Netstock Data Engineer Challenge - Weather

August 28, 2023

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	2023-08-37	Document created	Erhardt Nel

Contents

1 Introduction

intro text here

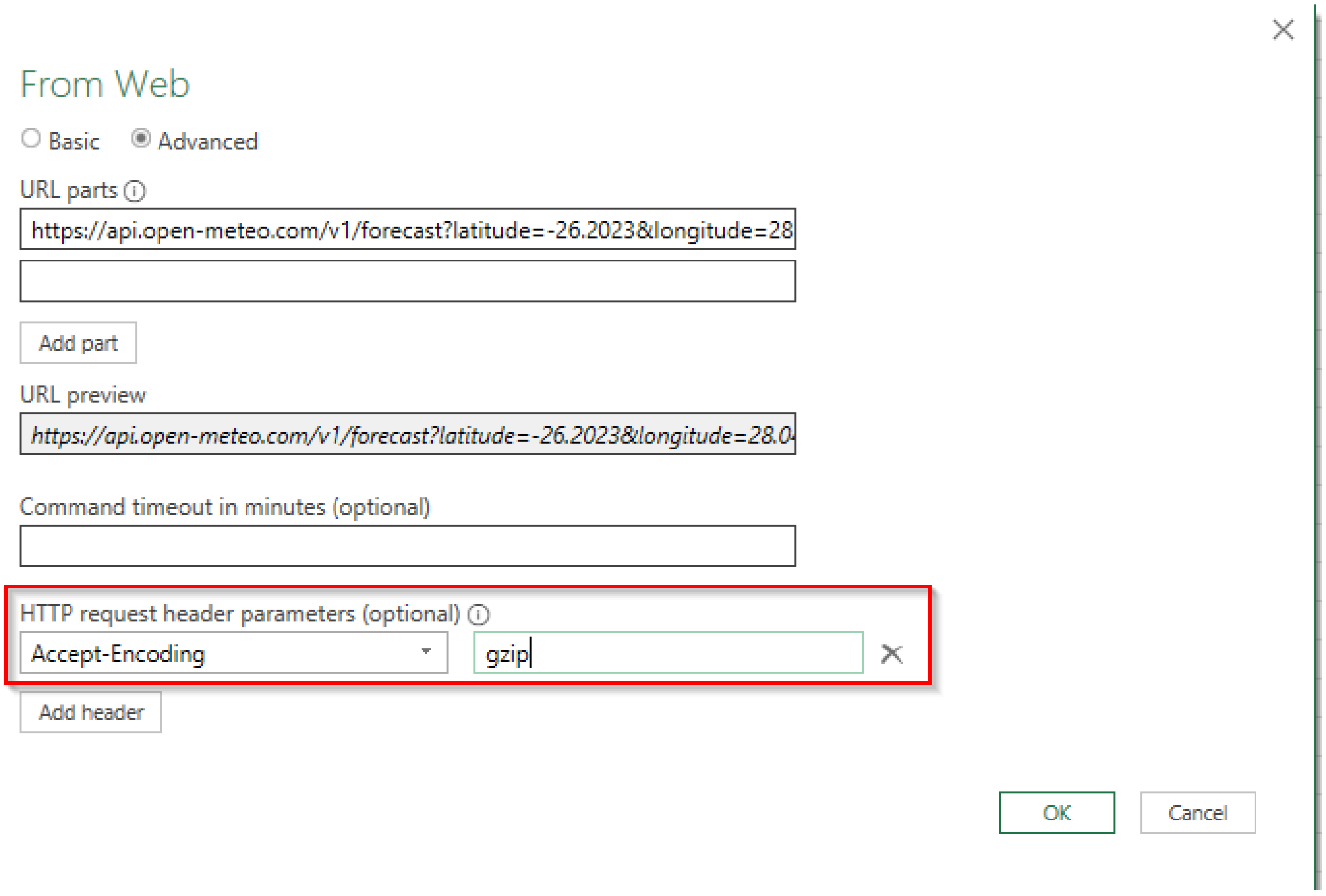
2 API Interface

The open-meteo.com weather API interface allow for hourly or daily weather forecasts. The decision was made to use the following global cities:

- Johannesburg - South Africa
- London - United Kingdom
- New York City - United States

3 Undestanding the API Data

The Api interface allows for unauthenticated access. I will be using a simple explorative tool like Microsot Excel to onnect to to the API to understand and unpack the returned data structures. This is achieved by setting up an excel web query to the API endpoint. It is important to add the Accept-Encoding gzip header to the request because the api returns a gzipped stream.


