BDAT 1008
Data Collection and Curation

Assignment 3

Assignment 3 – Spark (Threat Intelligence)

Report

Document Summary

Document Item	Status
Document Title	Report on Threat Intelligence Project using Spark
Date Last Modified	14-July-2023
Status	Final
Document Description	This document provides process of Real-world Data Pipeline Project.

Prepared By

Name	SID
Ruchika Gupta	200559617

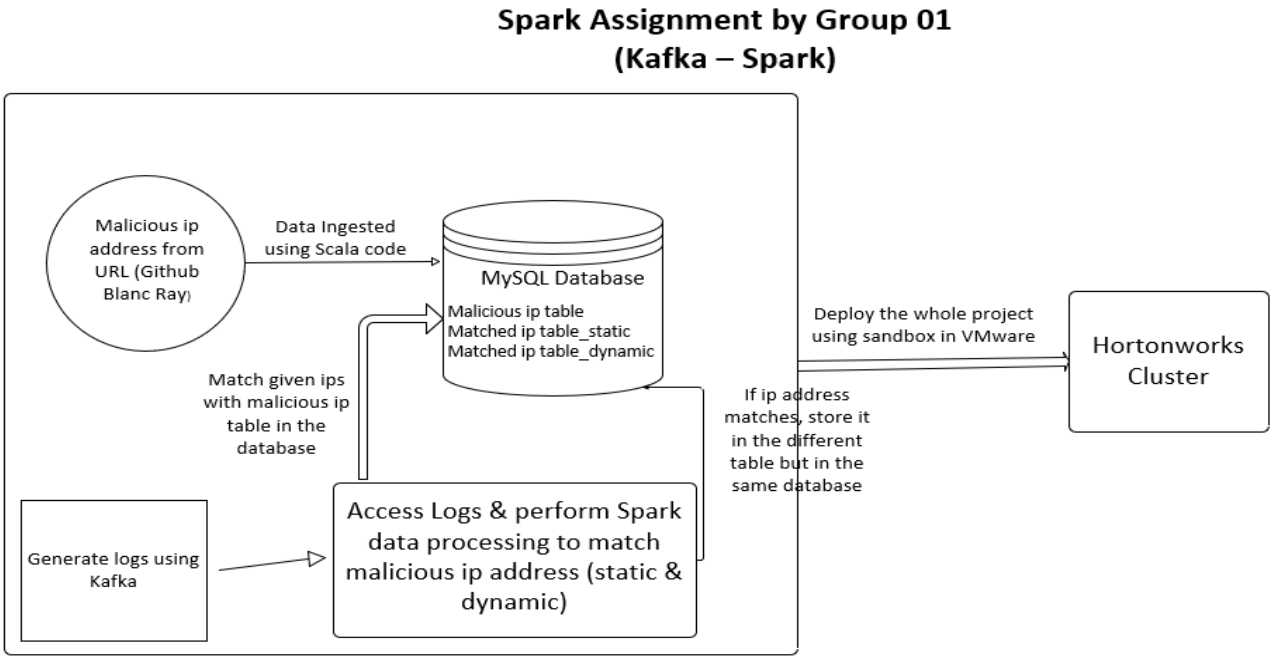
Table of Contents

Part A: 1st Approach (KafkaLogs - SparkStreaming)

Part B: 2nd Approach (Programmatically with SparkSession)

Part A: 1st Approach (KafkaLogs - SparkStreaming)

