DISTRIBUTED DATABASE ENGINEERING – LABORATORY REPORT

Implementation of a Multi-Node Hadoop Cluster for Road Safety Data Analysis

This report details the installation, configuration, and validation of a three-node Hadoop cluster for analysing 2018 road safety data. A MapReduce program is implemented to aggregate accident occurrences by severity, demonstrating how distributed systems handle large-scale datasets in alignment with industry best practices.

Table of Contents

1. Introduction & Design

1.1 Problem Statement

Road safety remains a major public health concern worldwide, requiring continued study into ways for limiting accidents and shaping transport policies. In the United Kingdom, for example, the Department for Transport regularly publishes detailed casualty figures, allowing for data-driven decision-making in areas like as infrastructure planning, car safety legislation, and driver awareness programs. Analysing these large-scale datasets frequently requires substantial processing resources, demanding a stable and scalable environment.

This laboratory exercise addresses the challenge of processing the **2018 road safety casualties dataset** within a multi-node Hadoop cluster. By employing the **MapReduce** framework—originally conceptualised by **Dean and Ghemawat (2008)**—the task is to count the total number of accident occurrences categorised by casualty severity. Such a distributed approach demonstrates how **Big Data** technologies can uncover insights from extensive real-world datasets efficiently and reliably.

1.2 Objectives

The overarching objectives of this project are as follows:

1. **Distributed Environment Setup**
    - Install and configure Java, Hadoop, and Apache Spark across three virtual machines to form a functional multi-node cluster.
    - Justify the choice of a 3-node architecture based on **fault tolerance**, **parallel processing**, and **scalability** considerations.
2. **Data Processing with MapReduce**
    - Develop a Java-based MapReduce application to parse the CSV file, filter out any header rows, and extract the *Casualty Severity* column for aggregation.
    - Validate the success of the MapReduce job through terminal outputs and the Hadoop web interfaces (e.g., NameNode at port 50070 and Resource Manager at port 8088).
3. **Evaluation & Critique**
    - Discuss potential enhancements, including using **Spark's** in-memory capabilities for iterative analytics.

1.3 System Overview & Design Approach

The system adopts a **distributed architecture** that includes a **primary node** (machine01) and **two secondary nodes** (machine02 and machine03). This setup follows standard practices in Big Data engineering:

- **Hadoop Ecosystem**
    - **HDFS (Hadoop Distributed File System):** Stores large datasets reliably across multiple nodes (Apache Hadoop Official Documentation, 2025).
    - **YARN (Yet Another Resource Negotiator):** Manages cluster resources, distributing MapReduce tasks among available DataNodes.

    - **MapReduce:** Handles batch processing by splitting input data among Mappers and aggregating partial results in the Reducers (Dean and Ghemawat, 2008).
- **Apache Spark Installation**
    - Although the primary demonstration uses MapReduce, Spark is installed to align with modern industry practices, where in-memory processing can expedite analytics (Apache Spark Official Documentation, 2025).
    - Future work could leverage Spark's **Resilient Distributed Datasets (RDDs)** or **DataFrame** APIs for complex tasks such as real-time analytics or iterative machine learning.
- **Why a Multi-Node Cluster?**

0. **Scalability:** Distributing data across multiple nodes mitigates performance bottlenecks as data volume grows.
1. **Fault Tolerance:** In the event of a node failure, the cluster can continue operation, a crucial feature for real-world data pipelines (Apache Hadoop Official Documentation, 2025).
2. **Parallel Processing:** Map tasks are carried out concurrently on different nodes, reducing total job completion time for large datasets.

1.4 Literature Context and Industry Relevance

Hadoop and Spark are distributed computing frameworks that power Big Data analytics in a variety of industries, including finance, healthcare, and transportation. Hadoop's MapReduce paradigm has proven reliable in large-scale batch processing (Dean and Ghemawat, 2008). Spark extends these capabilities by introducing in-memory computations that significantly speed up iterative or interactive workloads (Apache Spark Official Documentation, 2025).

By applying these technologies to road safety data, the project showcases how **Big Data** solutions can provide critical insights—such as identifying trends in casualty severity or highlighting high-risk areas—to guide policymaking and resource allocation (Department for Transport, 2020). This approach reflects the broader shift in industry towards data-driven strategies for public health and safety.

---

2. Tasks & Implementation

2.1 Overview

This section describes the process of installing, configuring, and validating a three-node Hadoop cluster (machine01, machine02, machine03) used for processing 2018 road safety data. I utilised Java, Hadoop, and Spark, and implemented a MapReduce program to aggregate casualty severity counts. Each step is documented with code snippets and is supported by corresponding screenshots (referenced as Figures 2.1 to 2.20) in the appendix. A flow chart is included at the end of this section to illustrate the overall data processing workflow. (see appendix C)

2.2 Installation of Java, Hadoop, and Spark

2.2.1 Java Installation & Configuration

1. **System Update & Pre-requisites**
   To ensure a clean and updated environment, I first updated the system's package list and installed necessary utilities:

   ```
   sudo apt-get update
   sudo apt-get install wget vim openssh-server net-tools rsync
   ```

   (see Figure 2.1)
   This step guarantees that tools required for downloading, editing, and managing SSH connections are installed.

2. **Download & Extract Java (JDK 1.8.0_251)**
   I downloaded the JDK 1.8.0_251 and placed it in `/usr/lib/jvm/`. I then extracted the contents:

   ```
   tar -xvf /home/n1138321/Downloads/jdk-8u251-linux-x64.tar.gz -C
   /usr/lib/jvm/
   ```

   (see Figure 2.2)
   Using this version ensures compatibility with Hadoop's Java-based applications.

3. **Setting Environment Variables**
   I updated the `~/.bashrc` file to define `JAVA_HOME` and update the system `PATH`:

   ```
   export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
   export PATH=$PATH:$JAVA_HOME/bin
   ```

   After saving the file, I applied the changes with:

   ```
   source ~/.bashrc
   ```

   (see Figure 2.3)
   These variables are critical since both Hadoop and Spark rely on Java for execution.

4. **Verification**
   I confirmed the installation by running:

   ```
   java -version
   ```

   (see Figure 2.4)
   The terminal output confirmed that JDK 1.8.0_251 is correctly installed, ensuring a stable base for further installations.

2.2.2 Hadoop Installation & Configuration

1. **Download & Extract Hadoop (2.9.0)**
   I downloaded Hadoop 2.9.0 from the labs, placed it in `/usr/local/`, and extracted it:

   ```
   tar -xvf /home/n1138321/Downloads/hadoop-2.9.0.tar.gz -C /usr/local/
   cd /usr/local
   sudo ln -s hadoop-2.9.0 hadoop
   ```

   (see Figure 2.5)
   Creating a symbolic link simplifies version management and configuration consistency.

2. **Defining Hadoop Environment Variables**
   I edited `~/.bashrc` to add:

   ```
   export HADOOP_HOME=/usr/local/hadoop
   export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
   export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
   ```

   and then ran:

   ```
   source ~/.bashrc
   ```

   (see Figure 2.6)
   These settings allow the system to locate Hadoop executables and configuration files.

3. **Editing Configuration Files**
   We then configured the following files:
   - **core-site.xml:** Set `<name>fs.defaultFS</name>` to `hdfs://machine01:9000` to designate machine01 as the NameNode.
   - **hdfs-site.xml:** Defined `dfs.replication=3` for redundancy, `dfs.namenode.name.dir=/abc/name`, and `dfs.datanode.data.dir=/abc/data1`.
   - **mapred-site.xml:** Enabled YARN by setting `<name>mapreduce.framework.name</name>` to `yarn`.
   - **yarn-site.xml:** Configured `yarn.resourcemanager.hostname=machine01`. *(see Figure 2.7)*
     These configurations ensure data is distributed across the cluster and that job scheduling is managed by YARN.
4. **SSH Key Configuration**
   To enable seamless communication among nodes, I set up password less SSH:

   ```
   ssh-keygen -t rsa
   cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
   ```

```
ssh localhost
```

(see Figure 2.8)
This is vital for Hadoop daemons to interact without manual intervention.

2.2.3 Apache Spark Installation & Configuration

1. **Download & Extract Spark**
   I extracted Spark into `/usr/local/spark/`:

   ```
   tar -xvf /home/n1138321/Downloads/spark-<version>.tgz -C /usr/local/
   mv /usr/local/spark-<version> /usr/local/spark
   ```

   Spark is installed to demonstrate an alternative, faster in-memory processing option.

2. **Setting Spark Environment Variables**
   In `~/.bashrc`, I added:

   ```
   export SPARK_HOME=/usr/local/spark
   export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
   ```

   Then, I executed:

   ```
   source ~/.bashrc
   ```

   (see Figure 2.9)
   Defining these variables is necessary for invoking Spark commands.

