Assessments solutions details:

==========

Problem 1 - Data Modeling

Solution:

I am using SQL DDL statements to create tables in PostgreSQL database.

file code-challenge-template/weather_data_load/create_ddl.py have created DDL statements in Postgres database.

If we run above python file (Locally or Deploy to AWS Lambda and schedule with clouwdwatch alarms and event bridge to trigger lambda function) it creates tables in Postgres database, make sure to change connectional details

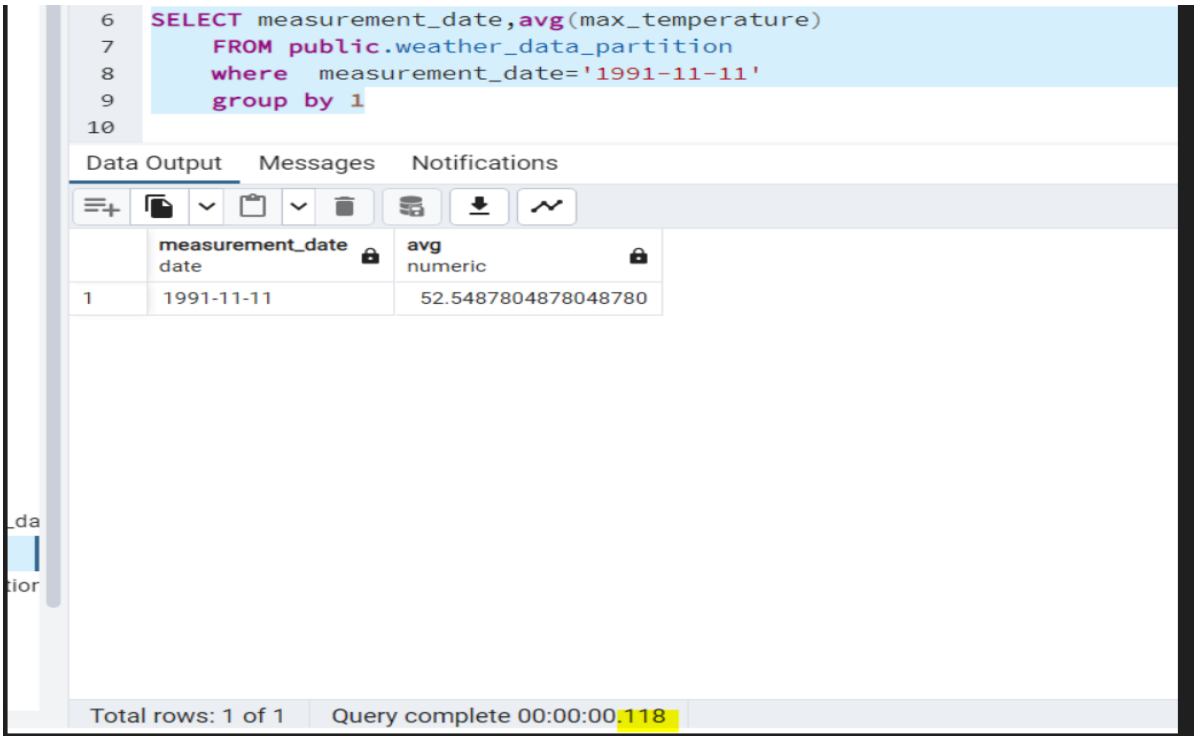
I have created a table WEATHER_DATA, WEATHER_AGGREAGTE_DATA and WEATHER_DATA_PARTITION to check performance while reading/write data to the table and below is my observation.

```
CREATE_WEATHER_DATA_TABLE ='''CREATE TABLE IF NOT EXISTS WEATHER_DATA(
                                weather_station_id VARCHAR(100) NOT NULL,
                                measurement_date DATE NOT NULL,
                                max_temperature INTEGER,
                                min_temperature INTEGER,
                                precipitation INTEGER,
                               primary key (weather_station_id, measurement_date)
                              );
                        '''
cursor.execute(CREATE_WEATHER_DATA_TABLE)

#Creating weather aggregate table
CREATE_WEATHER_AGGREGATE_DATA_TABLE ='''CREATE TABLE IF NOT EXISTS WEATHER_AGGREGATE_DATA(
                                weather_station_id VARCHAR(100) NOT NULL,
                                measurement_year INTEGER NOT NULL,
                                avg_max_temperature DOUBLE PRECISION,
                                avg_min_temperature DOUBLE PRECISION,
                                total_precipitation_cm DOUBLE PRECISION,
                               primary key (weather_station_id, measurement_year)
                              );
                        '''
cursor.execute(CREATE_WEATHER_AGGREGATE_DATA_TABLE)


# Creating table with primary key and range partition
#Doping WEATHER_DATA_PARTITION table if already exists.
# cursor.execute("DROP TABLE IF EXISTS WEATHER_DATA_PARTITION")
CREATE_WEATHER_DATA_PARTITION_TABLE ='''CREATE TABLE IF NOT EXISTS WEATHER_DATA_PARTITION(
                                weather_station_id VARCHAR(100),
                                measurement_date DATE NOT NULL,
                                max_temperature INTEGER ,
                                min_temperature INTEGER,
                                precipitation INTEGER,
                                primary key (weather_station_id, measurement_date)
                              )
                          PARTITION BY RANGE (measurement_date);
                        '''
```

