What database do you choose for heterogeneous security log events analysis?

Sofiane Lagraa, Radu State
Interdisciplinary Centre for Security, Reliability and Trust (SnT),
University of Luxembourg
firstname.lastname@uni.lu

***Abstract*—The heterogeneous massive logs incoming from multiple sources pose major challenges to professionals responsible for IT security and system administrator. One of the challenges is to develop a scalable heterogeneous logs database for storage and further analysis. In fact, it is difficult to decide which database is suitable for the needs, the best of a use case, execution time and storage performances.**

In this paper, we explore, study, and compare the performance of SQL and NoSQL databases on large heterogeneous event logs. We implement the relational database using MySQL, the column-oriented database using Impala on the top of Hadoop, and the graph database using Neo4j. We experiment the databases on a large heterogeneous logs and provide advice, the pros and cons of each SQL and NoSQL database. Our findings that Impala outperforms MySQL and Neo4j databases in terms of loading logs, execution time of simple queries, and storage of logs. However, Neo4j outperforms Impala and MySQL in the execution time of complex queries.

I. Introduction

In cyber-security, the concepts for network security such as prevention, detection, and investigation are based on the collection of heterogeneous logs incoming from multiple and different system or device sources [7], [8]. When a large and heterogeneous data poses cyber-security challenges, the choice of a database is a major architectural component for storing and processing large heterogeneous event logs for network security.

NoSQL databases are designed for a huge amount of data, tabular and non-tabular, store data differently than the structured and relational tables (SQL databases). There are different types of NoSQL databases based on their data model. The main types are: key-value, document, column, and graph. They provide a flexible schema and easily scale with large amounts of data and high user loads. They store the data relationship differently than relational databases do.

For developing an SQL or NoSQL database, storing and analysis of heterogeneous security, the admin has to explore the pros and cons of each database, in particular the following properties:

- Data Model: databases often leverage data models more adapted to use cases.
- Performance: the execution time of a query is the most important metric for measuring the performance of a database.
- Scalability: SQL databases are often complicated, expensive to manage, and hard to scale up. In contrast, NoSQL databases are designed to scale-out horizontally.
- Data Distribution: NoSQL databases are designed for distributed systems while the SQL databases are designed for data centralization.
- Flexibility: NoSQL databases are better to test new ideas and update data structures like `JSON, XML, CSV` format documents.
- Size of the database: the size of a database on the disk may change between database systems. The change of disk usage is due to the model of the data and the type of database.

978-3-903176-32-4 © 2021 IFIP

However, it is difficult to decide which database is the best for a use case in terms of time and storage performance, utilisation, and scalability of large and heterogeneous logs. In this paper, we explore the pros and cons of some of SQL and NoSQL databases. We study their performance on large heterogeneous log files. We implement a database from each type of: relational database such as MySQL, column-oriented database such as Impala, and graph database such as Neo4J. The key-value and document-oriented databases are not suitable for storing and querying heterogeneous and unstructured log data as we conclude in our study. We experiment the implementations on a public dataset published by Los Alamos National Laboratory. We provide insights to security admins about the choice of databases for large heterogeneous logs: whether an admin wants to store logs for simple processing with a simple query it is recommended using a query engine like Impala on the top of Hadoop. Whether an admin wants to process in-depth by developing a complex query, we recommend a graph database such as Neo4J.

The rest of the paper is organized as follows. Section II introduces and reviews the existing background of databases and their use in network security. Section III describes the heterogeneous log we use in this paper for our study. We propose our database models we developed in Section IV. We then present the experimental results in Section V. Section VI discusses the results. Section VII concludes and gives some insights.

II. Background and Related work

A. NoSQL Databases

There are four types of NoSQL databases:

1) Key-value-oriented databases: Key-value databases are one of the simplest types of databases where each item contains keys and values. They are used for performing simple queries and not designed to perform complex queries. Redis [1] and BerkeleyDB [2] are an example of key-value databases.

