URL	HEIGHT	WIDTH	IS_ACTIVE	CREATED_AT	HASH
1	7008	25019504	https://imgcy.trivago.com/hotelier-in	3000	4496	TRUE	2023-06-01	ff7f3701c00c1e1f
2	6998	43868437	https://imgcy.trivago.com/hotelier-in	5184	3456	TRUE	2023-06-01	8c03783c03ff7b7f
3	7008	25019504	https://imgcy.trivago.com/hotelier-in	3000	4496	TRUE	2023-06-01	ff7f3701c00c1e1f
4	6998	43868437	https://imgcy.trivago.com/hotelier-in	5184	3456	TRUE	2023-06-01	8c03783c03ff7b7f
5	7008	25019504	https://imgcy.trivago.com/hotelier-in	3000	4496	TRUE	2023-06-01	ff7f3701c00c1e1f

Flattened and ordered data



Process Flow :

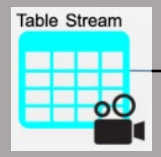
1. Data and metadata are stored in S3 buckets, with S3 events triggering an SQS message to start ingestion.
2. SQS messages trigger Snowpipe to load new data into raw tables using the `COPY INTO` command.
3. Streams detect changes in raw tables, initiating triggered tasks that call stored procedures to process the data.
4. Data is validated and inserted into structured tables, ensuring correct schema alignment.
5. Views dynamically rank images based on tags, resolution, and aspect ratio.
6. Scheduled tasks regularly run procedures to update the main image based on the latest rankings.

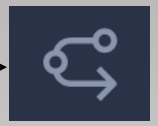
Change Data Capture (CDC)

```
create or replace stream APP_DB.APP_SCHEMA.IMAGES_RAW_STREAM on table IMAGES_RAW;

create or replace stream APP_DB.APP_SCHEMA.IMAGE_TAGS_RAW_STREAM on table IMAGE_TAGS_RAW;

create or replace stream APP_DB.APP_SCHEMA.MAIN_IMAGES_RAW_STREAM on table MAIN_IMAGES_RAW;
```

A stream object records data manipulation language (DML) changes made to tables, including inserts, updates, and deletes, as well as metadata about each change, so that actions can be taken using the changed data.

- Streams provides a change tracking mechanism for our tables and views, enabling and ensuring "exactly once" semantics for new or changed data.





IMAGES_RAW_STREAM

detect changes in images_raw table based on Snowpipe data loads.

LOAD_IMAGES_TASK

task automatically runs the SP- `LOAD_AND_TRUNCATE_IMAGES()` stored procedure whenever there is new or changed data detected in the `IMAGES_RAW_STREAM`.

- The `system$stream_has_data` function checks for new data in the stream
- if data is present, the task calls the stored procedure to process and move this data from the raw staging table to the main table.

Stored Procedure:
#load_and_truncate_images() performs data insertion and cleanup)

Task definition

```
create or replace task APP_DB.APP_SCHEMA.LOAD_IMAGES_TASK
    warehouse=COMPUTE_WH
    when system$stream_has_data('APP_DB.APP_SCHEMA.IMAGES_RAW_STREAM')
    as CALL APP_DB.APP_SCHEMA.LOAD_AND_TRUNCATE_IMAGES();
```

APP_DB / APP_SCHEMA / LOAD_AND_TRUNCATE_IMAGES()