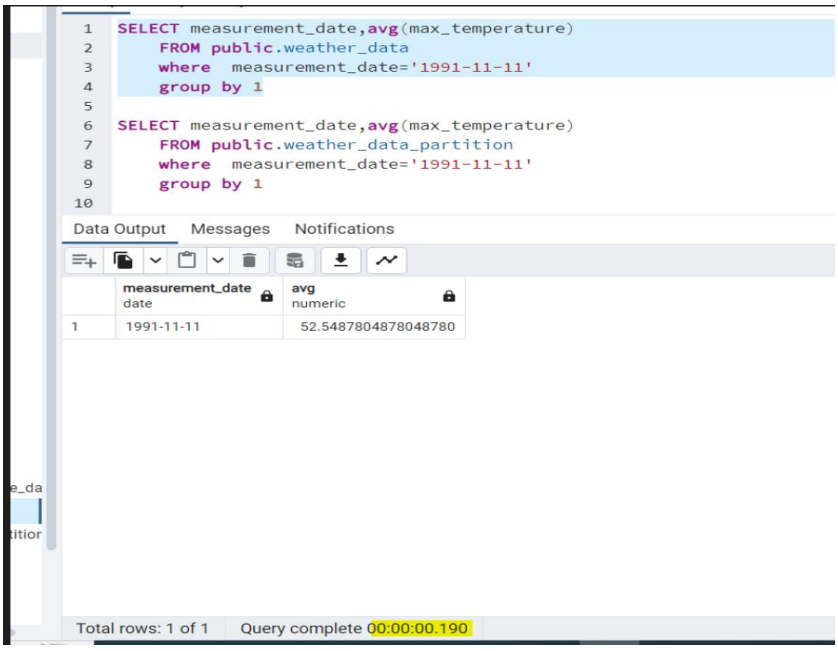
1. Writing to a partition table takes around 260 seconds when compared to non-partition table which is 240 seconds to load all text files data.

2. For analytics doing aggregation below screenshot shows difference in time in milli seconds

Partition table to return results: 118 msec



Non-partition table to return results: 190 msec.



Conclusion: if we frequently read this data and do aggregation with date field/station id better to have partitions in table to get better performance if table grows daily.

==========

Problem 2 - Ingestion:

Write code to ingest the weather data from the raw text files supplied into your database, using the model you designed. Check for duplicates: if your code is run twice, you should not end up with multiple rows with the same data in your database. Your code should also produce log output indicating start and end times and the number of records ingested.

Solution:

In /code-challenge-template/weather_data_load/load_weather_data.py ,

- Method load_data_to_postgress loads data to PostgreSQL table WEATHER_DATA/ WEATHER_DATA_PARTITION
- Method aggregate_data loads data to WEATHER_AGGREAGTE_DATA table

which loads to data to PostgreSQL table , we need to change path to the files and table name to load

I have loaded data to both partition and non-partition table with below code references..

```python
@timing_decorator
def load_data_to_postgress():
    "Fucntion loop through file in a directory and load to postgress tables"
    try:
        all_files = glob.glob(os.path.join(PATH, "*.txt"))
        recordcounter = 0
        for filename in all_files:
            print(f"filename...{filename}")
            with open(filename, 'r', encoding="utf-8") as file:
                for line in file:
                    # Split each line into values (assuming a comma-separated format)
                    values = line.strip().split('\t')
                    weather_station_id = os.path.basename(
                        filename.rsplit(".", 1)[0])
                    # Insert the values into the PostgreSQL table


                    upsert_sql = f"""INSERT INTO {TABLE_NAME} (weather_station_id, measurement_date, max_temperature,min_temperature,precipitation)
                                    VALUES (%s, %s, %s, %s, %s)
                                    ON CONFLICT (measurement_date,weather_station_id)
                                    DO UPDATE
                                    SET max_temperature = EXCLUDED.max_temperature, min_temperature = EXCLUDED.min_temperature,precipitation = EXCLUDED.precipitation;
                                """
                    cursor.execute(upsert_sql, (weather_station_id,
                                    values[0], values[1], values[2], values[3]))
                    inserted_rows = cursor.rowcount
                    recordcounter = inserted_rows + recordcounter
        # Get the number of rows inserted
        print(f"RecordCounter Rows: {recordcounter}")
        # commit
        connection.commit()
    except (Exception, psycopg2.Error) as error:
        print(f"Error: {error}")
        connection.rollback()  # Rollback the transaction if there's an error
```

it can print number of rows processed and time duration with start and end timestamp below

```
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00335315.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00336118.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00336196.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00336600.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00336781.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338313.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338534.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338552.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338769.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338822.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00338830.txt
filename...C:/Users/Bhargava Reddy Yeddu/Downloads/corteva/code-challenge-template/wx_data\USC00339312.txt
RecordCounter Rows: 1729958
Function 'load_data_to_postgress' started at 2023-10-31 15:57:45.
Function 'load_data_to_postgress' ended at 2023-10-31 16:02:25.
Total duration: 279.4793 seconds
Function 'wrapper' started at 2023-10-31 15:57:45.
Function 'wrapper' ended at 2023-10-31 16:02:25.
Total duration: 279.4793 seconds
executing aggregate fucntion
Number of rows insered to aggregate table-----> 4820
Connection closed.
Function 'aggregate_data' started at 2023-10-31 16:02:25.
Function 'aggregate_data' ended at 2023-10-31 16:02:26.
Total duration: 1.2053 seconds
```

if we run the code twice it won't insert duplicates because have added on conflicts on primary key update remaining columns

```
upsert_sql = f"""INSERT INTO {TABLE_NAME} (weather_station_id, measurement_date, max_temperature,min_temperature,precipitation)
            VALUES (%s, %s, %s, %s, %s)
            ON CONFLICT (measurement_date,weather_station_id)
            DO UPDATE
            SET max_temperature = EXCLUDED.max_temperature, min_temperature = EXCLUDED.min_temperature,precipitation = EXCLUDED.precipitation;
        """
```

```
                FROM aggregate_table
            ON CONFLICT (measurement_year,weather_station_id)
                    DO UPDATE
                    SET avg_max_temperature = EXCLUDED.avg_max_temperature,
                        avg_min_temperature = EXCLUDED.avg_min_temperature,
                        total_precipitation = EXCLUDED.total_precipitation;"""
```