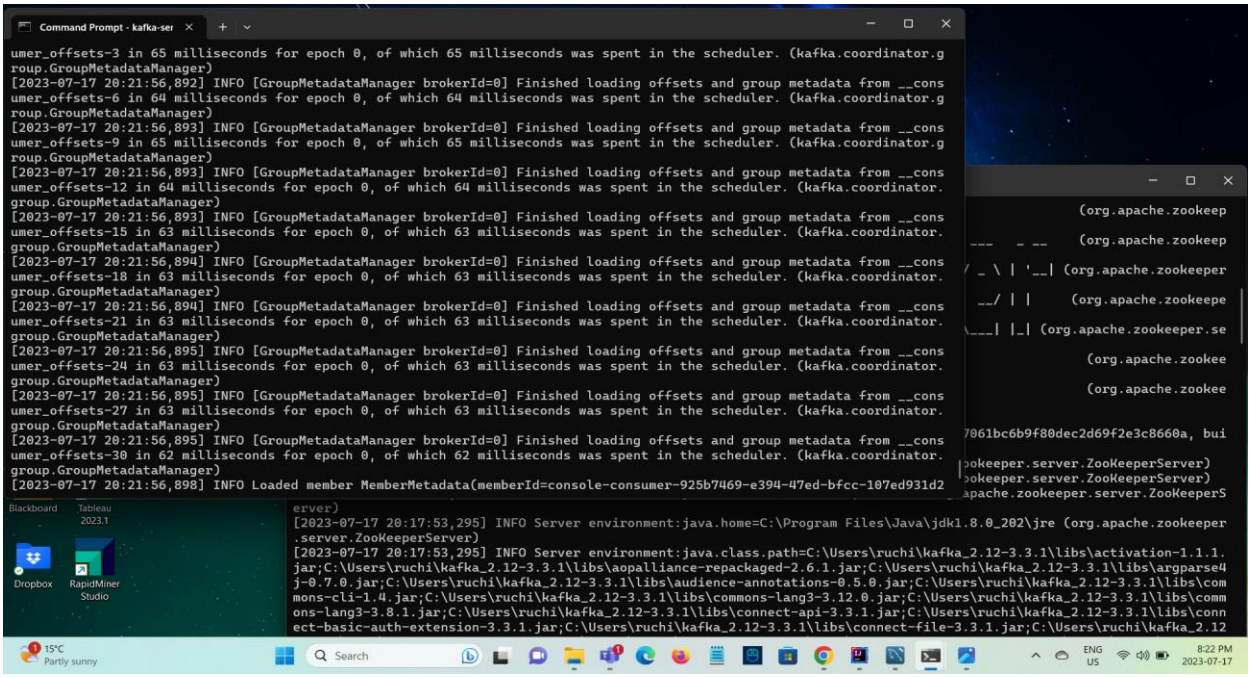
1. Layout of the project

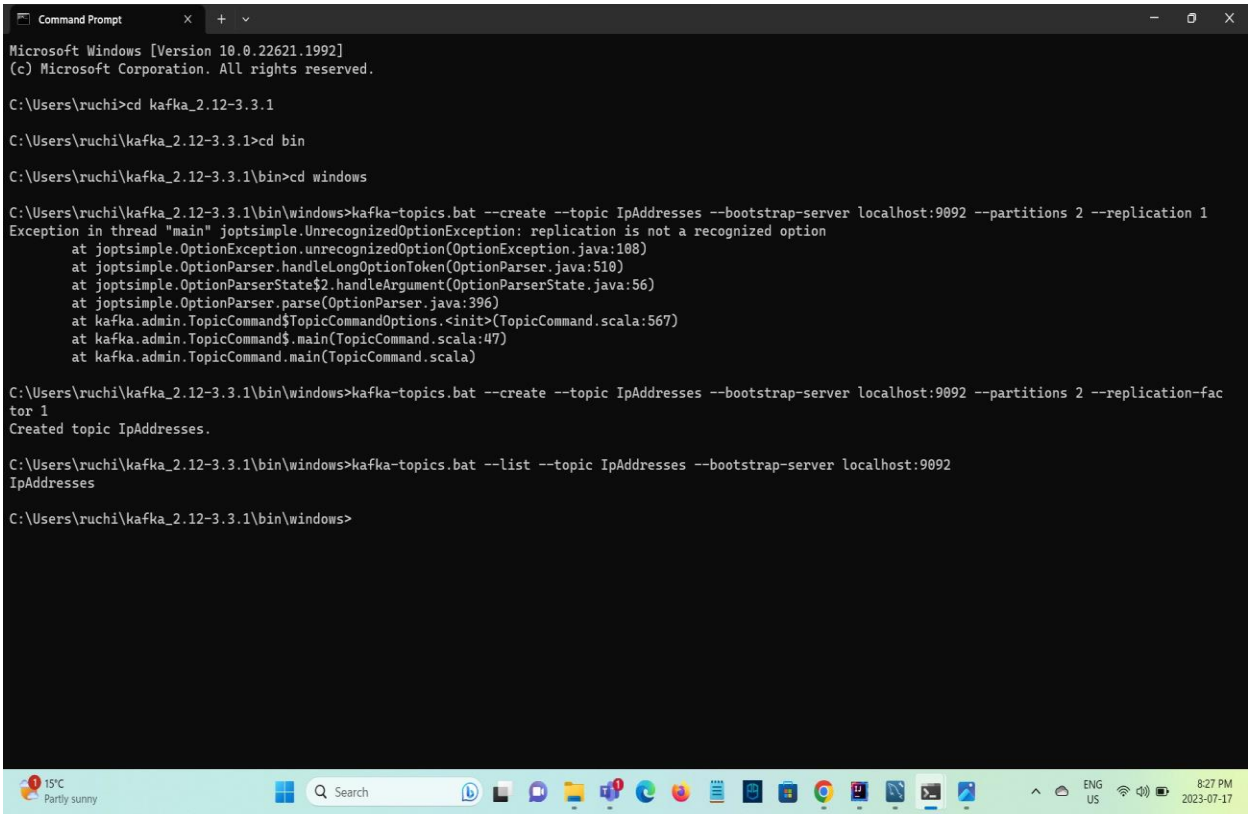


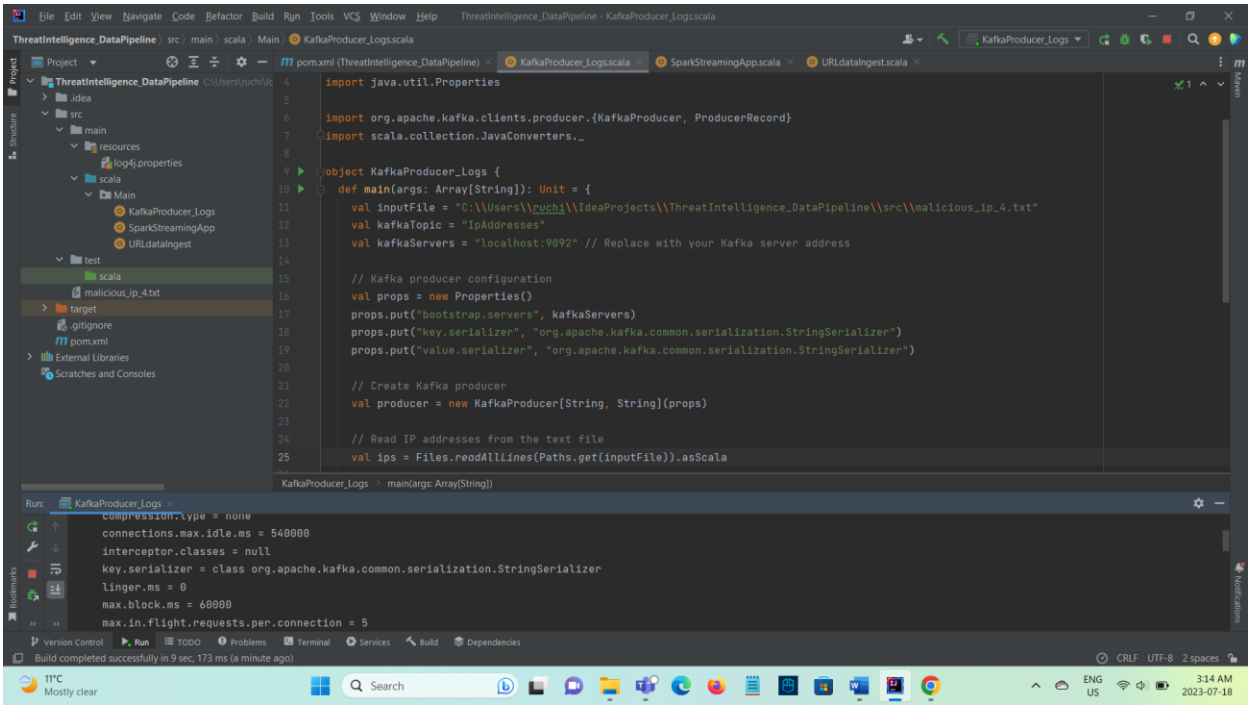
We ingested malicious IP addresses from GitHub Link to MySQL database using Scala code. Then we generated logs using Kafka producer app in the topic "IpAddresses" using Scala code. Then we developed SparkStreaming App to access those logs and further processed them to match from data already stored in the database and store the matched Ips in the same database but in different table using Spark. At last we will deploy the whole application to the Hortonworks cluster.

2. Generated Logs by KafkaProducerApp

We have started our zookeeper & Kafka servers. Then we created a topic "IpAddresses" in the Kafka broker. We successfully generated logs through KafkaProducerApp in the topic using Scala code.

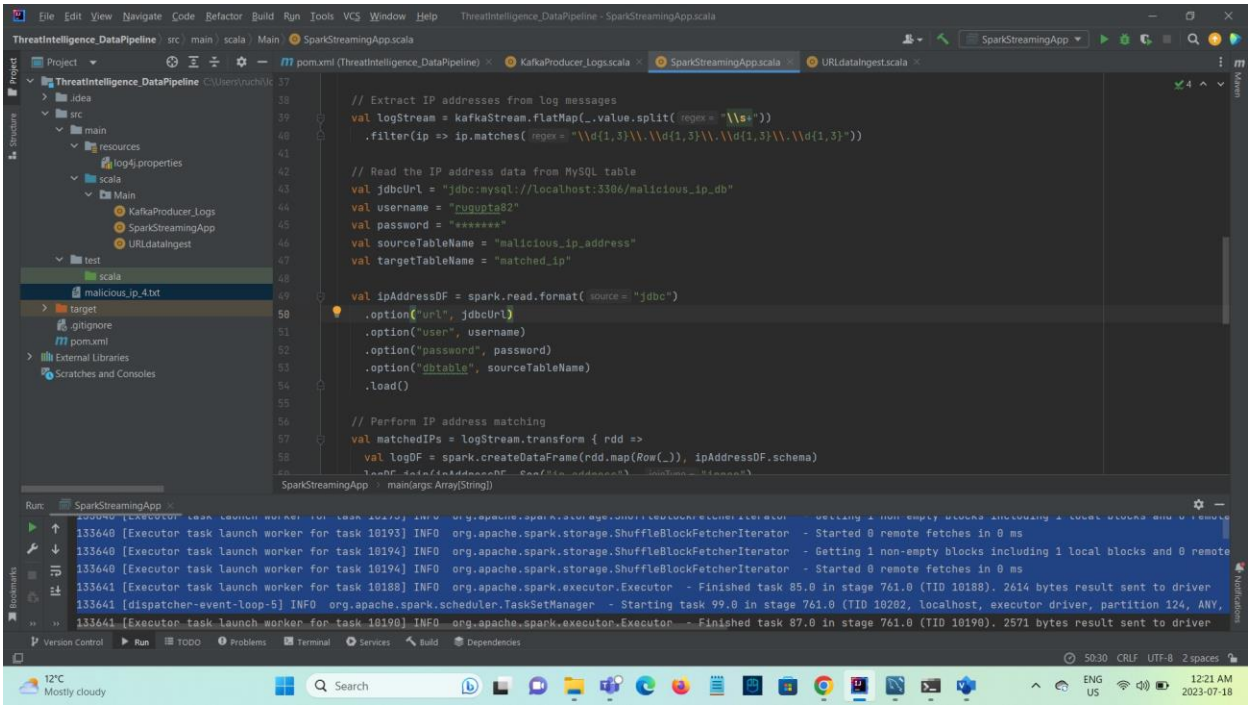

Command Prompt - kafka-ser
umer_offsets-3 in 65 milliseconds for epoch 0, of which 65 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,892] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-6 in 64 milliseconds for epoch 0, of which 64 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,893] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-9 in 65 milliseconds for epoch 0, of which 65 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,893] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-12 in 64 milliseconds for epoch 0, of which 64 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,893] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-15 in 63 milliseconds for epoch 0, of which 63 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,894] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-18 in 63 milliseconds for epoch 0, of which 63 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,894] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-21 in 63 milliseconds for epoch 0, of which 63 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,895] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-24 in 63 milliseconds for epoch 0, of which 63 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,895] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-27 in 63 milliseconds for epoch 0, of which 63 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,895] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __consumer_offsets-30 in 62 milliseconds for epoch 0, of which 62 milliseconds was spent in the scheduler. (kafka.coordinator.group.GroupMetadataManager)
[2023-07-17 20:21:56,898] INFO Loaded member MemberMetadata(memberId=console-consumer-925b7469-e394-47ed-bfcc-107ed931d2
Blackboard
Tableau 2023.1
Dropbox
RapidMiner Studio
[2023-07-17 20:17:53,295] INFO Server environment:java.home=C:\Program Files\Java\jdk1.8.0_202\jre (org.apache.zookeeper.server.ZooKeeperServer)
[2023-07-17 20:17:53,295] INFO Server environment:java.class.path=C:\Users\ruchi\kafka_2.12-3.3.1\libs\activation-1.1.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\aopalliance-repackaged-2.6.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\argparse4j-0.7.0.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\audience-annotations-0.5.0.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\commons-cli-1.4.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\commons-lang3-3.12.0.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\commons-lang3-3.8.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\connect-api-3.3.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\connect-basic-auth-extension-3.3.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\connect-file-3.3.1.jar;C:\Users\ruchi\kafka_2.12
15°C Partly sunny
Search
ENG US
8:22 PM 2023-07-17


Command Prompt
Microsoft Windows [Version 10.0.22621.1992]
(c) Microsoft Corporation. All rights reserved.
C:\Users\ruchi>cd kafka_2.12-3.3.1
C:\Users\ruchi\kafka_2.12-3.3.1>cd bin
C:\Users\ruchi\kafka_2.12-3.3.1\bin>cd windows
C:\Users\ruchi\kafka_2.12-3.3.1\bin\windows>kafka-topics.bat --create --topic IpAddresses --bootstrap-server localhost:9092 --partitions 2 --replication 1
Exception in thread "main" joptsimple.UnrecognizedOptionException: replication is not a recognized option
at joptsimple.OptionException.unrecognizedOption(OptionException.java:108)
at joptsimple.OptionParser.handleLongOptionToken(OptionParser.java:510)
at joptsimple.OptionParserState$2.handleArgument(OptionParserState.java:56)
at joptsimple.OptionParser.parse(OptionParser.java:396)
at kafka.admin.TopicCommand$TopicCommandOptions.<init>(TopicCommand.scala:567)
at kafka.admin.TopicCommand$.main(TopicCommand.scala:47)
at kafka.admin.TopicCommand.main(TopicCommand.scala)
C:\Users\ruchi\kafka_2.12-3.3.1\bin\windows>kafka-topics.bat --create --topic IpAddresses --bootstrap-server localhost:9092 --partitions 2 --replication-factor 1
Created topic IpAddresses.
C:\Users\ruchi\kafka_2.12-3.3.1\bin\windows>kafka-topics.bat --list --topic IpAddresses --bootstrap-server localhost:9092
IpAddresses
C:\Users\ruchi\kafka_2.12-3.3.1\bin\windows>
15°C Partly sunny
Search
ENG US
8:27 PM 2023-07-17



