Python - Powershell - SQL integration

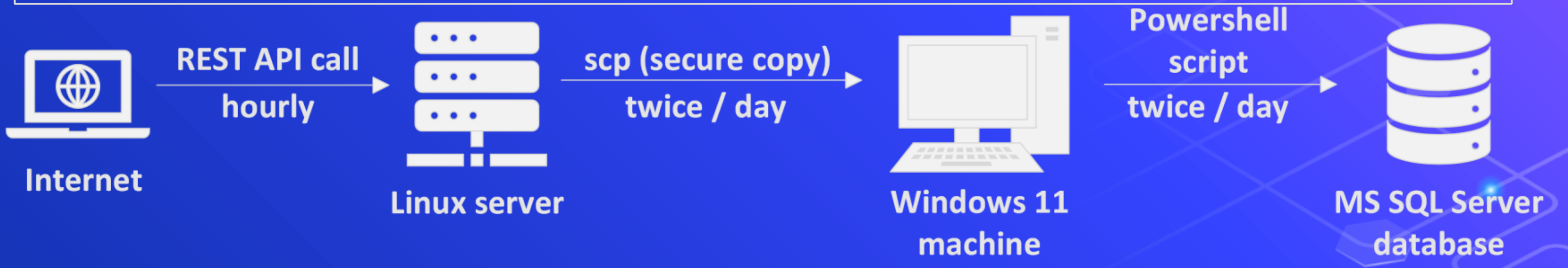
https://github.com/adisirbu/SQL-Project-Public

Using multiple techniques and programming languages to achieve a desired data flow and storage.

Objective: obtain a database of regularly updated weather information (overall conditions, temperature, humidity and wind characteristics) for 5 Romanian cities in a totally automatic way.

Using multiple techniques and programming languages to achieve a desired data flow

