DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

CSCI 5408 Data Management, Warehousing, Analytics

Assignment 3

November 12, 2020

Submitted by:

Samkit Shah[B00852292]

Table of Contents

Table of Figures

GitLab Repository : https://git.cs.dal.ca/ssshah/assignment-3

Cluster Setup Steps:

1. Created an account on https://cloud.google.com/
2. Go to console of GCP by clicking on the console button.
3. Created a new project named "Assignment-3"
4. Created a new Compute Engine of Ubuntu E2- Medium Tier instance.
5. Installed Scala, Java and git to satisfy the requirements of spark.
6. Installed Apache Spark using command: wget https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tg
7. After that installed apache spark and configured path.
8. Started the master node for apache spark using command: start master.sh
9. Started the apache slave for the above apache master using command: start-slave.sh<url>
10. Open spark dashboard in browser to verify the working cluster.
11. Installed pyspark, tweepy and mongodb for future use.

Figure 1 shows the compute engine with apache spark cluster.

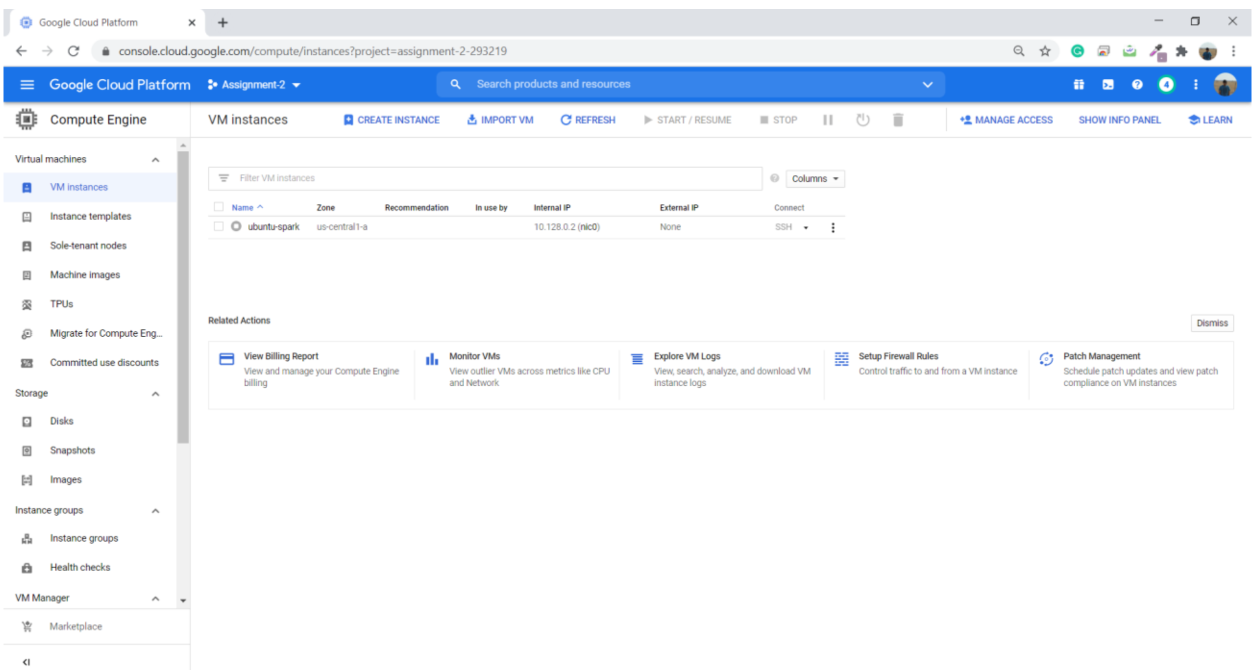

Figure 1 Compute Engine - Apache spark Cluster

Tweet Extraction Steps:

For data extraction from twitter, I have used tweepy library's search and stream API. Following is the approach used for data collection for search and stream of tweets.

1. Create a twitter development account.
2. Once Approved, generate consumer keys and access tokens to access twitter data using search and stream APIs.

After performing above two steps there are two part such as Search data extraction and Stream data Extraction. I have developed two python scripts to fetch the data from search API and stream API.

Along with that, tweets are stored in the MongoDb Database. I have created one database named RawDb using mongodb atlas. It has one collection named "tweets", where all tweets will be stored.

Search data extraction:

1. Authenticated identity of user using consumer API key and secret key.
2. Connect the MongoDb database using pymongo library.
3. The search API will fetch tweets based on the keywords mentioned in assignment.
4. A search API has returned the list of tweet object based on query. The tweet is of the four types such as Normal, Extended, Retweet, and Quoted Tweet.
5. Extracted approximately 2000 tweets.
6. Stored the JSON tweet object into the form of dictionary.
7. The tweets has different attributes according to type of tweet. The list of attributes for the tweet is as follow:
   Normal: Timestamp, tweet id, Content, username, user.screen_name, user.location.
   Extended: Normal tweet attributes & truncated, full_text.
   Retweet: Normal and extended tweet attribute, timestamp, id, text, user details.
   Quoted Tweet: Normal tweet attributes, id, text, user details of the original tweet.
8. Once the tweet object is composed then it will be stored in the RawDb.
9. Store tweet object into RawDb.

Stream Data Extraction:

1. Twitter_stream is a python script to fetch the live tweets data using stream API.
2. TwitterStreamListener is a class which is responsible to listen live tweets.
3. Connect the MongoDb database using pymongo library.
4. The user is authenticated using consumer key and access tokens.
5. On_data method is responsible to listen tweets and provide the tweets data in data variable.
6. In the response, it will check the type of the tweet attributes and compose the tweet according to type.
7. Compose_streaming_tweet_row function is responsible to create a dictionary of attributes of tweet.
8. The tweet object will be stored in the MongoDb database named RawDb.
9. This process will continue till 1000 tweets.

Output Screenshot for twitter data:

1. Initial Status of RawDB database:

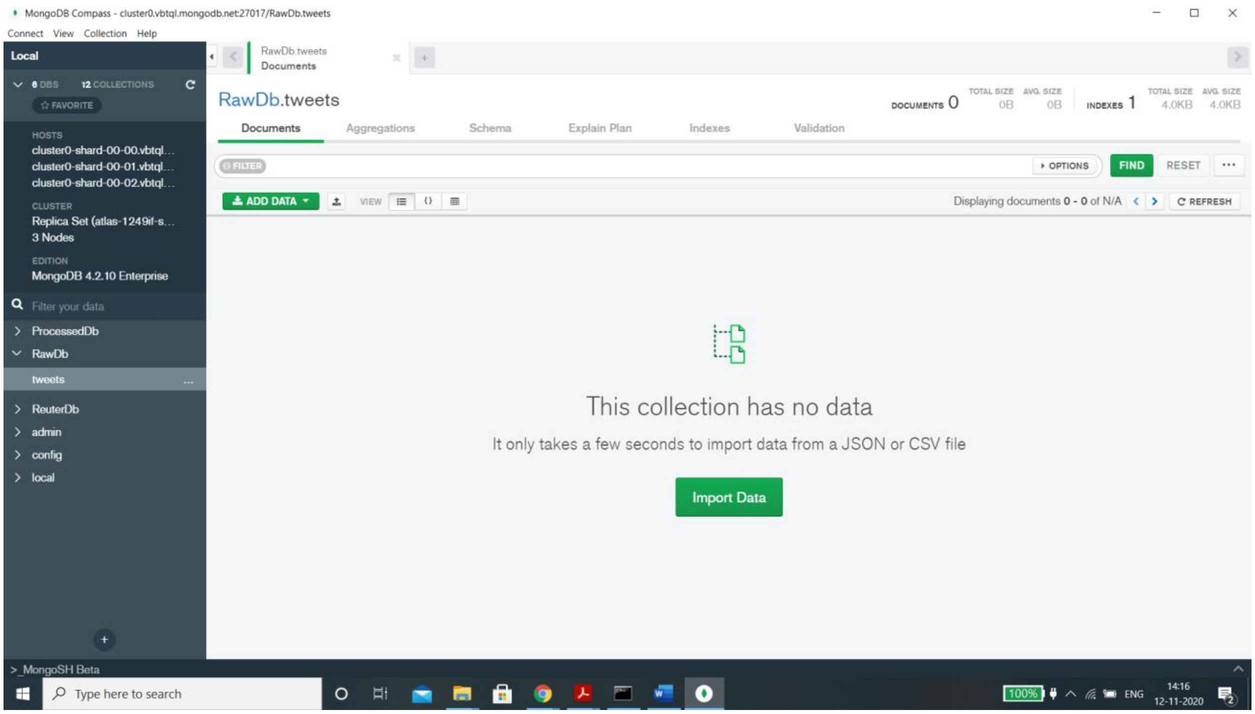

Figure 2 Initial Status of RawDb Database

2. Run python Script for search data extraction:

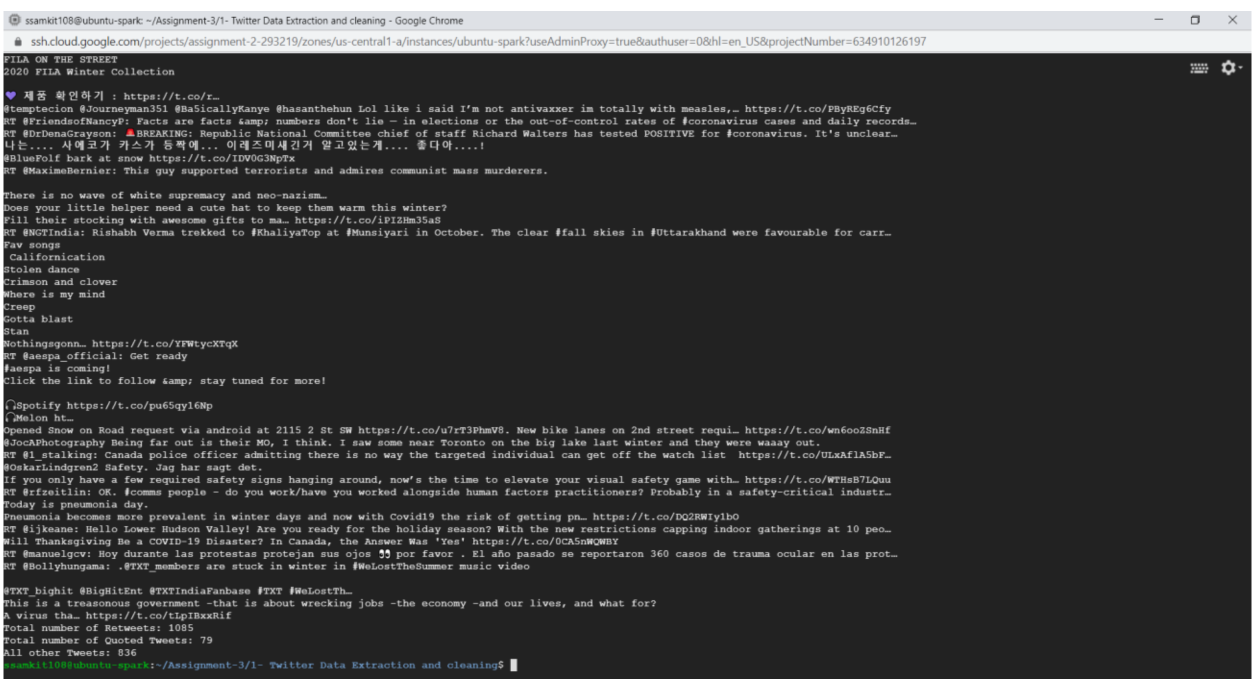

Figure 3 Search tweet execution on GCP

3. After Performing Search Data Extraction:

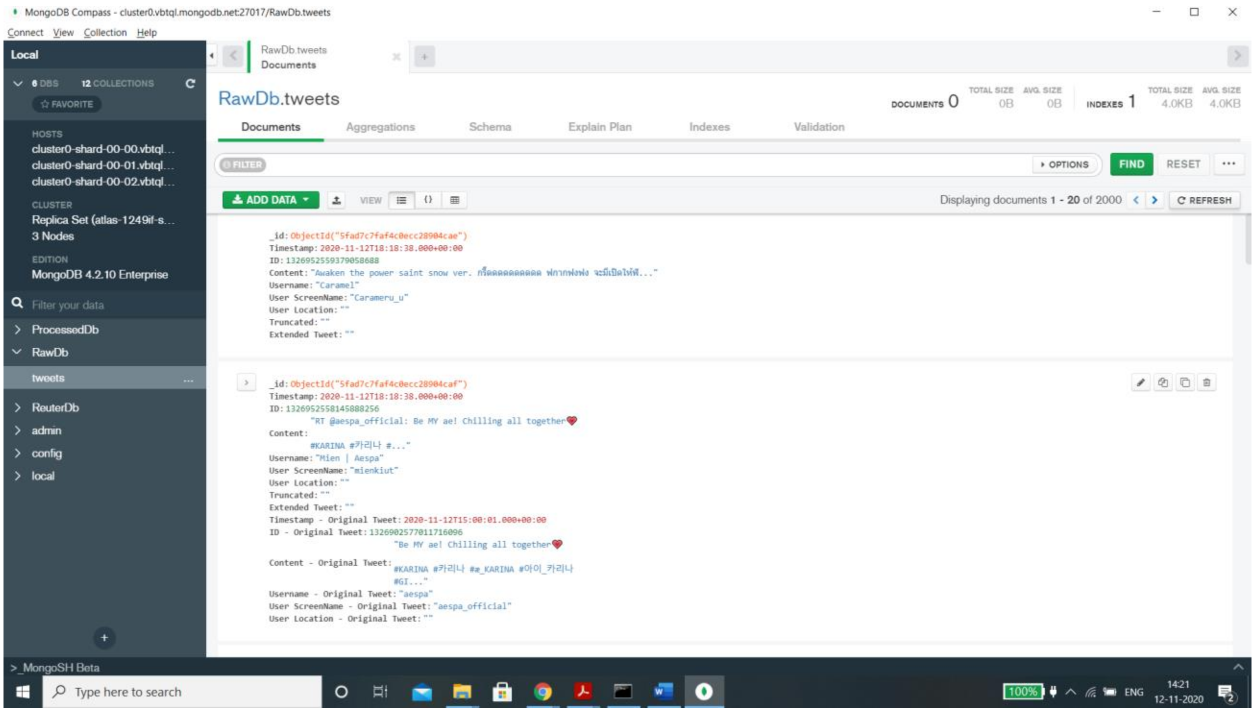

Figure 4 RawDb database

4. Run the python script to fetch the streaming data

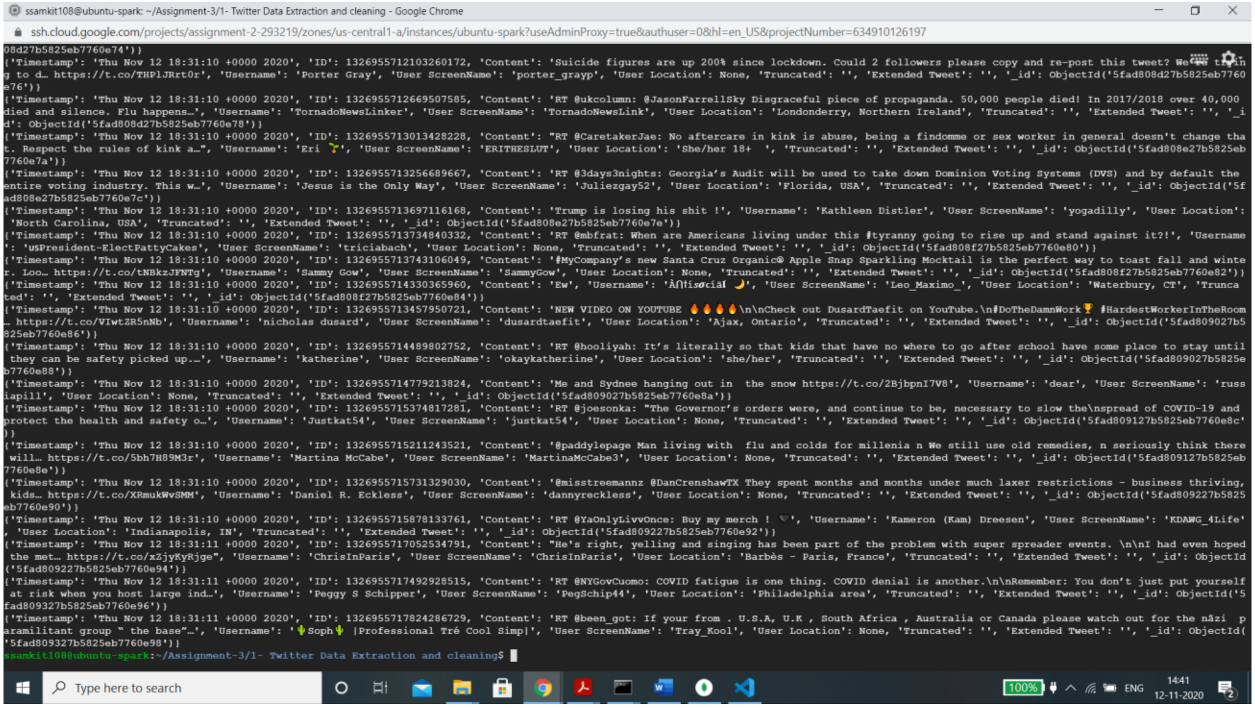

Figure 5 Stream tweet data execution