3. Access Logs & Process them by SparkStreaming

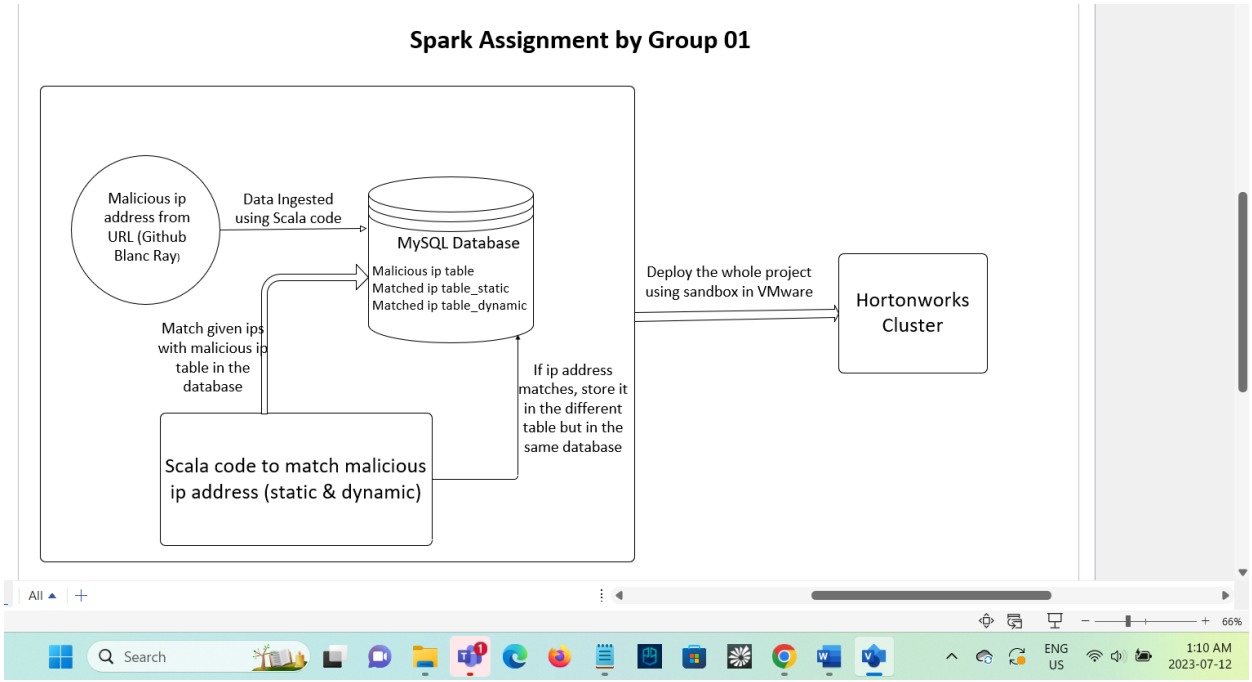
We developed Scala code to access those logs through SparkStreaming App and further processed them to match with Ips stored in the database by making connection with MySQL workbench using JDBC MySQL connector. At last storing those matched Ips in the same database but in a different table.



But due to compatibility issues we were not able to run this code successfully. Although we were able to access those logs but were not able to store them in the database. So, we moved on to another approach in which we implemented the whole project programmatically.

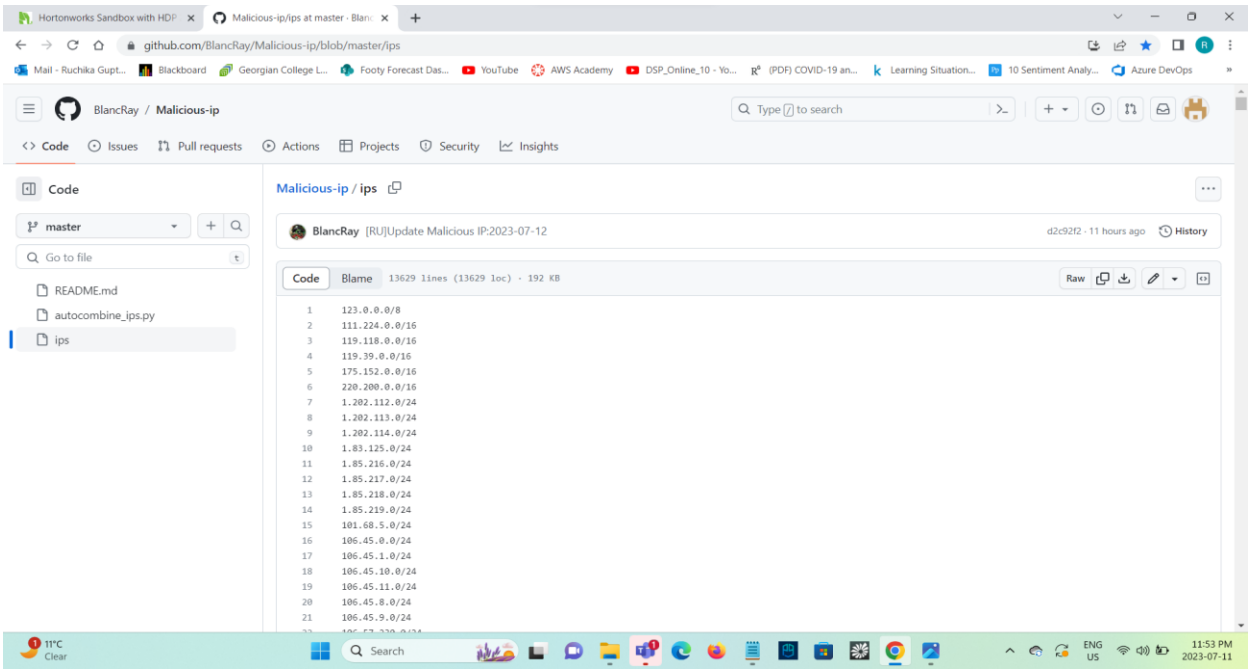
Part B: 2nd Approach (Programmatically with SparkSession)

1. Layout of the project

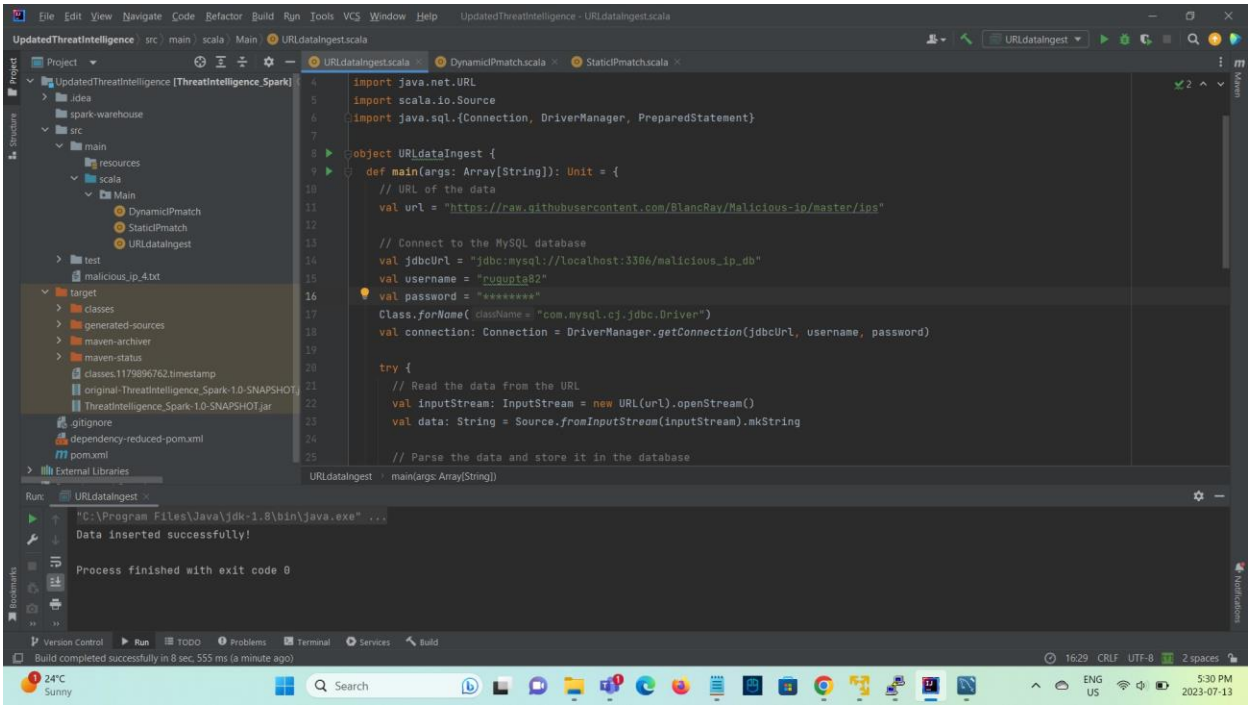


We ingested malicious IP address from a URL to the MYSQL database using Scala code. We developed Scala code to match given IP addresses to the addresses stored in database (which are either static or dynamic). The matched addresses are stored in the same database but in a different table. We deployed it in Hortonworks Cluster using Sandbox.

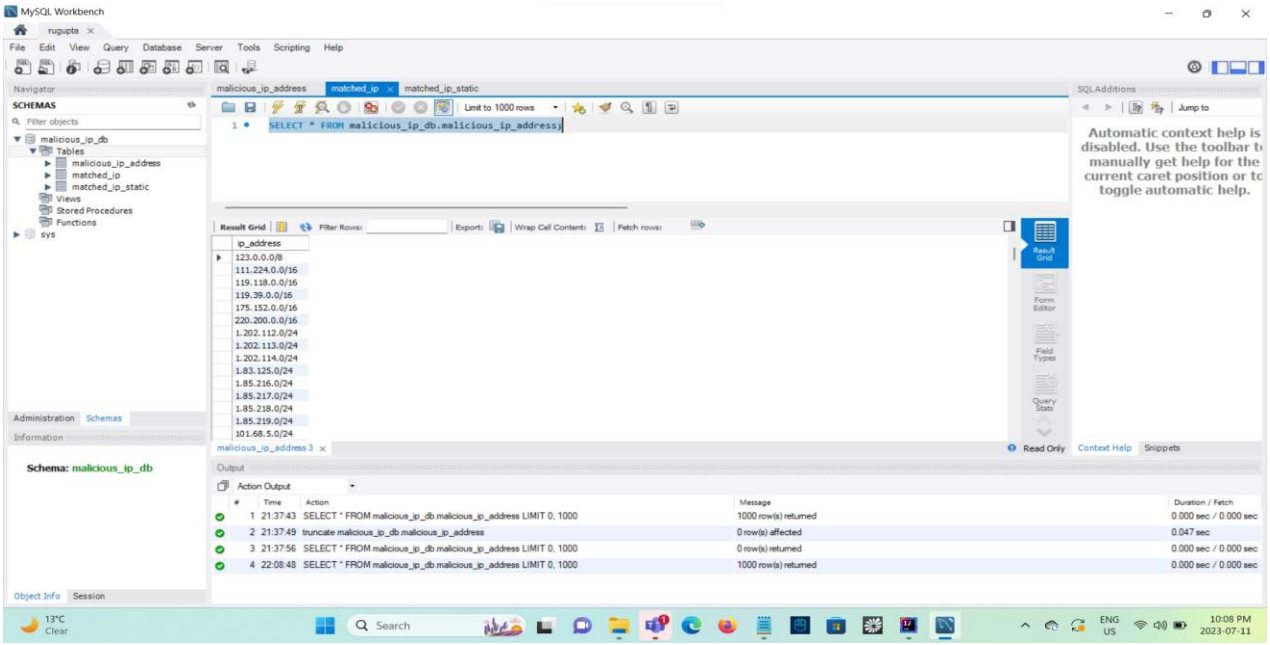
2. Data Ingestion using Scala Code



The above screenshot shows the different IPs generated from the URL.