2) Document-oriented databases: Document databases store data in documents similar to JSON (JavaScript Object Notation) or XML (Extensible Markup Language) objects. Each document contains pairs of fields and values. Thus, document databases use a key-value store. MongoDB [3] is an example of a document database.

3) Column-oriented databases: Column-oriented databases (or column-stores) store each database table column separately, with attribute values belonging to the same columns as opposed to traditional relational databases that store rows one after the other. Column databases often used in data warehouses. HBase [4] and Cassandra [5] are examples of Column-oriented databases.

4) Graph-oriented databases: Graph database models are characterized as those where data structures for the schema and instances are modeled as graphs, and data manipulation is expressed by graph-oriented operations [1]. The benefit of using a graph data model is given by: the natural modeling of graph data representation. For example, a graph representation relations in network security alerts [2]. Neo4J [6] is an example of a graph database.

B. Related work

In [7] the authors proposed HuMa framework, a multi-analysis approach to study complex security events in a large set of heterogeneous data incoming from firewalls, servers, and routers. In [6], the authors compared MongoDB with a commercial vendor and with a popular open source relational DBMS. They concluded that MongoDB is a viable alternative compared to relational databases. In [2], the authors proposed a Neo4j graph database-based approach for alert aggregation and visualization. The graph database model represents relations between sensors or alert types and common type of reported alerts or duplicated alerts. The graph is used by security analysts and network administrators. In [10], the authors proposed a Neo4j graph database-based hierarchical multi-domain network security situation awareness (NSSA) data storage method. They combine all the data that reflect the network security situation and launching queries for the visualization of query results using the Cypher query language. In [8], the authors proposed an approach for investigation of attack graph analysis based on Neo4j graph database. They combined a variety of data by creating relationships between firewalls, host vulnerabilities, potential attack patterns, and intrusion alerts. We notice that the existing works use databases in their solutions without studying and comparing the performance of their database regarding the processing, storage, and utilization. In addition, even the non-exhaustive works on the use of SQL or NoSQL databases for heterogeneous data, to the best of our knowledge, no study has been proposed to measure the performance of databases on large heterogeneous logs of security.

[1]https://redis.io/

[2]https://www.oracle.com/database/berkeley-db/db.html

[3]https://www.mongodb.com/

[4]http://hbase.apache.org/

[5]https://cassandra.apache.org/

[6]https://neo4j.com/

III. Large heterogeneous dataset overview

Data used in this paper are from Los Alamos National Laboratory (LANL). LANL provides a public comprehensive dataset[7] [4], [3]. It includes 58 consecutive days of:

- **windows-based authentication events** from both individual computers and centralized Active Directory domain controller servers: Each event is in the form of ”time, source user@domain, destination user@domain, source computer, destination computer, authentication type, logon type, authentication orientation, success/failure” and represents an authentication event at the given time.
- **process start and stop events** from individual Windows computers: Each event is in the form of ”time, user@domain, computer, process name, start/end” and represents a process event at the given time.
- **Domain Name Service (DNS)** lookups as collected on internal DNS servers. Each event is in the form of ”time, source computer, computer resolved” and presents a DNS lookup at the given time by the source computer for the resolved computer.
- **network flow** data as collected on at several key router locations. Each event is in the form of ”time, duration, source computer, source port, destination computer, destination port, protocol, packet count, byte count” and presents a network flow event at the given time and the given duration in seconds.

The events are labelled as malicious or normal via a RedTeam file. In total, the event logs set is approximately 92.63 gigabytes across the four data elements and presents 1,648,274,558 events in total for 12,425 users, 17,684 computers, and 62,974 processes. Table I summarizes the characteristics of heterogeneous event logs incoming from different services.

TABLE I: Characteristics of heterogeneous log files.