3. **Verification**
   Finally, I verified the installation by running:

   ```
   spark-shell --version
   ```

   (see Figure 2.10)
   This confirms that Spark is properly installed and ready for use, although our primary processing is done via MapReduce.

2.3 Multi-Node Cluster Configuration

2.3.1 Overview of Node Roles and Architecture

Our cluster comprises:

- **machine01:** Serves as the **NameNode** and **ResourceManager**, managing HDFS metadata and job scheduling.
- **machine02 and machine03:** Act as **DataNodes** and **NodeManagers**, storing data and executing tasks.

2.3.2 Cloning and Configuring Nodes

1. **Cloning Machine01 to Machine02 & Machine03**
   I cloned machine01 twice and renamed them accordingly. I then updated `/etc/hostname` and `/etc/hosts` on each node:

   ```
   sudo nano /etc/hostname
   sudo nano /etc/hosts
   ```

   (see Figure 2.11)
   This ensures proper network identification and communication across the cluster.

2. **Configuring HDFS for Each Node**
   - **machine02:** Edited `hdfs-site.xml` to change `dfs.datanode.data.dir` to `/abc/data2`. *(see Figure 2.12)*
   - **machine03:** Edited `hdfs-site.xml` to change `dfs.datanode.data.dir` to `/abc/data3`.
     Distributing HDFS directories helps achieve fault tolerance and load balancing.
3. **Password less SSH Across Nodes**
   On machine01, I executed:

   ```
   ssh-copy-id -i ~/.ssh/id_rsa.pub n1138321@machine02
   ssh-copy-id -i ~/.ssh/id_rsa.pub n1138321@machine03
   ```

   (see Figure 2.13)
   This step is crucial for automating tasks across the nodes.

4. **Finalising the Slaves File & Starting Services**
   I updated `/usr/local/hadoop/etc/hadoop/slaves` on machine01 with the hostnames of machine02 and machine03. I then formatted HDFS and started the Hadoop services:

   ```
   hdfs namenode -format
   start-all.sh
   jps
   ```

   (see Figure 2.14)
   This confirms that critical daemons (NameNode, DataNode, ResourceManager, etc.) are running and that the cluster is operational.

2.4 Implementing the MapReduce Program

1. **Developing the Application**
   I implemented a MapReduce job comprising three Java classes:
   - **Mapper:** Processes each CSV record, skips the header, and emits `<severity, 1>`. *(see Figure 2.15)*
   - **Reducer:** Aggregates counts for each severity level.
   - **Driver:** Configures the job, specifying input/output paths and linking the Mapper and Reducer. *(see appendix B)*
     This modular design ensures clarity and ease of maintenance.
2. **Compilation & Packaging**
   In Eclipse, I configured the build path to include the necessary Hadoop libraries. I then exported the project as a JAR file (`RoadSafetyCount.jar`). *(see Figure 2.16)*
   Packaging the application as a JAR facilitates deployment on the Hadoop cluster.
3. **Data Upload to HDFS**
   I created an HDFS directory and uploaded the CSV file:

   ```
   hadoop fs -mkdir -p /user/n1138321/RoadSafety
   hadoop fs -put
   /home/n1138321/Downloads/dftRoadSafetyData_Casualties_2018.csv
   /user/n1138321/RoadSafety
   ```

   (see Figure 2.17)
   This step ensures that the input data is available across the cluster for processing.

4. **Executing the MapReduce Job**
   The job was executed using the following command:

   ```
   hadoop jar /home/n1138321/RoadSafetyCount.jar
     /user/n1138321/RoadSafety/dftRoadSafetyData_Casualties_2018.csv \
     /user/n1138321/RoadSafety/Result1
   ```

   (see Figure 2.18)
   This command launches the MapReduce job, with YARN handling resource allocation and job scheduling.

5. **Monitoring & Verifying Output**
   - **Resource Manager UI:** I monitored job progress at `http://machine01:8088` to confirm successful execution. *(see Figure 2.19)*
   - **Output Verification:** The final output was verified by viewing the aggregated results:

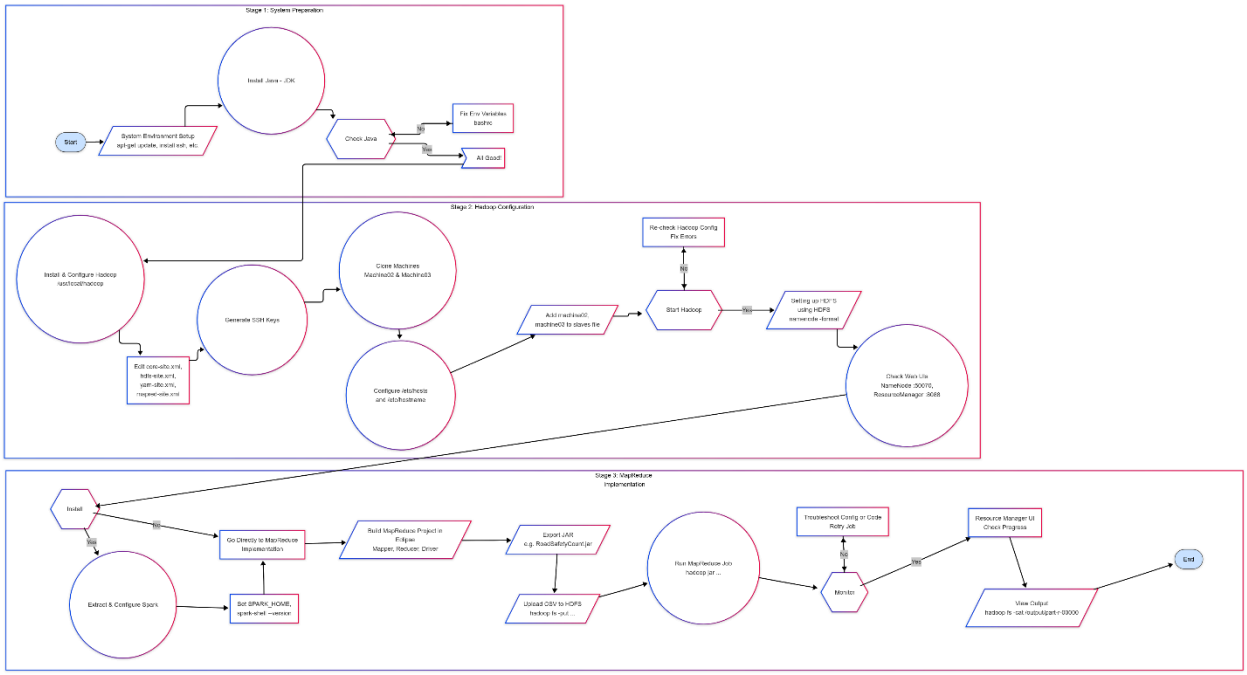
     ```
     hadoop fs -cat /user/n1138321/RoadSafety/Result1/part-r-00000
     ```

     (see Figure 2.20)
     This confirms that the MapReduce job correctly processed and aggregated the data.

2.5 Observations & Lessons Learned

- **Configuration Challenges:**
  I encountered minor SSH key issues on machine02, which were resolved by re-generating and re-copying the keys. Such troubleshooting deepened our understanding of inter-node communication.
- **Performance Considerations:**
  The default block size and a replication factor of 3 provided adequate redundancy and fault tolerance for our dataset. Future work may involve tuning memory allocations and block sizes to optimise performance for larger datasets.
- **Future Enhancements:**
  Although our current solution uses MapReduce, integrating **Spark** for in-memory processing could significantly accelerate iterative analyses. Additionally, containerising the cluster using Docker may simplify future deployments and scalability.

  (see figure 2.21)



---

3. Critique, Discussion, and Summary

3.1 Evaluation of the Implementation

Successes
The implementation of the multi-node Hadoop cluster demonstrated a clear and effective approach to distributed data processing. By installing and configuring Java, Hadoop, and Spark across three virtual machines (machine01, machine02, machine03), I achieved a functional environment capable of

parallelizing the analysis of large datasets. This setup aligns with industry standards, where multi-node clusters are used to ensure fault tolerance and scalability.

A key success was the ability to seamlessly integrate the MapReduce job, which processed the 2018 road safety dataset without significant runtime errors. The **Mapper** and **Reducer** classes were straightforward to develop, thanks to the clarity of Hadoop's MapReduce APIs. Additionally, the Hadoop Resource Manager and NameNode UIs provided transparent monitoring of job progress, validating the effectiveness of the cluster setup. The replication factor of three proved robust, ensuring data redundancy across all nodes.

The results indicate that Severity 3—representing the least severe category—has the highest count, followed by Severity 2, and finally Severity 1 (the most severe). This distribution suggests that while serious or fatal accidents occur less frequently, the majority of recorded incidents result in relatively minor injuries. Consequently, road safety interventions should continue to target severe accidents, but also acknowledge the larger volume of lower-severity incidents that cumulatively have a significant impact on public health and resource allocation.

