**High-Level Design**

1. **Project Overview**
   * 1. **Purpose**

The purpose of this document is to specify the high-level design for HDFS Name node Real Time Analytics. This document will act as an outline for implementation and discuss the design considerations.

* + 1. **Design Process**

The high-level design was selected by deciding what aspects of the system were most important and then building an architecture around them. Apache Hadoop 3.3.0 Hadoop cluster was setup with 3 nodes on Digital Ocean Cloud. 1 server where Name node and Secondary Name node runs and on 2 servers Data node runs. Python script to pull all required details from hdfs. The application is delivered as web application using the FLask web-framework. Two pages for time-based file operation metrics and user operation metrics on HDFS. Hadoop is being run as user “Hadoop” and required ssh keys are populated and added on remote Data Nodes for password-less login. Required metrics are collected by running python script which will communicate with hdfs cli.

1. **Requirements**

The overall requirement of this application are as follows

1. A web-based solution for gathering file operation details.
2. A web-based solution for gathering user operation details.
3. Open source apache Hadoop cluster with Name Node, Data Node.
4. **Proposed Solution**

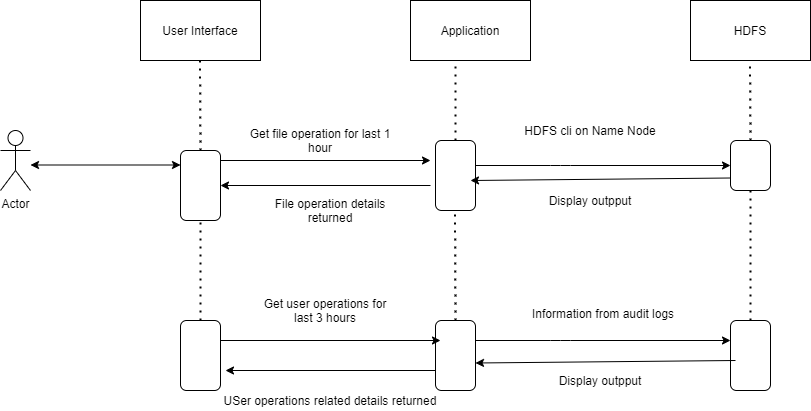
The proposed solution for the problem statement is obvious and straight forward implementation as per the requirement. Python Flask based web application which will be interacting with the hdfs cli in the backend and requested information will be displayed on the web-application. More enhancements can be made to the application by providing RBAC based authentication and different views as per mapped role. All necessary logging are enabled via log4j.properties and added in hadoop-env.sh.

1. **Capacity Planning**

Name node and Data node are installed with 8GB RAM 4 core vCPU with 155GB HDD. 1 server for Name node and 2 servers for Data node. Even though the utilization of memory or the CPU has never 30% of the maximum it has been designed to withstand considerable amount of load even for POC setup. Future enhancement includes adding additional Name nodes for fault tolerance. And the replication factor is set to 2 since we had 2 Data nodes for better fault-tolerance and high availability of data blocks.

JVM arguments are fine-tuned for performance optimization, i.e. Parallel GC threads are set to the number of cores available , Young generation set to be half the size of available Max heap so that number of world stop GC pause cycles can be reduced. More the young generation space allocated more would be the space for short lived objects. Use of Concurrent Mark Sweep GC improves performance because short pauses are more useful, and we do not want RPC calls from Data Nodes to be waiting for the Garbage collection which might increase the latency. Also, we have more Long lived data since files stored as an Inode Object may not be removed until a delete operation is performed.

1. **Sequence Diagram**



1. **Hardware and Platform Requirements**

**Operating System:**

Ubuntu 20.04

**Hadoop:**

Apache Hadoop 3.3.0

**Java:**

OpenJDK 11.0.11

**Packages Installed:**

Java open jdk

Java open jdk jre

Pdsh (to run multiple remote commands in parallel)

Python3

Flask

Werkzeug

Jinga2

Sqlite

Pandas

1. **System Connectivity**

Necessary firewall needs to be open for ports for communication between Name Node, Data Node. Python Flask application port needs to be allowed to be accessible from outside.

Hadoop Ports:

8040 RPC ports for communication between Data node and Name node.

9000 Port where name node is accessible

8088 Yarn Resource manager port