5. RawDb after adding stream data:

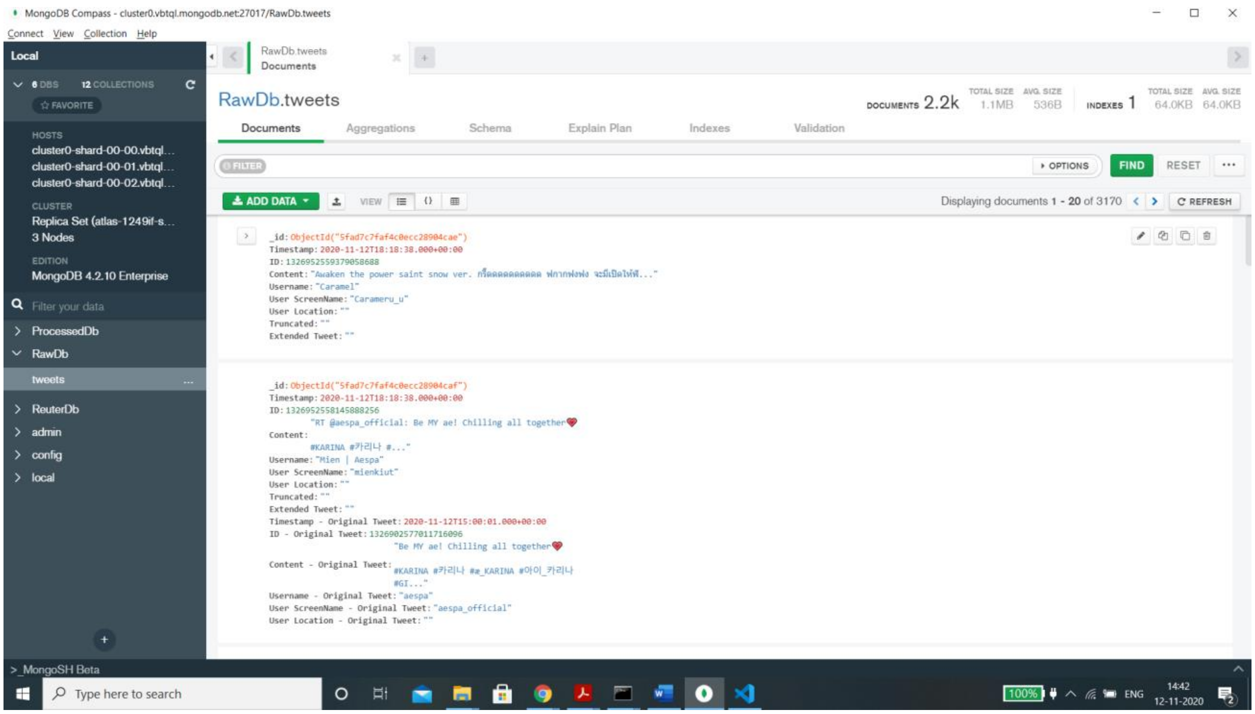

Figure 6 RawDb after added stream data

Cleaning Process:

All the tweets are stored in the RawDb. Now the task is to clean the tweets present in the RawDb and store it into the ProcessedDb. For cleaning purpose, I have written one python script named process.py. This script will fetch the data from the RawDb then clean() function will clean data and then it will store cleaned tweets to the ProcessedDb.

