ETL Workflow using Apache NiFi

From external sources to Big Data (Hive)

Name: Asifuzzaman

Id: 0726261

# **Introduction:**

As stated in the proposal, the primary purpose of this project is to build an end-to-end data workflow, starting with data ingestion from an external source and loading it into Big Data storage. The project is developed with open source big data tools from Apache. All the required tools are installed and configured from scratch in a Linux environment to explore the mechanism of the Apache Big Data ecosystem.

The next section gives a brief description of the dataset and project task. The project breakdown is given in detail in the following sections, with an overview of tools. All the supporting tutorials and forum discussions are provided so that this document can be used as a development manual. Also, the technical challenges during the development process and their solution are described or highlighted in the sections. In addition, all the required configuration and coding files are attached to the project folder. Also, important code and configuration steps are mentioned in the glossary.

In the conclusions, an overall performance overview is provided with future improvement areas.

# **ETL Workflow Description:**

As part of the ETL process, the data needs to be fetched from UCI Machine Learning Repository via API (HTTPS link). The compressed file will be extracted. The Bitcoin Heist Ransomware Address Dataset contains transaction details from January 2009 to December 2018 with transactions more than B0.3. The dataset contains 2,916,697 records and ten attributes. The target column contains the name of families, including the Ransomware ones (e.g., Cryptxxx, cryptolocker etc). A new column needs to be added to flag the probable Ransomware transaction. Also, column names are changed to avoid system keywords (e.g. year, day, and count). After the required transformation, data will be inserted into the data warehouse for future use.

Dataset Description: **[Click Here](https://archive.ics.uci.edu/ml/datasets/BitcoinHeistRansomwareAddressDataset)**

# **Technology Stack:**

There are precompiled sandbox or VM images available with an exhaustive Big Data stack (Oracle Big Data VM or Cloudera Data Platform/Data Flow. However, often they are restricted to installing new components in the built-in package. Also, it requires a lot of memory and processors. Hence only required tools are installed.

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Name | Version | Download Link |
| Virtualization | VMWare Workstation Player | 16 | [Download Here](https://www.vmware.com/ca/products/workstation-player/workstation-player-evaluation.html) |
| OS | Ubuntu | 20.04 | [Download Here](https://ubuntu.com/download/desktop) |
| Bigdata framework | Apache Hadoop | 3.1.2 | [Download Here](https://archive.apache.org/dist/hadoop/common/hadoop-3.1.2/hadoop-3.1.2.tar.gz) |
| Database | Apache Hive | 3.1.2 | [Download Here](https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz) |
| ETL Tool | Apache NiFi | 1.16.0 | [Download Here](https://dlcdn.apache.org/nifi/1.16.0/nifi-1.16.0-bin.tar.gz) |

# **Set up Big Data:**

The following section briefly describes the required steps to configure Hadoop and Hive. Essential commands are added to the glossary section.

## **Install VMware and Os:**

VMware is installed on a Windows machine (Laptop with six cores and 16GB memory). Although the procedure is simple, the memory and process allocation should be done accordingly.

6GB of memory, three cores, and 30GB of space are allocated for this project.

VM and Ubuntu installation guide: **[Click Here](https://unixcop.com/how-to-install-ubuntu-21-04-on-vmware-workstation-pro/)**

## **Hadoop:**

As a prerequisite for Hadoop, JDK 8 and OpenSHH needs to be installed. A separate os user is created for better management and security. Hadoop is downloaded from the mentioned link. The linux bash profile and configuration files should be updated with caution. The required commands are given in the glossary.

Installation guide: **[Click Here](https://phoenixnap.com/kb/install-hadoop-ubuntu)**

## **Hive:**

Apache Hive is a data warehouse software project built on Apache Hadoop to provide SQL query and analysis features. In the backend, it runs a map-reduce process to extract data from HDFS. Hive is installed to store and manage the data for further analysis.