The successful incorporation of **Spark**—even if not central to the main MapReduce task—further demonstrates a forward-looking mindset, acknowledging that in-memory computing can accelerate iterative analyses or real-time streaming tasks. This addition reflects current industry practices, where hybrid environments combine both MapReduce and Spark for different workloads.

Issues & Limitations
Despite the overall success, several minor challenges arose during the setup and execution phases. One notable issue was the initial configuration of **SSH keys** across nodes, which occasionally led to password prompts that disrupted automated Hadoop operations. Re-generating and re-copying the public keys resolved these inconsistencies.

Another limitation was the **manual configuration** of each XML file (e.g., core-site.xml, hdfs-site.xml, yarn-site.xml). Although this process provided in-depth understanding, it was time-consuming and prone to typographical errors. A small misconfiguration (for example, pointing the DataNode directory to a non-existent path) could halt the cluster startup.

In terms of **performance**, the default block size and replication settings sufficed for our demonstration dataset. However, with much larger or more complex data, the current configuration might face bottlenecks. Tuning parameters such as **YARN memory allocations**, **block size**, and **number of reducers** could be necessary to optimise job throughput.

Finally, while **Spark** was installed, I did not fully test its in-memory capabilities for more advanced analytics or iterative tasks. This leaves potential performance gains unexplored within the scope of this project.

3.2 Suggestions for Improvement

1. **Automated Deployment and Configuration:**
   To reduce manual editing and potential errors, tools like **Ansible**, **Puppet**, or **Chef** could automate the cluster configuration. This would also expedite future expansions or redeployments of the environment.
2. **Performance Tuning:**
   - **Block Size and Replication Factor:** Experimenting with larger block sizes may reduce overhead for bigger datasets, while adjusting replication to two or four could optimise performance or fault tolerance, depending on system constraints.
   - **Resource Allocation:** Fine-tuning **YARN** memory and CPU settings can significantly improve MapReduce and Spark job runtimes.
3. **Monitoring and Alerting:**
   Introducing tools like **Ambari**, **Cloudera Manager**, or **Grafana** can provide real-time dashboards and alerts for cluster resource usage, data node health, and job performance. Such monitoring fosters proactive management of potential issues.
4. **Deeper Spark Integration:**
   Although Spark was installed, running a comparative analysis between MapReduce and Spark on the same dataset would highlight the advantages of in-memory processing, particularly for iterative or interactive queries. This comparison could guide decisions on the best tool for specific workloads.
5. **Containerisation and Cloud Deployment:**
   Packaging the entire cluster setup using **Docker** or deploying it on a cloud platform (e.g., AWS, Azure) could simplify scalability and resource management. This approach also aligns with modern DevOps practices, making it easier to replicate the environment in different contexts. (Amazon Web Services, 2023).

3.3 Summary & Conclusions

The primary objective—setting up a multi-node Hadoop cluster—was successfully achieved. The **MapReduce** framework effectively counted accident occurrences by casualty severity, validating both the cluster configuration and the correctness of the data processing pipeline. The **observed successes** include a reliable distributed environment, seamless data ingestion into HDFS, and clear job tracking through Hadoop's web interfaces. Minor **challenges**, such as SSH key inconsistencies and manual XML configurations, emphasised the need for careful attention to detail in distributed systems.

From a broader perspective, this project aligns with industry trends in **Big Data** and **distributed database engineering**. Organisations commonly employ Hadoop for large-scale batch processing, benefiting from its built-in fault tolerance and scalable design. The optional Spark installation reflects ongoing industry shifts toward faster, in-memory analytics. Future enhancements—like performance tuning, monitoring, or containerisation—would further align this setup with real-world enterprise solutions.

Overall, this implementation has demonstrated not only the feasibility of processing substantial datasets in a distributed manner but also the importance of well-organised configurations, robust monitoring, and the potential for hybrid data processing approaches. By addressing the suggested improvements, this system can evolve into a more resilient and high-performing platform for advanced data analytics tasks in both academic and commercial environments.

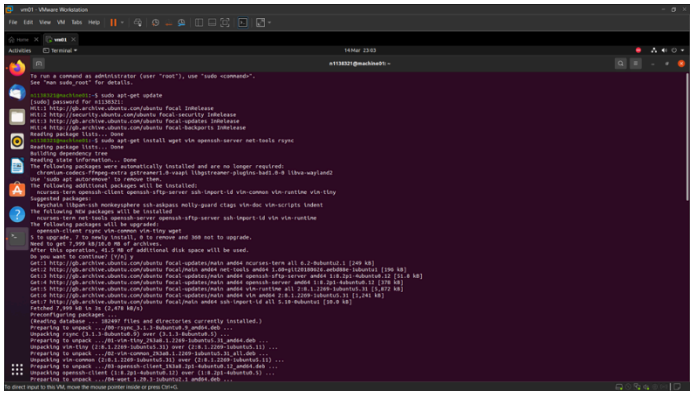
4. References

- Apache Hadoop Official Documentation (2025) Apache Hadoop. Available at: https://hadoop.apache.org/.
- Apache Spark Official Documentation (2025) Apache Spark. Available at: https://spark.apache.org/.
- Dean, J. and Ghemawat, S. (2008) 'MapReduce: Simplified Data Processing on Large Clusters', Communications of the ACM.
- Department for Transport (2020) *Reported Road Casualties in Great Britain: 2018 Annual Report.* London: Department for Transport. Available at: https://www.gov.uk/government/statistics/reported-road-casualties-great-britain-annual-report-2018.
- Amazon Web Services (2023) *What is Cloud Computing?* Available at: https://aws.amazon.com/what-is-cloud-computing/.

5. Appendices

- Appendix A: *Screenshots of configuration files and terminal outputs.*

  - ***Figure 2.1:*** *System update & installing pre-requisites.*



  - ***Figure 2.2:*** *Extracting the JDK into /usr/lib/jvm/.*

```
n1138321@machine01:/usr/lib/jvm$ ls -la
total 12
drwxr-xr-x   3 root  root  4096 Mar 14 23:27 .
drwxr-xr-x 115 root  root  4096 Mar 15 16:49 ..
drwxr-xr-x   7 10143 10143 4096 Mar 12  2020 jdk1.8.0_251
```

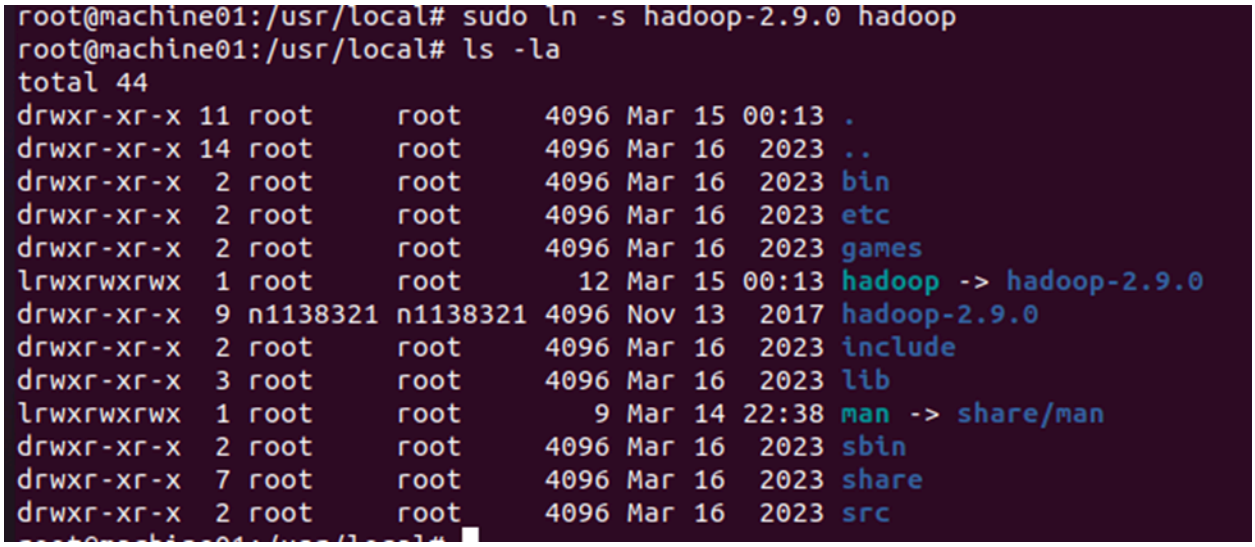
  - ***Figure 2.3:*** *Excerpt from the ~/.bashrc file, highlighting the newly added environment variables (JAVA_HOME and updated PATH) for JDK 1.8.0_251.*

```
fi

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export PATH=$PATH:$JAVA_HOME/bin
```