From Web

Basic [✓ Advanced]

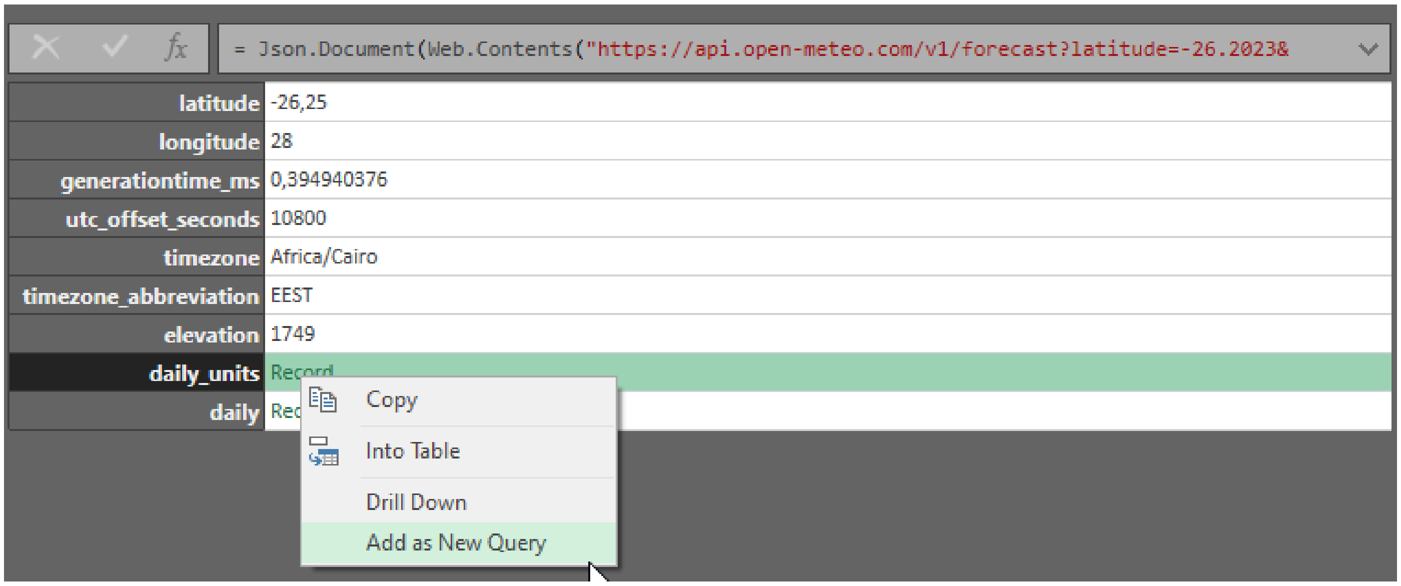
URL parts: https://api.open-meteo.com/v1/forecast?latitude=-26.2023&longitude=28

URL preview: https://api.open-meteo.com/v1/forecast?latitude=-26.2023&longitude=28.0

Command timeout in minutes (optional):

HTTP request header parameters (optional): Accept-Encoding gzip

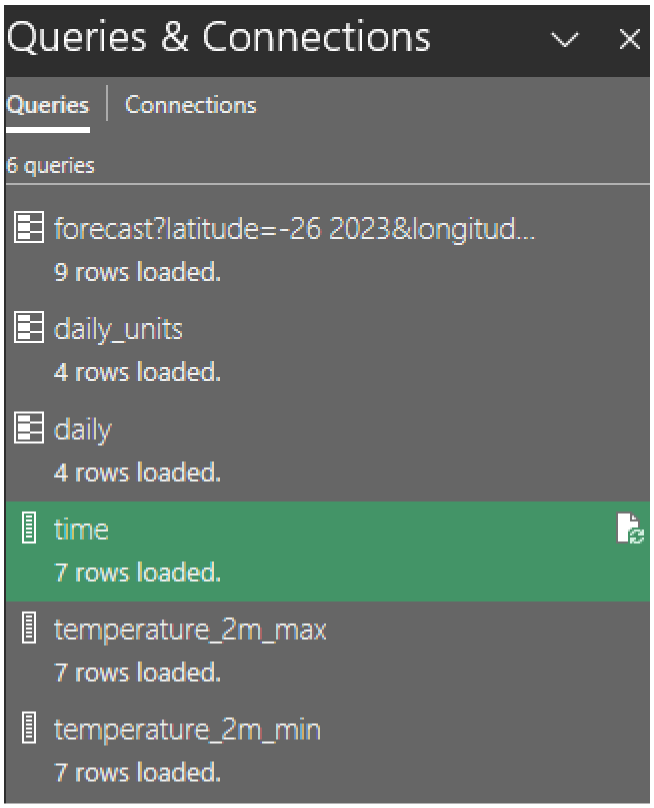
The Excel query returns mulliple nodes in JSON format. The next step is to expand each individual Json Array into a separate sub query.



```
= Json.Document(Web.Contents("https://api.open-meteo.com/v1/forecast?latitude=-26.2023&
```

