Data Engineering 2: Big Data Architectures

Streaming, Processing, & Visualization of BikeShare Data

Project Report

Team Members

Mr. Sanath Haritsa (11038004)
Mr. Aniket Ghetla (11038043)
Mr. Vandit Bhalla (11038246)

Dr. Mohammed Abufouda, Dr. Frank Schulz

Applied Data Science and Analytics

ABSTRACT

This data engineering project report presents a comprehensive overview of the development and implementation of a real-time bike sharing data pipeline. The report includes detailed information on the data source, problem statement, user stories, pipeline architecture, pipeline creation steps, challenges encountered, and future scope.

The pipeline is designed to stream live bike share data from an API, process it, store it, and visualize it to aid maintenance workers in identifying defective bikes and docks across various stations, as well as to help customers check the availability of bikes at nearby stations. The implementation leverages a cloud scheduler and cloud functions for periodic API calls, Dataflow for stream processing, and Looker Studio for visualization through interactive dashboards.

The resulting dashboard provides a user-friendly interface that displays the number of defective bikes and docks, along with the availability of bikes at different stations throughout Washington, DC. This project demonstrates the effective use of cloud-based tools for real-time data processing and visualization to enhance the operational efficiency of bike sharing systems.

Table of Contents

1. Introduction

This data engineering project report presents a comprehensive overview of the development and implementation of a real-time bike sharing data pipeline, aimed at enhancing the operational efficiency of bike sharing systems in Washington, DC. The report covers various aspects of the project, including the data source, problem statement, user stories, pipeline architecture, steps to create the pipeline, challenges encountered, and future scope.

The Capital Bikeshare program in Washington, DC, offers publicly available bikes for rent, generating substantial data on bike usage and availability. Efficiently managing this data in real-time is crucial for ensuring the system's reliability and user satisfaction. The primary challenge lies in processing and visualizing this real-time data to assist maintenance workers in identifying defective bikes and docks, and helping customers check bike availability at nearby stations. The project aims to create a robust data pipeline capable of streaming live bike share data from an API, processing it, storing it, and visualizing it in an interactive dashboard. The specific objectives include:

- Developing a system that enables maintenance workers to locate defective bikes and docks across various stations.
- Providing customers with real-time information on bike availability at different stations.

2. Related Work

Stream processing is not a new technology and thus there are already quite a few projects on the same that utilise the tools available on GCP to create a seamless data pipeline. One such project was available on the tech blog website ‘Medium’, the title of the blog is as follows, GCP Cloud Engineering Project - Google Cloud Storage, BigQuery, Functions, Scheduler, Pub/Sub. This project involves integrating a free currency API with BigQuery and handling Parquet files in a Google Cloud Storage (GCS) bucket to address real-life data processing challenges. The first part focuses on fetching exchange rates via a free API, storing this data in a BigQuery table named ‘currency_rate’, and running the application regularly using Google Cloud Functions and Scheduler. The process includes setting up environment variables, importing necessary libraries, and creating the BigQuery table if it doesn't already exist. The application fetches exchange rate data, cleans the response, converts it into a pandas DataFrame, and uploads it to BigQuery. The second part handles daily Parquet files in a GCS bucket, ensuring new files are imported into BigQuery only if they haven't been processed before. The records in these files are processed to select the one with the highest cost for each combination of dt, network, currency, and platform. A ‘cost_usd’ field is added by converting the cost using the latest exchange rate from the ‘currency_rate’ table. The application updates the target table with new records if their

cost is higher than existing ones and deletes records if the corresponding file is missing from the bucket. This comprehensive approach ensures accurate and efficient currency conversion and data management, demonstrating the project's effectiveness in solving real-life data processing issues.

This project helped guide our data flow and further increased our understanding of the GCP tools.

3. Data Source

This project utilizes data from Capital Bikeshare, a publicly available bike rental service in Washington, DC. The data source provides system data in two forms: Trip History Data (Zipped CSVs) and Real-Time System Data. The features of Trip History Data are entirely different from those available in Real-Time System Data. Given the project's focus on stream processing, Real-Time System Data was chosen.

a. APIs

The project utilizes the following APIs to fetch real-time data:

- [GET] https://gbfs.lyft.com/gbfs/2.3/dca-cabi/en/station_information.json
- [GET] https://gbfs.lyft.com/gbfs/2.3/dca-cabi/en/station_status.json

b. Data Features

The Real-Time System Data includes the following features:

- Station ID: Unique identifier for each station.
- Timestamp: The time at which the data was recorded.
- Name: Name of the station.
- Latitude: Geographical latitude of the station.
- Longitude: Geographical longitude of the station.
- Number of Bikes Available: Total number of bikes available at the station.
- Number of E-Bikes Available: Number of electric bikes available at the station.
- Number of Docks Disabled: Number of docks currently disabled at the station.
- Number of Bikes Disabled: Number of bikes currently disabled at the station.