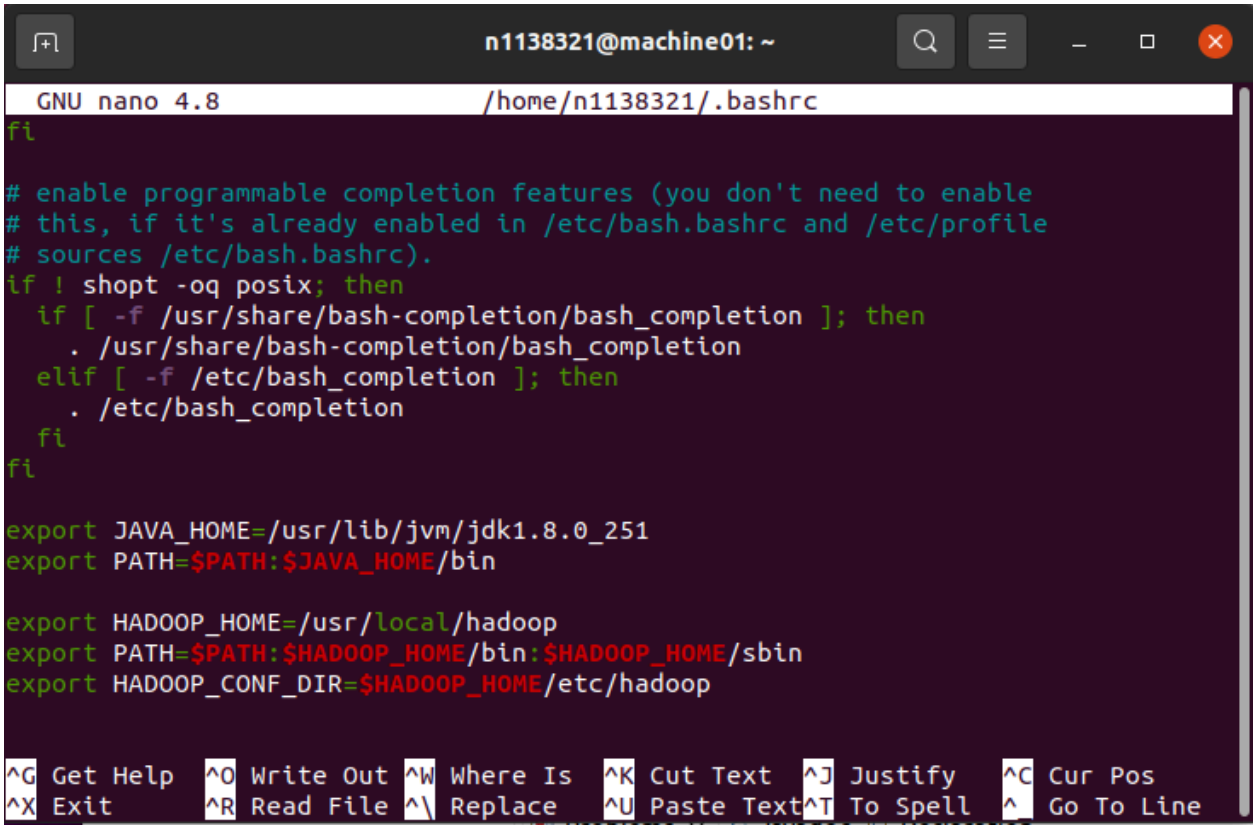
  - ***Figure 2.4:*** *java -version output showing JDK 1.8.0_251.*

```
n1138321@machine01:~$ source .bashrc
n1138321@machine01:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
```

