Architecture:

Extract the files from divvy data → gcs → big query → visualization(Lookeer)
----------prefect----------

Documentation of the Divvy project:

Building an End-to-End Batch Data Engineering Pipeline with Divvy Bike Sharing Data

Welcome to this comprehensive guide where we will walk you through the creation of an end-to-end batch data engineering project. In this project, we will be focusing on processing Divvy monthly bike sharing data. We will perform data extraction, transformation, and loading (ETL) processes and store the resulting data in Google Cloud Storage. The entire pipeline will be designed to facilitate scalable, flexible, and cost-efficient data processing, providing a solid foundation for analysis and further processing.

Project Overview

Our goal is to set up a data pipeline that automates the process of downloading monthly Divvy bike sharing data, extracting relevant information from the downloaded ZIP file, and finally loading the extracted data into Google Cloud Storage. The pipeline will be built using Python and popular libraries like `requests`, `zipfile`, and `prefect` for workflow orchestration. We will also leverage the benefits of Google Cloud Storage for storing the processed data.

Benefits of Google Cloud Storage for Data Lakes

Google Cloud Storage has several advantages when it comes to handling data lakes:

1. **Scalability**: Data lakes are capable of handling massive volumes of data. This capability allows us to store and analyze Divvy bike data as the dataset continues to grow over time.

2. **Flexibility**: Data lakes store data in its raw and unprocessed form. This approach enables different teams to analyze the data according to their unique requirements and apply various processing techniques as needed.

3. **Variety**: Divvy bike data is diverse and comes in various formats such as GPS coordinates, timestamps, and user information. Google Cloud Storage can accommodate this diversity without demanding upfront data structuring.

5. **Data Integration**: In addition to Divvy bike data, we can easily ingest data from other sources like weather and traffic data into the same data lake. This integrated approach allows for comprehensive and holistic analysis.

6. **Machine Learning**: Google Cloud Storage supports machine learning initiatives by enabling data scientists to develop models based on the entirety of the available data.

Project Implementation

Step 1: Data Download

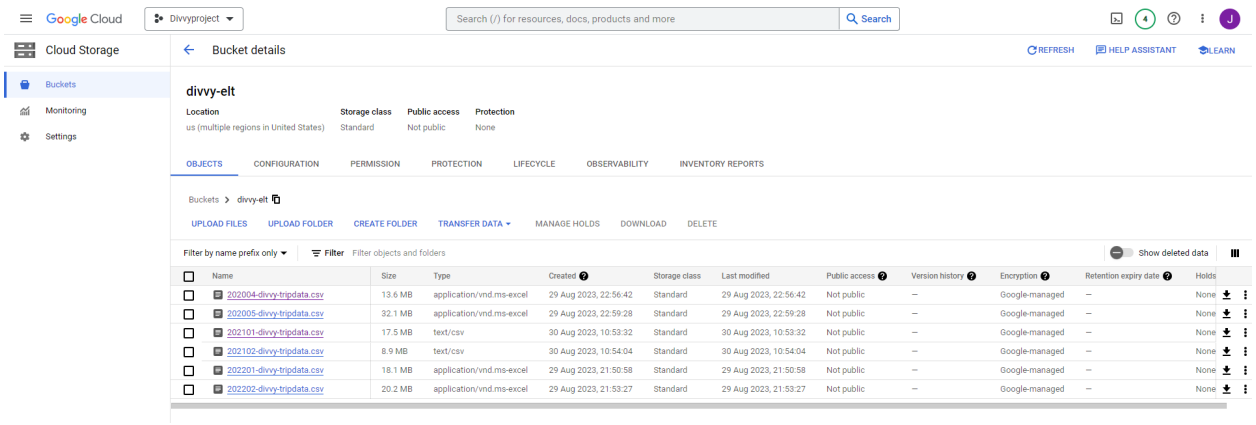
We start by downloading the monthly Divvy bike sharing data in ZIP format. The URL for the data is constructed based on the desired period, and the `requests` library is used to fetch the ZIP file.

Step 2: Data Extraction

Once the ZIP file is downloaded, we extract the relevant CSV file using the `zipfile` library. We ensure that the necessary CSV file is extracted and available for further processing.

Step 3: Data Loading to Google Cloud Storage

After extracting the CSV file, we load it into Google Cloud Storage. This step ensures that the data is securely stored in the cloud, ready for analysis and downstream processing.