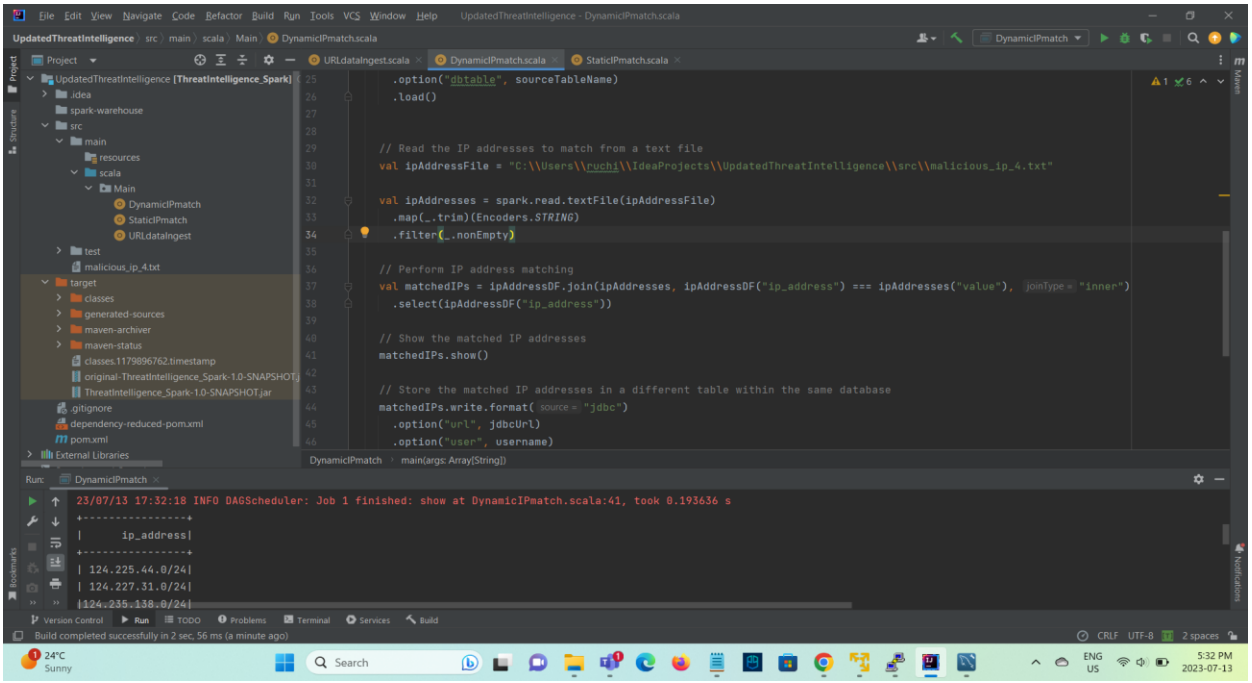


The screenshot above shows how we successfully inserted data. We moved the data in its current form into a Data Frame instance. With package, we provided a namespace to put our code in different files and directories. We indicated the URL and connected to the MySQL database and read data from the URL.

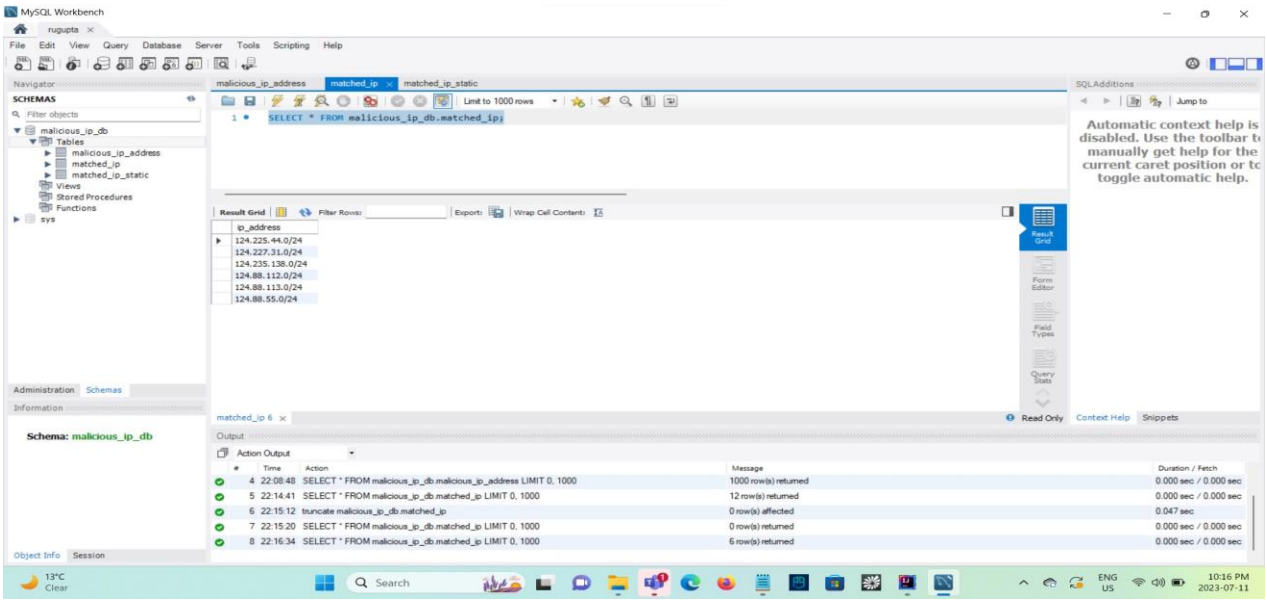


We selected all the fields by using "*****" from the malicious_ip_db. malicious_ip_address table and the result generated is shown on the screenshot above.

3. Dynamically matching and storing ip address using Scala.

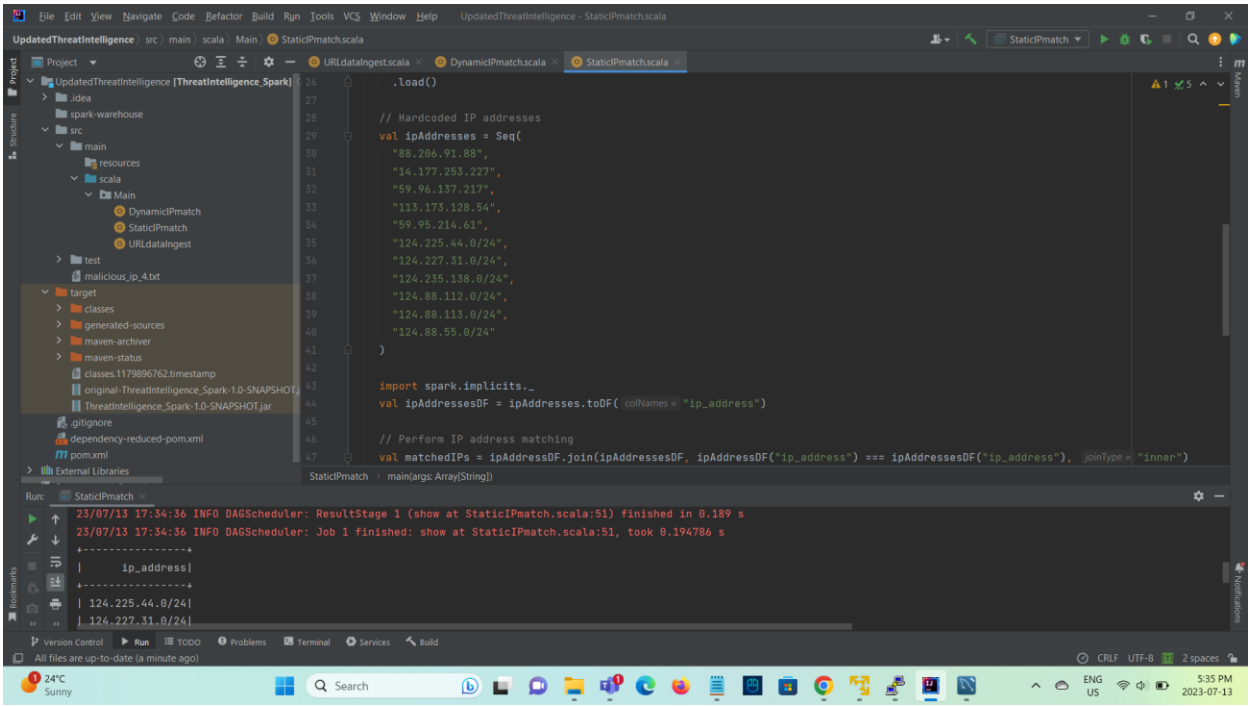


With the screenshot above, we performed IP address matching and showed the matched IP address by using matchedIPs.show(). We stored the matched IP addresses in a different table within the same database. The result of the matched Ips is shown in the above screenshot.