- ***Figure 2.5:*** *Hadoop 2.9.0 extraction and symbolic link in /usr/local/.*



- ***Figure 2.6:*** *~/.bashrc updated with Hadoop environment variables*

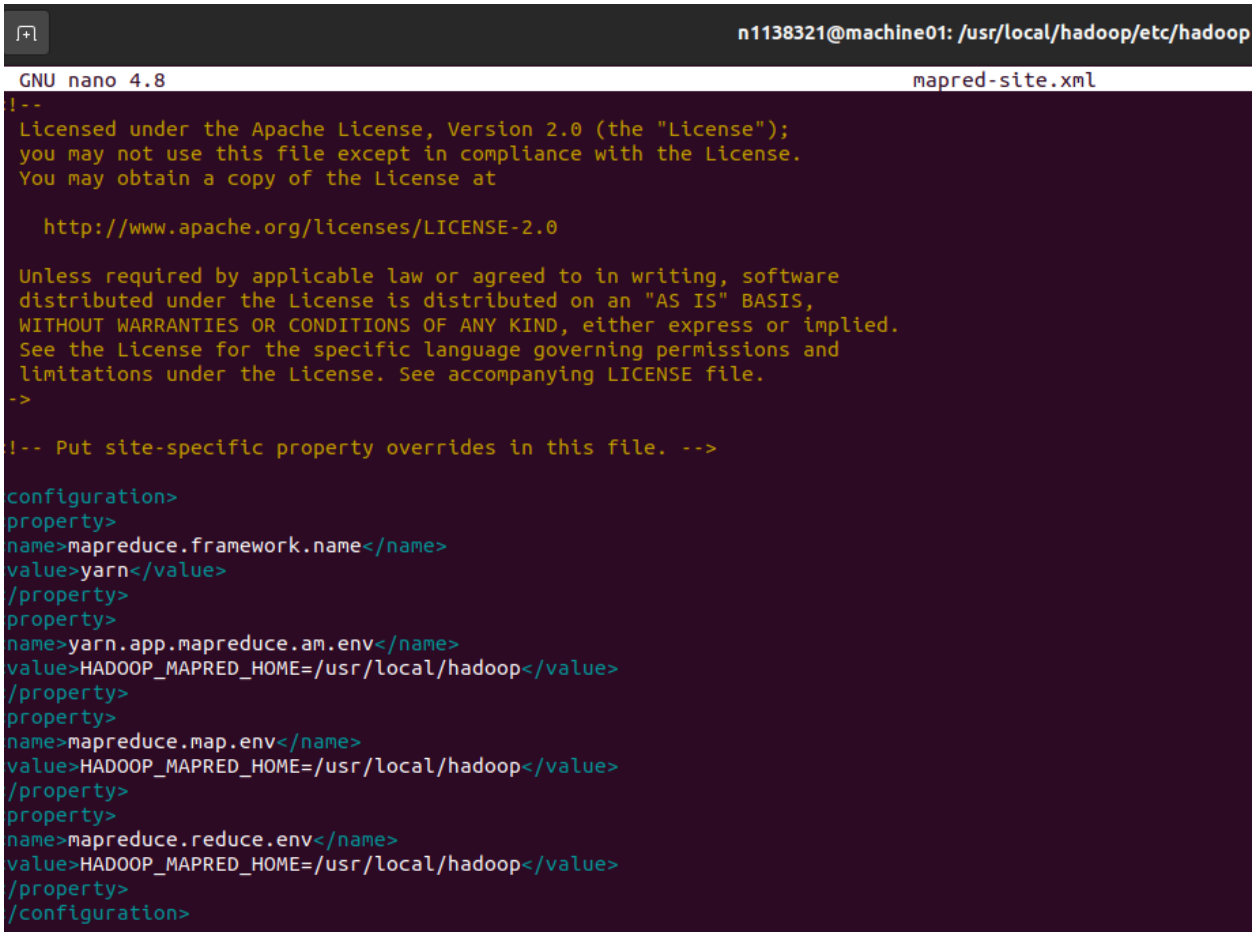


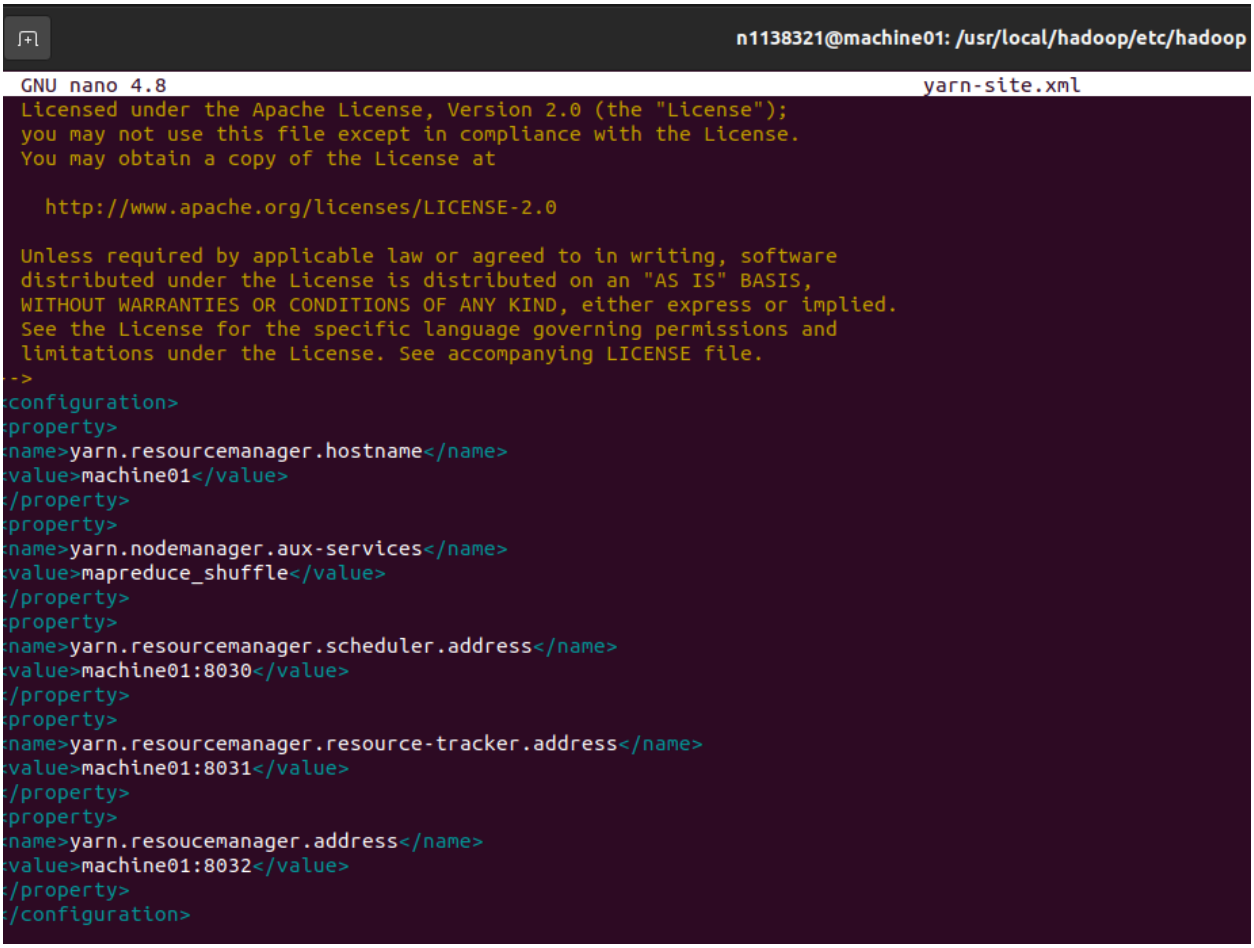
.

- ***Figure 2.7:*** *Snippets of* ***core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml***

```
 GNU nano 4.8          /usr/local/hadoop/etc/hadoop/core-site.xml

   http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://machine01:9000</value>
</property>
</configuration>
```

```
n1138321@machine01: /usr/local/hadoop/etc/hadoop
 GNU nano 4.8                                          hdfs-site.xml
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>dfs.webhdfs.enabled</name>
<value>true</value>
</property>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>/abc/name</value>
<final>true</final>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/abc/data1</value>
<final>true</final>
</property>
<property>
<name>dfs.namenode.http-address</name>
<value>machine01:50070</value>
</property>
<property>
<name>dfs.namenode.secondary.http-address</name>
<value>machine02:50090</value>
</property>
</configuration>
```


n1138321@machine01: /usr/local/hadoop/etc/hadoop
GNU nano 4.8
mapred-site.xml
!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
->
!-- Put site-specific property overrides in this file. -->
configuration>
property>
name>mapreduce.framework.name</name>
value>yarn</value>
/property>
property>
name>yarn.app.mapreduce.am.env</name>
value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
/property>
property>
name>mapreduce.map.env</name>
value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
/property>
property>
name>mapreduce.reduce.env</name>
value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
/property>
/configuration>


n1138321@machine01: /usr/local/hadoop/etc/hadoop
GNU nano 4.8
yarn-site.xml
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<configuration>
<property>
<name>yarn.resourcemanager.hostname</name>
<value>machine01</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>machine01:8030</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>machine01:8031</value>
</property>
<property>
<name>yarn.resoucemanager.address</name>
<value>machine01:8032</value>
</property>
</configuration>

- ***Figure 2.8:*** *SSH key generation and password less login confirmation.*

```
n1138321@machine01:~$ ssh localhost
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-134-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Sat Mar 15 00:50:28 2025 from 127.0.0.1
```

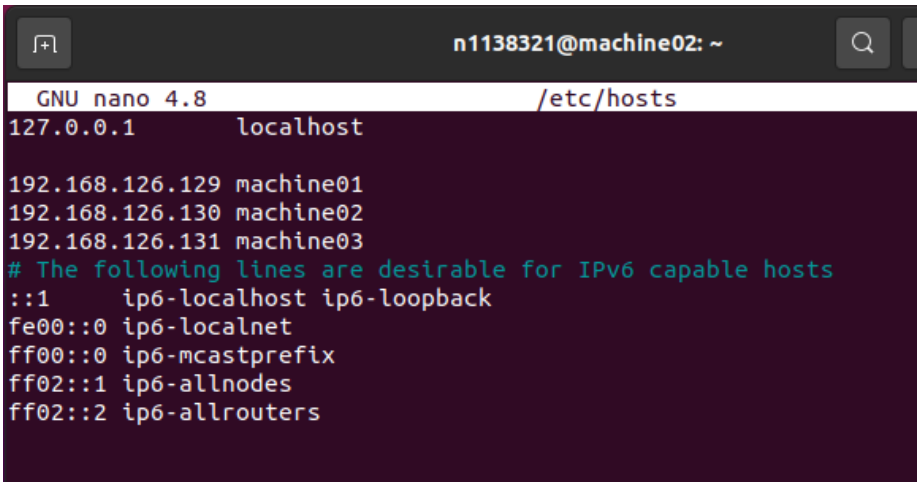
- ***Figure 2.9:*** *Spark extraction and folder structure in /usr/local/spark/.*

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_251
export PATH=$PATH:$JAVA_HOME/bin

export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop

export SPARK_HOME=/usr/local/spark
export PATH=$PATH:$SPARK_HOME/bin
export PATH=$PATH:$SPARK_HOME/sbin
```

- ***Figure 2.10:*** *spark-shell --version verifying Spark installation.*

```
root@machine01:/usr/local# ls -all
total 48
drwxr-xr-x 12 root     root     4096 Mar 18 19:00 .
drwxr-xr-x 14 root     root     4096 Mar 16  2023 ..
drwxr-xr-x  2 root     root     4096 Mar 16  2023 bin
drwxr-xr-x  2 root     root     4096 Mar 16  2023 etc
drwxr-xr-x  2 root     root     4096 Mar 16  2023 games
lrwxrwxrwx  1 n1138321 n1138321   12 Mar 15 00:13 hadoop -> hadoop-2.9.0
drwxr-xr-x 10 n1138321 n1138321 4096 Mar 15 01:05 hadoop-2.9.0
```

- ***Figure 2.11**: Updated /etc/hosts files on machine02.*



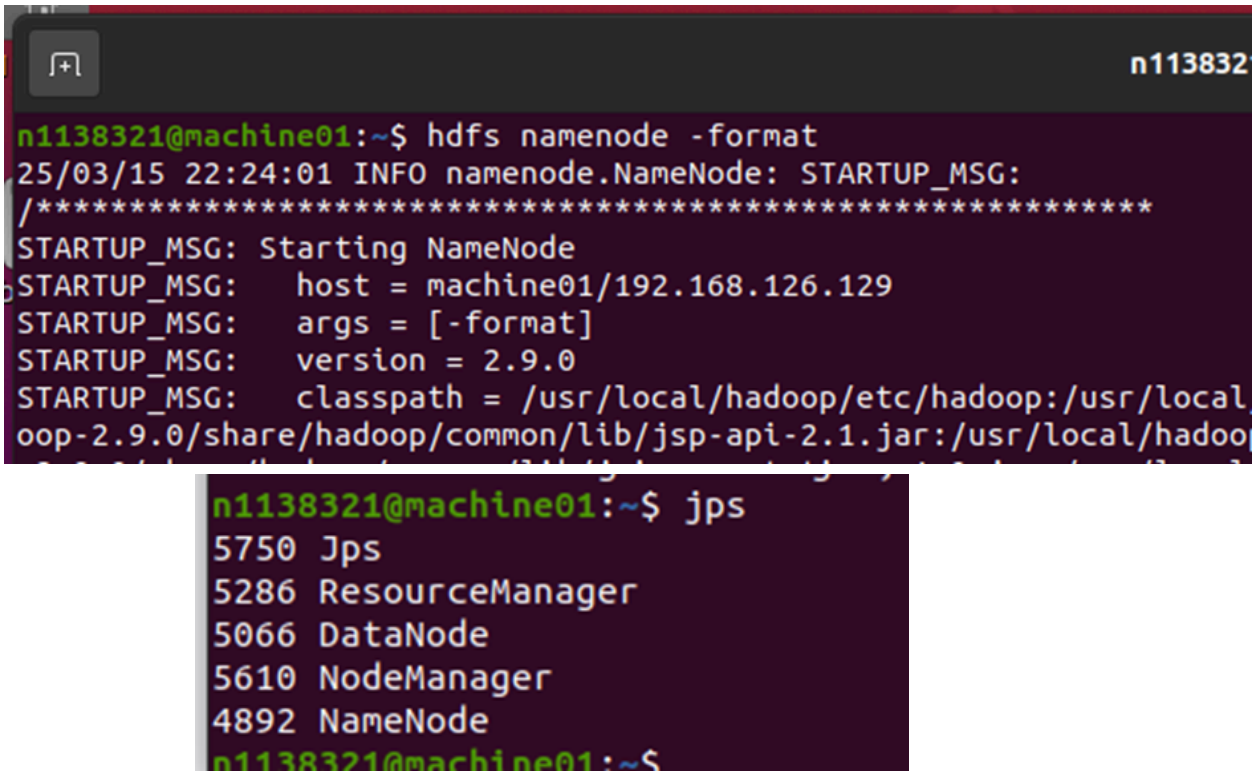
- ***Figure 2.12:*** *hdfs-site.xml modifications for machine02.*