This data provides a comprehensive view of the real-time status of the bike-sharing system, facilitating effective stream processing and real-time decision-making for maintenance and customer service.

4. Methodology

The implementation of the real-time data pipeline for the Capital Bikeshare project follows a structured approach utilizing Google Cloud Platform (GCP) services. The key components and steps involved in the methodology are as follows:

a. Cloud Scheduler for Scheduling Cloud Functions

Cloud Scheduler is used to schedule the execution of Cloud Functions at regular intervals. This ensures that data fetching from the API occurs consistently and reliably.

b. Cloud Functions Trigger

The Cloud Scheduler triggers a Python script hosted in Cloud Functions. This script is responsible for fetching real-time data from the Capital Bikeshare API.

c. Pub/Sub for Triggering and Publishing

The trigger and publishing of events are managed through Google Cloud Pub/Sub. The Python script, once executed, publishes the fetched data to a Pub/Sub topic, facilitating smooth data flow and integration between services.

d. Dataflow to connect Pub/Sub and BigQuery

The data published to Pub/Sub is then sent to Dataflow, a fully managed service for stream and batch data processing. Dataflow processes the incoming data and performs necessary transformations and computations.

e. BigQuery for Basic Aggregation

Processed data from Dataflow is directed to BigQuery, a scalable and fully-managed data warehouse. In BigQuery, basic aggregations and summarizations of the data are performed, making it ready for further analysis.

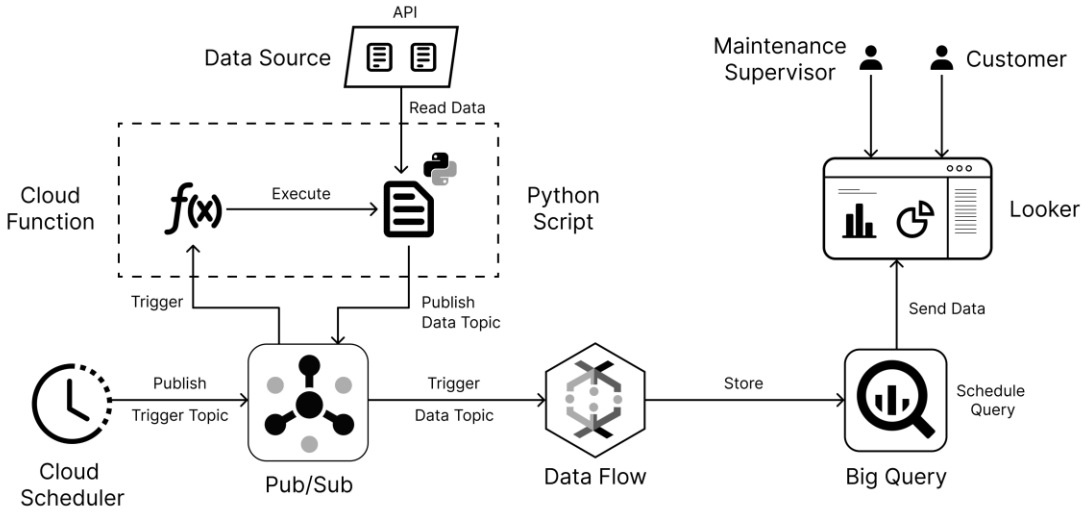
f. Looker Studio for Visualization

The aggregated data in BigQuery is then sent to Looker Studio via a scheduled query. Looker Studio is used to create interactive dashboards that provide insights into the bike-sharing system, such as the number of available bikes and docks.

This methodology ensures a seamless and automated workflow for real-time data collection, processing, aggregation, and visualization, providing valuable insights into the operational status of the Capital Bikeshare system.

5. Data Pipeline

The architecture for the live bike sharing data pipeline is designed to collect, process, store, and visualize real-time data from the Capital Bikeshare system. The pipeline involves several components and steps, each serving a specific function to ensure the seamless flow of data from the source to the end-users.