Data is cleaned by using Regular Expression. There is no use of any library. Mainly following parameters are cleaned as a part of cleaning process.

1. Multiple white spaces are replaced with single spaces.
2. Removed URLs in the tweets.
3. Removed Emoticons.
4. Removed new line character.
5. & symbol is corrected in tweets.
6. All the special characters (! , @, ‘, “ , &) are removed.

Output Screenshot for Cleaning:

1. **Initial status of ProcessedDb database**

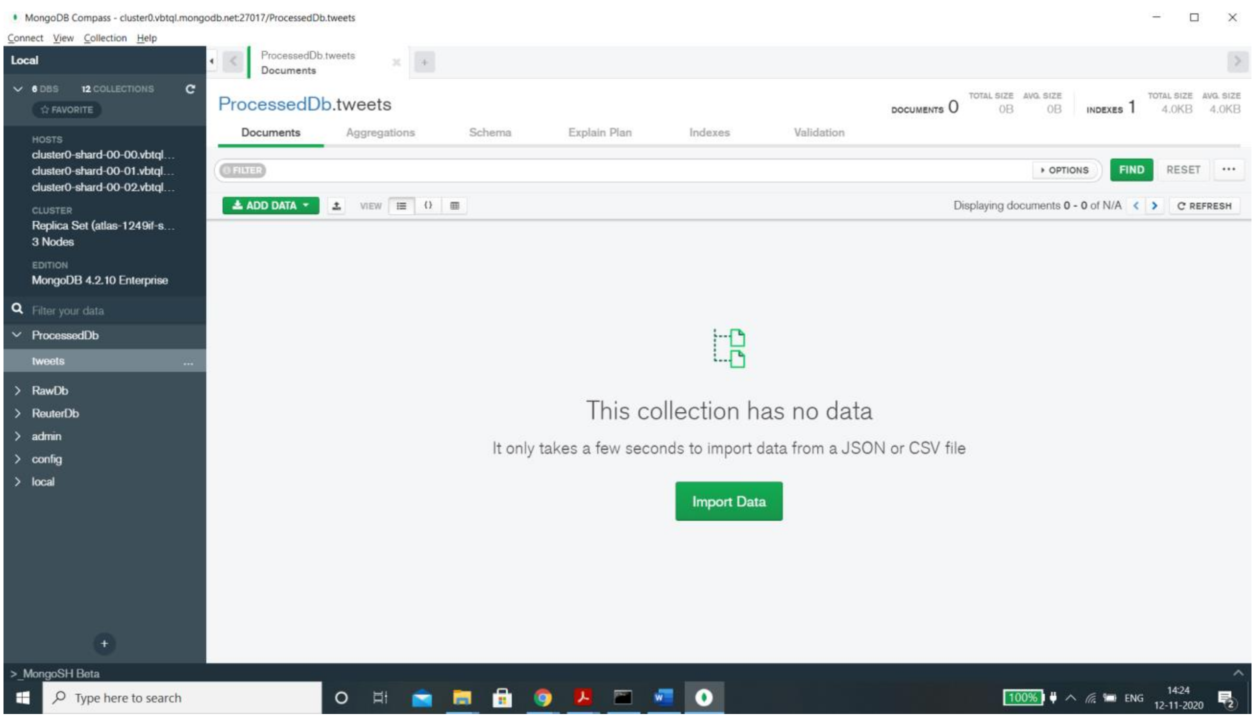

Figure 7 Initial ProcessedDb

2. Run the python script to clean data:

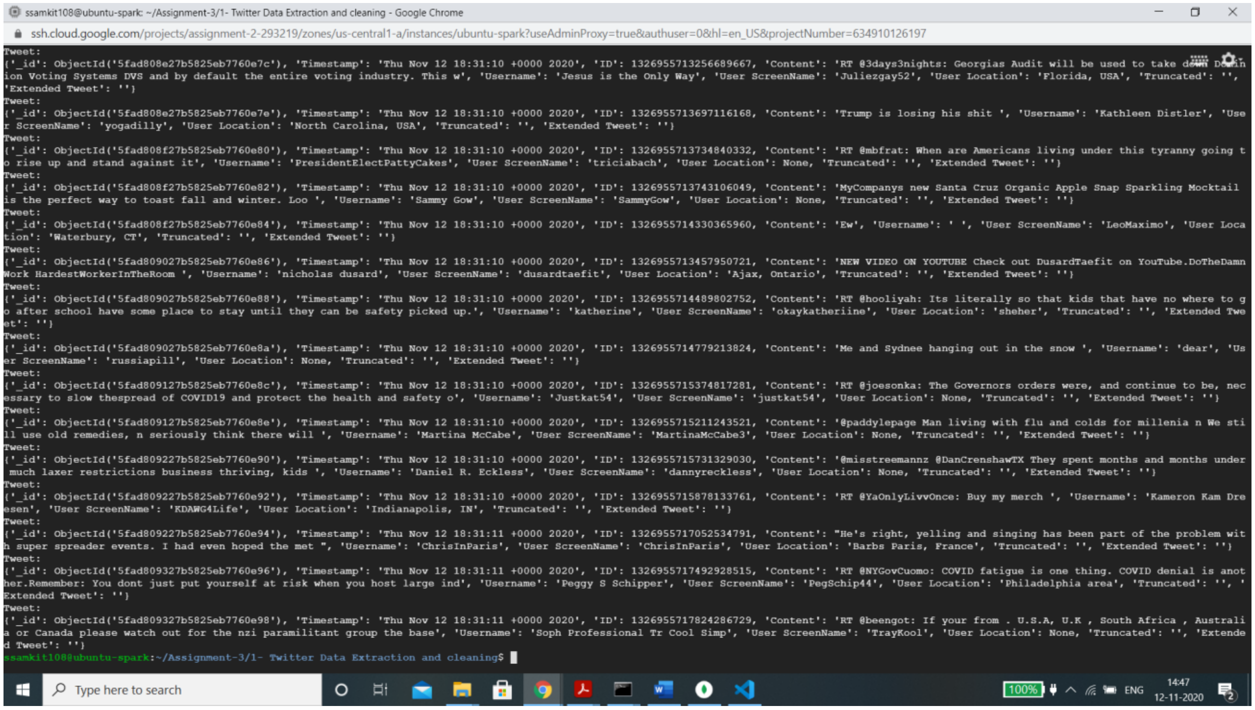

Figure 8 Execution of cleaning process

3. Cleaned data in ProcessedDb:

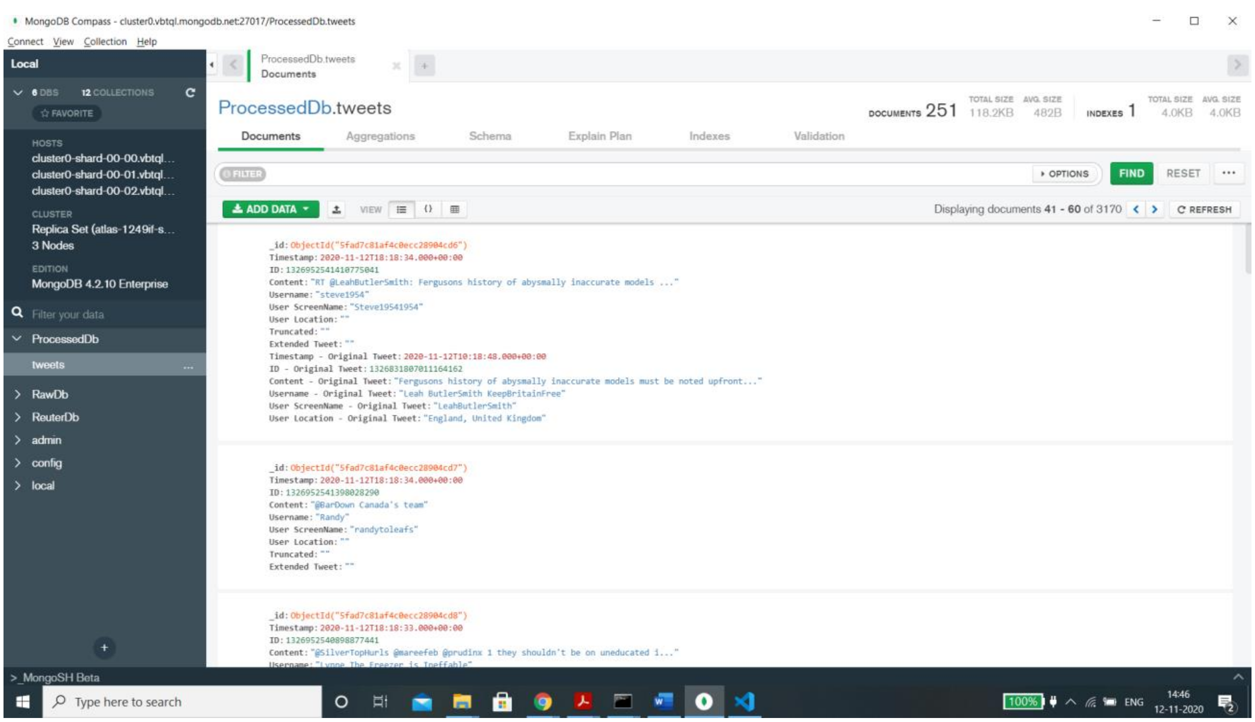

Figure 9 ProcessedDb after cleaning

News Articles data extraction:

Data extraction for news articles from the SGM file is done using python script. I have extracted data from the text tag. Basically it has mainly three tags such as title,dateline, and body. After reading both file I have cleaned it and stored in ReuterDb. The detailed steps are explained further.

1. Open and read both files.
2. Connect to MongoDb database named ReuterDb.
3. Create an instance of Articles collection for MongoDb.
4. Extract the content of the tag from the file using regular expression.
5. Clean the content of body, title and dateline.
6. Create an object of one article and store it in database.
7. Repeat this process until all files are not read.

One Document contains one news article. To clean the data of text tag, I have used regular expression.

Output for news articles data extraction:

1. **Initial status of ReuterDb**

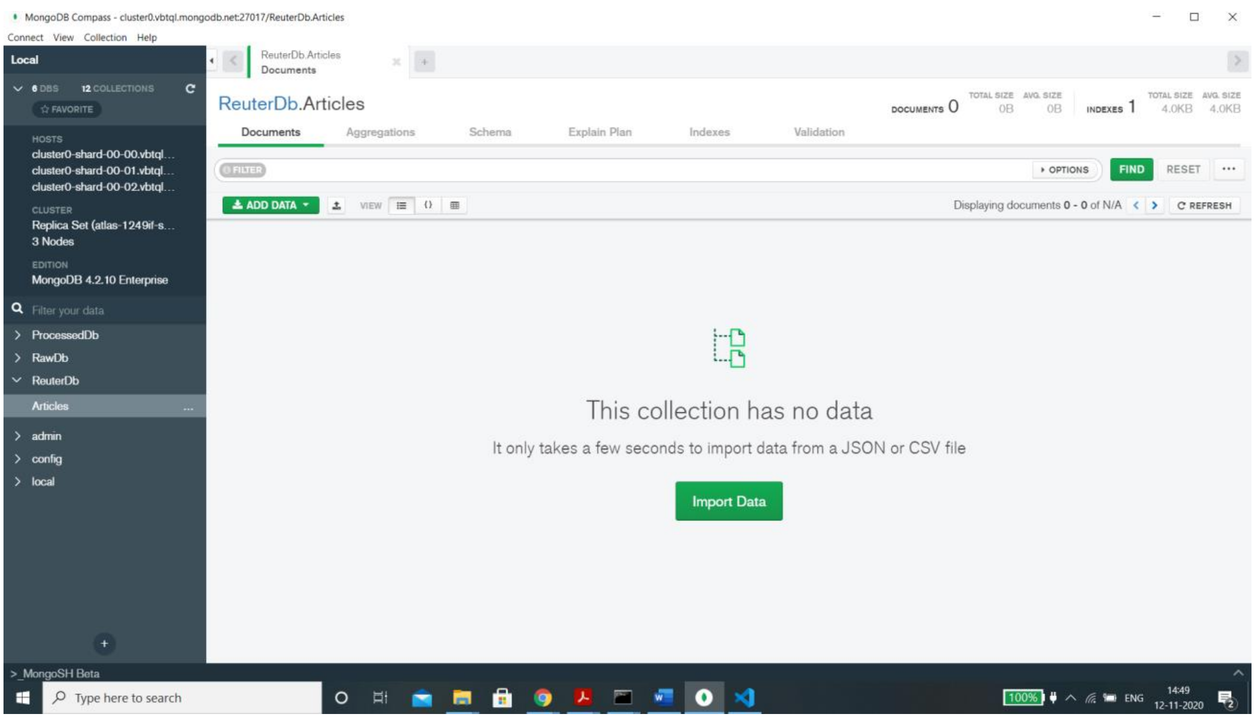

Figure 10 Initial state of ReuterDb

2. Executing news article data extraction:

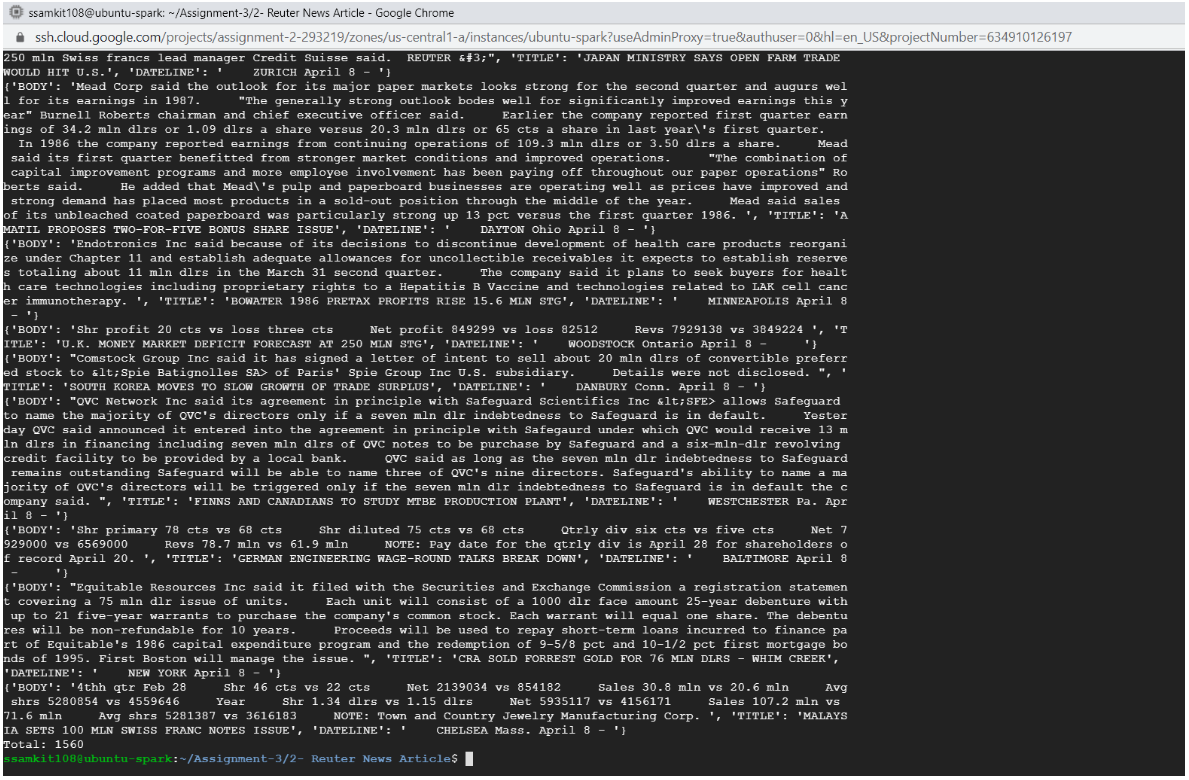

Figure 11 Reuter data extraction

3. After running script of data extraction:

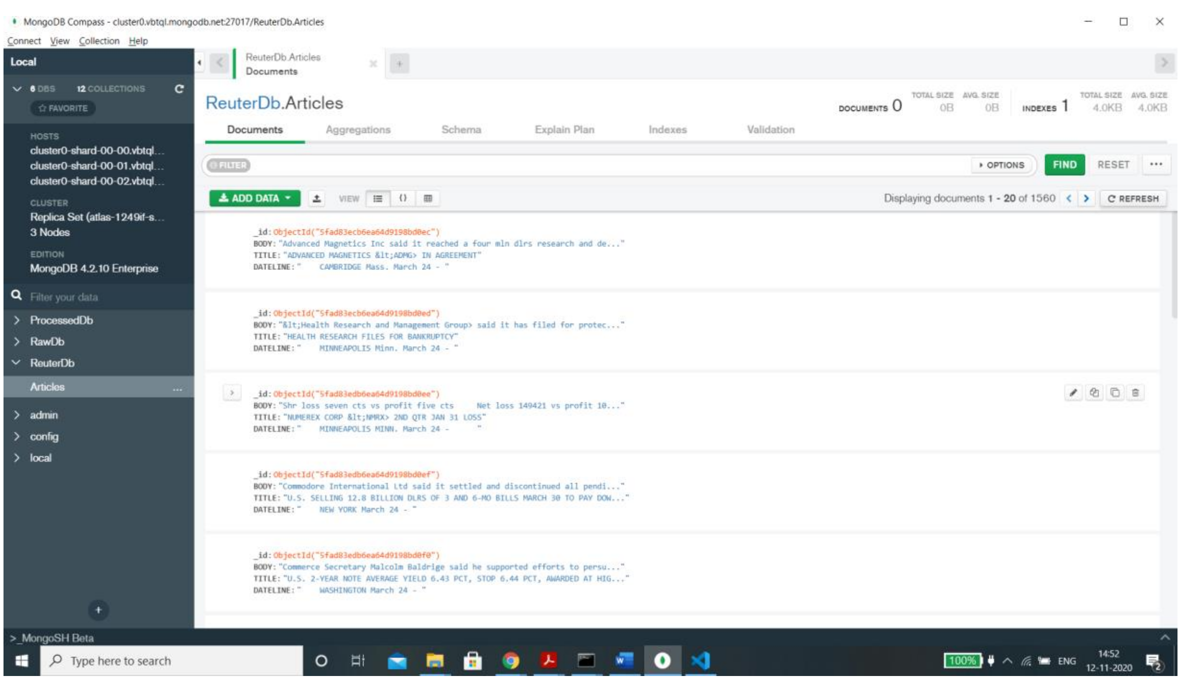

Figure 12 ProcessedDb after data added

Map Reduce to perform count:

Refer the MapReduce python script which is responsible to perform the task using Apache Spark. I have used pyspark library to achieve the distributed computation. Implemented MapReduce approach to count the frequency of keywords in tweets and news articles. The detailed steps are explained further.

1. Create SparkContext to allocate the spark cluster.
2. Sparkcontext is responsible to manage jobs and distributed processing.
3. Fetched tweets from the ProcessedDb database.
4. Splitted content of tweets using single space and extracted all words of tweets.
5. Created a list containing all words of tweets.
6. Filtered the words on the basis of given keywords and remove unnecessary words from the list.
7. Used lambda function to filter the words and stored relevant words in the list.
8. Fetched news articles from the ReuterDb database.
9. Splitted body of news article and extracted all words of news article.
10. Filtered the words on the basis of given keywords and removed unnecessary words from the list.
11. Used lambda function to filter the words and stored in separate list.
12. Merged two list of tweets and news article.
13. Applied MapReduce Approach to count the frequency of words. There are mainly to component one is Map and second is Reduce.
14. Map function takes a set of data in terms of list and convert it into the set of data in terms of key-value pair. Initially the value is always 1 because we are considering single word as a key.
15. Converted the conventional list into the distributed dataset (RDD) using paralleize() method of sparkcontext. This method will convert a collection to RDD for distributed processing.
16. Reduce function takes output of map function and combines those data into a set and incrementing value based on occurrence.
17. Reduce function will give word with the occurrences.
18. Displayed the word count as an output.

