Apache Hadoop & Hive

Jason Chan, Sharon Nikritin, Roy Ming Feng Zhu, Chi Ho Li, Matthew Owen Tjhie

What is Apache Hadoop?

A data warehouse that allows programmers to work with data in HDFS using a query language called HiveQL, which is similar to SQL

- Allows users to leverage Hadoop MapReduce using a SQL interface, enabling analytics at a massive scale, in addition to distributed and fault-tolerant data warehousing

Apache Hive

What is Apache Hive?

Hive is an open-source data warehouse software, which is built atop the Apache Hadoop ecosystem enabling it to efficiently process extensive datasets. Within Hive, data organization is accomplished through tables, partitions, and buckets. The tables resemble the functionality of traditional RDBMS database tables, which facilitates operations like filtering and joining, providing a structured format for data and helping organize data into structured formats.

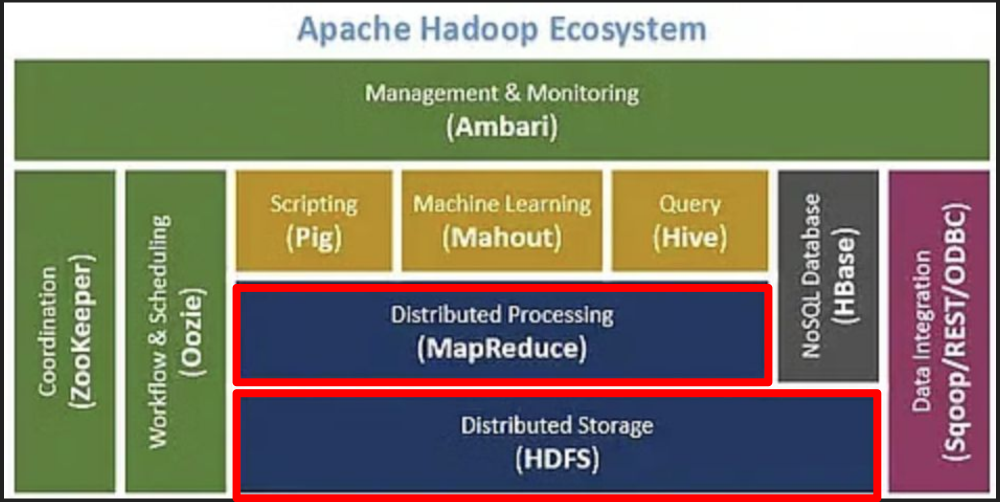
- Hive demonstrates scalability, allowing organizations to grow their data processing capabilities seamlessly
- The significance of Hive and Hadoop lies in their effectiveness in efficiently handling large data volumes, offering a proficient interface rooted in the familiarity of SQL.

Objective

The goal of this project is to explore the intricacies of Hadoop’s architecture, with an emphasis on its distributed file system and its capacity to manage extensive datasets. This project will also investigate how Apache Hive streamlines data retrieval and manipulation in a Hadoop environment, serving as a link between conventional SQL databases and the world of large-scale data processing platforms.

Architectural Analysis - Hadoop

Design Principles:
The design principle of hadoop is to allow large computations and data storage to be distributed across a cluster of machines, each contribute a part of the processing work, so that large amount of data can be processed with standard servers or computers. To achieve this, hadoop must also be reliable and easily scaled.


Apache Hadoop Ecosystem
Management & Monitoring (Ambari)
Coordination (ZooKeeper)
Workflow & Scheduling (Oozie)
Scripting (Pig)
Machine Learning (Mahout)
Query (Hive)
NoSQL Database (HBase)
Data Integration (Sqoop/REST/ODBC)
Distributed Processing (MapReduce)
Distributed Storage (HDFS)

Architectural Analysis - Hadoop

Components and Layers:

HDFS(Hadoop Distributed File System): Responsible for the distribution of storage of data across nodes in the cluster. It consist of NameNode, which manage namespace and blocks, and DataNode, which contains the block replica.

