



## 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

<b>BDAT 1008</b>
<b>Data Collection and Curation</b>

## Table of Contents

### Part A: 1<sup>st</sup> Approach (KafkaLogs - SparkStreaming)

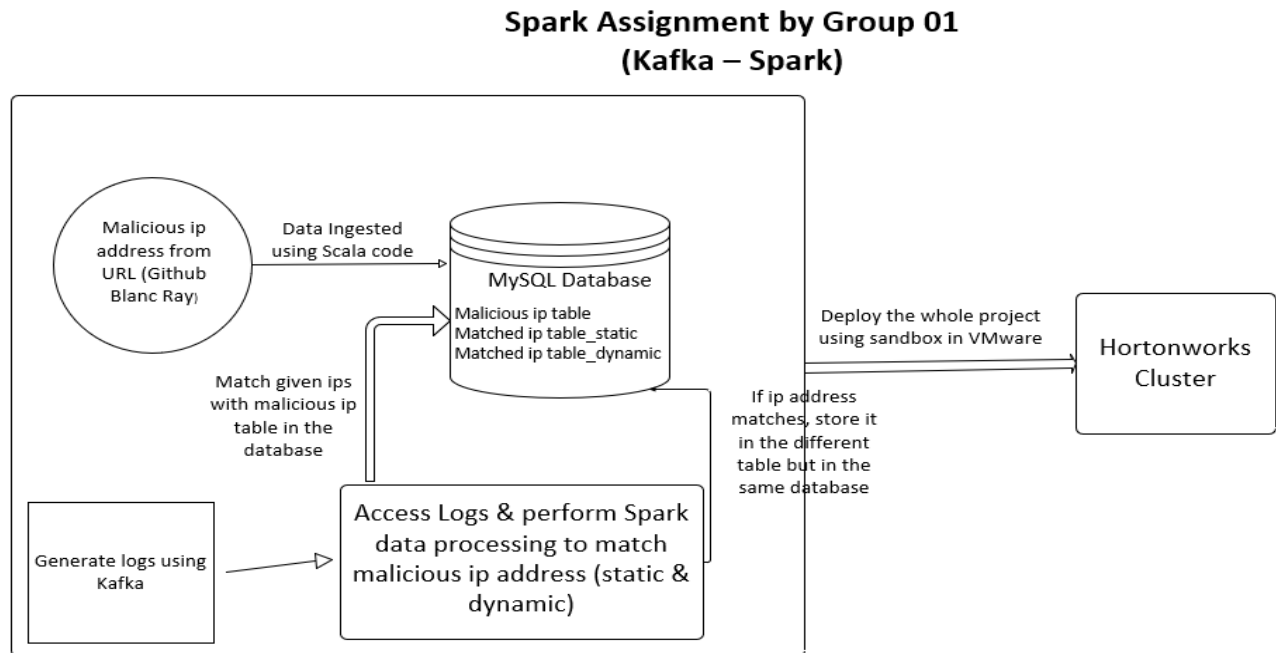
1. Layout of the project.....3
2. Generated Logs by KafkaProducerApp .....4
3. Access Logs & Process them by SparkStreaming.....5

### Part B: 2<sup>nd</sup> Approach (Programmatically with SparkSession)

4. Layout of the project.....6
5. Data Ingestion using Scala Code.....7
6. Dynamically matching and storing ip address using Scala.....8
7. Statically matching and storing ip address using Scala.....9
8. Packaged into jar file.....11
9. Login into Hortonworks sandbox using VMware through  
PuTTY.....12
10. Deploying the whole project on Hortonworks cluster.....12
  - Transferring text file and packaged jar file from local to cluster
  - Running spark job
11. Troubleshooting.....16
12. Achievement .....17

## Part A: 1<sup>st</sup> 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.

The screenshot shows a Windows Command Prompt window with the title "Command Prompt - kafka-ser". The output displays Kafka logs for a coordinator group, showing offsets and metadata loading for epoch 0. The logs include timestamps and broker IDs. A file explorer window is open in the background, showing the directory "C:\Users\ruchi\kafka\_2.12-3.3.1\libs\activation-1.1.1.jar".

```

[2023-07-17 20:21:56,892] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __cons
umer_offsets-6 in 64 milliseconds for epoch 0, of which 64 milliseconds was spent in the scheduler. (kafka.coordinator.g
roup.GroupMetadataManager)
[2023-07-17 20:21:56,893] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __cons
umer_offsets-9 in 65 milliseconds for epoch 0, of which 65 milliseconds was spent in the scheduler. (kafka.coordinator.g
roup.GroupMetadataManager)
[2023-07-17 20:21:56,893] INFO [GroupMetadataManager brokerId=0] Finished loading offsets and group metadata from __cons
umer_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 __cons
umer_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 __cons
umer_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 __cons
umer_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 __cons
umer_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 __cons
umer_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 __cons
umer_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
server)
[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\activation-repackaged-2.6.1.jar;C:\Users\ruchi\kafka_2.12-3.3.1\libs\argparse4
j-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\com
mons-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\comm
ons-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\conn
ect-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

```

The screenshot shows a Windows Command Prompt window with the title "Command Prompt". The output displays the execution of Kafka commands to create and list topics. The first command creates a topic named "IpAddresses" with 2 partitions and replication factor 1. The second command lists the topics, showing "IpAddresses".

```

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:188)
    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-fac
tor 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>

```

```

import java.util.Properties
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerRecord}
import scala.collection.JavaConverters._

object KafkaProducer_Logs {
  def main(args: Array[String]): Unit = {
    val inputFile = "C:\\Users\\ruchi\\IdeaProjects\\ThreatIntelligence_DataPipeline\\src\\malicious_ip_4.txt"
    val kafkaTopic = "IpAddresses"
    val kafkaServers = "localhost:9092" // Replace with your Kafka server address

    // Kafka producer configuration
    val props = new Properties()
    props.put("bootstrap.servers", kafkaServers)
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

    // Create Kafka producer
    val producer = new KafkaProducer[String, String](props)

    // Read IP addresses from the text file
    val ips = Files.readAllLines(Paths.get(inputFile)).asScala

    producer.send(new ProducerRecord(kafkaTopic, ips))
  }
}

```

### 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.

```

// Extract IP addresses from log messages
val logStream = kafkaStream.flatMap(_.value.split("\\s+"))
    .filter(ip => ip.matches("\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}"))

// Read the IP address data from MySQL table
val jdbcUrl = "jdbc:mysql://localhost:3306/malicious_ip_db"
val username = "root"
val password = "*****"
val sourceTableName = "malicious_ip_address"
val targetTableName = "matched_ip"

val ipAddressDF = spark.read.format("jdbc")
    .option("url", jdbcUrl)
    .option("user", username)
    .option("password", password)
    .option("dbtable", sourceTableName)
    .load()

// Perform IP address matching
val matchedIps = logStream.transform { rdd =>
  val logDF = spark.createDataFrame(rdd.map(Row(_)), ipAddressDF.schema)
  logDF.join(ipAddressDF, logDF.col("ip") === ipAddressDF.col("ip"))
}

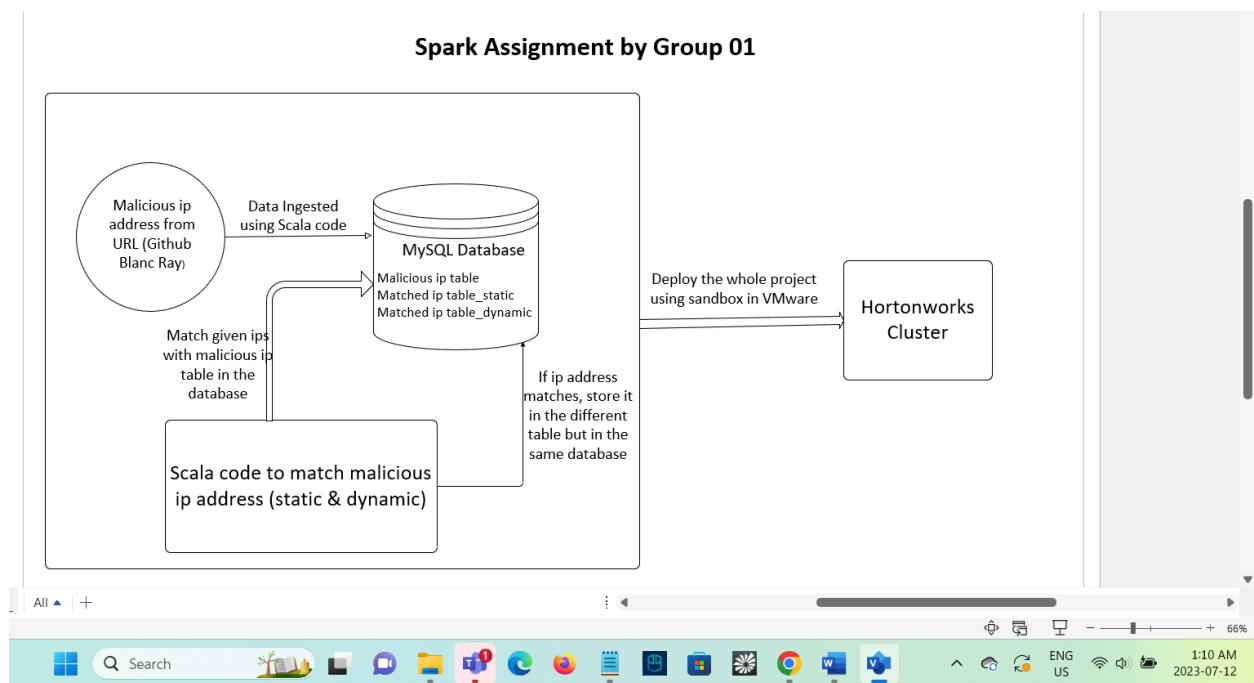
```