latitude	-26,25
longitude	28
generationtime_ms	0,394940376
utc_offset_seconds	10800
timezone	Africa/Cairo
timezone_abbreviation	EEST
elevation	1749
daily_units	Record
daily	Record

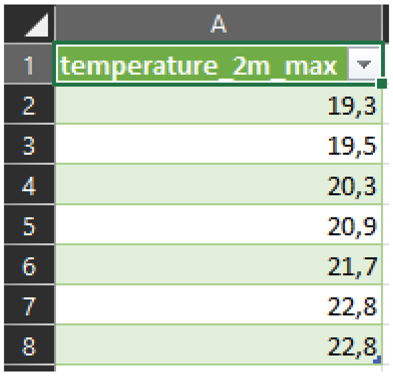
We now have individual queries to all the JSON node arrays from the API result. These queries can be individaully refreshed as data tables in Excel.



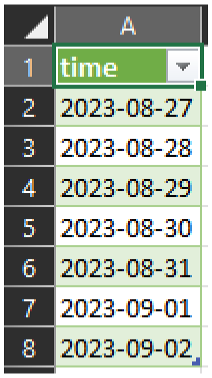
6 queries

- forecast?latitude=-26 2023&longitud... — 9 rows loaded.
- daily_units — 4 rows loaded.
- daily — 4 rows loaded.
- time — 7 rows loaded.
- temperature_2m_max — 7 rows loaded.
- temperature_2m_min — 7 rows loaded.

This porcess allows me to quickly unpack what the data structures are in Excel Sheets.



	A
1	temperature_2m_max
2	19,3
3	19,5
4	20,3
5	20,9
6	21,7
7	22,8
8	22,8



	A
1	time
2	2023-08-27
3	2023-08-28
4	2023-08-29
5	2023-08-30
6	2023-08-31
7	2023-09-01
8	2023-09-02

4 Connect to API and push data to Postgresql

4.1 Postgresql

The decision was made to post the weather API data to a local postgresql database hosted on a Debian Linux Virtual machine. I created two tables,

First - Temp Table: hld.weather

```
CREATE SCHEMA IF NOT EXISTS TEMP; DROP TABLE IF EXISTS TEMP.hld_weather;
CREATE TABLE IF NOT EXISTS TEMP.hld_weather(
        id TEXT PRIMARY KEY NOT NULL,
        city TEXT,
        dailytime TIMESTAMP,
        temp_max DECIMAL,
        temp_min DECIMAL,
        sunset TIMESTAMP,
        sunrise TIMESTAMP,
        windspeed DECIMAL
);
```

Second - Live Table: lve.weather

```
CREATE SCHEMA IF NOT EXISTS TEMP; DROP TABLE IF EXISTS TEMP.lve_weather;
CREATE TABLE IF NOT EXISTS TEMP.lve_weather(
        id TEXT PRIMARY KEY NOT NULL,
        city TEXT,
        dailytime TIMESTAMP,
        temp_max DECIMAL,
        temp_min DECIMAL,
        sunset TIMESTAMP,
        sunrise TIMESTAMP,
        windspeed DECIMAL
);
```

The Temp table is used to push delta changes to the live table by using the native Postgresql Merge functionality. This script will merge **HLD** table into the **LVE** table based on if the ID column is matched and if not it will insert a new record.

