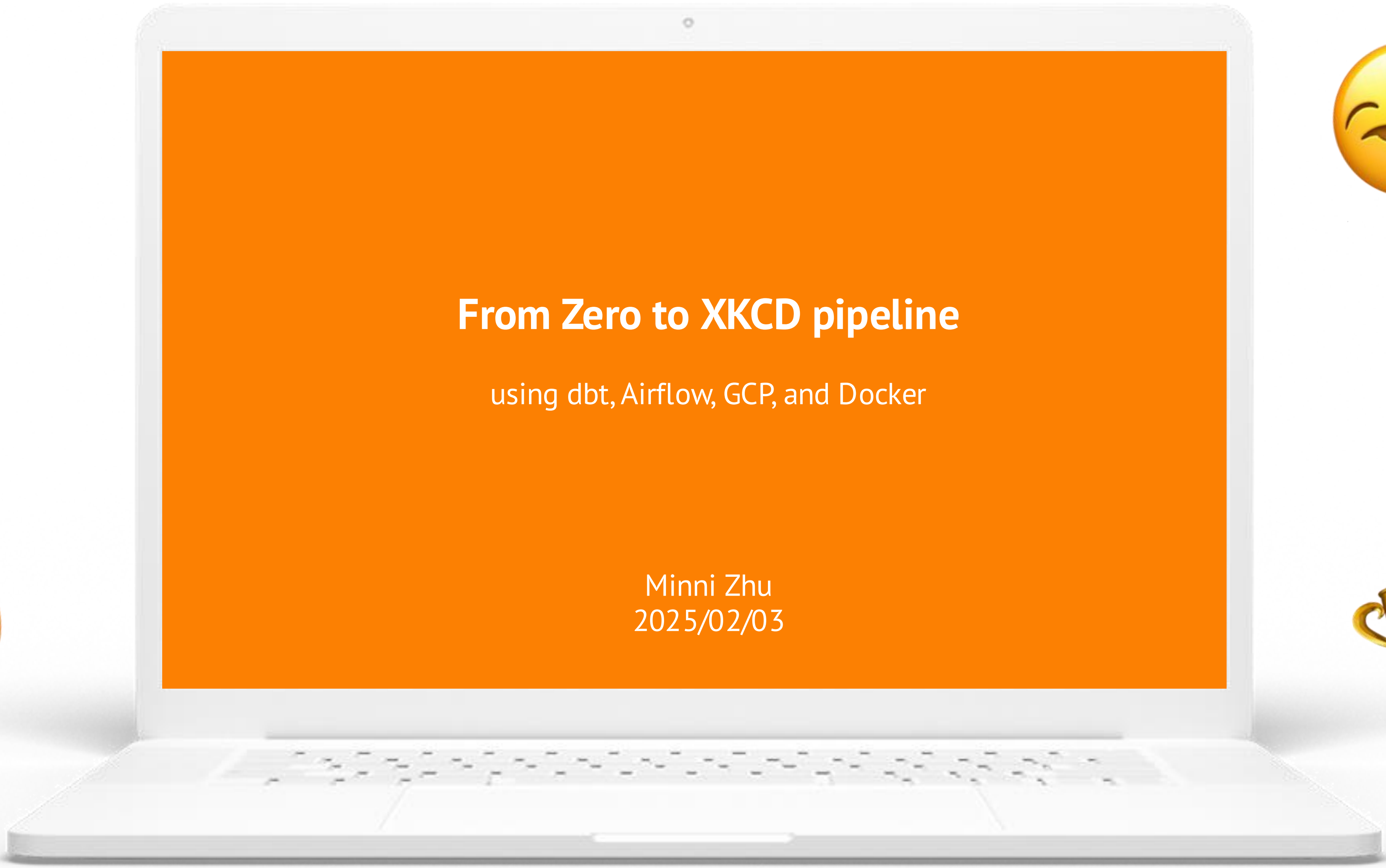
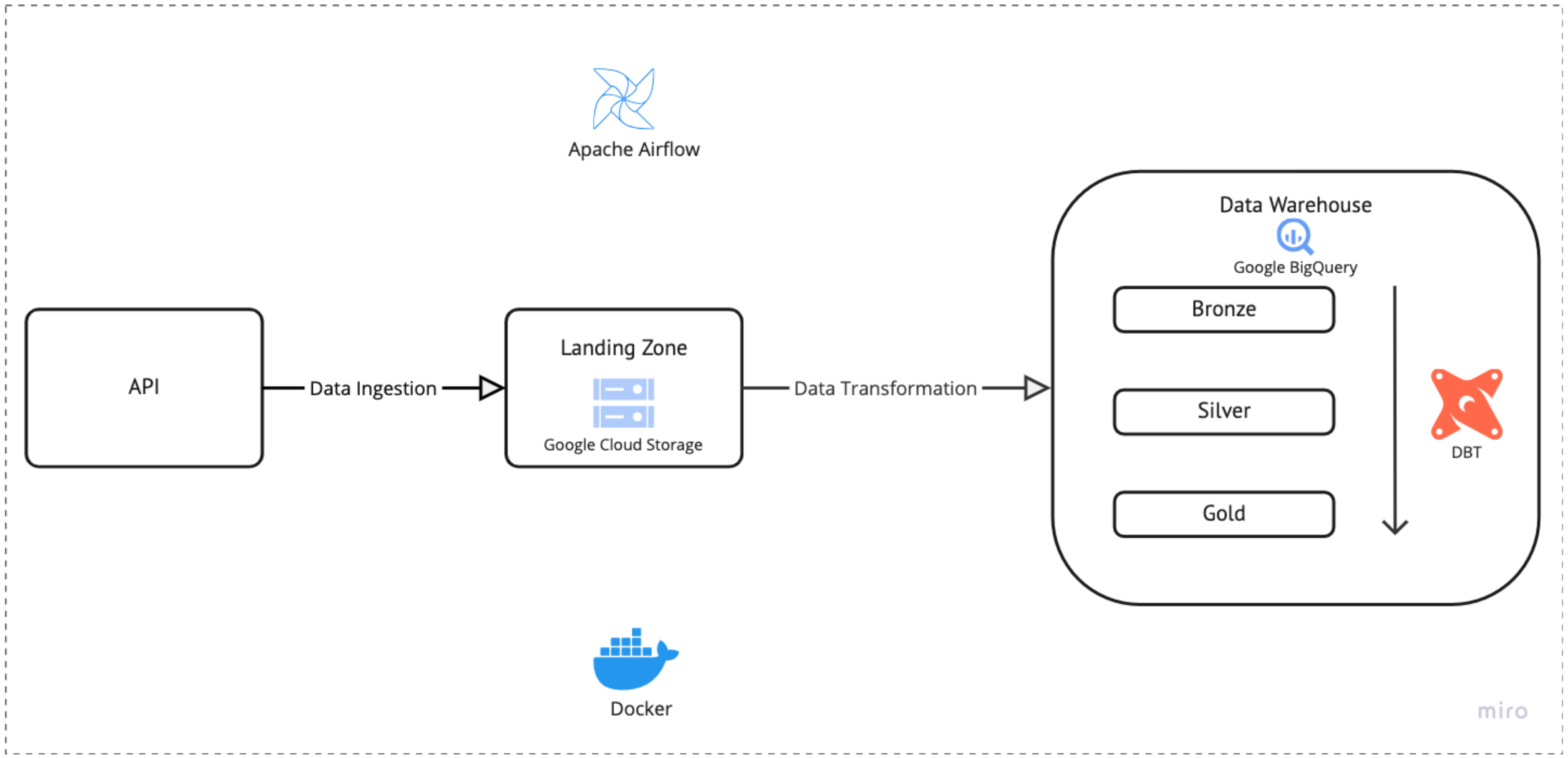


From Zero to XKCD pipeline

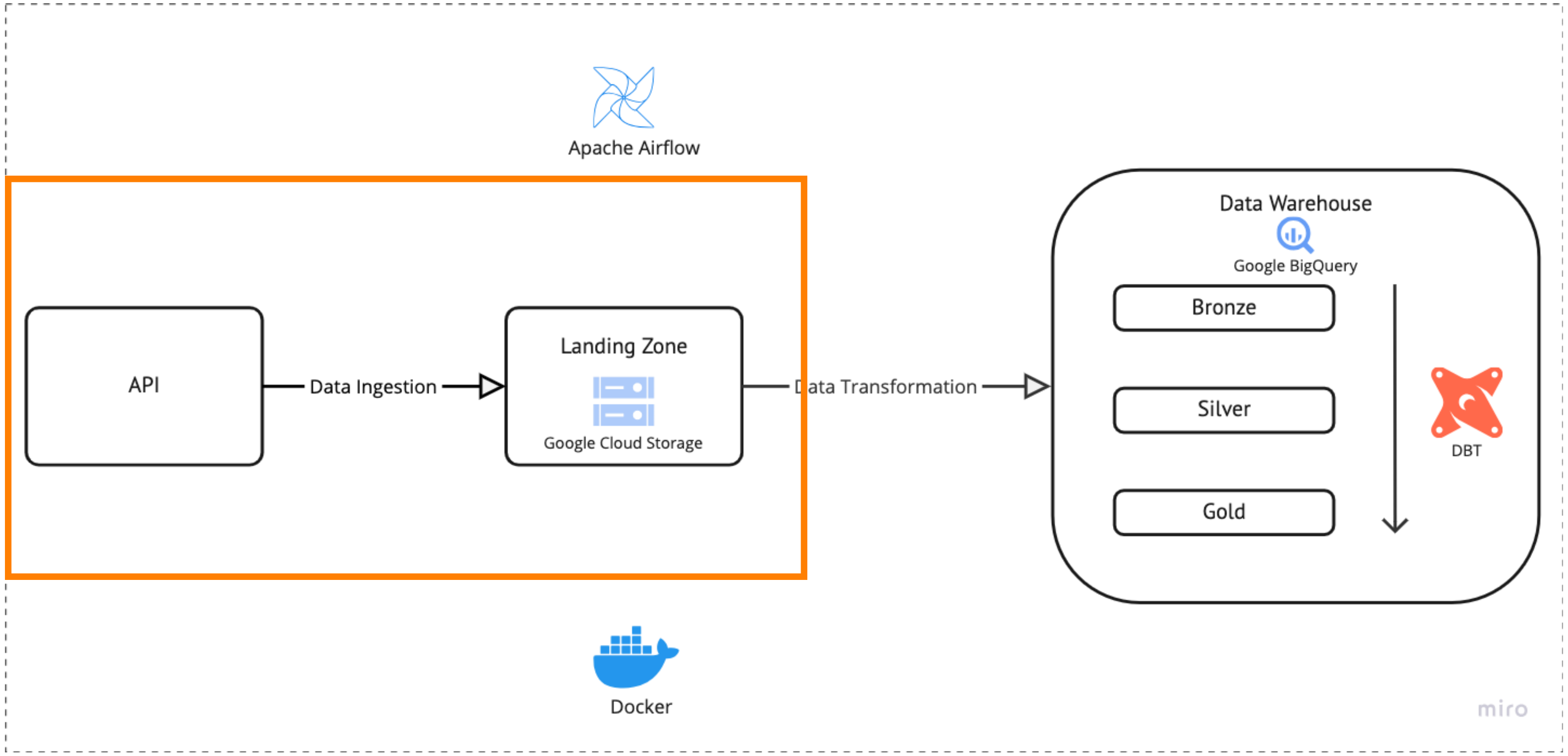
using dbt, Airflow, GCP, and Docker

Minni Zhu
2025/02/03

Pipeline Architechture



Get Started: Data Ingestion



Get Started: Data Ingestion

Before coding:

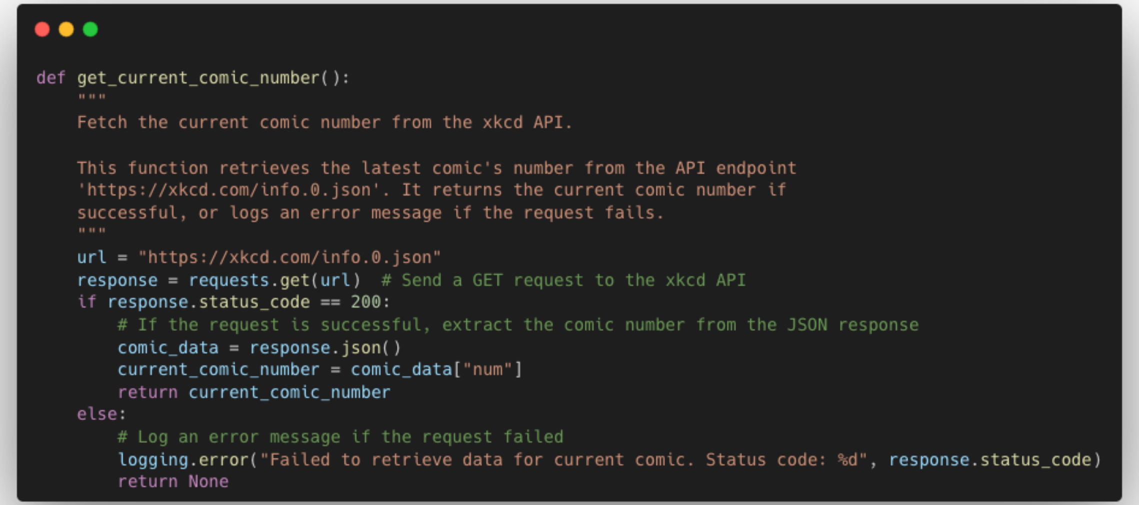
https://xkcd.com/info.0.json returns latest comic data

https://xkcd.com/{day}/info.0.json returns a specific day's comic data

Field	Meaning
month	The month the comic was published.
num	The comic number (unique identifier for each comic).
link	An optional link associated with the comic.
year	The year the comic was published.
news	Any news or updates related to the comic.
safe_title	A user-friendly title for the comic, meant to be more descriptive and suitable for display.
transcript	The transcript for the comic, which would include any dialogue or text.
alt	The alt text for the comic, often containing humor or additional context.
img	The URL of the comic image.
title	The title of the comic (typically the same as the safe_title).
day	The day the comic was published.

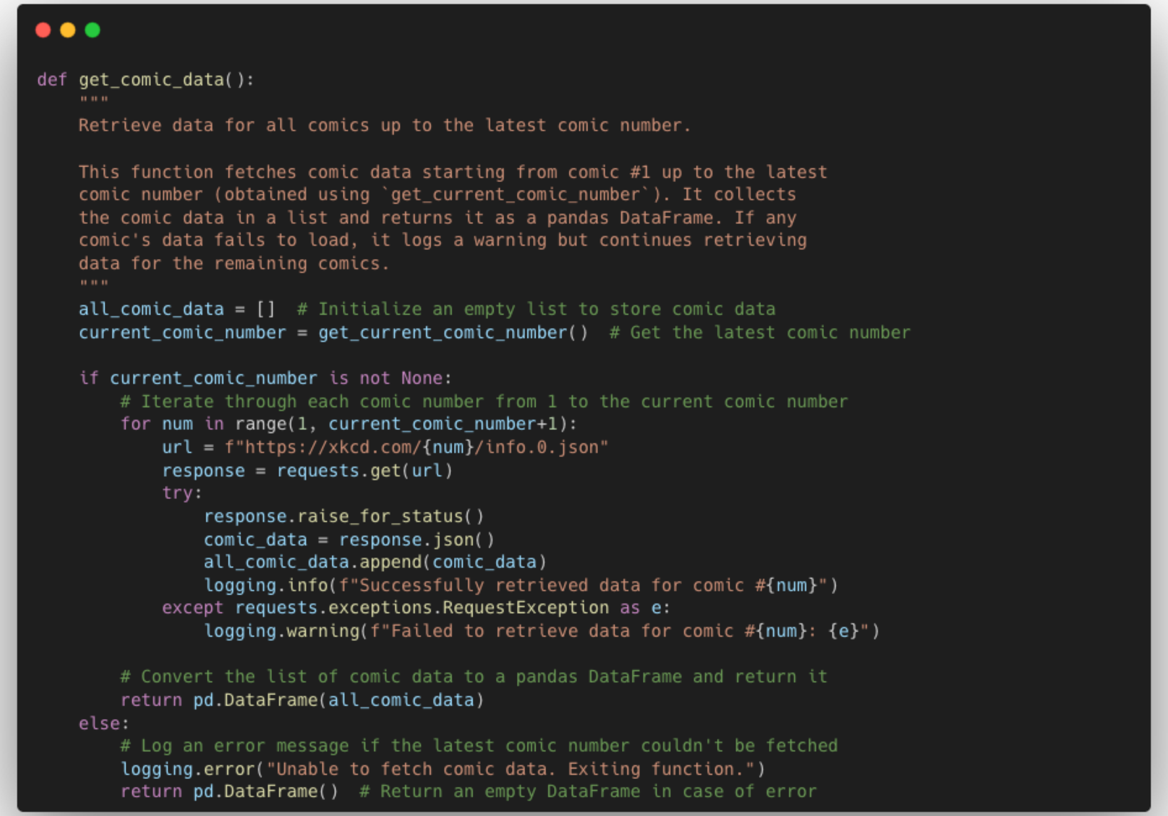
Data Ingestion 1.0 DAG– Step 1: Functions to extract comic data

get latest comic number



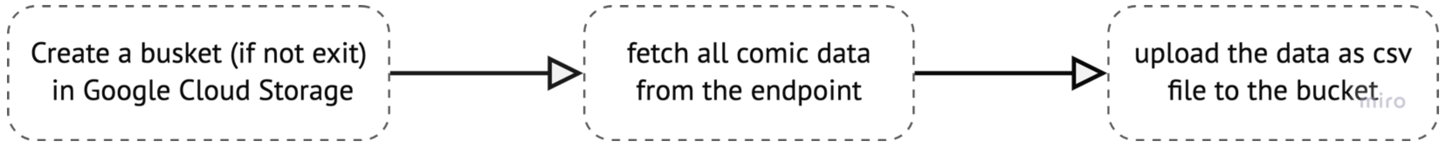
```python
def get_current_comic_number():
    """
    Fetch the current comic number from the xkcd API.

    This function retrieves the latest comic's number from the API endpoint
    'https://xkcd.com/info.0.json'. It returns the current comic number if
    successful, or logs an error message if the request fails.
    """
    url = "https://xkcd.com/info.0.json"
    response = requests.get(url)  # Send a GET request to the xkcd API
    if response.status_code == 200:
        # If the request is successful, extract the comic number from the JSON response
        comic_data = response.json()
        current_comic_number = comic_data["num"]
        return current_comic_number
    else:
        # Log an error message if the request failed
        logging.error("Failed to retrieve data for current comic. Status code: %d", response.status_code)
        return None
```

