Sergiu Buhatel

BlazingSQL on NVIDIA DGX A100

Table of Contents

[1. Introduction 1](#_Toc193552751)

[1 Installation 1](#_Toc193552752)

[2 Run 2](#_Toc193552753)

[2.1 Launch 2](#_Toc193552754)

[2.2 Stop 2](#_Toc193552755)

[2.3 Copy files 3](#_Toc193552756)

[2.4 GPUs usage 3](#_Toc193552757)

[2.5 Import data 3](#_Toc193552758)

[2.6 run\_query.py 5](#_Toc193552759)

[3 References 6](#_Toc193552760)

# Introduction

BlazingSQL is a SQL database that leverages the usage of GPUs instead of the CPUs.

This document describes the minimal steps to install and run BlazingSQL on NVIDIA DGX A100 machine.

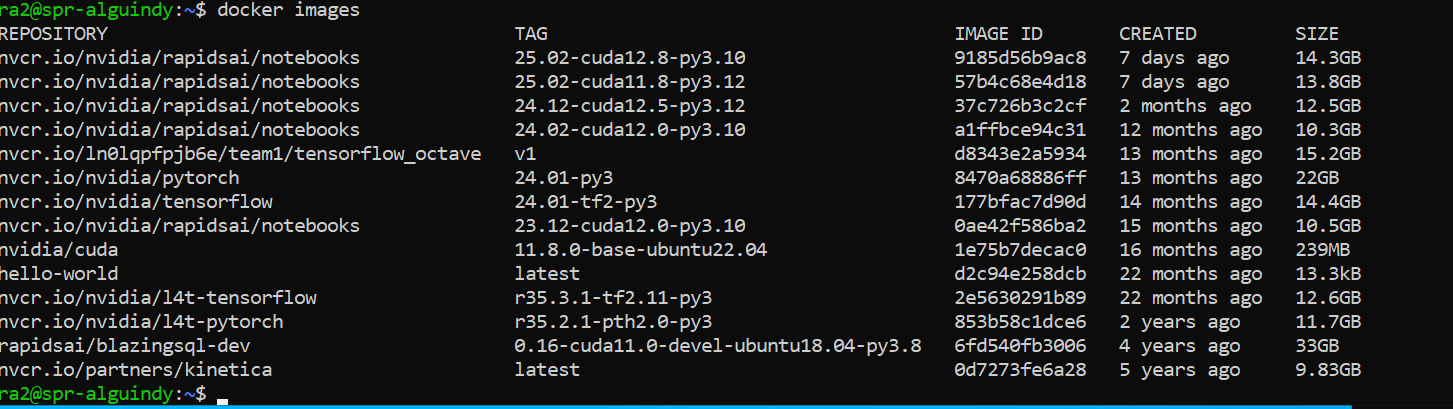
# Installation

Pull the image to the machine

docker pull rapidsai/blazingsql-dev:0.16-cuda11.0-devel-ubuntu18.04-py3.8

Check if the image was already pulled

docker images



# Run

## Launch

docker run --gpus all -v /home/ra2:/mnt/data -it --rm rapidsai/blazingsql-dev:0.16-cuda11.0-devel-ubuntu18.04-py3.8 bash

