# DATA SCIENCE TOOLS & TECHNIQUES

PROJECT REPORT

**SUBMITTED BY:**

Muhammad Atlas Malik, Muhammad Ahmed Qureshi, Kashaf Sajjad

**ROLL-NO:**

I24-8020, I24-8027, I24-8032

**INSTRUCTOR:**

Dr, Safdar Ali

# HADOOP INTRUSION DETECTION SYSTEM USING LOG ANALYSIS FOR NETWORK SECURITY

## ABSTRACT

In today's digital world, the volume of network traffic has increased a lot and is becoming complex day by day. So, securing the network infrastructure against cyber threats is very important. This project presents an efficient Intrusion Detection System (IDS) that detects anomalous behavior through network log analysis using big data technologies (Apache Flume, Hadoop, MapReduce and Apache Spark MLlib). The data was ingested into the Hadoop Distributed File System (HDFS) using Apache Flume. The MapReduce framework was used to preprocess the data and categorize network traffic based on ICMP, TCP and UDP protocols. The preprocessed dataset was loaded into Apache Spark and by using K-Means clustering algorithm from Spark MLlib, anomalous clusters were detected showing unusual network port activity. The project demonstrates how big data frameworks can be used to build a distributed intrusion detection system that is able to handle large-scale log data and provide timely alerts for network anomalies.

## INTRODUCTION

### Importance of intrusion detection in network security

The intrusion detection systems play a very important role in the domain of network security. They serve as an effective defense system against malicious activities and cyber-attacks. In today's digital world, the organizations and companies are heavily relying on interconnected systems so they are becoming more and more exposed to threats like malware, phishing and denial-of-service (DoS) attacks. Keeping this in view, the establishment of effective IDS mechanisms has become necessary for maintaining the confidentiality and integrity of the network resources. Following are the benefits of Intrusion Detection systems in Network Security:

- They detect unauthorized access and malicious activities in real-time.
- They help to identify potential threats like malware, DoS attacks and data breaches.
- They monitor the network traffic and system logs for potential anomalies.
- They support early threat detection and provide quick response to such incidences.
- They provide insights on how to improve overall network security.
- They improve the visibility of network behavior and are useful in post-attack forensic investigation.

### Challenges of analyzing large-scale log data

Following are the Challenges of analyzing large-scale log data:

- Logs may have a large volume in terabytes or petabytes. So, their storage and processing becomes difficult.

- The logs have various formats like JSON, text or XML etc. that require preprocessing.
- Sensitive data in logs require careful handling to avoid breaches during analysis.
- It is hard to detect threats quickly without efficient data streaming tools.
- Logs often contain irrelevant or duplicate data that further complicates the analysis.
- Traditional tools are not much scalable with increasing data volume and complexity.
- The analysis task require high computational resources and memory.
- It is complex task to integrate logs from various sources like servers, apps or network devices.

## Relevance of distributed computing in Cybersecurity

In Cybersecurity systems, the distributed computing frameworks help in the following ways:

- Hadoop HDFS is able to store a massive volume of security logs and Spark processes them in-memory for fast computation.
- Apache Spark's distributed processing enables real-time identification of suspicious patterns in network traffic.
- Hadoop and Spark easily scale across clusters to accommodate growing cybersecurity data.
- Both platforms use commodity hardware that helps to reduce infrastructure costs for large-scale security monitoring.
- With tools like Apache Flume and Spark Streaming, live data can be ingested and analyzed continuously.
- Hadoop's replication and Spark's resilient processing ensure reliable security analysis even during failures.
- Spark MLlib allows training and deployment of models for detecting network anomalies.
- Distributed systems can work along with intrusion detection systems for reliable protection.

## Objective of the project

The aim of our project is to build an efficient intrusion detection system using distributed computing technologies. Our target is to identify unusual network activity and improve network security through scalable big data analytics. Following are the objectives of this project:

- To develop a scalable Intrusion Detection System (IDS) using distributed computing frameworks.
- To inject real-time network data in HDFS using Apache Flume.
- To preprocess the logged data using MapReduce framework.
- To apply K-Means clustering algorithm from Apache Spark MLlib for detecting anomalous network port activity.
- To analyze patterns in network port usage and thus identify potential security threats.
- To demonstrate the advantage of distributed computing tools like Hadoop and Spark in handling large-scale log data.

## Dataset

In this project, we utilized the KDD Cup 1999 dataset from Kaggle. It was originally developed for The Third International Knowledge Discovery and Data Mining Tools Competition held in conjunction with KDD-99. The competition's main task was to build a predictive model that is able to differentiate between normal and malicious network connections which may be intrusions or attacks. In the dataset compressed file, there were a total of 9 files which are:

- corrected.gz
- kddcup.data.corrected
- kddcup.data.gz
- kddcup.data_10_percent.gz
- kddcup.data_10_percent_corrected
- kddcup.names
- kddcup.newtestdata_10_percent_unlabeled.gz
- kddcup.testdata.unlabeled.gz
- kddcup.testdata.unlabeled_10_percent.gz

We used the kddcup.data_10_percent_corrected file which is a subset of the full dataset containing 494,021 network connection records having both training and labeled test data for classification tasks.

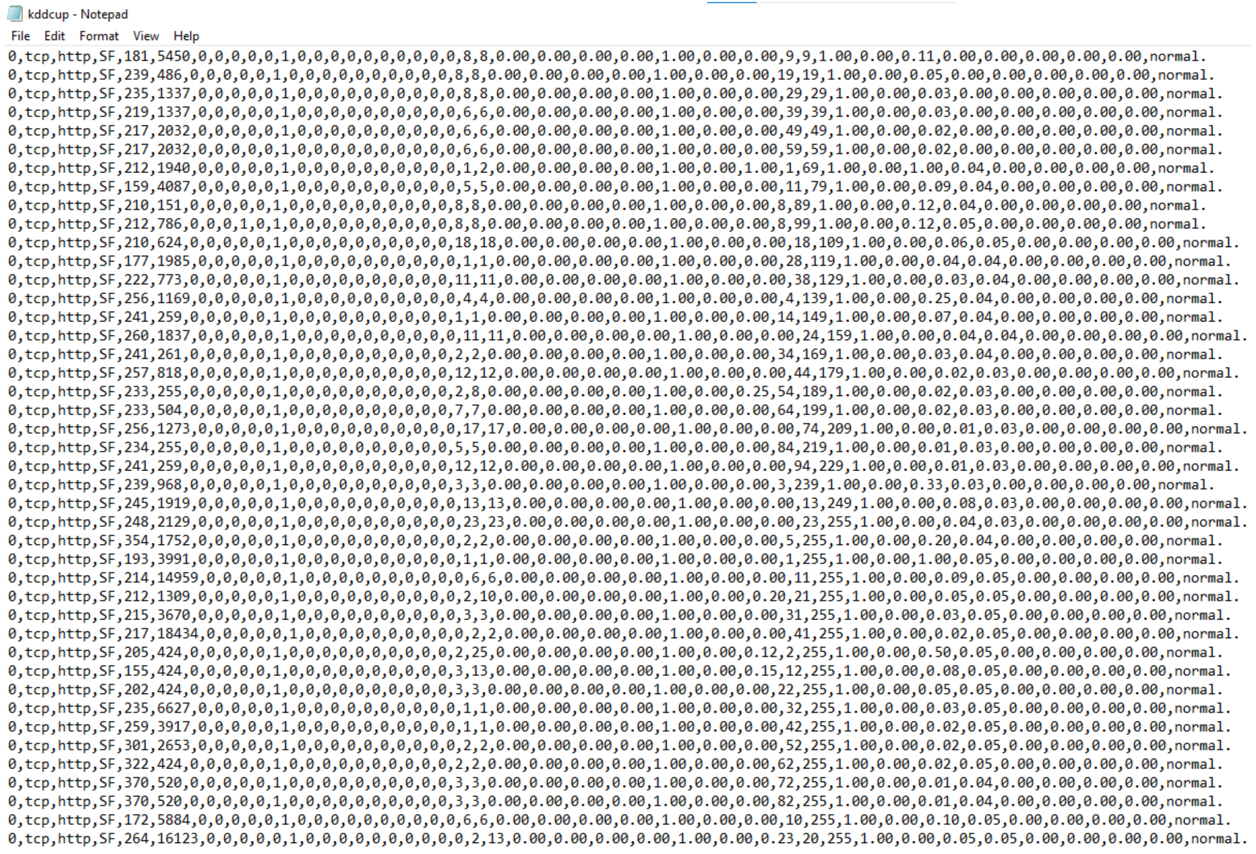
### *Structure of Dataset*

Each row in the dataset represents a single network connection and is structured as a comma-separated list of 42 columns which are:

- 41 features describing the connection
- 1 label indicating whether the connection is normal or an attack

The 41 features belong to different categories which are:

- Features that are basic like duration, protocol_type, service, flag, src_bytes and dst_bytes. These show the important characteristics of a connection.
- The content features like num_failed_logins, logged_in and root_shell. These help to detect suspicious behavior in a connection.
- Features related to time based traffic like count, srv_count and dst_host_count. These show the properties based on a time window of the past 2 seconds of connections.
- Host based traffic features like dst_host_srv_count and dst_host_same_src_port_rate. These capture patterns over the past 100 connections to the same host.



```
0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,49,49,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,59,59,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,212,1940,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,1,69,1.00,0.00,1.00,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,159,4087,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.00,1.00,0.00,0.00,11,79,1.00,0.00,0.09,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,210,151,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,89,1.00,0.00,0.12,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,212,786,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,99,1.00,0.00,0.12,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,210,624,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,18,18,0.00,0.00,0.00,0.00,1.00,0.00,0.00,18,109,1.00,0.00,0.06,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,177,1985,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,28,119,1.00,0.00,0.04,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,222,773,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,11,11,0.00,0.00,0.00,0.00,1.00,0.00,0.00,38,129,1.00,0.00,0.03,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,256,1169,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.00,1.00,0.00,0.00,4,139,1.00,0.00,0.25,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,241,259,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,14,149,1.00,0.00,0.07,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,260,1837,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,11,11,0.00,0.00,0.00,0.00,1.00,0.00,0.00,24,159,1.00,0.00,0.04,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,241,261,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,34,169,1.00,0.00,0.03,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,257,818,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,12,12,0.00,0.00,0.00,0.00,1.00,0.00,0.00,44,179,1.00,0.00,0.02,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,233,255,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,8,0.00,0.00,0.00,0.00,1.00,0.00,0.25,54,189,1.00,0.00,0.02,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,233,504,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,7,7,0.00,0.00,0.00,0.00,1.00,0.00,0.00,64,199,1.00,0.00,0.02,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,256,1273,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,17,17,0.00,0.00,0.00,0.00,1.00,0.00,0.00,74,209,1.00,0.00,0.01,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,234,255,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.00,1.00,0.00,0.00,84,219,1.00,0.00,0.01,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,241,259,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,12,12,0.00,0.00,0.00,0.00,1.00,0.00,0.00,94,229,1.00,0.00,0.01,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,239,968,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,3,239,1.00,0.00,0.33,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,245,1919,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,13,13,0.00,0.00,0.00,0.00,1.00,0.00,0.00,13,249,1.00,0.00,0.08,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,248,2129,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,23,23,0.00,0.00,0.00,0.00,1.00,0.00,0.00,23,255,1.00,0.00,0.04,0.03,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,354,1752,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,5,255,1.00,0.00,0.20,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,193,3991,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,1,255,1.00,0.00,1.00,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,214,14959,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,11,255,1.00,0.00,0.09,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,212,1309,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,10,0.00,0.00,0.00,0.00,1.00,0.00,0.20,21,255,1.00,0.00,0.05,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,215,3670,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,31,255,1.00,0.00,0.03,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,217,18434,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,41,255,1.00,0.00,0.02,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,205,424,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,25,0.00,0.00,0.00,0.00,1.00,0.00,0.12,2,255,1.00,0.00,0.50,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,155,424,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,13,0.00,0.00,0.00,0.00,1.00,0.00,0.15,12,255,1.00,0.00,0.08,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,202,424,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,22,255,1.00,0.00,0.05,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,235,6627,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,32,255,1.00,0.00,0.03,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,259,3917,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,42,255,1.00,0.00,0.02,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,301,2653,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,52,255,1.00,0.00,0.02,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,322,424,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,62,255,1.00,0.00,0.02,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,370,520,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,72,255,1.00,0.00,0.01,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,370,520,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,82,255,1.00,0.00,0.01,0.04,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,172,5884,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,10,255,1.00,0.00,0.10,0.05,0.00,0.00,0.00,0.00,normal.
0,tcp,http,SF,264,16123,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,13,0.00,0.00,0.00,0.00,1.00,0.00,0.23,20,255,1.00,0.00,0.05,0.05,0.00,0.00,0.00,0.00,normal.
```

## LITERATURE REVIEW

### Previous work on intrusion detection using machine learning

Over the recent years, machine learning has become a very common approach for developing Intrusion Detection Systems as the system can learn patterns from data with help of it and generalize to unseen data related to threats or malicious activity. There are a lot of research studies that have applied multiple ML algorithms to network intrusion detection so that they can improve the accuracy and speed of IDS.

