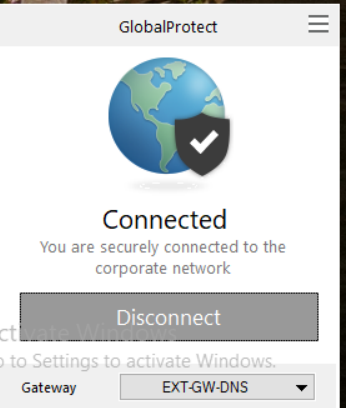
**Testing Virtual machine performance with PostgreSQL database in Cloud Stack with 4 and 8GB of RAM, 4CPU cores and 40GB of hard drive.**

“In this documentation you will learn how to apply multi-processing in Python using 5 different PostgreSQL queries. I am going to show how to connect to the PostgreSQL database that I created in two virtual machines in Cloud Stack. I am running 1, 10, 20, 50 and 100 processes simultaneously with the python code using each query on two different virtual machines. The first virtual machine specifications are 4 CPU cores, 4GB of RAM and 40GB of hard drive space. The second virtual machine specifications are 4 CPU cores, 8GB of RAM and 40 GB of hard drive space. I am only changing amount of RAM to see the difference in both VM’s performance. According to the goal of the project, the amount of CPU cores and hard drive space is limited to 4 and 40GB accordingly.”