Data flow

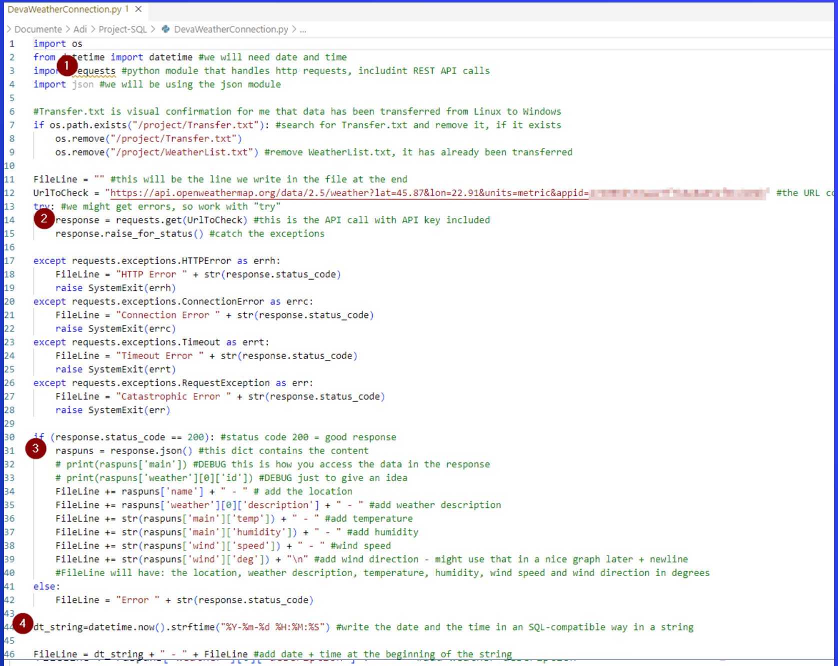


Programming languages used: Python, Powershell and SQL

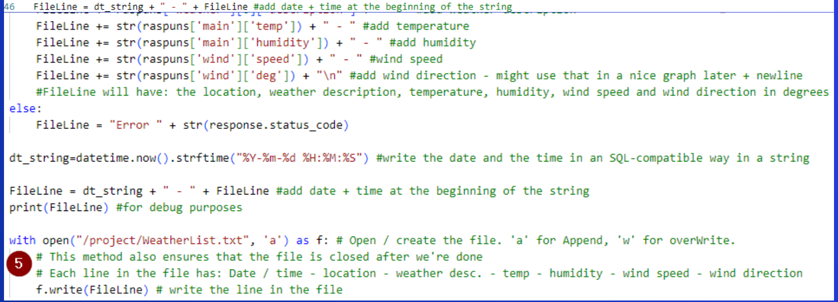
Techniques: REST API calls, passwordless scp (using SSH private – public key), Linux automation using crontab and Windows automation using Task Scheduler

1.
Linux server

Python script sample: DevaWeatherConnection.py


DevaWeatherConnection.py 1
Documente > Adi > Project-SQL > DevaWeatherConnection.py > ...
1
2
3
4

```python
import os
from datetime import datetime #we will need date and time
import requests #python module that handles http requests, includint REST API calls
import json #we will be using the json module

#Transfer.txt is visual confirmation for me that data has been transferred from Linux to Windows
if os.path.exists("/project/Transfer.txt"): #search for Transfer.txt and remove it, if it exists
    os.remove("/project/Transfer.txt")
    os.remove("/project/WeatherList.txt") #remove WeatherList.txt, it has already been transferred

FileLine = "" #this will be the line we write in the file at the end
UrlToCheck = "https://api.openweathermap.org/data/2.5/weather?lat=45.87&lon=22.91&units=metric&appid=[redacted]" #the URL c
try: #we might get errors, so work with "try"
    response = requests.get(UrlToCheck) #this is the API call with API key included
    response.raise_for_status() #catch the exceptions

except requests.exceptions.HTTPError as errh:
    FileLine = "HTTP Error " + str(response.status_code)
    raise SystemExit(errh)
except requests.exceptions.ConnectionError as errc:
    FileLine = "Connection Error " + str(response.status_code)
    raise SystemExit(errc)
except requests.exceptions.Timeout as errt:
    FileLine = "Timeout Error " + str(response.status_code)
    raise SystemExit(errt)
except requests.exceptions.RequestException as err:
    FileLine = "Catastrophic Error " + str(response.status_code)
    raise SystemExit(err)

if (response.status_code == 200): #status code 200 = good response
    raspuns = response.json() #this dict contains the content
    # print(raspuns['main']) #DEBUG this is how you access the data in the response
    # print(raspuns['weather'][0]['id']) #DEBUG just to give an idea
    FileLine += raspuns['name'] + " - " # add the location
    FileLine += raspuns['weather'][0]['description'] + " - " #add weather description
    FileLine += str(raspuns['main']['temp']) + " - " #add temperature
    FileLine += str(raspuns['main']['humidity']) + " - " #add humidity
    FileLine += str(raspuns['wind']['speed']) + " - " #wind speed
    FileLine += str(raspuns['wind']['deg']) + "\n" #add wind direction - might use that in a nice graph later + newline
    #FileLine will have: the location, weather description, temperature, humidity, wind speed and wind direction in degrees
else:
    FileLine = "Error " + str(response.status_code)

dt_string=datetime.now().strftime("%Y-%m-%d %H:%M:%S") #write the date and the time in an SQL-compatible way in a string

FileLine = dt_string + " - " + FileLine #add date + time at the beginning of the string
```



```python
    FileLine += str(raspuns['main']['temp']) + " - " #add temperature
    FileLine += str(raspuns['main']['humidity']) + " - " #add humidity
    FileLine += str(raspuns['wind']['speed']) + " - " #wind speed
    FileLine += str(raspuns['wind']['deg']) + "\n" #add wind direction - might use that in a nice graph later + newline
    #FileLine will have: the location, weather description, temperature, humidity, wind speed and wind direction in degrees
else:
    FileLine = "Error " + str(response.status_code)

dt_string=datetime.now().strftime("%Y-%m-%d %H:%M:%S") #write the date and the time in an SQL-compatible way in a string

FileLine = dt_string + " - " + FileLine #add date + time at the beginning of the string
```

```python
print(FileLine) #for debug purposes

with open("/project/WeatherList.txt", 'a') as f: # Open / create the file. 'a' for Append, 'w' for overWrite.
    # This method also ensures that the file is closed after we're done
    # Each line in the file has: Date / time - location - weather desc. - temp - humidity - wind speed - wind direction
    f.write(FileLine) # write the line in the file
```