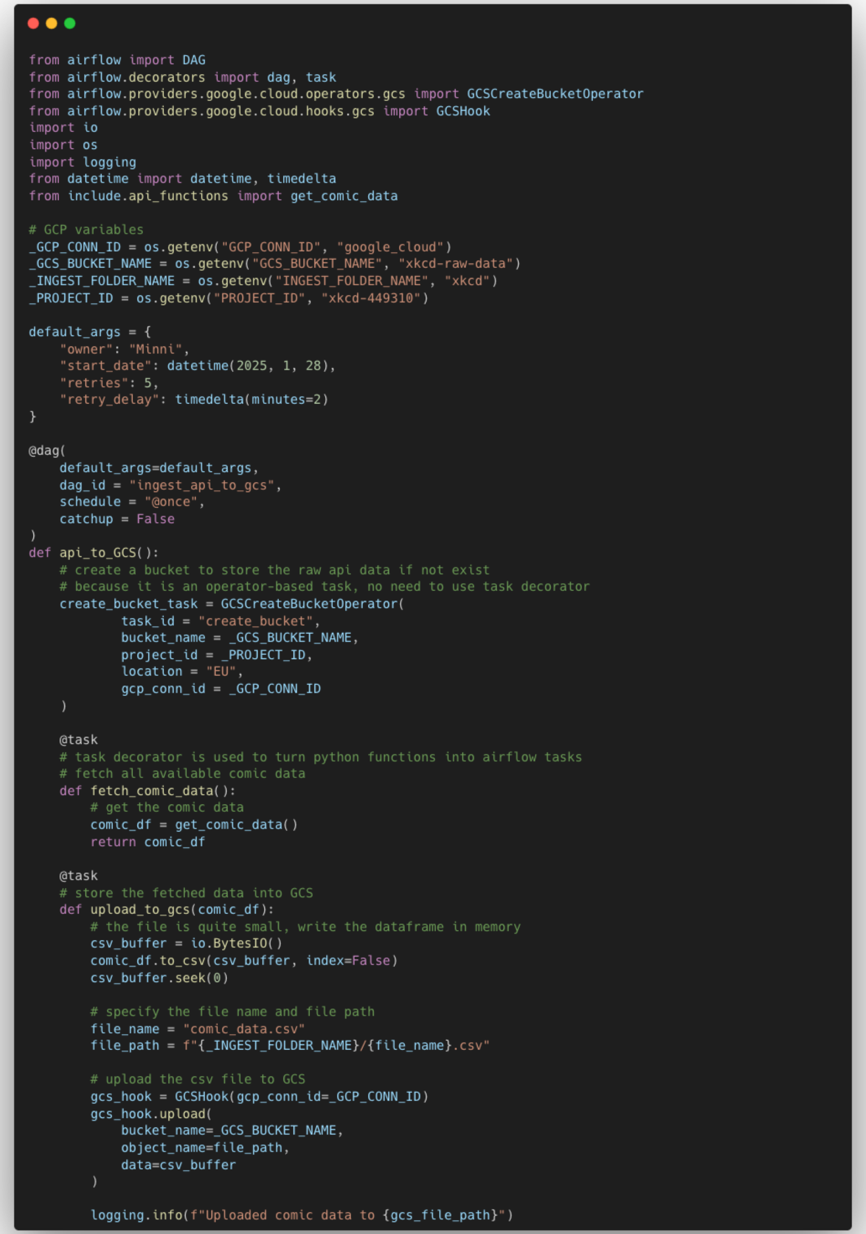
get all comic data



```python
def get_comic_data():
    """
    Retrieve data for all comics up to the latest comic number.

    This function fetches comic data starting from comic #1 up to the latest
    comic number (obtained using `get_current_comic_number`). It collects
    the comic data in a list and returns it as a pandas DataFrame. If any
    comic's data fails to load, it logs a warning but continues retrieving
    data for the remaining comics.
    """
    all_comic_data = []  # Initialize an empty list to store comic data
    current_comic_number = get_current_comic_number()  # Get the latest comic number

    if current_comic_number is not None:
        # Iterate through each comic number from 1 to the current comic number
        for num in range(1, current_comic_number+1):
            url = f"https://xkcd.com/{num}/info.0.json"
            response = requests.get(url)
            try:
                response.raise_for_status()
                comic_data = response.json()
                all_comic_data.append(comic_data)
                logging.info(f"Successfully retrieved data for comic #{num}")
            except requests.exceptions.RequestException as e:
                logging.warning(f"Failed to retrieve data for comic #{num}: {e}")

        # Convert the list of comic data to a pandas DataFrame and return it
        return pd.DataFrame(all_comic_data)
    else:
        # Log an error message if the latest comic number couldn't be fetched
        logging.error("Unable to fetch comic data. Exiting function.")
        return pd.DataFrame()  # Return an empty DataFrame in case of error
```

Data Ingestion 1.0 DAG – Step 2: DAG for data ingestion


Create a busket (if not exit) in Google Cloud Storage
fetch all comic data from the endpoint
upload the data as csv file to the bucket
miro



```python
from airflow import DAG
from airflow.decorators import dag, task
from airflow.providers.google.cloud.operators.gcs import GCSCreateBucketOperator
from airflow.providers.google.cloud.hooks.gcs import GCSHook
import io
import os
import logging
from datetime import datetime, timedelta
from include.api_functions import get_comic_data

# GCP variables
_GCP_CONN_ID = os.getenv("GCP_CONN_ID", "google_cloud")
_GCS_BUCKET_NAME = os.getenv("GCS_BUCKET_NAME", "xkcd-raw-data")
_INGEST_FOLDER_NAME = os.getenv("INGEST_FOLDER_NAME", "xkcd")
_PROJECT_ID = os.getenv("PROJECT_ID", "xkcd-449310")

default_args = {
    "owner": "Minni",
    "start_date": datetime(2025, 1, 28),
    "retries": 5,
    "retry_delay": timedelta(minutes=2)
}

@dag(
    default_args=default_args,
    dag_id = "ingest_api_to_gcs",
    schedule = "@once",
    catchup = False
)
def api_to_GCS():
    # create a bucket to store the raw api data if not exist
    # because it is an operator-based task, no need to use task decorator
    create_bucket_task = GCSCreateBucketOperator(
            task_id = "create_bucket",
            bucket_name = _GCS_BUCKET_NAME,
            project_id = _PROJECT_ID,
            location = "EU",
            gcp_conn_id = _GCP_CONN_ID
    )

    @task
    # task decorator is used to turn python functions into airflow tasks
    # fetch all available comic data
    def fetch_comic_data():
        # get the comic data
        comic_df = get_comic_data()
        return comic_df

    @task
    # store the fetched data into GCS
    def upload_to_gcs(comic_df):
        # the file is quite small, write the dataframe in memory
        csv_buffer = io.BytesIO()
        comic_df.to_csv(csv_buffer, index=False)
        csv_buffer.seek(0)

        # specify the file name and file path
        file_name = "comic_data.csv"
        file_path = f"{_INGEST_FOLDER_NAME}/{file_name}.csv"

        # upload the csv file to GCS
        gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
        gcs_hook.upload(
            bucket_name=_GCS_BUCKET_NAME,
            object_name=file_path,
            data=csv_buffer
        )

        logging.info(f"Uploaded comic data to {gcs_file_path}")
```

Data Ingestion 1.0 – Takeaways

Helps simplify defining tasks in a DAG

Task Decorator

Operators and Hooks

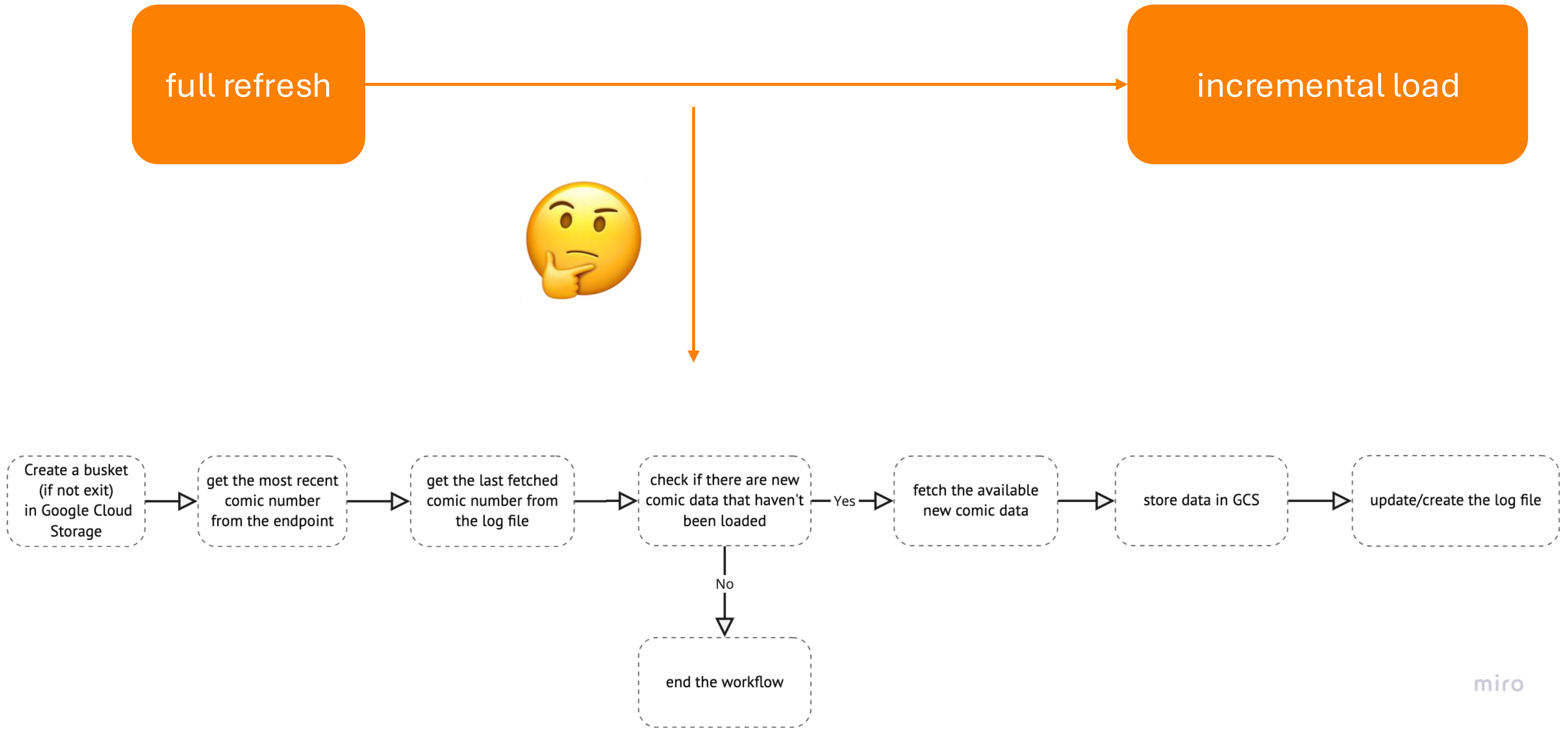
Used to interact with various services

DAG structure

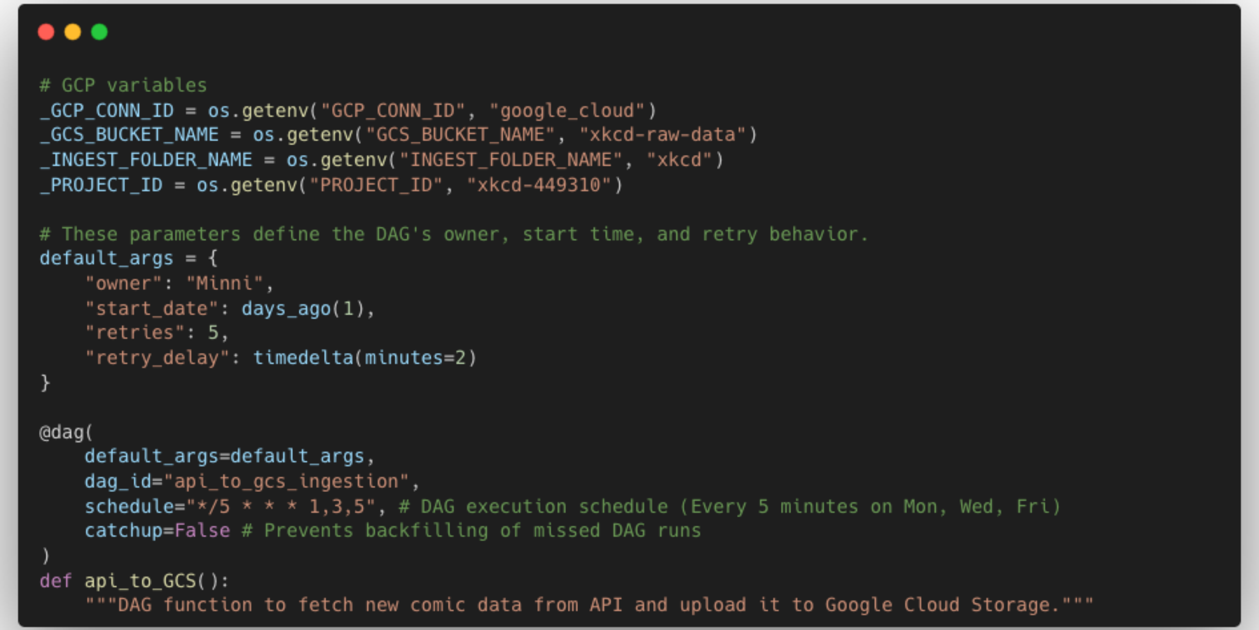
Airflow UI

A huge help for debugging, tracking task progress, and visualizing workflows

Data Ingestion 2.0 – Improving Efficiency with Incremental Load

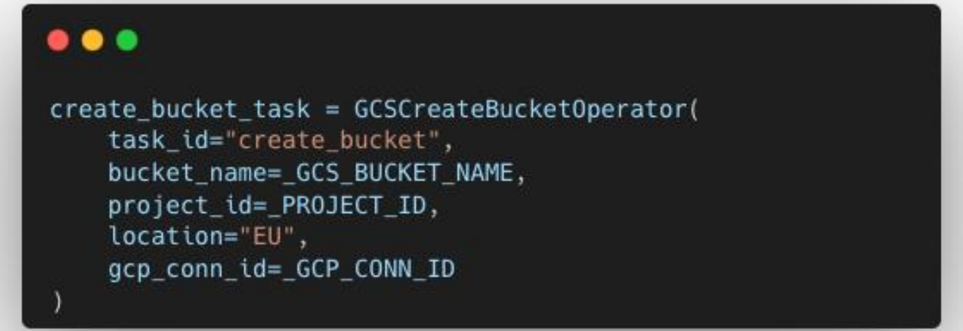

full refresh
incremental load
Create a busket (if not exit) in Google Cloud Storage
get the most recent comic number from the endpoint
get the last fetched comic number from the log file
check if there are new comic data that haven't been loaded
Yes
No
fetch the available new comic data
store data in GCS
update/create the log file
end the workflow