Data validation

I have validated the data using /code-challenge-template/src/data_quality_validation.py py files it checks the number of records in all text files and check those records are matching to PostgreSQL table if not it will print in logs below. Also I would like to add quality check to number of records per day and is there any null/bad data in column measurement date etc..

```python
@timing_decorator
def postvalidation():
    """Function to check records which present in text files with postgress tables."""
    all_files = glob.glob(os.path.join(PATH, "*.txt"))
    # Below added -1 because last line is adding to the count even it is ended
    text_file_count = -1
    for filename in all_files:
        with open(filename, 'r', encoding="utf-8") as file:
            for line in file:
                if line != "\t":
                    text_file_count += 1
    print('Number of rows in text file ', text_file_count)
    cursor.execute(f"""select count(*) from {TABLE_NAME} """)
    result = cursor.fetchone()
    table_count = result[0]
    diff_count=text_file_count-table_count
    print(f'Number of rows in table {table_count} \n  count diff {diff_count}')
    if text_file_count != table_count:
        print("Validation failed with count mismatch checking rows which is not present in table , going to print rows below....")
        for filename in all_files:
            with open(filename, 'r', encoding="utf-8") as file:
                for line in file:
                    # Split each line into values (assuming a comma-separated format)
                    values = line.strip().split('\t')
                    # Convert to a datetime object
                    date_obj = datetime.strptime(values[0], '%Y%m%d')
                    # Format the datetime object as yyyy-mm-dd
                    formatted_date = date_obj.strftime('%Y-%m-%d')
                    weather_station_id = os.path.basename(
                        filename.rsplit(".", 1)[0])
                    query_check = f"""   select * from {TABLE_NAME}
                                            where
                                              weather_station_id='{weather_station_id}'
                                              and
                                              measurement_date='{formatted_date}'
                                    """
```

```
C:\Users\Bhargava Reddy Yeddu\Downloads\corteva\code-challenge-template\weather_data_load>python data_quality_validation.py
invoking the fucntion
Number of rows in text file  1729957
Number of rows in table 1729957
  count diff 0
Function 'postvalidation' started at 2023-10-31 16:03:42.
Function 'postvalidation' ended at 2023-10-31 16:03:42.
Total duration: 0.4730 seconds

C:\Users\Bhargava Reddy Yeddu\Downloads\corteva\code-challenge-template\weather_data_load>
```

==========

Problem 3 - Data Analysis

For every year, for every weather station, calculate:

* Average maximum temperature (in degrees Celsius)

* Average minimum temperature (in degrees Celsius)

* Total accumulated precipitation (in centimeters)

Solution: Job took 2 seconds to complete

In python file /code-challenge-template/src/load_weather_data.py method aggregate_data loads data to aggregate to the PostgreSQL table and this aggregate table is created in crate_ddl.py file

Below is code to calculate average maximum temperature, average minimum temperature and total accumulated precipitation in centimeters.

Also not considered missing data when calculating these statistics.

```
agg_query = f"""INSERT INTO {AGG_TABLE_NAME}
        (weather_station_id, measurement_year, avg_max_temperature,avg_min_temperature,total_precipitation_cm)
            WITH aggregate_table AS (
                with total_precipitation as (select weather_station_id, DATE_PART('year',measurement_date) as measurement_year,sum(precipitation)/100 as total_prec
                                            from public.weather_data
                                            where ( precipitation != -9999)
                                            GROUP BY weather_station_id,DATE_PART('year',measurement_date)),
                    avg_max_temperature as (select weather_station_id, DATE_PART('year',measurement_date) as measurement_year,cast(avg(max_temperature)/10 as decir
                                            from public.weather_data
                                            where (max_temperature != -9999 )
                                            GROUP BY weather_station_id,DATE_PART('year',measurement_date)),
                    avg_min_temperature as (select weather_station_id, DATE_PART('year',measurement_date) as measurement_year,cast(avg(min_temperature)/10 as decir
                                            from public.weather_data
                                            where (min_temperature != -9999)
                                            GROUP BY weather_station_id,DATE_PART('year',measurement_date))

                    select tpp.weather_station_id,tpp.measurement_year,
                           tpp.total_precipitation_cm,max_temp.avg_max_temperature,min_temp.avg_min_temperature
                    from  total_precipitation tpp
                    INNER JOIN avg_max_temperature max_temp
                    ON tpp.weather_station_id = max_temp.weather_station_id and tpp.measurement_year = max_temp.measurement_year
                    INNER JOIN avg_min_temperature min_temp
                    ON tpp.weather_station_id = min_temp.weather_station_id and tpp.measurement_year = min_temp.measurement_year
                )
                SELECT
                    weather_station_id,
                    measurement_year,
                    avg_max_temperature,
                    avg_min_temperature,
                    total_precipitation_cm
                FROM aggregate_table
            ON CONFLICT (measurement_year,weather_station_id)
                DO UPDATE
                SET avg_max_temperature = EXCLUDED.avg_max_temperature,
                    avg_min_temperature = EXCLUDED.avg_min_temperature,
                    total_precipitation_cm = EXCLUDED.total_precipitation_cm;"""
```

Below is execution time

```
executing aggregate fucntion
Number of rows insered to aggregate table-----> 4792
Connection closed.
Function 'aggregate_data' started at 2023-10-31 19:53:27.
Function 'aggregate_data' ended at 2023-10-31 19:53:29.
Total duration: 2.0766 seconds
```

Considered -9999 to Null while calculating avg and sum.