1. You will need Python os, requests and json modules

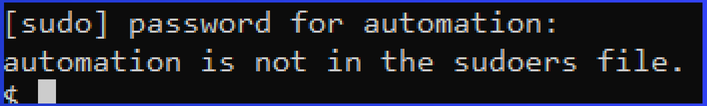
2. The way I obtained the information was to make REST API calls to openweathermap.org. You can get a free API key serving up to 1000 calls / day.

3. The content returned by the call is json-formatted. For Python, this is a dict object.

4. The date / time is formatted in a way suitable for SQL since the beginning.

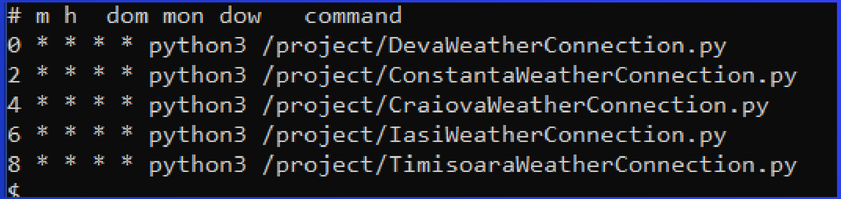
5. For every API call (1 / hour for every city for a total of 120 calls / day), the information is appended as a new line in a file called WeatherList.txt. This file serves as a buffer where the info is kept until transfer to Windows.

Linux: what else?



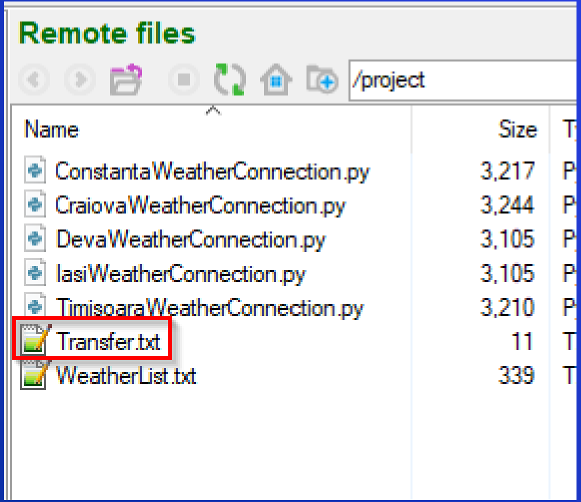
```
[sudo] password for automation:
automation is not in the sudoers file.
$
```

I created a new user “*automation*” which does not have sudo privileges. For security reasons, “root” was not involved. It is *automation* who runs the scripts.



```
# m h  dom mon dow   command
0 * * * * python3 /project/DevaWeatherConnection.py
2 * * * * python3 /project/ConstantaWeatherConnection.py
4 * * * * python3 /project/CraiovaWeatherConnection.py
6 * * * * python3 /project/IasiWeatherConnection.py
8 * * * * python3 /project/TimisoaraWeatherConnection.py
$
```

Every city has its own script, running hourly, but at different minutes (0, 2, 4, 6 and 8). This was achieved using crontab.