Hive requires a conventional relational DB to store the necessary metadata for its management. The default installation comes with a derby database. However, accessing Hive from external tools or servers requires concurrent sessions. HiveServer2 facilitates the necessary services. It requires MySQL database as the metadatabase. Special care should be given to download the correct version of MySQL JDBC connector. The detailed steps are given in the glossary.

Installation guide: **[Click Here](https://phoenixnap.com/kb/install-hive-on-ubuntu)**

HiveServer2 details: **[Click Here](https://cwiki.apache.org/confluence/display/Hive/Setting+Up+HiveServer2" \l "SettingUpHiveServer2-HiveServer2)**

HiveServer2 config: <https://youtu.be/BZAfoQMrkmk>

## **NiFi:**

Apache NiFi was built to automate the flow of data between systems. It supports almost all the databases and sources with a GUI-based data flow design facility that is easy to understand and manage. Also, the data flow can be saved and imported as a template to build redundant flows. It scales up the development time. NiFi is backed by ZooKeeper and can be worked in the distributed cluster.

NiFi is developed to manage huge data volumes with high throughput and low latency. It is advised to install NiFi on a separate server with dedicated raid space for logs and contents for the production environment. Also, the archive configuration and Java heap size need to be changed to run it smoothly. The log files need to be checked regularly for warnings.

Installation Guide: **[Click Here](https://nifi.apache.org/docs/nifi-docs/html/getting-started.html)**

# **ETL Workflow Development in NiFi:**

NiFi's fundamental design concepts closely relate to the main ideas of Flow-Based Programming. Data or "FlowFile" is moved from one step to another for required processing and transformation. Each task is completed by the "FlowFile Processor". Connection defines the relationship among processors.

Details Overview of NiFi: **[Click Here](https://nifi.apache.org/docs/nifi-docs/html/overview.html" \l "the-core-concepts-of-nifi)**

A screenshot of a computer

Description automatically generated

Figure 1 Nifi Processors Design - Final Workflow

As shown above, total 10 FlowFile Processors are configured to complete all the steps discussed in the ETL Workflow Description. The marked blue processors are for data fetching and storing in the file system. Data ingestion in Hive table is performed in two different methods (Marked blue and orange). The processes are described in the following sections.

## **InvokeHTTP:**

It is an HTTP client processor which can interact with a configurable HTTP endpoint. SSL certificate needs to be downloaded from the site and configured in the processor to fetch data from Rest API/HTTPs. The figure below shows the required configuration, including URL, HTTP Method (Get/post), SSL Context Service, and others. The required process to configure the processor and SSL certificate is given in the glossary.

Graphical user interface

Description automatically generated

## **UnpackContent:**

This processor takes a compressed file as an input and delivery uncompressed files as output. The compression type and file name can be filtered from this processor.

Graphical user interface, text, application, email

Description automatically generated

## **PutFile:**

The uncompressed file is forwarded to the PutFile processor to store it in the local file system.

Graphical user interface

Description automatically generated

## **Insert directly into Hive Table Using JDBC Connection:**

This method ingests data in the Hive table straight from the NiFi application using the Hive JDBC connection.

Twitter Data Example: [Click Here](https://www.velotio.com/engineering-blog/building-an-etl-workflow-using-apache-nifi-and-hive)

### **ExecutePythonScript (ExecuteStreamCommand):**

ExecuteStreamCommand processor can execute external commands on the content of the FlowFile and creates a new FlowFile with the results. The FlowFile content in the input can be accessed as STDIN, and the processor can forward STOUT from the command as an output to the next processor.

The below figure shows the configuration of the processor. It takes a python script as the command. The python script takes the STDIN and updates the dataset with an additional "Ransomware" flag column based on the label value.

In addition, it supports code blocks (Groovy, Jython, Javascript, JRuby) instead of the script from the local machine.

Graphical user interface

Description automatically generated

### **RenameHeader (ReplaceText):**

The ReplaceText processor is used to rename the header names with system keywords like year, day, and count.