One of the most commonly referenced study is the application of Decision Trees and Naive Bayes classifiers on the KDD Cup 1999 dataset. The results were good enough in detecting known attack patterns. The DTs especially gained popularity because of their interpretability and efficiency.

There are also studies related to use of Support Vector Machines in intrusion detection. SVMs are effective in handling high-dimensional data and by using them, the classification accuracy has also been improved. But the problem is that they are computationally expensive especially when the datasets are too large. Their scalability is reduced in real-time systems.

Other than these models, for complex classification tasks, Artificial Neural Networks (ANN), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown relatively high performance. It is seen from the studies that these models can automatically learn hierarchical features from raw data so we do not need to extract features manually. But the disadvantage is that these deep learning models require much more computational resources and large datasets for their training.

In different studies, the ensemble methods like Random Forests and Gradient Boosting have also been successfully applied to IDS. Using these methods, multiple classifiers are combined to improve performance and reduce overfitting. This helps to create a balance between accuracy and efficiency.

## Studies using the KDD dataset and clustering techniques

There are many studies that have utilized the KDD Cup 1999 dataset along with clustering techniques to develop efficient intrusion detection systems. In the network traffic, there might be patterns of anomalies that are identified using clustering methods like K-Means, DBSCAN and Hierarchical Clustering. K-Means has been commonly used as it is simple which involves grouping data points based on feature similarity that helps to detect abnormal behaviors. Also, DBSCAN has been proved to be effective in identifying noise and outliers without any need to predefine the number of clusters. This makes it suitable for detecting unknown attacks. There are studies in which some researchers have also combined clustering with classification techniques. In such IDS, clustering helps to group similar data and supervised models are used to classify the clusters which improves detection accuracy. Overall, clustering techniques have shown reasonable results in detecting anomalies and intrusions using the KDD dataset.

## Emergence of big data tools in IDS applications

The emergence of big data tools has significantly improved the performance of Intrusion Detection Systems (IDS). These tools help to store, process and analyze very large volumes of network traffic data in real time. Traditional IDS have problems related to scalability and performance when there is high-dimensional big data generated by modern networks. In order to solve these problems, people have adopted big data technologies such as Hadoop and Spark as these offer distributed computing capabilities and high-throughput data handling.

Hadoop has HDFS (Hadoop Distributed File System) for data storage and MapReduce model for processing. These frameworks help in the efficient parallel processing of network data so they prove to be best for intrusion detection tasks. Further, Apache Spark has in-memory data processing that speeds up the analysis, especially for iterative ML algorithms used in anomaly detection. These BDA tools allow IDS to analyze both structured and unstructured data from multiple sources so they become more efficient than traditional methods.

Also, by using big data frameworks, we have the option to integrate them with the required machine learning libraries for the particular task like MLlib, Mahout, HIVE, Flume, Sqoop and TensorFlow etc. By doing this, people have developed advanced IDS models that can learn from past attack patterns and help us avoid future threats. Now, it is possible to implement scalable and highly accurate intrusion detection systems suitable for modern enterprise and cloud environments. The organizations are being able to maintain good network security defenses by using the big data tools and thus, the cyber threats are dealt efficiently.

### Gaps addressed

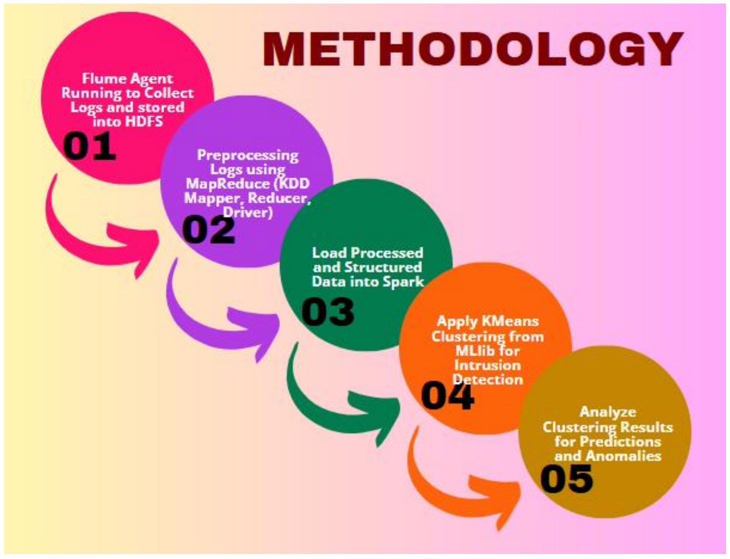
Following are the gaps addressed by the existing studies:

- Traditional IDS have difficulties in processing large-scale network log data in real time.
- Most existing systems lack scalability and are not optimized for distributed environments.
- Many studies focus only on classification and ignore the challenge of real-time log analysis.
- Mostly, IDS models do not use big data tools so they have slow performance when dataset becomes very large.
- Existing solutions may fail to detect new or evolving attack patterns effectively.

To address these gaps, in our project, we implement a Hadoop-based IDS using the KDD dataset. This enabled efficient distributed analysis for threat detection in network data.

## METHODOLOGY

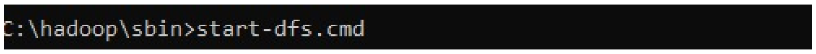
The methodology section revolves around building a scalable intrusion detection system (IDS) using big data tools. Firstly, we configured Apache Flume to inject network log data collected from Kaggle into Hadoop Distributed File System (HDFS). Then the data was processed using a MapReduce job consisting of custom mapper, reducer and driver classes. We loaded this structured data into spark where we used KMeans clustering algorithm from MLlib. In this way, we analyzed and made predictions related to anomalies in the KDD dataset.


METHODOLOGY
Flume Agent Running to Collect Logs and stored into HDFS
01
Preprocessing Logs using MapReduce (KDD Mapper, Reducer, Driver)
02
Load Processed and Structured Data into Spark
03
Apply KMeans Clustering from MLlib for Intrusion Detection
04
Analyze Clustering Results for Predictions and Anomalies
05

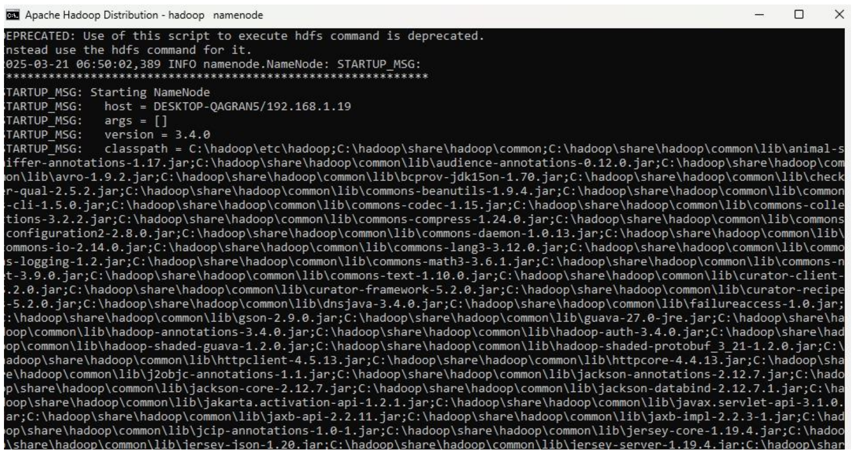
### Apache Flume Setup and Hadoop Setup

To begin the implementation of our project, we installed the Apache Flume as we needed to have a log collection service to stream log data and store in HDFS. We configured it by editing the file in which we defined the source, channel and sink components. The sink was connected to the HDFS so that the logs can be directly written into Hadoop. We also installed Hadoop and configured it so that we can log data in it and run the MapReduce jobs for preprocessing.

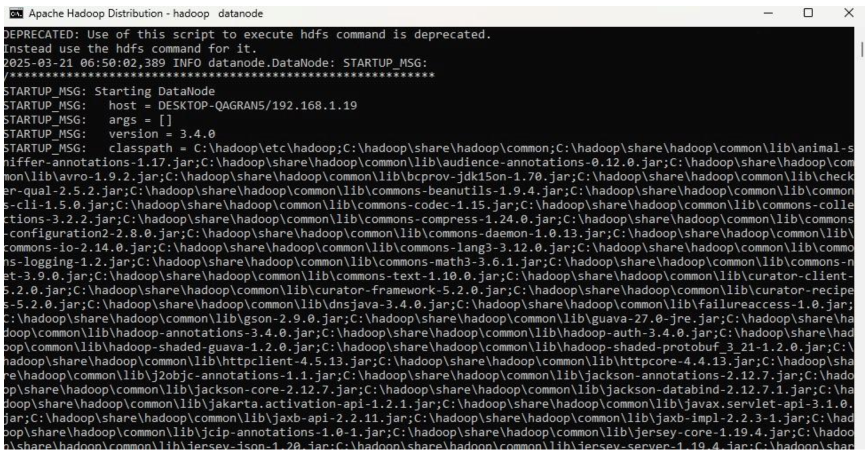
Starting HDFS:



```
C:\hadoop\sbin>start-dfs.cmd
```

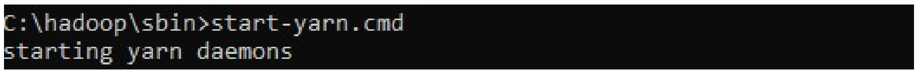


```
EPRECATED: Use of this script to execute hdfs command is deprecated.
nstead use the hdfs command for it.
025-03-21 06:50:02,389 INFO namenode.NameNode: STARTUP_MSG:
************************************************************
TARTUP_MSG: Starting NameNode
TARTUP_MSG:   host = DESKTOP-QAGRAN5/192.168.1.19
TARTUP_MSG:   args = []
TARTUP_MSG:   version = 3.4.0
TARTUP_MSG:   classpath = C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;C:\hadoop\share\hadoop\common\lib\animal-s
iffer-annotations-1.17.jar;C:\hadoop\share\hadoop\common\lib\audience-annotations-0.12.0.jar;C:\hadoop\share\hadoop\com
on\lib\avro-1.9.2.jar;C:\hadoop\share\hadoop\common\lib\bcprov-jdk15on-1.70.jar;C:\hadoop\share\hadoop\common\lib\check
r-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-beanutils-1.9.4.jar;C:\hadoop\share\hadoop\common\lib\common
-cli-1.5.0.jar;C:\hadoop\share\hadoop\common\lib\commons-codec-1.15.jar;C:\hadoop\share\hadoop\common\lib\commons-colle
tions-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-compress-1.24.0.jar;C:\hadoop\share\hadoop\common\lib\commons
configuration2-2.8.0.jar;C:\hadoop\share\hadoop\common\lib\commons-daemon-1.0.13.jar;C:\hadoop\share\hadoop\common\lib\
ommons-io-2.14.0.jar;C:\hadoop\share\hadoop\common\lib\commons-lang3-3.12.0.jar;C:\hadoop\share\hadoop\common\lib\commo
s-logging-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-math3-3.6.1.jar;C:\hadoop\share\hadoop\common\lib\commons-n
t-3.9.0.jar;C:\hadoop\share\hadoop\common\lib\commons-text-1.10.0.jar;C:\hadoop\share\hadoop\common\lib\curator-client-
.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-framework-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-recipe
-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\dnsjava-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\failureaccess-1.0.jar;
:\hadoop\share\hadoop\common\lib\gson-2.9.0.jar;C:\hadoop\share\hadoop\common\lib\guava-27.0-jre.jar;C:\hadoop\share\ha
oop\common\lib\hadoop-annotations-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-auth-3.4.0.jar;C:\hadoop\share\had
op\common\lib\hadoop-shaded-guava-1.2.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-shaded-protobuf_3_21-1.2.0.jar;C:\
adoop\share\hadoop\common\lib\httpclient-4.5.13.jar;C:\hadoop\share\hadoop\common\lib\httpcore-4.4.13.jar;C:\hadoop\sha
e\hadoop\common\lib\j2objc-annotations-1.1.jar;C:\hadoop\share\hadoop\common\lib\jackson-annotations-2.12.7.jar;C:\hado
p\share\hadoop\common\lib\jackson-core-2.12.7.jar;C:\hadoop\share\hadoop\common\lib\jackson-databind-2.12.7.1.jar;C:\ha
oop\share\hadoop\common\lib\jakarta.activation-api-1.2.1.jar;C:\hadoop\share\hadoop\common\lib\javax.servlet-api-3.1.0.
ar;C:\hadoop\share\hadoop\common\lib\jaxb-api-2.2.11.jar;C:\hadoop\share\hadoop\common\lib\jaxb-impl-2.2.3-1.jar;C:\had
op\share\hadoop\common\lib\jcip-annotations-1.0-1.jar;C:\hadoop\share\hadoop\common\lib\jersey-core-1.19.4.jar;C:\hadoo
\share\hadoop\common\lib\jersey-json-1.20.jar;C:\hadoop\share\hadoop\common\lib\jersey-server-1.19.4.jar;C:\hadoop\shar
```