==========

Problem 4 - REST API

Choose a web framework (e.g. Flask, Django REST Framework). Create a REST API with the following GET endpoints:

/api/weather

/api/weather/stats

Both endpoints should return a JSON-formatted response with a representation of the ingested/calculated data in your database. Allow clients to filter the response by date and station ID (where present) using the query string. Data should be paginated.

Include a Swagger/OpenAPI endpoint that provides automatic documentation of your API.

Your answer should include all files necessary to run your API locally, along with any unit tests.

Solution:

REST API Created below end points using flask framework.

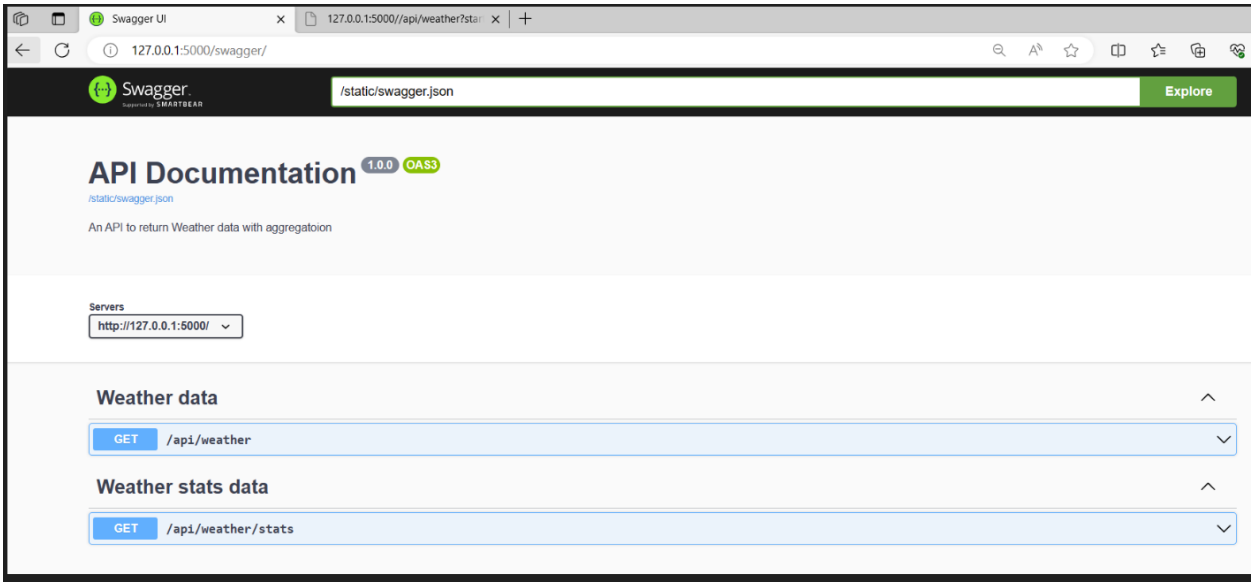
/api/weather
/api/weather/stats

Server.py code-challenge-template/flask_application/server.py which is starting point to the flask app to run

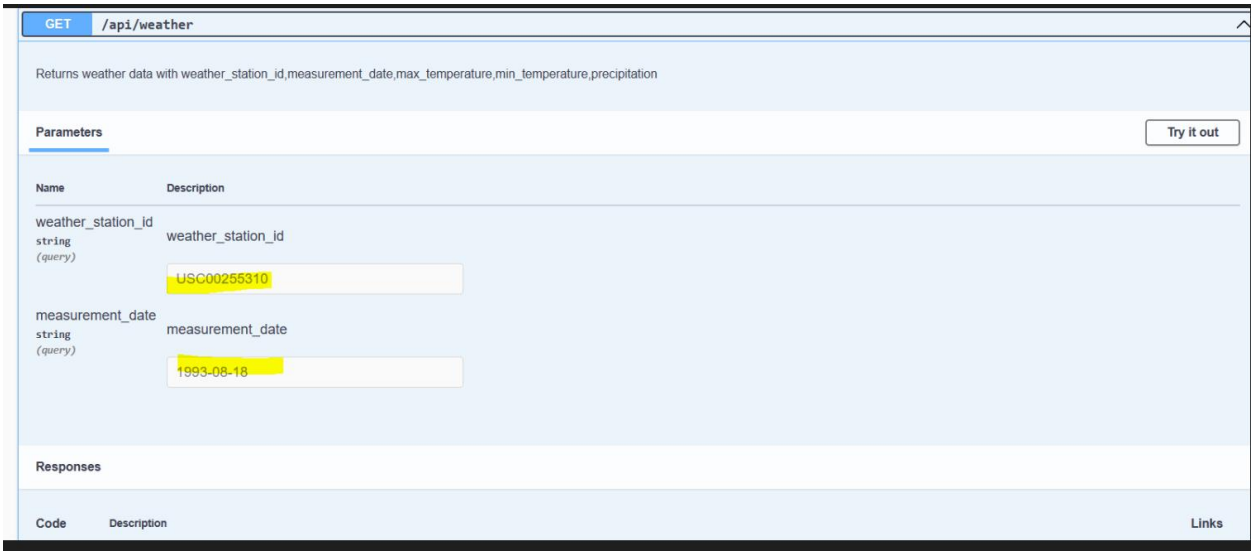
flask_application/template : which holds html render templates

flask_application/static/swagger.json which have swagger json file for api detailed documentation.

