Sergiu Buhatel, Research Assistant – Computing/ML

MySQL (Windows) to OmniSci (Ubuntu) Migration on NVIDIA DGX A100

Table of Contents

[1 Introduction 2](#_Toc197784857)

[2 MySQL 2](#_Toc197784858)

[3 OmniSci 3](#_Toc197784859)

[3.1 NVIDIA DGX A100 3](#_Toc197784860)

[3.2 Install 3](#_Toc197784861)

[3.3 Configuration 4](#_Toc197784862)

[3.3.1 omnisci.conf 4](#_Toc197784863)

[3.4 Run 4](#_Toc197784864)

[3.4.1 launch.sh 4](#_Toc197784865)

[3.4.2 Stopping and Restarting 4](#_Toc197784866)

[3.5 SQL 5](#_Toc197784867)

[4 Import 5](#_Toc197784868)

[4.1 SSH tunnel 5](#_Toc197784869)

[4.2 Step 1 – Import all fields as Text 6](#_Toc197784870)

[4.2.1 Virtual Environment 6](#_Toc197784871)

[4.2.2 Activate the virtual environment 6](#_Toc197784872)

[4.2.3 Launch the Import 6](#_Toc197784873)

[4.2.4 Check the Tables 6](#_Toc197784874)

[4.3 Step 2- Create tables with the right field types 7](#_Toc197784875)

[4.3.1 Virtual Environment 7](#_Toc197784876)

[4.3.2 Activate the virtual environment 7](#_Toc197784877)

[4.3.3 Launch the Import 7](#_Toc197784878)

[4.3.4 Check the Tables 8](#_Toc197784879)

[4.3.5 Target Query 8](#_Toc197784880)

[5 GitHub 8](#_Toc197784881)

[6 References 8](#_Toc197784882)

# Introduction

This paper outlines the process of migrating a MySQL database from a Windows environment to an OmniSci database running on Ubuntu. The target system is an NVIDIA DGX A100 machine equipped with 8 GPUs.

The primary motivation for this migration is to take advantage of GPU acceleration to significantly improve query performance. The current MySQL setup on Windows suffers from slow query execution times. By transitioning to OmniSci on the DGX A100, the system can leverage the parallel processing power of GPUs, resulting in faster query performance.

# MySQL

The MySQL database is currently hosted on a Windows machine with the IP address 134.117.176.72. This server is equipped with 768 GB of RAM and runs Windows Server 2022 Standard. Despite its substantial hardware resources, query performance remains suboptimal.

The database can be accessed using MySQL Workbench by connecting to the blazing\_sql schema. This schema name reflects an earlier migration attempt to BlazingSQL. However, due to certain limitations encountered with BlazingSQL, OmniSci was chosen as a more reliable and performant alternative.

A screenshot of a computer

AI-generated content may be incorrect.

The connection details for the MySQL database are as follows:

* **Username:** root
* **Password:** root
* **Port:** 3307
* **Database Name:** blazing\_sql

# OmniSci

## NVIDIA DGX A100

The NVIDIA DGX A100 machine is accessible at IP address **10.128.20.12**. It is equipped with 1 TB of RAM and 8 GPUs.

From the Windows machine at IP address 134.117.176.72, you can connect to the NVIDIA DGX A100 using the following SSH command:

ssh <username>@10.128.20.12

In my case, since my username is ra2, I ran:

ssh ra2@10.128.20.12

The DGX A100 runs **Ubuntu 20.04.6 LTS**, which can be verified by executing:

lsb\_release -a

To transfer files from the current directory of the Windows machine to the Ubuntu machine, you can use the scp command. For example:

scp \* <username>@10.128.20.12:/home/<username>

In my case, since my username is ra2, I ran:

scp \* ra2@10.128.20.12:/home/ra2

This command copies all files from the current Windows directory to the /home/<username> directory on the DGX A100. Ensure that the target directory exists before executing the command; if it does not, create it beforehand to avoid errors.

## Install

To install OmniSci using Docker, run the following command:

sudo docker pull omnisci/core-os-cuda:latest

To confirm that the image has been pulled successfully, list available Docker images and filter for OmniSci:

docker images | grep omnisci

## Configuration

cd /home/ra2/omnisci/omnisci-docker-storage

vi omnisci.conf

### omnisci.conf

allowed-import-paths = ["/data/twitter\_tables"]

This configuration enables bulk data import from the Docker container directory /data/twitter\_tables when using the COPY FROM SQL command within omnisql. If additional directories are required, they can be added as a comma-separated list. To apply the changes, the OmniSci Docker container must be restarted. As the configuration file has already been created, no further modifications are currently needed.

## Run

cd /home/ra2/omnisci

./launch.sh

This will launch OmniSci, provided is not running yet.

### launch.sh

sudo -E docker run --runtime=nvidia \

--name omnisci \

-v /home/ra2/omnisci/omnisci-docker-storage:/omnisci-storage \

-v /home/ra2:/data \

-p 6273-6280:6273-6280 \

omnisci/core-os-cuda:latest

This configuration mounts the local directory /home/ra2/omnisci/omnisci-docker-storage into the docker container as /omnisci-storage, and the local directory /home/ra2 as /data inside the docker container.

The OmniSci database is stored persistently outside the container at /home/ra2/omnisci/omnisci-docker-storage, ensuring that data remains intact even if the container is removed or recreated.

### Stopping and Restarting

Follow these steps to stop a running OmniSci Docker container and restart it cleanly:

Check whether the container is running. If it is, stop the container and then remove it using its container ID.

docker ps | grep omnisci

docker stop <container id>

docker rm <container id>

Then relaunch the container.

cd /home/ra2/omnisci

./launch.sh

## SQL

Access the running Docker container and start OmniSQL:

docker exec -it omnisci bash

./bin/omnisql

When prompted for a password, enter: HyperInteractive

To list all tables: \t

To view details of a specific table: \d <table\_name>

At the prompt, you can run any SQL command—just be sure to end each command with a semicolon (;).