The Code
```
import requests
import zipfile
from pathlib import Path
from prefect import flow, task
from prefect_gcp.cloud_storage import GcsBucket


@task(retries=3)
def download_zip_file(url, download_path):
```

```
    os.makedirs(os.path.dirname(download_path), exist_ok=True)

    response = requests.get(url)

    if response.status_code == 200:
        with open(download_path, 'wb') as file:
            file.write(response.content)
        return download_path
    else:
        raise Exception(f"Failed to download the zip file from {url}.")

@task(retries=3)
def extract_csv_from_zip(downloaded_path, zip_extracted_path):
    with zipfile.ZipFile(downloaded_path, "r") as zip_ref:
        csv_file_name = [name for name in zip_ref.namelist() if name.endswith(".csv")]
        if csv_file_name:
            zip_ref.extract(csv_file_name[0], path=zip_extracted_path)
            csv_file_path = os.path.join(zip_extracted_path, csv_file_name[0])
            print(f"CSV file extracted and saved: {csv_file_path}")
        else:
            print("No CSV file found in the zip archive.")

@task()
def load_gcs(extract_csv_path,csv_path) -> None:
    """Upload data to Google cloud storage"""
    gcp_bucket = GcsBucket.load("divvy-gcs")
    gcp_bucket.upload_from_path(from_path=f".\data\{csv_path}", to_path=extract_csv_path)
    return

@flow()
def elt_web_to_gcs() -> None:
    period = 202004
    zip_url = f"https://divvy-tripdata.s3.amazonaws.com/{period}-divvy-tripdata.zip"
    zip_download_path = f".\data\{period}-divvy-tripdata.zip"
    zip_extracted_path = ".\data"
    csv_path = ".\data\{period}-divvy-tripdata.csv"
    downloaded_path = download_zip_file(zip_url, zip_download_path)
    extract_csv_task = extract_csv_from_zip(downloaded_path, zip_extracted_path)
    print(f"Zip file downloaded to: {downloaded_path}")  # Move this line here
    load_gcs(extract_csv_task)

if __name__ == '__main__':
    elt_web_to_gcs()
# The code provided in your post goes here
```

```
if __name__ == '__main__':
    elt_web_to_gcs()
```

Conclusion

Congratulations! You have successfully built an end-to-end batch data engineering pipeline for Divvy bike sharing data. By following this guide, you've learned how to automate the extraction, transformation, and loading of data using Python and `prefect` for workflow orchestration. Storing the data in Google Cloud Storage ensures that your data is well-managed, scalable, and ready for various analyses, including machine learning initiatives.

……

Analyzing Divvy Bike Data with ETL Pipeline: Google Cloud Storage to BigQuery

Welcome to this documentation/blog, where we will explore a practical example of building an Extract, Transform, Load (ETL) pipeline using Python, Google Cloud Storage (GCS), and BigQuery. In this scenario, we'll follow the journey of a data analyst who wants to analyze Divvy bike data. Let's dive in!

Introduction

In the world of data analysis, ETL pipelines play a crucial role in making data accessible, understandable, and actionable. Our case study revolves around a data analyst who needs to analyze Divvy bike data, and we will demonstrate how to design and implement an ETL pipeline to achieve this.

Prerequisites

Before we begin, ensure you have the following prerequisites in place:

1. **Python Environment**: Make sure you have Python installed on your system.

2. **Google Cloud Platform (GCP) Account**: You should have a GCP account set up with appropriate permissions.

3. **Required Libraries**: Install the required Python libraries using `pip`:
   ```
   pip install pandas google-cloud-bigquery prefect prefect-gcp
   ```

The ETL Pipeline

Our ETL pipeline consists of three main stages: Extract, Transform, and Load. Let's break down each stage and understand the code implementation.