File	Size	#Events	File	Size	#Events
auth.txt	70 GB	1,051,430,459	flows.txt	4.9 GB	129,977,412
proc.txt	15 G	426,045,096	dns.txt	776 MB	40,821,591
Total	92.63 GB	1,648,274,558			

[7]https://csr.lanl.gov/data/cyber1/

IV. Implementation of database models on heterogeneous logs

We use one database from each type of database, except key-values and document-oriented databases. Key-values databases don't manage multiple and heterogeneous data and JOIN query cannot be possible. However, document-oriented databases such as MongoDB manage multiple and heterogeneous data and trying to JOIN in MongoDB would defeat the purpose of using it. For instance, MongoDB creates a document for each row or event log. For example: the event log from netflow files (flows) `{1, 0, C1423, N1136, C1707, N1, 6, 5, 847}` corresponding to `time, duration, source_computer, source_port, destination_computer, destination_port, protocol, packet_count, byte_count`, respectively, is represented by the following document in MongoDB.

```
{"_id" : ObjectId("5f5ccf24976d541a396e9f4c"), "time" : 1,
    "duration" : 0, "source_computer" : "C1423", "
    source_port" : "N1136", "destination_computer" : "C1707
    ", "destination_port" : "N1", "protocol" : 6, "
    packet_count" : 5, "byte_count" : 847}
```

The ObjectId value is the identifier of the document created by MongoDB. It consists of: timestamp value, random value, and incrementing counter. It also is the default primary key for a MongoDB document and is usually found in the `_id`. Thus, 129,977,412 documents are created for the netflows in flows file, which is the total number of event logs. For each heterogeneous log file, a collection of documents is created, and a simple query can be launched such as retrieving a value or counting. However, complex queries with JOIN operations cannot be possible across different types of collections of documents. Thus, we performed a comparative study between the column-oriented database, relational database, and graph database.

A. Column-oriented database

Hadoop is a framework that allows for the storage and distributed processing of large data sets across computers in clusters using simple programming models. Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications. For processing data in HDFS there are several libraries and tools such as Spark, Hive, Impala. In this study, we use *Cloudera Impala*. Impala is an open source, native analytic database for Apache Hadoop, and SQL engine for Hadoop that was designed to bring a parallel database management system (DBMS) to the Hadoop environment. It was written from the ground up in C++ and Java. It works both for analytical and transactional and single-row workloads. It runs within Hadoop, by widely using Hadoop file formats for reading the data, and runs on the same nodes that run Hadoop processes. It utilizes Hadoop components such as HDFS, HBase, Metastore, YARN, and is able to read the majority of the widely used file formats such as `csv, plain text, JSON, Parquet, Avro`. A major Impala goal is to make SQL-on-Hadoop operations fast and efficient. It provides low latency and high concurrency for analytic read-mostly data on Hadoop Distributed File System. It demonstrates its superior performance compared against other popular SQL-on-Hadoop systems such as SparkSQL, Hive, and Presto [5].

In Impala, a database is a logical container for a group of tables. Each database defines a separate namespace. Creating a database is a lightweight operation. There are minimal database-specific properties to configure and create tables. The only condition is to put the log files in the HDFS or HBase. In fact, each database is physically represented by a directory in HDFS. In order to use Impala, we have to move the logs from disk to HDFS using `$ hdfs dfs -put` command. We use Impala-shell to create the tables and load the corresponding logs in the namespace. In Listing 1 of Appendix A, we present a code to create a log database model in Impala. Impala keeps its table creations in a traditional MySQL database known as the metastore.

B. Relational database

We chose to evaluate MySQL because it is a popular and widely used relational database for different applications and systems. MySQL uses SQL as language for querying the database. The database model we create for the study is the same as the column-oriented database. It follows a relational data model to store the logs. However, the structure of the heterogeneous logs is dynamic but in this study we keep it static, i.e., the number of fields don't change over time. Thus, the structure of the database would have been static and we have had to follow a structure set implicitly in the database.

C. Graph database model

We regroup all event logs in a graph database. The use of a graph database allows to interconnect heterogeneous event logs in a database storage and perform graph operations processing over them. The advantages of Neo4j database compared to MySQL and Impala are: finding paths between source and destination nodes, finding intrinsic and complex patterns that couldn't be discovered using an SQL query.