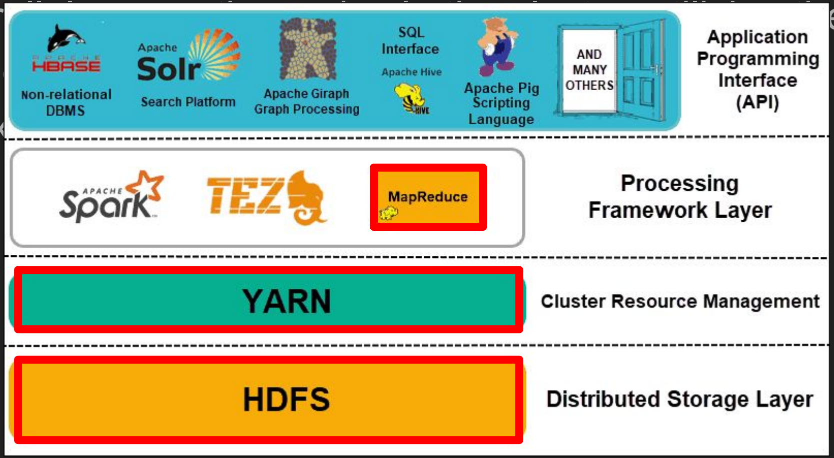
YARN (Yet Another Resource Negotiator): Manages and schedules resources for applications in the cluster.

MapReduce: Handles the processing of data in par e

Map and Reduce tasks. It consist of JobTracker an

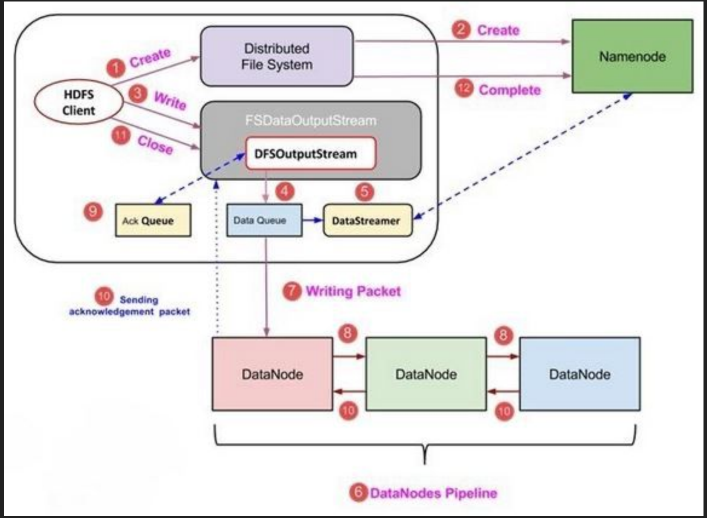
scheduling and resource management, and task ex

.



Architectural Analysis - Hadoop

Data Flow Mechanisms:
Data is written into HDFS, and it is replicated across multiple nodes for parallel processing and fault tolerance. Then MapReduce jobs are executed, processing data where it resides in HDFS. Finally processed results are written back to HDFS, and users can retrieve the results from there.



Architectural Analysis - Hadoop

Scalability:

Horizontal Scalability: Easily scales by adding more nodes to the cluster to handle increased storage and processing requirements.
Data Parallelism: Achieves scalability through data parallelism, breaking down tasks into smaller chunks that can be processed independently.

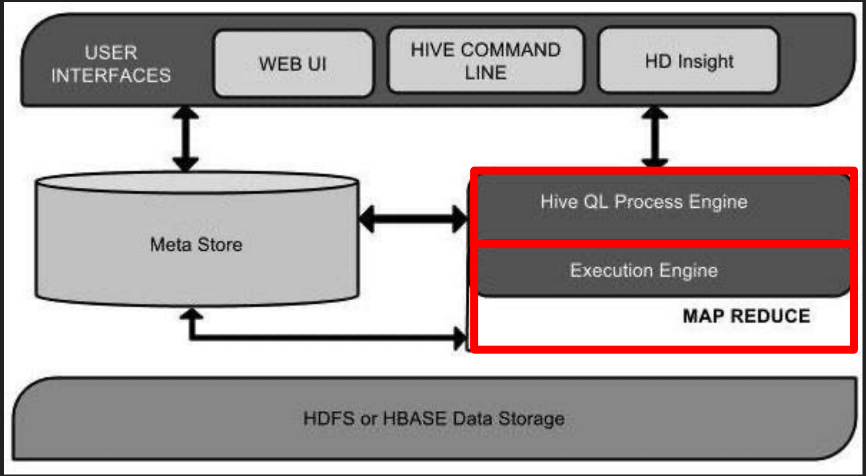
Fault Tolerance:

Data Replication: HDFS replicates data across nodes, providing fault tolerance in case of node failures.
Task Redundancy: MapReduce tasks are rerun on other nodes if a node fails during processing.

Architectural Analysis - Hive

Design Principles:

SQL-Like Query Language (HiveQL): Hive provides a high-level query language (HiveQL) similar to SQL, making it accessible to users familiar with relational databases.
Abstraction Over MapReduce: Abstracts the complexity of MapReduce programming, enabling users to query data without writing low-level MapReduce code.