Data Ingestion 2.0 DAG – Define variables and dag configuration



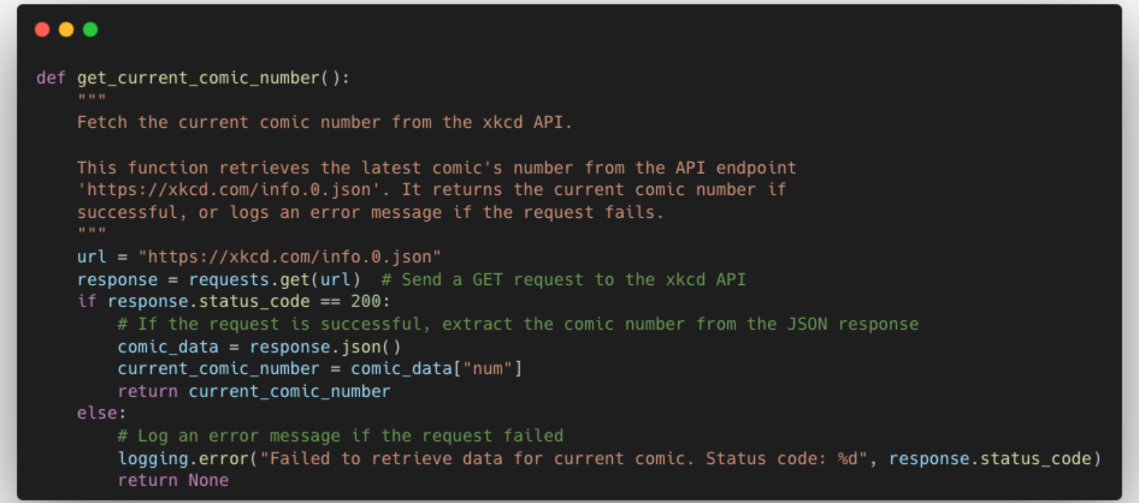
```python
# GCP variables
_GCP_CONN_ID = os.getenv("GCP_CONN_ID", "google_cloud")
_GCS_BUCKET_NAME = os.getenv("GCS_BUCKET_NAME", "xkcd-raw-data")
_INGEST_FOLDER_NAME = os.getenv("INGEST_FOLDER_NAME", "xkcd")
_PROJECT_ID = os.getenv("PROJECT_ID", "xkcd-449310")

# These parameters define the DAG's owner, start time, and retry behavior.
default_args = {
    "owner": "Minni",
    "start_date": days_ago(1),
    "retries": 5,
    "retry_delay": timedelta(minutes=2)
}

@dag(
    default_args=default_args,
    dag_id="api_to_gcs_ingestion",
    schedule="*/5 * * * 1,3,5", # DAG execution schedule (Every 5 minutes on Mon, Wed, Fri)
    catchup=False # Prevents backfilling of missed DAG runs
)
def api_to_GCS():
    """DAG function to fetch new comic data from API and upload it to Google Cloud Storage."""
```

Data Ingestion 2.0 DAG – Create a bucket in GCS

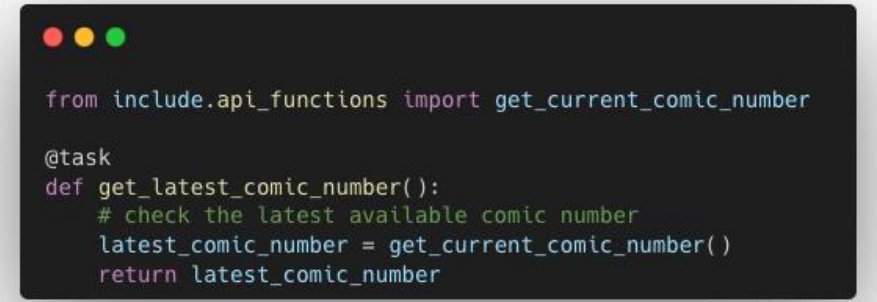


```
create_bucket_task = GCSCreateBucketOperator(
    task_id="create_bucket",
    bucket_name=_GCS_BUCKET_NAME,
    project_id=_PROJECT_ID,
    location="EU",
    gcp_conn_id=_GCP_CONN_ID
)
```

Data Ingestion 2.0 DAG – Get the most recent comic number

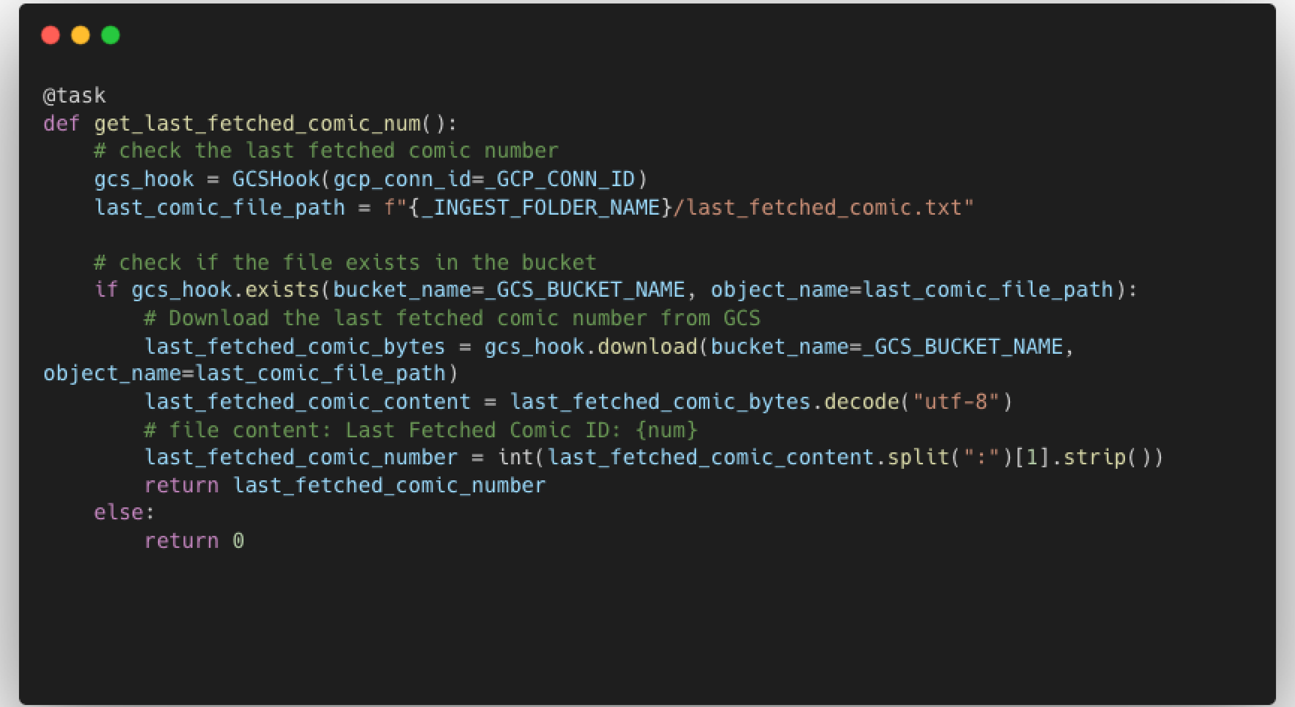


```python
def get_current_comic_number():
    """
    Fetch the current comic number from the xkcd API.

    This function retrieves the latest comic's number from the API endpoint
    'https://xkcd.com/info.0.json'. It returns the current comic number if
    successful, or logs an error message if the request fails.
    """
    url = "https://xkcd.com/info.0.json"
    response = requests.get(url)  # Send a GET request to the xkcd API
    if response.status_code == 200:
        # If the request is successful, extract the comic number from the JSON response
        comic_data = response.json()
        current_comic_number = comic_data["num"]
        return current_comic_number
    else:
        # Log an error message if the request failed
        logging.error("Failed to retrieve data for current comic. Status code: %d", response.status_code)
        return None
```



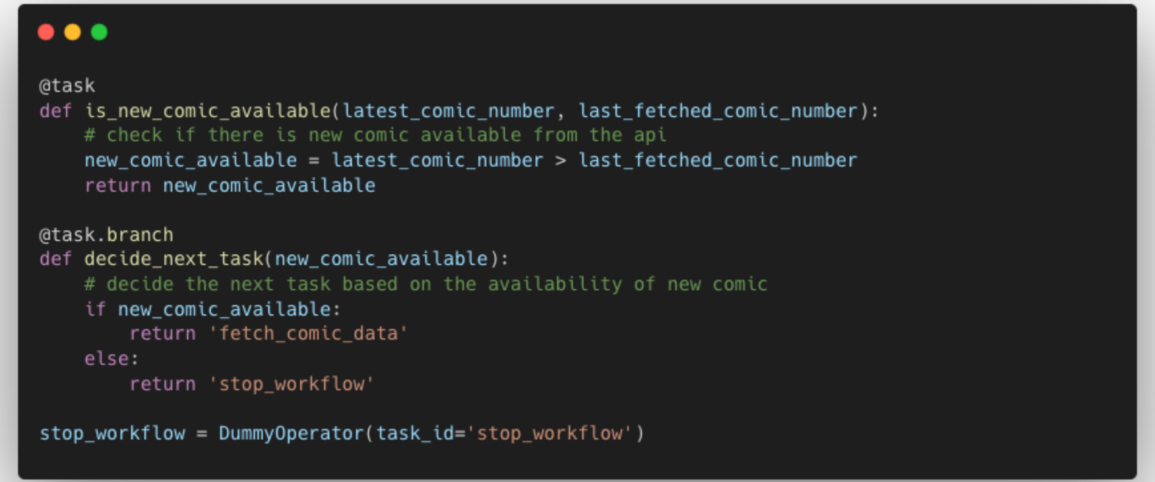
```python
from include.api_functions import get_current_comic_number

@task
def get_latest_comic_number():
    # check the latest available comic number
    latest_comic_number = get_current_comic_number()
    return latest_comic_number
```

Data Ingestion 2.0 DAG – Get the last fetched comic number from the log file



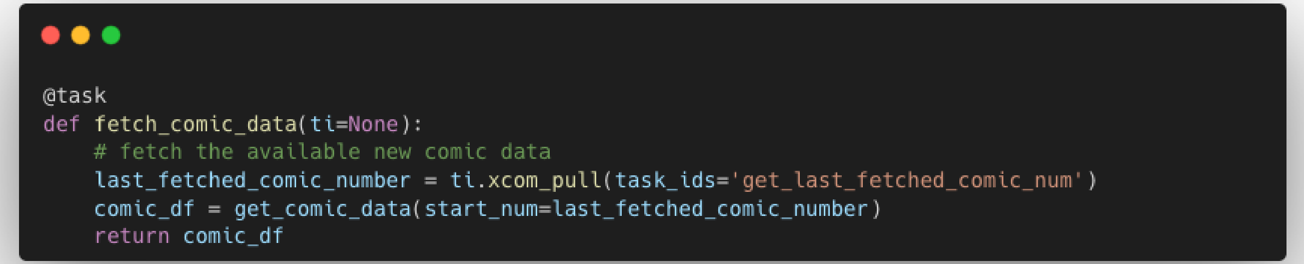
```
@task
def get_last_fetched_comic_num():
    # check the last fetched comic number
    gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
    last_comic_file_path = f"{_INGEST_FOLDER_NAME}/last_fetched_comic.txt"

    # check if the file exists in the bucket
    if gcs_hook.exists(bucket_name=_GCS_BUCKET_NAME, object_name=last_comic_file_path):
        # Download the last fetched comic number from GCS
        last_fetched_comic_bytes = gcs_hook.download(bucket_name=_GCS_BUCKET_NAME,
object_name=last_comic_file_path)
        last_fetched_comic_content = last_fetched_comic_bytes.decode("utf-8")
        # file content: Last Fetched Comic ID: {num}
        last_fetched_comic_number = int(last_fetched_comic_content.split(":")[1].strip())
        return last_fetched_comic_number
    else:
        return 0
```

Data Ingestion 2.0 DAG – Check if there is new comic data

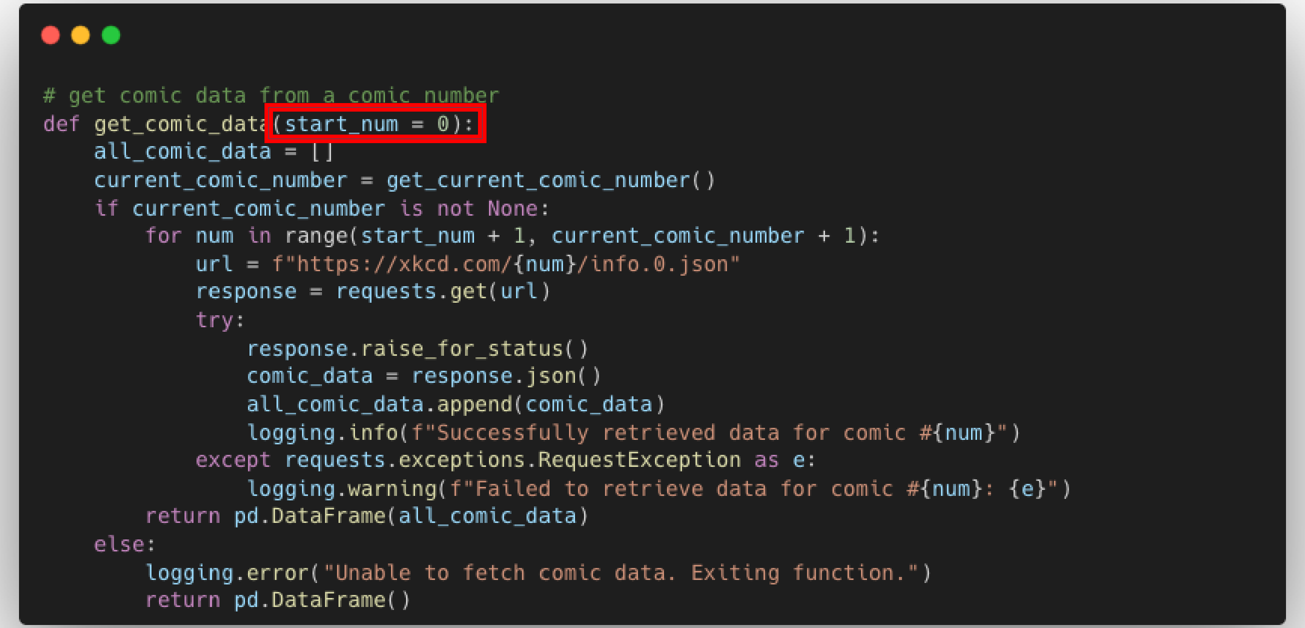


```python
@task
def is_new_comic_available(latest_comic_number, last_fetched_comic_number):
    # check if there is new comic available from the api
    new_comic_available = latest_comic_number > last_fetched_comic_number
    return new_comic_available

@task.branch
def decide_next_task(new_comic_available):
    # decide the next task based on the availability of new comic
    if new_comic_available:
        return 'fetch_comic_data'
    else:
        return 'stop_workflow'

stop_workflow = DummyOperator(task_id='stop_workflow')
```