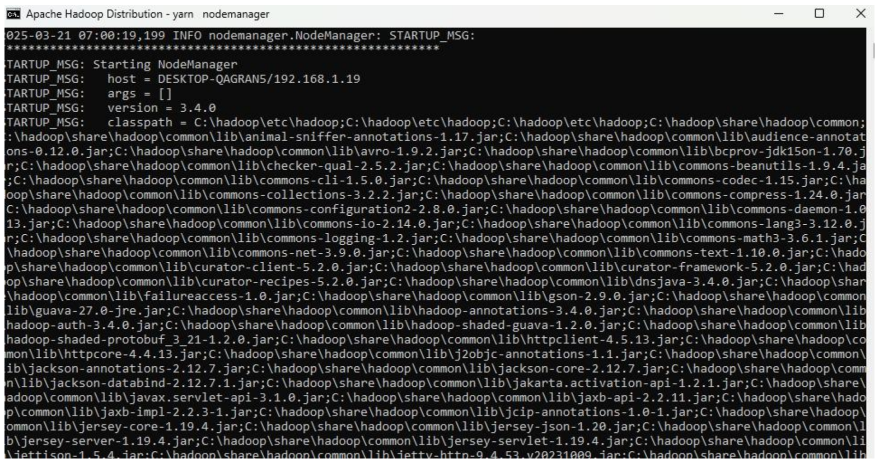


```
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
2025-03-21 06:50:02,389 INFO datanode.DataNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting DataNode
STARTUP_MSG:   host = DESKTOP-QAGRAN5/192.168.1.19
STARTUP_MSG:   args = []
STARTUP_MSG:   version = 3.4.0
STARTUP_MSG:   classpath = C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;C:\hadoop\share\hadoop\common\lib\animal-s
niffer-annotations-1.17.jar;C:\hadoop\share\hadoop\common\lib\audience-annotations-0.12.0.jar;C:\hadoop\share\hadoop\com
mon\lib\avro-1.9.2.jar;C:\hadoop\share\hadoop\common\lib\bcprov-jdk15on-1.70.jar;C:\hadoop\share\hadoop\common\lib\check
er-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-beanutils-1.9.4.jar;C:\hadoop\share\hadoop\common\lib\common
s-cli-1.5.0.jar;C:\hadoop\share\hadoop\common\lib\commons-codec-1.15.jar;C:\hadoop\share\hadoop\common\lib\commons-colle
ctions-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-compress-1.24.0.jar;C:\hadoop\share\hadoop\common\lib\commons
-configuration2-2.8.0.jar;C:\hadoop\share\hadoop\common\lib\commons-daemon-1.0.13.jar;C:\hadoop\share\hadoop\common\lib\
commons-io-2.14.0.jar;C:\hadoop\share\hadoop\common\lib\commons-lang3-3.12.0.jar;C:\hadoop\share\hadoop\common\lib\commo
ns-logging-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-math3-3.6.1.jar;C:\hadoop\share\hadoop\common\lib\commons-n
et-3.9.0.jar;C:\hadoop\share\hadoop\common\lib\commons-text-1.10.0.jar;C:\hadoop\share\hadoop\common\lib\curator-client-
5.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-framework-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-recipe
s-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\dnsjava-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\failureaccess-1.0.jar;
C:\hadoop\share\hadoop\common\lib\gson-2.9.0.jar;C:\hadoop\share\hadoop\common\lib\guava-27.0-jre.jar;C:\hadoop\share\ha
doop\common\lib\hadoop-annotations-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-auth-3.4.0.jar;C:\hadoop\share\had
oop\common\lib\hadoop-shaded-guava-1.2.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-shaded-protobuf_3_21-1.2.0.jar;C:\
hadoop\share\hadoop\common\lib\httpclient-4.5.13.jar;C:\hadoop\share\hadoop\common\lib\httpcore-4.4.13.jar;C:\hadoop\sha
re\hadoop\common\lib\j2objc-annotations-1.1.jar;C:\hadoop\share\hadoop\common\lib\jackson-annotations-2.12.7.jar;C:\hado
op\share\hadoop\common\lib\jackson-core-2.12.7.jar;C:\hadoop\share\hadoop\common\lib\jackson-databind-2.12.7.1.jar;C:\ha
doop\share\hadoop\common\lib\jakarta.activation-api-1.2.1.jar;C:\hadoop\share\hadoop\common\lib\javax.servlet-api-3.1.0.
jar;C:\hadoop\share\hadoop\common\lib\jaxb-api-2.2.11.jar;C:\hadoop\share\hadoop\common\lib\jaxb-impl-2.2.3-1.jar;C:\had
oop\share\hadoop\common\lib\jcip-annotations-1.0-1.jar;C:\hadoop\share\hadoop\common\lib\jersey-core-1.19.4.jar;C:\hadoo
p\share\hadoop\common\lib\jersey-json-1.20.jar;C:\hadoop\share\hadoop\common\lib\jersey-server-1.19.4.jar;C:\hadoop\shar
```

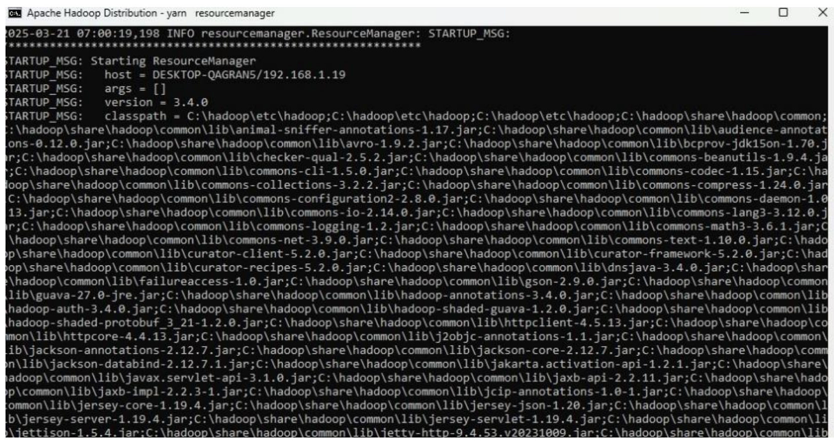
Starting YARN:



```
C:\hadoop\sbin>start-yarn.cmd
starting yarn daemons
```

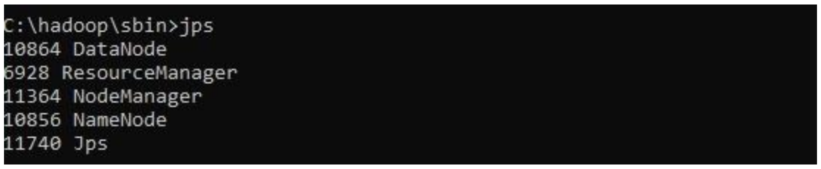


```
025-03-21 07:00:19,199 INFO nodemanager.NodeManager: STARTUP_MSG:
************************************************************
TARTUP_MSG: Starting NodeManager
TARTUP_MSG:   host = DESKTOP-QAGRAN5/192.168.1.19
TARTUP_MSG:   args = []
TARTUP_MSG:   version = 3.4.0
TARTUP_MSG:   classpath = C:\hadoop\etc\hadoop;C:\hadoop\etc\hadoop;C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;
:\hadoop\share\hadoop\common\lib\animal-sniffer-annotations-1.17.jar;C:\hadoop\share\hadoop\common\lib\audience-annotat
ons-0.12.0.jar;C:\hadoop\share\hadoop\common\lib\avro-1.9.2.jar;C:\hadoop\share\hadoop\common\lib\bcprov-jdk15on-1.70.j
r;C:\hadoop\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-beanutils-1.9.4.ja
;C:\hadoop\share\hadoop\common\lib\commons-cli-1.5.0.jar;C:\hadoop\share\hadoop\common\lib\commons-codec-1.15.jar;C:\ha
oop\share\hadoop\common\lib\commons-collections-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-compress-1.24.0.jar
C:\hadoop\share\hadoop\common\lib\commons-configuration2-2.8.0.jar;C:\hadoop\share\hadoop\common\lib\commons-daemon-1.0
13.jar;C:\hadoop\share\hadoop\common\lib\commons-io-2.14.0.jar;C:\hadoop\share\hadoop\common\lib\commons-lang3-3.12.0.j
r;C:\hadoop\share\hadoop\common\lib\commons-logging-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-math3-3.6.1.jar;C
\hadoop\share\hadoop\common\lib\commons-net-3.9.0.jar;C:\hadoop\share\hadoop\common\lib\commons-text-1.10.0.jar;C:\hado
p\share\hadoop\common\lib\curator-client-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-framework-5.2.0.jar;C:\had
op\share\hadoop\common\lib\curator-recipes-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\dnsjava-3.4.0.jar;C:\hadoop\shar
\hadoop\common\lib\failureaccess-1.0.jar;C:\hadoop\share\hadoop\common\lib\gson-2.9.0.jar;C:\hadoop\share\hadoop\common
lib\guava-27.0-jre.jar;C:\hadoop\share\hadoop\common\lib\hadoop-annotations-3.4.0.jar;C:\hadoop\share\hadoop\common\lib
hadoop-auth-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-shaded-guava-1.2.0.jar;C:\hadoop\share\hadoop\common\lib
hadoop-shaded-protobuf_3_21-1.2.0.jar;C:\hadoop\share\hadoop\common\lib\httpclient-4.5.13.jar;C:\hadoop\share\hadoop\co
mon\lib\httpcore-4.4.13.jar;C:\hadoop\share\hadoop\common\lib\j2objc-annotations-1.1.jar;C:\hadoop\share\hadoop\common\
ib\jackson-annotations-2.12.7.jar;C:\hadoop\share\hadoop\common\lib\jackson-core-2.12.7.jar;C:\hadoop\share\hadoop\comm
n\lib\jackson-databind-2.12.7.1.jar;C:\hadoop\share\hadoop\common\lib\jakarta.activation-api-1.2.1.jar;C:\hadoop\share\
adoop\common\lib\javax.servlet-api-3.1.0.jar;C:\hadoop\share\hadoop\common\lib\jaxb-api-2.2.11.jar;C:\hadoop\share\hado
p\common\lib\jaxb-impl-2.2.3-1.jar;C:\hadoop\share\hadoop\common\lib\jcip-annotations-1.0-1.jar;C:\hadoop\share\hadoop\
ommon\lib\jersey-core-1.19.4.jar;C:\hadoop\share\hadoop\common\lib\jersey-json-1.20.jar;C:\hadoop\share\hadoop\common\l
b\jersey-server-1.19.4.jar;C:\hadoop\share\hadoop\common\lib\jersey-servlet-1.19.4.jar;C:\hadoop\share\hadoop\common\li
\jettison-1.5.4.jar;C:\hadoop\share\hadoop\common\lib\jetty-http-9.4.53.v20231009.jar;C:\hadoop\share\hadoop\common\lib
```



