

Netstock Data Engineer Challenge - Weather

August 28, 2023

Copyright © 2023 NETSTOCK. All rights reserved. No portion of this document may be reproduced by any process without the written permission of NETSTOCK.

Terms of use: This publication has been prepared for general guidance on matters of interest only.

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

Contents

1	Introduction	1
2	API Interface	1
3	Understanding the API Data	1
4	Connect to API and push data to Postgresql	4
4.1	Postgresql Installation	4
4.2	Postgresql Setup	4
4.3	Extract Scripts	6
4.3.1	Extract Script Dependencies	6
4.4	PostgreSQL Table Data	9
5	Documentation Environment	10
5.1	Asciidoc	10
5.1.1	Asciidoc Documentation	10
5.1.2	Asciidoc Installation on Linux	10
6	Elements needed to make the solution Production Ready	11
6.1	Basic Penetration Test	11

List of Figures

1	Output of Live table	9
---	--------------------------------------	---

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 Understanding 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 zipped stream.

×

From Web

☐ Basic ☒ Advanced

URL parts ①

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

Add part

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) ①

Accept-Encoding ▾

gzip

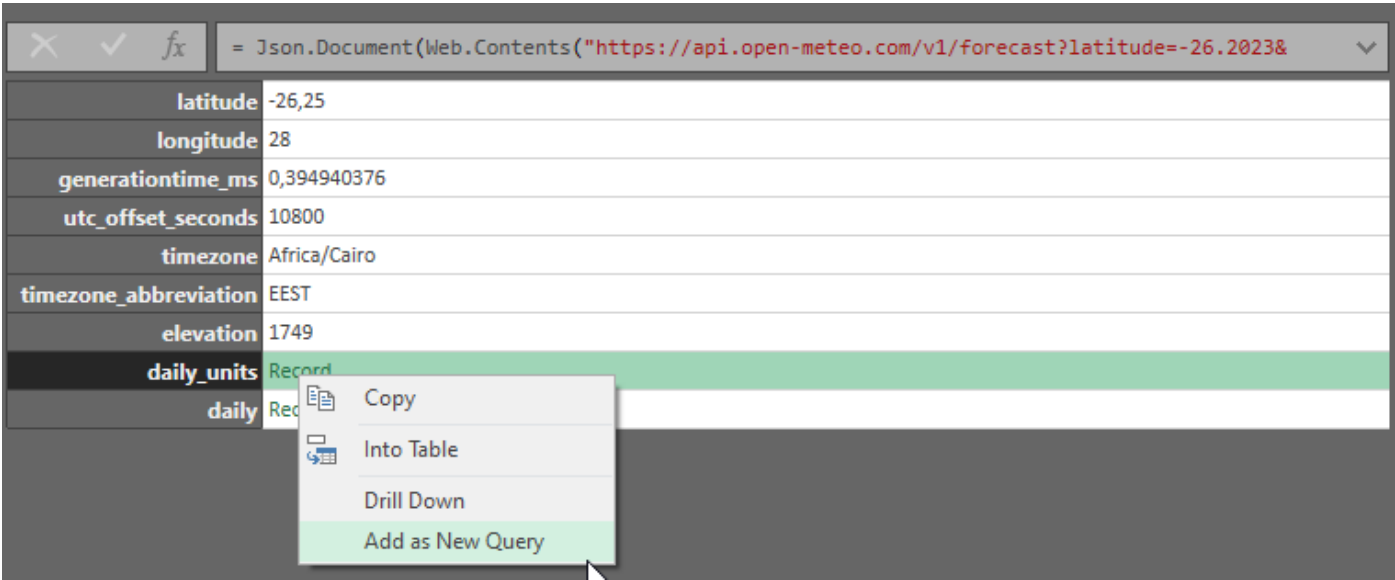
×

Add header

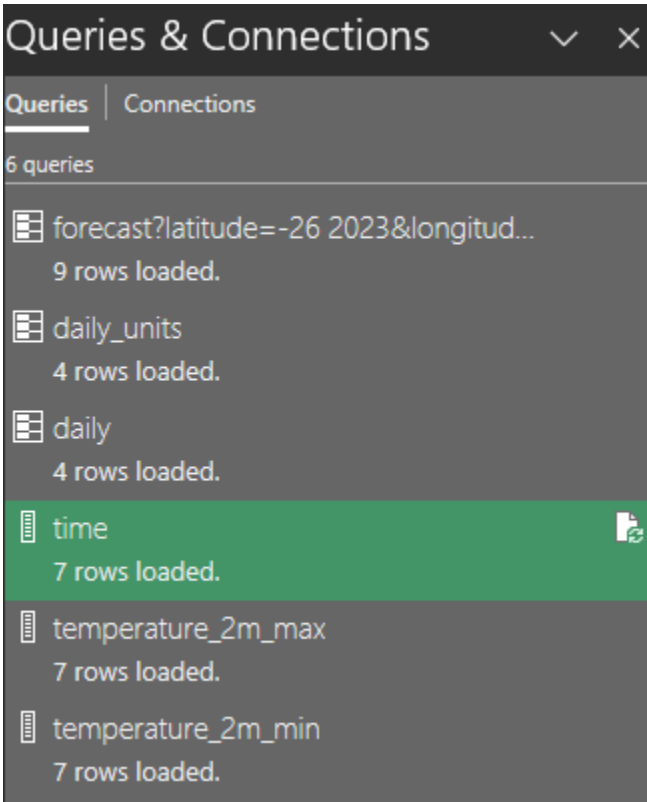
OK

Cancel

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



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



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 Installation

The following commands were used to install postgresql on my virtual machine.

```
sudo apt install postgresql postgresql-contrib
```

I then allowed any client from any ip to connect to the Postgresql server by editing the /etc/postgresql/15/main/postgresql.conf configuration file and adding the following line to it.

```
listen_addresses = *
```

Please see Production Ready section below on steps to harden the Postgresql installation.

4.2 Postgresql Setup

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);
```

4.3 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.

4.3.1 Extract Script Dependencies

The following commands were used to install the dependencies on my virtual machine.

```
sudo apt-get install jq  
sudo apt-get install curl
```

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
##-----
```

The output of the log.txt file contains the following information.

```
CREATE SCHEMA IF NOT EXISTS TEMP; DROP TABLE IF EXISTS TEMP.hld_weather;
psql:/media/sf_shared/netstock/postgresql/cleanup.sql:1: NOTICE:  schema "temp" already exists ↵
, skipping
CREATE SCHEMA
DROP TABLE
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);
CREATE TABLE
COPY 88
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);
MERGE 88
```

