Importing Premium DNS 365 into ClickHouse

Posted on May 30, 2024

This project aimed to upload data from DNS_Premium_365 dataset to a local ClickHouse database for efficient, optimized and rapid querying capabilities.

Approach

Decompressing

The Premium DNS files are very large, and decompressing them with a single-threaded tool such as gzip proves to take a long time. The uncompressed data can be over 1TB. Therefore, we recommend using a multi-threaded compression/decompression tool such as pigz, located at https://zlib.net/pigz/ Pigz will utilize multiple-cores, as many as the system has to offer, reducing decompression time by as much as 70%+.

Data Preprocessing

Utilizing shell scripts, we processed 17 .gz files from the DNS_Premium_365 dataset for March 2024. Each file's row count and size were logged post-decompression to validate data integrity and prepare for subsequent processing.

```bash
#!/bin/bash

src_dir="/mnt/vm-share/datafeeds/DNS_Database_Download_Premium365/2024-03-07"
temp_dir="/mnt/nvme2/temp"

mkdir -p "$temp_dir"

# 1-15
for i in {1..17}; do
    file="${src_dir}/premium_dns_database.2024-03-07.full.${i}.csv.gz"
    if [ -f "$file" ]; then
        echo "Processing file: $file"
        # unzip
        temp_file="${temp_dir}/$(basename "$file" .gz)"
        zcat "$file" > "$temp_file"
        # Row #
        rows=$(wc -l < "$temp_file")
        echo "Number of rows: $rows"

        # Size
        uncompressed_size=$(ls -l "$temp_file" | awk '{print $5}')
        echo "Uncompressed size: $uncompressed_size bytes"
    else
        echo "File does not exist: $file"
    fi
    echo "--------------------------------"
done
```

TLD Pre-Filtering (Optional)

Use shell script to filter our TLDs of interest from the decompressed files, generating corresponding domain name lists. This step was critical to ensure the quality of the data preparation phase.

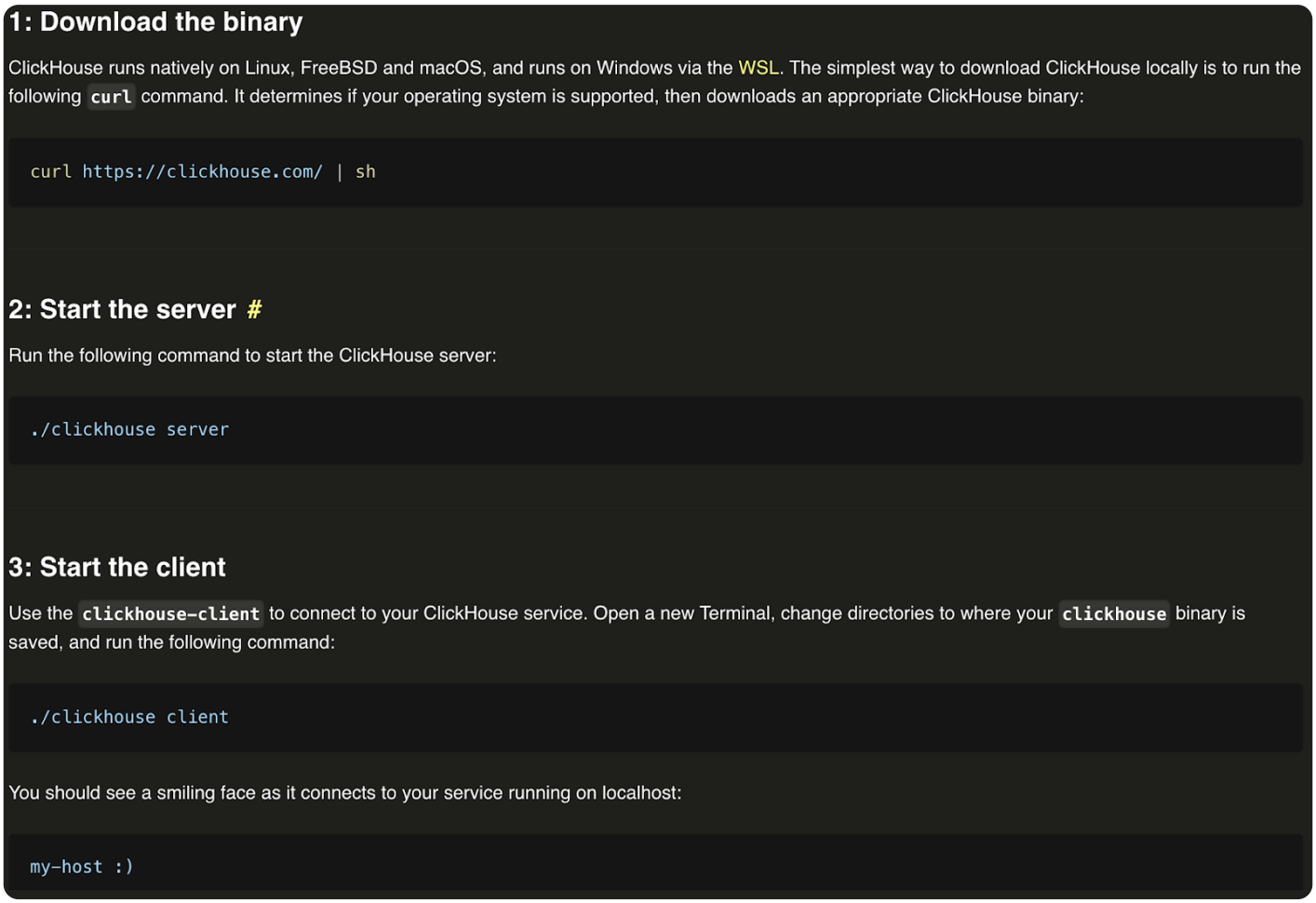
The TLDs we are using are: ["com", "org", "net", "ru", "de", "cn", "uk", "br", "jp", "ai", "io", "fr", "it", "ir", "il", "ua", "kp", "xyz", "online", "top", "shop", "site", "icu", "club", "store", "vip"]

```bash
#!/bin/bash

# Dir
src_dir="/mnt/nvme2/temp"

# TLDs
tlds=("com" "org" "net" "ru" "de" "cn" "uk" "br" "jp" "ai" "io" "fr"
      "it" "ir" "il" "ua" "kp" "xyz" "online" "top" "shop" "site"
      "icu" "club" "store" "vip")

# File prefix
file_prefix="premium_dns_database.2024-03-07.full"

# Process each file from 1 to 17
for i in {1..17}; do
    file="$src_dir/${file_prefix}.${i}.csv"
    echo "Processing file: $file"
    output_file="${file%.csv}_tlds_output.csv"
    echo "" > "$output_file"
    for tld in "${tlds[@]}"; do
        rg "\.$tld," "$file" >> "$output_file"
    done
    echo "Output saved to: $output_file"
    echo "--------------------------------"
done
```