In our study, we use Neo4j as a graph database. Neo4j is the most popular graph database and it is freely available and well documented. Neo4j is a native disk-based storage manager, high performance, scalable, and robust graph database solving queries with multiple relationships storing data in the nodes and relationships [9]. Cypher Query Language is used for sub-graph querying. It is a declarative language used for loading, storing and retrieving data from the graph database.

1) Event log import method: To import the heterogeneous event logs to the Neo4j database, there are globally two options are available:

- Load CSV [8]: used using the Cypher command to load into an existing database.
- Neo4j Import Tool [9]: is only for newly created databases.

However, we need to organize the log files in order to be imported into Neo4j database.

[8]https://neo4j.com/developer/guide-import-csv/
[9]https://neo4j.com/docs/operations-manual/current/tools/import/

The *csv* files holding the event logs need to be processed and structured in the way of the corresponding header file, e.g., the *SourceComputer.csv* file needs to be structured as the *SourceComputer-header.csv*.

2) Network security to graph database model: A graph database is a storage system that uses graph structures, designed for managing graph-like data following the basic principles of database systems. The fundamental abstraction behind a database system is its database model. Graph database model schema is composed of following nodes: Source user, source computer, destination computer, resolved computer. The nodes are connected through the following relationships: USES, IS_RUNNING, IS_RESOLVED, CONNECT, FLOWS. Figure 1 provides and summarizes the proposed graph database model.

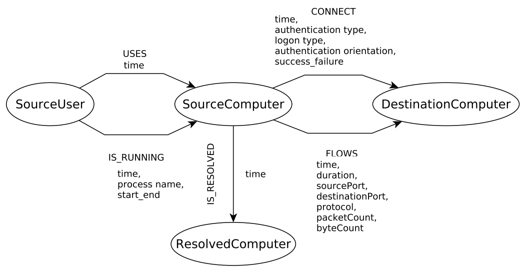


Fig. 1: Graph database model.

The nodes and edges i.e. relationships can be imported into Neo4j using the import script shown in Listing 2 of Appendix B. Different indexes are created using the Cypher language for indexing nodes for quick access, processing and querying.

3) Characteristics of graph database model: Our current implementation consists in the following components: 1) a processing unit for extracting and transforming heterogeneous event logs, 2) a logs set transformation pipeline built using bash scripts, 3) a graph model construction component.

Table II shows the characteristics of the graph database. It took 2 hours, 34 minutes, and 2 seconds to import the whole heterogeneous event logs. The total size of the graph database in a disk space is 250 GB, which is 2.7 times greater than the original event logs.

TABLE II: Graph database characteristics

Node Type	Count	Relationship Type	Count
SourceUser	80,553	USES	1,051,430,459
SourceComputer	16,249	IS_RESOLVED	40,821,591
DestinationComputer	15,895	IS_RUNNING	402,894,548
ComputerResolved	13,776	CONNECT	1,051,430,459
		FLOWS	119,905,148
All Nodes	126,473	All Relationships	2,546,577,057

Figure 2 shows an example of the users who use a common computer. The sub-graph is extracted using Cypher query language.

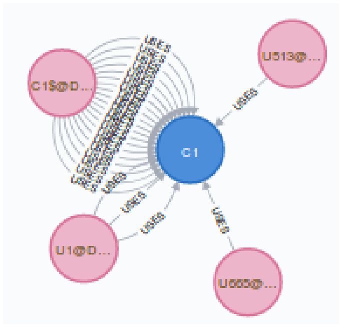


Fig. 2: Users who use a common computer.

V. Experiments

In this section, we evaluate our experiments on various heterogeneous dataset described in Section III. We also compare the performance of the three databases we propose. The aim of this comparison is to show that beside the storage efficiency and the execution time of a simple and complex query. We first describe the experimental environment, then the queries used for comparing the database, and finally we present our results and discuss them.