Remote files

/project

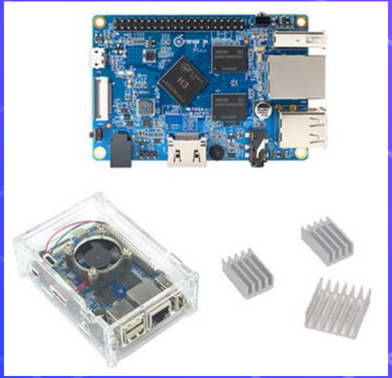
Name	Size	T
ConstantaWeatherConnection.py	3,217	P
CraiovaWeatherConnection.py	3,244	P
DevaWeatherConnection.py	3,105	P
IasiWeatherConnection.py	3,105	P
TimisoaraWeatherConnection.py	3,210	P
Transfer.txt	11	T
WeatherList.txt	339	T

All the information is written – line by line and in mode “append” - to the same WeatherList.txt file. If the script finds the “Transfer.txt” file in the folder, it means previous information was successfully transferred to Windows, so the buffer is no longer needed. Then and only then, the script erases both Transfer.txt and WeatherList.txt to restart the cycle of writing information to the buffer.

Why Linux?

- In order to get regular information, I needed a server, i.e. a computer running non-stop.
- A system-on-a-chip is a cheap way of setting a home server.

I used an Orange Pi PC running Ubuntu 20.04. It is very small, unobtrusive and it needs 15 W max.

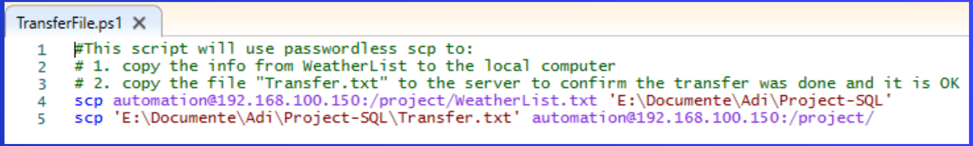


- Linux automation is easy to set up with crontab.
- Windows to Linux communication and control is easy to set up using ssh.

2.
Windows machine

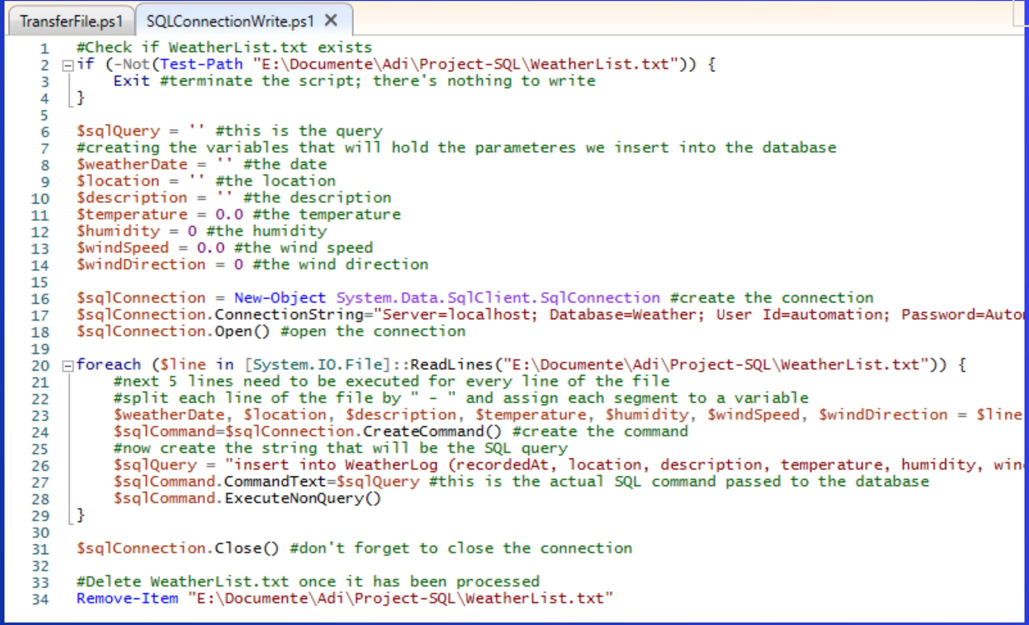
The scripts

There are 2 Powershell scripts:



TransferFile.ps1 ✕

```
#This script will use passwordless scp to:
# 1. copy the info from WeatherList to the local computer
# 2. copy the file "Transfer.txt" to the server to confirm the transfer was done and it is OK
scp automation@192.168.100.150:/project/WeatherList.txt 'E:\Documente\Adi\Project-SQL'
scp 'E:\Documente\Adi\Project-SQL\Transfer.txt' automation@192.168.100.150:/project/
```



TransferFile.ps1 | SQLConnectionWrite.ps1 ✕

```
#Check if WeatherList.txt exists
if (-Not(Test-Path "E:\Documente\Adi\Project-SQL\WeatherList.txt")) {
    Exit #terminate the script; there's nothing to write
}

$sqlQuery = '' #this is the query
#creating the variables that will hold the parameteres we insert into the database
$weatherDate = '' #the date
$location = '' #the location
$description = '' #the description
$temperature = 0.0 #the temperature
$humidity = 0 #the humidity
$windSpeed = 0.0 #the wind speed
$windDirection = 0 #the wind direction

$sqlConnection = New-Object System.Data.SqlClient.SqlConnection #create the connection
$sqlConnection.ConnectionString="Server=localhost; Database=Weather; User Id=automation; Password=Auto
$sqlConnection.Open() #open the connection

foreach ($line in [System.IO.File]::ReadLines("E:\Documente\Adi\Project-SQL\WeatherList.txt")) {
    #next 5 lines need to be executed for every line of the file
    #split each line of the file by " - " and assign each segment to a variable
    $weatherDate, $location, $description, $temperature, $humidity, $windSpeed, $windDirection = $line
    $sqlCommand=$sqlConnection.CreateCommand() #create the command
    #now create the string that will be the SQL query
    $sqlQuery = "insert into WeatherLog (recordedAt, location, description, temperature, humidity, win
    $sqlCommand.CommandText=$sqlQuery #this is the actual SQL command passed to the database
    $sqlCommand.ExecuteNonQuery()
}

$sqlConnection.Close() #don't forget to close the connection

#Delete WeatherList.txt once it has been processed
Remove-Item "E:\Documente\Adi\Project-SQL\WeatherList.txt"
```

Script #1 connects to Linux and copies via **scp**:

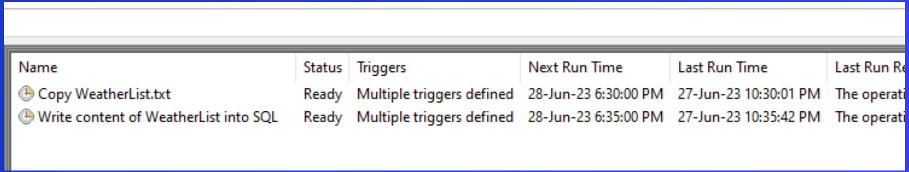
- One file **to** Windows with the weather data
- One file **from** Windows that tells Linux the data transfer was successful. The presence of this file in the Linux folder = "it is ok to erase the buffer"

5 minutes after the script above executes, the second script connects to the local SQL database (*server = localhost*) with the user "*automation*" and writes all the information from the file into the database ("***insert into…***").