The Mapreduce program is executed on GCP in Apache spark cluster. To execute that I have used Spark-submit command which execute python script in apache spark. It is shown in figure 13 and 14.

Output of MapReduce Program:

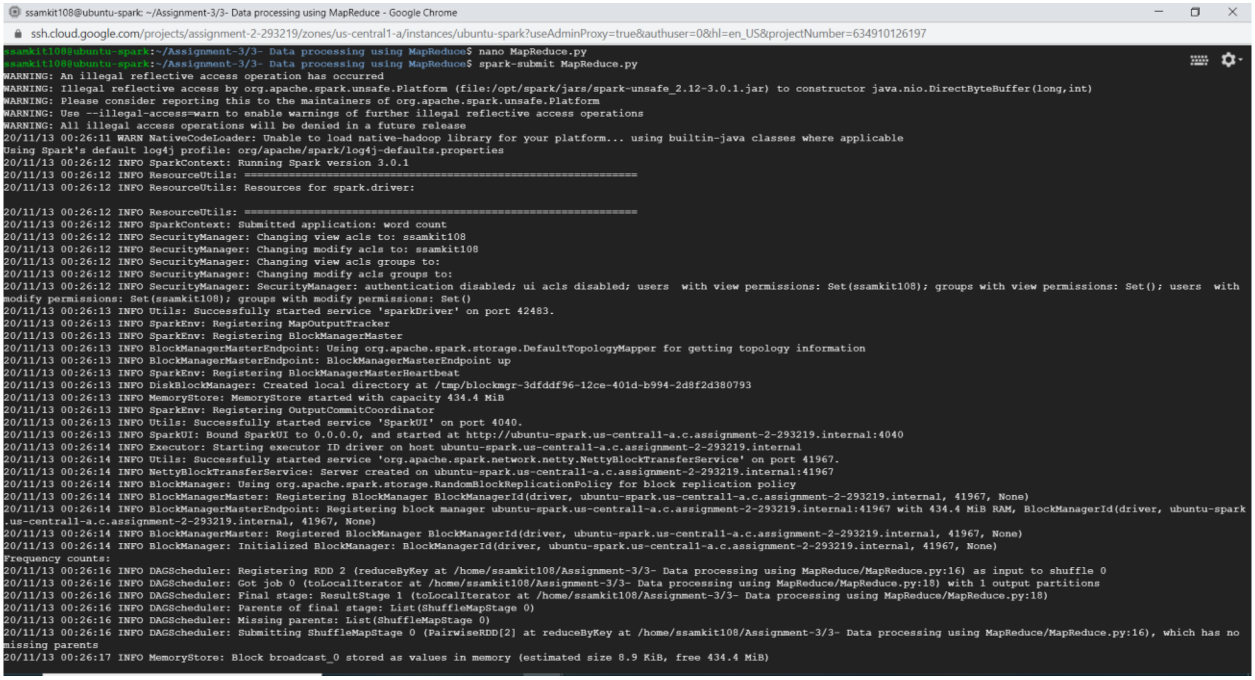

Figure 13 MapReduce Output-1

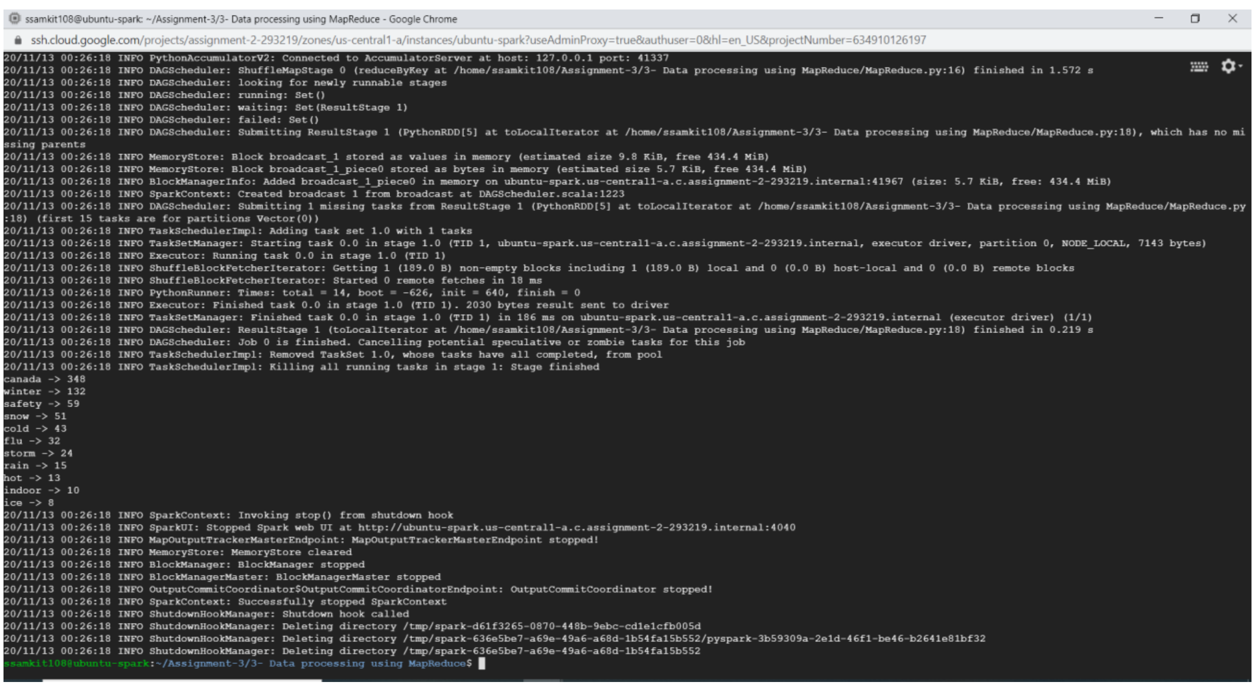

Figure 14 MapReduce Output-2

Highest Frequency: Canada → 348

Lowest Frequency: Ice → 8

Data Visualization using Graph Database:

Created Node of various keywords. The cypher query for creating nodes are given below:

```
CREATE (storm:Storm {
  title: "Storm"
})
CREATE (winter:Winter {
  title: "Winter"
})
CREATE (canada:Canada {
  title: "Canada"
})
CREATE (hot:Hot {
  title: "Hot"
})
CREATE (cold:Cold {
  title: "Cold"
})
CREATE (flu:Flu {
  title: "Flu"
})
CREATE (snow:Snow {
  title: "Snow"
})
CREATE (indoor:Indoor {
  title: "Indoor"
})
CREATE (safety:Safety {
  title: "Safety"
})
CREATE (rain:Rain {
  title: "Rain"
})
CREATE (ice:Ice {
  title: "Ice"
})
```