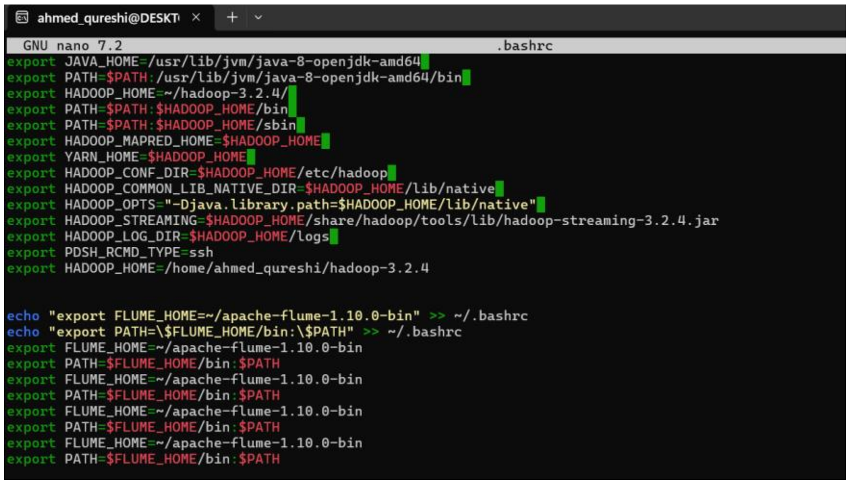
```
025-03-21 07:00:19,198 INFO resourcemanager.ResourceManager: STARTUP_MSG:
************************************************************
TARTUP_MSG: Starting ResourceManager
TARTUP_MSG:   host = DESKTOP-QAGRAN5/192.168.1.19
TARTUP_MSG:   args = []
TARTUP_MSG:   version = 3.4.0
TARTUP_MSG:   classpath = C:\hadoop\etc\hadoop;C:\hadoop\etc\hadoop;C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;
:\hadoop\share\hadoop\common\lib\animal-sniffer-annotations-1.17.jar;C:\hadoop\share\hadoop\common\lib\audience-annotat
ons-0.12.0.jar;C:\hadoop\share\hadoop\common\lib\avro-1.9.2.jar;C:\hadoop\share\hadoop\common\lib\bcprov-jdk15on-1.70.j
r;C:\hadoop\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-beanutils-1.9.4.ja
;C:\hadoop\share\hadoop\common\lib\commons-cli-1.5.0.jar;C:\hadoop\share\hadoop\common\lib\commons-codec-1.15.jar;C:\ha
oop\share\hadoop\common\lib\commons-collections-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-compress-1.24.0.jar
C:\hadoop\share\hadoop\common\lib\commons-configuration2-2.8.0.jar;C:\hadoop\share\hadoop\common\lib\commons-daemon-1.0
13.jar;C:\hadoop\share\hadoop\common\lib\commons-io-2.14.0.jar;C:\hadoop\share\hadoop\common\lib\commons-lang3-3.12.0.j
r;C:\hadoop\share\hadoop\common\lib\commons-logging-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-math3-3.6.1.jar;C
\hadoop\share\hadoop\common\lib\commons-net-3.9.0.jar;C:\hadoop\share\hadoop\common\lib\commons-text-1.10.0.jar;C:\hado
p\share\hadoop\common\lib\curator-client-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-framework-5.2.0.jar;C:\had
op\share\hadoop\common\lib\curator-recipes-5.2.0.jar;C:\hadoop\share\hadoop\common\lib\dnsjava-3.4.0.jar;C:\hadoop\shar
\hadoop\common\lib\failureaccess-1.0.jar;C:\hadoop\share\hadoop\common\lib\gson-2.9.0.jar;C:\hadoop\share\hadoop\common
lib\guava-27.0-jre.jar;C:\hadoop\share\hadoop\common\lib\hadoop-annotations-3.4.0.jar;C:\hadoop\share\hadoop\common\lib
hadoop-auth-3.4.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-shaded-guava-1.2.0.jar;C:\hadoop\share\hadoop\common\lib
hadoop-shaded-protobuf_3_21-1.2.0.jar;C:\hadoop\share\hadoop\common\lib\httpclient-4.5.13.jar;C:\hadoop\share\hadoop\co
mon\lib\httpcore-4.4.13.jar;C:\hadoop\share\hadoop\common\lib\j2objc-annotations-1.1.jar;C:\hadoop\share\hadoop\common\
ib\jackson-annotations-2.12.7.jar;C:\hadoop\share\hadoop\common\lib\jackson-core-2.12.7.jar;C:\hadoop\share\hadoop\comm
n\lib\jackson-databind-2.12.7.1.jar;C:\hadoop\share\hadoop\common\lib\jakarta.activation-api-1.2.1.jar;C:\hadoop\share\
adoop\common\lib\javax.servlet-api-3.1.0.jar;C:\hadoop\share\hadoop\common\lib\jaxb-api-2.2.11.jar;C:\hadoop\share\hado
p\common\lib\jaxb-impl-2.2.3-1.jar;C:\hadoop\share\hadoop\common\lib\jcip-annotations-1.0-1.jar;C:\hadoop\share\hadoop\
ommon\lib\jersey-core-1.19.4.jar;C:\hadoop\share\hadoop\common\lib\jersey-json-1.20.jar;C:\hadoop\share\hadoop\common\l
b\jersey-server-1.19.4.jar;C:\hadoop\share\hadoop\common\lib\jersey-servlet-1.19.4.jar;C:\hadoop\share\hadoop\common\li
\jettison-1.5.4.jar;C:\hadoop\share\hadoop\common\lib\jetty-http-9.4.53.v20231009.jar;C:\hadoop\share\hadoop\common\lib
```

Verifying running daemons:



```
C:\hadoop\sbin>jps
10864 DataNode
6928 ResourceManager
11364 NodeManager
10856 NameNode
11740 Jps
```

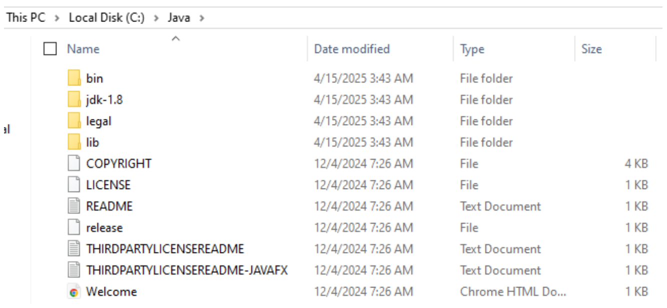
Flume and Hadoop Environment Setup:



```
  GNU nano 7.2                                              .bashrc
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:/usr/lib/jvm/java-8-openjdk-amd64/bin
export HADOOP_HOME=~/hadoop-3.2.4/
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export HADOOP_STREAMING=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.2.4.jar
export HADOOP_LOG_DIR=$HADOOP_HOME/logs
export PDSH_RCMD_TYPE=ssh
export HADOOP_HOME=/home/ahmed_qureshi/hadoop-3.2.4


echo "export FLUME_HOME=~/apache-flume-1.10.0-bin" >> ~/.bashrc
echo "export PATH=\$FLUME_HOME/bin:\$PATH" >> ~/.bashrc
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
```

## Apache Spark Environment Setup

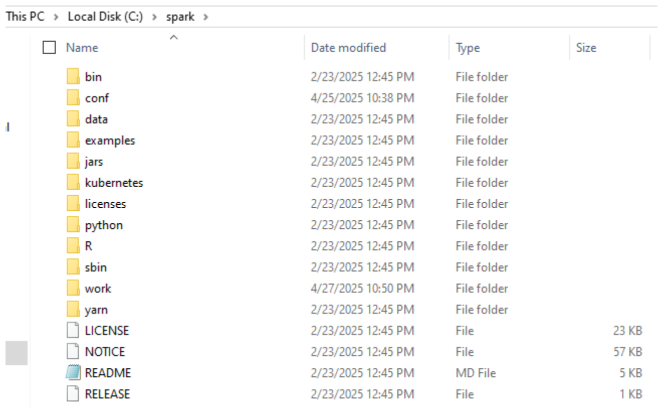
Firstly, we installed Java Development Kit (JDK)



This PC › Local Disk (C:) › Java ›

| Name | Date modified | Type | Size |
|---|---|---|---|
| bin | 4/15/2025 3:43 AM | File folder | |
| jdk-1.8 | 4/15/2025 3:43 AM | File folder | |
| legal | 4/15/2025 3:43 AM | File folder | |
| lib | 4/15/2025 3:43 AM | File folder | |
| COPYRIGHT | 12/4/2024 7:26 AM | File | 4 KB |
| LICENSE | 12/4/2024 7:26 AM | File | 1 KB |
| README | 12/4/2024 7:26 AM | Text Document | 1 KB |
| release | 12/4/2024 7:26 AM | File | 1 KB |
| THIRDPARTYLICENSEREADME | 12/4/2024 7:26 AM | Text Document | 1 KB |
| THIRDPARTYLICENSEREADME-JAVAFX | 12/4/2024 7:26 AM | Text Document | 1 KB |
| Welcome | 12/4/2024 7:26 AM | Chrome HTML Do... | 1 KB |

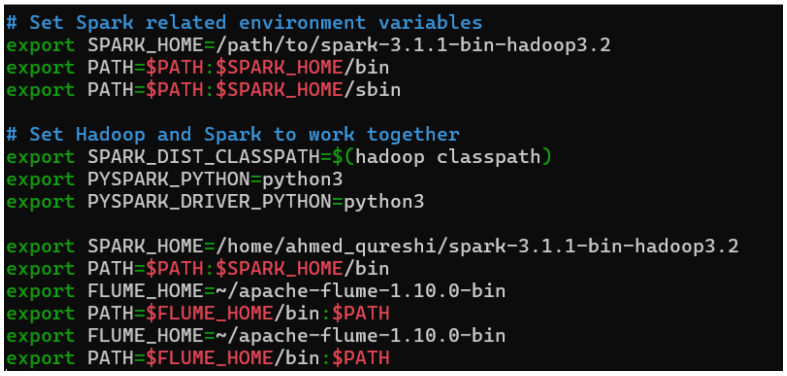
Then installed Apache Spark



This PC › Local Disk (C:) › spark ›

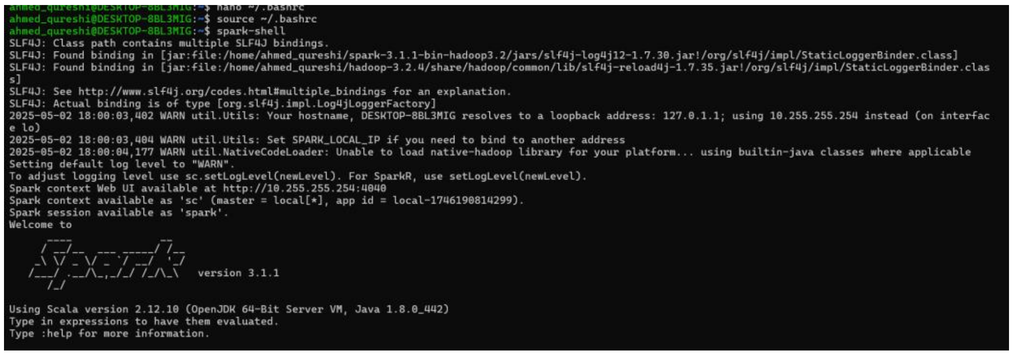
| Name | Date modified | Type | Size |
|---|---|---|---|
| bin | 2/23/2025 12:45 PM | File folder | |
| conf | 4/25/2025 10:38 PM | File folder | |
| data | 2/23/2025 12:45 PM | File folder | |
| examples | 2/23/2025 12:45 PM | File folder | |
| jars | 2/23/2025 12:45 PM | File folder | |
| kubernetes | 2/23/2025 12:45 PM | File folder | |
| licenses | 2/23/2025 12:45 PM | File folder | |
| python | 2/23/2025 12:45 PM | File folder | |
| R | 2/23/2025 12:45 PM | File folder | |
| sbin | 2/23/2025 12:45 PM | File folder | |
| work | 4/27/2025 10:50 PM | File folder | |
| yarn | 2/23/2025 12:45 PM | File folder | |
| LICENSE | 2/23/2025 12:45 PM | File | 23 KB |
| NOTICE | 2/23/2025 12:45 PM | File | 57 KB |
| README | 2/23/2025 12:45 PM | MD File | 5 KB |
| RELEASE | 2/23/2025 12:45 PM | File | 1 KB |