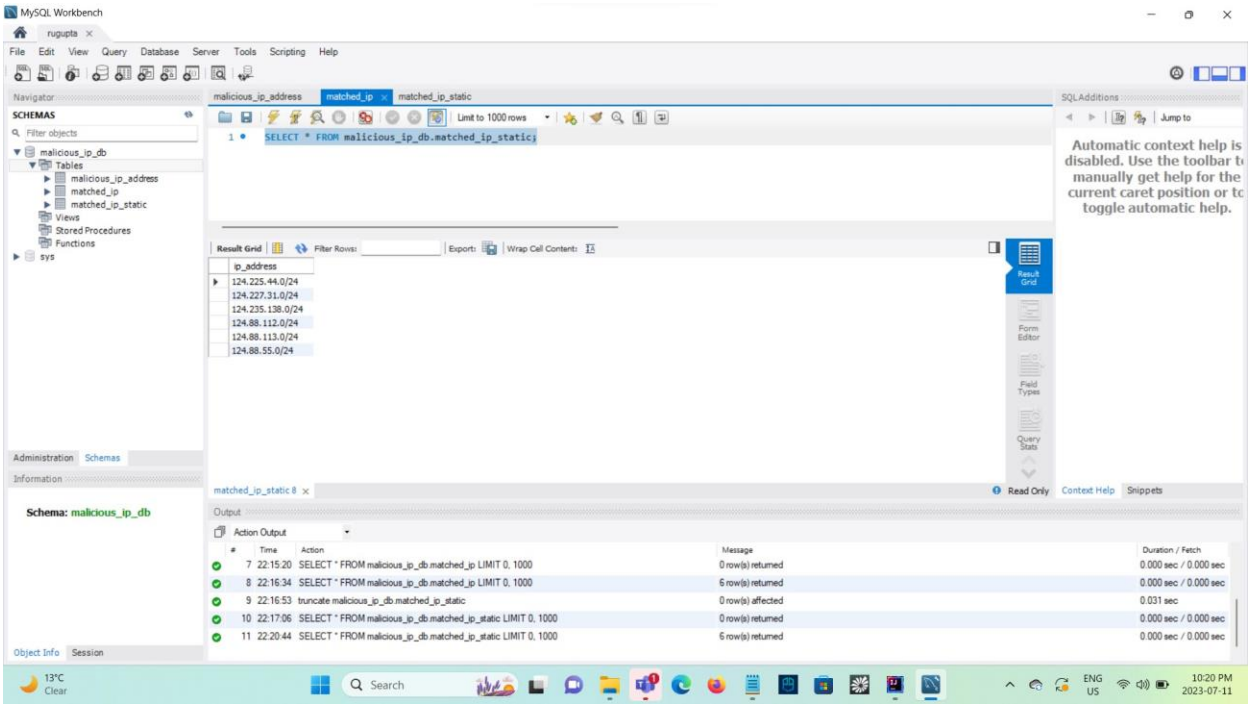


We retrieved all the matched Ips using the select "*" from malicious_ip_db. matched_ip table and the result is shown in the screenshot above. The matched IP addresses are in a different table but same database as malicious_ip_address.

4. Statically matching and storing Ip address using Scala.

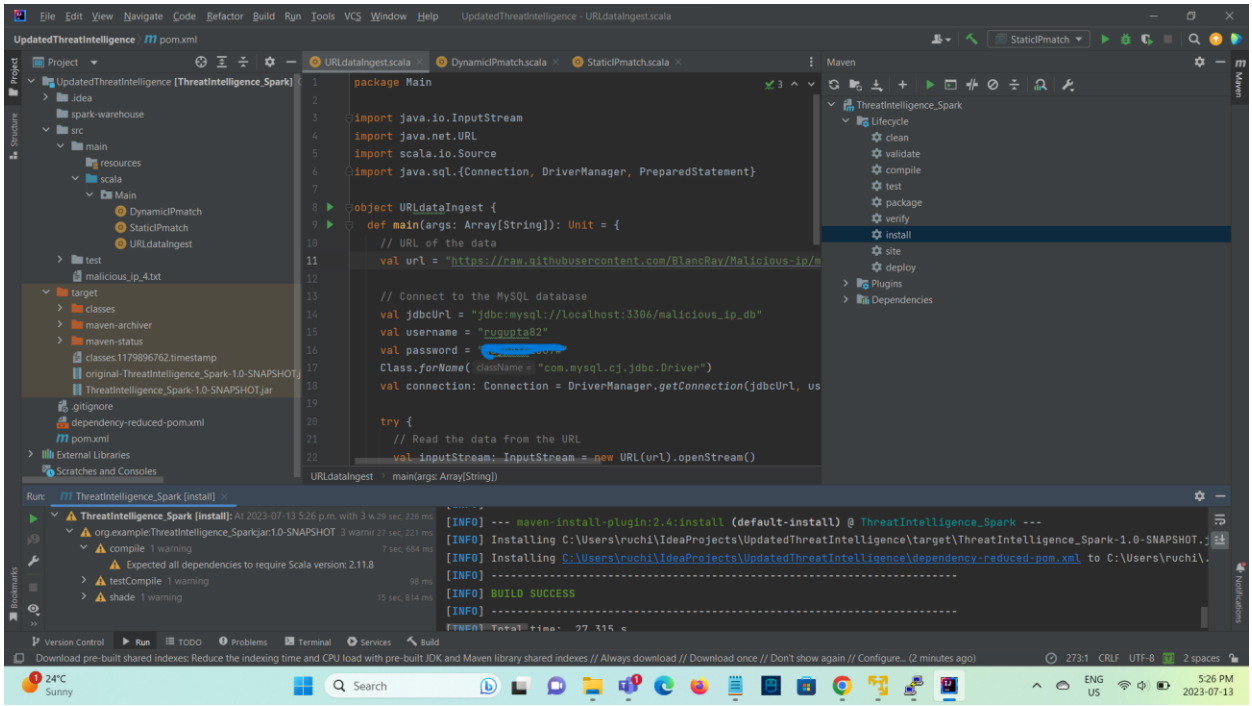


We manually assigned IP addresses that do not change in the settings of the machine. We were able to import the Ips using the import spark.implicits. the hardcoded addresses are stored with the matched Ip addresses and those that matched are stored in a different table called matched_ip_static.
We created a Spark Session with the object name as “static_IpMatcher”, and we set the spark master URL as “local(*)” which indicates that we are not running it in cluster mode. We read IP address data from MYSQL table named as “malicious_ip_address” which is the source table and “matched_ip_static” which is the target table.

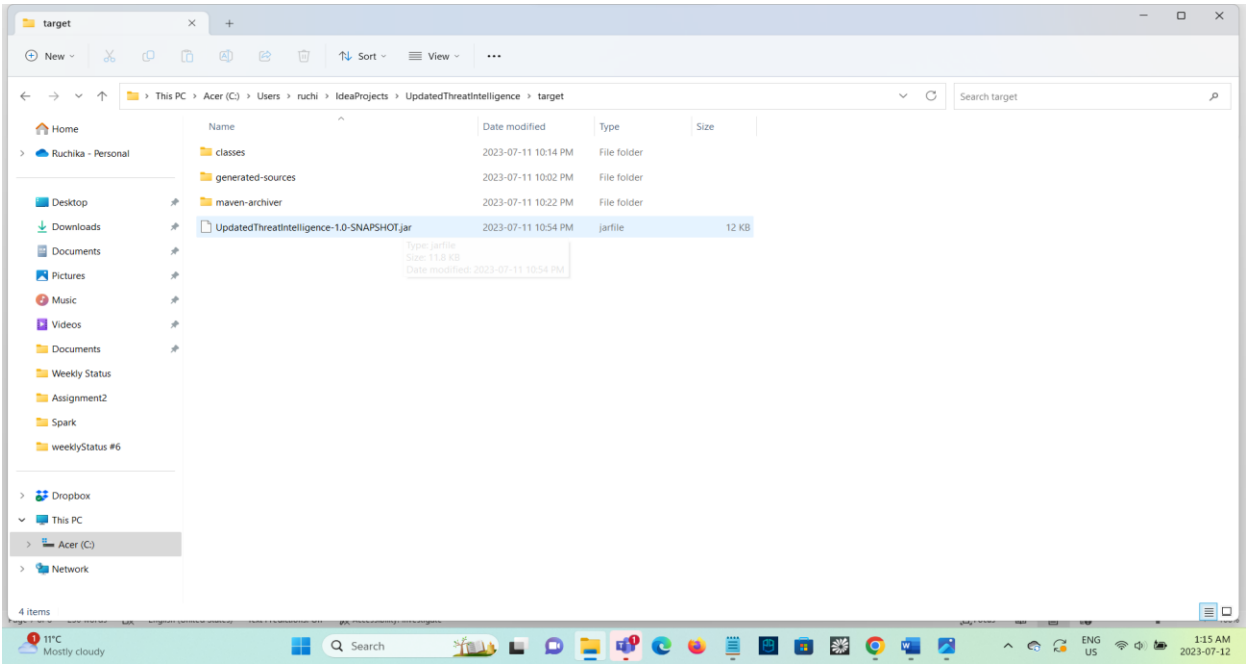


We retrieved all the matched Ips using the select “*” from malicious_ip_db. matched_ip table and the result is shown in the screenshot above. The result shows the IP addresses of the matched_ip_static table.

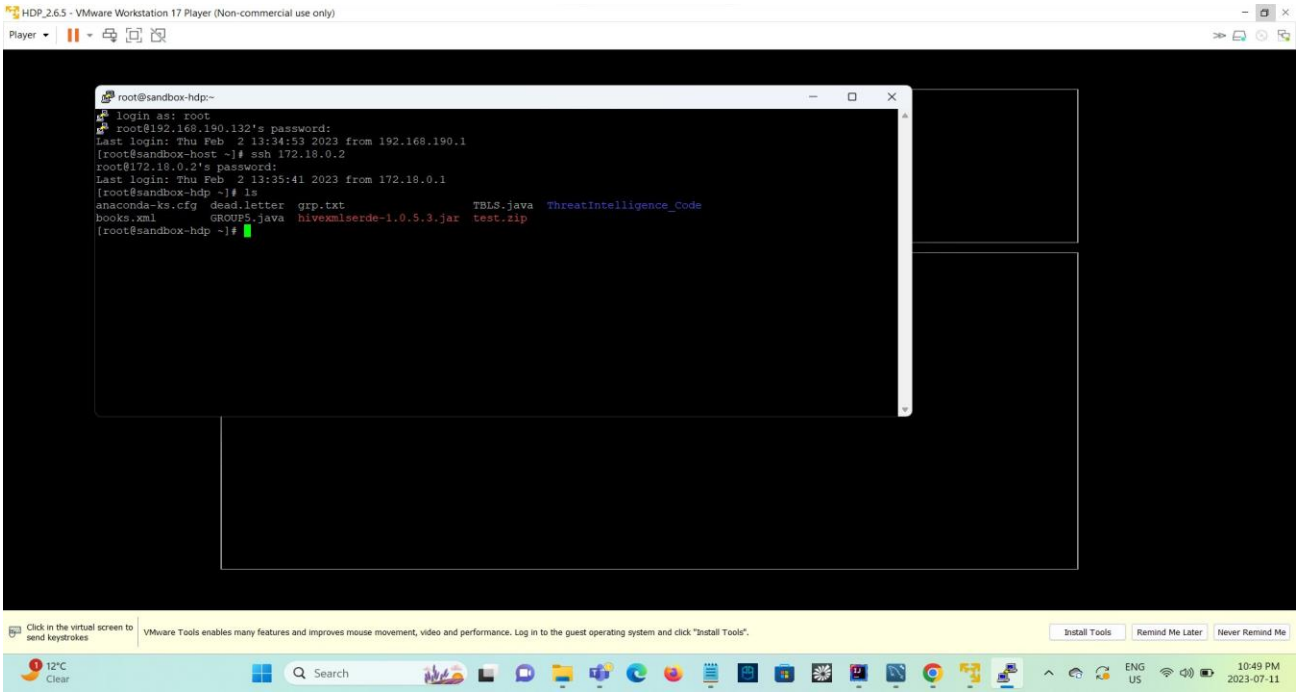
5. Packaged into jar file.