```
<configuration>
<property>
<name>dfs.webhdfs.enabled</name>
<value>true</value>
</property>
<property>
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>/abc/data2</value>
<final>true</final>
</property>
<property>
<name>dfs.namenode.http-address</name>
<value>machine01:50070</value>
</property>
<property>
<name>dfs.namenode.secondary.http-address</name>
<value>machine02:50090</value>
</property>
<property>
<name>dfs.namenode.checkpoint.period</name>
<value>600</value>
</property>

</configuration>
```

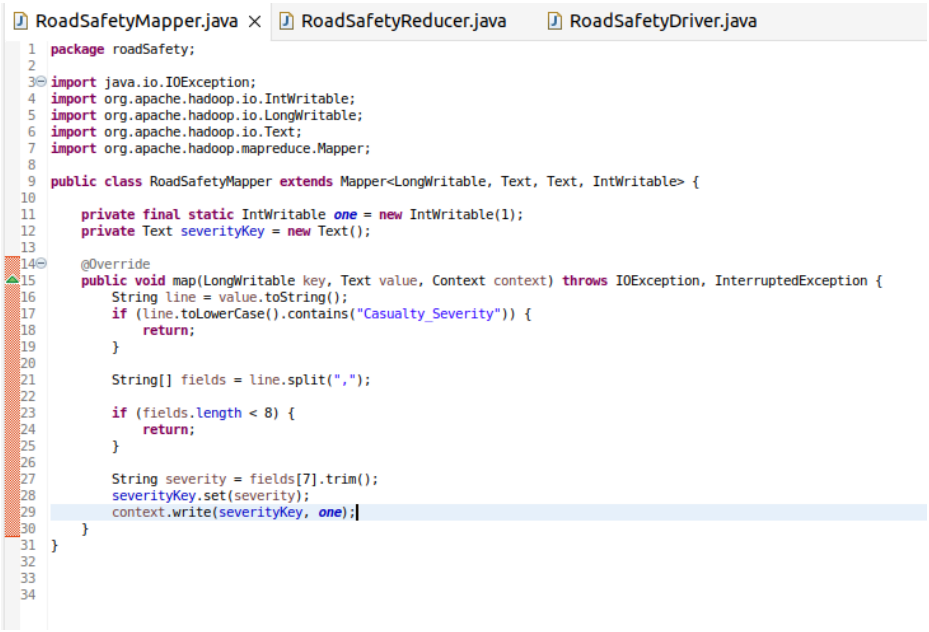
- ***Figure 2.13:*** *Terminal output from machine01 after executing the ssh-copy-id command to set up passwordless SSH. The output confirms that the public SSH key is already installed on machine02 and machine03, indicating that passwordless login is successfully configured.*

```
n1138321@machine01:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub n1138321@machine02
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/n1138321/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed

/usr/bin/ssh-copy-id: WARNING: All keys were skipped because they already exist on the remote system.
                (if you think this is a mistake, you may want to use -f option)

n1138321@machine01:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub n1138321@machine03
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/n1138321/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed

/usr/bin/ssh-copy-id: WARNING: All keys were skipped because they already exist on the remote system.
                (if you think this is a mistake, you may want to use -f option)
```

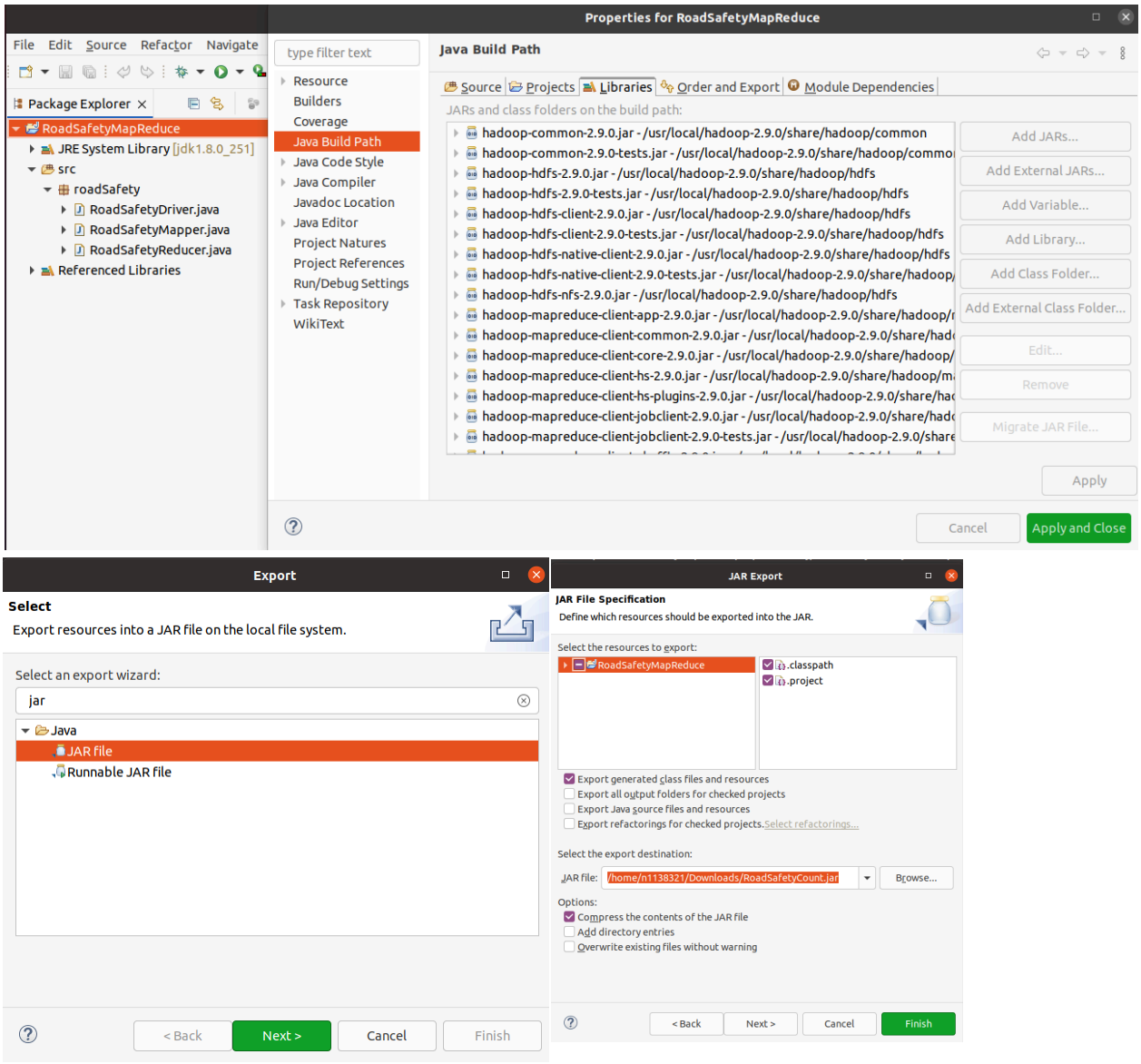
- ***Figure 2.14:*** *Terminal output of hdfs namenode -format, and jps showing running services.*



- ***Figure 2.15:*** *Mapper code snippet in Eclipse.*



- ***Figure 2.16:*** *Eclipse build path configuration and export to JAR.*



- ***Figure 2.17:*** *HDFS directory listing showing the CSV file.*

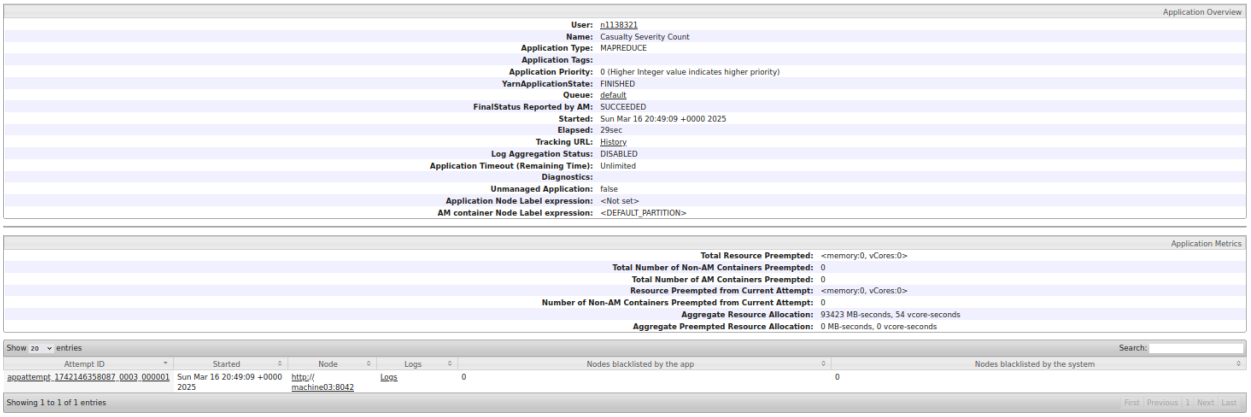
```
n1138321@machine01:~$ hadoop fs -ls /user/n1138321/RoadSafety
Found 1 items
-rw-r--r--   3 n1138321 supergroup    7466192 2025-03-16 18:12 /user/n1138321/R
oadSafety/dftRoadSafetyData_Casualties_2018.csv
```

- ***Figure 2.18:*** *Terminal output of the MapReduce job execution.*