Graphical user interface, website

Description automatically generated

### **FilterRecords (QueryRecords):**

QueryRecords processor can perform SQL-like queries directly on the FlowFile content. Also, the data format can be changed with this processor. In this case, the CSV format is converted into JSON for further processing compatibility. Record Reader/Writer value needs to be configured with the arrow sign on the right.

In this figure, a new property "data" is included with the "+" sign in the top right corner, and an SQL query is provided as an input. The SQL query should not have any ";" at the end as the processor.

Graphical user interface, table

Description automatically generated

### **ConvertJSONToSQL:**

This processor transforms each entry of the JSON file into an SQL INSERT statement. The database JDBC connection pool needs to be created for this processor. The detail of the configuration is given below. Database connection URL, Database user, and Password are provided. The path of hive-site.xml should be provided in the Hive configuration resources box. Although it is not a mandatory parameter, without the Validation query "Select 1 as Test\_column" the connection cannot be established.

Moreover, table and schema names need to be provided as input. Also, SQL parameter attributes have to be defined. In this case, "hiveql" is the correct input. The output FlowFile is a queue of insert statements. The hive table creation DDL is given in the glossary.

Graphical user interface, application

Description automatically generated

Table

Description automatically generated

### **PutHiveQL:**

It receives insert statements as input in the FlowFile and executes it in the Hive database through a JDBC connection.

Table

Description automatically generated

## **Insert Using External Hive Table:**

The aforementioned process executes single insert statements in a queue. It requires a lot of time due to JDBC connection overhead. In addition, Hive works differently than transactional databases and is not suitable for single insert statements. Hence, to improve the data insertion time below method is proposed using Hive external table functionality.

In this method, CSV data is transferred into a file location of HDFS. A Hive external table is defined in the database, which points to the same directory, and the table properties should match the columns and delimiter of the CSV file. Basically, the external table is an abstraction that presents the data in the CSV file as a table. However, it doesn't hold any information. The data will stay in the CSV file even if the external table architecture is dropped. Then an insert statement from the external table to the normal Hive table transfers all the data into the database. Since the operation happens within the HDFS, the execution time is much faster than the JDBC connection request.

### **ExecuteShellScript:**

A shell script is called using ExecuteStreamCommand processor to add the Ransomware flag column in the dataset and transfer the updated CSV file to the HDFS location.

Graphical user interface, website

Description automatically generated

### **InsertFromExtrenalTable:**

The SelectHiveQL has an additional property, "HiveQL Post-Query". This property is used in this step to execute the insert statement from the external to Hive table. For the primary "HiveQl Select Query", a dummy statement has been provided.

Graphical user interface, table

Description automatically generated

# **Results:**

The external table method takes less than a minute to insert 2.9Mn records in the Hive database. In contrast, the JDBC connection takes 5 minutes to execute a batch of 1000 insert statements. Hence external table method should be used for bulk data insertion in the data warehouse environment.

Table

Description automatically generated with low confidence

Text

Description automatically generated

# **Conclusion:**

Overall, NiFi is a reasonably simple ETL design tool. The GUI makes it easy to understand. The 200+ built-in processors serve all the purposes of modern data ingestion needs. The connections can hold the FlowFile in case of failure. It provides an efficient way to execute the workflow from the point of failure. Using Kafka, NiFi can serve the purpose of message queueing as well. Also, the custom script execution makes NiFi versatile to make any custom operations. However, it losses cache information if the primary node gets disconnected. NiFi cluster can solve this problem.

# **Glossary:**

## **Hadoop Installation:**

* + 1. **Install OpenJDK:**

sudo apt install openjdk-8-jdk -y

* + 1. **Install OpenSSH:**

sudo apt install openssh-server openssh-client -y

* + 1. **Add a new user for Hadoop:**

sudo adduser hadoop

* + 1. **Enable passwordless SSH:**

ssh-keygen -t rsa -P '' -f ~/.ssh/id\_rsa

cat ~/.ssh/id\_rsa.pub >> ~/.ssh/authorized\_keys

chmod 0600 ~/.ssh/authorized\_keys

* + 1. **Download and extract Hadoop:**

wget https://archive.apache.org/dist/hadoop/common/hadoop-3.1.2/hadoop-3.1.2.tar.gz

tar xzf hadoop-3.1.2.tar.gz

* + 1. **Single node Hadoop configure:**

**Open bash profile:**

nano .bashrc

**Include Hadoop config:**

#Hadoop Related Options

export HADOOP\_HOME=/home/hadoop/hadoop-3.2.1

export HADOOP\_INSTALL=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

export HADOOP\_OPTS"-Djava.library.path=$HADOOP\_HOME/lib/nativ"