Data Ingestion 2.0 DAG – Fetch the new available comic data

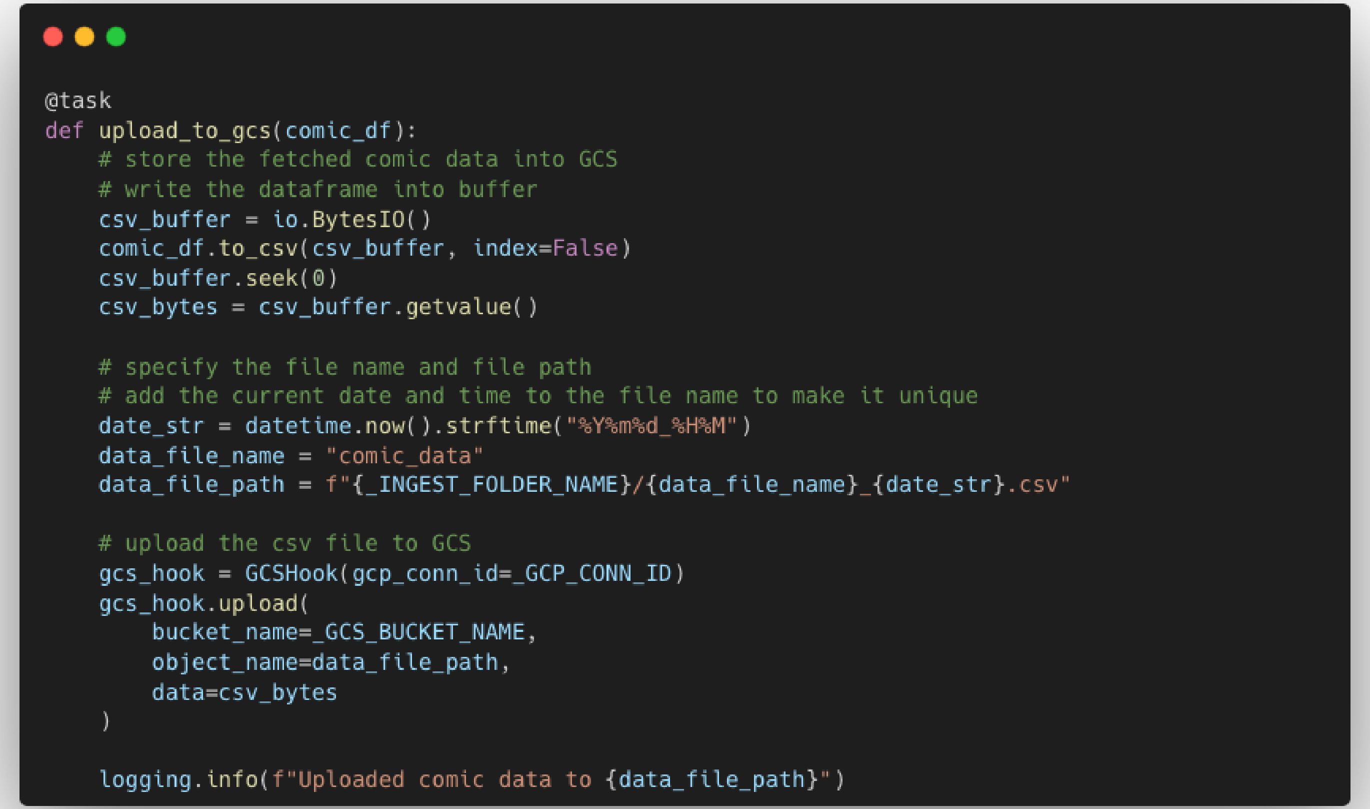


```python
@task
def fetch_comic_data(ti=None):
    # fetch the available new comic data
    last_fetched_comic_number = ti.xcom_pull(task_ids='get_last_fetched_comic_num')
    comic_df = get_comic_data(start_num=last_fetched_comic_number)
    return comic_df
```



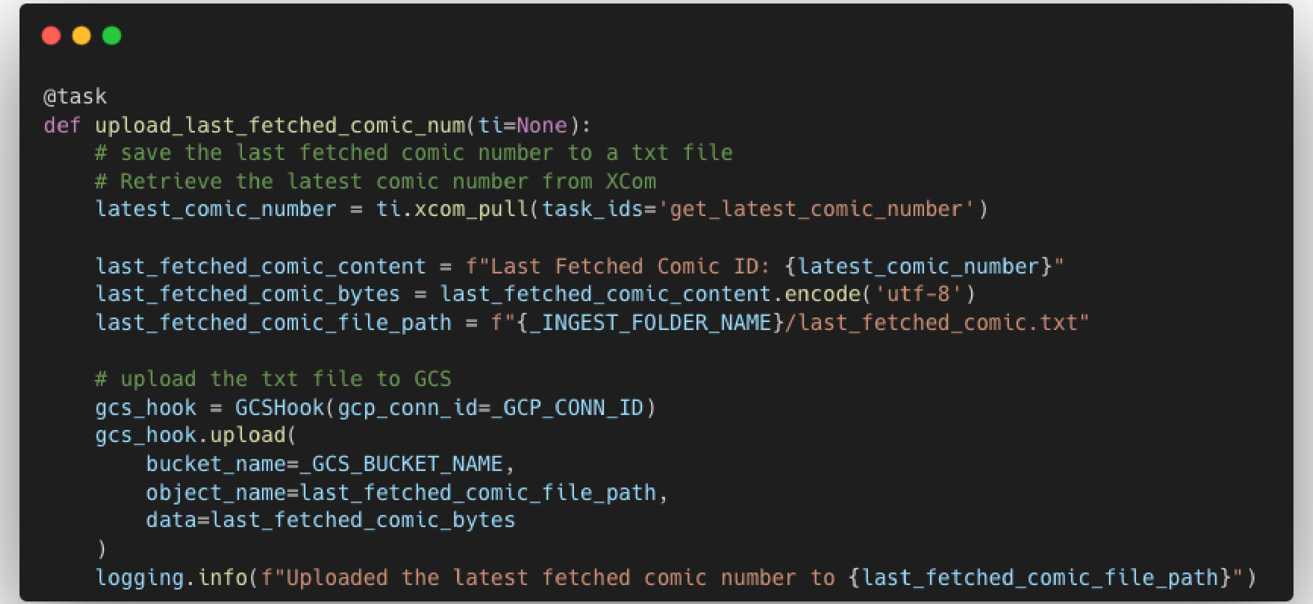
```python
# get comic data from a comic number
def get_comic_data(start_num = 0):
    all_comic_data = []
    current_comic_number = get_current_comic_number()
    if current_comic_number is not None:
        for num in range(start_num + 1, current_comic_number + 1):
            url = f"https://xkcd.com/{num}/info.0.json"
            response = requests.get(url)
            try:
                response.raise_for_status()
                comic_data = response.json()
                all_comic_data.append(comic_data)
                logging.info(f"Successfully retrieved data for comic #{num}")
            except requests.exceptions.RequestException as e:
                logging.warning(f"Failed to retrieve data for comic #{num}: {e}")
        return pd.DataFrame(all_comic_data)
    else:
        logging.error("Unable to fetch comic data. Exiting function.")
        return pd.DataFrame()
```

Data Ingestion 2.0 DAG – Store the new data in GCS



```python
@task
def upload_to_gcs(comic_df):
    # store the fetched comic data into GCS
    # write the dataframe into buffer
    csv_buffer = io.BytesIO()
    comic_df.to_csv(csv_buffer, index=False)
    csv_buffer.seek(0)
    csv_bytes = csv_buffer.getvalue()

    # specify the file name and file path
    # add the current date and time to the file name to make it unique
    date_str = datetime.now().strftime("%Y%m%d_%H%M")
    data_file_name = "comic_data"
    data_file_path = f"{_INGEST_FOLDER_NAME}/{data_file_name}_{date_str}.csv"

    # upload the csv file to GCS
    gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
    gcs_hook.upload(
        bucket_name=_GCS_BUCKET_NAME,
        object_name=data_file_path,
        data=csv_bytes
    )

    logging.info(f"Uploaded comic data to {data_file_path}")
```

Data Ingestion 2.0 DAG – Update or create the log file to store the last fetched comic number



```python
@task
def upload_last_fetched_comic_num(ti=None):
    # save the last fetched comic number to a txt file
    # Retrieve the latest comic number from XCom
    latest_comic_number = ti.xcom_pull(task_ids='get_latest_comic_number')

    last_fetched_comic_content = f"Last Fetched Comic ID: {latest_comic_number}"
    last_fetched_comic_bytes = last_fetched_comic_content.encode('utf-8')
    last_fetched_comic_file_path = f"{_INGEST_FOLDER_NAME}/last_fetched_comic.txt"

    # upload the txt file to GCS
    gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
    gcs_hook.upload(
        bucket_name=_GCS_BUCKET_NAME,
        object_name=last_fetched_comic_file_path,
        data=last_fetched_comic_bytes
    )
    logging.info(f"Uploaded the latest fetched comic number to {last_fetched_comic_file_path}")
```

Data Ingestion 2.0 – Challenges and Takeaways

Error Handeling

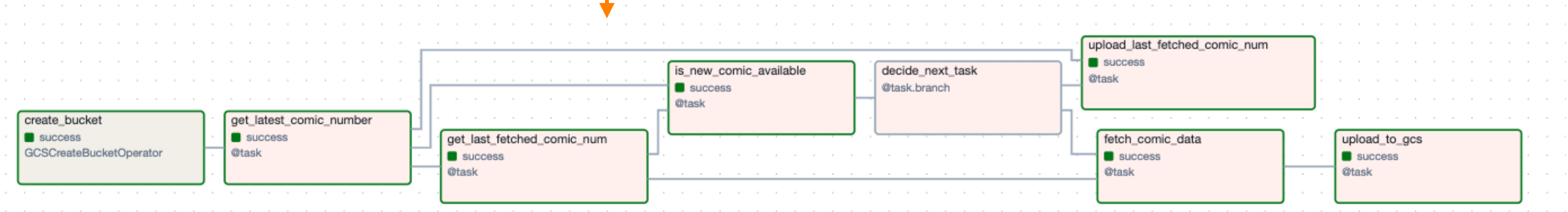
DAG graph

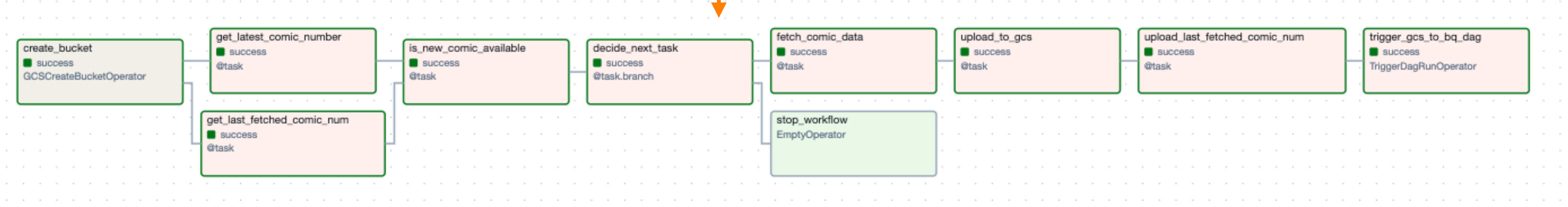
Xcom

Task Branching

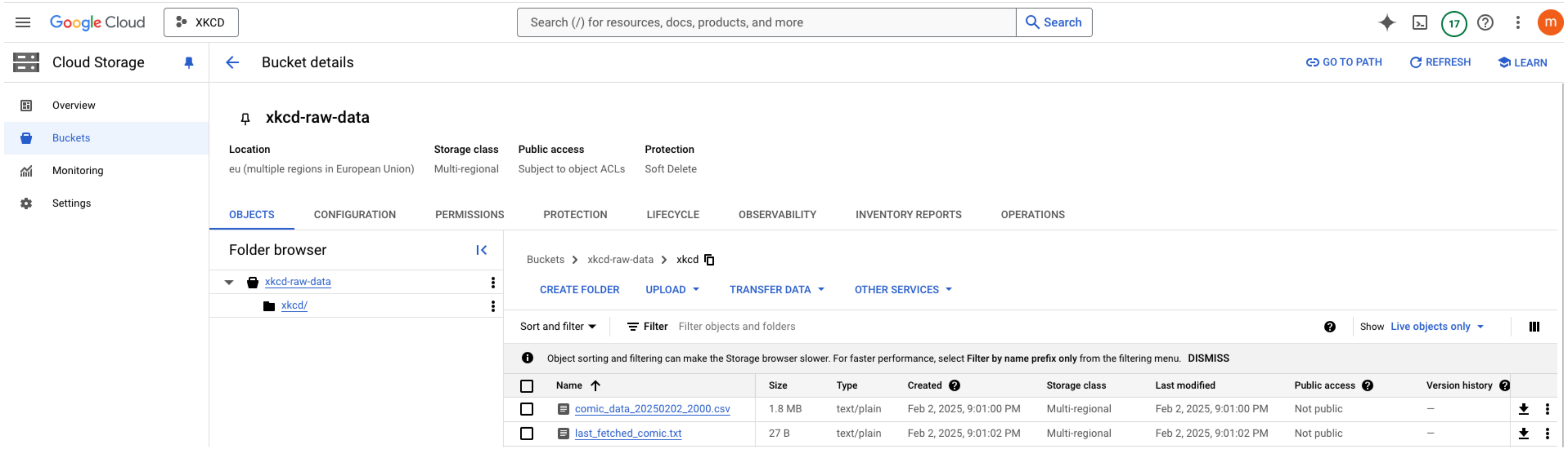
decide which tasks to run based on certain conditions

pass data between tasks without having to store it outside the pipeline





Data Ingestion 2.0 – Check data in GCS



xkcd-raw-data

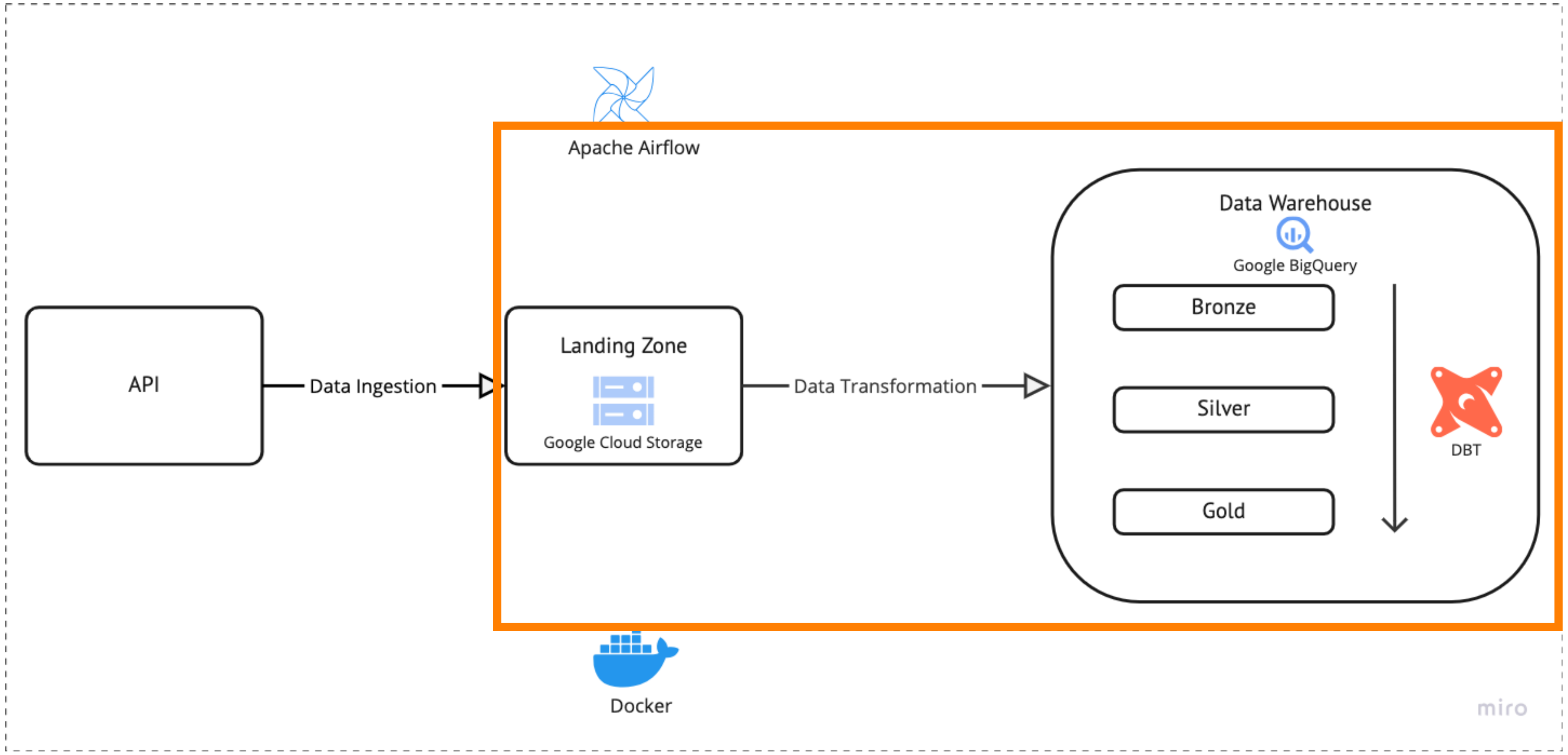
Location	Storage class	Public access	Protection
eu (multiple regions in European Union)	Multi-regional	Subject to object ACLs	Soft Delete

Buckets > xkcd-raw-data > xkcd

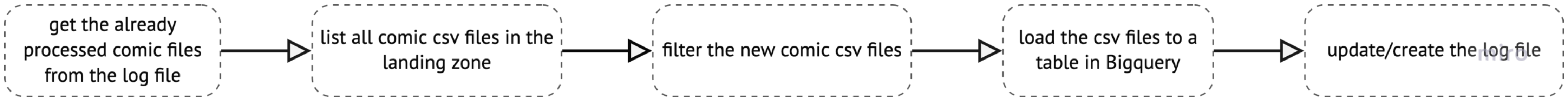
Object sorting and filtering can make the Storage browser slower. For faster performance, select **Filter by name prefix only** from the filtering menu. DISMISS