```
n1138321@machine01:~$ hadoop jar /home/n1138321/Downloads/RoadSafetyCount.jar /user/n1138321/RoadSafety/dftRoadSafetyData_Casualties_2018.csv /user/n1138321/RoadSafety/Result1
25/03/16 20:49:06 INFO client.RMProxy: Connecting to ResourceManager at machine01/192.168.126.129:8032
25/03/16 20:49:07 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
25/03/16 20:49:08 INFO input.FileInputFormat: Total input files to process : 1
25/03/16 20:49:08 INFO mapreduce.JobSubmitter: number of splits:1
25/03/16 20:49:08 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
25/03/16 20:49:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1742146358087_0003
25/03/16 20:49:09 INFO impl.YarnClientImpl: Submitted application application_1742146358087_0003
25/03/16 20:49:09 INFO mapreduce.Job: The url to track the job: http://machine01:8088/proxy/application_1742146358087_0003/
25/03/16 20:49:09 INFO mapreduce.Job: Running job: job_1742146358087_0003
25/03/16 20:49:17 INFO mapreduce.Job: Job job_1742146358087_0003 running in uber mode : false
25/03/16 20:49:17 INFO mapreduce.Job:  map 0% reduce 0%
25/03/16 20:49:26 INFO mapreduce.Job:  map 100% reduce 0%
25/03/16 20:49:40 INFO mapreduce.Job:  map 100% reduce 100%
25/03/16 20:49:40 INFO mapreduce.Job: Job job_1742146358087_0003 completed successfully
25/03/16 20:49:40 INFO mapreduce.Job: Counters: 49
```

```
File System Counters
        FILE: Number of bytes read=54
        FILE: Number of bytes written=404427
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=7466341
        HDFS: Number of bytes written=44
```

- ***Figure 2.19:*** *Resource Manager UI at machine01:8088 displaying the running job.*



Application Overview

User:	n1138321
Name:	Casualty Severity Count
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Sun Mar 16 20:49:09 +0000 2025
Elapsed:	29sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

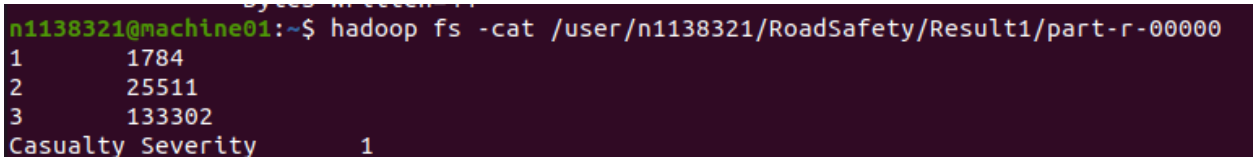
Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	93423 MB-seconds, 54 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Show 20 entries — Search:

Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by the system
appattempt_1742146358087_0003_000001	Sun Mar 16 20:49:09 +0000 2025	http://machine03:8042	Logs	0	0

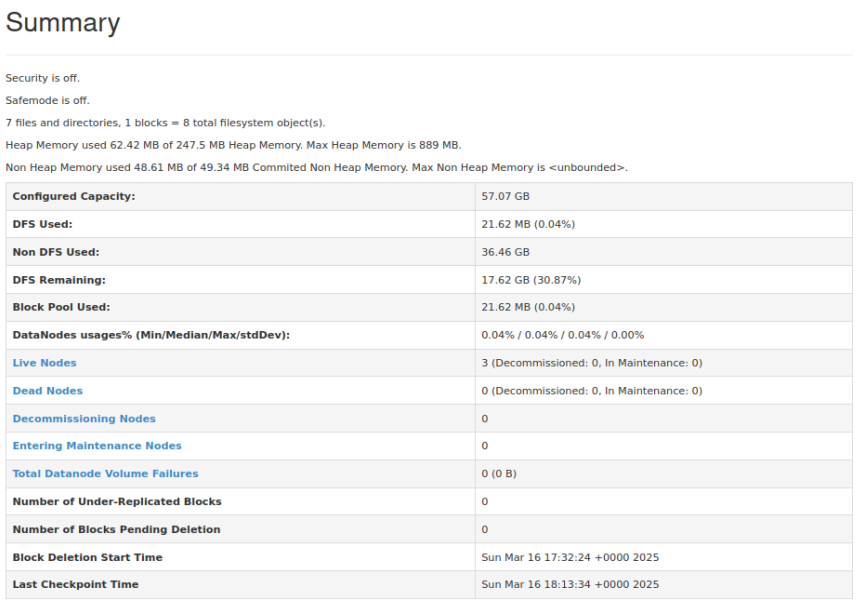
Showing 1 to 1 of 1 entries — First Previous 1 Next Last

- ***Figure 2.20:*** *Contents of part-r-00000 with casualty severity counts.*