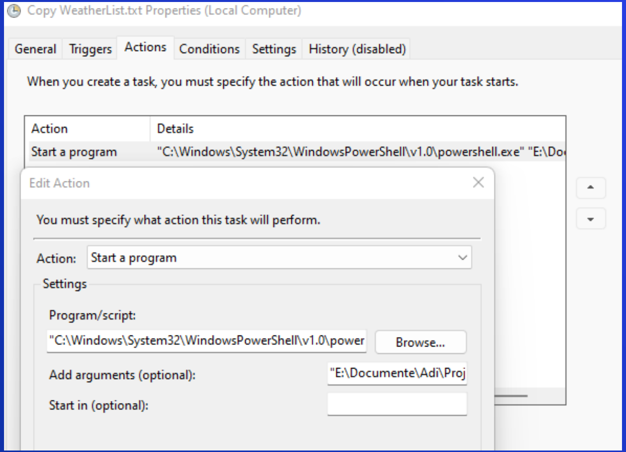
After closing the sql connection, it then deletes the WeatherList.txt file that is no longer useful, closing the data transfer loop.

The automation

Windows automation is done with the Task Scheduler utility. Each script has its own task scheduled to run every afternoon and every evening. The second script (writing the content into the sql database) runs 5 mins after the first.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Re
Copy WeatherList.txt	Ready	Multiple triggers defined	28-Jun-23 6:30:00 PM	27-Jun-23 10:30:01 PM	The operati
Write content of WeatherList into SQL	Ready	Multiple triggers defined	28-Jun-23 6:35:00 PM	27-Jun-23 10:35:42 PM	The operati



Copy WeatherList.txt Properties (Local Computer)

When you create a task, you must specify the action that will occur when your task starts.

Action	Details
Start a program	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" "E:\Do

Edit Action

You must specify what action this task will perform.

Action: Start a program

Settings

Program/script: "C:\Windows\System32\WindowsPowerShell\v1.0\power

Add arguments (optional): "E:\Documente\Adi\Proj

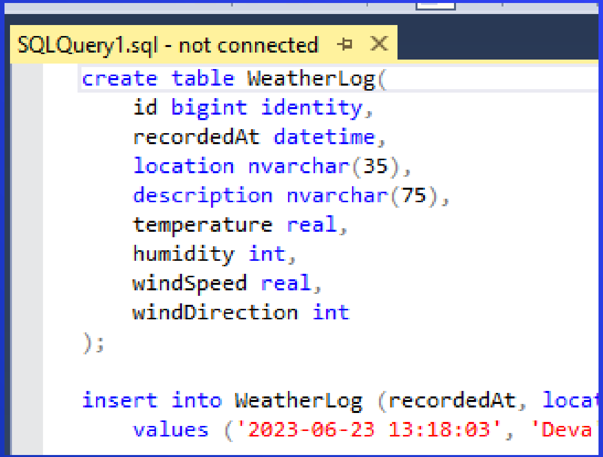
Start in (optional):

How to automate a Powershell script? The task action = “Start a program”, the program is **Powershell.exe** (just put the complete path to powershell.exe) and **the argument is the script path** (again, just put **the whole path** inside quotes). You can add the argument “**-File**” just before the path, but it is implicit. For some reason, Powershell really likes folders and files without spaces in the script path.

What happens if the computer is not on when the scripts are scheduled? Nothing, they wait for the next run time while the info is still being recorded in the Linux folder and file.

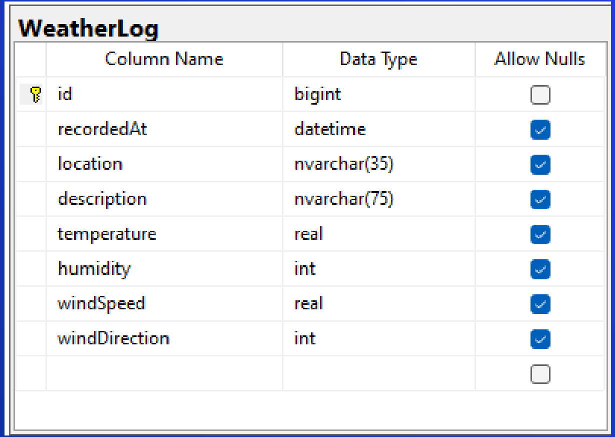
3.
SQL server and database

The database



SQLQuery1.sql - not connected

```
create table WeatherLog(
    id bigint identity,
    recordedAt datetime,
    location nvarchar(35),
    description nvarchar(75),
    temperature real,
    humidity int,
    windSpeed real,
    windDirection int
);

insert into WeatherLog (recordedAt, locat
    values ('2023-06-23 13:18:03', 'Deva
```