**Apply change:**

source ~/.bashrc

**Open hadoop-env.sh:**

nano $HADOOP\_HOME/etc/hadoop/hadoop-env.sh

**Include Java Home:**

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

**Open core-site.xml:**

nano $HADOOP\_HOME/etc/hadoop/core-site.xml

**Include Namenode address and temporary directory:**

<configuration>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/hadoop/tmpdata</value>

</property>

<property>

<name>fs.default.name</name>

<value>hdfs://127.0.0.1:9000</value>

</property>

</configuration>

**Open hdfs-site.xml:**

nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml

**Include namenode and datanode directories and set replication factor:**

<configuration>

<property>

<name>dfs.data.dir</name>

<value>/home/hadoop/dfsdata/namenode</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/home/hadoop/dfsdata/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

</configuration>

**Open mapred-site.xml:**

nano $HADOOP\_HOME/etc/hadoop/mapred-site.xml

**Set default MapReduce framework to yarn:**

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Open yarn-site.xml:**

nano $HADOOP\_HOME/etc/hadoop/yarn-site.xml

**Configure yarn:**

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>127.0.0.1</value>

</property>

<property>

<name>yarn.acl.enable</name>

<value>0</value>

</property>

<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PERPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

</property>

</configuration>

**Format namenode:**

hdfs namenode -format

* + 1. **Start Hadoop :**

cd /home/hadoop/hadoop-3.1.2/sbin

**Start name and datanode:**

./start-dfs.sh

**Start yarn:**

./start-yarn.sh

**Check process:**

jps

* + 1. **Access Hadoop from browser:**

|  |  |
| --- | --- |
| namenode | <http://localhost:9870> |
| datanode | <http://localhost:9864> |
| yarn manager | <http://localhost:8088> |

## **Hive Installation:**

**Download and extract Hive**

wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz

sudo tar xvzf apache-hive-3.1.2-bin.tar.gz

sudo mv apache-hive-3.1.2-bin hive-3.1.2

**Configure Hive**

**Open bash profile:**

sudo nano .bashrc

**Configure Hive home:**

export HIVE\_HOME=/home/hadoop/hive-3.1.2

export PATH=$PATH:$HIVE\_HOME/bin

**Apply change:**

source ~/.bashrc

**Create Hive tmp Directories in HDFS:**

hdfs dfs -mkdir /tmp

hdfs dfs -chmod g+w /tmp

hdfs dfs -ls /

**Create Hive warehouse Directories in HDFS:**

hdfs dfs -mkdir -p /user/hive/warehouse

hdfs dfs -chmod g+w /user/hive/warehouse

hdfs dfs -ls /user/hive

**Open hive-env.sh:**

cd $HIVE\_HOME/conf

sudo cp hive-env.sh.template hive-env.sh

**Configure hive-env.sh:**

HADOOP\_HOME=/home/hadoop/hadoop-3.1.2

export HIVE\_CONF\_DIR=/home/hadoop/hive-3.1.2/conf

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64

**Install MySQL:**

sudo apt install mysql-server

**Check correct version of MySQL JDBC Connector:**

cd /usr/share/java

ls mysql-connector-java-8.0.28.jar

**Download if not found:**

wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.28.tar.gz

sudo tar xvzf mysql-connector-java-8.0.28.tar.gz

**Link between jar file and hive lib folder and copy jar to the lib:**

sudo ln -s /usr/share/java/mysql-connector-java-8.0.28.jar $HIVE\_HOME/lib/ mysql-connector-java-8.0.28.jar