**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.**

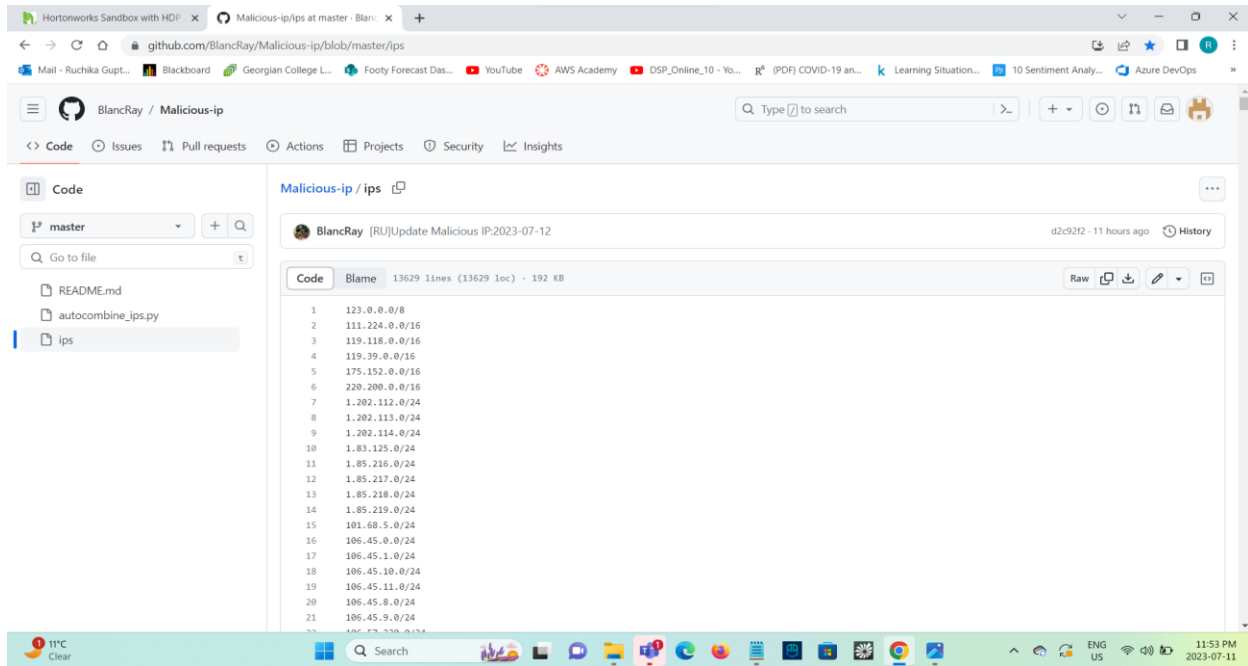
## Part B: 2<sup>nd</sup> Approach (Programmatically with SparkSession)

### 1. Layout of the project

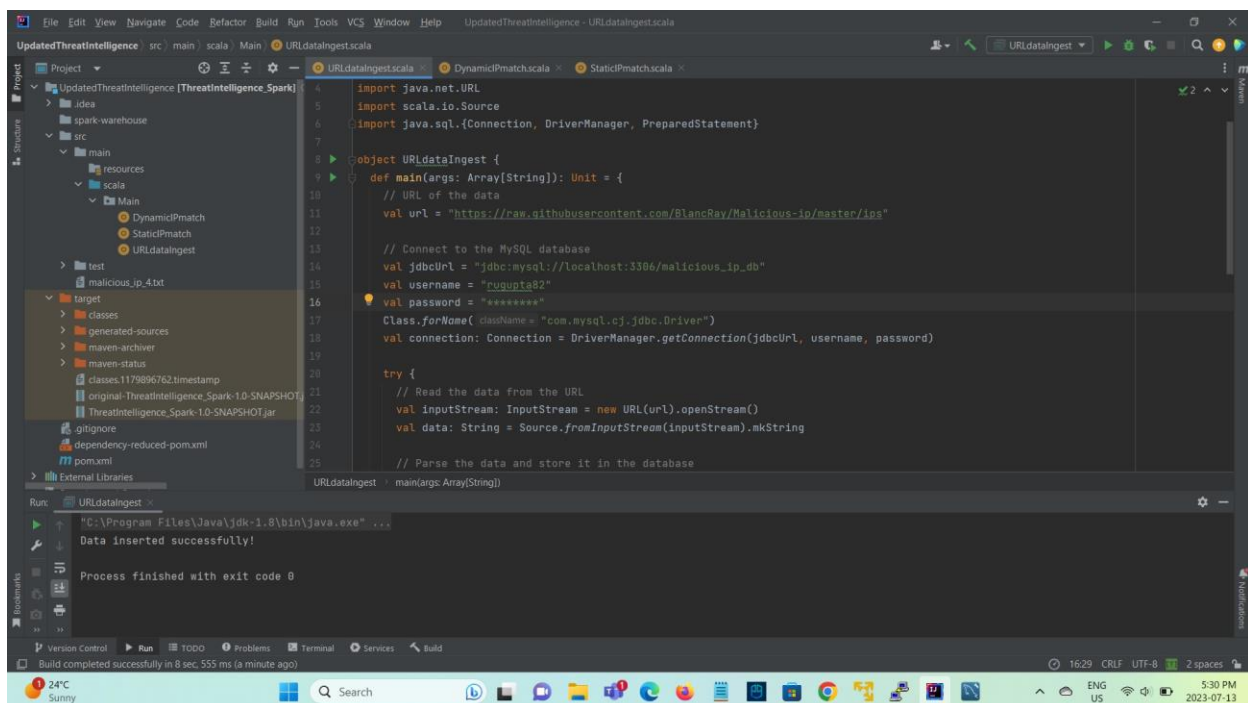


We ingested malicious IP address from a URL to the MYSQL database using Scala code. We developed Scala code to match given IP addresses to the addresses stored in database ( which are either static or dynamic). The matched addresses are stored in the same database but in a different table. We deployed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