Stored Procedure | 3 days ago | user-defined procedure

Procedure definition

```
CREATE OR REPLACE PROCEDURE APP_DB.APP_SCHEMA.LOAD_AND_TRUNCATE_IMAGES()
RETURNS VARCHAR(16777216)
LANGUAGE SQL
EXECUTE AS OWNER
AS '
BEGIN
    -- Insert data into IMAGES table from IMAGES_RAW
    INSERT INTO APP_DB.APP_SCHEMA.IMAGES
    SELECT
        TRY_CAST(SRC:hotel_id::STRING AS NUMBER) AS HOTEL_ID,
        TRY_CAST(SRC:image_id::STRING AS NUMBER) AS IMAGE_ID,
        SRC:cdn_url::STRING AS CDN_URL,
        TRY_CAST(SRC:height::STRING AS NUMBER) AS HEIGHT,
        TRY_CAST(SRC:width::STRING AS NUMBER) AS WIDTH,
        TRY_CAST(SRC:is_active::STRING AS BOOLEAN) AS IS_ACTIVE,
        TRY_CAST(SRC:created_at::STRING AS DATE) AS CREATED_AT,
        SRC:hash::STRING AS HASH
    FROM APP_DB.APP_SCHEMA.IMAGES_RAW
    WHERE
        TRY_CAST(SRC:hotel_id::STRING AS NUMBER) IS NOT NULL
        AND TRY_CAST(SRC:image_id::STRING AS NUMBER) IS NOT NULL
        AND SRC:cdn_url::STRING IS NOT NULL
        AND TRY_CAST(SRC:height::STRING AS NUMBER) IS NOT NULL
        AND TRY_CAST(SRC:width::STRING AS NUMBER) IS NOT NULL
        AND TRY_CAST(SRC:is_active::STRING AS BOOLEAN) IS NOT NULL
        AND TRY_CAST(SRC:created_at::STRING AS DATE) IS NOT NULL
        AND SRC:hash::STRING IS NOT NULL;

    -- Truncate the IMAGES_RAW table
    TRUNCATE TABLE APP_DB.APP_SCHEMA.IMAGES_RAW;

    RETURN ''Data loaded into IMAGES and IMAGES_RAW truncated.'';
END;
';
```

This stored procedure performs the following steps:

- **Data Insertion**: Inserts valid and necessary data from the `IMAGES_RAW` staging table into the `IMAGES` main table. It ensures data integrity by using `TRY_CAST` to convert and validate the data types.
- **Data Cleanup**: After successfully inserting the data, it truncates the `IMAGES_RAW` table to clear out processed data, making it ready for new incoming data.
- **Return Statement**: Returns a message indicating successful data loading and truncation.

This process ensures that new and updated image data is consistently and accurately moved from the raw staging area to the main table, ready for further processing and analysis.

4. Design Decisions

- We chose **Amazon S3** for its **durability**, **flexibility**, and **seamless integration with Snowflake**. S3 provides reliable, secure storage for our image data, ensuring data integrity and accessibility. The integration with Snowflake is facilitated by Snowflake's external stage feature, which allows us to efficiently query, and process data stored in S3.

- Fault tolerance is achieved through the inherent features of both S3 and Snowflake. S3 offers high availability and durability, while **Snowpipe** provides **automatic retries** during data ingestion, ensuring continuous data flow even in case of temporary failures. Snowflake's **automatic failover** and **recovery** mechanisms maintain system availability

- Our pipeline is designed for scalability to handle varying loads efficiently.
  Amazon S3: Provides virtually unlimited storage capacity, allowing us to scale effortlessly as data volumes increase.
  Snowflake Warehouses: Configured with different **T-shirt sizes** (e.g., X-Small to 6X-Large) to match processing needs, with auto-scaling capabilities to handle peak loads and automatic suspension during idle times.

- Snowpipe: Supports near **real-time ingestion** through micro-batching, enabling prompt processing of incoming data.