Architecture for Live Bike Sharing Data Pipeline

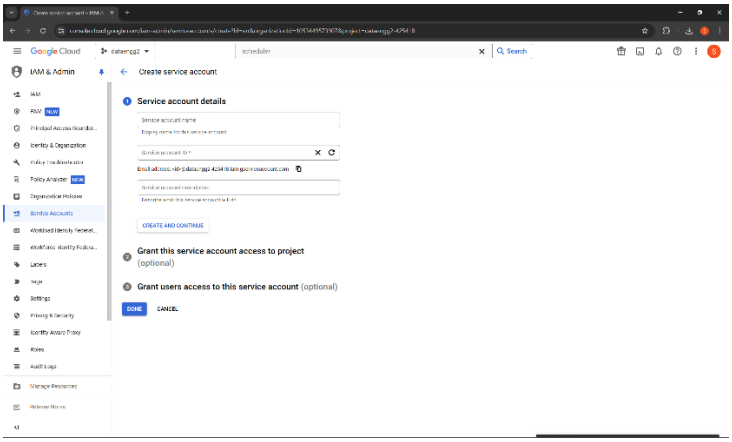
- The process begins with the data source, which is the API provided by Capital Bikeshare. This API supplies real-time data about bike stations.
- The Cloud Scheduler is configured to publish messages at regular intervals to a Pub/Sub topic. This ensures that the data fetching process is triggered consistently.
- Google Cloud Pub/Sub is used to manage the messaging and event-driven architecture. The scheduler's messages are sent to a trigger topic, which in turn triggers the Cloud Function.
- A Cloud Function executes a Python script upon receiving a trigger from Pub/Sub. The Python script fetches real-time data from the API and publishes it to another Pub/Sub topic for data processing.
- The data published to the Pub/Sub data topic is ingested by Google Cloud Dataflow. Dataflow processes and transforms the data, preparing it for storage and analysis.

- Processed data from Dataflow is stored in BigQuery. Basic aggregations and summarizations are performed in BigQuery tables.
- Looker Studio is used to create interactive dashboards. A scheduled query in BigQuery sends the aggregated data to Looker Studio, where it is visualized for end-users. The dashboards are accessible to maintenance supervisors and customers, providing insights into bike availability and system status.

6. Implementation

a. Create a service account with required permissions

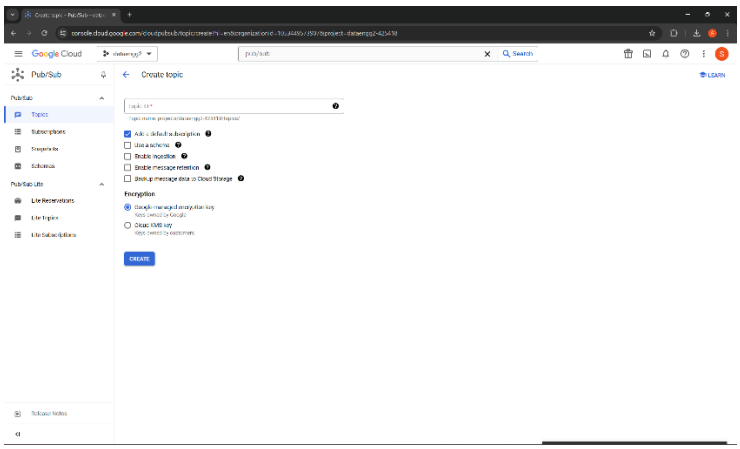
- Click on the hamburger icon on the left side
- Hover over IAM & Admin
- Click on service accounts
- Click on CREATE SERVICE ACCOUNT
- Add the following roles:
    - BigQuery Admin
    - Cloud Functions Admin
    - Cloud Scheduler Admin
    - Storage Admin
    - Dataflow Admin
    - Dataflow Worker
    - Looker Admin
    - Pub/Sub Admin
- Click on Done.
- Follow steps in this link to create service account keys (https://cloud.google.com/iam/docs/keys-create-delete#creating).



b. Create Pub/Sub Topics

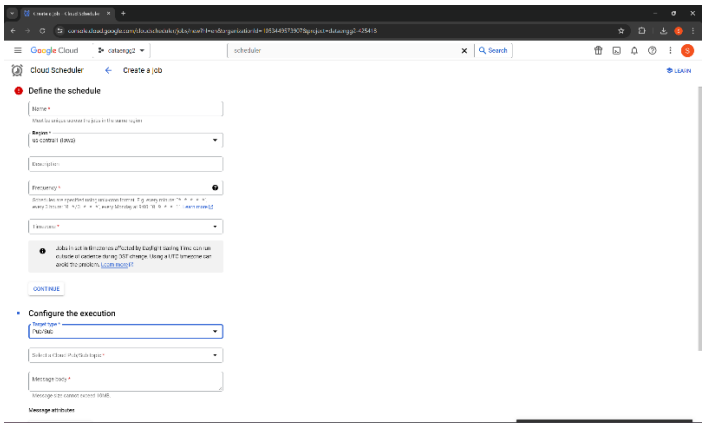
- Search for pub/sub
- Click on CREATE TOPIC