We set Environment Variables for Spark



```
# Set Spark related environment variables
export SPARK_HOME=/path/to/spark-3.1.1-bin-hadoop3.2
export PATH=$PATH:$SPARK_HOME/bin
export PATH=$PATH:$SPARK_HOME/sbin

# Set Hadoop and Spark to work together
export SPARK_DIST_CLASSPATH=$(hadoop classpath)
export PYSPARK_PYTHON=python3
export PYSPARK_DRIVER_PYTHON=python3

export SPARK_HOME=/home/ahmed_qureshi/spark-3.1.1-bin-hadoop3.2
export PATH=$PATH:$SPARK_HOME/bin
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
export FLUME_HOME=~/apache-flume-1.10.0-bin
export PATH=$FLUME_HOME/bin:$PATH
```

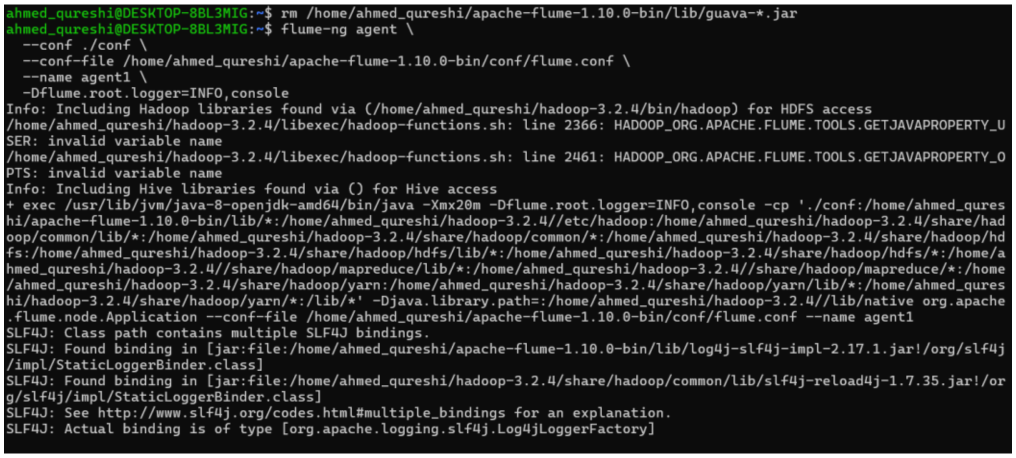
We Tested Spark Installation



```
ahmed_qureshi@DESKTOP-8BL3MIG:~$ nano ~/.bashrc
ahmed_qureshi@DESKTOP-8BL3MIG:~$ source ~/.bashrc
ahmed_qureshi@DESKTOP-8BL3MIG:~$ spark-shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ahmed_qureshi/spark-3.1.1-bin-hadoop3.2/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/common/lib/slf4j-reload4j-1.7.35.jar!/org/slf4j/impl/StaticLoggerBinder.clas
s]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
2025-05-02 18:00:03,402 WARN util.Utils: Your hostname, DESKTOP-8BL3MIG resolves to a loopback address: 127.0.1.1; using 10.255.255.254 instead (on interfac
e lo)
2025-05-02 18:00:03,404 WARN util.Utils: Set SPARK_LOCAL_IP if you need to bind to another address
2025-05-02 18:00:04,177 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://10.255.255.254:4040
Spark context available as 'sc' (master = local[*], app id = local-1746190814299).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_442)
Type in expressions to have them evaluated.
Type :help for more information.
```

## Running Apache Flume Agent

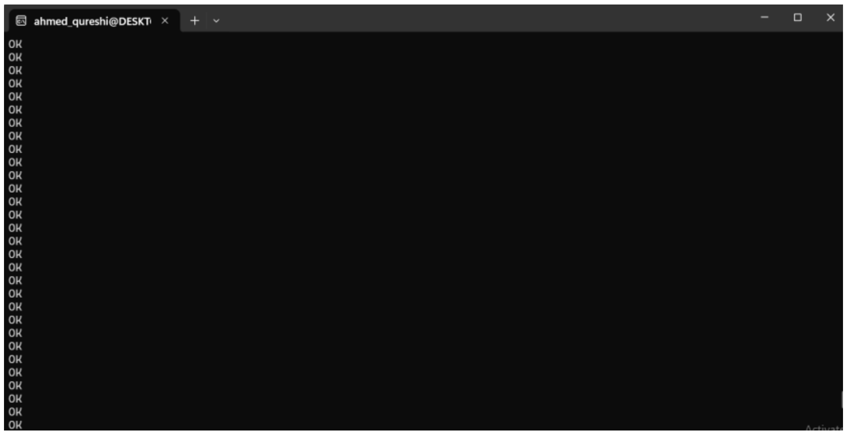
We executed the Apache Flume agent to transfer the network data taken from kaggle to the Hadoop Distributed File System (HDFS).



```
ahmed_qureshi@DESKTOP-8BL3MIG:~$ rm /home/ahmed_qureshi/apache-flume-1.10.0-bin/lib/guava-*.jar
ahmed_qureshi@DESKTOP-8BL3MIG:~$ flume-ng agent \
  --conf ./conf \
  --conf-file /home/ahmed_qureshi/apache-flume-1.10.0-bin/conf/flume.conf \
  --name agent1 \
  -Dflume.root.logger=INFO,console
Info: Including Hadoop libraries found via (/home/ahmed_qureshi/hadoop-3.2.4/bin/hadoop) for HDFS access
/home/ahmed_qureshi/hadoop-3.2.4/libexec/hadoop-functions.sh: line 2366: HADOOP_ORG.APACHE.FLUME.TOOLS.GETJAVAPROPERTY_U
SER: invalid variable name
/home/ahmed_qureshi/hadoop-3.2.4/libexec/hadoop-functions.sh: line 2461: HADOOP_ORG.APACHE.FLUME.TOOLS.GETJAVAPROPERTY_O
PTS: invalid variable name
Info: Including Hive libraries found via () for Hive access
+ exec /usr/lib/jvm/java-8-openjdk-amd64/bin/java -Xmx20m -Dflume.root.logger=INFO,console -cp './conf:/home/ahmed_qures
hi/apache-flume-1.10.0-bin/lib/*:/home/ahmed_qureshi/hadoop-3.2.4//etc/hadoop:/home/ahmed_qureshi/hadoop-3.2.4/share/had
oop/common/lib/*:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/common/*:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/hd
fs:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/hdfs/lib/*:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/hdfs/*:/home/a
hmed_qureshi/hadoop-3.2.4//share/hadoop/mapreduce/lib/*:/home/ahmed_qureshi/hadoop-3.2.4//share/hadoop/mapreduce/*:/home
/ahmed_qureshi/hadoop-3.2.4/share/hadoop/yarn:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/yarn/lib/*:/home/ahmed_qures
hi/hadoop-3.2.4/share/hadoop/yarn/*:/lib/*' -Djava.library.path=:/home/ahmed_qureshi/hadoop-3.2.4//lib/native org.apache
.flume.node.Application --conf-file /home/ahmed_qureshi/apache-flume-1.10.0-bin/conf/flume.conf --name agent1
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/ahmed_qureshi/apache-flume-1.10.0-bin/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j
/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/ahmed_qureshi/hadoop-3.2.4/share/hadoop/common/lib/slf4j-reload4j-1.7.35.jar!/or
g/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
```

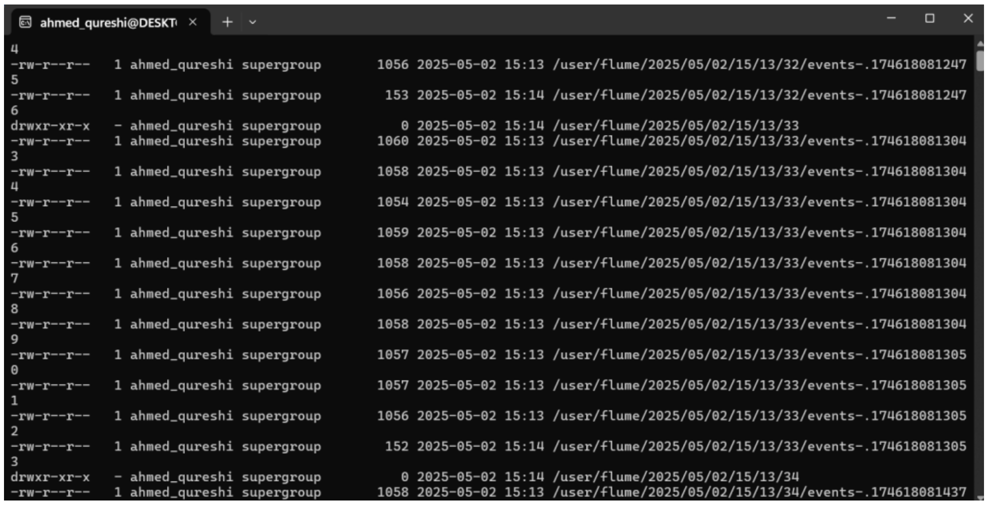
## Logging and Storing in HDFS

We processed the logs from Apache Flume to HDFS.



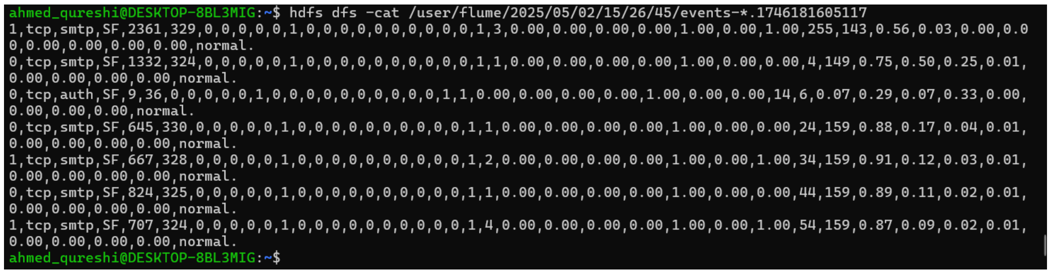
```
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
```

The logs were written on to the HDFS

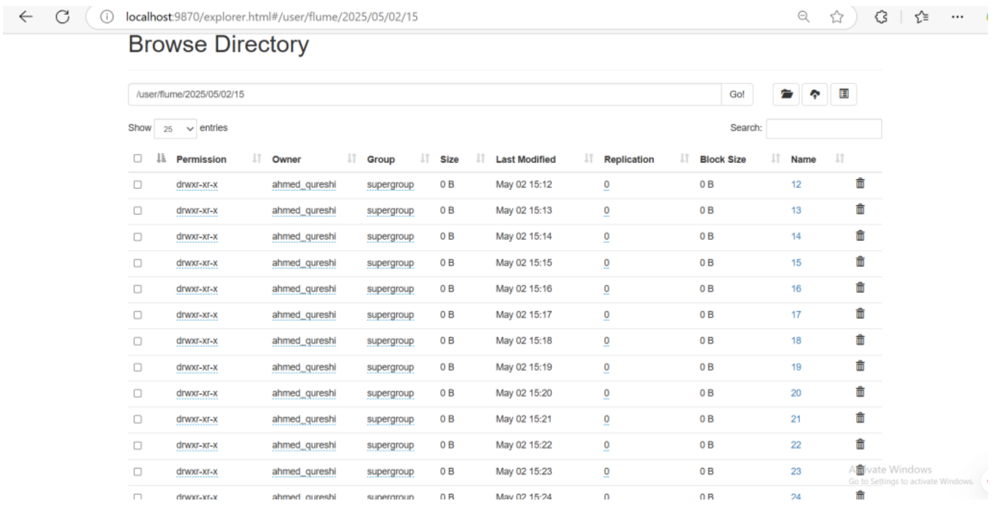


```
4
-rw-r--r--   1 ahmed_qureshi supergroup       1056 2025-05-02 15:13 /user/flume/2025/05/02/15/13/32/events-.174618081247
5
-rw-r--r--   1 ahmed_qureshi supergroup        153 2025-05-02 15:14 /user/flume/2025/05/02/15/13/32/events-.174618081247
6
drwxr-xr-x   - ahmed_qureshi supergroup          0 2025-05-02 15:14 /user/flume/2025/05/02/15/13/33
-rw-r--r--   1 ahmed_qureshi supergroup       1060 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
3
-rw-r--r--   1 ahmed_qureshi supergroup       1058 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
4
-rw-r--r--   1 ahmed_qureshi supergroup       1054 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
5
-rw-r--r--   1 ahmed_qureshi supergroup       1059 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
6
-rw-r--r--   1 ahmed_qureshi supergroup       1058 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
7
-rw-r--r--   1 ahmed_qureshi supergroup       1056 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
8
-rw-r--r--   1 ahmed_qureshi supergroup       1058 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081304
9
-rw-r--r--   1 ahmed_qureshi supergroup       1057 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081305
0
-rw-r--r--   1 ahmed_qureshi supergroup       1057 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081305
1
-rw-r--r--   1 ahmed_qureshi supergroup       1056 2025-05-02 15:13 /user/flume/2025/05/02/15/13/33/events-.174618081305
2
-rw-r--r--   1 ahmed_qureshi supergroup        152 2025-05-02 15:14 /user/flume/2025/05/02/15/13/33/events-.174618081305
3
drwxr-xr-x   - ahmed_qureshi supergroup          0 2025-05-02 15:14 /user/flume/2025/05/02/15/13/34
-rw-r--r--   1 ahmed_qureshi supergroup       1058 2025-05-02 15:13 /user/flume/2025/05/02/15/13/34/events-.174618081437
```