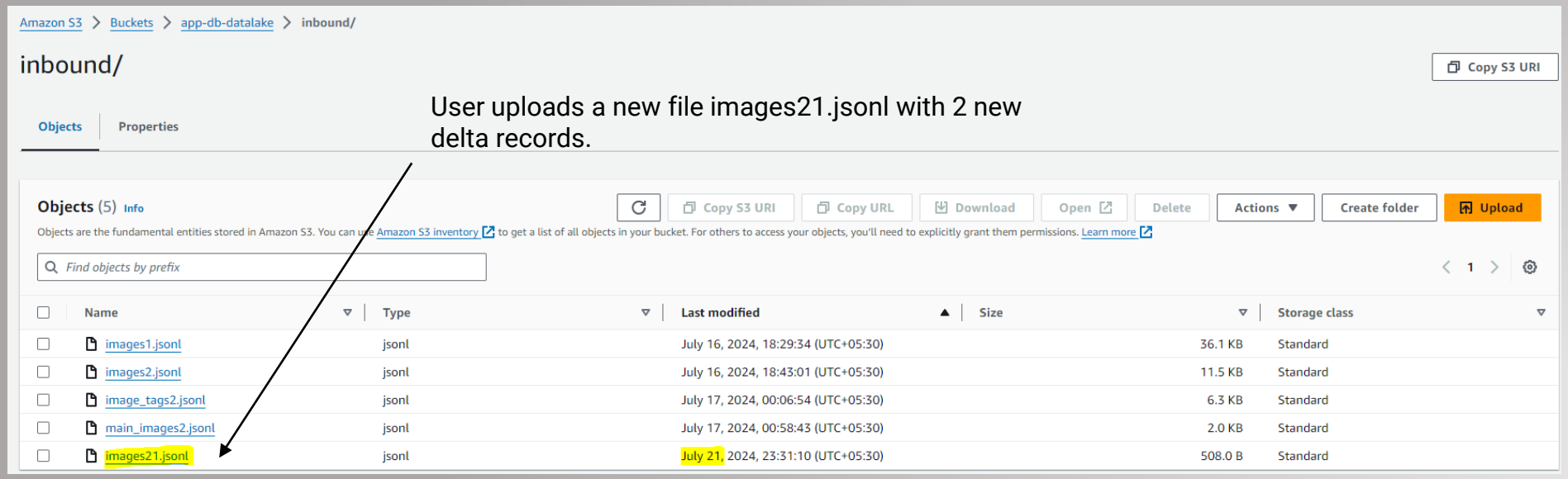
5. Testing, Validations & Monitoring

To ensure the correctness and robustness of the pipeline, it's important to define and implement a comprehensive testing and validation strategy.

Test Case 1: Data Ingestion Validation

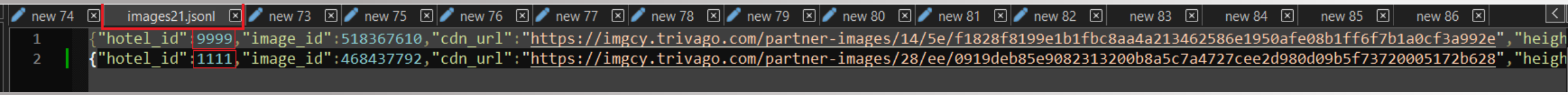
Objective: Verify that new images uploaded to Amazon S3 are correctly ingested into Snowflake tables via Snowpipe.

Step1: Upload delta records to S3 bucket on the Snowflake Stage path.
Step2: Verify if delta records have been loaded by Snowpipe (check raw table, check history table, monitor copy history, check task history)
Step3: Verify main image stored procedure execution to see if data is available in main image table.



Step1 :
User uploads a new file images21.jsonl with 2 new records for new hotels added with hotel_ids 9999 and 1111.

Test records added to images21.jsonl



Step2 :
Verify if delta records have been loaded by Snowpipe (check raw table, check main history table, monitor copy history, check task history)

By executing: SYSTEM$PIPE_STATUS(<pipe>) immediately after user uploads a file, we can verify that Snowpipe has detected a new file images21.jsonl in the queue.

```
//SELECT * FROM SYSTEM$PIPE_STATUS('APP_DB.APP_SCHEMA.IMAGES_PIPE');

{"executionState":"RUNNING",
"pendingFileCount":0,
"lastIngestedTimestamp":"2024-07-21T18:01:28.39Z",
"lastIngestedFilePath":"images21.jsonl",
"notificationChannelName":"arn:aws:sqs:eu-north-1:211125613752:sf-snowpipe-AIDATCKARGS4OT5WADX57-6RxgKZ2nHdqTmkGSQRDkdQ","numOutstandingMessagesOnChannel":1,
"lastReceivedMessageTimestamp":"2024-07-21T18:01:28.245Z",
"lastForwardedMessageTimestamp":"2024-07-21T18:01:30.501Z",
"lastPulledFromChannelTimestamp":"2024-07-21T18:08:08.087Z",
"lastForwardedFilePath":"app-db-datalake/inbound/images21.jsonl"}
```

Querying images table below, please note the hotel_ids we added were 1111, 9999. hotel_ids are ordered to show both the new entries.

```
157  SELECT * FROM APP_DB.APP_SCHEMA.IMAGES;
```