Extract

In the extraction stage, we retrieve data from Google Cloud Storage. The code uses the `GcsBucket` class from the `prefect_gcp.cloud_storage` module to fetch the data and save it locally.

```python
@task(retries=3)
def extract_from_gcs(period: int) -> Path:
    gcs_path = f"{period}-divvy-tripdata.csv"
    gcs_bucket = GcsBucket.load("divvy-gcs")
    gcs_bucket.get_directory(from_path=gcs_path, local_path="./gcs_data/")
    return Path(f"./gcs_data/{gcs_path}")
```

Transform

Once we have the data locally, the transformation stage converts the raw CSV file into a pandas DataFrame for analysis.

```python
@task()
def transform(path: Path) -> pd.DataFrame:
    df = pd.read_csv(path)
    return df
```

Load

The final stage involves loading the transformed DataFrame into a BigQuery table. This is accomplished using the Google Cloud client library. The credentials are managed using the `GcpCredentials` class from `prefect_gcp`.

```python
@task()
def load_bq(df: pd.DataFrame) -> None:
    gcp_block = GcpCredentials.load("divvy-gcs-creds")
    df.to_gbq(
        destination_table="divvy.rid",
```

```
        project_id="divvyproject-397220",
        credentials=gcp_block.get_credentials_from_service_account(),
        chunksize=20_000,
        if_exists="replace"
    )
```

ETL Flow

The entire ETL flow is orchestrated using Prefect, a workflow management system. The `etl_gcs_to_bq` function defines the flow, connecting the three stages.

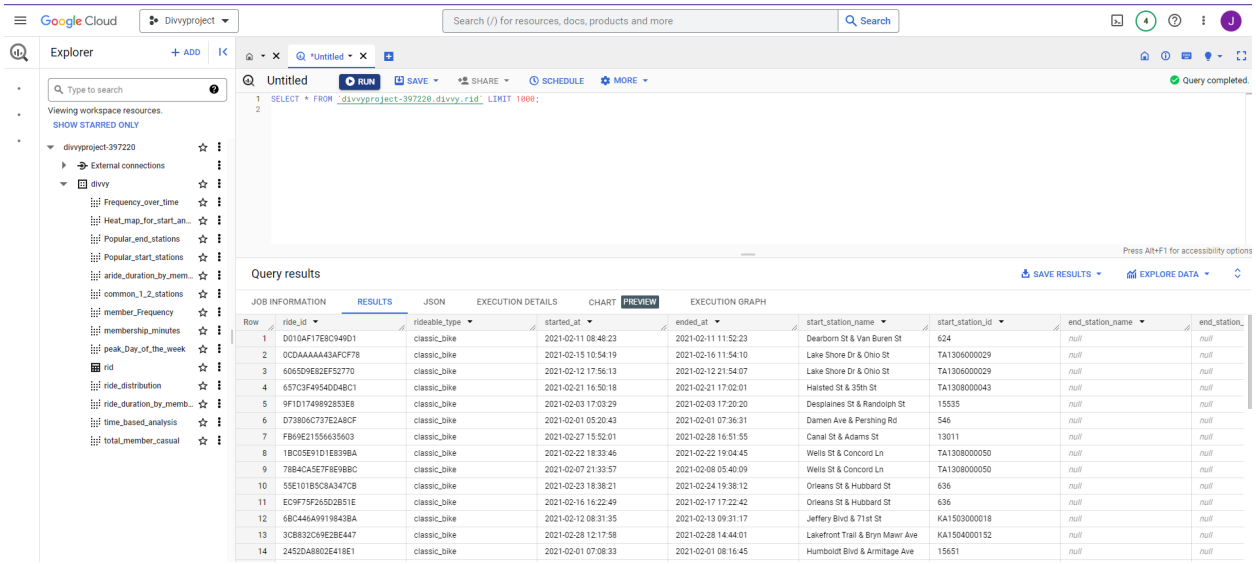
```python
@flow()
def etl_gcs_to_bq():
    period = 202301

    path = extract_from_gcs(period)
    df = transform(path)
    load_bq(df)
```

Running the Pipeline

To run the ETL pipeline, execute the following code snippet:

```python
if __name__ == "__main__":
    etl_gcs_to_bq()
```



Conclusion

In this documentation/blog, we've explored a practical example of building an ETL pipeline to analyze Divvy bike data using Google Cloud Storage and BigQuery. By following the Extract, Transform, Load paradigm, we've demonstrated how to fetch data from GCS, transform it into a DataFrame, and load it into BigQuery for analysis. This pipeline forms the foundation for data analysts to gain insights and make informed decisions.

Happy analyzing!

……

Simplified Data Processing with Parameterized ELT Pipeline and Dockerization

In the world of data engineering, the key to efficient and streamlined processes lies in automation and optimization. Imagine a scenario where a Data Engineer is tasked with handling and processing Divvy bike data for analysis. In this documentation/blog, we will explore how this process can be enhanced to become more automated and manageable. We will delve into parameterization, Dockerization, and visualization to create an end-to-end solution that empowers data-driven decision-making.