ClickHouse Database Processing

Installation and Initialization

Please see the official documents for the installation process of ClickHouse at https://clickhouse.com/docs/en/getting-started/quick-start



Database Creation

Executed SQL commands to create the `production_db` database. By using command:
./clickhouse client --query="CREATE DATABASE production_db

Data Table Design and Import

Tables can be created according to customer needs and the data structure. Below is an example on how to create a table:

./clickhouse client --query="CREATE TABLE production_db.domain_new2 (domain String, timestamp UInt64, ip String) ENGINE = MergeTree() ORDER BY domain"

The script was utilized to automate the process for the 17 DNS files, it will iterate through the 17 files, creating corresponding tables and inserting data into them.

```
#!/bin/bash

CSV_DIR="/mnt/nvme2/temp"
CLICKHOUSE_CLIENT="/mnt/nvme/clickhouse/clickhouse client"
DATABASE_NAME="production_db3"

#Create database
echo "Creating database $DATABASE_NAME if it does not exist"
$CLICKHOUSE_CLIENT --query="CREATE DATABASE IF NOT EXISTS $DATABASE_NAME;"

for i in {1..17}
do
  TABLE_NAME="domain_$i"
  FILE_NAME="$CSV_DIR/premium_dns_database.2024-03-07.full.${i}.csv"

  echo "Creating table $TABLE_NAME if it does not exist"
  $CLICKHOUSE_CLIENT --query="CREATE TABLE IF NOT EXISTS $DATABASE_NAME.$TABLE_NAME (domain String, timestamp UInt64, ip String) ENGINE = MergeTree() ORDER BY domain;"

  if [ "$FILE_NAME" == "$CSV_DIR/premium_dns_database.2024-03-07.full.1.csv" ]; then
    echo "Starting import of $FILE_NAME into $DATABASE_NAME.$TABLE_NAME with skipping the first line"
    tail -n +2 "$FILE_NAME" | $CLICKHOUSE_CLIENT --query="INSERT INTO $DATABASE_NAME.$TABLE_NAME FORMAT CSV"
  else
    echo "Starting import of $FILE_NAME into $DATABASE_NAME.$TABLE_NAME without skipping the first line"
    $CLICKHOUSE_CLIENT --query="INSERT INTO $DATABASE_NAME.$TABLE_NAME FORMAT CSV" < "$FILE_NAME"
  fi
  echo "Finished importing $FILE_NAME into $DATABASE_NAME.$TABLE_NAME"
done
```