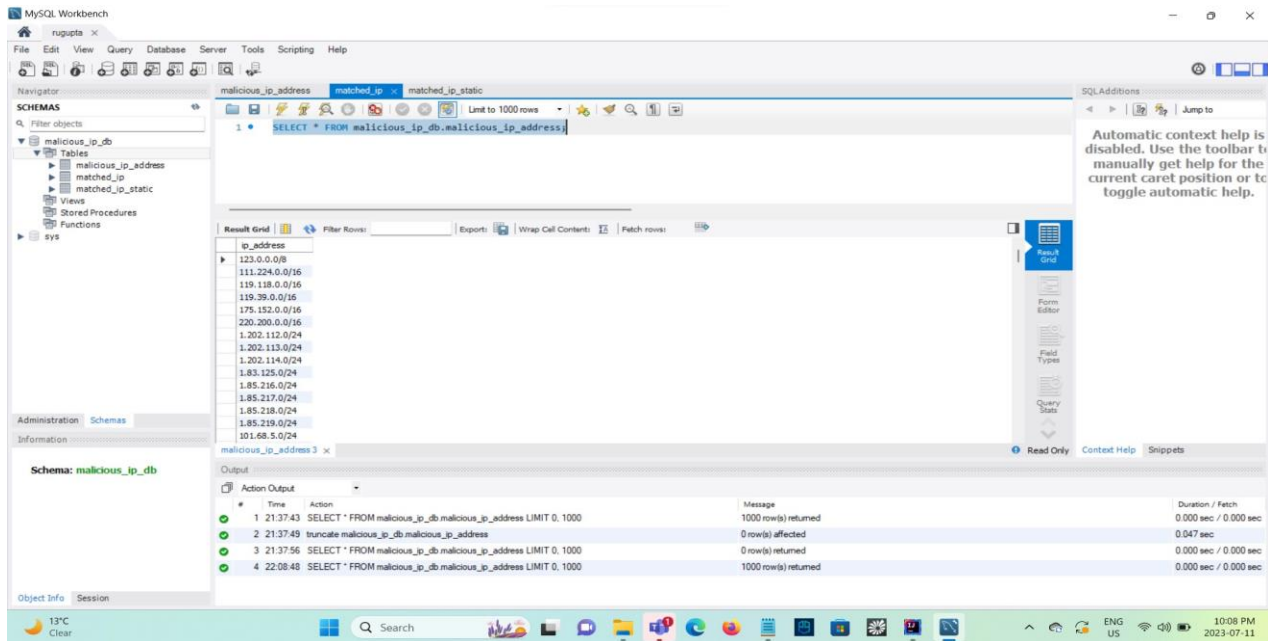
## 2. Data Ingestion using Scala Code



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

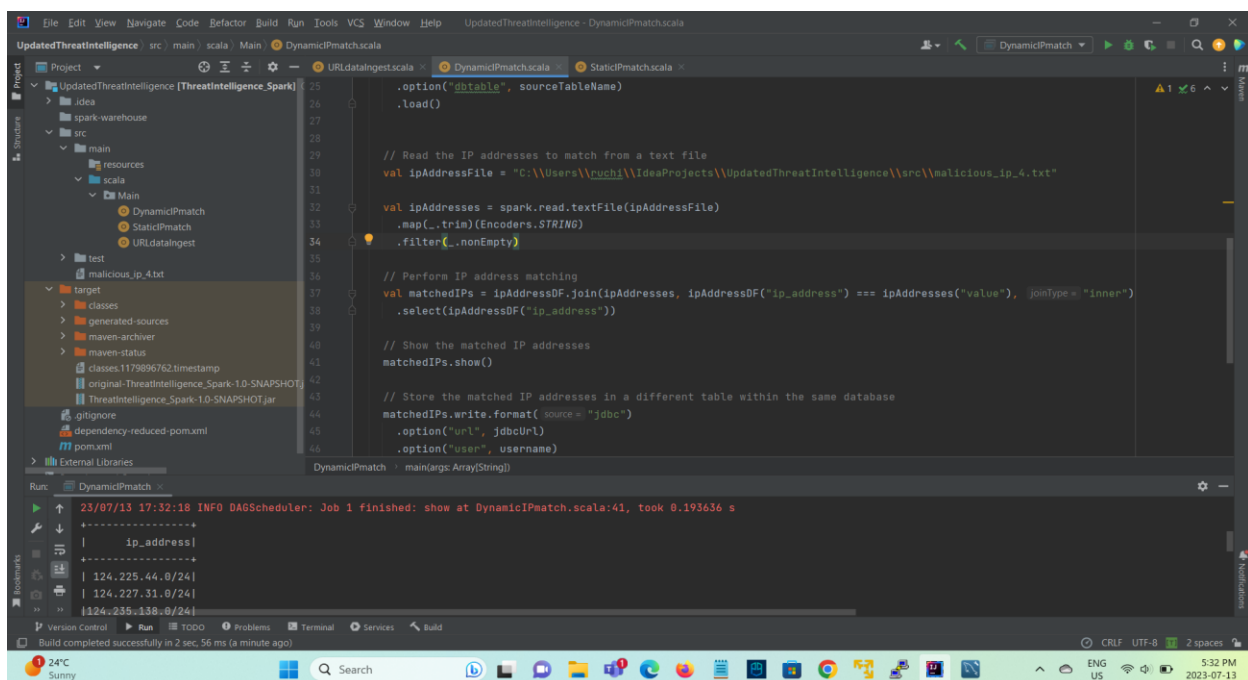


The screenshot above shows how we successfully inserted data. We moved the data in its current form into a Data Frame instance. With package, we provided a namespace to put our code in different files and directories. We indicated the URL and connected to the MySQL database and read data from the URL.



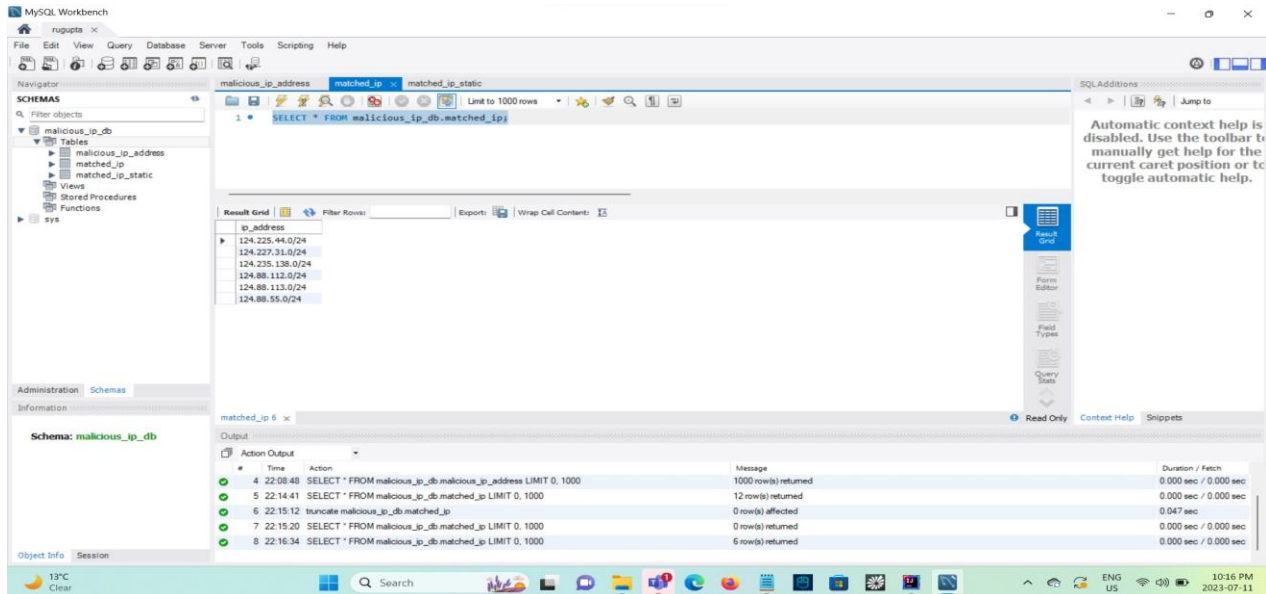
We selected all the fields by using “\*” from the malicious\_ip\_db. malicious\_ip\_address table and the result generated is shown on the screenshot above.

### 3. Dynamically matching and storing ip address using Scala.



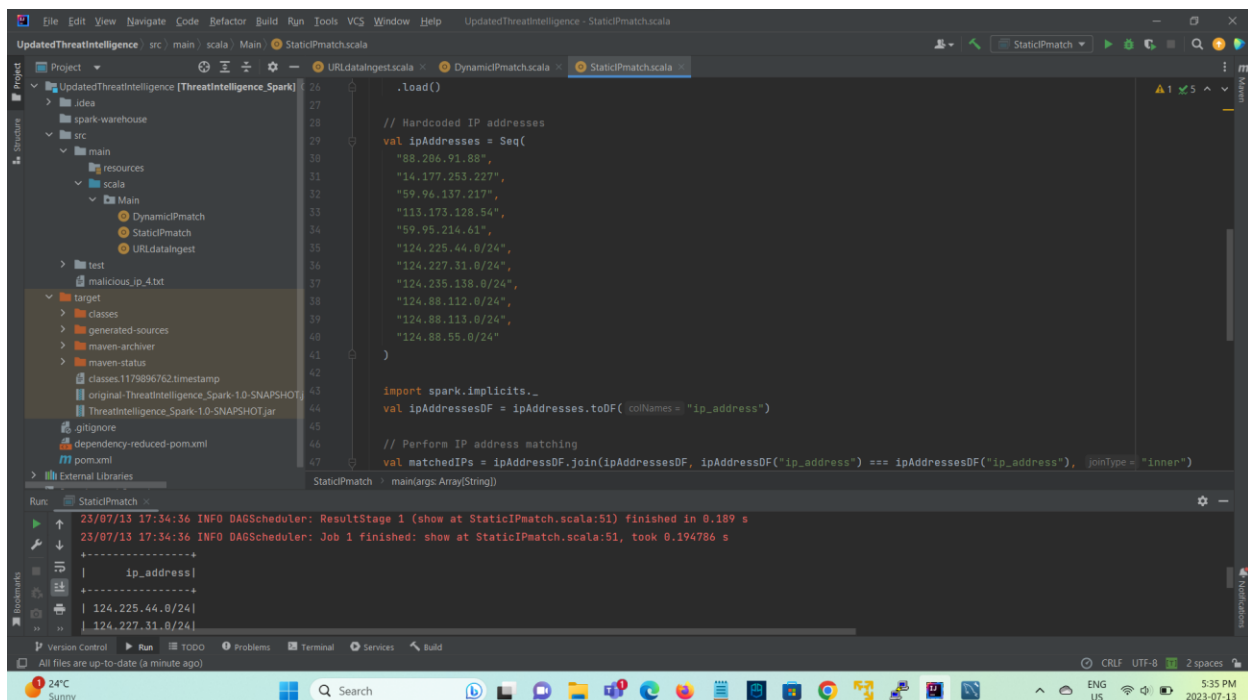


With the screenshot above, we performed IP address matching and showed the matched IP address by using `matchedIPs.show()`. We stored the matched IP addresses in a different table within the same database. The result of the matched Ips is shown in the above screenshot.