WeatherLog

Column Name	Data Type	Allow Nulls
id	bigint	☐
recordedAt	datetime	✓
location	nvarchar(35)	✓
description	nvarchar(75)	✓
temperature	real	✓
humidity	int	✓
windSpeed	real	✓
windDirection	int	✓
		☐

The database has one table with 8 columns, the first one being an identity which is also primary key. I limited its size to 250MB, so as not to grow indefinitely.

For security reasons, I do not connect to the db with “sa”; I created a new “automation” user which had db_owner role and can only connect to this one database. The password for automation is used in cleartext in the Powershell script that connects to the database, so make sure to protect your “sa” user.

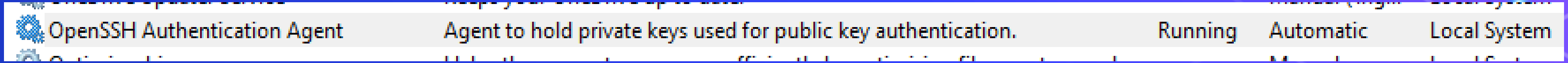
This database grows with 120 rows every day.

4. Network and security

Network

The networking presents a difficulty: how to connect automatically from Windows to Linux? This is done using ssh via private / public key pair. For this, the basic steps are:

- Install **OpenSSH** component for Windows 10/11. It's an *optional Windows component*, so you may already have it. It has a server and a client, but I used the client (the Linux machine is the ssh server).



OpenSSH Authentication Agent	Agent to hold private keys used for public key authentication.	Running	Automatic	Local System

- Use Powershell or cmd to **generate a public / private rsa key pair** with the command **ssh-keygen**. The private key is kept on the Windows machine and the public key is transferred to Linux. Instead of using a password, the ssh connection will compare the 2 keys. If they are identical, the connection will be allowed **without a password**.
- Use the following command to add the public key to the authorized_keys file of the Linux user you are trying to connect to (in this case: automation)

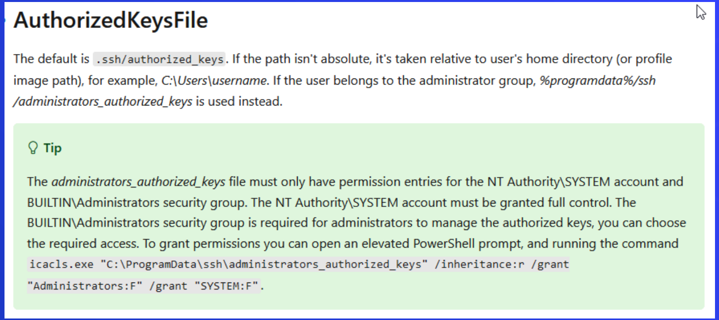
```
type $env:USERPROFILE\.ssh\id_rsa.pub | ssh {IP-ADDRESS-OR-FQDN} "cat >> .ssh/authorized_keys"
```

- The whole tutorial is found here: **https://chrisjhart.com/Windows-10-ssh-copy-id/**
- Once this is done, the **ssh** and implicitly, **scp** commands work without a password and are, therefore, able to be executed automatic by a Powershell script

Network

Why Windows -> Linux connection and not the other way around?

- It is easier. Linux integrates an ssh server by default
- It makes more sense. SSH connection is from client to server and Linux is the server – the computer that is always on, the server that always listens on port 22
- If you want to connect from Linux to Windows (which is, of course, doable with ssh), **there are some extra steps in case your Windows account is an administrator** – and it probably is. This needs a proper tutorial, but you can see some info here: **https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_server_configuration** It's the source of the photo below



AuthorizedKeysFile

The default is `.ssh/authorized_keys`. If the path isn't absolute, it's taken relative to user's home directory (or profile image path), for example, *C:\Users\username*. If the user belongs to the administrator group, *%programdata%/ssh/administrators_authorized_keys* is used instead.