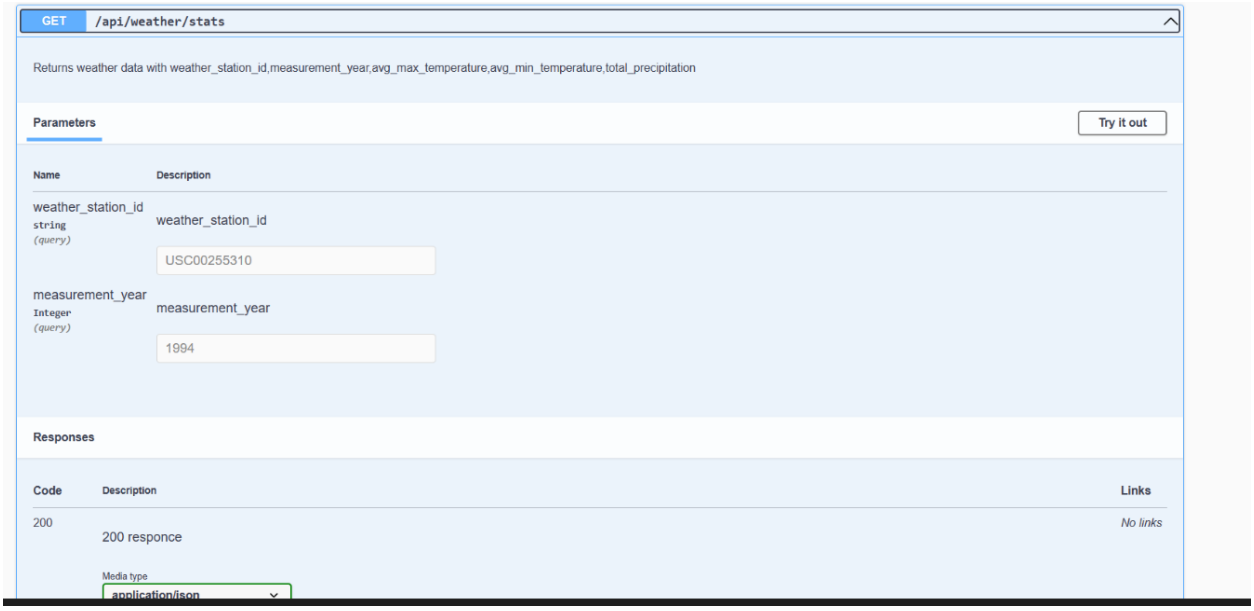
Choosen Flask framework in local and created end points /api/weather and /api/weather/stats



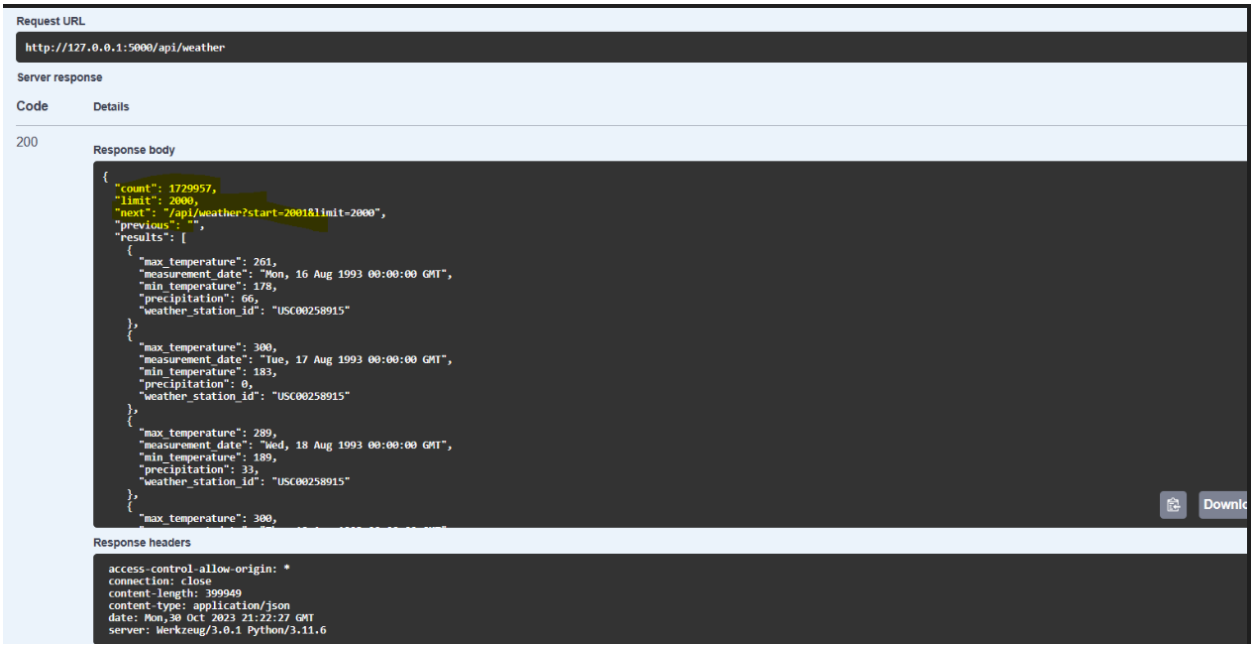
Both end points allow to filter data with station id and measurement date/years