- Provide a name and create a pub/sub topic for trigger
- Provide name and create another topic for data ingestion



c. Create Cloud Scheduler

- Search for cloud scheduler
- Click on CREATE JOB
- Provide a name, select a region, set frequency to (* * * * *) for every minute, select a timezone, and click on continue
- Select target type as pub/sub, select the trigger topic, provide a message body and create the job.



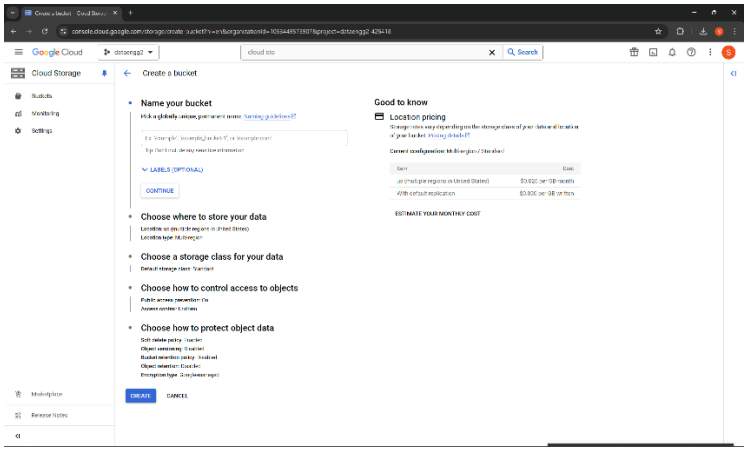
d. Prepare Python Code

- Replace the environ variable with the path of the generated service account key
- Create requirements.txt file
- Zip the following files: main.py, requirements.txt, and the key file

```python
def stream_bikes_data(event, context):
    pubsub_msg = base64.b64decode(event['data']).decode('utf-8')
    print(pubsub_msg)

    os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "dataengg2-425418-cfcf7799ad5b.json"

    station_info_response = requests.get('https://gbfs.lyft.com/gbfs/2.3/dca-cabi/en/station_information.json')
    if station_info_response.status_code == 200:
        station_info = station_info_response.json()['data']['stations']

    station_status_response = requests.get('https://gbfs.lyft.com/gbfs/2.3/dca-cabi/en/station_status.json')
    if station_status_response.status_code == 200:
        station_status = station_status_response.json()['data']['stations']

    station_info = pd.DataFrame(station_info)
    station_status = pd.DataFrame(station_status)

    station_info = station_info[['station_id', 'name', 'lat', 'lon']]

    for i in range(len(station_status)):
        station_status.loc[i, 'num_bikes_available'] = station_status.loc[i, 'vehicle_types_available'][0]['count']
        station_status.loc[i, 'num_ebikes_available'] = station_status.loc[i, 'vehicle_types_available'][1]['count']

    station_status = station_status[['station_id', 'num_bikes_available', 'num_ebikes_available', 'num_docks_disabled', 'num_bikes_disabled']]

    stations_df = station_info.merge(station_status, left_on='station_id', right_on='station_id', how='inner')

    def convert_to_timestamp(row):
        return strftime('%Y-%m-%d %H:%M:%S', localtime(row['timestamp']))

    stations_df['timestamp'] = station_status_response.json()['last_updated']
    stations_df['timestamp'] = stations_df.apply(lambda row: convert_to_timestamp(row), axis = 1)

    publisher = pubsub_v1.PublisherClient()
    topic_name = 'projects/{project_id}/topics/{topic}'.format(
        project_id='dataengg2-425418',
        topic='data_topic',
    )

    data = stations_df.to_dict('records')
    for x in data:
        future = publisher.publish(topic_name, json.dumps(x).encode("utf-8"))
        future.result()
```