Name	Size	Type	Created	Storage class	Last modified	Public access	Version history
comic_data_20250202_2000.csv	1.8 MB	text/plain	Feb 2, 2025, 9:01:00 PM	Multi-regional	Feb 2, 2025, 9:01:00 PM	Not public	–
last_fetched_comic.txt	27 B	text/plain	Feb 2, 2025, 9:01:02 PM	Multi-regional	Feb 2, 2025, 9:01:02 PM	Not public	–

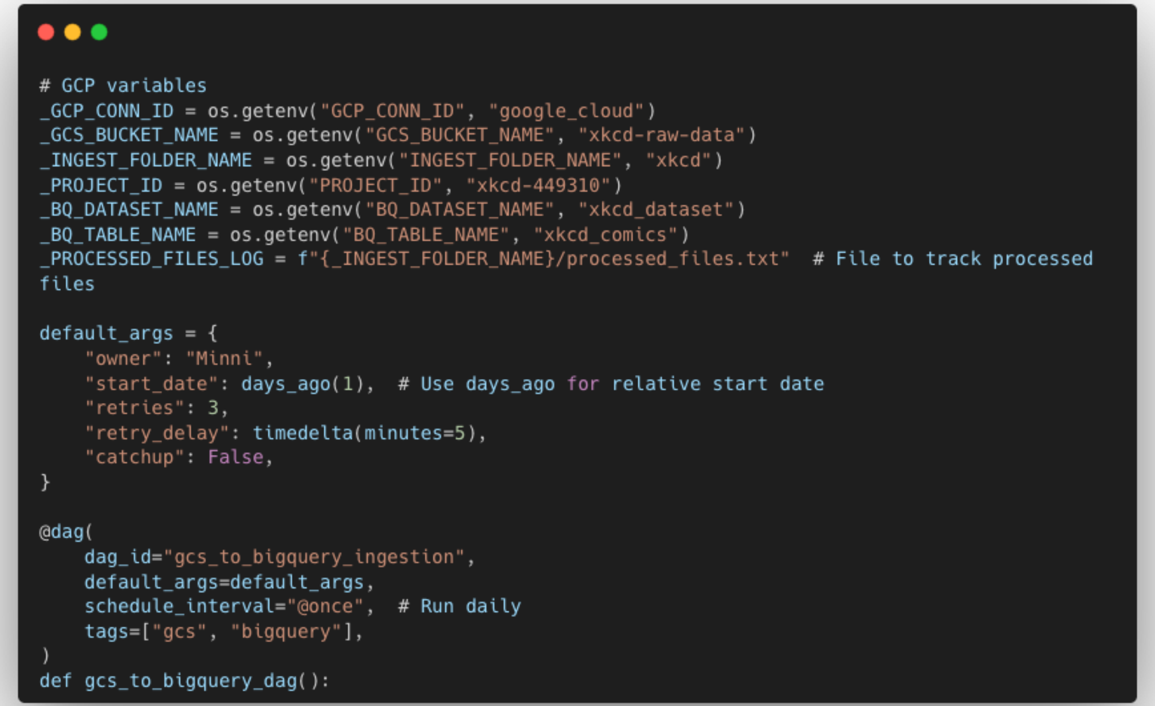
Move on – Data transformation



Data transformation – step 1: GCS (landing zone) to Bigquery (data warehouse)

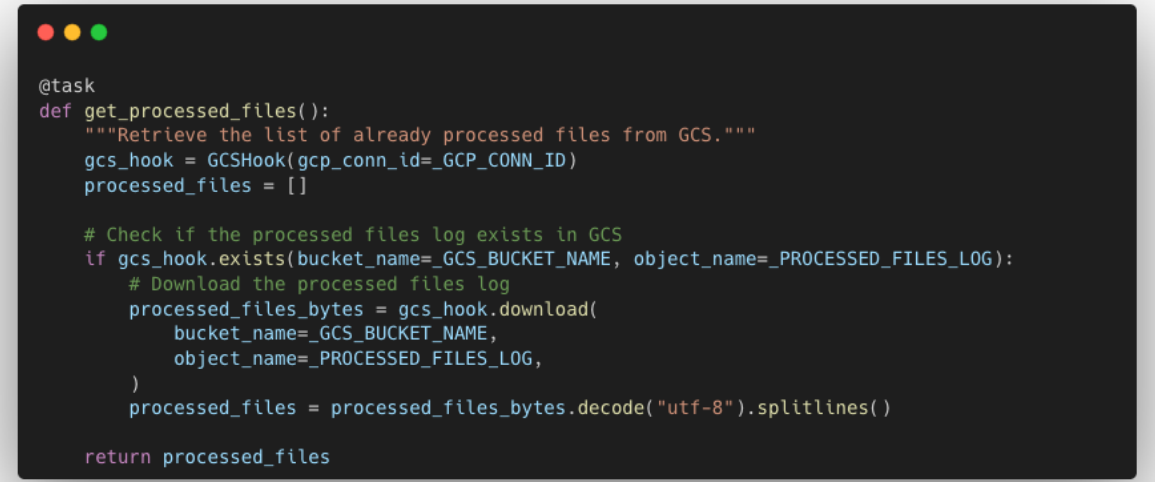

get the already processed comic files from the log file
list all comic csv files in the landing zone
filter the new comic csv files
load the csv files to a table in Bigquery
update/create the log file

Data transformation DAG – Define variables and dag configuration



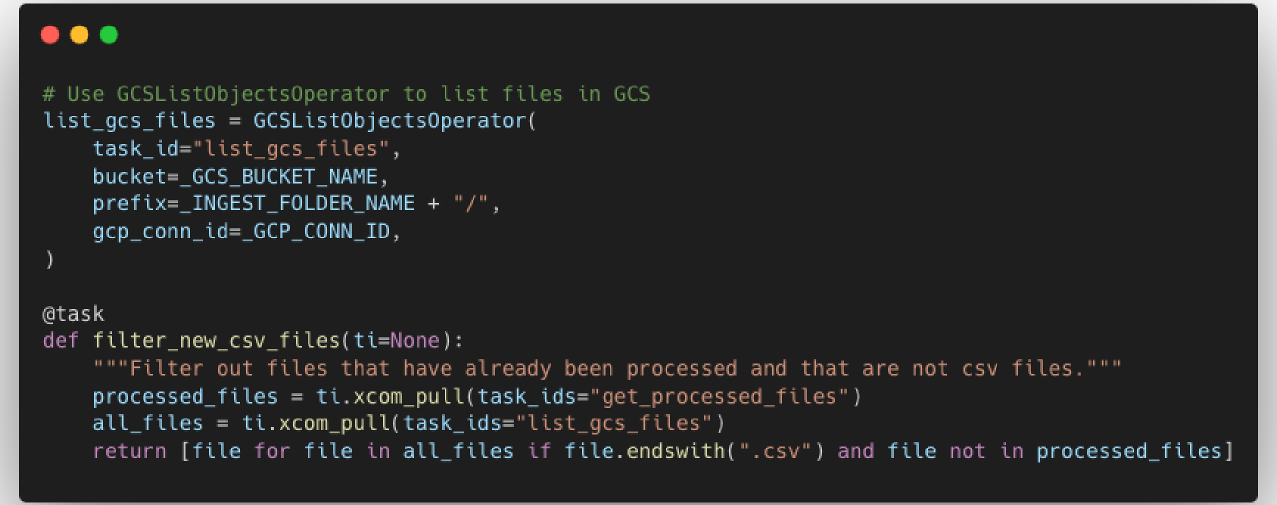
```
# GCP variables
_GCP_CONN_ID = os.getenv("GCP_CONN_ID", "google_cloud")
_GCS_BUCKET_NAME = os.getenv("GCS_BUCKET_NAME", "xkcd-raw-data")
_INGEST_FOLDER_NAME = os.getenv("INGEST_FOLDER_NAME", "xkcd")
_PROJECT_ID = os.getenv("PROJECT_ID", "xkcd-449310")
_BQ_DATASET_NAME = os.getenv("BQ_DATASET_NAME", "xkcd_dataset")
_BQ_TABLE_NAME = os.getenv("BQ_TABLE_NAME", "xkcd_comics")
_PROCESSED_FILES_LOG = f"{_INGEST_FOLDER_NAME}/processed_files.txt"  # File to track processed
files

default_args = {
    "owner": "Minni",
    "start_date": days_ago(1),  # Use days_ago for relative start date
    "retries": 3,
    "retry_delay": timedelta(minutes=5),
    "catchup": False,
}

@dag(
    dag_id="gcs_to_bigquery_ingestion",
    default_args=default_args,
    schedule_interval="@once",  # Run daily
    tags=["gcs", "bigquery"],
)
def gcs_to_bigquery_dag():
```

Data transformation DAG – Get already processed file list



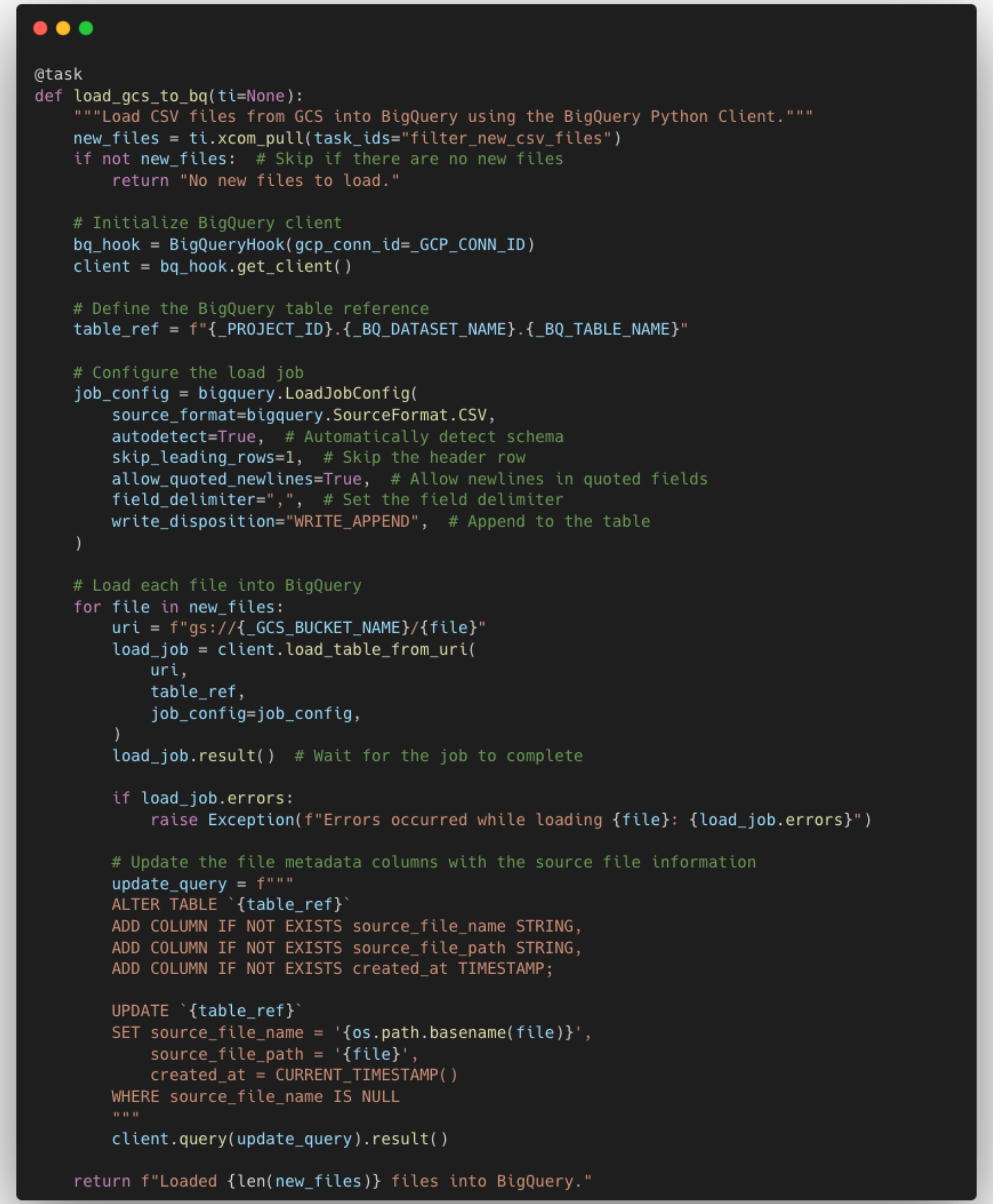
```python
@task
def get_processed_files():
    """Retrieve the list of already processed files from GCS."""
    gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
    processed_files = []

    # Check if the processed files log exists in GCS
    if gcs_hook.exists(bucket_name=_GCS_BUCKET_NAME, object_name=_PROCESSED_FILES_LOG):
        # Download the processed files log
        processed_files_bytes = gcs_hook.download(
            bucket_name=_GCS_BUCKET_NAME,
            object_name=_PROCESSED_FILES_LOG,
        )
        processed_files = processed_files_bytes.decode("utf-8").splitlines()

    return processed_files
```

Data transformation DAG – Filter files that haven’t been processed



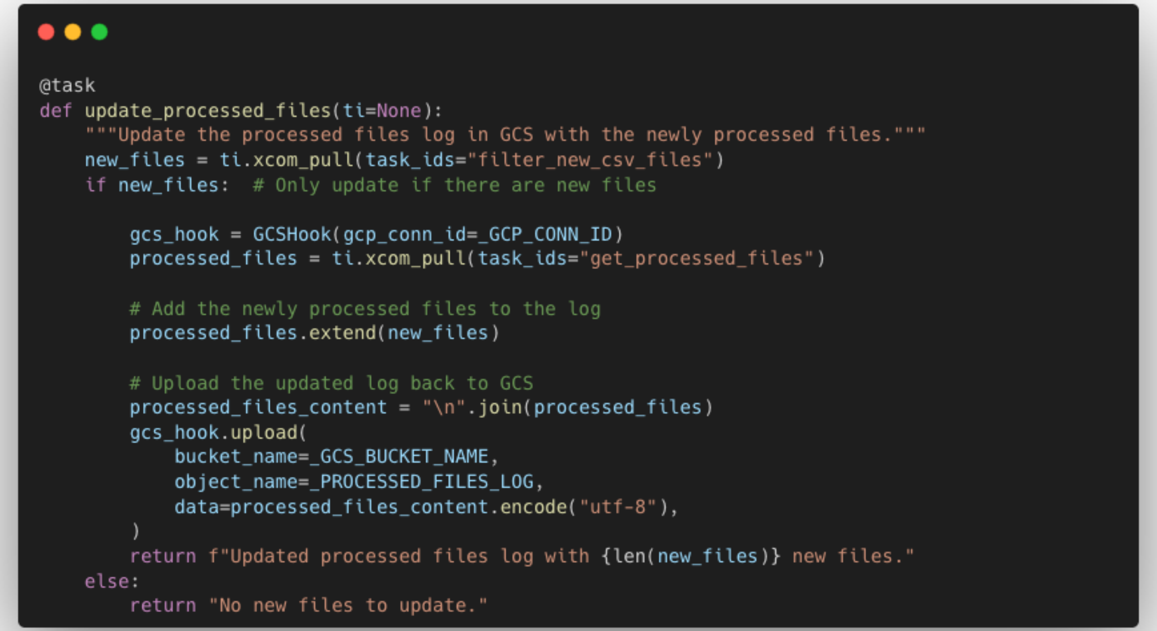
```python
# Use GCSListObjectsOperator to list files in GCS
list_gcs_files = GCSListObjectsOperator(
    task_id="list_gcs_files",
    bucket=_GCS_BUCKET_NAME,
    prefix=_INGEST_FOLDER_NAME + "/",
    gcp_conn_id=_GCP_CONN_ID,
)

@task
def filter_new_csv_files(ti=None):
    """Filter out files that have already been processed and that are not csv files."""
    processed_files = ti.xcom_pull(task_ids="get_processed_files")
    all_files = ti.xcom_pull(task_ids="list_gcs_files")
    return [file for file in all_files if file.endswith(".csv") and file not in processed_files]
```