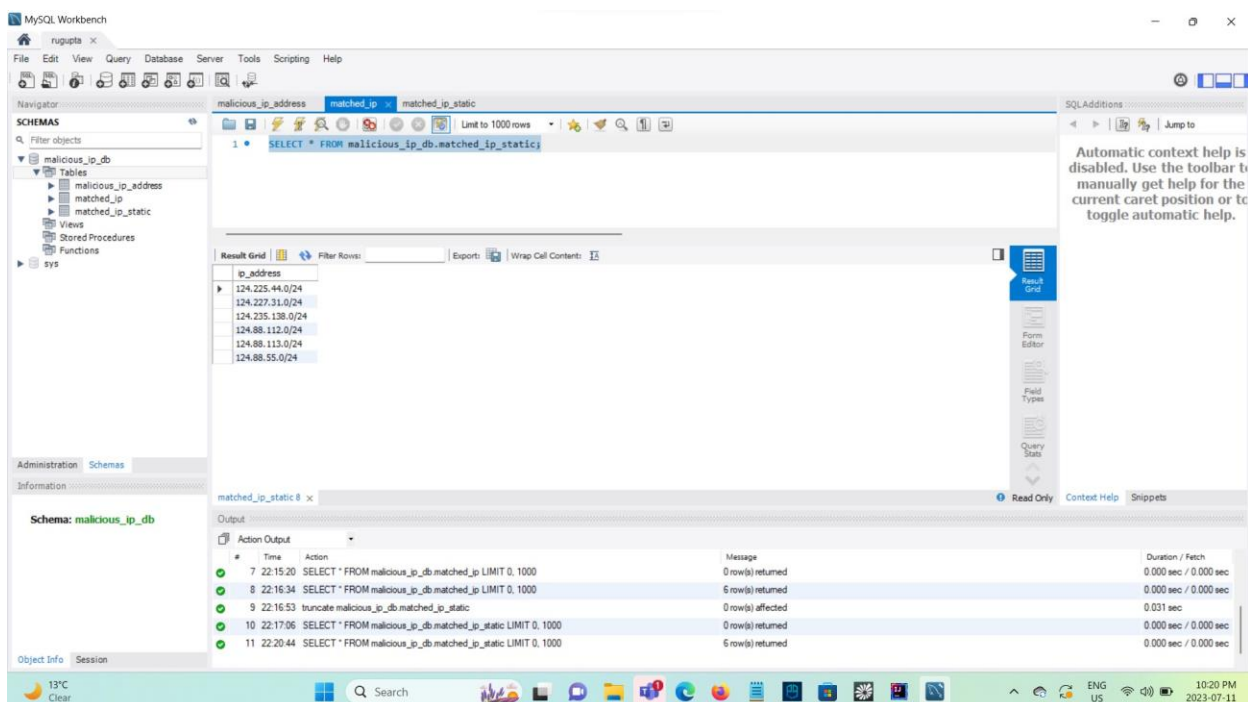
We retrieved all the matched Ips using the select “\*” from `malicious_ip_db.matched_ip` table and the result is shown in the screenshot above. The matched IP addresses are in a different table but same database as `malicious_ip_address`.

#### 4. Statically matching and storing Ip address using Scala.



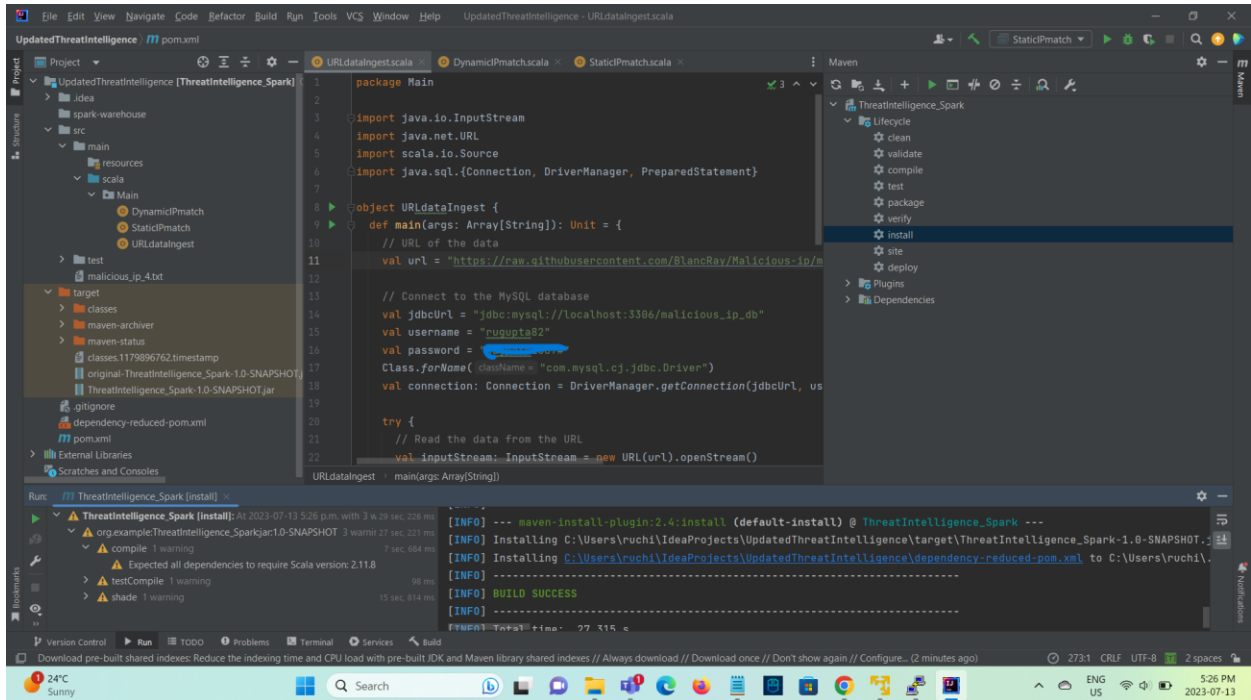
We manually assigned IP addresses that do not change in the settings of the machine. We were able to import the Ips using the import spark.implicits. the hardcoded addresses are stored with the matched Ip addresses and those that matched are stored in a different table called matched\_ip\_static.

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.

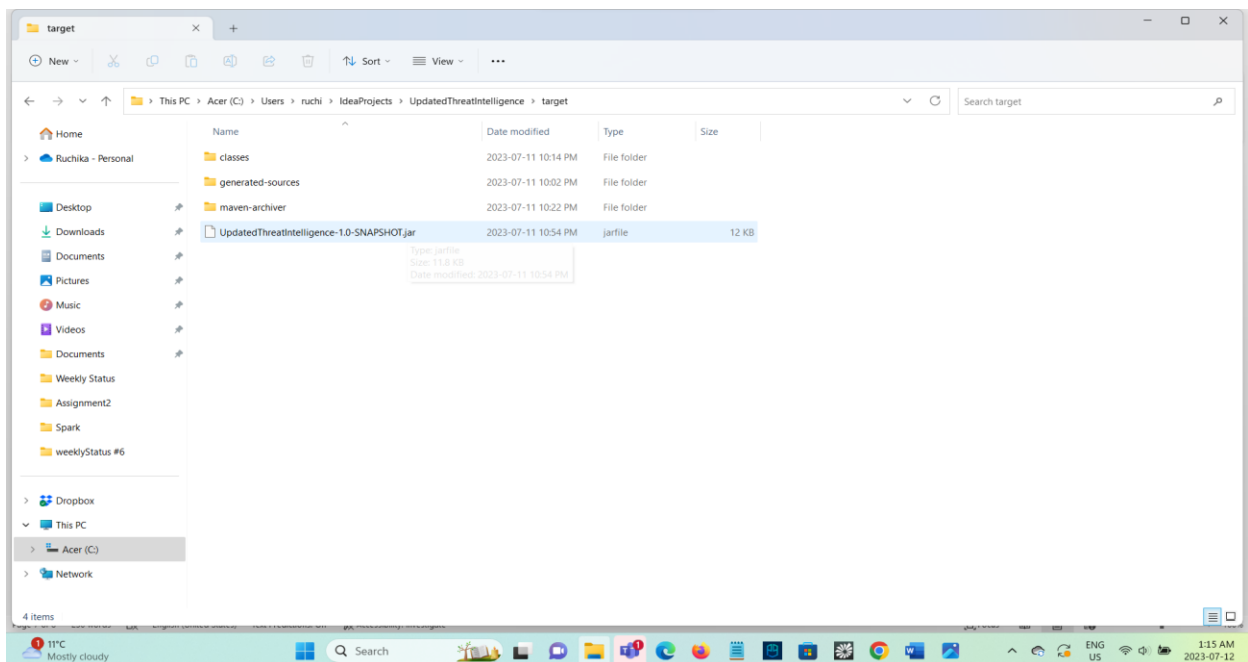


We retrieved all the matched Ips using the select “\*” from malicious\_ip\_db. matched\_ip table and the result is shown in the screenshot above. The result shows the IP addresses of the matched\_ip\_static table.