Introduction

Our journey begins with the goal of making the data processing pipeline smoother for Data Engineers. We will enhance the existing ETL pipeline by introducing parameterization and Dockerization, taking full advantage of tools like Prefect, Google Cloud Platform, and Looker.

Parameterized ELT Process

The first step towards streamlining the process is parameterization. By parameterizing the ELT (Extract, Load, Transform) pipeline, we enable automation and customization. Data Engineers can simply pass the required parameters to the pipeline, eliminating the need for manual adjustments.

To achieve this, we've incorporated Prefect's subflows, allowing us to pass parameters like the `period` (month) of data we want to process.

Dockerization for Portability

Taking the automation a step further, we've Dockerized the parameterized ELT process. Dockerization involves encapsulating the entire pipeline and its dependencies into a Docker image. This image can be stored in a Docker Hub registry, making it easily accessible.

By creating a Docker image of the pipeline, we achieve portability and consistency. Whenever the pipeline is required, it can be pulled from the Docker Hub and executed with the desired parameters. This eliminates the need to set up complex environments and dependencies each time.

Business Question-driven Views and Looker Integration

Once the data is loaded into BigQuery through our enhanced ELT process, the next challenge is visualization. Addressing this, we've created different views tailored to answer specific business questions. These views are designed with specific analytics objectives in mind, ensuring simplicity and relevance.
Business Questions

1. Ride Distribution Analysis:
  - Analyze the distribution of rideable types (classic bike, electric bike) to see which type is more popular.
  - Create a histogram or bar chart showing the frequency of rides over time (e.g., per day, per week).

2. Ride Duration Analysis:
  - Calculate and visualize the distribution of ride durations (difference between `started_at` and `ended_at`).
  - Compare the average ride duration between different rideable types or member types (member vs. casual).

3. Start and End Stations:
  - Identify the most popular start and end stations based on their usage frequency.
  - Create a heat map to visualize the flow of rides between different station pairs.

4. Member vs. Casual Analysis:
  - Compare the distribution of ride durations between member and casual riders.
  - Analyze whether there are any differences in the most common rideable types between members and casual riders.

5. Time-based Analysis:
  - Analyze the rides' distribution across different times of the day or days of the week.
  - Identify peak hours for rides.

6. Ride Patterns:
  - Identify any recurring patterns or trends in ride start and end locations.
  - Analyze whether certain stations are more commonly used for one-way trips.

7. Ride Duration Trends:
  - Plot the average ride duration over time to identify any trends or changes.

I have integrated Looker, a popular data visualization and exploration platform, with our pipeline. By loading these curated views onto Looker, we facilitate data exploration and reporting. This enables business users to interact with the data and gain insights through intuitive visualizations.
https://lookerstudio.google.com/reporting/0a2d8aab-760c-4f7a-bb56-ba5a13e00ab2

Prefect: The Orchestration Powerhouse

At the heart of our streamlined data pipeline is Prefect, a powerful workflow management system. Prefect allows us to seamlessly orchestrate the parameterized ELT process, Docker image execution, and Looker integration. Its features enable monitoring, retries, and logging, ensuring reliability and traceability in the pipeline execution.

Future Enhancements

In the pursuit of continuous improvement, we envision enhancing the transformation step by incorporating dbt (data build tool). dbt provides advanced transformation capabilities, including modeling, testing, and documentation, contributing to a more comprehensive data processing ecosystem.

Moreover, we plan to automate the infrastructure setup using tools like Terraform. Infrastructure as Code (IaC) principles will be employed to create a reproducible and scalable environment for our data operations.

Conclusion

In this documentation/blog, we've journeyed through the evolution of a data processing pipeline for Divvy bike data analysis. By introducing parameterization, Dockerization, and visualization, we've transformed the process into a seamless, automated, and insightful solution.

Our approach empowers Data Engineers to focus on streamlining the process rather than managing intricate technicalities. With Prefect's orchestration capabilities, Docker's portability, and Looker's visualization prowess, we've crafted a holistic ecosystem for efficient data-driven insights.

As the data landscape continues to evolve, embracing tools and practices like dbt and Terraform will further elevate our capabilities, leading to an even more sophisticated and automated data infrastructure.

Welcome to the era of simplified, empowered data engineering!