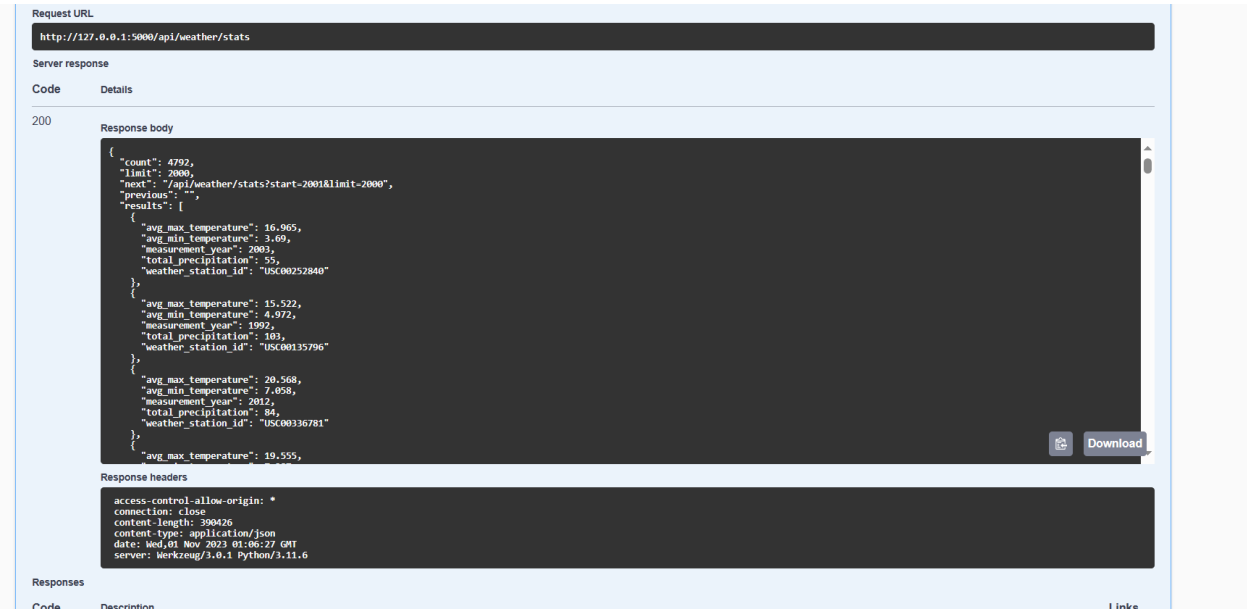


Stats end point with filter option.



Both end points return data in JSON format with Paginated





Below show both end point filtered with station id or/and date data in JSON format with Paginated

Curl

```
curl -X 'GET' \ 'http://127.0.0.1:5000/api/weather?weather_station_id=USC00255310&measurement_date=1993-08-18' \ -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/api/weather?weather_station_id=USC00255310&measurement_date=1993-08-18
```

Server response

Code	Details
200	Response body

```
{
  "count": 1,
  "limit": 2000,
  "next": "",
  "previous": "",
  "results": [
    {
      "max_temperature": 344,
      "measurement_date": "Wed, 18 Aug 1993 00:00:00 GMT",
      "min_temperature": 206,
      "precipitation": 28,
      "weather_station_id": "USC00255310"
    }
  ],
  "start": 1
}
```

Download

Response headers

```
access-control-allow-origin: *
connection: close
content-length: 300
content-type: application/json
date: Wed,01 Nov 2023 01:06:58 GMT
server: Werkzeug/3.0.1 Python/3.11.6
```

Curl

```
curl -X 'GET' \ 'http://127.0.0.1:5000/api/weather/stats?weather_station_id=USC00110072&measurement_year=1985' \ -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:5000/api/weather/stats?weather_station_id=USC00110072&measurement_year=1985
```

Server response

Code	Details
200	Response body

```
{
  "count": 1,
  "limit": 2000,
  "next": "",
  "previous": "",
  "results": [
    {
      "avg_max_temperature": 15.335,
      "avg_min_temperature": 4.326,
      "measurement_year": 1985,
      "total_precipitation": 78,
      "weather_station_id": "USC00110072"
    }
  ],
  "start": 1
}
```

Download

Response headers

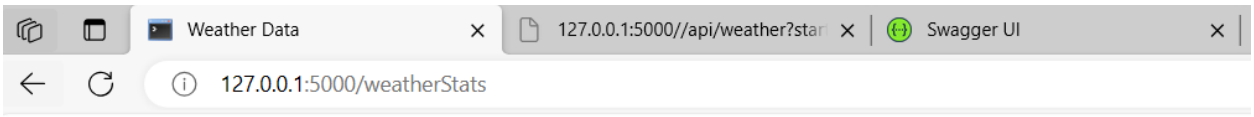
```
access-control-allow-origin: *
connection: close
content-length: 294
content-type: application/json
date: Wed,01 Nov 2023 01:05:28 GMT
server: Werkzeug/3.0.1 Python/3.11.6
```

Responses

Code	Description	Links
200		*No links*

Additionally :

Created UI to perform same above steps in UI



Weather stats Data to filter station id and measurement_year

Weather Station ID:

Measurement Date:

Submit

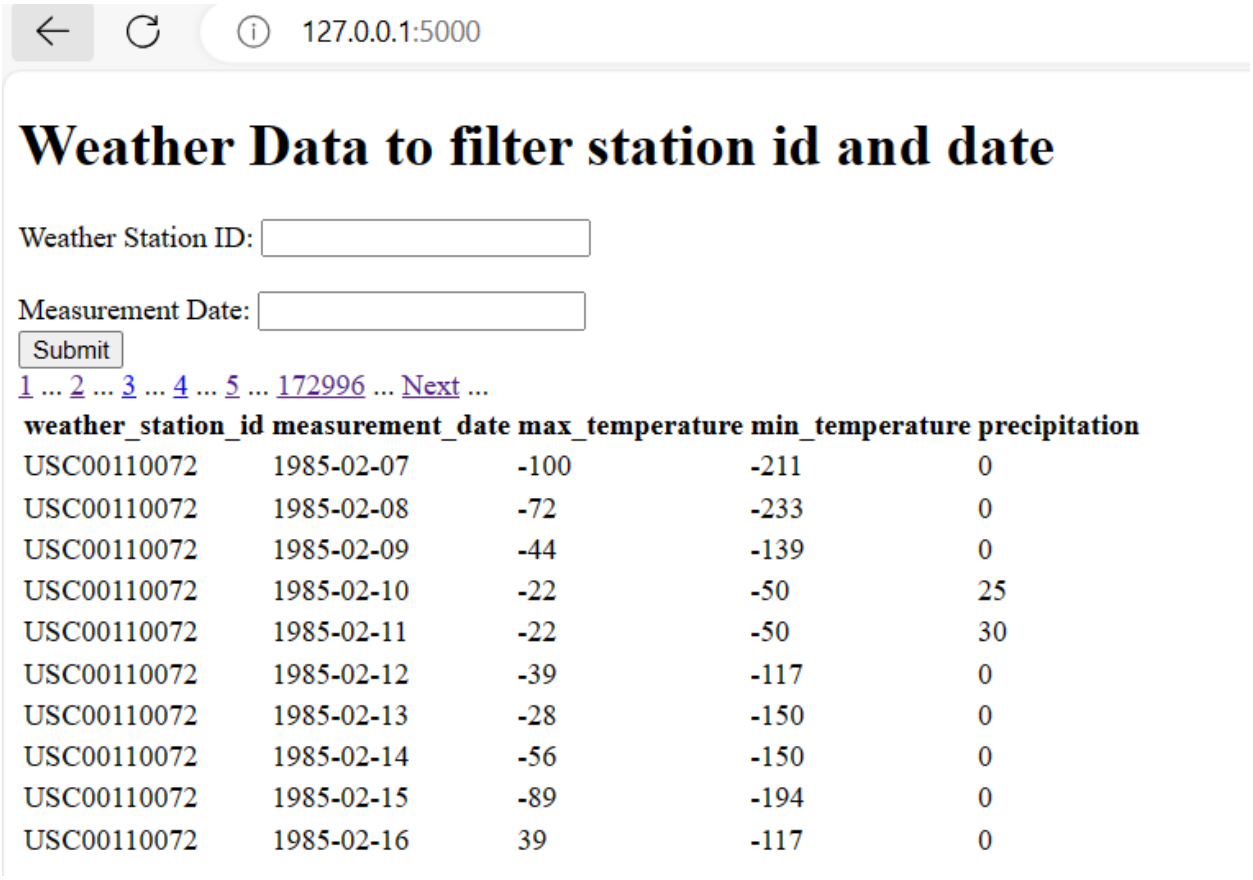
1 ... 2 ... 3 ... 4 ... 5 ... ... 482 ... Next ...

weather_station_id	measurement_year	avg_max_temperature	avg_min_temperature	total_precipitation
USC00338313	2012	None	None	558
USC00118740	2013	161.73972602739727	51.06849315068493	8954
USC00250435	1990	185.43561643835616	59.71780821917808	6240
USC00258480	2004	182.006309148265	59.16403785488959	7334
USC00335297	2011	165.71625344352617	52.00275482093664	13287
USC00132724	1995	125.8932584269663	18.31179775280899	6992
USC00121747	1997	166.375	57.401114206128135	10734
USC00128036	2006	190.29041095890412	64.6	17268
USC00254440	2013	159.60821917808218	-0.8054794520547945	3654
USC00126001	2012	211.61834319526628	103.30835734870317	6861

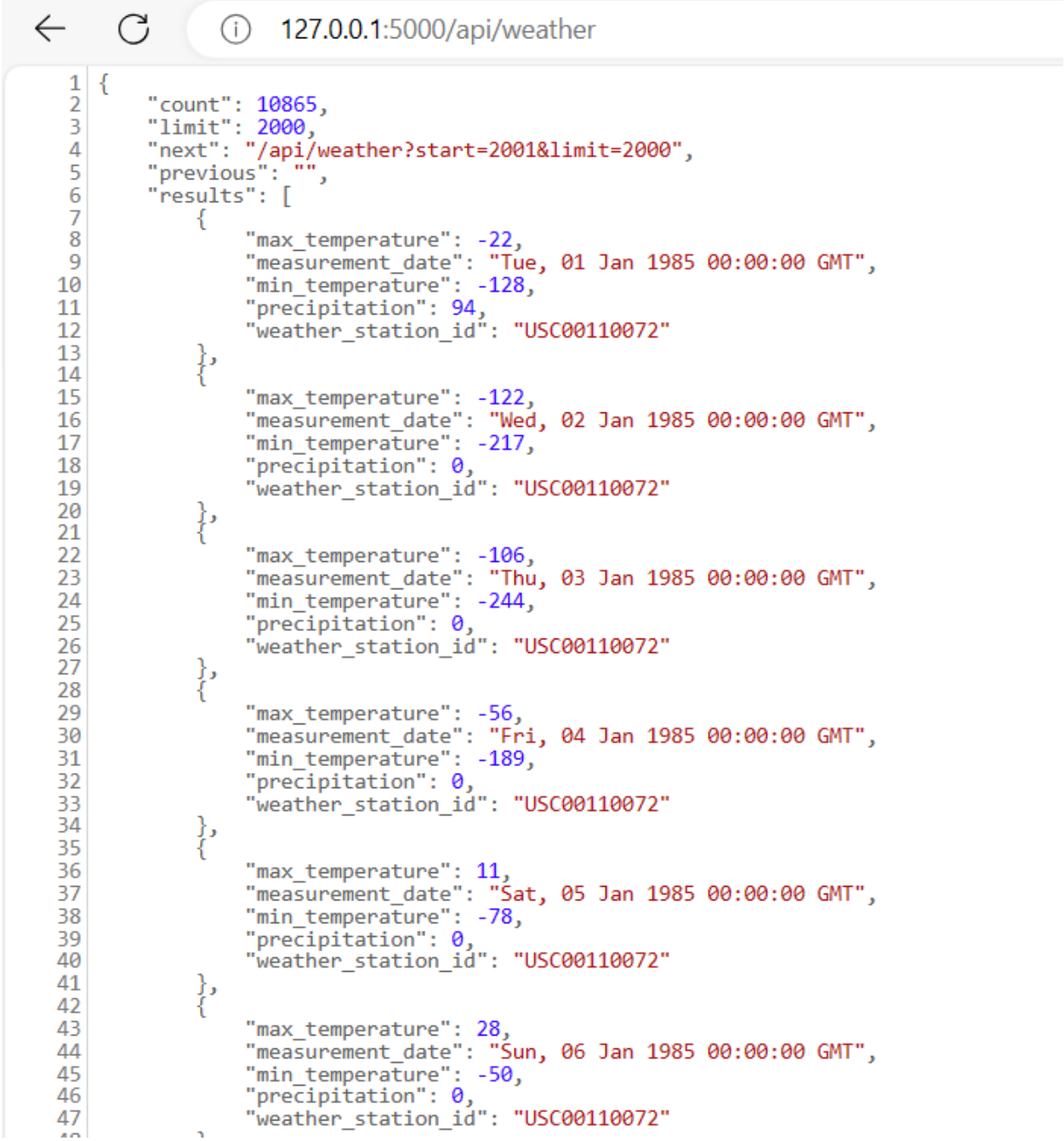
Below I filtered only station id and it show data in JSON format wit Paginated


```
{
    "count": 30,
    "limit": 2000,
    "next": "",
    "previous": "",
    "results": [
        {
            "avg_max_temperature": 14.345,
            "avg_min_temperature": 2.911,
            "measurement_year": 2008,
            "total_precipitation": 127,
            "weather_station_id": "USC00110072"
        },
        {
            "avg_max_temperature": 13.96,
            "avg_min_temperature": 4.745,
            "measurement_year": 1993,
            "total_precipitation": 142,
            "weather_station_id": "USC00110072"
        },
        {
            "avg_max_temperature": 15,
            "avg_min_temperature": 3.667,
            "measurement_year": 1995,
            "total_precipitation": 95,
            "weather_station_id": "USC00110072"
        },
        {
            "avg_max_temperature": 16.351,
            "avg_min_temperature": 4.318,
            "measurement_year": 1999,
            "total_precipitation": 92,
            "weather_station_id": "USC00110072"
        },
        {
            "avg_max_temperature": 16.163,
            "avg_min_temperature": 4.701,
            "measurement_year": 2005,
            "total_precipitation": 64,
            "weather_station_id": "USC00110072"
        },
        {
            "avg_max_temperature": 17.394,
            "avg_min_temperature": 5.404,
            "measurement_year": 2012,
            "total_precipitation": 74,
```