4.4 PostgreSQL Table Data

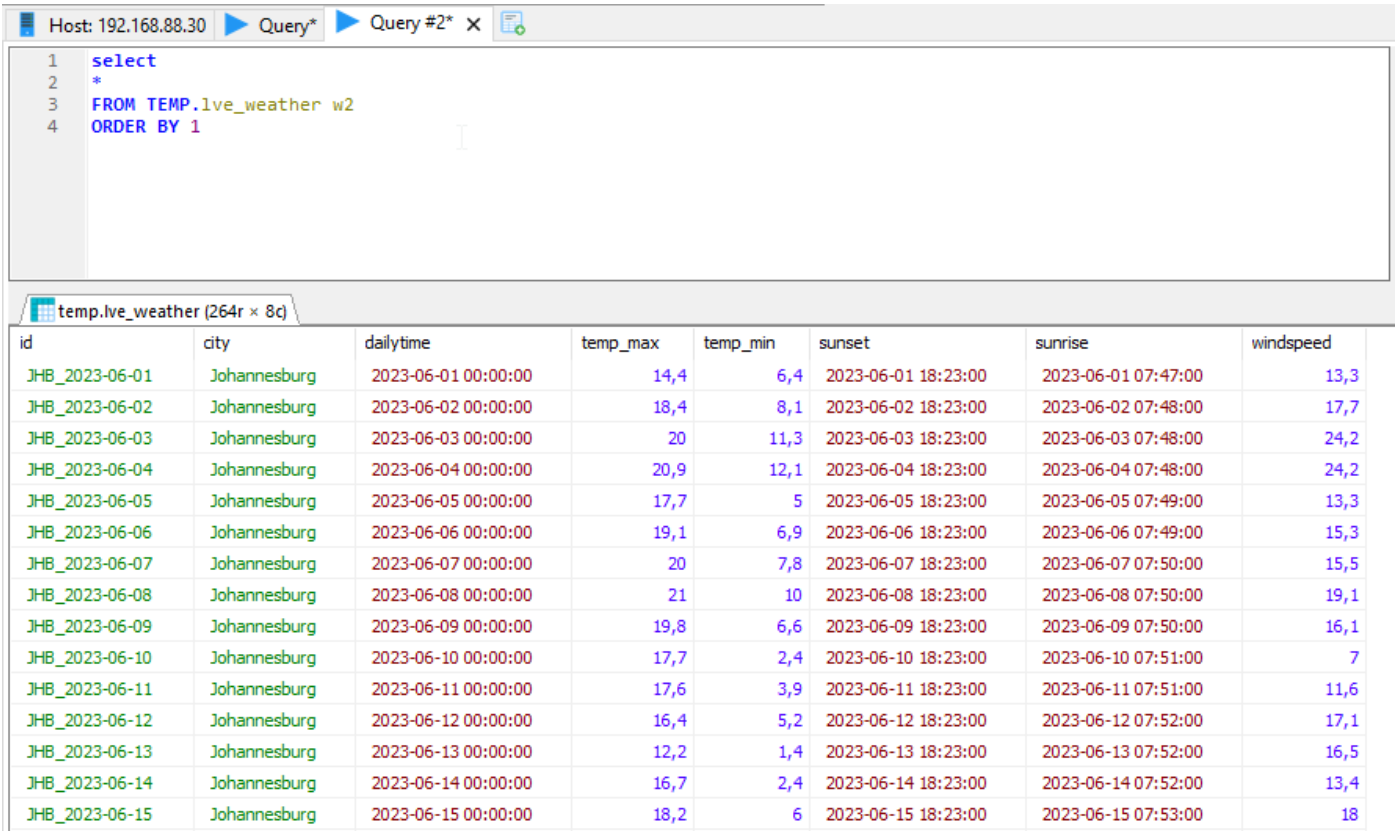


Figure 1: Output of Live table

5 Documentation Environment

5.1 AsciiDoc

I have used the Python based AsciiDoc toolchain with the DBLaTeX rendering engine. AsciiDoc allows you to add content in a text based markup language. The tool chain allows you to convert your output to both PDF and HTML. The documentation source files are 100% text based which makes it ideal to put it under source control.

I use a custom MAKE file to convert asciidoc.txt file to PDF or HTML by issuing the command "make pdf" or "make html"

Custom Make file

```
TARGET=asciidoc

.PHONY: clean html xml pdf

html: $(TARGET).html

pdf: $(TARGET).pdf

all: html pdf

%.html: %.txt
    asciidoctor $<

%.pdf: %.txt
    a2x --no-xmllint -v -a docinfo -fpdf --dlatex-opts="-s asciidoc-dlatex-custom.sty" <-
        --dlatex-opts="--param=doc.lot.show=figure,table" $<
    gs -sDEVICE=pdfwrite -dCompatibilityLevel=1.4 -dPDFSETTINGS=/printer -dNOPAUSE -dQUIET <-
        -dBATCH -sOutputFile=output.pdf asciidoc.pdf

clean:
    rm -f *~ $(TARGET).pdf $(TARGET).html docbook-xsl.css *.xml
```

5.1.1 AsciiDoc Documentation

AsciiDoc Markup Documentation

5.1.2 AsciiDoc Installation on Linux

```
# Install asciidoc with the dlatex backend
sudo apt-get install asciidoc-dlatex

#install asciidoctor for the html backend
sudo apt-get install asciidoctor
```

6 Elements needed to make the solution Production Ready

6.1 Basic Penetration Test

A basic nmap scan against the virtual machine revealed port 22 [SSH] and 5432 [Postgresql] to be opened.

```
sudo nmap 192.168.88.30
```

```
Starting Nmap 7.93 ( https://nmap.org ) at 2023-08-28 00:54 SAST