Use Case Example:

1. After uploading data to ClickHouse, we can select our database using the 'SHOW DATABASES' and 'USE your_database' commands in ClickHouse. We can also use 'SHOW TABLES' to view existing tables.

```
vm-dev1 :) use production_db3

USE production_db3

Query id: 4181558b-c530-42ab-87db-a19f2feeafdd

Ok.

0 rows in set. Elapsed: 0.001 sec.

vm-dev1 :) show tables

SHOW TABLES

Query id: 521ef255-13b5-442e-99dc-b4e81e266747

┌─name─────┐
│ domain_1 │
│ domain_10 │
│ domain_11 │
│ domain_12 │
│ domain_13 │
│ domain_14 │
│ domain_15 │
│ domain_16 │
│ domain_17 │
│ domain_18 │
│ domain_19 │
│ domain_2 │
│ domain_3 │
│ domain_4 │
│ domain_5 │
│ domain_6 │
│ domain_7 │
│ domain_8 │
│ domain_9 │
└──────────┘

19 rows in set. Elapsed: 0.004 sec.
```

2. In ClickHouse, SQL commands can be used for data analysis, such as searching for specific domains or IP addresses. For example, the command below retrieves 1.20 billion FQDNs with the .com top-level domain (TLD) in 8.343 seconds.

```
vm-dev1 :) SELECT COUNT(*)
FROM domain_1
WHERE domain like '%.com';

SELECT COUNT(*)
FROM domain_1
WHERE domain LIKE '%.com'

Query id: 5e6ca20b-c975-40b6-b539-d27573b0c4e7

   ┌─count()─┐
1. │ 1199463181 │ -- 1.20 billion
   └────────────┘

1 row in set. Elapsed: 8.343 sec. Processed 3.02 billion rows, 183.73 GB (361.73 million rows/s., 22.02 GB/s.)
Peak memory usage: 110.23 MiB.
```

3. Script could be used to iterate through all tables to find specific domains or IP addresses. The example below is to iterate through 19 tables to find ip that startswith 104.154.*.

```python
#!/usr/bin/env python3
import subprocess
import pandas as pd

CLICKHOUSE_CLIENT = "/mnt/nvme/clickhouse/clickhouse"
CLIENT_ARG = "client"
DATABASE_NAME = "production_db3"
OUTPUT_FILE = "/mnt/nvme2/temp/output_ips.csv"
BATCH_SIZE = 1000000

with open(OUTPUT_FILE, 'w') as file:
    file.write('fqdn,timestamp,ip\n')
    file.flush()

for i in range(1, 20):
    table_name = f"domain_{i}"
    print(f"Currently processing: {table_name}", flush=True)
    query = f"SELECT domain, timestamp, ip FROM {DATABASE_NAME}.{table_name} WHERE match(ip, '\\\\b104\\\\.154\\\\.\\d+\\\\.\\d+\\\\b') FORMAT CSV"
    print(f"Executing query: {query}", flush=True)

    process = subprocess.Popen([CLICKHOUSE_CLIENT, CLIENT_ARG, "--query", query], stdout=subprocess.PIPE, text=True)

    df = pd.read_csv(process.stdout, header=None, names=['domain', 'timestamp', 'ip'])
#    print(df.head())
    actual_rows = df.shape[0]
    print(f"Read {actual_rows} rows from output.", flush=True)


    if df.empty:
        print(f"No results in {table_name} for matching IP pattern", flush=True)
    else:
        df.sort_values(by='domain', inplace=True)
        results_count = len(df)
        print(f"Total results for {table_name}: {results_count}", flush=True)
        for start in range(0, results_count, BATCH_SIZE):
            end = start + BATCH_SIZE
            df_slice = df[start:min(end, results_count)]
            df_slice.to_csv(OUTPUT_FILE, mode='a', header=False, index=False)
            print(f"Written rows {start} to {min(end, results_count)} to {OUTPUT_FILE}", flush=True)

print("Processing complete.", flush=True)
```

Conclusion

In conclusion, this project successfully utilized ClickHouse and automation scripts to efficiently process and analyze a large volume of domain and IP address data. Leveraging SQL commands and scripts for automatic table creation and bulk data insertion ensured seamless data management. The ability to iterate through all tables to identify specific domains or IP addresses demonstrated significant analytical capabilities. Moving forward, these methods can be scaled and adapted to handle even larger datasets, reaffirming their effectiveness in data processing and analysis tasks.