**Create MySQL Database and User:**

sudo mysql

CREATE USER ETL@'%' identified by '1040';

CREATE USER 'ETL'@'localhost' identified by '1040';

CREATE DATABASE hive;

GRANT ALL PRIVILEGES ON hive.\* TO 'ETL'@'%';

GRANT ALL PRIVILEGES ON hive.\* TO 'ETL'@'localhost';

show databases;

use hive;

SOURCE /home/hadoop/apache-hive-3.1.2-bin/scripts/metastore/upgrade/mysql/hive-schema-3.1.0.mysql.sql

**Open hive-site.xml:**

cd $HIVE\_HOME/conf

cp hive-default.xml.template hive-site.xml

nano hive-site.xml

**Change below values in hive-site.xml:**

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://localhost/hive?createDatabaseIfNotExist=true</value>

<name>javax.jdo.option.ConnectionUserName</name>

<value>ETL</value>

<name>javax.jdo.option.ConnectionPassword</name>

<value>1040</value>

<name>javax.jdo.option.ConnectionDriverName</name>

<value> com.mysql.cj.jdbc.Driver</value>

<name>hive.metastore.warehouse.dir</name>

<value>/user/hive/warehouse</value>

<name>hive.metastore.uris</name>

<value>thrift://localhost:9083</value>

<name>hive.server2.enable.doAs</name>

<value>false</value>

<name>hive.exec.local.scratchdir</name>

<value>/tmp/${user.name}</value>

<name>hive.downloaded.resources.dir</name>

<value>/tmp/${user.name}\_resources</value>

<name>hive.server2.active.passive.ha.enable</name>

<value>true</value>

<name>hive.metastore.db.type</name>

<value>mysql</value>

<name>hive.metastore.event.db.notification.api.auth</name>

<value>false</value>

<name>hive.server2.thrift.bind.host</name>

<value>localhost</value>

**Add:**

<property>

<name>hive.server2.thrift.bind.port</name>

<value>100000</value>

<description>Bind host on which to run the HiveServer2 Thrift service.</description>

</property>

**Config hive-config.sh:**

cd $HIVE\_HOME/bin

# Allow alternate conf dir location.

HIVE\_CONF\_DIR="${HIVE\_CONF\_DIR:-$HIVE\_HOME/conf}"

export HIVE\_CONF\_DIR=$HIVE\_CONF\_DIR

export HIVE\_AUX\_JARS\_PATH=$HIVE\_AUX\_JARS\_PATH

# Default to use 256MB

export HADOOP\_HEAPSIZE=${HADOOP\_HEAPSIZE:-256}

export HADOOP\_HOME=/home/hadoop/hadoop-3.1.2

**Initiate MySQL:**

cd $HIVE\_HOME/bin

schematool -initSchema -dbType mysql

**Start Metastore Service and Server2:**

hive --service metastore

hiveserver2

**Access Hive from Browser:**

<http://localhost:10002/>

**Access Hive from Terminal:**

#cd $HIVE\_HOME/bin/

#hive

**beeline -u jdbc:hive2://localhost:10000**

**Create DATABASE in HIVE and test**

CREATE DATABASE ETL;

SHOW DATABASES;

USE ETL;

## **NiFi Installation:**

**Download NiFi:**

wget https://dlcdn.apache.org/nifi/1.16.0/nifi-1.16.0-bin.tar.gz

sudo tar xvzf nifi-1.16.0-bin.tar.gz

**Install as a Service:**

cd nifi-1.16.0/

bin/nifi.sh install

sudo service nifi start

**Change username and pass:**

./bin/nifi.sh set-single-user-credentials ETLU NiFiETLU1040

sudo service nifi stop

sudo service nifi start

**NiFi Portal:**

<https://localhost:8443/nifi/login>

**Change default properties:**

**Open nifi.properties and update below:**

nifi.content.repository.archive.max.retention.period=60 sec

nifi.content.repository.archive.max.usage.percentage=10%

**Comment below:**

#nifi.content.repository.directory.default=./content\_repository

nifi.content.repository.directory.contentS1R1 =

**Include Below:**

/home/hadoop/nifi\_content\_dir/d1

nifi.content.repository.directory.contentS1R2 = /home/hadoop/nifi\_content\_dir/d2

nifi.content.repository.directory.contentS1R3 = /home/hadoop/nifi\_content\_dir/d3