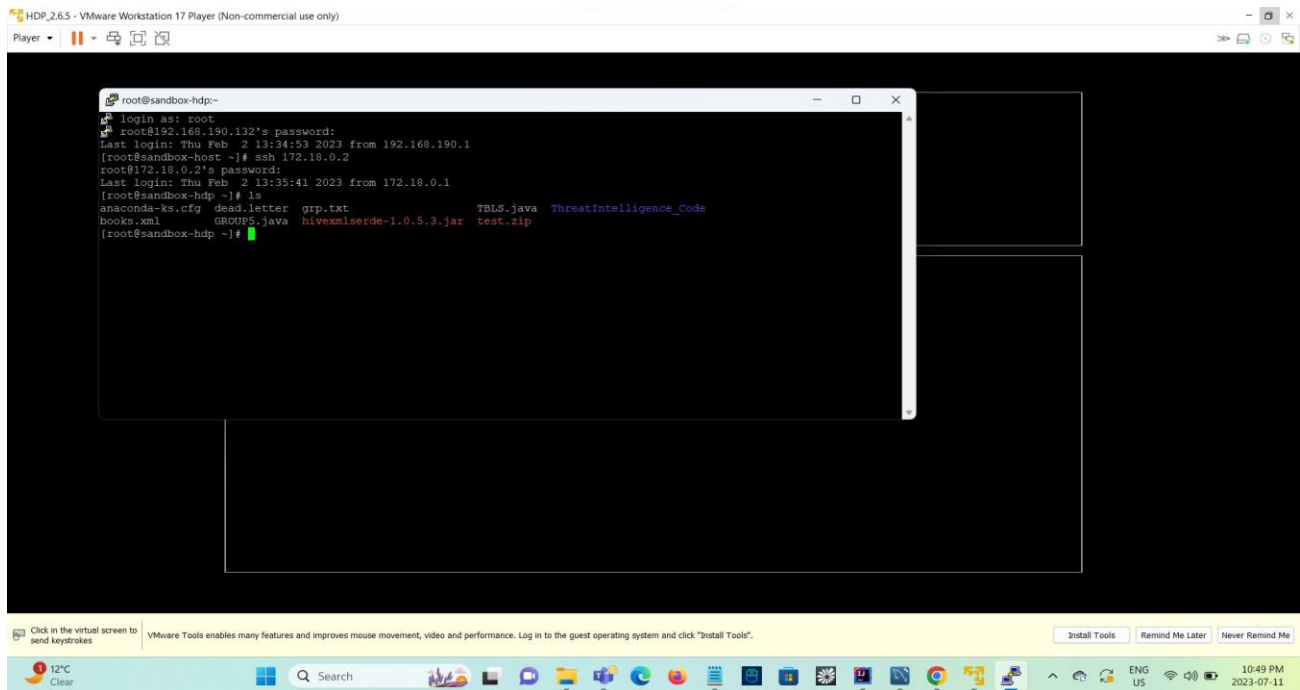
## 5. Packaged into jar file.



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.



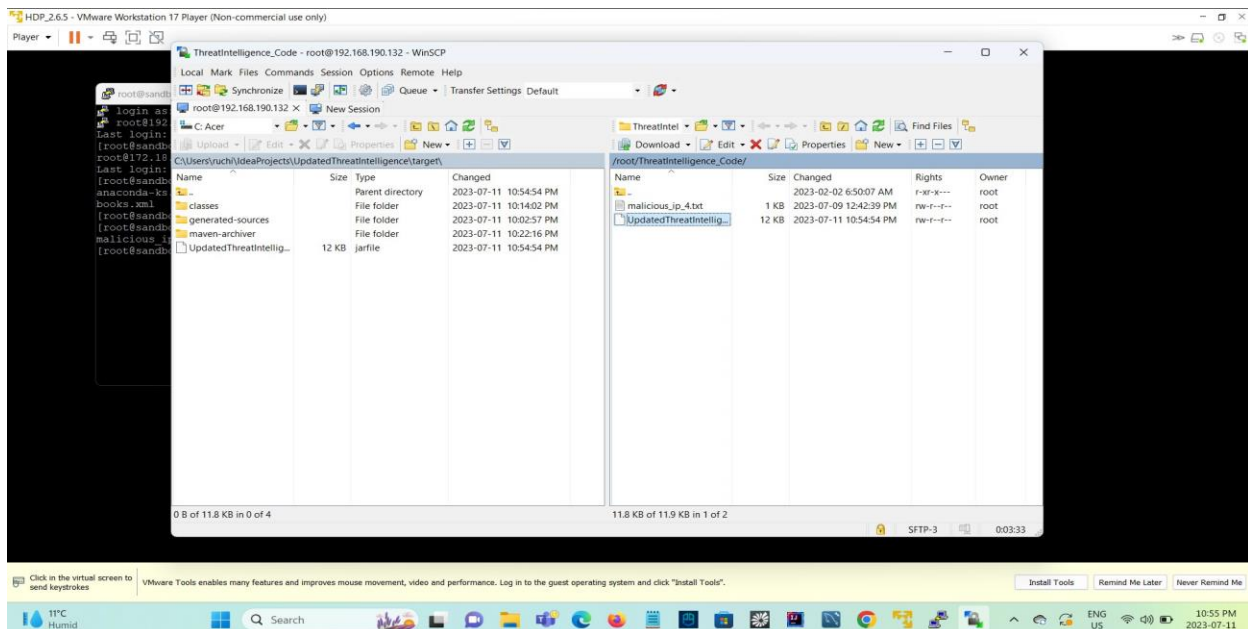
## 6. Login into Hortonworks sandbox using VMware through PuTTY



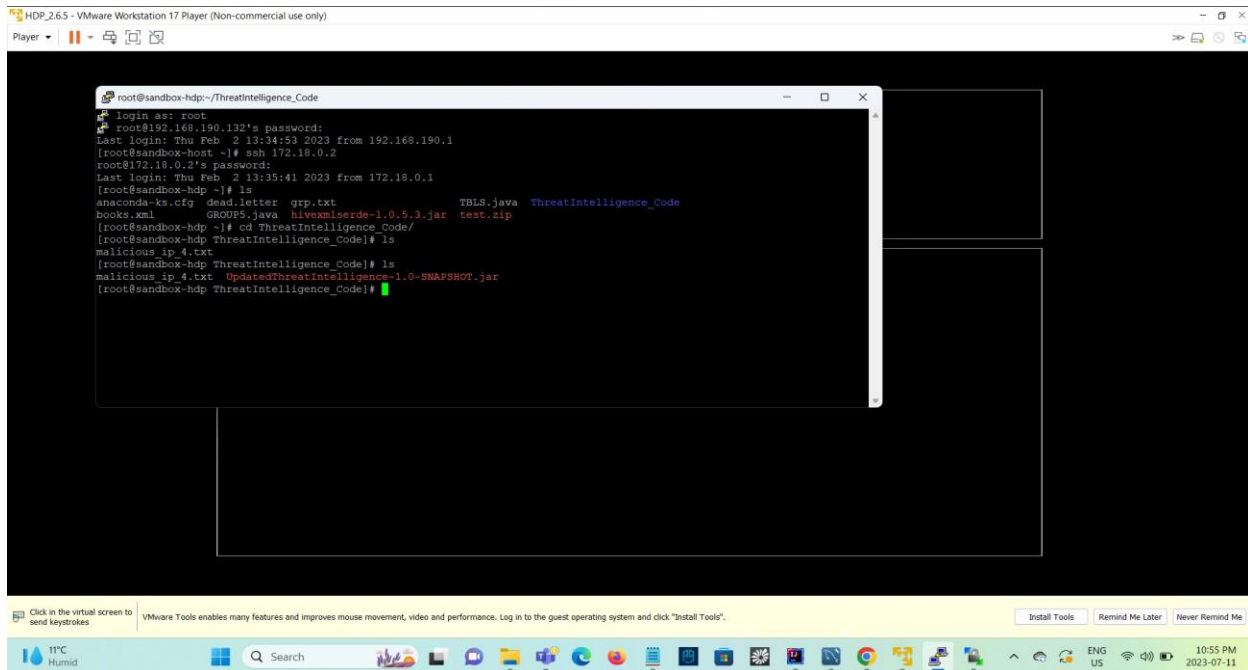
The screenshot shows the process for logging into the Cluster through PuTTY and socket shell(SSH) connection. In the sandbox, we created a directory called ThreatIntelligence\_code. We used "ls" command to show all the information about the directories and files in the sandbox-hdp directory.