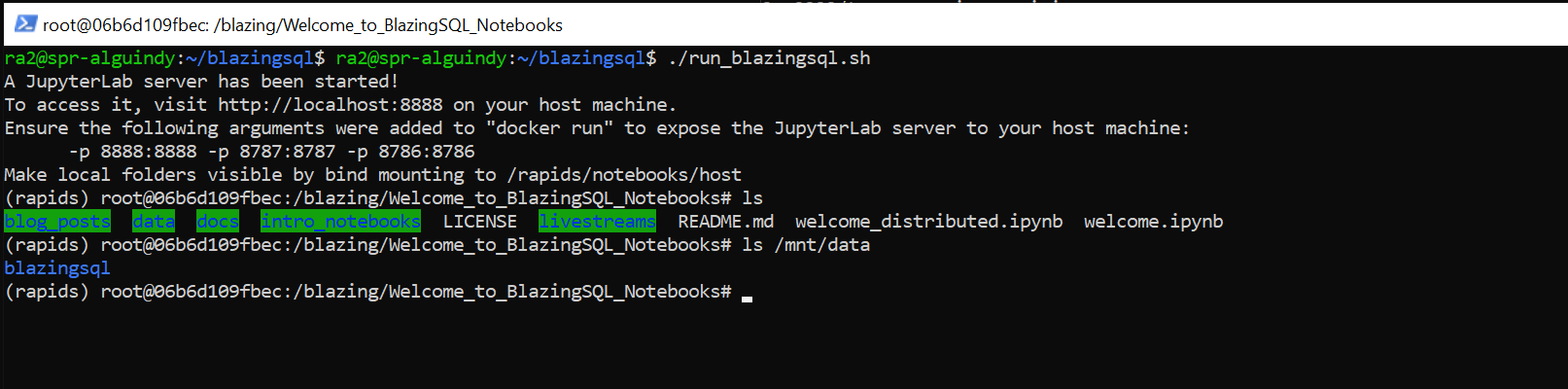
/home/ra2 is a directory on the machine, and /mnt/data is a mount inside the running container.

The above command line can be store in a .sh file, for easy usage. For example, is stored inside /home/ra2/blazingsql/run\_blazingsql.sh

Can be launched

cd /home/ra2/blazingsql

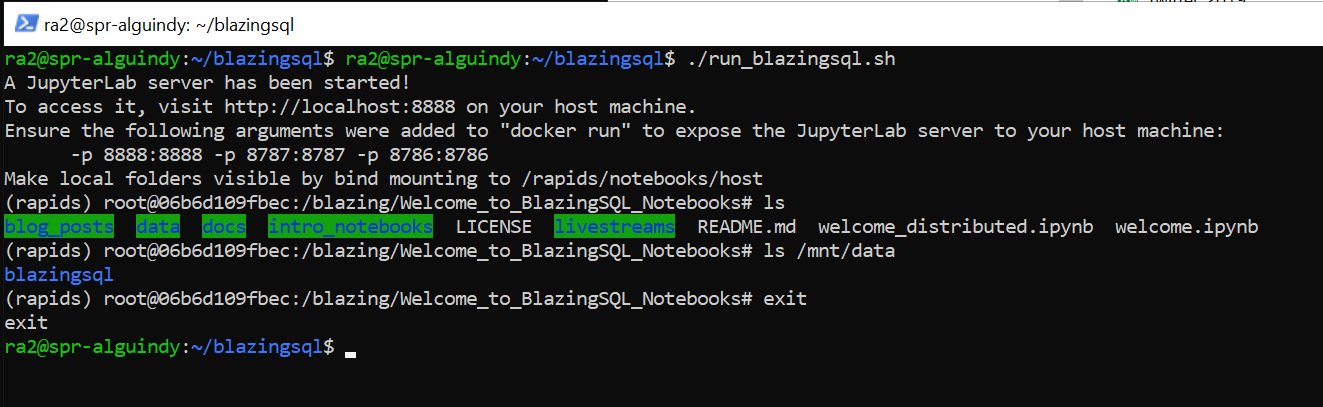
./run\_blazingsql.sh



As we can see, /mnt/data inside the running container, is mirroring /home/ra2 on the machine.