**Increase java heap:**

Open /home/hadoop/nifi-1.16.0/conf/ bootstrap.conf and update below

# JVM memory settings

java.arg.2=-Xms1024m

java.arg.3=-Xmx1024m

## **InvokeHTTP SSL Configuration:**

**Zip File Link:**

[**https://archive.ics.uci.edu/ml/machine-learning-databases/00526/data.zip**](https://archive.ics.uci.edu/ml/machine-learning-databases/00526/data.zip)

**Download SSL Certificate:**

**[https://medium.com/@menakajain/export-download-ssl-certificate-from-server-site- url-bcfc41ea46a2](https://medium.com/@menakajain/export-download-ssl-certificate-from-server-site-%09url-bcfc41ea46a2)**

**Include certificate in Java keystore:**

sudo keytool -trustcacerts -keystore /etc/ssl/certs/java/cacerts -storepass changeit - alias UCI -import -file /home/hadoop/Downloads/ics-uci-edu.pem

**Restart NiFi:**

sudo service nifi stop

sudo service nifi start

**Design NiFi flow with Rest API:** **[https://www.youtube.com/watch?v=Jk7H8w3evN0&list=PLLjJVrbVz7FnrQxYtHoDfx8C xUGL9PG\_C&index=14](https://www.youtube.com/watch?v=Jk7H8w3evN0&list=PLLjJVrbVz7FnrQxYtHoDfx8C%20%20%09xUGL9PG_C&index=14)**

**Graphical user interface, table

Description automatically generated**

## **Hive Table Creation:**

**Normal Hive Table:**

CREATE TABLE IF NOT EXISTS etl.uci\_ransomware (address string, year\_at int, day\_at int, length int, weight string, count\_of int, looped int, neighbors int, income string, label string, ransomware int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

CREATE TABLE IF NOT EXISTS etl.uci\_ransomware\_v2 (address string, year\_at int, day\_at int, length int, weight string, count\_of int, looped int, neighbors int, income string, label string, ransomware int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

**External Table HDFS Location and Structure:**

hdfs dfs -mkdir -p /user/hive/uci\_ransomware\_ext

hdfs dfs -chmod g+w /user/hive/uci\_ransomware\_ext

hdfs dfs -ls /user/hive

hdfs dfs -rm -R /user/hive/uci\_ransomware\_ext/BitcoinHeistData\_v2.csv

CREATE EXTERNAL TABLE IF NOT EXISTS etl.uci\_ransomware\_ext (address string, year\_at int, day\_at int, length int, weight string, count\_of int, looped int, neighbors int, income string, label string, ransomware int)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/user/hive/uci\_ransomware\_ext'

tblproperties ("skip.header.line.count"="1");

**Insert from External Table to Hive Table:**

INSERT OVERWRITE TABLE etl.uci\_ransomware\_v2 select \* from etl.uci\_ransomware\_ext

**Truncate table:**

truncate table uci\_ransomware;

truncate table uci\_ransomware\_v2;

## **HUE Installation:**

**Install Docker:** <https://docs.docker.com/engine/install/ubuntu/#install-from-a-package>

**Pull Hue:**

sudo docker run -it -p 8888:8888 gethue/hue:latest

## **Start Services:**

cd /home/hadoop/hadoop-3.1.2/sbin

./start-dfs.sh

./start-yarn.sh

hive --service metastore

hiveserver2

sudo service nifi start

## **Stop Services:**

stop-dfs.sh

stop-yarn.sh

sudo service nifi stop