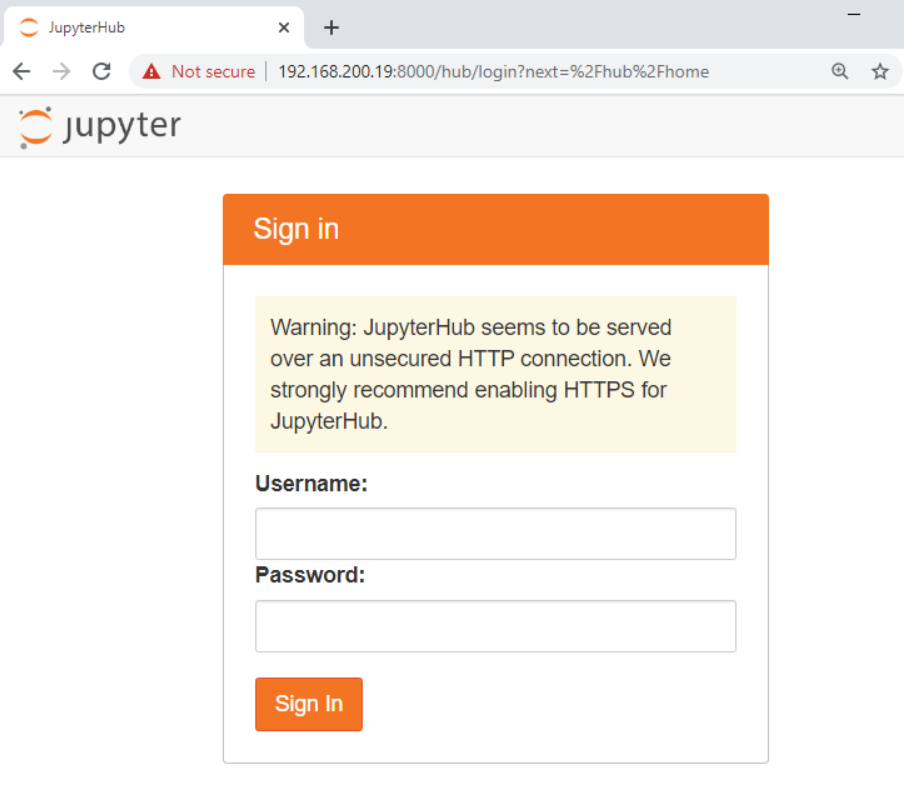
**Step 1. Connect to Global Protect**

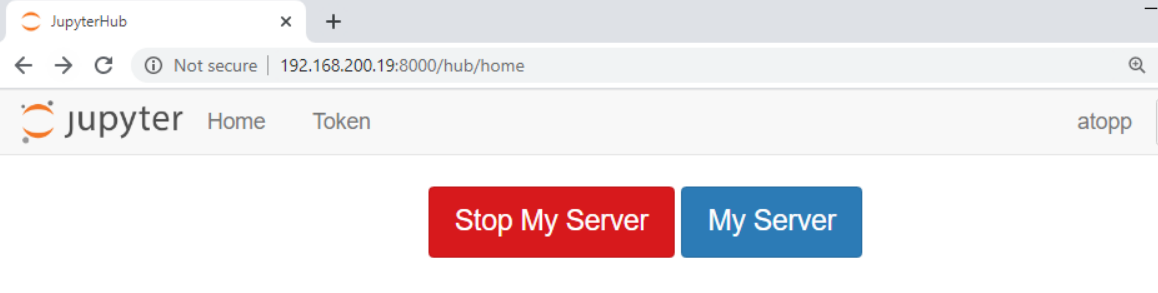
To connect to the virtual machine on Cloud Stack, connect to the private network Global Protect first on PC or Windows Virtual Machine like shown below.



**Step 2. Connect to Jupiter Hub (cobalt)**

Open the browser and paste the IP address (192.168.200.19:8000) to be connected to the JupyterHub. Then log in with your username and password with cobalt account. Then proceed with “My Server” and the Jupyter environment will be ready.





**Step 3.** **Install python libraries**

To work with PostgreSQL in Jupiter Lab, the psycopg2 and pygresql should be installed. Working in Jupyter Notebook, it can be done with pip install like on the print screen below (in Jupyter Notebook):



Although, working in Jupiter Hub in cobalt, it should be installed in cobalt. In order to do it, follow the steps below:

* Log in to cobalt through ssh (PuTTY) with root user and its password, then follow 3 commands below for installation:
* [root@cobalt ~]# yum install python3-devel
* [root@cobalt ~]# pip3 install psycopg2
* [root@cobalt ~]# pip3 install pygresql

**Step 4.** **Connect to the postgres database on virtual machine in CloudStack in Python**

First, import two libraries before writing the code in Jupyter Hub. To work with PostgreSQL and multi-processing import psycopg2 and multiprocessing:

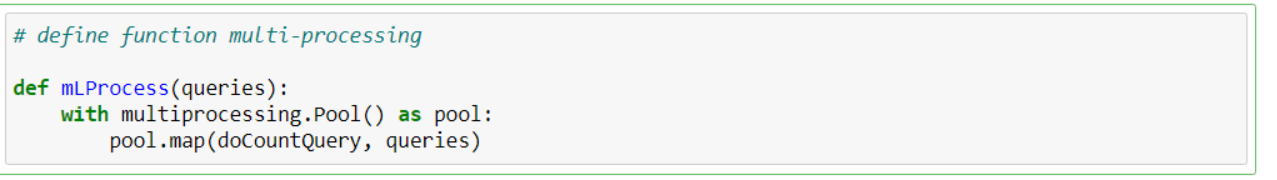


Second, create function which connects to PostgreSQL database with psycopg2 using host, port, database name and password (if the database was created with no password, it should not be included). Below print screen shows the connection function with an existing virtual machine and psql parameters that was used in the project:



**Step 5.** **Multi-processing**

Function below calls queries with several users by including connection function and each query name in PSQL:



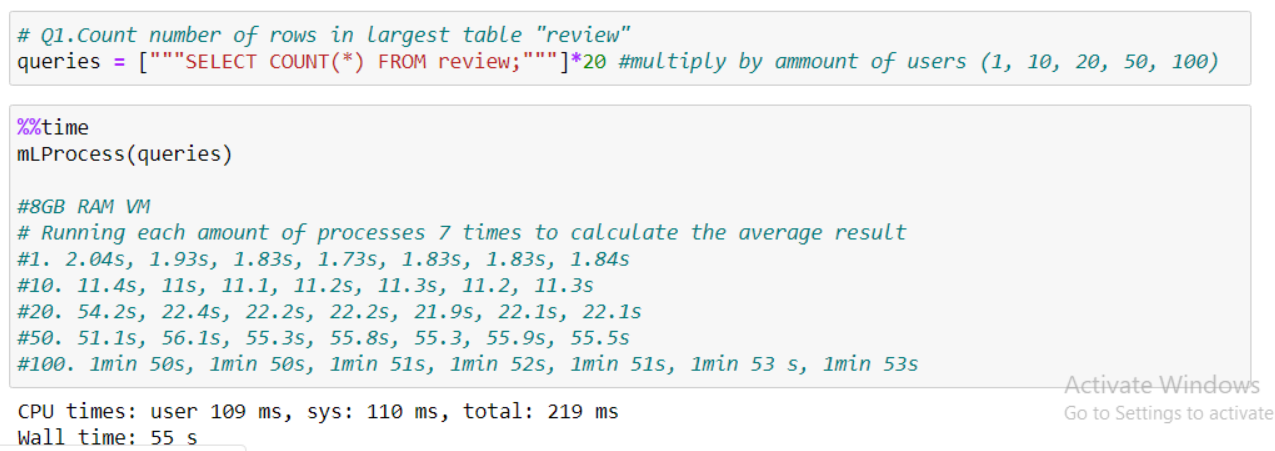
**Step 6. Testing queries with multi-processing and documenting results (run time) in the comments**

Since there are created functions with connection and multiprocessing function, define all 5 queries and test several processes by calling the functions like shown below for each query. Time the runs when calling the multiprocessing function with magic function %%time which shows wall time for documentation. By multiplying each query run by number of users (processes), the run time for that amount of processes will be received:

**Query 1 test**

Count number of rows in the largest table (“review”).

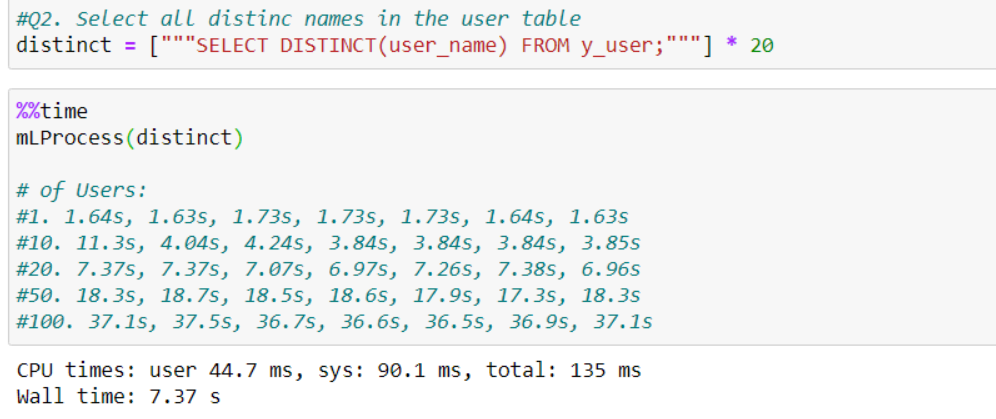
Define new name for the first query and include number of processes, then call multiprocessing function mLProcess(). After that CPU times and wall time will be shown. Wall time will be recorded after finding out the average. In this project, all queries were run without an output.



**Query 2 test**

Select all distinct names in the user table.