## Stop

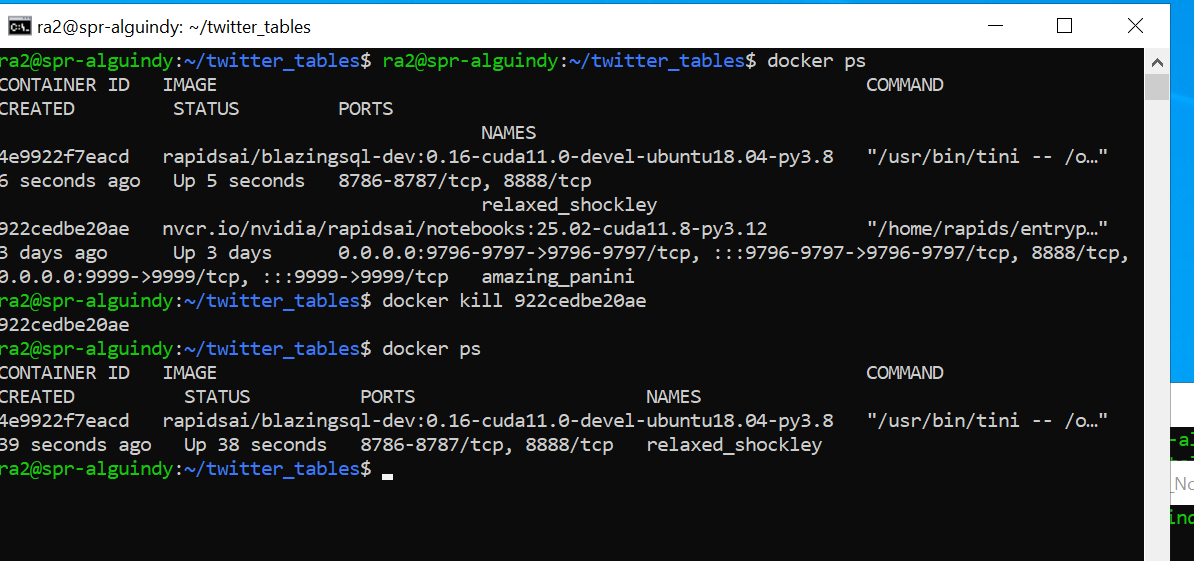
To exit the running container, type exit.



Another way is to find the running container in a different command shell and stop it explicitly.