A. Experimental setup

The experiments are performed on a 60 cores Intel(R) Xeon(R) CPU E5-4650 v4 @ 2.20GHz with 376 GB of RAM. The machine runs on Linux. MySQL, Impala and Neo4j databases are implemented using their import scripts. We use the following versions for MySQL, Impala, and Neo4j: 5.7.31, 2.5.0-cdh5.7.0, 3.5.14, respectively.

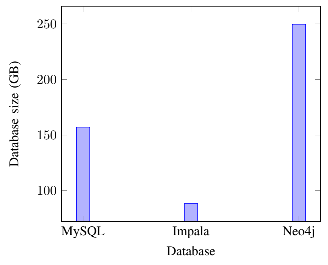
B. Query

We created four queries of varying complexity to run on each database. The queries are:

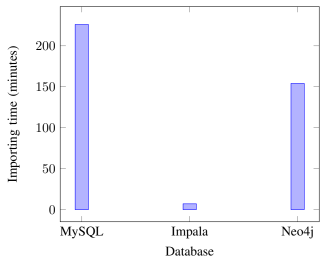
- Q1: Count the number of event logs in the database.
- Q2: Extract the event logs where the user $C101@DOM$ authenticates to a server.
- Q3: Extract the event logs where the user $C101@DOM$ authenticates to a server by starting a process.
- Q4: Extract the event logs where the user $C101@DOM$ authenticates to a server by starting a process and checking a DNS lookup of the source computer.

C. Results

Figure 3 shows the performance metrics of database on heterogeneous event logs in terms of database disk usage (Figure 3a) and importing time (Figure 3b) of logs from disk to database. The size of Neo4j database is 2,82X and 1,58X greater than Impala and MySQL, respectively. The importing time of Neo4j is 4.5X faster than MySQL, but 50X slower than Impala. In fact, Impala is based on HDFS where the logs are stored on HDFS. Thus, Impala inherits all Hadoop file system properties such as the importing time and file partitions. The user has just to put the logs in HDFS, create the tables and load log files from it. The loading operation moves the log files from HDFS directory to Impala directory. Thus, the importing time is equal to moving time from an HDFS folder to another one in an HDFS.



(a) Database size



(b) Importing time

Fig. 3: Performance metrics of database on heterogeneous event logs.

Figure 4 shows a comparison of the execution time of each query in each database. We see that Impala outperforms MySQL and Neo4j for the queries Q1, Q2, and Q3. However, Neo4j outperforms MySQL and Impala for the query Q4. We note that the queries Q1 and Q2 are simple queries without a JOIN operation, but the queries Q3 and Q4 are medium and complex queries, respectively with JOIN operations. Impala outperforms MySQL and Neo4j for the queries Q1, Q2, and Q3 is due to the partitioning of the data into blocks in the HDFS where a processing is performed in each block.

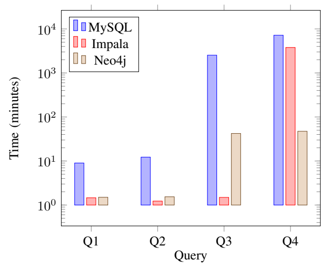


Fig. 4: Comparison of query execution time in each database.