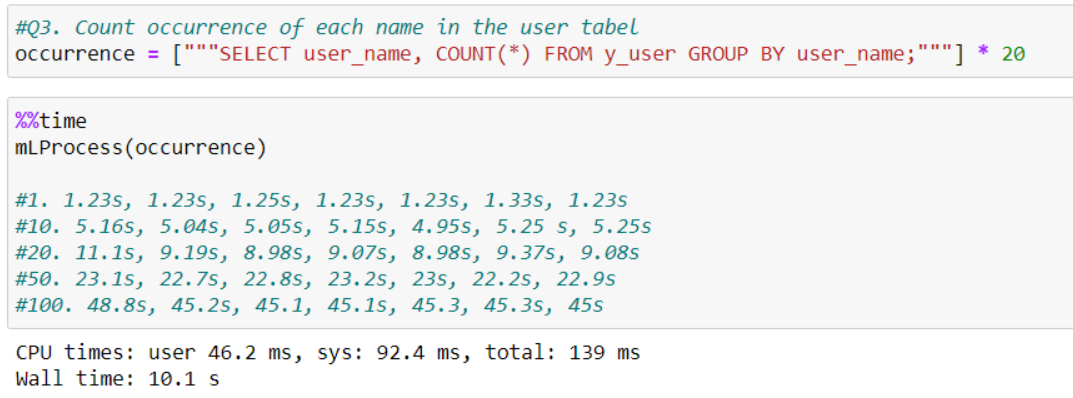
Define new name for the second query and include number of processes, then call multiprocessing function mLProcess():



**Query 3 test.**

Count occurrence of each name in the user table.

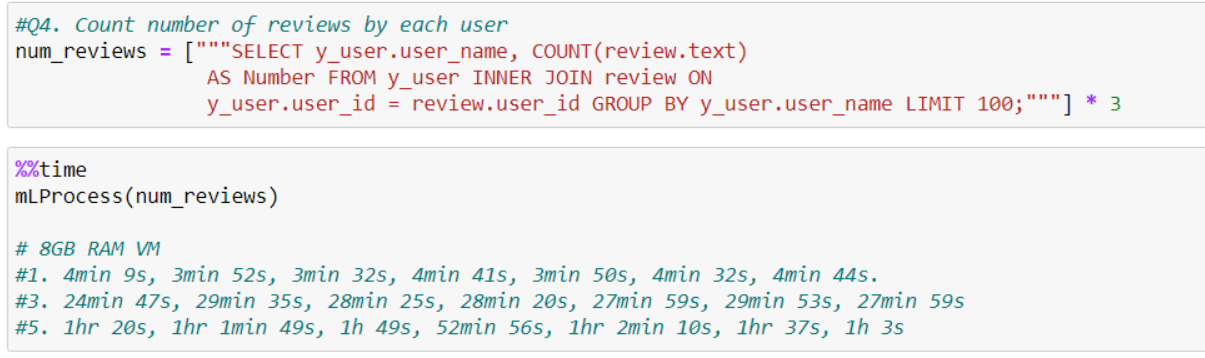
Define new name for the second query and include number of processes, then call multiprocessing function mLProcess():



**Query 4 test.**

Count number of reviews by each user using join.

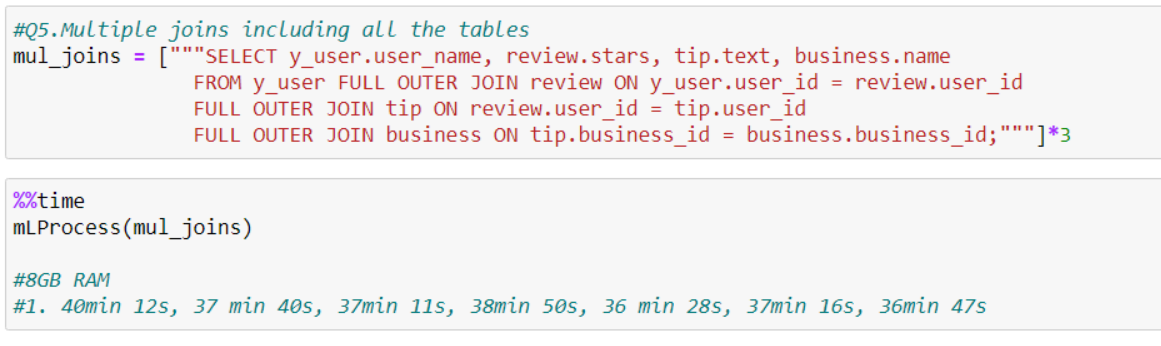
Define new name for the second query and include number of processes, then call multiprocessing function mLProcess():



**Query 5 test**

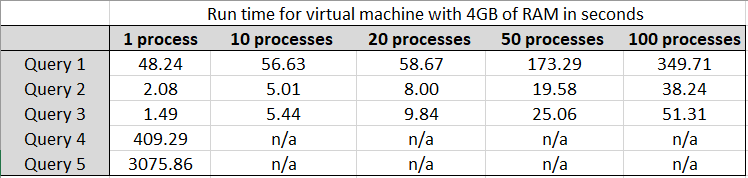
Create any query to try including all the tables with multiple joins.