Data transformation DAG – Load the new comic data to Bigquery



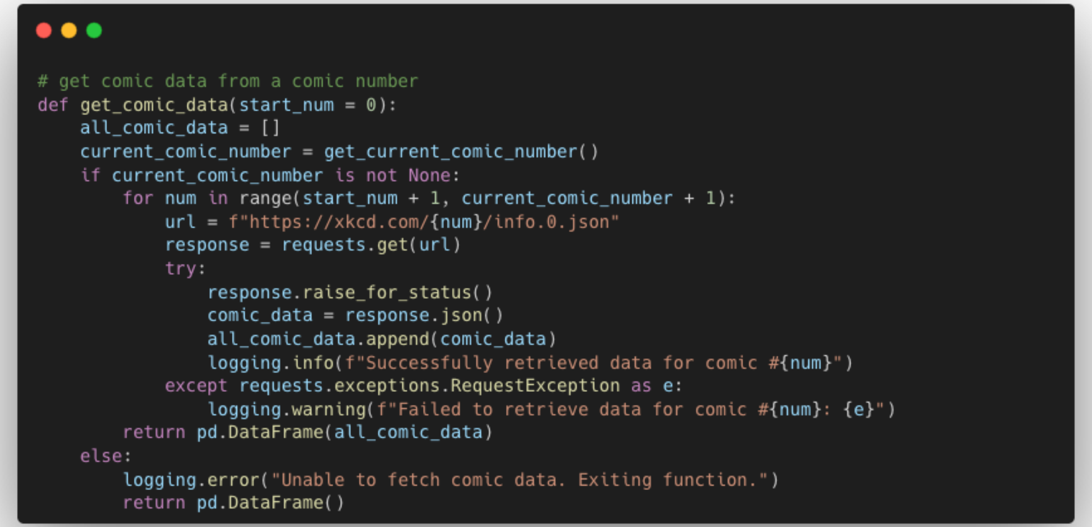
```python
@task
def load_gcs_to_bq(ti=None):
    """Load CSV files from GCS into BigQuery using the BigQuery Python Client."""
    new_files = ti.xcom_pull(task_ids="filter_new_csv_files")
    if not new_files:  # Skip if there are no new files
        return "No new files to load."

    # Initialize BigQuery client
    bq_hook = BigQueryHook(gcp_conn_id=GCP_CONN_ID)
    client = bq_hook.get_client()

    # Define the BigQuery table reference
    table_ref = f"{_PROJECT_ID}.{_BQ_DATASET_NAME}.{_BQ_TABLE_NAME}"

    # Configure the load job
    job_config = bigquery.LoadJobConfig(
        source_format=bigquery.SourceFormat.CSV,
        autodetect=True,  # Automatically detect schema
        skip_leading_rows=1,  # Skip the header row
        allow_quoted_newlines=True,  # Allow newlines in quoted fields
        field_delimiter=",",  # Set the field delimiter
        write_disposition="WRITE_APPEND",  # Append to the table
    )

    # Load each file into BigQuery
    for file in new_files:
        uri = f"gs://{_GCS_BUCKET_NAME}/{file}"
        load_job = client.load_table_from_uri(
            uri,
            table_ref,
            job_config=job_config,
        )
        load_job.result()  # Wait for the job to complete

        if load_job.errors:
            raise Exception(f"Errors occurred while loading {file}: {load_job.errors}")

        # Update the file metadata columns with the source file information
        update_query = f"""
        ALTER TABLE `{table_ref}`
        ADD COLUMN IF NOT EXISTS source_file_name STRING,
        ADD COLUMN IF NOT EXISTS source_file_path STRING,
        ADD COLUMN IF NOT EXISTS created_at TIMESTAMP;

        UPDATE `{table_ref}`
        SET source_file_name = '{os.path.basename(file)}',
            source_file_path = '{file}',
            created_at = CURRENT_TIMESTAMP()
        WHERE source_file_name IS NULL
        """
        client.query(update_query).result()

    return f"Loaded {len(new_files)} files into BigQuery."
```

Data transformation DAG – Update the log file

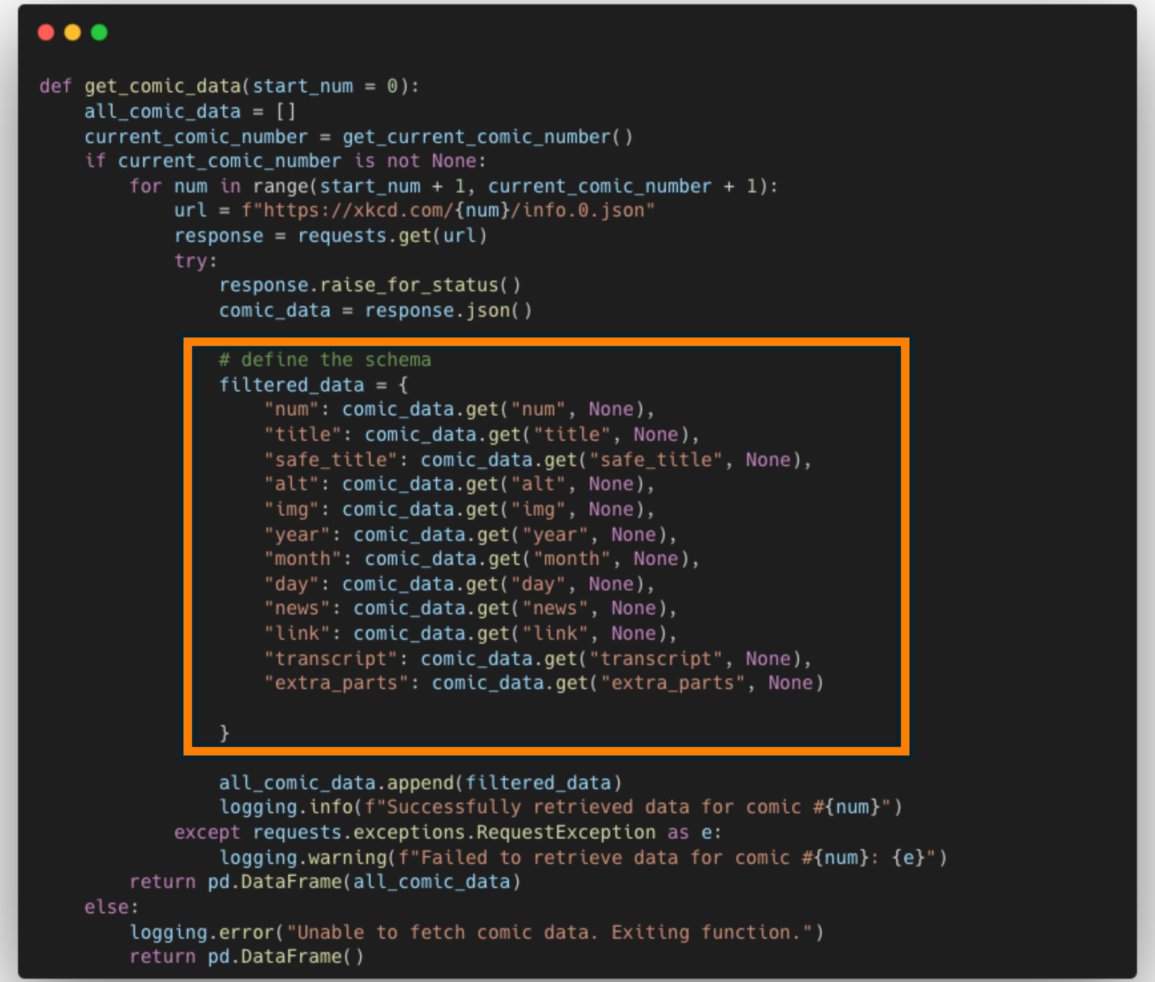


```python
@task
def update_processed_files(ti=None):
    """Update the processed files log in GCS with the newly processed files."""
    new_files = ti.xcom_pull(task_ids="filter_new_csv_files")
    if new_files:  # Only update if there are new files

        gcs_hook = GCSHook(gcp_conn_id=_GCP_CONN_ID)
        processed_files = ti.xcom_pull(task_ids="get_processed_files")

        # Add the newly processed files to the log
        processed_files.extend(new_files)

        # Upload the updated log back to GCS
        processed_files_content = "\n".join(processed_files)
        gcs_hook.upload(
            bucket_name=_GCS_BUCKET_NAME,
            object_name=_PROCESSED_FILES_LOG,
            data=processed_files_content.encode("utf-8"),
        )
        return f"Updated processed files log with {len(new_files)} new files."
    else:
        return "No new files to update."
```

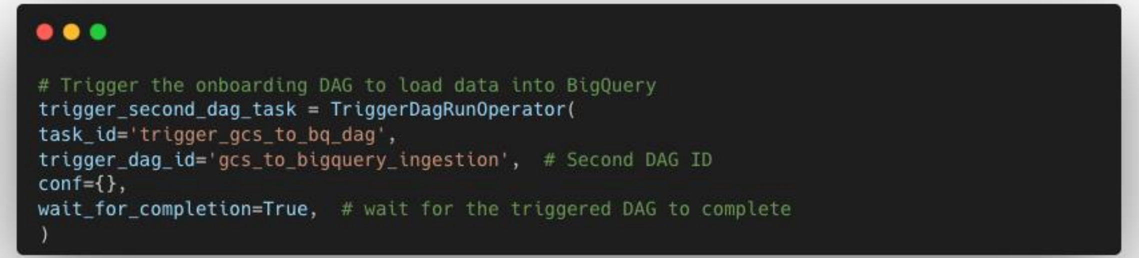
Data transformation – Oops, ERROR!!



```python
# get comic data from a comic number
def get_comic_data(start_num = 0):
    all_comic_data = []
    current_comic_number = get_current_comic_number()
    if current_comic_number is not None:
        for num in range(start_num + 1, current_comic_number + 1):
            url = f"https://xkcd.com/{num}/info.0.json"
            response = requests.get(url)
            try:
                response.raise_for_status()
                comic_data = response.json()
                all_comic_data.append(comic_data)
                logging.info(f"Successfully retrieved data for comic #{num}")
            except requests.exceptions.RequestException as e:
                logging.warning(f"Failed to retrieve data for comic #{num}: {e}")
        return pd.DataFrame(all_comic_data)
    else:
        logging.error("Unable to fetch comic data. Exiting function.")
        return pd.DataFrame()
```

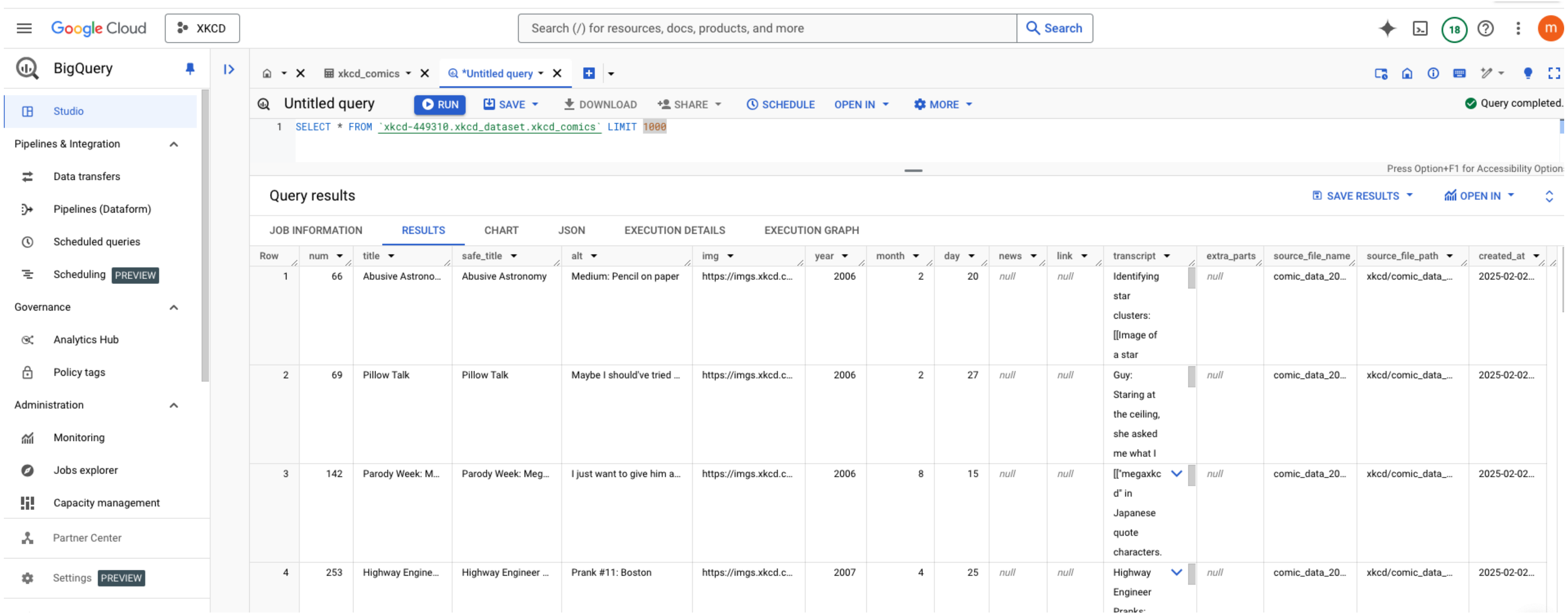


```python
def get_comic_data(start_num = 0):
    all_comic_data = []
    current_comic_number = get_current_comic_number()
    if current_comic_number is not None:
        for num in range(start_num + 1, current_comic_number + 1):
            url = f"https://xkcd.com/{num}/info.0.json"
            response = requests.get(url)
            try:
                response.raise_for_status()
                comic_data = response.json()

                # define the schema
                filtered_data = {
                    "num": comic_data.get("num", None),
                    "title": comic_data.get("title", None),
                    "safe_title": comic_data.get("safe_title", None),
                    "alt": comic_data.get("alt", None),
                    "img": comic_data.get("img", None),
                    "year": comic_data.get("year", None),
                    "month": comic_data.get("month", None),
                    "day": comic_data.get("day", None),
                    "news": comic_data.get("news", None),
                    "link": comic_data.get("link", None),
                    "transcript": comic_data.get("transcript", None),
                    "extra_parts": comic_data.get("extra_parts", None)

                }

                all_comic_data.append(filtered_data)
                logging.info(f"Successfully retrieved data for comic #{num}")
            except requests.exceptions.RequestException as e:
                logging.warning(f"Failed to retrieve data for comic #{num}: {e}")
        return pd.DataFrame(all_comic_data)
    else:
        logging.error("Unable to fetch comic data. Exiting function.")
        return pd.DataFrame()
```



```
# Trigger the onboarding DAG to load data into BigQuery
trigger_second_dag_task = TriggerDagRunOperator(
task_id='trigger_gcs_to_bq_dag',
trigger_dag_id='gcs_to_bigquery_ingestion',  # Second DAG ID
conf={},
wait_for_completion=True,  # wait for the triggered DAG to complete
)
```

Data transformation – Check the raw table in Bigquery



```
SELECT * FROM `xkcd-449310.xkcd_dataset.xkcd_comics` LIMIT 1000
```

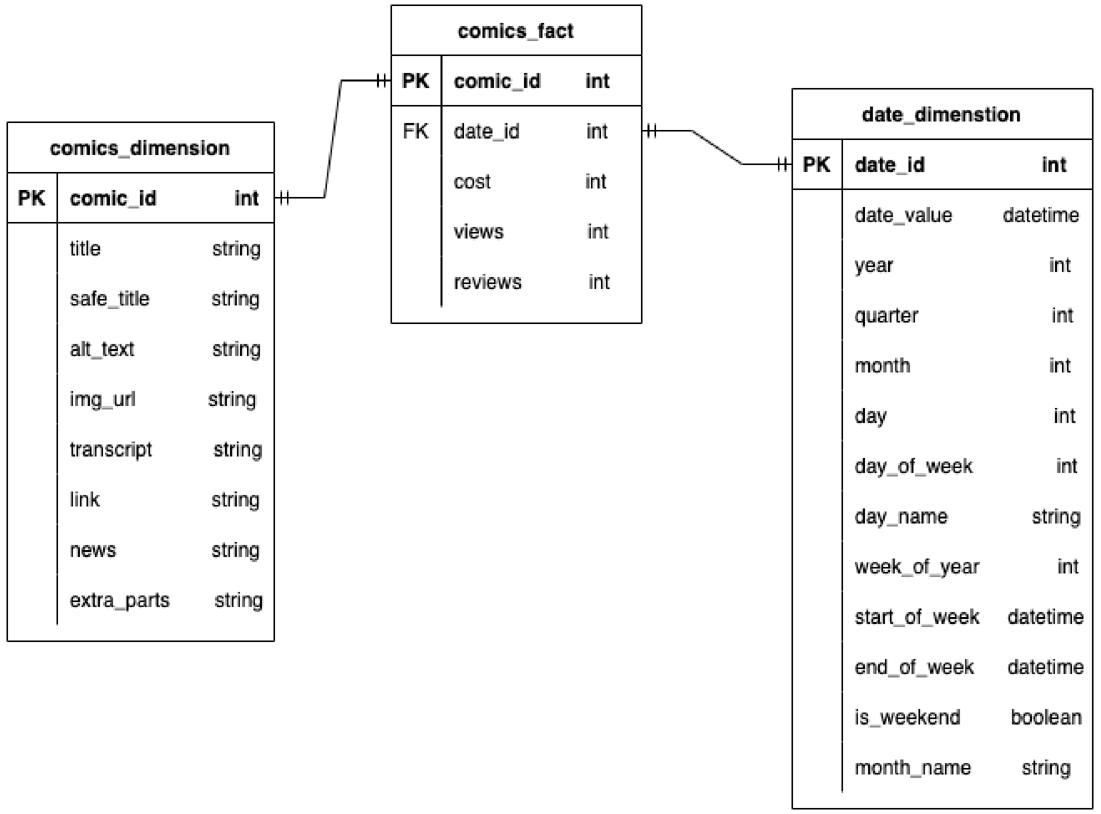
Row	num	title	safe_title	alt	img	year	month	day	news	link	transcript	extra_parts	source_file_name	source_file_path	created_at
1	66	Abusive Astrono...	Abusive Astronomy	Medium: Pencil on paper	https://imgs.xkcd.c...	2006	2	20	null	null	Identifying star clusters: [[Image of a star	null	comic_data_20...	xkcd/comic_data_...	2025-02-02...
2	69	Pillow Talk	Pillow Talk	Maybe I should've tried ...	https://imgs.xkcd.c...	2006	2	27	null	null	Guy: Staring at the ceiling, she asked me what I	null	comic_data_20...	xkcd/comic_data_...	2025-02-02...
3	142	Parody Week: M...	Parody Week: Meg...	I just want to give him a...	https://imgs.xkcd.c...	2006	8	15	null	null	[["megaxkcd" in Japanese quote characters.	null	comic_data_20...	xkcd/comic_data_...	2025-02-02...
4	253	Highway Engine...	Highway Engineer ...	Prank #11: Boston	https://imgs.xkcd.c...	2007	4	25	null	null	Highway Engineer Pranks:	null	comic_data_20...	xkcd/comic_data_...	2025-02-02...