We viewed the contents of the logs written using flume



```
ahmed_qureshi@DESKTOP-8BL3MIG:~$ hdfs dfs -cat /user/flume/2025/05/02/15/26/45/events-*.1746181605117
1,tcp,smtp,SF,2361,329,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,3,0.00,0.00,0.00,0.00,1.00,0.00,1.00,255,143,0.56,0.03,0.00,0.0
0,0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,1332,324,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,4,149,0.75,0.50,0.25,0.01,
0.00,0.00,0.00,0.00,normal.
0,tcp,auth,SF,9,36,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,14,6,0.07,0.29,0.07,0.33,0.00,
0.00,0.00,0.00,normal.
0,tcp,smtp,SF,645,330,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,24,159,0.88,0.17,0.04,0.01,
0.00,0.00,0.00,0.00,normal.
1,tcp,smtp,SF,667,328,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,34,159,0.91,0.12,0.03,0.01,
0.00,0.00,0.00,0.00,normal.
0,tcp,smtp,SF,824,325,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,44,159,0.89,0.11,0.02,0.01,
0.00,0.00,0.00,0.00,normal.
1,tcp,smtp,SF,707,324,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,4,0.00,0.00,0.00,0.00,1.00,0.00,1.00,54,159,0.87,0.09,0.02,0.01,
0.00,0.00,0.00,0.00,normal.
ahmed_qureshi@DESKTOP-8BL3MIG:~$
```



localhost:9870/explorer.html#/user/flume/2025/05/02/15

Browse Directory

/user/flume/2025/05/02/15 Go!

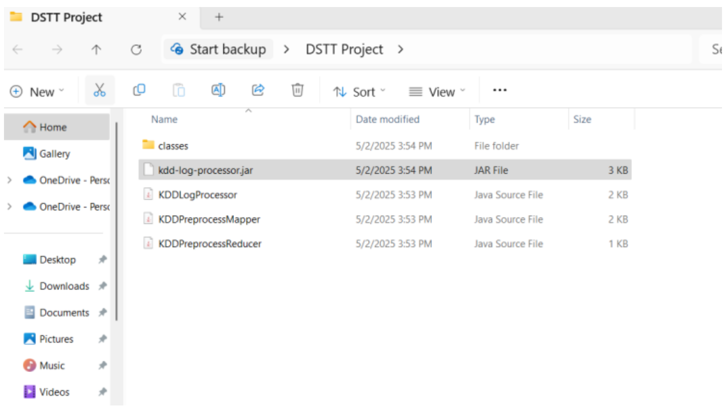
Show 25 entries Search:

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:12 | 0 | 0 B | 12 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:13 | 0 | 0 B | 13 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:14 | 0 | 0 B | 14 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:15 | 0 | 0 B | 15 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:16 | 0 | 0 B | 16 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:17 | 0 | 0 B | 17 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:18 | 0 | 0 B | 18 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:19 | 0 | 0 B | 19 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:20 | 0 | 0 B | 20 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:21 | 0 | 0 B | 21 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:22 | 0 | 0 B | 22 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:23 | 0 | 0 B | 23 |
| drwxr-xr-x | ahmed_qureshi | supergroup | 0 B | May 02 15:24 | 0 | 0 B | 24 |

## Preprocessing with MapReduce

For the processing part, we used the MapReduce framework having two main components that are the Mapper and the Reducer. The Mapper read each log entry, extract relevant features such as categorizing network protocol and emit intermediate key-value pairs. These outputs are then grouped and passed to the Reducer which aggregates the data.

The KDDMapper, KDDReducer and KDDDriver files are shown in the directory:



| Name | Date modified | Type | Size |
|---|---|---|---|
| classes | 5/2/2025 3:54 PM | File folder | |
| kdd-log-processor.jar | 5/2/2025 3:54 PM | JAR File | 3 KB |
| KDDLogProcessor | 5/2/2025 3:53 PM | Java Source File | 2 KB |
| KDDPreprocessMapper | 5/2/2025 3:53 PM | Java Source File | 2 KB |
| KDDPreprocessReducer | 5/2/2025 3:53 PM | Java Source File | 1 KB |

### ***KDD Log Processor***

The KDDLogProcessor is the driver class for a Hadoop MapReduce job designed to process the KDD dataset. We begin it by setting up a new Hadoop configuration and initializing a job named KDD Log Processor. This job specifies the main class, the Mapper class and the Reducer class. We defined the output key as Text and value types as IntWritable. We passed the input and output paths for the job as arguments so the data is selected easily. We submit the job for execution and the system exits based on its success or failure.

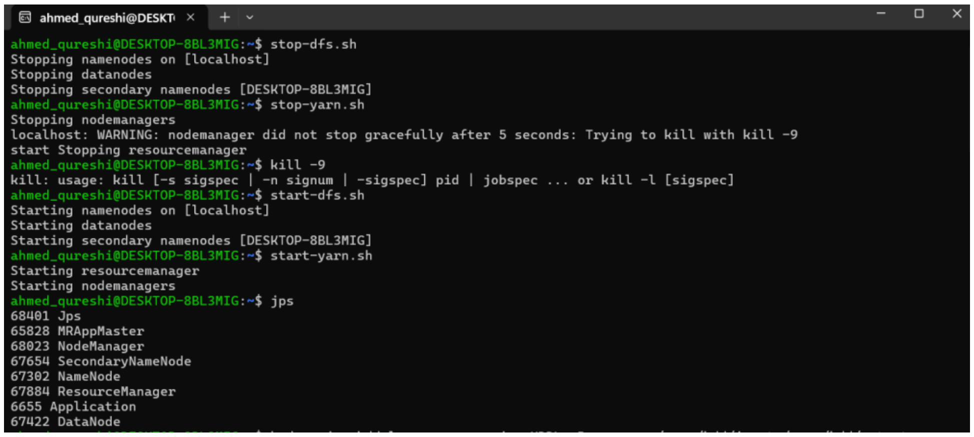
### ***KDD Mapper***

We defined the KDDPreprocessMapper class to implement mapper function of the Hadoop MapReduce framework. We processed each line/record of the KDD dataset and splitted the line by commas to extract individual fields. We specifically targeted the second field that represents the protocol types like TCP, UDP and ICMP and used it as the key. When each record of protocol type appears, a key-value pair is generated where the key is the protocol and the value is the integer 1. This mapping phase helped us to count the frequency of different protocol types in the dataset. Thus, we prepared the data for aggregation in the subsequent reduce phase.

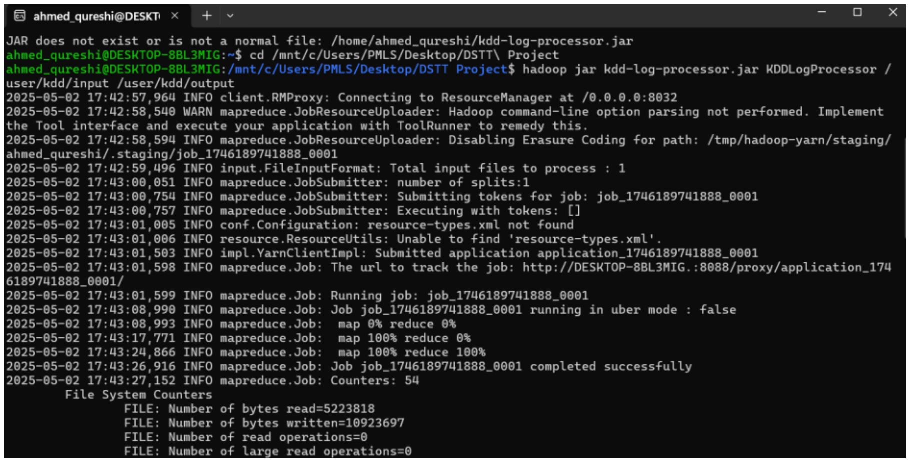
### ***KDD Reducer***

WE defined the KDDPreprocessReducer class to implement Reducer function in a Hadoop MapReduce job. Using this, we aggregated the results from the KDDPreprocessMapper class. For each unique key representing a specific protocol type, the reducer receives an iterable list of integer values each being 1 from the mapper. It takes the sum of these values to calculate the total number of times each protocol appears in the dataset. The final output is a key-value pair where the key is the protocol type and the value is the total count. This displayed a summarized view of protocol distribution in the KDD dataset.

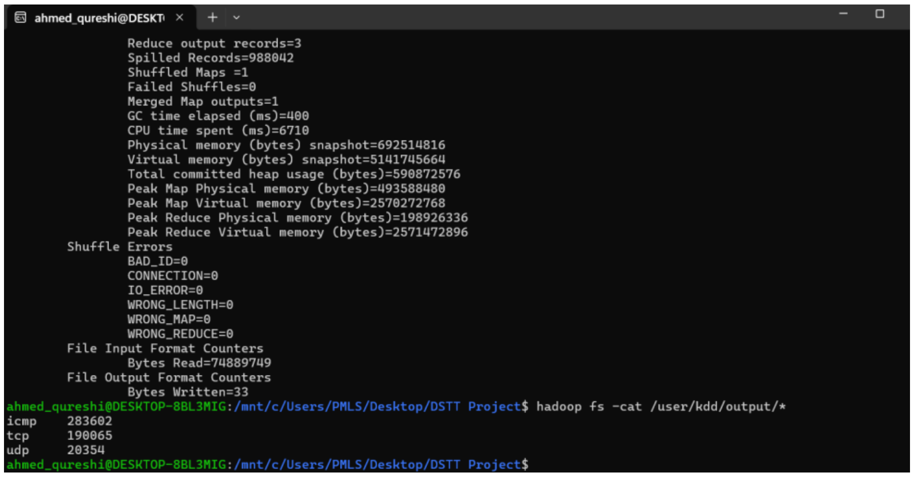
The Map reduce job is shown in figures.



```
ahmed_qureshi@DESKTOP-8BL3MIG:~$ stop-dfs.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [DESKTOP-8BL3MIG]
ahmed_qureshi@DESKTOP-8BL3MIG:~$ stop-yarn.sh
Stopping nodemanagers
localhost: WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kill -9
start Stopping resourcemanager
ahmed_qureshi@DESKTOP-8BL3MIG:~$ kill -9
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
ahmed_qureshi@DESKTOP-8BL3MIG:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [DESKTOP-8BL3MIG]
ahmed_qureshi@DESKTOP-8BL3MIG:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
ahmed_qureshi@DESKTOP-8BL3MIG:~$ jps
68401 Jps
65828 MRAppMaster
68023 NodeManager
67654 SecondaryNameNode
67302 NameNode
67884 ResourceManager
6655 Application
67422 DataNode
```