Define new name for the second query and include number of processes, then call multiprocessing function mLProcess():



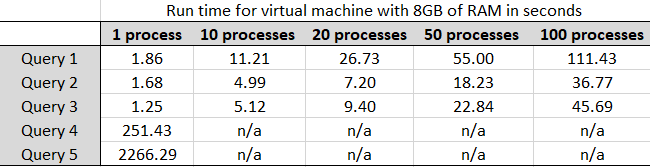
**Step 7. Graphing the results in excel**

Below there is a table with all the average run times for each query and processes that have been tested. Since the virtual machine contains only 40GB of hard drive, it is impossible to run queries 4 and 5 because they contain joins and require temporary space on the hard drive. To run those queries with 10, 20, 50 and 100 processes, the VM would need up to 500GB of free disk space.



**Step 8. Creating new virtual machine and repeating the whole process with the new VM.**

The new virtual machine has the same amount CPU cores and hard drive space, but different RAM size (8GB). Run time (performance) of the same queries will be analyzed and compared to the 4GB RAM virtual machine. After that, the run time table and graph for the new VM will be compared to the data from the 4GB RAM VM .



Looking at the graph, there is gradual raise of time while running query 2 and 3, but when running query 1 there is a rapid rise in time as the number of processes increase.

**Comparison of average run time between virtual machine with 4GB of RAM and virtual machine with 8 GB of RAM**

Since all 5 queries were run in both VMs, they could be compared for better performance. Each query will be compared for performance.

**Query 1:**

Looking at the query 1 graph between first and the second VMs, there is a huge difference. This query requires a lot of work because it is counting every rows of a 3.5 GB data in a table. There is a significant difference in the performance between the two VMs especially when running 1 vs 100 processes.

There is a noticeable difference running query 2, especially while running 50 and 100 processes.

To run this query, it takes the least amount of time in comparison with other queries, so the difference between processes and VMs is not very big. Although, it increases in 50 and 100 processes.

Since query 4 with joins is unable to run with 10, 20, 50 and 100 processes, run time with 1, 3 and 5 processes were recorded. The virtual machine with 8GB of RAM has a much better performance as expected, especially with larger tables and queries with joins. The more time it takes a query to run, the better the VM with 8GB of RAM will perform.

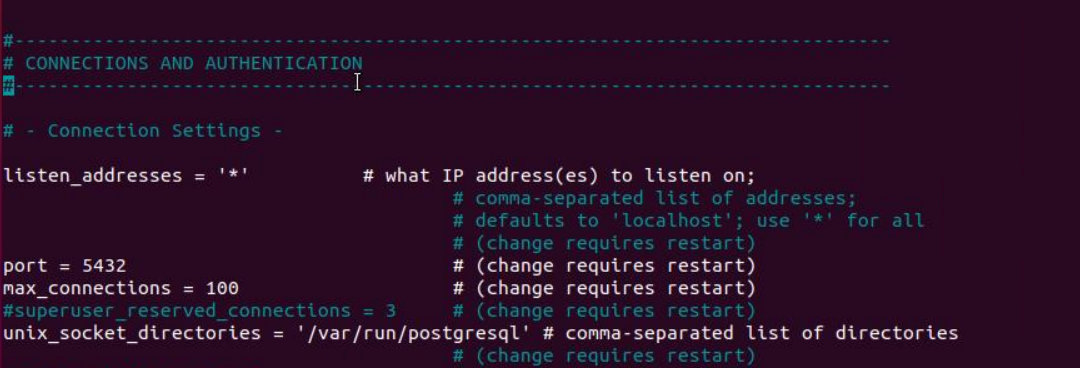
Again, there is a significant difference between running the same query in the two VMs. The 4GB RAM VM showed 51 min of run time approximately and the 8GB RAM VM showed 38 min of run time. The difference is over 10 minutes!