e. Create Cloud Storage Bucket

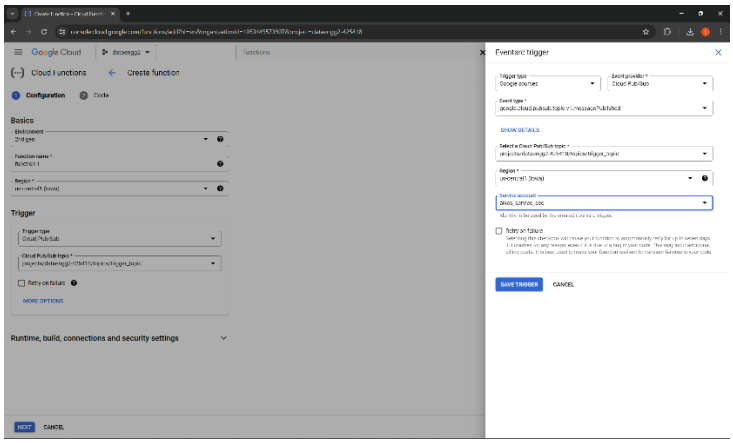
- Search for cloud storage
- Click CREATE
- Provide a name, choose a region and create the bucket



f. Create Cloud Functions

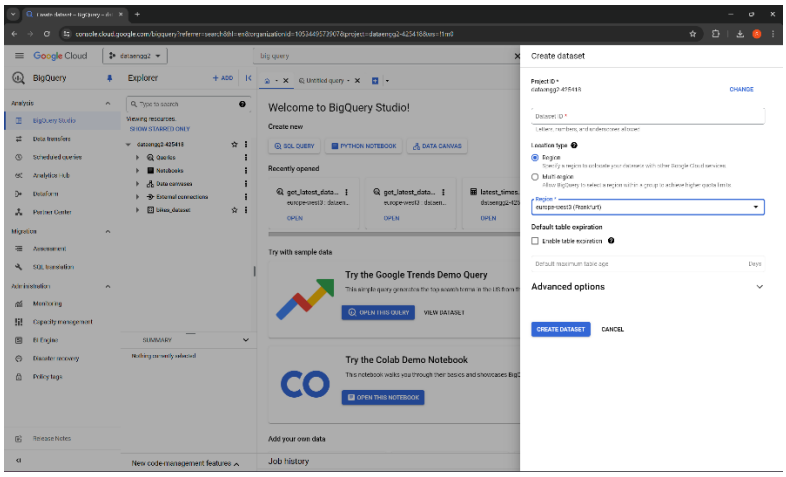
- Search for cloud functions
- Click on CREATE FUNCTION
- Provide a name and select a region.
- Select trigger type as Pub/Sub and select the appropriate trigger topic.
- Click on MORE OPTIONS and select the service account.
- Click NEXT.
- Select runtime -> Python 3.11
- Select Source code -> ZIP upload
- Give the entry point -> stream_bikes_data
- Select the bucket

- Upload the ZIP file with the code and deploy



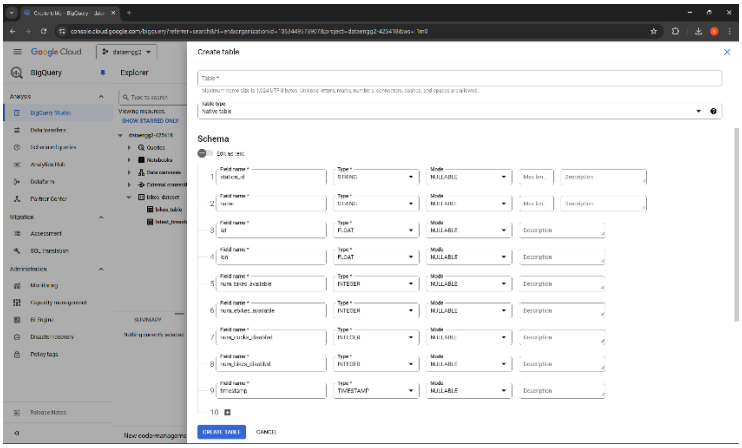
g. Create a BigQuery Dataset

- Search for big query
- Click on the three dots next to the project name
- Select create dataset
- Provide a name, select a region and click on create dataset