docker ps

docker kill <container id>

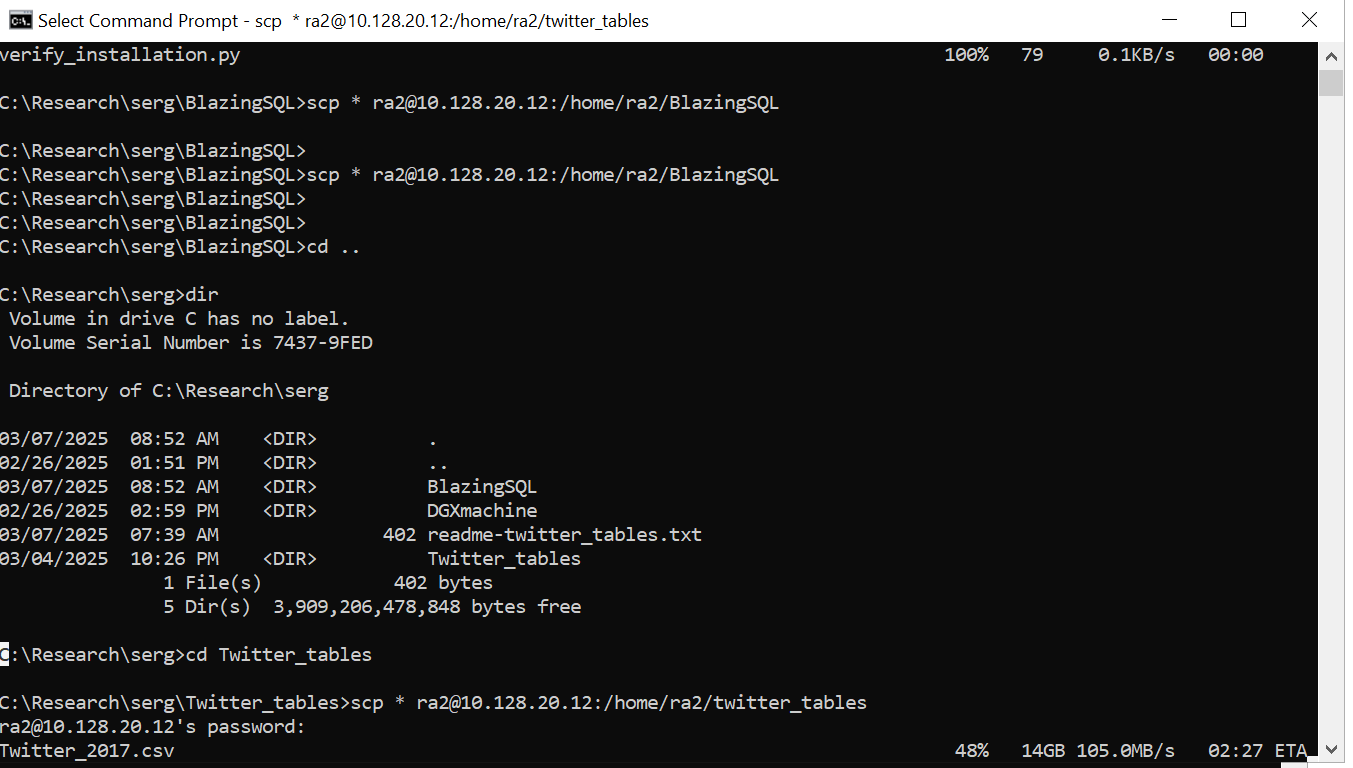


## Copy files

To copy files from Windows machine to Nvidia DGX machine, open a command prompt on windows and change directory to the directory containing the files you want to copy.

cd C:\Research\serg\Twitter\_tables

scp \* ra2@10.128.20.12:/home/ra2/twitter\_tables



## GPUs usage

To continuously check GPUs usage, open a shell command and type:

watch -n 1 nvidia-smi

Press Ctrl/C to exit.

## Import data

There is an issue running the imported full file of fixed\_Twitter\_2017.csv. The select is hanging. The total number of lines for fixed\_Twitter\_2017.csv is 35.5 million.

cd /home/ra2/twitter\_tables

wc -l fixed\_Twitter\_2017.csv

38514699 fixed\_Twitter\_2017.csv

I created a file including the first 21 million records, and the import and select respond right away. As soon as I pass the 22 million records, the select is hanging. For that reason, I created Twitter\_2017\_first\_21\_mil.csv and put it into this directory: /home/ra2/smaller\_twitter\_tables. Bellow is how to import the file and run it.

cd /home/ra2/smaller\_twitter\_tables

wc -l Twitter\_2017\_first\_21\_mil.csv

21000000 Twitter\_2017\_first\_21\_mil.csv

# Launch BlazingSQL

cd /home/ra2/blazingsql

./run\_blazingsql.sh

# Once on BlazingSQL is launched, the # prompt will appear

# Change directory to the mounted directory that contains the import code and the twitter files

# Recall that /home/ra2 is mounted as /mnt/data when BlazingSQL is launched

cd /mnt/data/smaller\_twitter\_tables/

# Import and run query. Specify as the first parameter the csv to be imported and as the second parameter

# the table that needs to be created.

python run\_query.py Twitter\_2017\_first\_21\_mil.csv Twitter\_2017\_first\_21\_mil

A computer screen shot of a black screen

AI-generated content may be incorrect.

## run\_query.py

import time

import argparse

from blazingsql import BlazingContext

import cudf # RAPIDS cuDF for GPU-accelerated dataframes

# Command-line argument parser

parser = argparse.ArgumentParser(description="Load a CSV file into BlazingSQL and query it.")

parser.add\_argument("file", help="Path to the CSV file.")

parser.add\_argument("table", help="Name of the table to create.")

args = parser.parse\_args()

bc = BlazingContext()

start\_time = time.time()

bc.create\_table(args.table, args.file)

end\_time = time.time()

print(f"Time taken to load the file: {end\_time - start\_time} seconds")

query = f"SELECT 'AFL' AS AFL, screen\_name4, COUNT(\*) AS numConnections" \

f" FROM {args.table}" \

f" WHERE screen\_name4 <> 'AFL'" \

f" AND tweet\_text LIKE '%AFL%'" \

f" GROUP BY screen\_name4"

# This one has unsupported SQL features for BlazingSQL, and therefore, the above SQL command was used

# as a close equivalent.

#query = f"SELECT \"AFL\", screen\_name4, count(\*) as numConnections FROM {args.table} " \

# f"WHERE screen\_name4!= 'AFL' AND tweet\_text REGEXP '\\$\\bAFL\\$' group by screen\_name4"

result = bc.sql(query)

print(result)

# References

<https://hub.docker.com/r/rapidsai/blazingsql-dev>

<https://towardsdatascience.com/intro-to-blazingsql-in-5-minutes-172162d3a95/>

<https://developer.nvidia.com/blog/beginners-guide-to-querying-data-using-sql-on-gpus-in-python/>

ss