```

```
Nmap scan report for 192.168.88.30
```


```
Host is up (0.000035s latency).
```

PORT	STATE	SERVICE
22/tcp	open	ssh
5432/tcp	open	postgresql

```
Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
```

The version of Postgresql is revealed as 15.3 by issuing the following command: `psql -c "SELECT version();"`

There are no known exploits listed for version 15.3 of postgresql



☐ Verified ☐ Has App

Filters

Reset All

Show 15

Search: postgresql

Date	D	A	V	Title	Type	Platform	Author
2023-05-23				PnSCADA v2.x - Unauthenticated PostgreSQL Injection	WebApps	Hardware	Momen Eldawakhly
2023-04-05				PostgreSQL 9.6.1 - Remote Code Execution (RCE) (Authenticated)	Remote	Multiple	Paulo Trindade
2022-03-30				PostgreSQL 9.3-11.7 - Remote Code Execution (RCE) (Authenticated)	Remote	Multiple	b4keSn4ke
2019-05-08				PostgreSQL 9.3 - COPY FROM PROGRAM Command Execution (Metasploit)	Remote	Multiple	Metasploit
2018-08-13				PostgreSQL 9.4-0.5.3 - Privilege Escalation	Local	Linux	Johannes Segitz
2014-06-13				PostgreSQL 8.4.1 - JOIN Hashtable Size Integer Overflow Denial of Service	DoS	Multiple	Bernt Marius Johnsen
2010-01-27				PostgreSQL - 'bitsubstr' Buffer Overflow	DoS	Linux	Intevydis
2009-03-11				PostgreSQL 8.3.6 - Conversion Encoding Remote Denial of Service	DoS	Linux	Afonin Denis
2009-03-10				PostgreSQL 8.3.6 - Low Cost Function Information Disclosure	Local	Multiple	Andres Freund
2005-02-01				PostgreSQL 7.x - Multiple Vulnerabilities	DoS	Linux	ChoiX
2000-04-23				PostgreSQL 6.3.2/6.5.3 - Cleartext Passwords	Local	Immunix	Robert van der Meulen
2009-01-25				PostgreSQL 8.2/8.3/8.4 - UDF for Command Execution	Local	Linux	Bernardo Damele