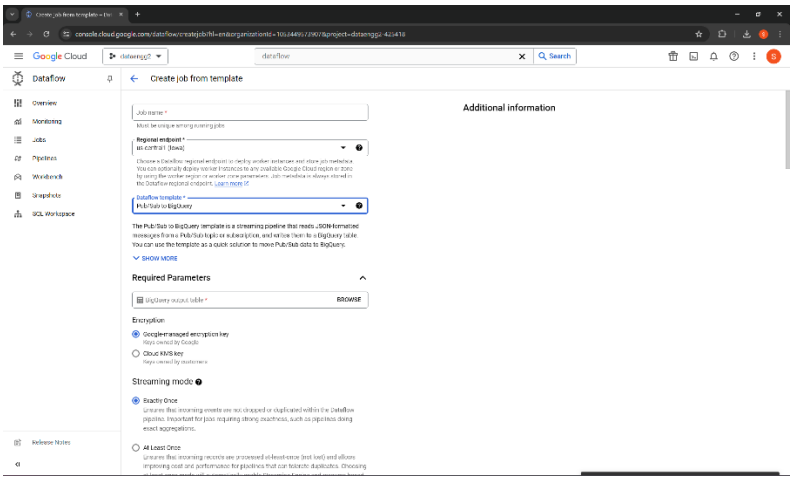
h. Create BigQuery Table

- On the left side panel, click on the three dots next to the created dataset
- Click on create table
- Provide table name
- Create a schema as shown in the image below and create table



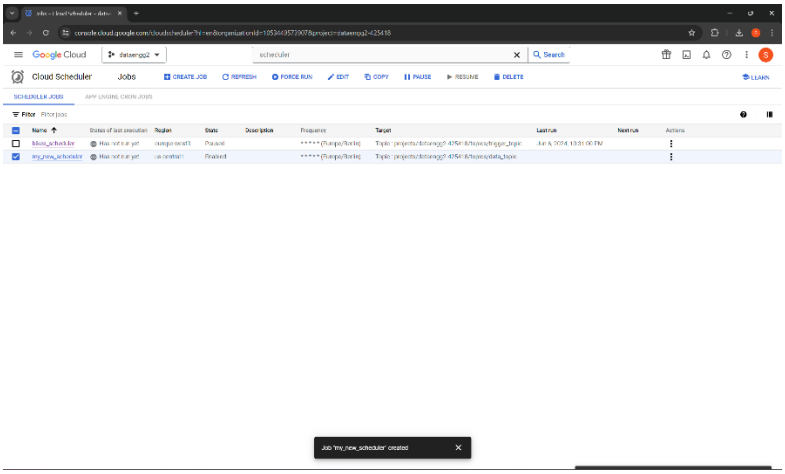
i. Create Dataflow

- Search for dataflow
- Click on CREATE JOB FROM TEMPLATE
- Provide a name, select a region.
- Select dataflow template -> Pub/Sub to Big Query
- Select the created Big Query output table
- Select the Enable Streaming Engine option
- Select Input Pub/Sup topic as the data topic (created in step 2.d.)
- Set Max Workers as 1
- Select service email account
- Under Additional experiments, write -> enable_preflight_validation=false
- Run the job and wait for it to finish