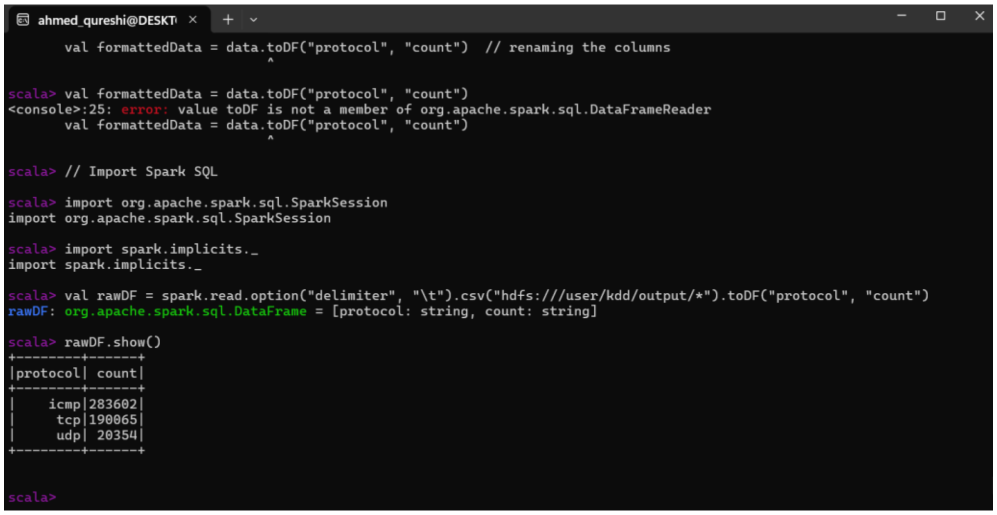
```
JAR does not exist or is not a normal file: /home/ahmed_qureshi/kdd-log-processor.jar
ahmed_qureshi@DESKTOP-8BL3MIG:~$ cd /mnt/c/Users/PMLS/Desktop/DSTT\ Project
ahmed_qureshi@DESKTOP-8BL3MIG:/mnt/c/Users/PMLS/Desktop/DSTT Project$ hadoop jar kdd-log-processor.jar KDDLogProcessor /
user/kdd/input /user/kdd/output
2025-05-02 17:42:57,964 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2025-05-02 17:42:58,540 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement
the Tool interface and execute your application with ToolRunner to remedy this.
2025-05-02 17:42:58,594 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
ahmed_qureshi/.staging/job_1746189741888_0001
2025-05-02 17:42:59,496 INFO input.FileInputFormat: Total input files to process : 1
2025-05-02 17:43:00,051 INFO mapreduce.JobSubmitter: number of splits:1
2025-05-02 17:43:00,754 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1746189741888_0001
2025-05-02 17:43:00,757 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-05-02 17:43:01,005 INFO conf.Configuration: resource-types.xml not found
2025-05-02 17:43:01,006 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2025-05-02 17:43:01,503 INFO impl.YarnClientImpl: Submitted application application_1746189741888_0001
2025-05-02 17:43:01,598 INFO mapreduce.Job: The url to track the job: http://DESKTOP-8BL3MIG.:8088/proxy/application_174
6189741888_0001/
2025-05-02 17:43:01,599 INFO mapreduce.Job: Running job: job_1746189741888_0001
2025-05-02 17:43:08,990 INFO mapreduce.Job: Job job_1746189741888_0001 running in uber mode : false
2025-05-02 17:43:08,993 INFO mapreduce.Job:  map 0% reduce 0%
2025-05-02 17:43:17,771 INFO mapreduce.Job:  map 100% reduce 0%
2025-05-02 17:43:24,866 INFO mapreduce.Job:  map 100% reduce 100%
2025-05-02 17:43:26,916 INFO mapreduce.Job: Job job_1746189741888_0001 completed successfully
2025-05-02 17:43:27,152 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=5223818
                FILE: Number of bytes written=10923697
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
```



```
                Reduce output records=3
                Spilled Records=988042
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=400
                CPU time spent (ms)=6710
                Physical memory (bytes) snapshot=692514816
                Virtual memory (bytes) snapshot=5141745664
                Total committed heap usage (bytes)=590872576
                Peak Map Physical memory (bytes)=493588480
                Peak Map Virtual memory (bytes)=2570272768
                Peak Reduce Physical memory (bytes)=198926336
                Peak Reduce Virtual memory (bytes)=2571472896
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=74889749
        File Output Format Counters
                Bytes Written=33
ahmed_qureshi@DESKTOP-8BL3MIG:/mnt/c/Users/PMLS/Desktop/DSTT Project$ hadoop fs -cat /user/kdd/output/*
icmp    283602
tcp     190065
udp     20354
ahmed_qureshi@DESKTOP-8BL3MIG:/mnt/c/Users/PMLS/Desktop/DSTT Project$
```

## Loading Preprocessed Data in Spark

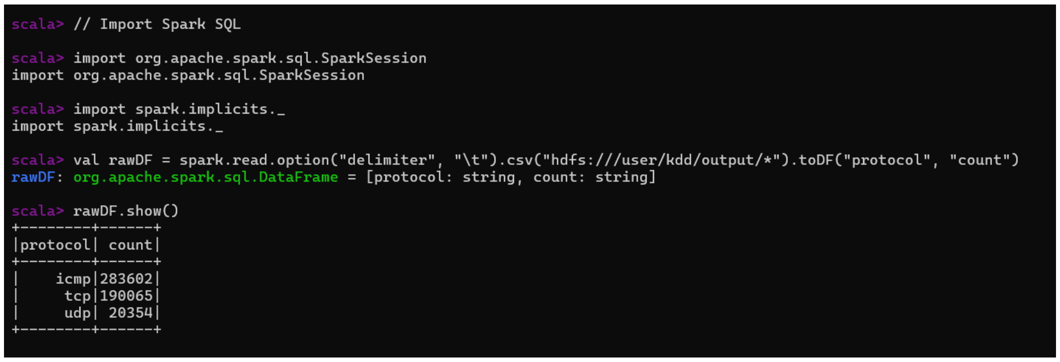
After the preprocessing of data using Hadoop MapReduce, we loaded the structured data into Apache Spark for further analysis. We read the MapReduce output files stored in HDFS using Spark's built-in function which is spark.read.csv().After loading data in Spark DataFrame or RDD, the Spark's powerful in-memory processing capabilities are utilized to perform data preparation for clustering task of machine learning.



```
       val formattedData = data.toDF("protocol", "count")  // renaming the columns
                                    ^

scala> val formattedData = data.toDF("protocol", "count")
<console>:25: error: value toDF is not a member of org.apache.spark.sql.DataFrameReader
       val formattedData = data.toDF("protocol", "count")
                                    ^

scala> // Import Spark SQL

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import spark.implicits._
import spark.implicits._

scala> val rawDF = spark.read.option("delimiter", "\t").csv("hdfs:///user/kdd/output/*").toDF("protocol", "count")
rawDF: org.apache.spark.sql.DataFrame = [protocol: string, count: string]

scala> rawDF.show()
+--------+------+
|protocol| count|
+--------+------+
|    icmp|283602|
|     tcp|190065|
|     udp| 20354|
+--------+------+


scala>
```

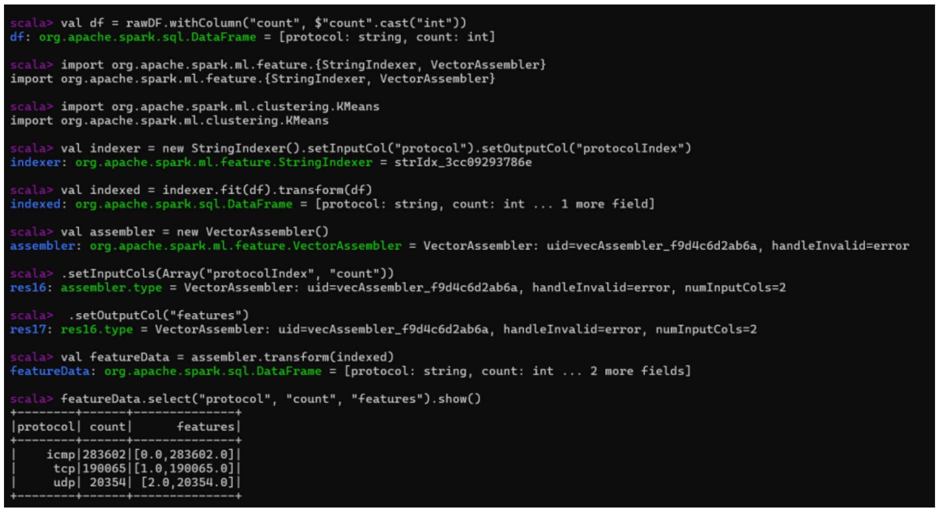
## Feature Engineering for MLlib

We performed the feature engineering for MLlib by loading the preprocessed KDD dataset from HDFS into Apache Spark. The tab-delimited output stored in HDFS was read into a Spark DataFrame with two columns that are protocol and count.



```
scala> // Import Spark SQL

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> import spark.implicits._
import spark.implicits._

scala> val rawDF = spark.read.option("delimiter", "\t").csv("hdfs:///user/kdd/output/*").toDF("protocol", "count")
rawDF: org.apache.spark.sql.DataFrame = [protocol: string, count: string]

scala> rawDF.show()
+--------+------+
|protocol| count|
+--------+------+
|    icmp|283602|
|     tcp|190065|
|     udp| 20354|
+--------+------+
```

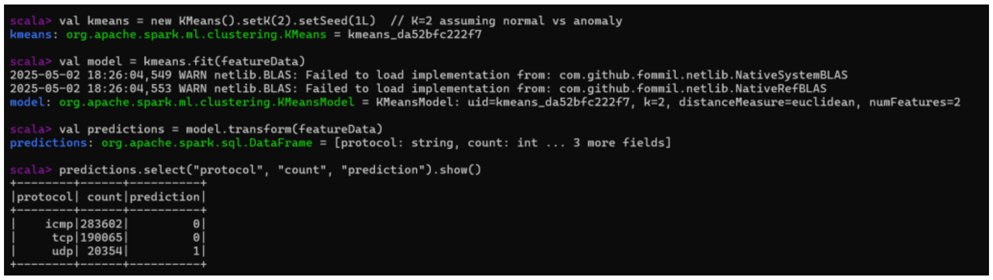
We encoded the protocol column which was originally a string into numeric format using technique of StringIndexer as MLlib algorithm of KMeans require numerical inputs. The count column was cast to integer type to enable numerical processing. The indexed protocol and count features were assembled into a single feature vector using the VectorAssembler. This created a format suitable for input into MLlib model of KMeans clustering. So, this structured and transformed dataset was ready for training and evaluation.



```
scala> val df = rawDF.withColumn("count", $"count".cast("int"))
df: org.apache.spark.sql.DataFrame = [protocol: string, count: int]

scala> import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}
import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

scala> import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.clustering.KMeans

scala> val indexer = new StringIndexer().setInputCol("protocol").setOutputCol("protocolIndex")
indexer: org.apache.spark.ml.feature.StringIndexer = strIdx_3cc09293786e

scala> val indexed = indexer.fit(df).transform(df)
indexed: org.apache.spark.sql.DataFrame = [protocol: string, count: int ... 1 more field]

scala> val assembler = new VectorAssembler()
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_f9d4c6d2ab6a, handleInvalid=error

scala> .setInputCols(Array("protocolIndex", "count"))
res16: assembler.type = VectorAssembler: uid=vecAssembler_f9d4c6d2ab6a, handleInvalid=error, numInputCols=2

scala>  .setOutputCol("features")
res17: res16.type = VectorAssembler: uid=vecAssembler_f9d4c6d2ab6a, handleInvalid=error, numInputCols=2

scala> val featureData = assembler.transform(indexed)
featureData: org.apache.spark.sql.DataFrame = [protocol: string, count: int ... 2 more fields]

scala> featureData.select("protocol", "count", "features").show()
+--------+------+--------------+
|protocol| count|      features|
+--------+------+--------------+
|    icmp|283602|[0.0,283602.0]|
|     tcp|190065|[1.0,190065.0]|
|     udp| 20354| [2.0,20354.0]|
+--------+------+--------------+
```

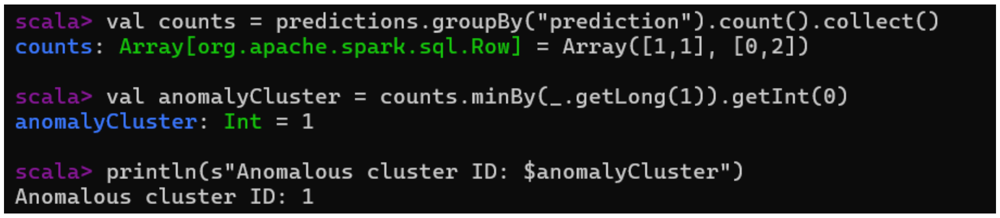
### KMeans Clustering for Anomaly Detection