## 7. Deploying the whole project on Hortonworks cluster

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

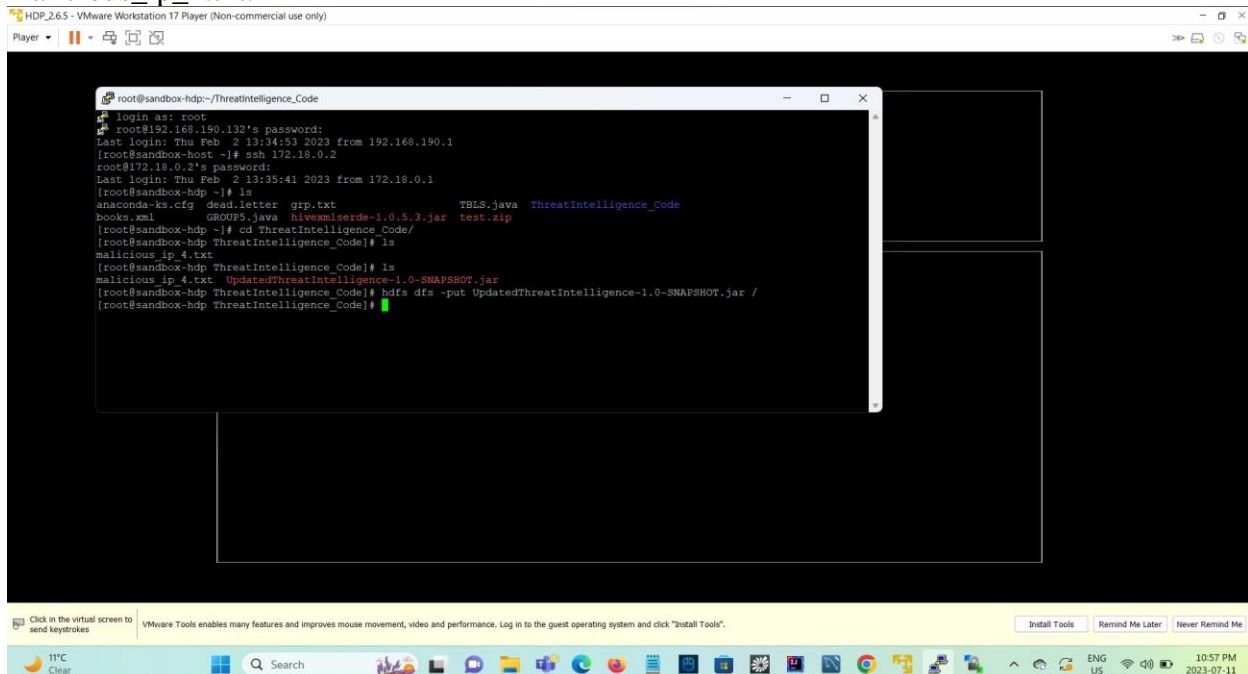


The screenshot above shows how we have copied our malicious\_ip\_4.txt text file and jar file named “updatedThreatIntelligence\_code” from the local computer to the sandbox with the help of WinSCP. We transferred the jar file from the local computer to the sandbox using WinSCP.



```
root@sandbox-hdp:~/ThreatIntelligence_Code
login as: root
root@192.168.190.132's password:
Last login: Thu Feb  2 13:34:53 2023 from 192.168.190.1
[root@sandbox-host ~]# ssh 172.18.0.2
root@172.18.0.2's password:
Last login: Thu Feb  2 13:35:41 2023 from 172.18.0.1
[root@sandbox-hdp ~]# ls
anaconda-ks.cfg  dead.letter  grp.txt      TBL5.java  ThreatIntelligence_Code
books.xml        GROUP5.java  Hivexmlserde-1.0.5.3.jar  test.zip
[root@sandbox-hdp ~]# cd ThreatIntelligence_Code/
[root@sandbox-hdp ThreatIntelligence_Code]# ls
malicious_ip_4.txt
[root@sandbox-hdp ThreatIntelligence_Code]# ls
malicious_ip_4.txt  UpdatedThreatIntelligence-1.0-SNAPSHOT.jar
[root@sandbox-hdp ThreatIntelligence_Code]#
```

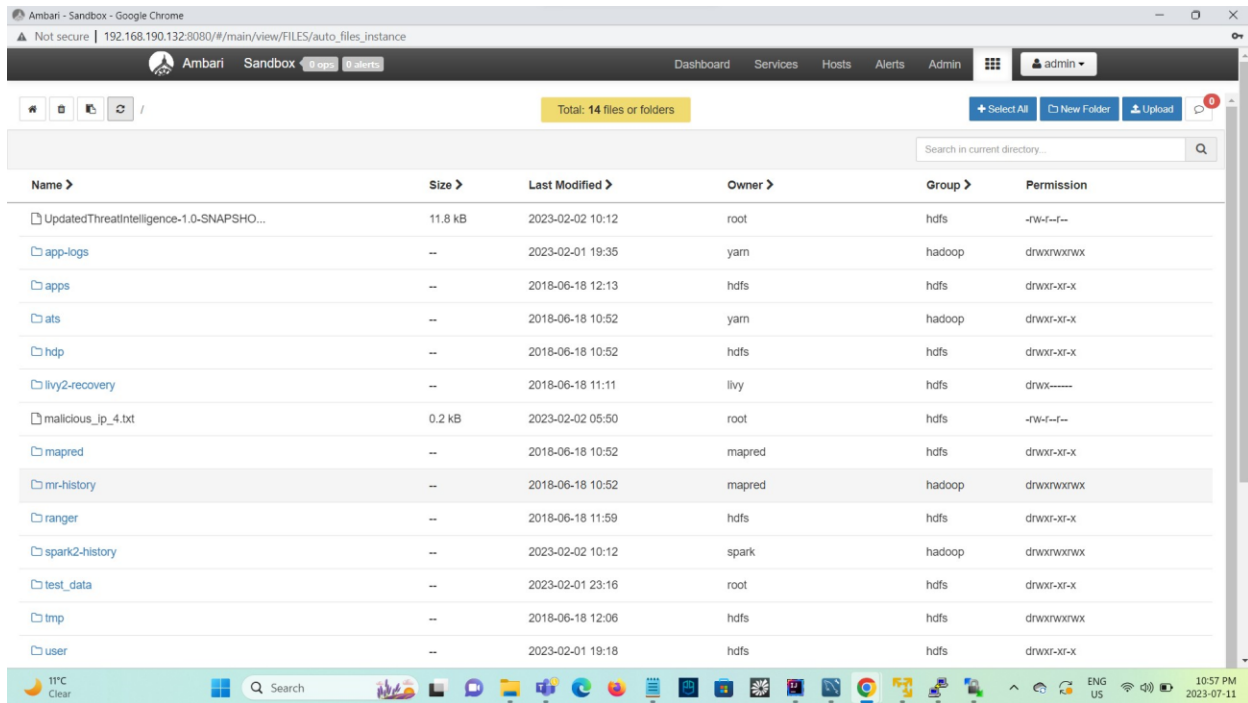
The screenshot above shows files transferred to the cluster using “ls” command to list all the directories and files in the working sandbox-hdp directory. We changed the current working directory to ThreatIntelligence\_code and listed all the files in it using “ls” command to show the malicious\_ip\_4.txt.



```
root@sandbox-hdp:~/ThreatIntelligence_Code
login as: root
root@192.168.190.132's password:
Last login: Thu Feb  2 13:34:53 2023 from 192.168.190.1
[root@sandbox-host ~]# ssh 172.18.0.2
root@172.18.0.2's password:
Last login: Thu Feb  2 13:35:41 2023 from 172.18.0.1
[root@sandbox-hdp ~]# ls
anaconda-ks.cfg  dead.letter  grp.txt      TBL5.java  ThreatIntelligence_Code
books.xml        GROUP5.java  Hivexmlserde-1.0.5.3.jar  test.zip
[root@sandbox-hdp ~]# cd ThreatIntelligence_Code/
[root@sandbox-hdp ThreatIntelligence_Code]# ls
malicious_ip_4.txt
[root@sandbox-hdp ThreatIntelligence_Code]# ls
malicious_ip_4.txt  UpdatedThreatIntelligence-1.0-SNAPSHOT.jar
[root@sandbox-hdp ThreatIntelligence_Code]# hdfs dfs -put UpdatedThreatIntelligence-1.0-SNAPSHOT.jar /
[root@sandbox-hdp ThreatIntelligence_Code]#
```