To deploy the project on Hortonworks, we packaged the project by clicking on the install and lifecycle buttons and, the project was moved into a jar file.



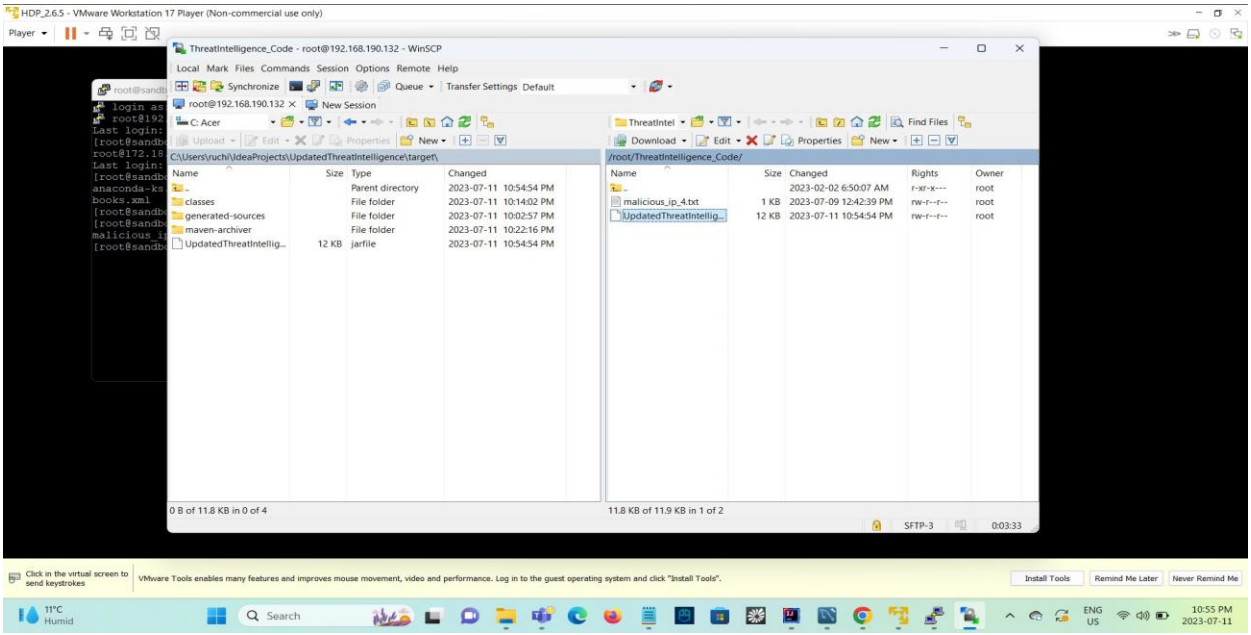
6. Login into Hortonworks sandbox using VMware through PuTTY



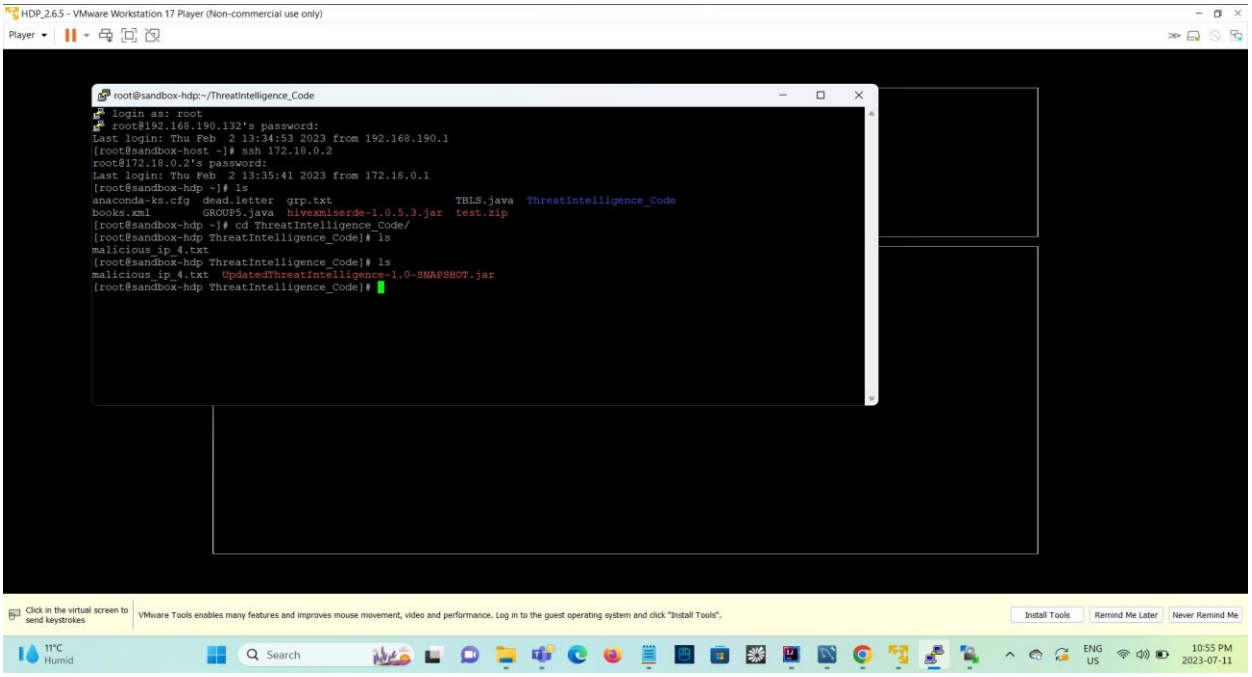
The screenshot shows the process for logging into the Cluster through PuTTY and socket shell(SSH) connection. In the sandbox, we created a directory called ThreatIntelligence_code. We used "ls" command to show all the information about the directories and files in the sandbox-hdp directory.

7. Deploying the whole project on Hortonworks cluster

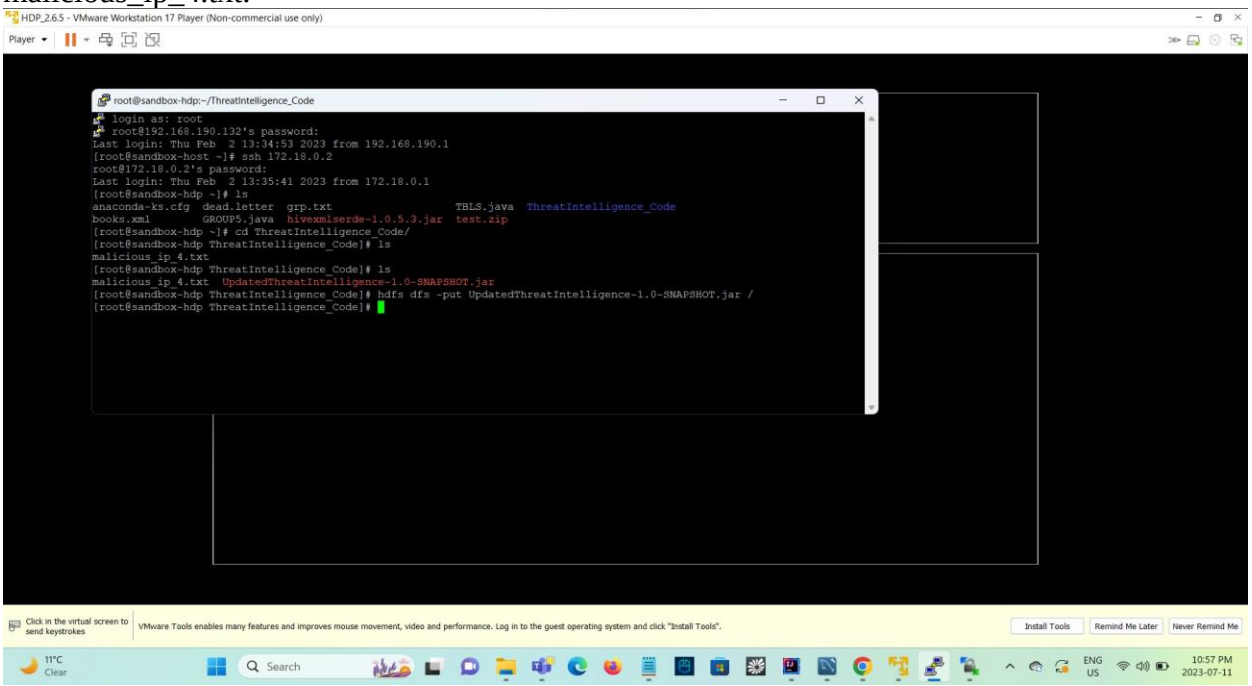
- Transferring text file and packaged jar file from local to cluster