# Import

I attempted several methods to import the data, but none were fully successful.

First, I tried importing CSV files that were previously exported from MySQL. While I was able to fix many formatting issues, one persistent problem remained: some records were split across multiple lines in the CSV files. Despite various attempts to clean the data, I couldn't fully resolve this issue.

Next, I exported the MySQL tables to Parquet format and attempted to import them into OmniSci. Unfortunately, the import failed.

I then tried connecting the two databases directly, bypassing intermediate files. Then, I ran into problems mapping the data types correctly. To work around this, I performed the import in two steps: first, I imported all fields as text; then, I converted the data to the correct types. Since OmniSci SQL does not support type casting within SQL queries, I used pandas to perform the conversions.

## SSH tunnel

To allow the NVIDIA DGX A100 machine to access the MySQL database running on a Windows system, I set up an SSH tunnel:

ssh -L 3307:localhost:3306 [research\_assistant2@134.117.176.72](mailto:research_assistant2@134.117.176.72)

In this case, research\_assistant2 is the username on the Windows machine.

This allowed secure access to the remote MySQL database from the local environment.

## Step 1 – Import all fields as Text

### Virtual Environment

Create a virtual environment for python if this does not exist already.

cd /home/ra2/mysql

Python -m venv env

Pip install -r requirement

### Activate the virtual environment

cd /home/ra2/mysql

source env/bin/activate

### Launch the Import

This command launches the import process for a single table, tweets\_2017. It reads the tweets\_2017 data from MySQL, drops any existing version of the table in OmniSci, recreates it, and populates it with the imported data.

python import\_as\_text\_mysql\_to\_omnisci.py tweets\_2017

At this stage, the tables have already been created. All fields in the newly created tweets\_2017 table in OmniSci are of type TEXT.

The python code for **import\_as\_text\_mysql\_to\_omnisci.py** is available at: <https://github.com/sergiubuhatel/ra2/blob/main/MySQL/import_as_text_mysql_to_omnisci.py>

The file is also available on the machine at:

**/home/ra2/mysql/import\_as\_text\_mysql\_to\_omnisci.py**

Repeat the process for the remaining tables: tweets\_2018, tweets\_2019, tweets\_2021, tweets\_2022, and tweets\_2023.

Run the imports **sequentially**—not in parallel—as running them simultaneously overwhelm the machines and cause the processes to fail. I previously attempted a parallel import, and all processes failed as a result.

### Check the Tables

At this point, we should have the following tables available: tweets\_2017, tweets\_2018, tweets\_2019, tweets\_2021, tweets\_2022, and tweets\_2023.

To verify this, you can connect to OmniSci using the following commands:

docker exec -it omnisci bash

./bin/omnisql

Once inside the SQL shell, list all tables by typing: \t

To verify that each newly created OmniSci table contains the same number of records as its corresponding MySQL table, follow these steps:

On the Windows machine (using MySQL Workbench):

select count(\*) from blazing\_sql.tweets\_2017;

On the Ubuntu machine (inside omnisql):

select count(\*) from tweets\_2017;

Both queries should return the same result.

## Step 2- Create tables with the right field types

### Virtual Environment

Create a virtual environment for python if this does not exist already.

cd /home/ra2/omnisci

Python -m venv env

Pip install -r requirement

### Activate the virtual environment

cd /home/ra2/ omnisci

source env/bin/activate

### Launch the Import

Using the existing tweets\_2017 table in OmniSci, create a new table named Twitter\_2017 with field types that match those from the MySQL database. The following command will drop the Twitter\_2017 table if it already exists, recreate it with the correct schema, and then populate it with data from tweets\_2017.

python fix\_column\_types.py tweets\_2017 Twitter\_2017

Repeat the process for the remaining tables.

To ensure the commands run sequentially, you can place them all in a shell script and execute the script. This will help avoid overloading the system and ensure each step completes successfully.

The python code for **fix\_column\_types.py** is available at: <https://github.com/sergiubuhatel/ra2/blob/main/OmniSci/fix_column_types.py>

The file is also available on the machine at:

**/home/ra2/omnisci/fix\_column\_types.py**

### Check the Tables

At this point, we should have the following tables available: Twitter\_2017, Twitter\_2018, Twitter\_2019, Twitter\_2021, Twitter\_2022, and Twitter\_2023.

To verify this, you can connect to OmniSci using the following commands:

docker exec -it omnisci bash

./bin/omnisql

Once inside the SQL shell, list all tables by typing: \t

First, ensure that the number of imported records in each of the nely created tables are the same as in the original table by comparing the results of the two SQL queries:

select count(\*) from Twitter\_2017;

select count(\*) from tweets\_2017;

Both queries should return the same result.

### Target Query

Run this query for Twitter\_2017:

select 'AFL' AS AFL, screen\_name4, COUNT(\*) AS numConnections from Twitter\_2023 WHERE screen\_name4 <> 'AFL' AND tweet\_text LIKE '%AFL%' GROUP BY screen\_name4;

Repeat the same SQL query for all other tables. Each query takes only a few seconds to run, and some may return results almost instantly.

# GitHub

All the Python code, Bourne Shell scripts, Word document, and other related files are publicly available on GitHub at <https://github.com/sergiubuhatel/ra2>.

# References

<https://catalog.ngc.nvidia.com/orgs/partners/containers/omnisci-os>

<https://www.heavy.ai/>