Tip

The *administrators_authorized_keys* file must only have permission entries for the NT Authority\SYSTEM account and BUILTIN\Administrators security group. The NT Authority\SYSTEM account must be granted full control. The BUILTIN\Administrators security group is required for administrators to manage the authorized keys, you can choose the required access. To grant permissions you can open an elevated PowerShell prompt, and running the command `icacls.exe "C:\ProgramData\ssh\administrators_authorized_keys" /inheritance:r /grant "Administrators:F" /grant "SYSTEM:F"`.

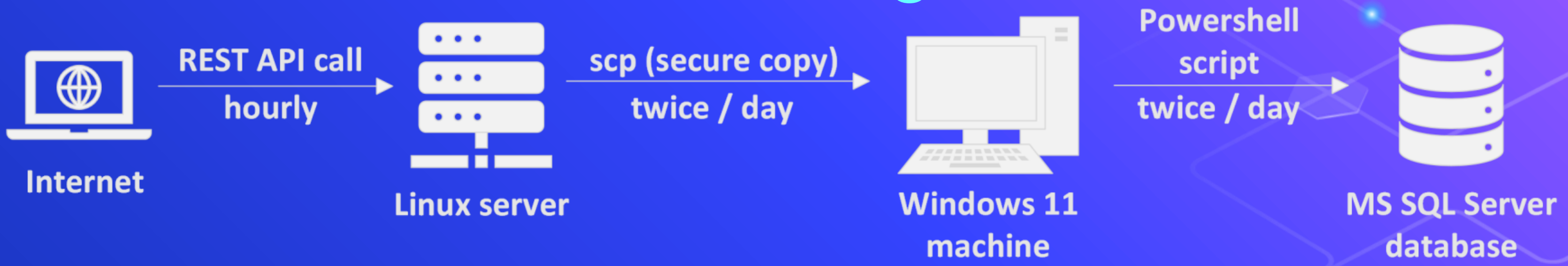
Security

Always have security and good practices in mind. The security measures taken for this project are:

- Use **try / catch** statements whenever a block of code may not run as intended. I used it for the API call to openweathermap.org. The API key may be wrong or no longer working, the server may be unresponsive, internet connection bugs... You do not want the script running indefinitely and / or possibly ruin the whole automation loop
- Avoid using administrator / root accounts whenever possible.
- I used a non-administrator ("automation") account for the SQL server + the account only has access to one database. One of the reasons is that its password is used in cleartext in the Powershell script that connects to the database
- I used a non-administrator ("automation") account on Linux – with the proper rights to access folder, to write to file, to delete a file –, which are a headache to manage, as it is customary in Linux.

5.
How does it work again?

The logic



Linux operations 5x / hour:

- Look in the project folder
- If **“Transfer.txt” file present** (meaning previous data has been successfully transferred to Windows), then erase the Transfer.txt file and also erase the **WeatherList.txt** file containing now-useless data. Run the API call, re-create the **WeatherList.txt** file and write data in it.
- If **“Transfer.txt” file not present**: simply run the API call and **append** the data to **WeatherList.txt**

Windows operations 2x / day:

- Copy WeatherList.txt file from Linux
- Put Transfer.txt file in the Linux folder **to mark the successful transfer of data** – so that the Linux script knows to erase old data.
- Connect to the SQL database and write the data from the WeatherList.txt file

If the computer is not running and it misses 1,2...100 scheduled task runs, **nothing bad happens**. Data just accumulates on the Linux server in the file, awaiting next transfer.

Thanks!

All the files can be found in the public repository: https://github.com/adisirbu/SQL-Project-Public