VI. Discussions

Table III provides a comparison of the experimented databases on the basis of functional and nonfunctional features such as, database model, developed language of the database, query language used in the database, data storage, updating individual records. We notice that the use of Neo4j is quite difficult in terms of modeling data into a graph and having in mind a sub-graph query instead of rows or columns oriented queries. Neo4j requires an organization of the log files so that they can be imported into the graph database. The importing time requires several minutes to hours, depending on the size of the logs. It is difficult to find the best graph model for modeling the heterogeneous logs. Measuring the performance of different graph models is a challenging task that could be considered as a future work. In addition, the user has to learn the Cypher query language for retrieving patterns (sub-graph) in a graph database. In contrast, the use of relational database (MySQL) or column-oriented (Impala) is very easy in terms of modeling, querying without learning a new query language. Regarding the performance, Impala is well adapted for querying, monitoring and providing log analytics from heterogeneous security sources. However, Neo4j is suitable for querying complex patterns, but the storage of the graph database takes more than 2X the origin size while Impala keeps the same size of the origin logs. In fact, for using Impala, the user has just to put the logs in HDFS, create the tables and load log files from it. The loading operation moves the log files from HDFS directory to Impala directory. MySQL is not suitable for heterogeneous and large log files. Even a simple query takes time. In addition, MySQL is not suitable for heterogeneous and large logs. One of the characteristics of the heterogeneity is the change of the event log structure. The structure of the event log means that the number of fields can change during the life cycle of the database.

	MySQL	**Impala**	**Neo4J**
Database Model	Relational DBMS	HDFS/ HBase	Graph
Developped Language	C and C++	C++ and Java	Java and Scala
Query Language	SQL (full)	SQL (subset)	Cypher
Data storage	File system	HDFS	File system
Update individual records	Yes	No	Yes
Delete individual records	Yes	No	Yes
Index support	Extensive	Limited	Extensive
Scalability	No	Yes	Yes
Complexity of values	Low	Medium	High
Performance of queries	Low	High	Variable
Structure	Static	Dynamic	Dynamic
Easy to use	Yes	Yes	Variable
Easy to model	Yes	Yes	No

TABLE III: Comparison of databases.

VII. Conclusion

In this paper, we propose a study of the problem of storage and processing heterogeneous event logs incoming from multi-sources such as firewall, server, router. The study consists in comparing the performance of existing databases on heterogeneous event logs. We implement and compare SQL and NoSQL databases. A relational database using MySQL. A column-oriented database using Impala, and agraph database using Neo4j. The Key-value and document oriented databases are not suitable for the heterogeneous data. From our study, we conclude that the best database to store the large heterogeneous data is Hadoop/HDFS infrastructure. It keeps the same size as the original logs, easy to migrate the data, and easy to manipulate for further processing. We use Impala as a distributed SQL query engine for Apache Hadoop. Impala outperforms MySQL and Neo4j in terms of execution of queries, size of the logs on disk, the importing time, and easy to use. However, Neo4j outperforms Impala and MySQL in terms of execution time of complex queries. However, it's difficult to model event logs into graph models and the user has to learn the Cypher query language used in Neo4j.

References

[1] R. Angles and C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.

[2] M. Husák and M. Čermák. A graph-based representation of relations in network security alert sharing platforms. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 891–892, 2017.

[3] A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

[4] A. D. Kent. Cybersecurity data sources for dynamic network research. In *Dynamic Networks in Cybersecurity*. Imperial College Press, 2015.

[5] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for hadoop. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[6] K. Mahmood, T. Risch, and M. Zhu. Utilizing a nosql data store for scalable log analysis. In B. C. Desai and M. Toyama, editors, *Proceedings of the 19th International Database Engineering & Applications Symposium, Yokohama, Japan, July 13-15, 2015*, pages 49–55, 2015.

[7] J. Navarro, V. Legrand, S. Lagraa, J. François, A. Lahmadi, G. D. Santis, O. Festor, N. Lammari, F. Hamdi, A. Deruyver, Q. Goux, M. Allard, and P. Parrend. Huma: A multi-layer framework for threat analysis in a heterogeneous log environment. In *Foundations and Practice of Security - 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers*, volume 10723 of *Lecture Notes in Computer Science*, pages 144–159. Springer, 2017.

[8] S. Noel, E. Harley, K. H. Tam, and G. Gyor. Big-data architecture for cyber attack graphs representing security relationships in nosql graph databases, 2015.

[9] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly, 2015.

[10] X. Tao, Y. Liu, F. Zhao, C. Yang, and Y. Wang. Graph database-based network security situation awareness data storage method. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):294, Dec 2018.