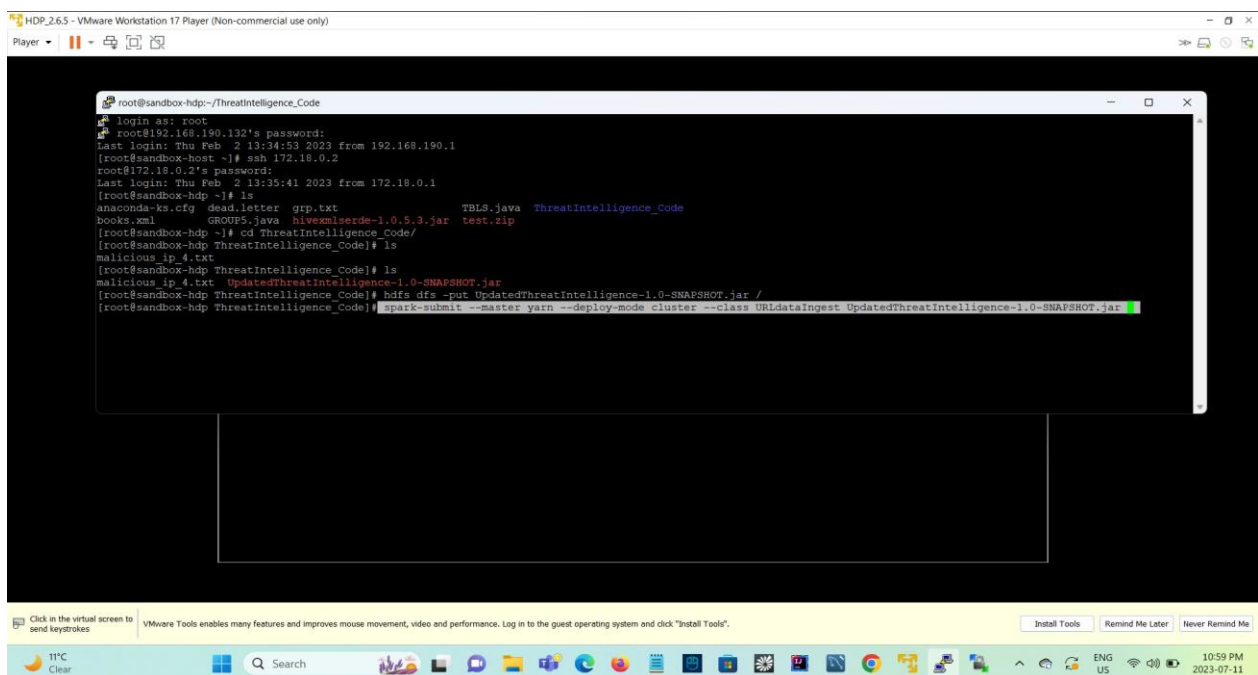


The screenshot above shows using put command to transfer jar file and the malicious Ip addresses text file to the HDFS.

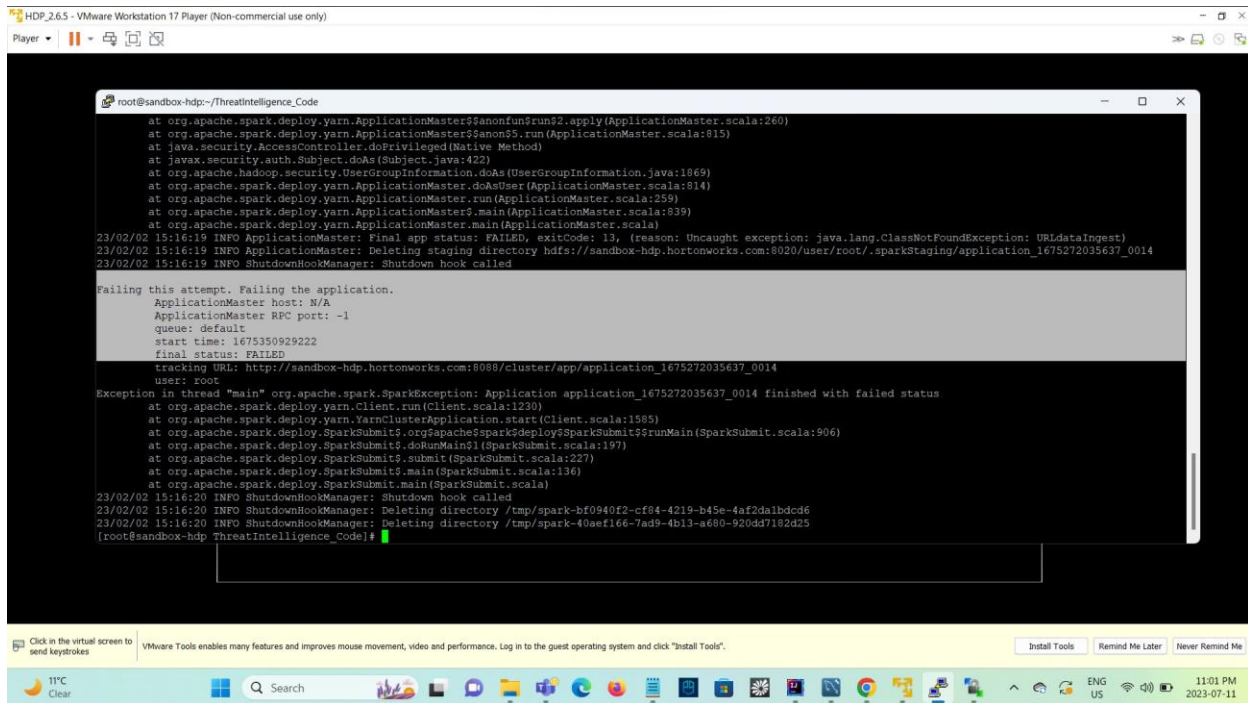


The screenshot above shows the two files i.e., jar file and text file transferred from local computer to the HDFS.

- Running spark job

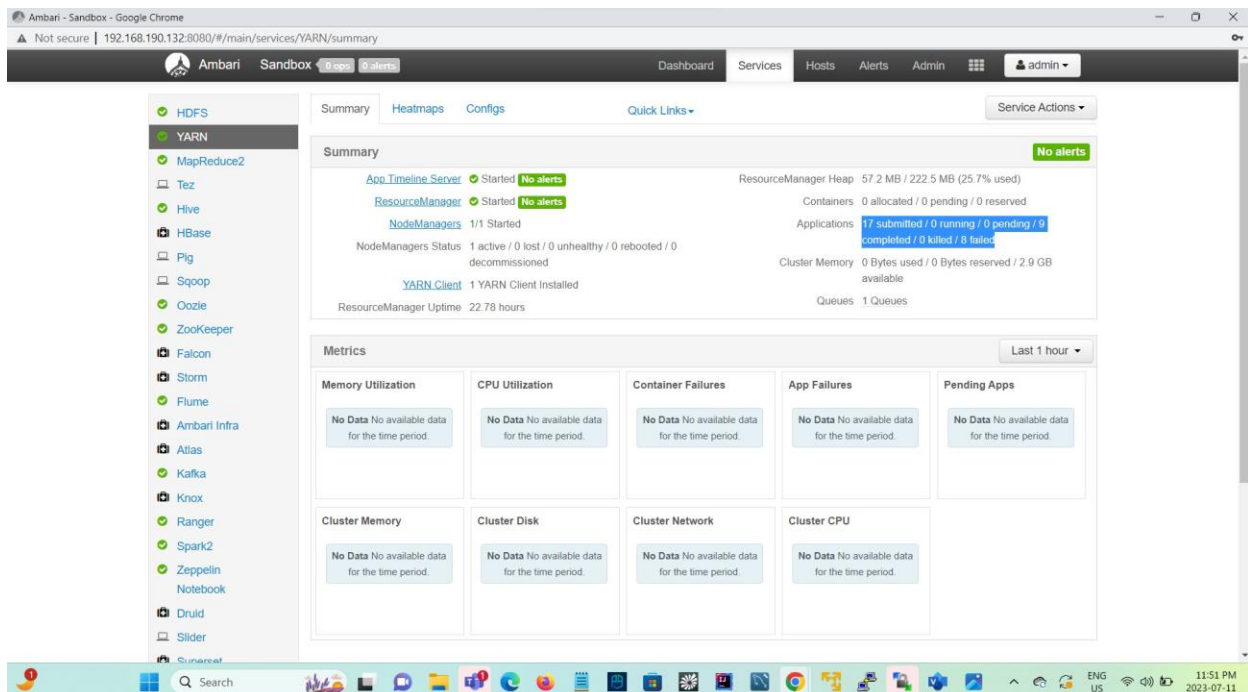


The “spark\_submit” command was used to launch an application in the cluster. With the “spark\_submit” command, we were able to run the jar file.



```
root@sandbox-hdp:~/ThreatIntelligence_Code
at org.apache.spark.deploy.yarn.ApplicationMaster$.anonfun$run$2.apply(ApplicationMaster.scala:260)
at org.apache.spark.deploy.yarn.ApplicationMaster$.anonfun$run$2.apply(ApplicationMaster.scala:260)
at java.security.AccessController.doPrivileged(Native Method)
at javax.security.auth.Subject.doAs(Subject.java:422)
at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1869)
at org.apache.spark.deploy.yarn.ApplicationMaster.doAsUser(ApplicationMaster.scala:814)
at org.apache.spark.deploy.yarn.ApplicationMaster.run(ApplicationMaster.scala:259)
at org.apache.spark.deploy.yarn.ApplicationMaster$.main(ApplicationMaster.scala:839)
at org.apache.spark.deploy.yarn.ApplicationMaster.main(ApplicationMaster.scala)
23/02/02 15:16:19 INFO ApplicationMaster: Final app status: FAILED, exitCode: 13, (reason: Uncaught exception: java.lang.ClassNotFoundException: URDataIngest)
23/02/02 15:16:19 INFO ApplicationMaster: Deleting staging directory hdfs://sandbox-hdp.hortonworks.com:8020/user/root/.sparkStaging/application_167527035637_0014
23/02/02 15:16:19 INFO ShutdownHookManager: Shutdown hook called

Failing this attempt. Failing the application.
ApplicationMaster host: N/A
ApplicationMaster RPC port: -1
queue: default
start time: 167535092222
final status: FAILED
tracking URL: http://sandbox-hdp.hortonworks.com:8088/cluster/app/application_167527035637_0014
user: root
Exception in thread "main" org.apache.spark.SparkException: Application application_167527035637_0014 finished with failed status
at org.apache.spark.deploy.yarn.Client.run(Client.scala:1230)
at org.apache.spark.deploy.yarn.YarnClusterApplication.start(Client.scala:1585)
at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:906)
at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:197)
at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:227)
at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:136)
at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
23/02/02 15:16:20 INFO ShutdownHookManager: Shutdown hook called
23/02/02 15:16:20 INFO ShutdownHookManager: Deleting directory /tmp/spark-bf0940f2-cf84-4219-b45e-4af2da1bcd6
23/02/02 15:16:20 INFO ShutdownHookManager: Deleting directory /tmp/spark-40aef166-7ad9-4b13-a680-920dd7182d25
[root@sandbox-hdp ThreatIntelligence_Code]#
```