Data transformation – Challenge and Takeaway

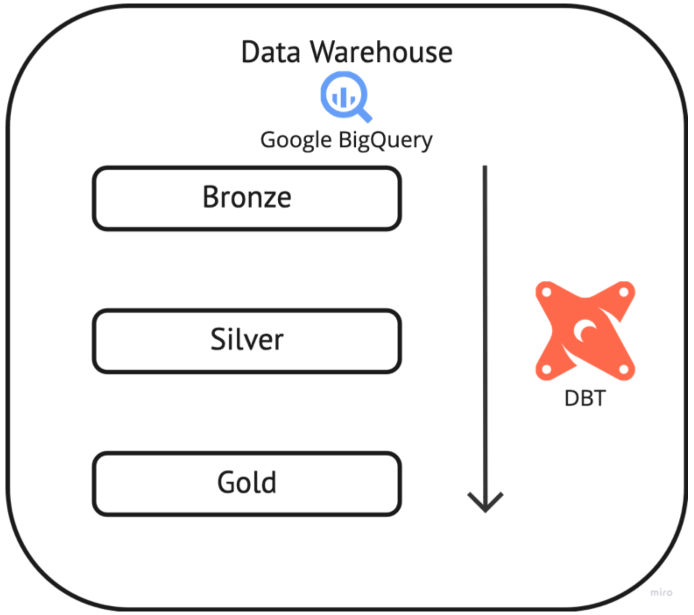
figuring out the correct operator or method

official documentation

Data transformation – Step 2: Get started with DBT – date model



Data transformation – Step 2: Get started with DBT – Medallion structure



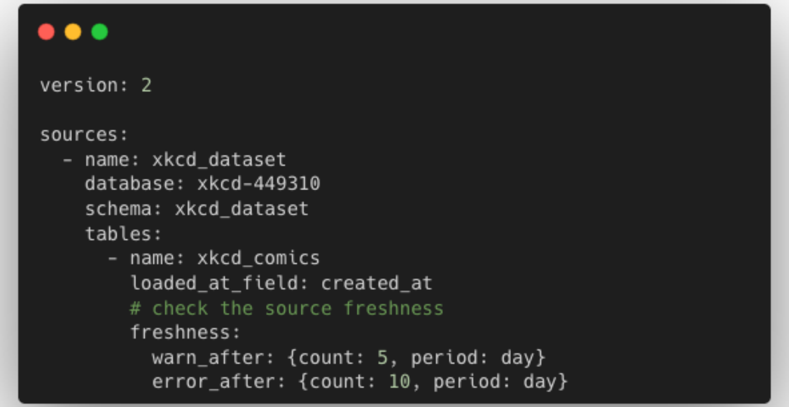
Bronze Layer:
- Raw, unprocessed data
- Changed column names and added basic logging information
- An incremental table with comic_id

Silver Layer:
- Comic Dimension table: Selects all descriptive columns from the bronze comic data; an incremental table
- Comic Fact table: Calculates metrics (cost, views, reviews) and joins with the date dimension table to get the date_id; an incremental table

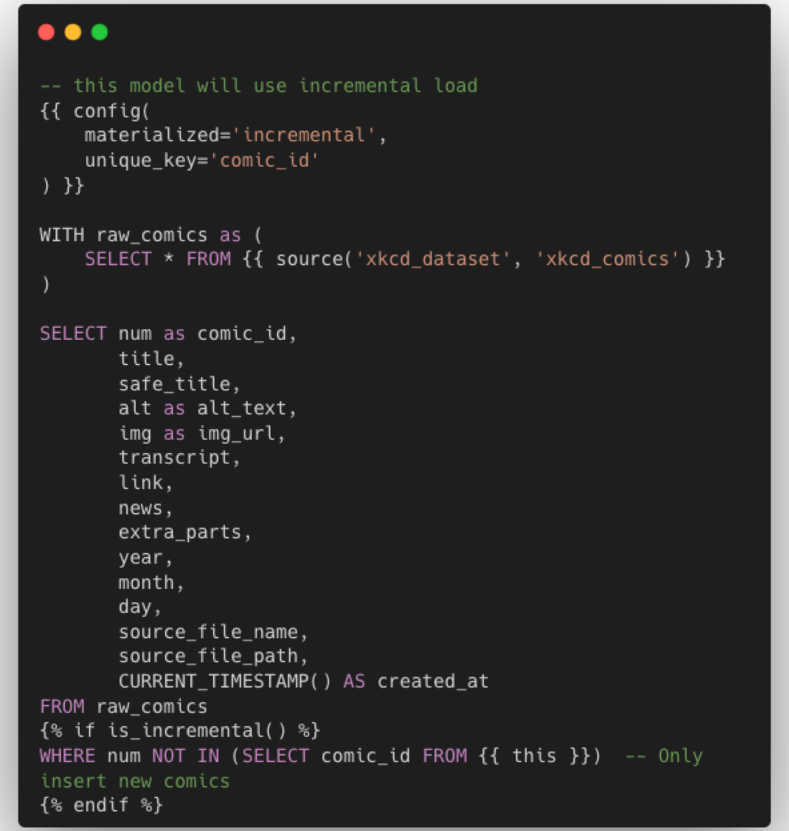
Gold Layer:
- One incremental table with all extended date information from 2006 to the current day

In each layer, an extra timestamp is added

Data transformation – Bronze

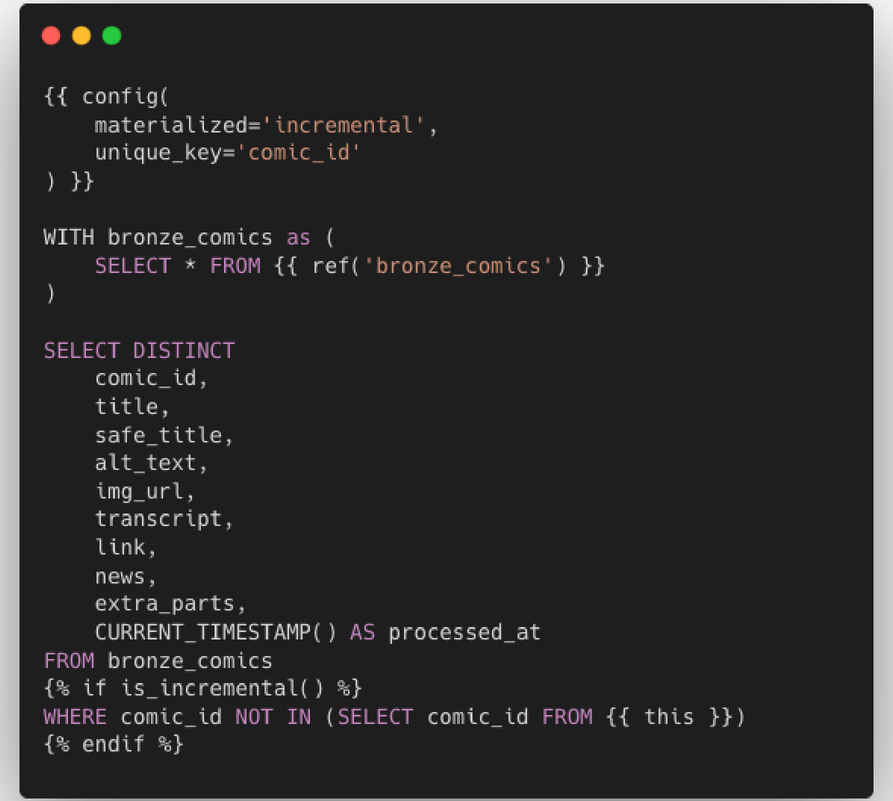


```yaml
version: 2

sources:
  - name: xkcd_dataset
    database: xkcd-449310
    schema: xkcd_dataset
    tables:
      - name: xkcd_comics
        loaded_at_field: created_at
        # check the source freshness
        freshness:
          warn_after: {count: 5, period: day}
          error_after: {count: 10, period: day}
```



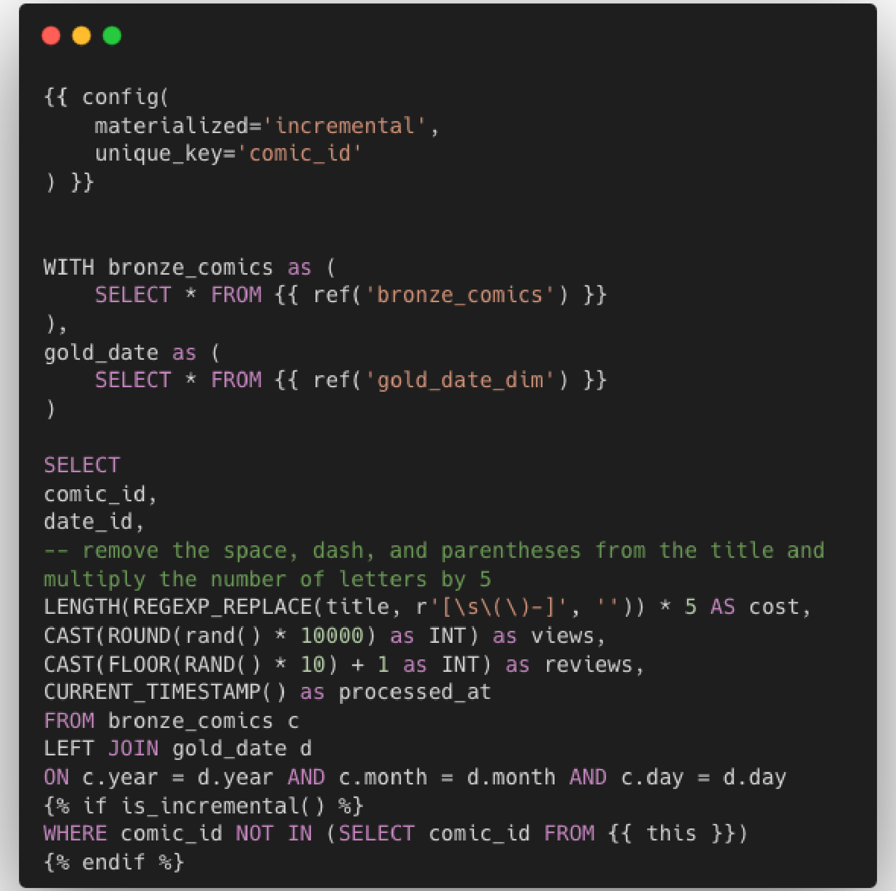
```sql
-- this model will use incremental load
{{ config(
    materialized='incremental',
    unique_key='comic_id'
) }}

WITH raw_comics as (
    SELECT * FROM {{ source('xkcd_dataset', 'xkcd_comics') }}
)

SELECT num as comic_id,
       title,
       safe_title,
       alt as alt_text,
       img as img_url,
       transcript,
       link,
       news,
       extra_parts,
       year,
       month,
       day,
       source_file_name,
       source_file_path,
       CURRENT_TIMESTAMP() AS created_at
FROM raw_comics
{% if is_incremental() %}
WHERE num NOT IN (SELECT comic_id FROM {{ this }})  -- Only
insert new comics
{% endif %}
```

Data transformation – Silver comic dimension



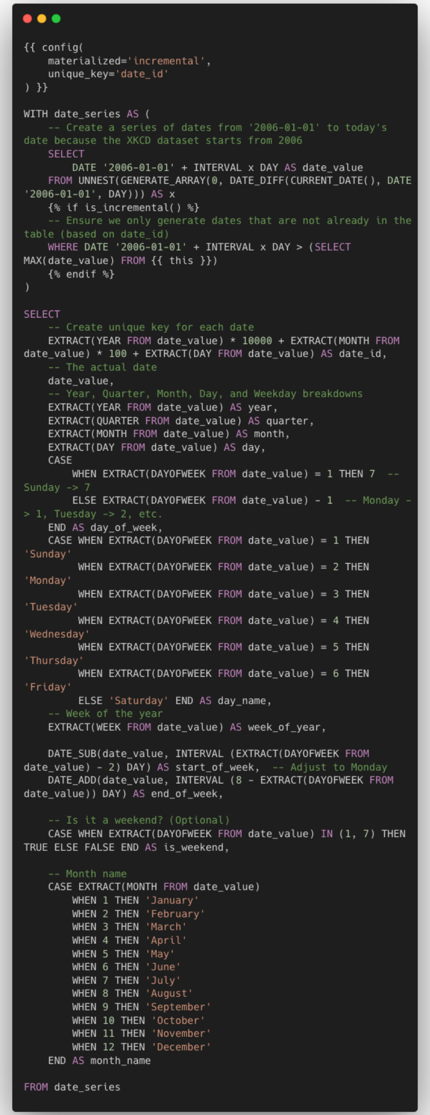
```
{{ config(
    materialized='incremental',
    unique_key='comic_id'
) }}

WITH bronze_comics as (
    SELECT * FROM {{ ref('bronze_comics') }}
)

SELECT DISTINCT
    comic_id,
    title,
    safe_title,
    alt_text,
    img_url,
    transcript,
    link,
    news,
    extra_parts,
    CURRENT_TIMESTAMP() AS processed_at
FROM bronze_comics
{% if is_incremental() %}
WHERE comic_id NOT IN (SELECT comic_id FROM {{ this }})
{% endif %}
```

Data transformation – Silver comic fact



```
{{ config(
    materialized='incremental',
    unique_key='comic_id'
) }}


WITH bronze_comics as (
    SELECT * FROM {{ ref('bronze_comics') }}
),
gold_date as (
    SELECT * FROM {{ ref('gold_date_dim') }}
)

SELECT
comic_id,
date_id,
-- remove the space, dash, and parentheses from the title and
multiply the number of letters by 5
LENGTH(REGEXP_REPLACE(title, r'[\s\(\)-]', '')) * 5 AS cost,
CAST(ROUND(rand() * 10000) as INT) as views,
CAST(FLOOR(RAND() * 10) + 1 as INT) as reviews,
CURRENT_TIMESTAMP() as processed_at
FROM bronze_comics c
LEFT JOIN gold_date d
ON c.year = d.year AND c.month = d.month AND c.day = d.day
{% if is_incremental() %}
WHERE comic_id NOT IN (SELECT comic_id FROM {{ this }})
{% endif %}
```