After creating node, I have added properties to each node. These properties are identified from the relevant tweets and some properties are added by critical thinking. All the cypher queries are mentioned below:

```
MATCH (n:Canada {
  title:"Canada"
}) set n.coronaCases=277061
RETURN n

MATCH (n:Canada {
  title:"Canada"
}) set n.Capital='Ottawa'
RETURN n

MATCH (n:Canada {
  title:"Canada"
```

```
}) set n.Currency= "Canadian Dollor", n.Area='9.98 milllion sq km',
n.languages='English, French'
RETURN n

MATCH (n:Cold {
  title:"Cold"
}) set n.Temperature="<10 Celcius"
RETURN n

MATCH (n:Strom {
  title:"Strom"
}) set n.location="Halifax", n.Country="Canada", n.Date="07/11/2019",
n.Time="18:13", n.Type="Strom Surge/Tide", n.Property_Damage="68.100 million
CAD"
RETURN n

MATCH (n:Snow {
  title:"Snow"
}) set n.Density="0.1-0.8 g/cm^3", n.Strength="2.5 kPa", n.melting_temp="0
degree Celcius"
RETURN n
MATCH (n:hot {
  title:"hot"
}) set n.Temperature=">10 Celcius"
RETURN n

MATCH (n:Flu {
  title:"Flu"
}) set n.Symptoms=["Fever", "runny nose", "sore throat", "headache"],
n.duration="1 week"
RETURN n
```

There are various relations between two nodes. Relationship are added through cypher query language. Some nodes do not have any logical relations but some have. All the relations are shown in the figure 14 and cypher query is mentioned below:

```
MATCH (a:Winter), (b:Snow) CREATE (b)-[r:FALL_IN]->(a)
MATCH (a:Strom), (b:rain) Create (a)-[r:CAUSES]->(b)
MATCH (a:rain), (b:flu) Create (a)-[r:GET_SICKED]->(b)
MATCH (a:Canada), (b:Strom) Create (a)-[r:ISSUED_ALERTS]->(b)
MATCH (a:Strom), (b:Safety) Create (a)-[r:INCREASES_RISK]->(b)
MATCH (a:Strom), (b:Indoor) Create (a)-[r:STAY]->(b)
MATCH (a:Cold), (b:Flu) Create (a)-[r:IS_SYMPTON]->(b)
MATCH (a:Winter), (b:Flu) Create (b)-[r:STRIKES_IN]->(a)
MATCH (a:Indoor), (b:hot) Create (a)-[r:IS]->(b)
MATCH (a:rain), (b:Flu) Create (a)-[r:IS_A_SEASON]->(b)
MATCH (a:Canada), (b:Cold) Create (a)-[r:WEATHER]->(b)
MATCH (a:Snow), (b:ice) Create (a)-[r:CONVERTS_INTO]->(b)
```

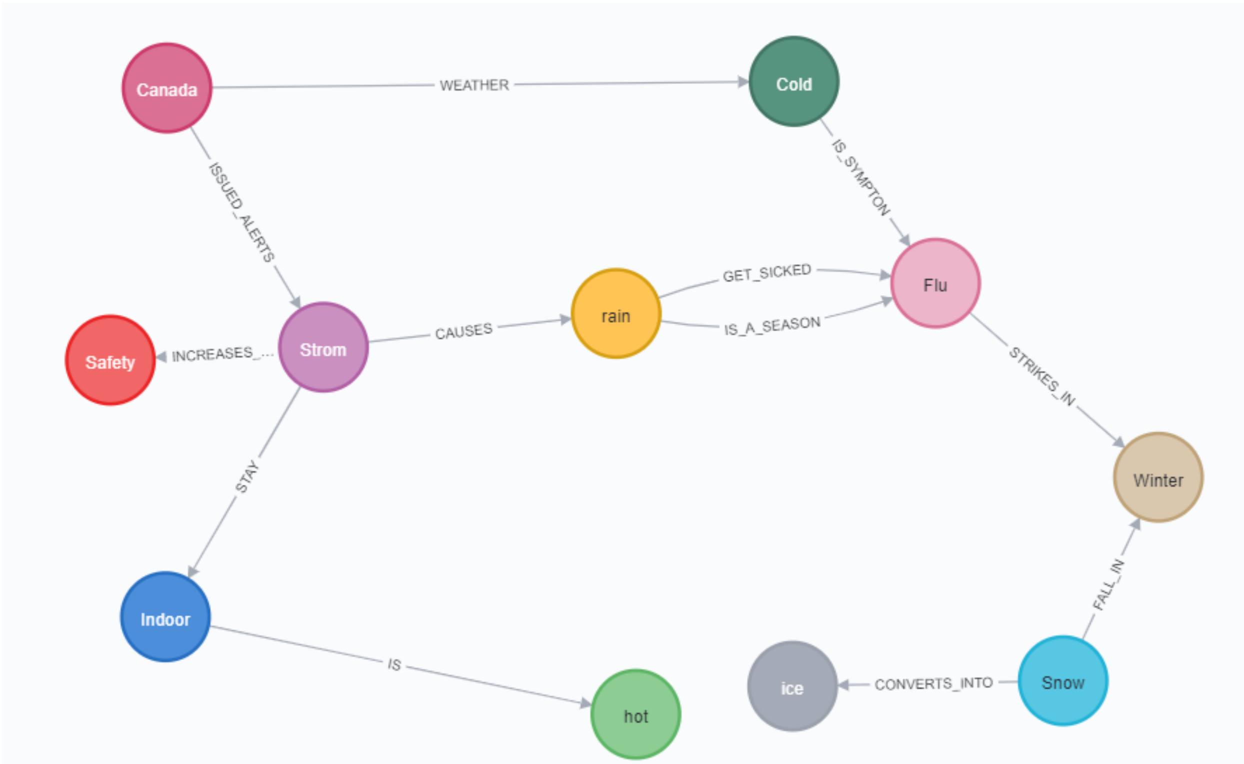

Figure 15 Graph of Nodes

References:

1. "Managed MongoDB Hosting | Database-as-a-Service", MongoDB, 2020. [Online]. Available: https://www.mongodb.com/cloud/atlas. [Accessed: 10- Nov- 2020]
2. "Cloud Computing Services | Google Cloud", Google Cloud, 2020. [Online]. Available: https://cloud.google.com/. [Accessed: 6- Nov- 2020]
3. "Querying with Cypher - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2020. [Online]. Available: https://neo4j.com/developer/cypher/querying/. [Accessed: 6- Nov- 2020]
4. "Tweepy", Tweepy.org, 2020. [Online]. Available: https://www.tweepy.org/. [Accessed: 6- Nov- 2020]
5. "Neo4j Graph Platform – The Leader in Graph Databases", Neo4j Graph Database Platform, 2020. [Online]. Available: https://neo4j.com/. [Accessed: 6- Nov- 2020]
6. "Welcome to Spark Python API Docs" [Online].Available : https://spark.apache.org/docs/latest/api/python/index.html [Accessed 08-Nov-2020]