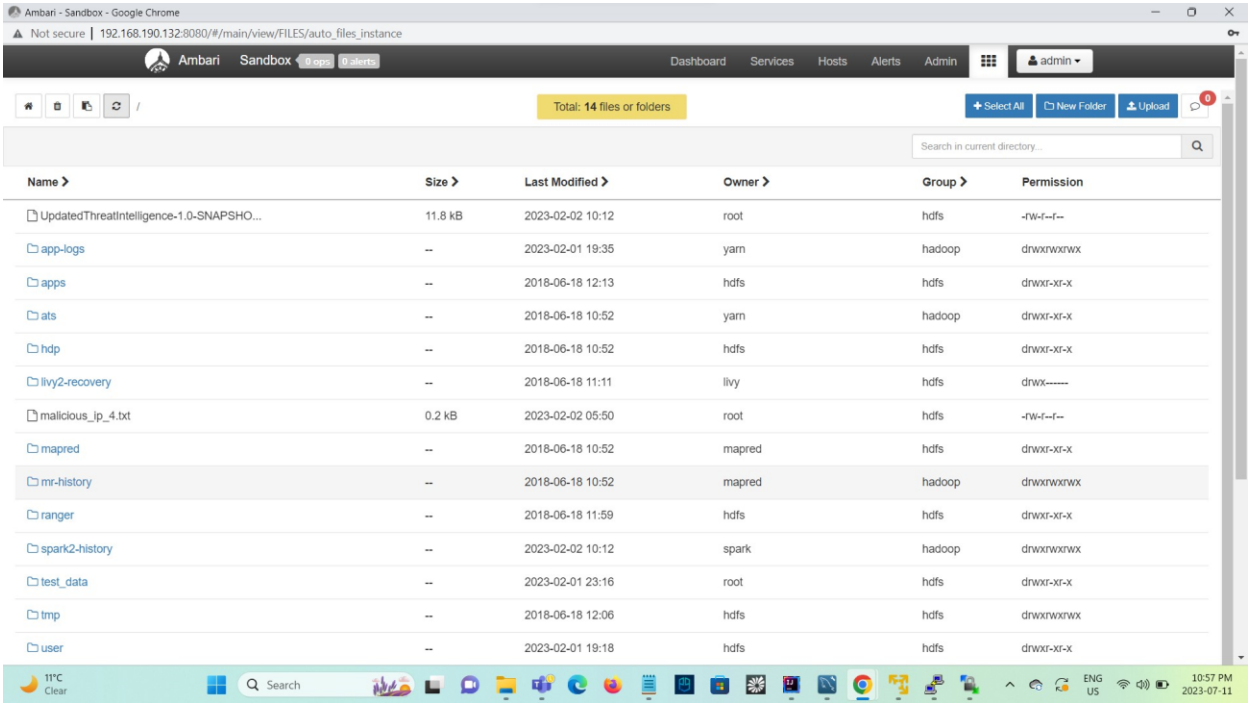
The screenshot above shows how we have copied our malicious_jp_4.txt text file and jar file named “updatedThreatIntelligence_code” from the local computer to the sandbox with the help of WinSCP. We transferred the jar file from the local computer to the sandbox using WinSCP.



The screenshot above shows files transferred to the cluster using “ls” command to list all the directories and files in the working sandbox-hdp directory. We changed the current working directory to ThreatIntelligence_code and listed all the files in it using “ls” command to show the malicious_ip_4.txt.

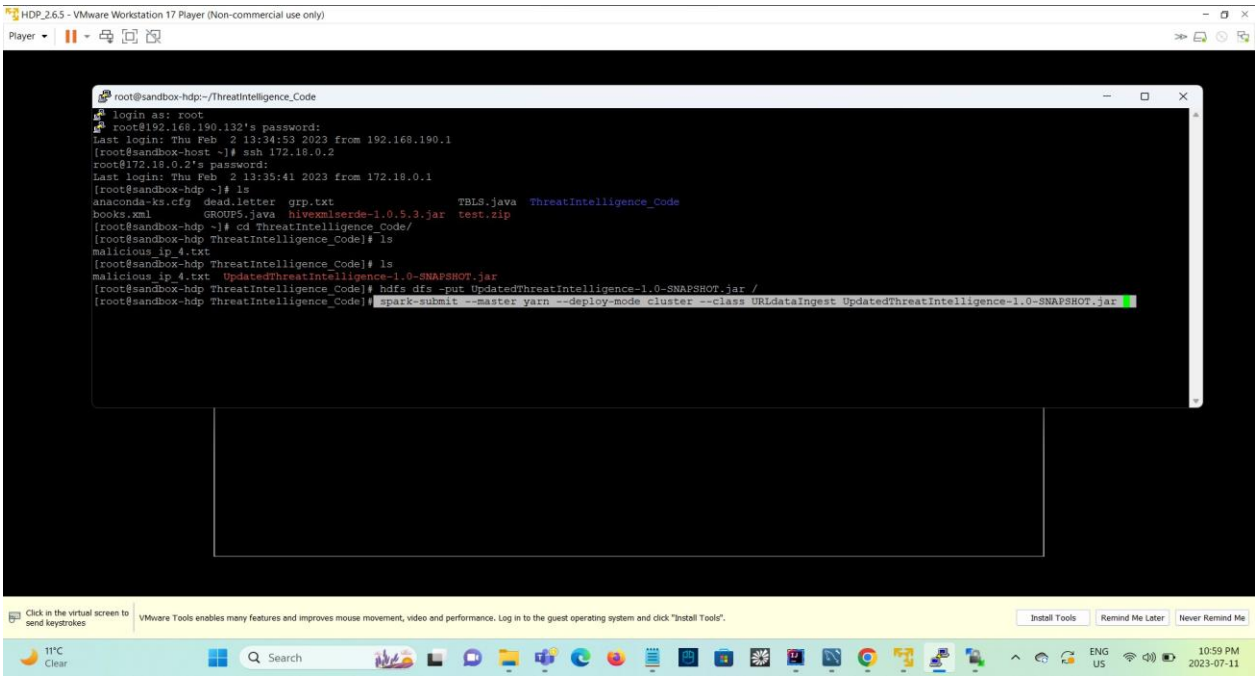


The screenshot above shows using put command to transfer jar file and the malicious Ip addresses text file to the HDFS.

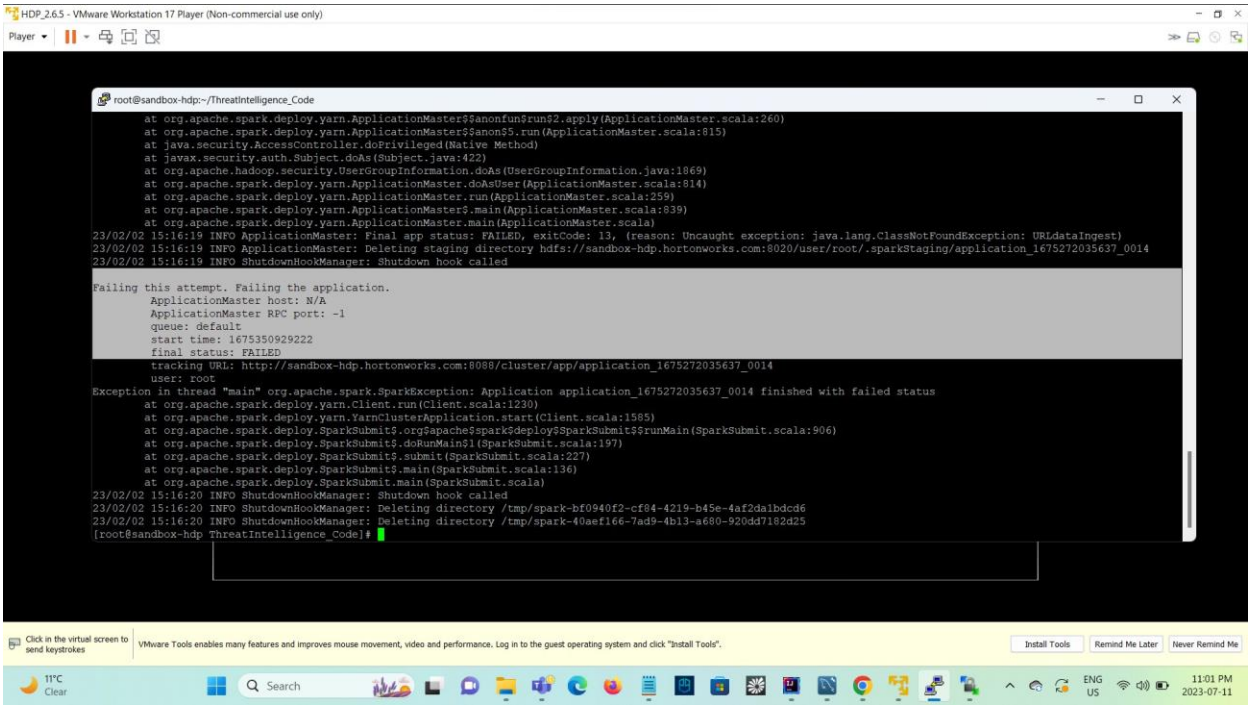


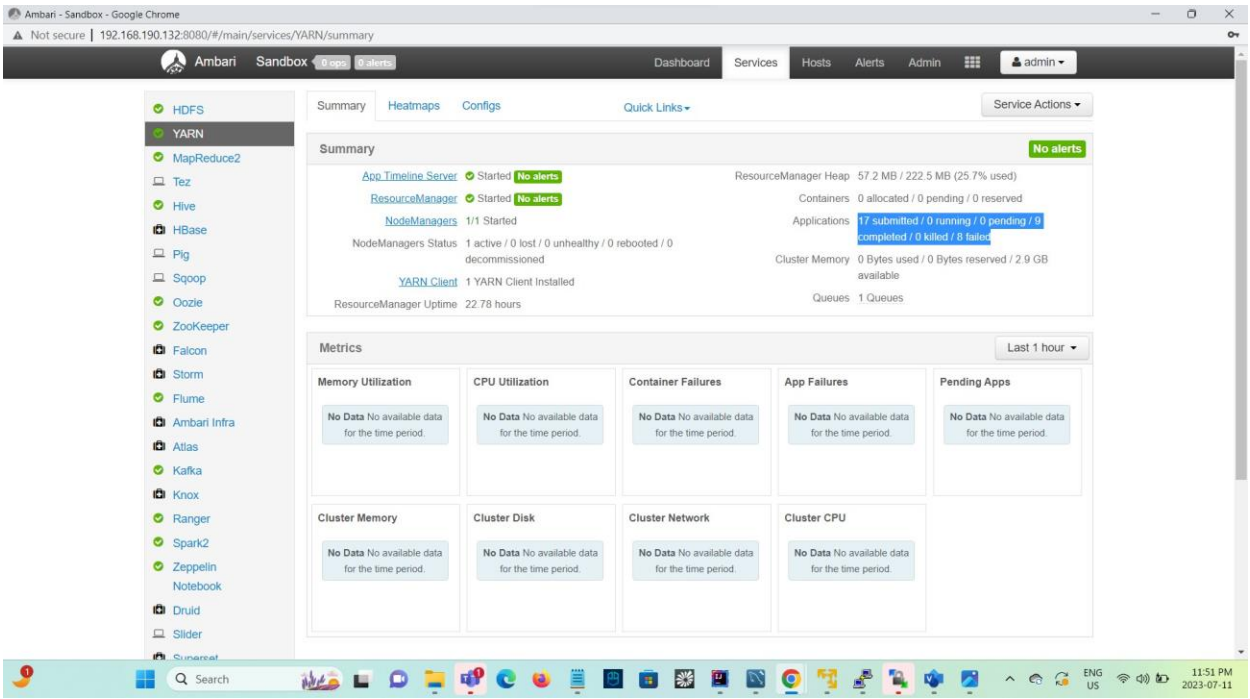
The screenshot above shows the two files i.e., jar file and text file transferred from local computer to the HDFS.