Data transformation – Gold date dimension

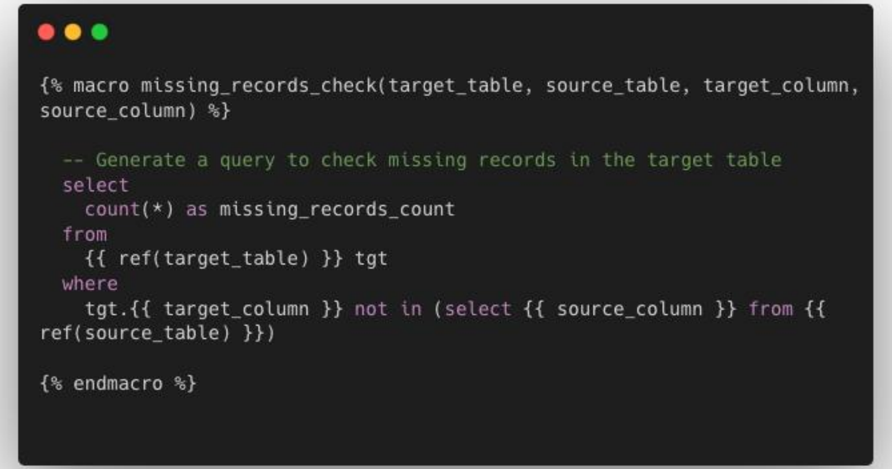


```sql
{{ config(
    materialized='incremental',
    unique_key='date_id'
) }}

WITH date_series AS (
    -- Create a series of dates from '2006-01-01' to today's
date because the XKCD dataset starts from 2006
    SELECT
        DATE '2006-01-01' + INTERVAL x DAY AS date_value
    FROM UNNEST(GENERATE_ARRAY(0, DATE_DIFF(CURRENT_DATE(), DATE
'2006-01-01', DAY))) AS x
    {% if is_incremental() %}
    -- Ensure we only generate dates that are not already in the
table (based on date_id)
    WHERE DATE '2006-01-01' + INTERVAL x DAY > (SELECT
MAX(date_value) FROM {{ this }})
    {% endif %}
)

SELECT
    -- Create unique key for each date
    EXTRACT(YEAR FROM date_value) * 10000 + EXTRACT(MONTH FROM
date_value) * 100 + EXTRACT(DAY FROM date_value) AS date_id,
    -- The actual date
    date_value,
    -- Year, Quarter, Month, Day, and Weekday breakdowns
    EXTRACT(YEAR FROM date_value) AS year,
    EXTRACT(QUARTER FROM date_value) AS quarter,
    EXTRACT(MONTH FROM date_value) AS month,
    EXTRACT(DAY FROM date_value) AS day,
    CASE
        WHEN EXTRACT(DAYOFWEEK FROM date_value) = 1 THEN 7  --
Sunday -> 7
        ELSE EXTRACT(DAYOFWEEK FROM date_value) - 1  -- Monday -
> 1, Tuesday -> 2, etc.
    END AS day_of_week,
    CASE WHEN EXTRACT(DAYOFWEEK FROM date_value) = 1 THEN
'Sunday'
         WHEN EXTRACT(DAYOFWEEK FROM date_value) = 2 THEN
'Monday'
         WHEN EXTRACT(DAYOFWEEK FROM date_value) = 3 THEN
'Tuesday'
         WHEN EXTRACT(DAYOFWEEK FROM date_value) = 4 THEN
'Wednesday'
         WHEN EXTRACT(DAYOFWEEK FROM date_value) = 5 THEN
'Thursday'
         WHEN EXTRACT(DAYOFWEEK FROM date_value) = 6 THEN
'Friday'
         ELSE 'Saturday' END AS day_name,
    -- Week of the year
    EXTRACT(WEEK FROM date_value) AS week_of_year,

    DATE_SUB(date_value, INTERVAL (EXTRACT(DAYOFWEEK FROM
date_value) - 2) DAY) AS start_of_week,  -- Adjust to Monday
    DATE_ADD(date_value, INTERVAL (8 - EXTRACT(DAYOFWEEK FROM
date_value)) DAY) AS end_of_week,

    -- Is it a weekend? (Optional)
    CASE WHEN EXTRACT(DAYOFWEEK FROM date_value) IN (1, 7) THEN
TRUE ELSE FALSE END AS is_weekend,

    -- Month name
    CASE EXTRACT(MONTH FROM date_value)
        WHEN 1 THEN 'January'
        WHEN 2 THEN 'February'
        WHEN 3 THEN 'March'
        WHEN 4 THEN 'April'
        WHEN 5 THEN 'May'
        WHEN 6 THEN 'June'
        WHEN 7 THEN 'July'
        WHEN 8 THEN 'August'
        WHEN 9 THEN 'September'
        WHEN 10 THEN 'October'
        WHEN 11 THEN 'November'
        WHEN 12 THEN 'December'
    END AS month_name

FROM date_series
```

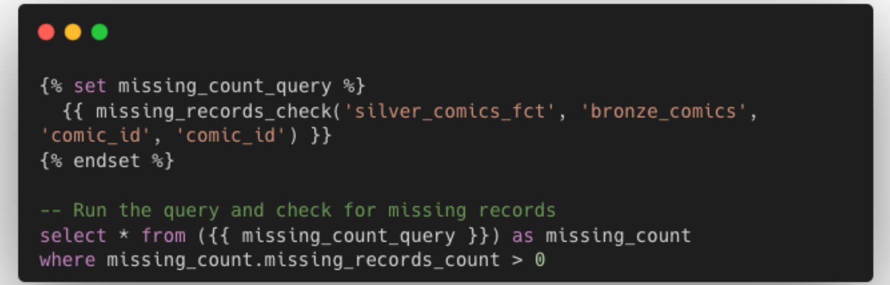
Data transformation – Data quality – built-in tests

Model Name	Column Name	Test Type
bronze_comics	comic_id	unique
		not_null
silver_comics_fct	comic_id	unique
		not_null
	date_id	not_null
	cost	not_null
	views	not_null
	reviews	not_null
silver_comics_dim	comic_id	unique
		not_null
silver_date_dim	date_id	unique
		not_null

Data transformation – Data quality – customed test

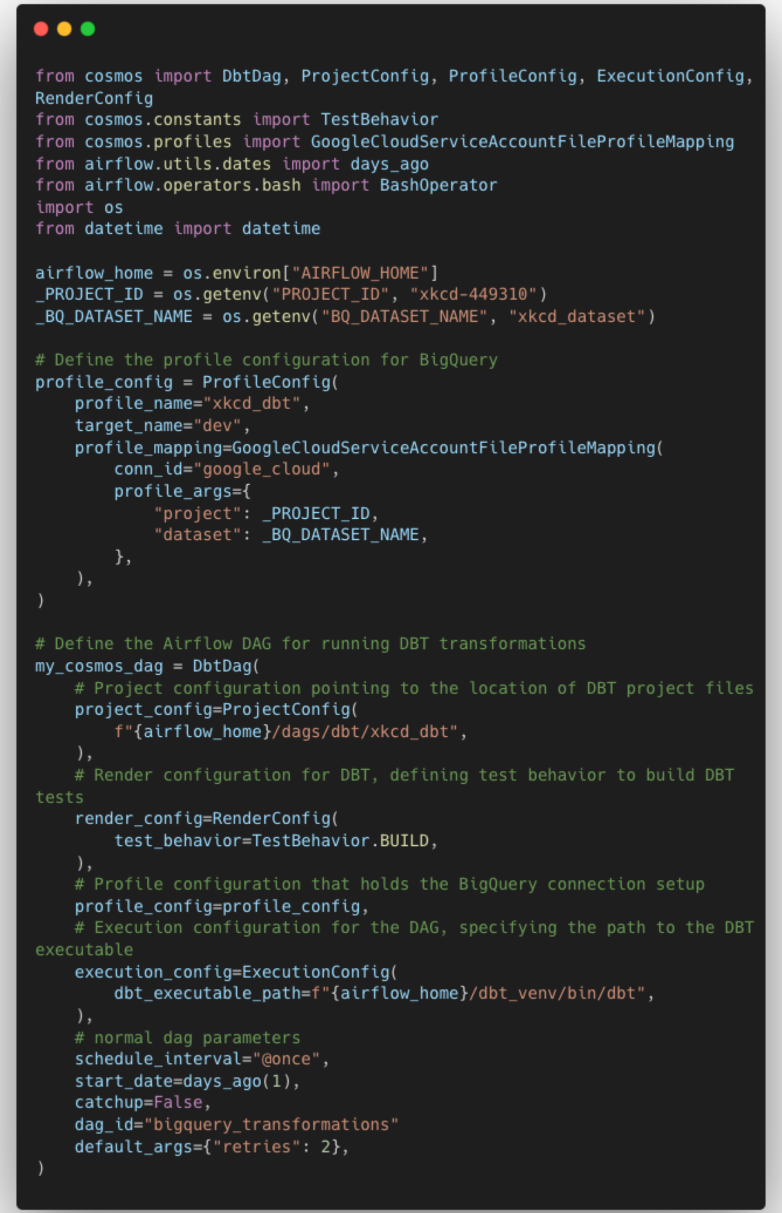


```
{% macro missing_records_check(target_table, source_table, target_column,
source_column) %}

  -- Generate a query to check missing records in the target table
  select
    count(*) as missing_records_count
  from
    {{ ref(target_table) }} tgt
  where
    tgt.{{ target_column }} not in (select {{ source_column }} from {{
ref(source_table) }})

{% endmacro %}
```



```
{% set missing_count_query %}
  {{ missing_records_check('silver_comics_fct', 'bronze_comics',
'comic_id', 'comic_id') }}
{% endset %}

-- Run the query and check for missing records
select * from ({{ missing_count_query }}) as missing_count
where missing_count.missing_records_count > 0
```

Data transformation DAG – Integrate DBT into Airflow using cosmos



```
from cosmos import DbtDag, ProjectConfig, ProfileConfig, ExecutionConfig,
RenderConfig
from cosmos.constants import TestBehavior
from cosmos.profiles import GoogleCloudServiceAccountFileProfileMapping
from airflow.utils.dates import days_ago
from airflow.operators.bash import BashOperator
import os
from datetime import datetime

airflow_home = os.environ["AIRFLOW_HOME"]
_PROJECT_ID = os.getenv("PROJECT_ID", "xkcd-449310")
_BQ_DATASET_NAME = os.getenv("BQ_DATASET_NAME", "xkcd_dataset")

# Define the profile configuration for BigQuery
profile_config = ProfileConfig(
    profile_name="xkcd_dbt",
    target_name="dev",
    profile_mapping=GoogleCloudServiceAccountFileProfileMapping(
        conn_id="google_cloud",
        profile_args={
            "project": _PROJECT_ID,
            "dataset": _BQ_DATASET_NAME,
        },
    ),
)

# Define the Airflow DAG for running DBT transformations
my_cosmos_dag = DbtDag(
    # Project configuration pointing to the location of DBT project files
    project_config=ProjectConfig(
        f"{airflow_home}/dags/dbt/xkcd_dbt",
    ),
    # Render configuration for DBT, defining test behavior to build DBT
tests
    render_config=RenderConfig(
        test_behavior=TestBehavior.BUILD,
    ),
    # Profile configuration that holds the BigQuery connection setup
    profile_config=profile_config,
    # Execution configuration for the DAG, specifying the path to the DBT
executable
    execution_config=ExecutionConfig(
        dbt_executable_path=f"{airflow_home}/dbt_venv/bin/dbt",
    ),
    # normal dag parameters
    schedule_interval="@once",
    start_date=days_ago(1),
    catchup=False,
    dag_id="bigquery_transformations"
    default_args={"retries": 2},
)
```

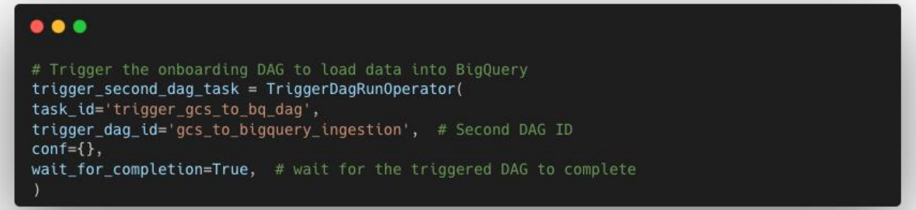
Data transformation – Challenges and Takeaway

macro

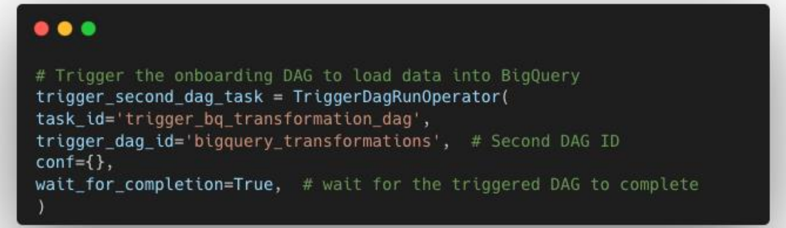
dbt project structure

data quality

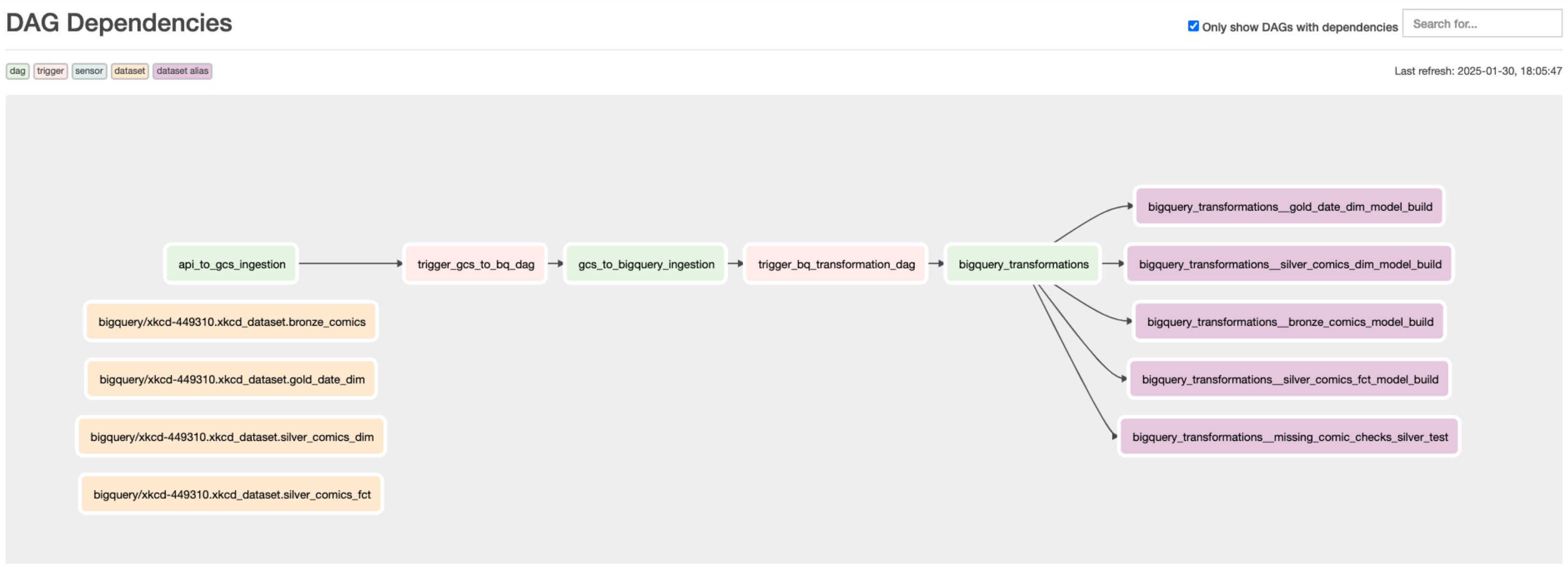
Putting it all together



```
# Trigger the onboarding DAG to load data into BigQuery
trigger_second_dag_task = TriggerDagRunOperator(
task_id='trigger_gcs_to_bq_dag',
trigger_dag_id='gcs_to_bigquery_ingestion',  # Second DAG ID
conf={},
wait_for_completion=True,  # wait for the triggered DAG to complete
)
```



```
# Trigger the onboarding DAG to load data into BigQuery
trigger_second_dag_task = TriggerDagRunOperator(
task_id='trigger_bq_transformation_dag',
trigger_dag_id='bigquery_transformations',  # Second DAG ID
conf={},
wait_for_completion=True,  # wait for the triggered DAG to complete
)
```


DAG Dependencies
Only show DAGs with dependencies
Search for...
dag
trigger
sensor
dataset
dataset alias
Last refresh: 2025-01-30, 18:05:47
api_to_gcs_ingestion
trigger_gcs_to_bq_dag
gcs_to_bigquery_ingestion
trigger_bq_transformation_dag
bigquery_transformations
bigquery_transformations__gold_date_dim_model_build
bigquery_transformations__silver_comics_dim_model_build
bigquery_transformations__bronze_comics_model_build
bigquery_transformations__silver_comics_fct_model_build
bigquery_transformations__missing_comic_checks_silver_test
bigquery/xkcd-449310.xkcd_dataset.bronze_comics
bigquery/xkcd-449310.xkcd_dataset.gold_date_dim
bigquery/xkcd-449310.xkcd_dataset.silver_comics_dim
bigquery/xkcd-449310.xkcd_dataset.silver_comics_fct

What can I do better?

- Error Handling Logic and Logging
- Avoid Simultaneous Task Execution
- Airflow Variables
- Integrate Spark in DBT and Airflow
- Add More Tests (eg. Accepted Values)
- Send Notifications

What should I do to bring it to the productionised setting?

- Security
- CI/CD
- Monitoring and Alerting
- Integrate Spark in DBT and Airflow
- Add More Tests (eg. Accepted Values)

THANK YOU!

Q&A