Below shows weather data with paginated and each page shows only 10 records



Filtering data show json



Below is page returns all data wirh paginated and filter option

==========

Extra Credit - Deployment

(Optional.) Assume you are asked to get your code running in the cloud using AWS. What tools and AWS services would you use to deploy the API, database, and a scheduled version of your data ingestion code? Write up a description of your approach.

Solution:

We can do deployments in choosing different services and tools, Below approach is my approach.

AWS Services:

1) Lambda

2) EMR

3) AWS RDS Postgress aurora

4) S3

5) IAM

6) Lake formation

7) Event Bridge (Scheduling events from CloudWatch alarms)

Step1: Ingestion

1) Assume the text files (some weather and crop yield data) or RDBMS or any On-prem is source.

2) We need to ingest the data from on-prem to AWS S3 using below services

a) AWS EMR to connect to source RDBMS and using jdbc connector with spark and save to s3

or

b) EC2 instance load flat files from sftp location to s3.

Step 2:

1) I prefer start reading files using lambda function , Because it is serverless and it is cheap when compare with using EC2 and EMR etc.. Assuming if its not meet below limitations

The disk space (ephemeral) is limited to 512 MB.

The default deployment package size is 50 MB.

The memory range is from 128 to 3008 MB.

The maximum execution timeout for a function is 15 minutes*.

Requests limitations by Lambda:

Request and response (synchronous calls) body payload size can be up to 6 MB.

The event request (asynchronous calls) body can be up to 128 KB.

2) Using psycopg2 or sqlalchemy(preferred with pandas or spark dataframes) packages and running load_weather_data code in lambda directly loads to postgress table, but we need to setpup below to lambda

1) Subnects and subnet group id to allow trafic from postgress and lambda

2) Enable cloudwatch logs to capture the logs.

Deploying Flask in AWS EC2

Third party tool:

1) Concourse

2) Terraform

3) Swagger UI

AWS Services:

1) AWS EC2

2) AWS API Gateway

3) AWS Lambda

4) AWS RDS Postgress Instance or DynamoDB (Creating database table0s)

5) AWS CloudWatch

6) AWS Secrect Manager to store RDS postgress username,password..connections details etc..

7) AWS Route 53

8) AWS Elastic Beanstalk

Step1 : Using Terreform creating all above aws services

AWS Lambda

1.a.Create lambda fucntion with needed s3 path for python file and layers for needed python packages(psycopg2)

API Gateway:

1.b.Create API Gateway and provide swagger template

1.c.Create API Gateway stage

1.d.Create API deployment

1.e. Create api gateways responce for different responce

Swagger template:

1.f.Create template file using swagger_template.json

AWS RDS Cluster

1.g. Create AWS RDS cluster

AWS Route53

1.h. Create REST API Route53 for domain name

Step2 : Setup concourse pipeline to deploy terraform code

2.a. Verify plan

2.b. Apply changes

Flow:

1)Create Custom domain name in API gateway for swagger

2)From swagger we invoke API gateway and it triggers lambda function and payload to pass to lambda

All above flow needs additional aws services like event bridge for schdueling, cloudwatch for logs, cloudwatch rules for scheduling, s3 for storage files,ECS to run flask applications,lakeformation for permissions.