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

## 8. Troubleshooting

```

root@sandbox-hdp:~/ThreatIntelligence_Code
ApplicationMaster RPC port: -1
queue: default
start time: 1675351677754
final status: FAILED
tracking URL: http://sandbox-hdp.hortonworks.com:8088/cluster/app/application_1675272035637_0017
user: root
Exception in thread "main" org.apache.spark.SparkException: Application application_1675272035637_0017 finished with failed status
    at org.apache.spark.deploy.yarn.Client.run(Client.scala:1230)
    at org.apache.spark.deploy.yarn.YarnClusterApplication.start(Client.scala:1585)
    at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:906)
    at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:157)
    at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:227)
    at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:136)
    at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
23/02/02 15:28:40 INFO ShutdownHookManager: Shutdown hook called
23/02/02 15:28:40 INFO ShutdownHookManager: Deleting directory /tmp/spark-add05a65-5a94-4f24-86d9-5934d2640ff6
23/02/02 15:28:40 INFO ShutdownHookManager: Deleting directory /tmp/spark-05422be5-7a45-453a-95a3-e1c656d10c62
[root@sandbox-hdp ThreatIntelligence_Code]# spark-submit --version
SPARK_MAJOR_VERSION is set to 2, using Spark2
Welcome to

Spark version 2.3.0.2.6.5.0-292

Using Scala version 2.11.8, OpenJDK 64-Bit Server VM, 1.8.0_171
Branch HEAD
Compiled by user jenkins on 2018-05-11T08:28:14Z
Revision b6b578cd8ff89bfc079e1ea3bda074262c3800ad
Url git@github.com:hortonworks/spark2.git
Type --help for more information.
[root@sandbox-hdp ThreatIntelligence_Code]#

```

```

PS C:\WINDOWS\System32> spark-submit --version
spark-submit : welcome to
At line:1 char:1
+ spark-submit --version
+ ~~~~~
+ CategoryInfo          : NotSpecified: (Welcome:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

Spark version 2.3.0

Using Scala version 2.11.8, Java HotSpot(TM) 64-Bit Server VM, 1.8.0_371
Branch master
Compiled by user sameera on 2018-02-22T19:24:29Z
Revision a0d7949896c70f427ef3942ff340c9484ff0aab
Url git@github.com:sameeragarwal/spark.git
Type --help for more information.

PS C:\WINDOWS\System32> cd \Users\ruchi\IdeaProjects\UpdatedThreatIntelligence
PS C:\Users\ruchi\IdeaProjects\UpdatedThreatIntelligence> dir

Directory: C:\Users\ruchi\IdeaProjects\UpdatedThreatIntelligence

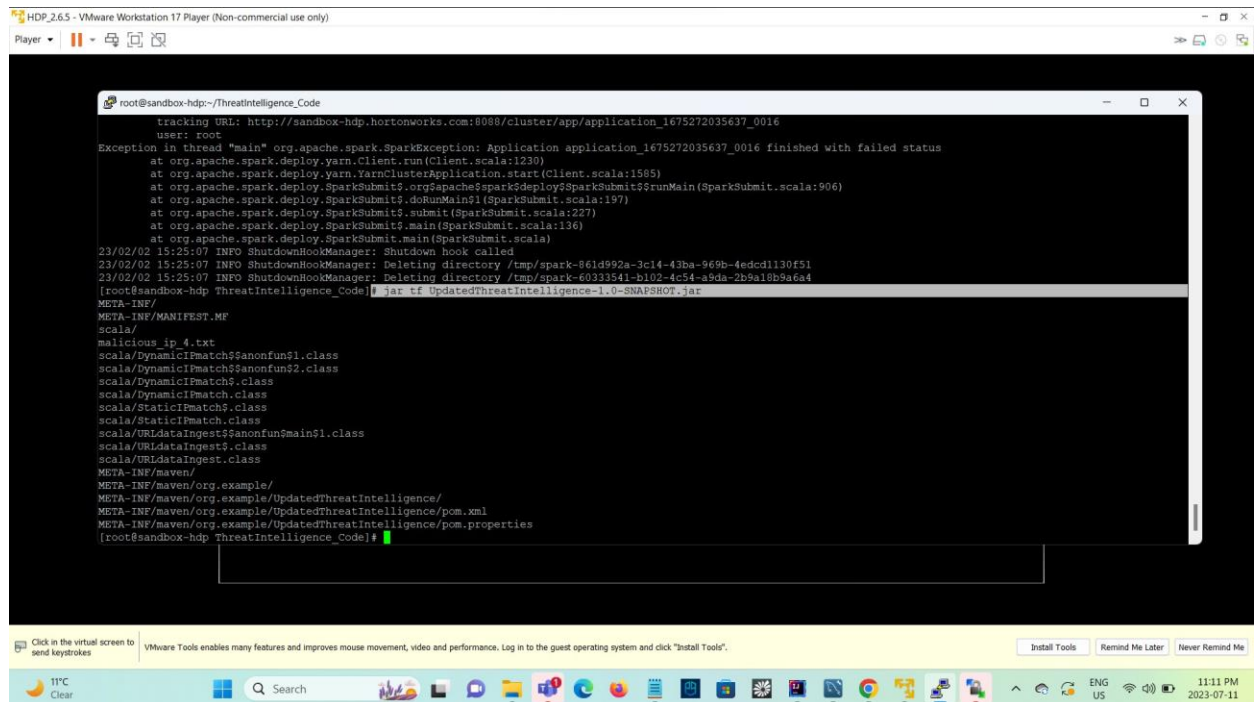
Mode                LastWriteTime         Length Name
----                -
d-----         2023-07-11 10:20 PM             .idea
d-----         2023-07-11  9:30 PM             src
d-----         2023-07-11 10:22 PM             target
-a-----         2023-07-11  9:30 PM             490 .gitignore
-a-----         2023-07-11 10:06 PM             1703 pom.xml
-a-----         2023-07-11  9:34 PM             283 UpdatedThreatIntelligence.iml

PS C:\Users\ruchi\IdeaProjects\UpdatedThreatIntelligence> cd .\target
PS C:\Users\ruchi\IdeaProjects\UpdatedThreatIntelligence\target> dir

Directory: C:\Users\ruchi\IdeaProjects\UpdatedThreatIntelligence\target

Mode                LastWriteTime         Length Name
----                -
d-----         2023-07-11 10:14 PM             classes

```



```
tracking URL: http://sandbox-hdp.hortonworks.com:8088/cluster/app/application_1675272035637_0016
user: root
Exception in thread "main" org.apache.spark.SparkException: Application application_1675272035637_0016 finished with failed status
    at org.apache.spark.deploy.yarn.Client.run(Client.scala:1230)
    at org.apache.spark.deploy.yarn.YarnClusterApplication.start(Client.scala:1585)
    at org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(SparkSubmit.scala:906)
    at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:197)
    at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:227)
    at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:136)
    at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
23/02/02 15:25:07 INFO ShutdownHookManager: Shutdown hook called
23/02/02 15:25:07 INFO ShutdownHookManager: Deleting directory /tmp/spark-861d992a-3c14-43ba-969b-4edcd1130f51
23/02/02 15:25:07 INFO ShutdownHookManager: Deleting directory /tmp/spark-60333541-b102-4c54-a5da-2b8a18b9a6a4
[root@sandbox-hdp ThreatIntelligence_Code]# jar tf UpdatedThreatIntelligence-1.0-SNAPSHOT.jar
META-INF/
META-INF/MANIFEST.MF
scala/
malicious_ip_4.txt
scala/DynamicIPmatch$$anonfun$1.class
scala/DynamicIPmatch$$anonfun$2.class
scala/DynamicIPmatch$.class
scala/DynamicIPmatch.class
scala/StaticIPmatch$.class
scala/StaticIPmatch.class
scala/URLdataIngest$$anonfun$main$1.class
scala/URLdataIngest$.class
scala/URLdataIngest.class
META-INF/maven/
META-INF/maven/org.example/UpdatedThreatIntelligence/
META-INF/maven/org.example/UpdatedThreatIntelligence/pom.xml
META-INF/maven/org.example/UpdatedThreatIntelligence/pom.properties
[root@sandbox-hdp ThreatIntelligence_Code]#
```

## 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.