- Running spark job



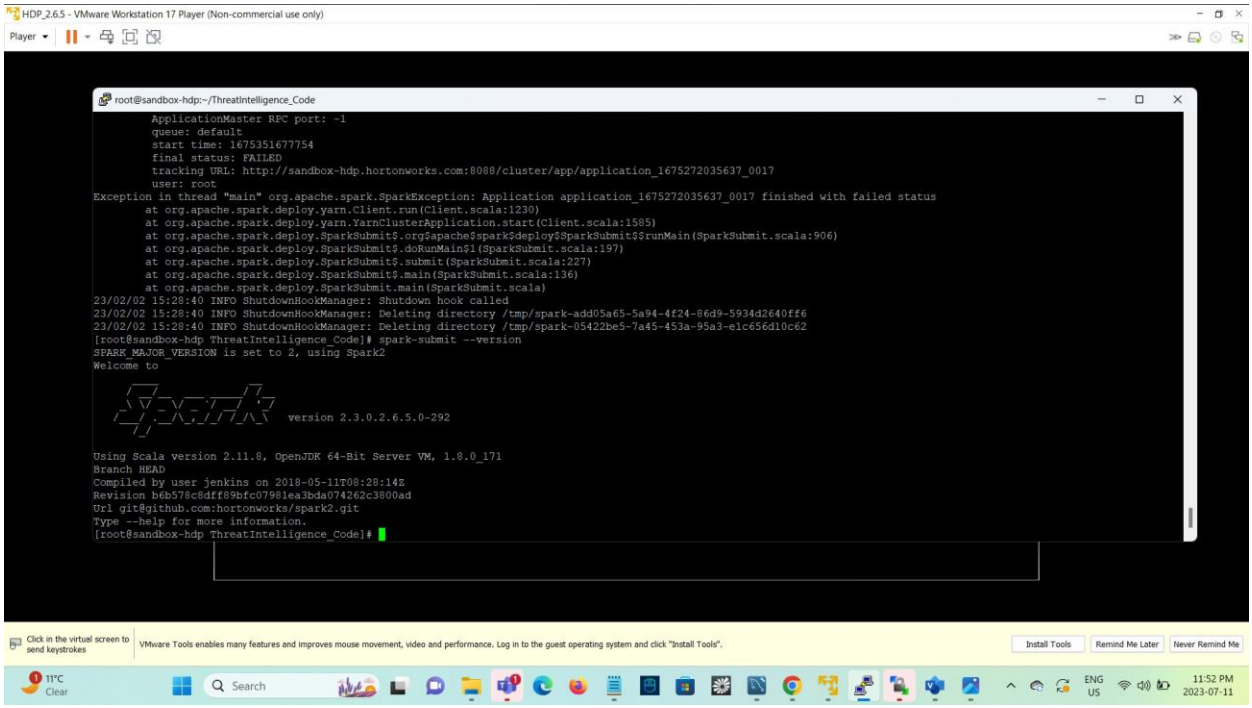
The "spark_submit" command was used to launch an application in the cluster. With the "spark_submit" command, we were able to run the jar file.

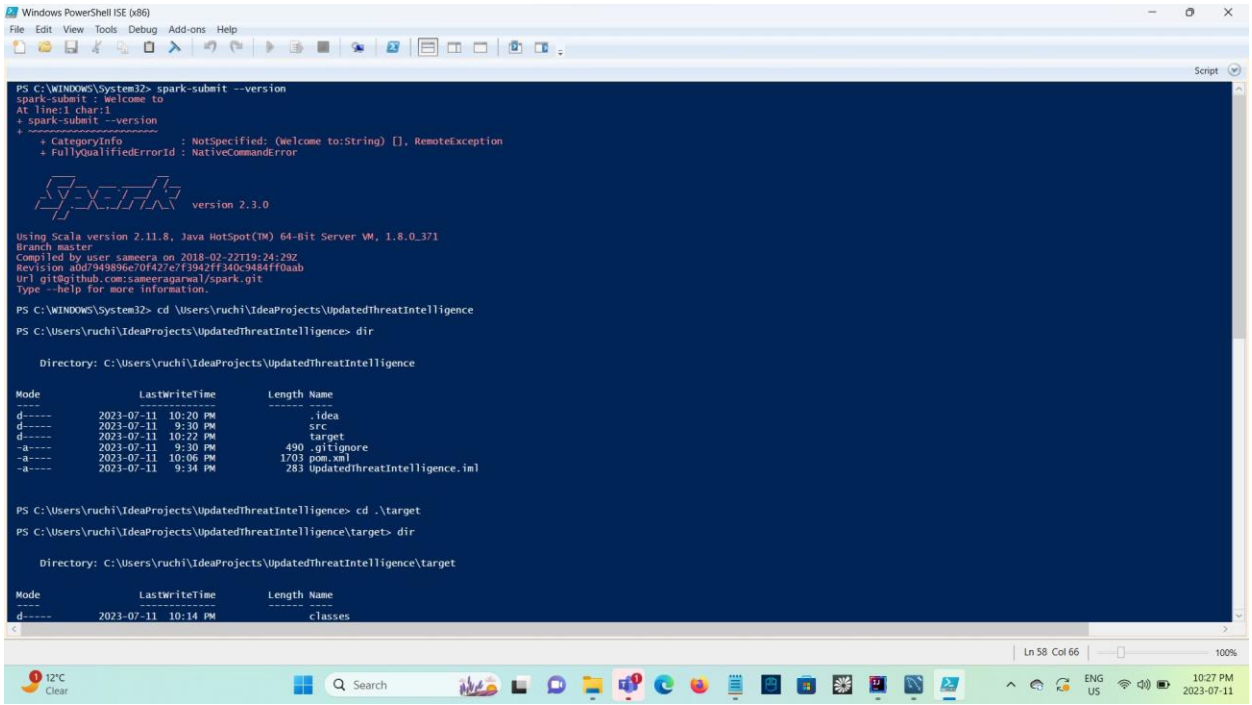


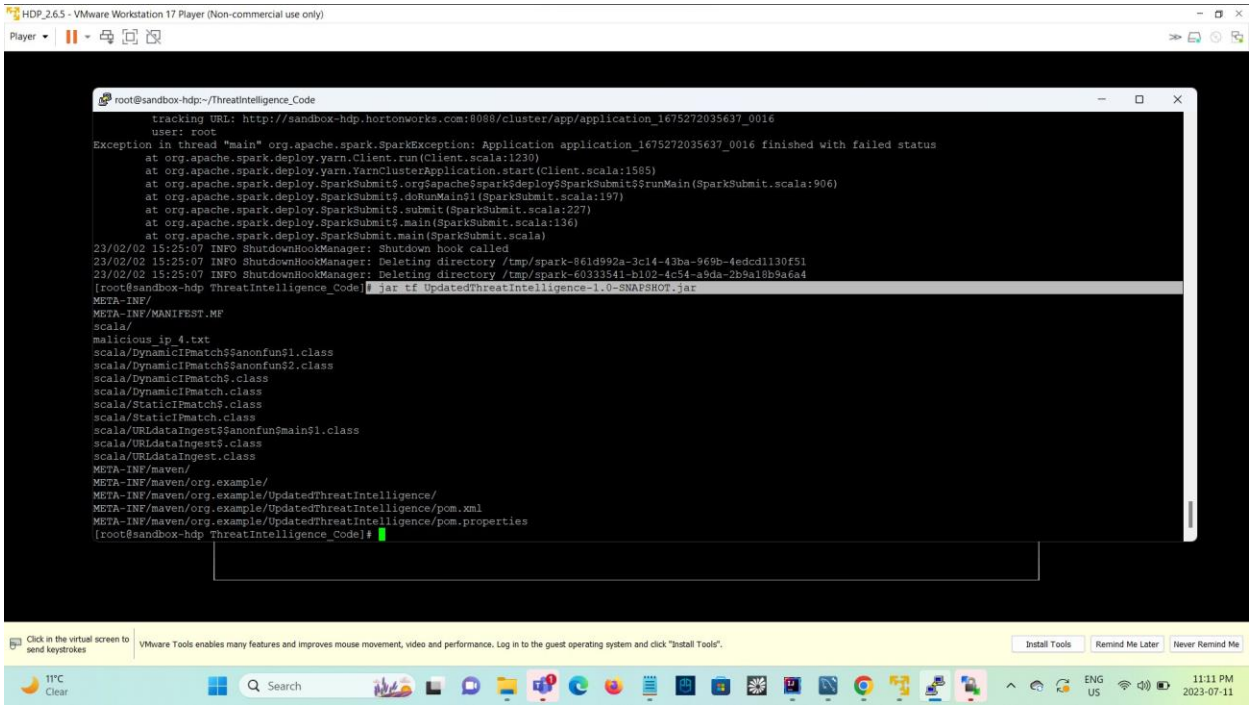


The Ambari was also checked to ensure all the services were running as shown on the screenshot above.

8. Troubleshooting







Achievement

We learned how to import malicious IP addresses from URL to MYSQL database using Scala code and its deployment on Hortonwork Cluster using Sandbox. Also, this assignment has given us an in-depth knowledge of using Scala code to seamlessly import data.