Results | Chart

	HOTEL_ID	IMAGE_ID	CDN_URL	HEIGHT	WIDTH	IS_ACTIVE	CREATED_AT	HASH
1	9999	518367610	https://imgcy.trivago.com	1365	2048	TRUE	2023-07-19	fffe0458dc3c0000
2	7009	634688267	https://imgcy.trivago.com	1513	2953	FALSE	2023-09-26	3cfffffed6800000
3	7009	1904347574	https://imgcy.trivago.com	1365	2048	TRUE	2024-06-28	0076f77f7e10181a

```
157  SELECT * FROM APP_DB.APP_SCHEMA.IMAGES;
```

Results | Chart

	HOTEL_ID	IMAGE_ID	CDN_URL	HEIGHT	WIDTH	IS_ACTIVE	CREATED_AT	HASH
1	1111	468437792	https://imgcy.trivago.com	533	800	TRUE	2023-07-06	1c3cfe80c3def0f0
2	6998	43868437	https://imgcy.trivago.com	5184	3456	TRUE	2023-06-01	8c03783c03ff7b7f
3	6998	43868437	https://imgcy.trivago.com	5184	3456	TRUE	2023-06-01	8c03783c03ff7b7f

Step3: Verify if newly added records are loaded into main_images table

Querying main_images table below, please note that the hotel_ids that we added (1111, 9999) are now available in main_images table.

```
156  SELECT * FROM APP_DB.APP_SCHEMA.MAIN_IMAGES;
157
```

Results | Chart

	HOTEL_ID	IMAGE_ID	CDN_URL
1	1111	468437792	https://imgcy.trivago.com/partner-images/28/ee/0919deb85e9082313200b8
2	9999	518367610	https://imgcy.trivago.com/partner-images/14/5e/f1828f8199e1b1fbc8aa4a21
3	7003	150861912	https://imgcy.trivago.com/hotelier-images/06/a2/03a8769a6395f78cab4635
4	7012	150827705	https://imgcy.trivago.com/hotelier-images/08/e6/28d51ead80422bedda17d7

By querying METRICS_TABLE, we can verify that 2 new hotels with images are added as part of last execution, as the number of main images has increased to 14 from 12.

Please note that the stored procedure UPDATE_MAIN_IMAGES() is responsible to update the metrics in this table. The SP references the ranked_images view and updates the main_images table. While doing this, it also calculates these metrices and inserts into below table for audit purposes.

```
174  SELECT * FROM APP_DB.APP_SCHEMA.METRICS_TABLE;
```

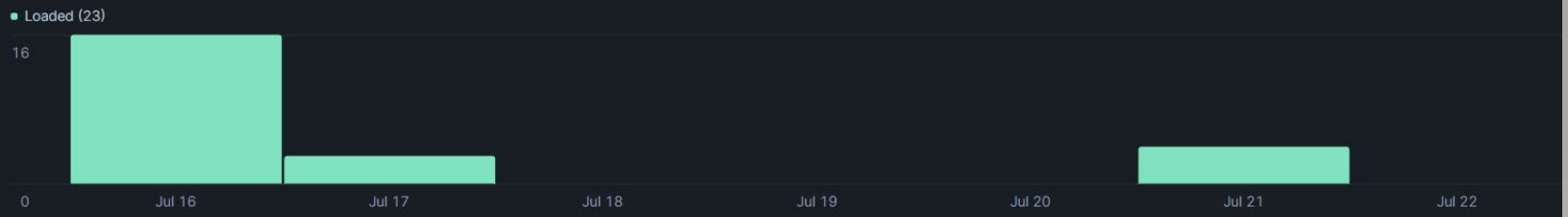
Results | Chart

	ID	NUM_IMAGES_PROCESSED	NUM_HOTELS_WITH_IMAGES	NUM_MAIN_IMAGES	NUM_NEW_MAIN_IMAGES	NUM_UPDATED_MAIN_IMAGES	NUM_DELETED_MAIN_IMAGES	PROCESSED_AT
1	1	408	5	12	12	12	null	2024-07-21 08:57:15.0
2	2	408	5	12	12	12	null	2024-07-21 08:57:40.8
3	3	546	7	14	14	14	null	2024-07-21 11:14:42.5