**Issues while connecting to the VM database in Python.**

When running the connection function with VM IP address, most likely an error connecting to the database will be received when working with Ubuntu VM. In this case, the problem is in the configuration PostgreSQL files on the virtual machine it is connecting to. In order to fix that problem, follow the steps below:

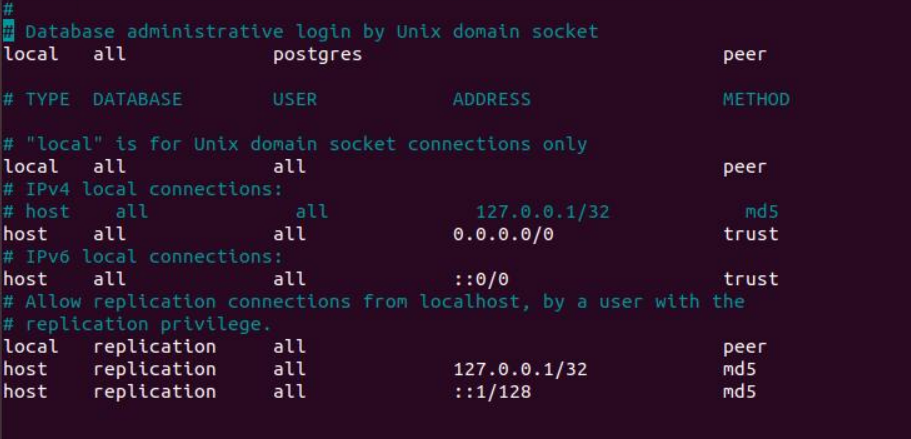
1. Open the virtual machine in CloudSack
2. In Linux terminal open the files postgresql.conf and change listen\_addresses = ‘\*’ and uncomment it. It allows to listen to all IP addresses.





1. Then open file pg\_hba.conf with sudo nano command and changes a few lines in the file:





After updating both files, restart PostgreSQL and try to connect to the database in python.



References

Claria, F., Claria, F., & Axones. (2018, December 04). An Introduction to Using SQL Aggregate Functions with JOINs. Retrieved from <https://academy.vertabelo.com/blog/introduction-using-aggregate-functions-joins/>

DaniloDanilo 2, WolphWolph 63.7k88 gold badges113113 silver badges138138 bronze badges, Raj MoreRaj More 42.6k2929 gold badges118118 silver badges188188 bronze badges, AdaTheDevAdaTheDev 119k2424 gold badges175175 silver badges180180 bronze badges, Marc\_smarc\_s 632k143143 gold badges12001200 silver badges13331333 bronze badges, & Alex HowanskyAlex Howansky 41k55 gold badges6262 silver badges8989 bronze badges. (1960, June 01). Count number of users from a certain country. Retrieved from <https://stackoverflow.com/questions/3773351/count-number-of-users-from-a-certain-country>

Mitrani, A. (2019, November 08). Python and PostgreSQL: How To Access A PostgreSQL Database Like A Data Scientist. Retrieved from <https://towardsdatascience.com/python-and-postgresql-how-to-access-a-postgresql-database-like-a-data-scientist-b5a9c5a0ea43>

Multi-Table Joins. (n.d.). Retrieved from http://etutorials.org/SQL/Postgresql/Part I General PostgreSQL Use/Chapter 1. Introduction to PostgreSQL and SQL/Multi-Table Joins/

PostgreSQL - JOINS. (n.d.). Retrieved from <https://www.tutorialspoint.com/postgresql/postgresql_using_joins.htm>

Python PostgreSQL - Limit. (n.d.). Retrieved from <https://www.tutorialspoint.com/python_data_access/python_postgresql_limit.htm>