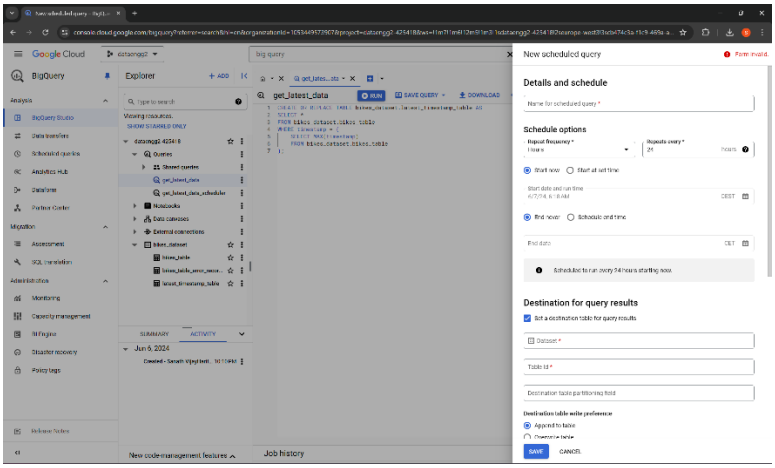
j. Run the Scheduler

- Search for cloud scheduler
- Select the job
- Click on force run



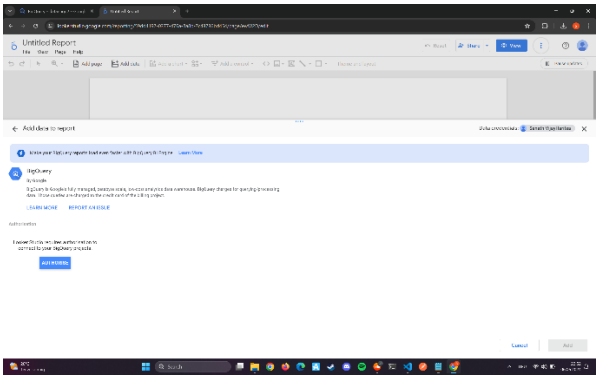
k. Schedule a BigQuery query

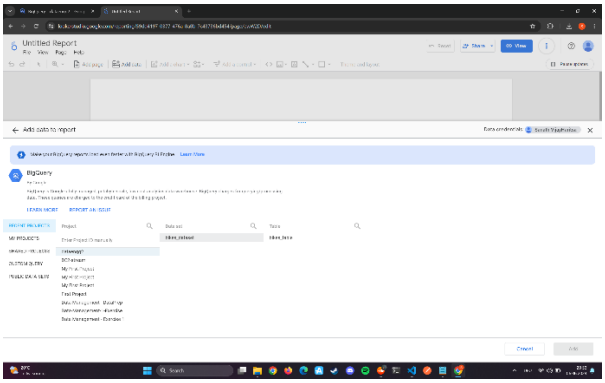
- Search for big query
- Create a new query as shown in the image
- Click on schedule
- Provide a name, select repeat frequency as every 5 minutes
- Select a region, select the service account and save it



l. Looker Studio Setup

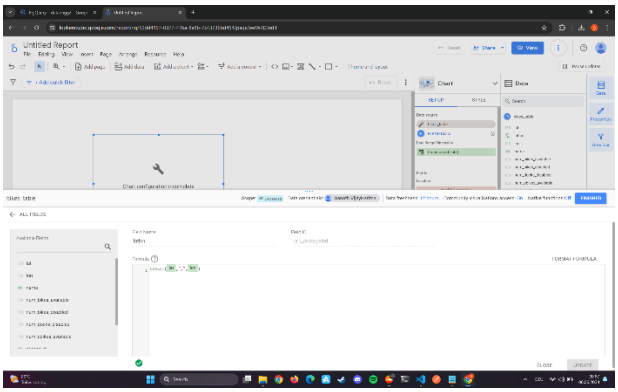
- Open Looker studio
- Authorize BigQuery
- Connect to the dataset

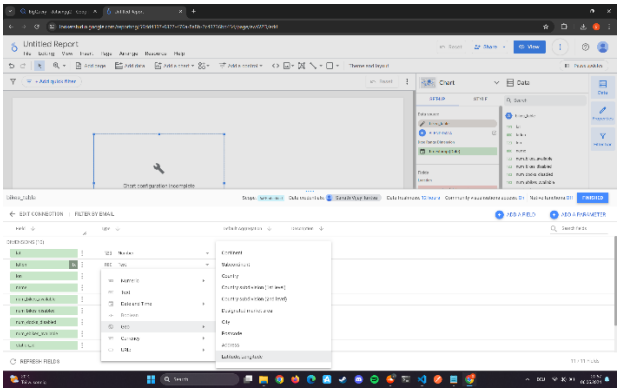




m. Creating Calculated Fields

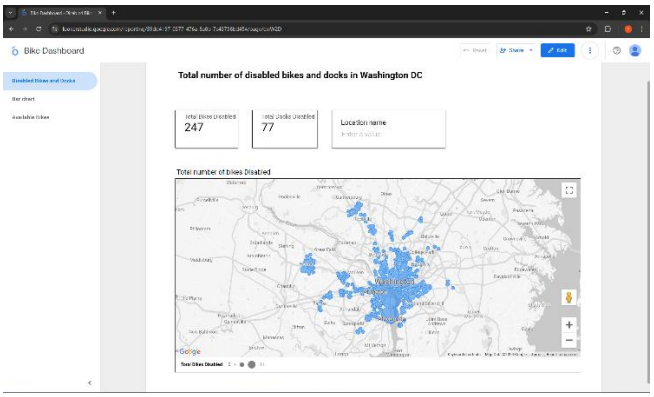
- Create a new calculated field LatLon
- Click add field
- Enter the field name, field ID
- In the formula tab write CONCAT("lat",","lon")
- Click update
- Change the type of the new field from "text to latitude, longitude"