Merge SQL script

```
merge into TEMP.lve_weather sda
using TEMP.hld_weather sdn
on sda.id = sdn.id
when matched then
  update set city = sdn.city, dailytime = sdn.dailytime, temp_max = sdn.temp_max, temp_min = ↩
      sdn.temp_min, sunset = sdn.sunset, sunrise = sdn.sunrise, windspeed = sdn.windspeed
when not matched then
  insert (id, city, dailytime, temp_max,temp_min,sunset,sunrise,windspeed)
  values (sdn.id, sdn.city, sdn.dailytime, sdn.temp_max,sdn.temp_min,sdn.sunset,sdn.sunrise, ↩
      sdn.windspeed);
```

4.2 Extract Scripts

The extract scripts were build with the KISS principle by limiting external dependencies on the virtual machine. To that extend I have used native linux tools wrapped in a bash script to perform the following tasks:

1. Connect to the API using Curl
2. Validate that the return string is valid JSON by using the jq linux command
3. Extract the JSON array values [time, temperature_2m_max, temperature_2m_min, sunset, sunrise, windspeed_10m_max] using the Linux jq command and adding the selectors.
4. The extracted values were then combined into a csv file for each of the cities.
5. The csv file is then posted to PostgreSQL using the **psql** commandline syntax.
6. Errors and exceptions were written to a log.txt file
7. A Scheduled cron job that runs hourly, connects to the API and update the SQL tables.

Script to fetch Johannesburg Data from API

```
#!/bin/bash

## This script fetch the different cities json data and check if valid json is returned.
## It then transform the different columns and post the data to a postgresql temp table.
## Using the native postgresql merge statment to merge delta changes into the master table.

##-------------------------------------------------------
##--- Global Variables ---
##-------------------------------------------------------
CITY="latitude=-26.2023&longitude=28.0436"
STARTDATE="2023-06-01"
ENDDATE="2023-08-27"
WORKFOLDER=/media/sf_shared/netstock/postgresql
SQLLOG=$WORKFOLDER/log.txt
CITYNAME="Johannesburg"
##-------------------------------------------------------

##-- Load the JSON from the webservice into a temp variable --
JSONRAW=$(curl -s "https://api.open-meteo.com/v1/forecast?$CITY&daily=temperature_2m_max, ↩
    temperature_2m_min,sunrise,sunset,windspeed_10m_max&timezone=Africa%2FCairo&start_date= ↩
    $STARTDATE&end_date=$ENDDATE")
##-- Cleanup the log file
touch $SQLLOG
rm $SQLLOG

##-------------------------------------------------------
##-- Check if the returned JSON is valid  and transform the columns into csv --
if [ $(echo -n $JSONRAW | jq empty >  /dev/null 2>&1; echo $?) -eq 0 ]; then
        # Clean up the temp table
        psql -d netstock -a -f $WORKFOLDER/cleanup.sql >> $SQLLOG 2>&1
        echo $JSONRAW | jq '.daily.time[]' | tr -d '"' | awk '{print "JHB_"$1}' > $WORKFOLDER/ ↩
            id.csv
        echo $JSONRAW | jq '.daily.time[]' | tr -d '"' | awk '{print "Johannesburg"}' >  ↩
            $WORKFOLDER/city.csv
        echo $JSONRAW | jq '.daily.time[]' > $WORKFOLDER/timedaily.csv
        echo $JSONRAW | jq '.daily.temperature_2m_max[]' > $WORKFOLDER/tempmax.csv
        echo $JSONRAW | jq '.daily.temperature_2m_min[]' > $WORKFOLDER/tempmin.csv
        echo $JSONRAW | jq '.daily.sunset[]' > $WORKFOLDER/sunset.csv
        echo $JSONRAW | jq '.daily.sunrise[]' > $WORKFOLDER/sunrise.csv
        echo $JSONRAW | jq '.daily.windspeed_10m_max[]' > $WORKFOLDER/wind.csv
        touch $WORKFOLDER/out.csv
        rm $WORKFOLDER/out.csv
        #echo "dailytime,temperature_max,temperature_min,sunset,sunrise,windspeed" > out.csv
        paste -d',' $WORKFOLDER/id.csv $WORKFOLDER/city.csv $WORKFOLDER/timedaily.csv  ↩
            $WORKFOLDER/tempmax.csv $WORKFOLDER/tempmin.csv $WORKFOLDER/sunset.csv $WORKFOLDER ↩
            /sunrise.csv $WORKFOLDER/wind.csv >>$WORKFOLDER/out.csv
        psql -d netstock -c "\copy temp.hld_weather from $WORKFOLDER/out.csv with delimiter  ↩
            ',' csv null as 'NULL';" >> $SQLLOG 2>&1
        #merge Delta changes into master table
        psql -d netstock -a -f $WORKFOLDER/merge.sql >> $SQLLOG 2>&1
else
        echo "json is invalid"
fi
##-------------------------------------------------------
## Do a cleanup
rm $WORKFOLDER/*.csv
##-------------------------------------------------------
```