Appendix

A. Impala

Listing 1 shows the creation of the log model in Impala. It creates a table form each source and loads `csv` files for each one using `LOAD DATA` statement. This statement streamlines the Extract Transform and Load (ETL) process for Impala tables by moving (not copied) all log files in a directory from an HDFS location into the Impala log directory for the created tables.

```
create table flows (timer int , duration float ,
    source_computer string , sourceport string ,
    destinationcomputer string , destinationport string ,
    protocol string , packet_count int , byte_count string)
    row format delimited fields terminated by ',';
create table auth (timer int , sourceuser string ,
    destinationuser string , sourcecomputer string ,
    destinationcomputer string , authenticationtype string ,
     logontype string , authenticationorientation string ,
    successfailure string) row format delimited fields
    terminated by ',';
create table dns (timer int , sourcecomputer string ,
    computerresolved string) row format delimited fields
    terminated by ',';
create table proc (timer int , user  string , computer
    string ,processname  string  ,  startend string) row
    format delimited fields terminated by ',';
load data  inpath '/HDFS/auth.txt' overwrite into table
    auth;
load data  inpath '/HDFS/proc.txt' overwrite into table
    proc;
load data  inpath '/HDFS/flows.txt' overwrite into table
    flows;
load data  inpath '/HDFS/dns.txt' overwrite into table dns;
```

Listing 1: Heterogeneous event logs to Impala-shell

B. Neo4j

The file logs are processed and organized into `CSV` files of nodes and relations of a graph in order to load them into Neo4j. Each node must have a unique ID such as the IP, the name of the computer or a name of a computer (`--nodes:name_of_the_node`) to be able to be referenced when creating relationships between nodes in the same import. Relationships are created by connecting the specified node IDs. For each relationship, a name of the relation is specified `--relationships:name_of_the_relation`. When dealing with very large log files, it is more practical to have their headers in a separate file. This makes it easier to edit or update the header by avoiding opening a large log file just to change a header. The header file of a log file must be specified with its log file during the import. Other Neo4j parameters are enabled for a high processing and loading of log files such as `--high-io=true` which specify that the storage system can support parallel IO with high throughput. It is true for SSDs, large raid arrays and network-attached storage. Other parameters are enabled such as `--ignore-duplicate-nodes=true`, `--ignore-missing-nodes=true` which ignore duplication, or missing nodes, respectively.

```
export LOGS=/path/to/folder/containing/txt-files/
export HEADERS=/path/to/folder/containing/txt-headers/
./bin/neo4j-admin import \
    --database=heterogeneousDB.db \
#authentication data
    --nodes:User
        $HEADERS/User-header.csv, \
        $LOGS/User.csv \
    --nodes:SourceComputer
        $HEADERS/SourceComputer-header.csv, \
        $LOGS/SourceComputer.csv \
    --nodes:DestinationComputer
        $HEADERS/DestinationComputer-header.csv, \
        $LOGS/DestinationComputer.csv \
    --relationships:USES
        $HEADERS/uses_rel-header.csv, \
        $LOGS/uses_rel.csv \
    --relationships:CONNECT
        $HEADERS/connect_rel-header.csv, \
        $LOGS/connect_rel.csv \
#process data
    --relationships:IS_RUNNING
        $HEADERS/is_running_rel-header.csv, \
        $LOGS/is_running_rel.csv \
#flow data
    --relationships:FLOWS
        $HEADERS/flows_rel-header.csv, \
        $LOGS/flows_rel.csv \
#dns data
    --nodes:ComputerResolved
        $HEADERS/ComputerResolved-header.csv, \
        $LOGS/ComputerResolved.csv \
    --relationships:IS_RESOLVED
        $HEADERS/is_rosolved_rel-header.csv, \
        $LOGS/is_rosolved_rel.csv \
# neo4j parameters
    --ignore-missing-nodes=true \
    --ignore-duplicate-nodes=true \
    --high-io=true
```

Listing 2: Heterogeneous event logs to graph database: Import Script