```
n1138321@machine01:~$ hadoop fs -cat /user/n1138321/RoadSafety/Result1/part-r-00000
1       1784
2       25511
3       133302
Casualty Severity       1
```

- ***Figure 2.21***: *Hadoop and Spark web interfaces. The NameNode and ResourceManager dashboards confirm cluster health and MapReduce execution, while the Spark Master page shows active workers, memory usage, and running applications.*



Summary

Security is off.

Safemode is off.

7 files and directories, 1 blocks = 8 total filesystem object(s).

Heap Memory used 62.42 MB of 247.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 48.61 MB of 49.34 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	57.07 GB
DFS Used:	21.62 MB (0.04%)
Non DFS Used:	36.46 GB
DFS Remaining:	17.62 GB (30.87%)
Block Pool Used:	21.62 MB (0.04%)
DataNodes usages% (Min/Median/Max/stdDev):	0.04% / 0.04% / 0.04% / 0.00%
Live Nodes	3 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	Sun Mar 16 17:32:24 +0000 2025
Last Checkpoint Time	Sun Mar 16 18:13:34 +0000 2025

Application Overview

User:	n1138321
Name:	Casualty Severity Count
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Sun Mar 16 20:49:09 +0000 2025
Elapsed:	29sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

Application Metrics

Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	93423 MB-seconds, 54 vcore-seconds
Aggregate Preempted Resource Allocation:	0 MB-seconds, 0 vcore-seconds

Show 20 entries Search:

Attempt ID	Started	Node	Logs	Nodes blacklisted by the app	Nodes blacklisted by the system
appattempt_1742146358087_0003_000001	Sun Mar 16 20:49:09 +0000 2025	http://machine03:8042	Logs	0	0

Showing 1 to 1 of 1 entries First Previous 1 Next Last

Logged in as: dr.who

Nodes of the cluster

- Cluster
  - About
  - Nodes
  - Node Labels
  - Applications
    - NEW
    - NEW_SAVING
    - SUBMITTED
    - ACCEPTED
    - RUNNING
    - FINISHED
    - FAILED
    - KILLED
  - Scheduler
- Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
0	0	0	0	0	0 B	24 GB	0 B	0	24	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
3	0	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

Node Labels	Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
	/default-rack	RUNNING	machine02:46035	machine02:8042	Sat Mar 15 23:16:34 +0000 2025		0	0 B	8 GB	0	8	2.9.0
	/default-rack	RUNNING	machine01:46431	machine01:8042	Sat Mar 15 23:16:35 +0000 2025		0	0 B	8 GB	0	8	2.9.0
	/default-rack	RUNNING	machine03:33283	machine03:8042	Sat Mar 15 23:16:34 +0000 2025		0	0 B	8 GB	0	8	2.9.0

Showing 1 to 3 of 3 entries First Previous 1 Next Last

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
3	0	0	3	0	0 B	24 GB	0 B	0	24	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
3	0	0	0	0	0	0

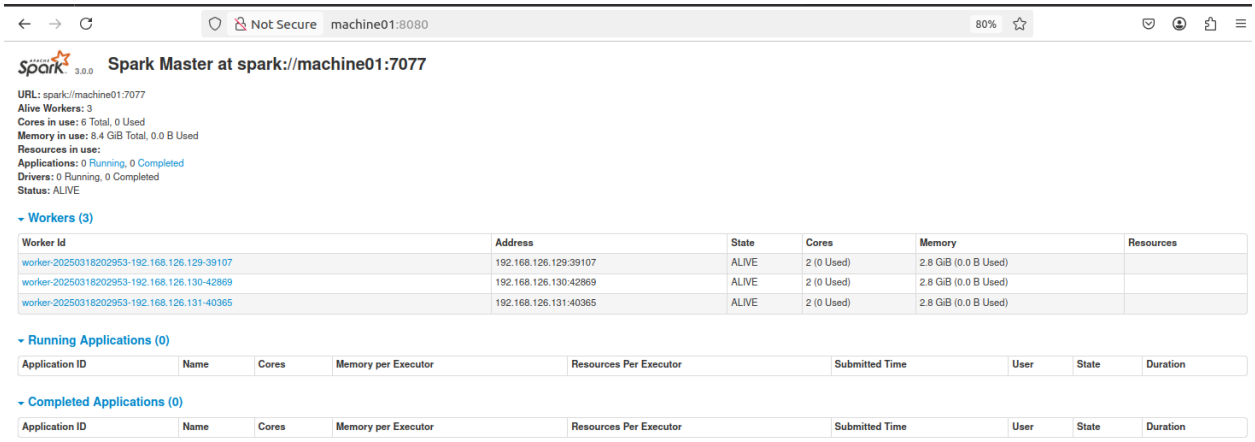
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	Application Priority	StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes
application_1742146358087_0003	n1138321	Casualty Severity Count	MAPREDUCE	default	0	Sun Mar 16 20:49:09 +0000 2025	Sun Mar 16 20:49:39 +0000 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1742146358087_0002	n1138321	Casualty Severity Count	MAPREDUCE	default	0	Sun Mar 16 18:32:18 +0000 2025	Sun Mar 16 18:32:33 +0000 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0
application_1742146358087_0001	n1138321	Casualty Severity Count	MAPREDUCE	default	0	Sun Mar 16 18:22:28 +0000 2025	Sun Mar 16 18:27:56 +0000 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0

Showing 1 to 3 of 3 entries First Previous 1 Next Last



- Appendix B: Full code listings (Mapper, Reducer, Driver classes).

RoadSafetyMapper.java

```java
package roadSafety;

import java.io.IOException;

public class RoadSafetyMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text severityKey = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        if (line.toLowerCase().contains("Casualty_Severity")) {
            return;
        }

        String[] fields = line.split(",");

        if (fields.length < 8) {
            return;
        }

        String severity = fields[7].trim();
        severityKey.set(severity);
        context.write(severityKey, one);
    }
}
```

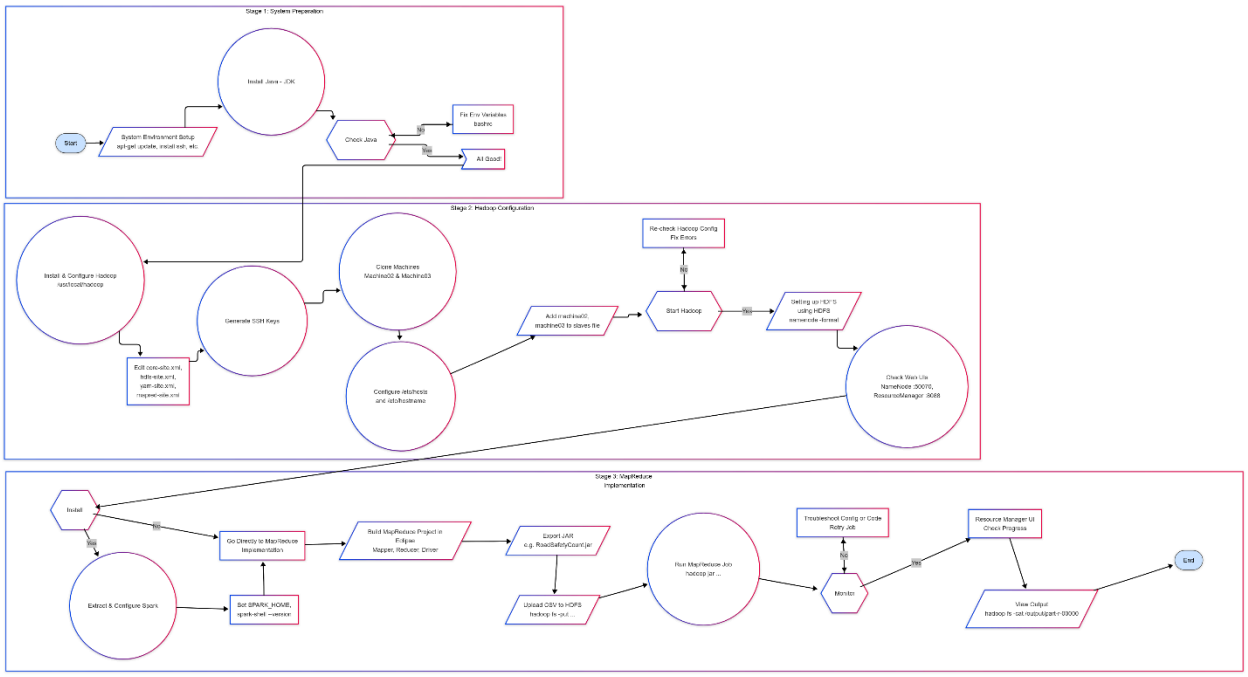
RoadSafetyReducer.java

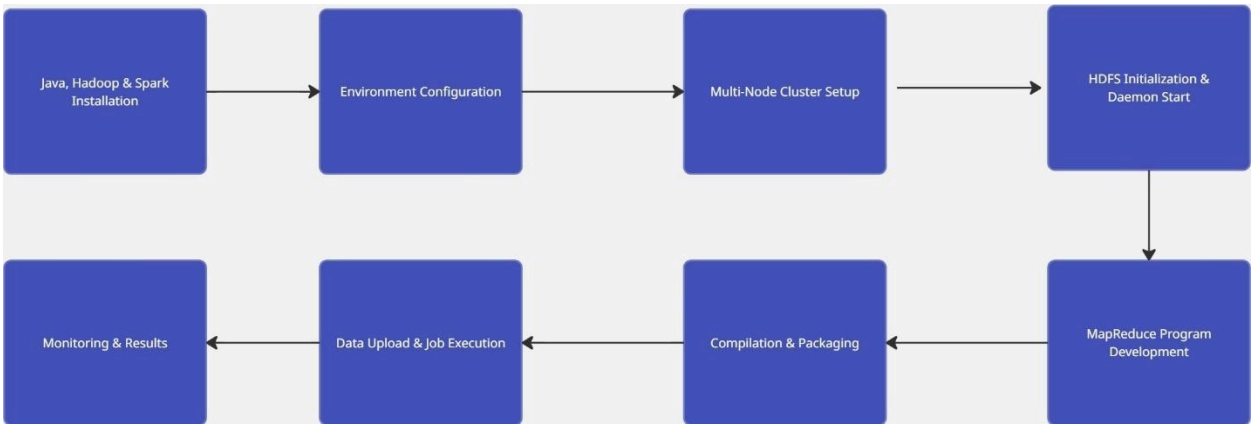
```java
package roadSafety;

import java.io.IOException;

public class RoadSafetyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

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

RoadSafetyMapper.java RoadSafetyReducer.java RoadSafetyDriver.java

```java
package roadSafety;

import org.apache.hadoop.conf.Configuration;

public class RoadSafetyDriver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: RoadSafetyDriver <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Casualty Severity Count");
        job.setJarByClass(RoadSafetyDriver.class);
        job.setMapperClass(RoadSafetyMapper.class);
        job.setCombinerClass(RoadSafetyReducer.class);
        job.setReducerClass(RoadSafetyReducer.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);
    }
}
```

- Appendix C:




Java, Hadoop & Spark Installation
Environment Configuration
Multi-Node Cluster Setup
HDFS Initialization & Daemon Start
MapReduce Program Development
Compilation & Packaging
Data Upload & Job Execution
Monitoring & Results