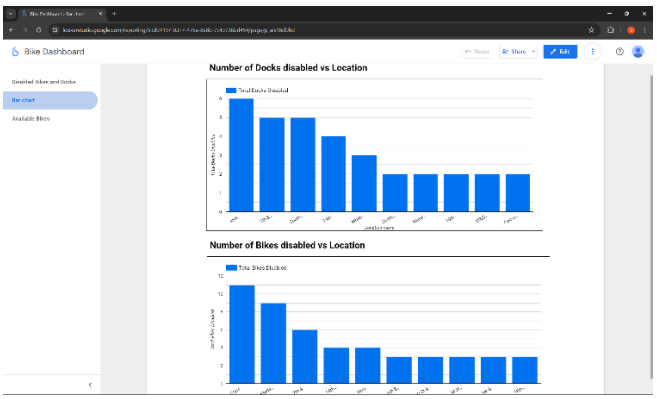


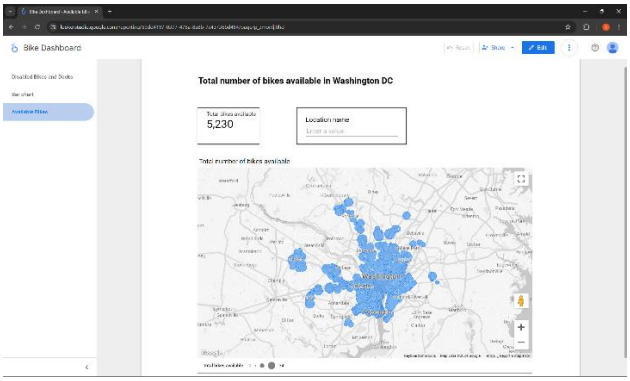


n. Visualization using Charts

- Click on the Add diagram
- Add Google Maps(card with bubbles)
- Add LatLon to "position"
- Add bikes disabled to "size"







7. Problems Faced

Apart from the learning curve of GCP tools, the difficulties that we faced were:

- Discrepancy between the values of certain data points between the two APIs.
- Lack of proper API documentation resulting in increased time needed for understanding.
- GCP Cloud shell was not being provisioned hence had to resort to using GUI for performing all actions.
- Dataflow VM was not getting created, after debugging it was found out that “Europe-West3 (Frankfurt)” region was the issue, thus had to deploy in another region.
- Lack of experience with Looker Studio, thus leading to increased time for dashboard creation.

8. Future Scope

Even though we were able to fulfil our goals there is always room for improvement hence our prospects with this project are:

- Use Apache Beam with Dataflow, currently Big Query only supports batch processing, and its functionality is limited. Using Apache Beam will enable us achieve stream processing and calculate aggregations in real-time.

- Use a visualization tool that gets refreshed automatically as Looker Studio needs to be refreshed manually to see changes in the free version.

9. Results and Conclusions

Using the robust tools available on Google Cloud Platform (GCP), we successfully built a prototype for streaming, processing, and visualizing real-time data. This comprehensive system leverages Cloud Scheduler, Pub/Sub, Cloud Functions, Dataflow, and BigQuery to ensure seamless data flow from the Capital Bikeshare API to the end-user interface. The interactive dashboard created on Looker Studio serves as the project's visual centrepiece. It allows users to easily monitor the number of defective bikes and docks, as well as the availability of bikes at various stations across Washington, DC. This not only enhances operational efficiency but also fulfils the project's user stories by providing valuable insights to maintenance workers and customers. The maintenance workers can quickly identify and address issues, while customers can find available bikes at nearby stations, significantly improving their overall experience. Furthermore, since both applications were implemented successfully, the project can be considered a success.

References

[1] https://capitalbikeshare.com/system-data

[2] https://cloud.google.com/scheduler/docs

[3] https://cloud.google.com/pubsub/docs

[4] https://cloud.google.com/functions/docs

[5] https://docs.python.org/3/

[6] https://pypi.org/project/requests/

[7] https://pandas.pydata.org/docs/

[8] https://cloud.google.com/bigquery/docs

[9] https://cloud.google.com/dataflow/docs

[10] https://developers.google.com/looker-studio

[11] https://medium.com/@dogukannulu/gcp-cloud-engineering-project-part-1-[12] google-cloud-storage-bigquery-functions-scheduler-c2c68b092561

[13] https://www.youtube.com/watch?v=6ahfcLa3oG8

Tasks Distribution

Vandit:

1. Investigation into data sources
2. Generating user stories
3. Identifying key data points to be used
4. Creating and scheduling the query on BigQuery
5. Preparing report and presentation

Aniket:

1. Investigation into visualization tool
2. Connecting Looker Studio to Big Query
3. Creating visualizations to solve the application problems
4. Preparing report and presentation

Sanath:

1. Investigation into data pipeline components
2. Creating and connecting data pipeline components
3. Creating python script for data fetch from API
4. Providing respective content for report and presentation