Copy History

Last 7 days | Status All | Database All | Pipe All

All Copies — Data up to 45 minute late



23 Copies (Jul 16, 2024, 12 AM - Jul 23, 2024, 12 AM)

Search

FILE NAME	LOADED ↓	STATUS	SIZE	ROWS	PIPE	TARGET TABLE	DATABASE	SCHEMA
images21.jsonl	1h 12m ago	Loaded	508.0B	2	IMAGES_PIPE	IMAGES_RAW	APP_DB	APP_SCHEMA
main_images2.jsonl	4d 23h ago	Loaded	2.0KB	11	MAIN_IMAGES_...	MAIN_IMAGES_...	APP_DB	APP_SCHEMA
main_images1.jsonl	4d 23h ago	Loaded	2.0KB	11	MAIN_IMAGES_...	MAIN_IMAGES_...	APP_DB	APP_SCHEMA
image_tags2.jsonl	5d 0h ago	Loaded	6.3KB	65	IMAGE_TAGS_PI...	IMAGE_TAGS_R...	APP_DB	APP_SCHEMA

We can monitor data loading activity for all tables in our account, or for a specific table, by using Snowsight (image on left from snowsight) or SQL (as below):

*select * from table(information_schema.copy_history(TABLE_NAME=>'APP_DB.APP_SCHEMA.IMAGES_RAW', START_TIME=> DATEADD(hours, -1, CURRENT_TIMESTAMP())));*

Task History

Task Graph Runs | Task Runs

Last 7 days | Task Graph Status All | Database All | Root Task All

All Task Graph Runs PREVIEW — Data up to 45 minute late

250+ of 899 Task Graph Runs (Jul 16, 2024, 12 AM - Jul 23, 2024, 12 AM)

Search

ROOT TASK NAME	GRAPH STATUS	DURATION	SCHEDULED TIME ↓	DATABASE	SCHEMA
LOAD_IMAGES_TASK	Succeeded	682ms	Jul 21, 2024, 11:41:43 PM	APP_DB	APP_SCHEMA

LOAD_IMAGES_TASK completed successfully, stored procedure was triggered to load the delta data to history table.

About monitoring capability and flexibility with Snowsight:
Using Snowsight, we can review the execution history for tasks in different ways:

•View the execution history of all tasks run in your account, for example to identify critical tasks that failed to run, long-running tasks, or tasks increasing costs.

•View the execution history for a specific task or a task graph, to gather more information about the task.

6. Summary and Future Work

Summary of the Pipeline:
To achieve the task requirements, we designed and implemented a comprehensive pipeline that efficiently selects the best main image for each hotel on a meta-search platform. The pipeline encompasses the following key steps:

1. We utilized Amazon S3 for storing raw image data and associated tags. This choice provides high durability, availability, and seamless integration with Snowflake.
2. Snowpipe was configured to automate the ingestion of data from S3 into Snowflake.
3. We created views and stored procedures to process and rank images. The logic included filtering active images, joining image tags, and ranking images based on tags and resolution. This ensures that the highest quality and most relevant image is selected for each hotel.
4. Implementing CDC allowed us to update the main images for hotels dynamically, ensuring that the best image is always set as the main image.
5. We developed procedures to calculate and store key metrics, such as the number of images processed, the number of hotels with images, & the count of main images.
6. We scheduled tasks to run the stored procedures automatically. This ensures the continuous and autonomous operation of the pipeline without manual intervention.

Testing the Pipeline:
To validate the pipeline's functionality and reliability, we conducted thorough testing:

1. We introduced incremental (delta) data to the pipeline to verify that new and updated data are processed correctly. This involved ensuring that the CDC logic accurately updates main images based on the latest data.
2. We compared the pipeline's output against expected results, verifying that the image ranking and selection logic are accurate. This step ensured that the highest-ranked image for each hotel is consistently set as the main image and that all metrics are calculated correctly.

Future Work:
Several enhancements and extensions can be considered to further improve and automate the pipeline:

1. Integrating an external orchestrator like Apache Airflow can provide more advanced scheduling, monitoring, and management capabilities for the pipeline.
2. Creating dedicated load scripts can improve job auditing and tracking to ensure better management of data ingestion and processing tasks.
3. Automating the generation of metrics reports and sending them via email can enhance visibility and monitoring of the pipeline's performance.
4. Implementing a process to archive logs and historical images to a container or bucket will help manage storage efficiently and ensure data retention for future reference.
5. Further refining IAM roles, policies, and permissions can enhance the security of data and processing tasks, ensuring compliance with organizational policies.

Thank You