We used the KMeans clustering algorithm from Spark MLlib to categorize our KDD data. We fed this data into the KMeans algorithm with k=2 as the data is to be clustered into two categories which are normal and anomalous. After training, the model was used to assign each data point a cluster label i.e. prediction. This effectively categorized different types of protocol usage patterns based on frequency that can highlight anomalies. The results were displayed to observe protocol types, their counts and the assigned cluster.



```
scala> val kmeans = new KMeans().setK(2).setSeed(1L)  // K=2 assuming normal vs anomaly
kmeans: org.apache.spark.ml.clustering.KMeans = kmeans_da52bfc222f7

scala> val model = kmeans.fit(featureData)
2025-05-02 18:26:04,549 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
2025-05-02 18:26:04,553 WARN netlib.BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
model: org.apache.spark.ml.clustering.KMeansModel = KMeansModel: uid=kmeans_da52bfc222f7, k=2, distanceMeasure=euclidean, numFeatures=2

scala> val predictions = model.transform(featureData)
predictions: org.apache.spark.sql.DataFrame = [protocol: string, count: int ... 3 more fields]

scala> predictions.select("protocol", "count", "prediction").show()
+--------+------+----------+
|protocol| count|prediction|
+--------+------+----------+
|    icmp|283602|         0|
|     tcp|190065|         0|
|     udp| 20354|         1|
+--------+------+----------+
```

We also confirmed the anomaly cluster



```
scala> val counts = predictions.groupBy("prediction").count().collect()
counts: Array[org.apache.spark.sql.Row] = Array([1,1], [0,2])

scala> val anomalyCluster = counts.minBy(_.getLong(1)).getInt(0)
anomalyCluster: Int = 1

scala> println(s"Anomalous cluster ID: $anomalyCluster")
Anomalous cluster ID: 1
```

## RESULTS AND DISCUSSION

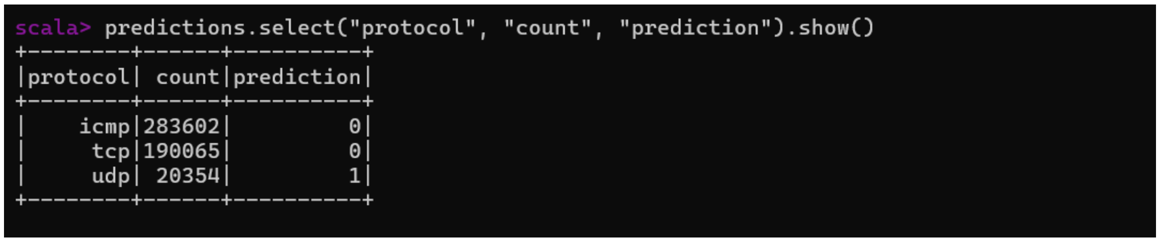
### Prediction and Output Analysis

After executing the MapReduce job on our KDD dataset, the network records were successfully categorized based on protocol types. The output shows:

- The ICMP protocol traffic occurred 283602 times.

- The TCP protocol traffic had 190065 occurrences. It is used for reliable data transmission online.
- The UDP traffic showed 20354 occurrences. It is known for fast and connectionless communication.

The final predicted output by the model is:



```
scala> predictions.select("protocol", "count", "prediction").show()
+--------+------+----------+
|protocol| count|prediction|
+--------+------+----------+
|    icmp|283602|         0|
|     tcp|190065|         0|
|     udp| 20354|         1|
+--------+------+----------+
```

After the clustering using the KMeans algorithm of Spark MLlib, the final predictions were:

- Cluster 0 to ICMP and TCP traffic
- Cluster 1 to UDP traffic

This suggests that the KMeans model identified UDP traffic as behaviorally distinct from ICMP and TCP. The UDP is often associated with anomalies due to its unverified and faster data transfer mechanism, this cluster 1 likely indicates anomalous or suspicious traffic. On the other hand, ICMP and TCP are in the same cluster 0 so, they are treated as normal traffic patterns by the model.

## COMPARATIVE ANALYSIS

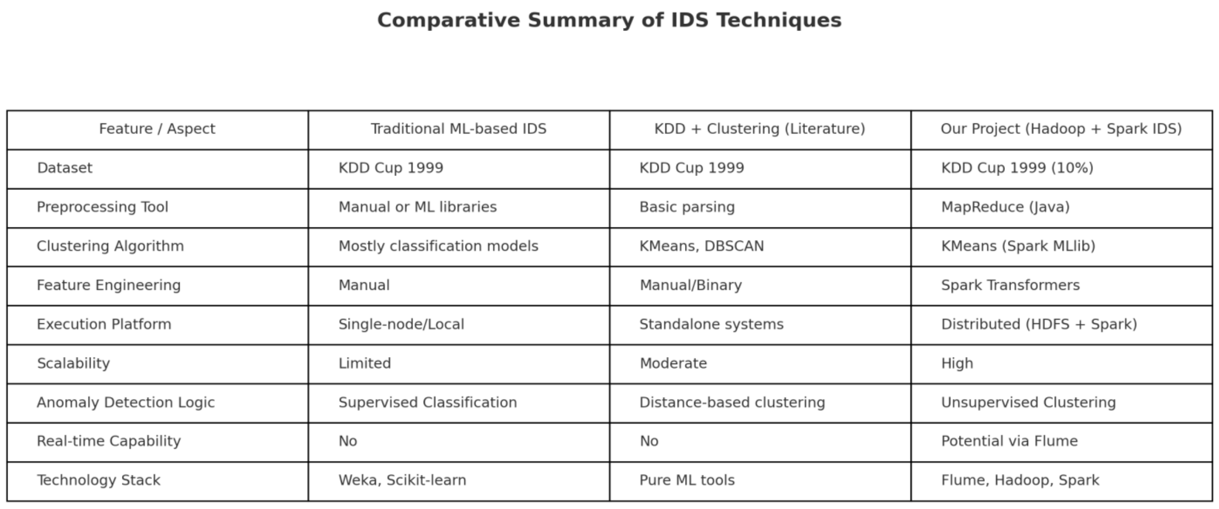
### Comparison with Existing Literature

In comparison to previous studies where traditional machine learning models were applied on intrusion detection using static datasets in local environments, our project integrates big data tools end to end with a more scalable and modular solution. Most of the previous research employed algorithms such as Decision Trees, Naive Bayes, and Support Vector Machines on static datasets within local environments, but our approach streams large-scale logs with Apache Flume,Hadoop MapReduce, and Spark MLlib.

In particular, as opposed to other current KDD-based clustering techniques that either make use of elementary preprocessing or are restricted to smaller-scale computational environments, our architecture does distributed preprocessing with Java-based custom Mapper (KDDPreprocessMapper) and Reducer (KDDPreprocessReducer) classes. This facilitates protocol-level aggregation of large amounts of network logs. The tab-delimited output after processing is automatically consumed by Apache Spark, wherein MLlib functions like StringIndexer, VectorAssembler, and KMeans are leveraged for automated feature engineering and clustering-based anomaly detection.

One of the most important novelties of our implementation is taking advantage of Spark's in-memory processing to train and make inferences on clustered data efficiently. Our experiment indicates that UDP traffic, commonly associated with anomalous behavior because it is connectionless, was successfully separated into a separate cluster — confirming the strength of our clustering pipeline.

This is in opposition to other models, which either require label availability or substantial manual feature engineering. Our system is scalable and flexible, which offers room for future applications in real-time intrusion detection.



**Comparative Summary of IDS Techniques**

| Feature / Aspect | Traditional ML-based IDS | KDD + Clustering (Literature) | Our Project (Hadoop + Spark IDS) |
|---|---|---|---|
| Dataset | KDD Cup 1999 | KDD Cup 1999 | KDD Cup 1999 (10%) |
| Preprocessing Tool | Manual or ML libraries | Basic parsing | MapReduce (Java) |
| Clustering Algorithm | Mostly classification models | KMeans, DBSCAN | KMeans (Spark MLlib) |
| Feature Engineering | Manual | Manual/Binary | Spark Transformers |
| Execution Platform | Single-node/Local | Standalone systems | Distributed (HDFS + Spark) |
| Scalability | Limited | Moderate | High |
| Anomaly Detection Logic | Supervised Classification | Distance-based clustering | Unsupervised Clustering |
| Real-time Capability | No | No | Potential via Flume |
| Technology Stack | Weka, Scikit-learn | Pure ML tools | Flume, Hadoop, Spark |

## Comparison Summary

The diagram above is a comparison of our Hadoop-Spark-intrusion detection system with traditional machine learning-based, as well as earlier KDD-based clustering work. The classical IDS models generally use supervised models like Decision Trees or SVMs, and are generally limited to local domains with no scalability or real-time operation. Studies that use clustering algorithms like KMeans or DBSCAN provide advancements in anomaly detection but lack behind on distributed processing and automation.

Conversely, our project demonstrates a scalable, modular pipeline using Apache Flume for real-time log ingestion, Hadoop MapReduce for preprocessing, and Apache Spark MLlib for unsupervised clustering. As observed from the table, this end-to-end system improves feature engineering, anomaly detection, and large-scale data handling—demonstrating its relevance to current network security needs.

## CONCLUSION

So, in this project, we successfully implemented a scalable intrusion detection system using big data tools which are Apache Flume, Hadoop and Apache Spark. The KDD dataset was ingested using Flume and stored in HDFS where preprocessing was done through a MapReduce job to categorize network traffic based on protocol types. We used the KMeans clustering algorithm of Spark MLlib for feature engineering and anomaly detection. Our output results showed that UDP traffic was flagged as anomalous while ICMP and TCP were stated as normal traffic. This approach depicts the effectiveness of combining big data technologies with ML for efficient making of IDS in large-scale network environments.

## APPENDIX

### KDD Log Processor Code

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class KDDLogProcessor {

    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "KDD Log Processor");

        job.setJarByClass(KDDLogProcessor.class);

        job.setMapperClass(KDDPreprocessMapper.class);

        job.setReducerClass(KDDPreprocessReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);


        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));


        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

## KDD Mapper Code

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class KDDPreprocessMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text featureKey = new Text();


    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String[] fields = value.toString().split(",");
        if (fields.length > 1) {
            String protocol = fields[1]; // Example: protocol type field in KDD dataset
            featureKey.set(protocol);
            context.write(featureKey, one);
        }
    }
}
```

### KDD Reducer Code

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class KDDPreprocessReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

## Scala Code

```
// Import Spark SQL

import org.apache.spark.sql.SparkSession

import spark.implicits._

// Read the tab-delimited HDFS output

val rawDF = spark.read.option("delimiter",
"\t").csv("hdfs:///user/kdd/output/*").toDF("protocol", "count")

// Show the loaded data

rawDF.show()

// Convert count to integer

val df = rawDF.withColumn("count", $"count".cast("int"))

// Import MLlib tools

import org.apache.spark.ml.feature.{StringIndexer, VectorAssembler}

import org.apache.spark.ml.clustering.KMeans

// Index the protocol string to numeric

val indexer = new StringIndexer().setInputCol("protocol").setOutputCol("protocolIndex")

val indexed = indexer.fit(df).transform(df)

// Assemble features: [protocolIndex, count]

val assembler = new VectorAssembler()

  .setInputCols(Array("protocolIndex", "count"))

  .setOutputCol("features")

val featureData = assembler.transform(indexed)

featureData.select("protocol", "count", "features").show()

val kmeans = new KMeans().setK(2).setSeed(1L)  // K=2 assuming normal vs anomaly

val model = kmeans.fit(featureData)

// Make predictions

val predictions = model.transform(featureData)

predictions.select("protocol", "count", "prediction").show()
predictions.groupBy("prediction").count().show()
```

## REFERENCES

- Real-time Network Intrusion Detection Using Hadoop-Based Bayesian Classifier: https://www.sciencedirect.com/science/article/abs/pii/B9780124114746000189
- Hadoop-based Intrusion Detection Technology and Data Visualization for Website Security: https://www.atlantis-
- press.com/proceedings/cnct16/25870798#:~:text=The%20core%20components%20of%20hardware,transferring%20server%20and%20Hadoop%20cluster.