Architectural Analysis - Hive

Components and Layers:

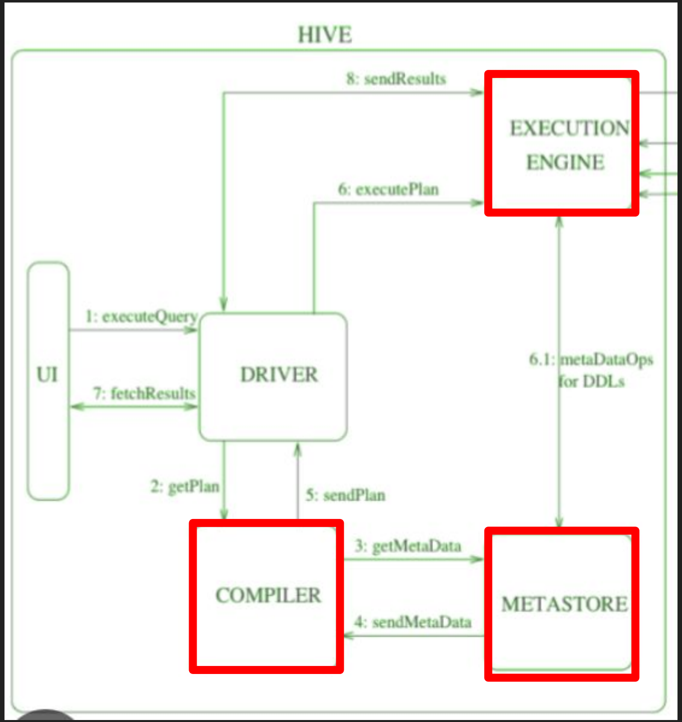
UI – The user interface for users to submit queries and other operations to the system.

Driver – The component which receives the queries and fetches results.

Compiler – The component that parses the query from the driver, does semantic analysis on the different query blocks and query expressions and eventually generates an execution plan with the help of the table and partition metadata looked up from the metastore.

Metastore – The component that stores all the structure information of the various tables and partitions in the warehouse including column and column type information, and the corresponding HDFS files where the data is stored.

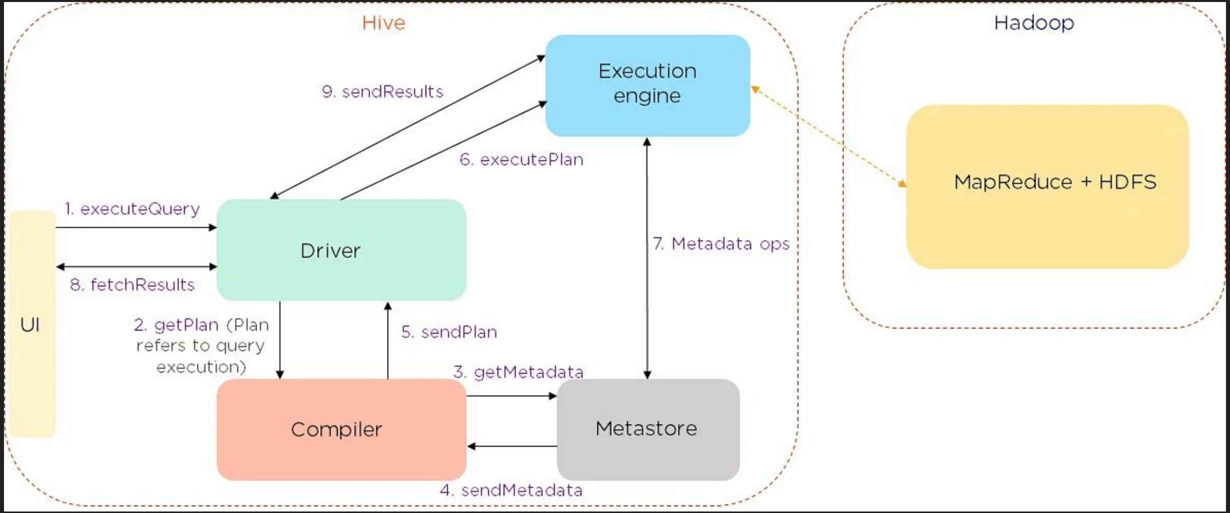
Execution Engine – The component which executes the execution plan created by the compiler.



Architectural Analysis - Hive

Data Flow Mechanisms:

Query Parsing: HiveQL queries are parsed and optimized into a logical execution plan.
Task Execution: The execution engine processes the logical plan and executes tasks in parallel across the cluster.
Result Storage: Processed results can be stored in various formats, including HDFS.



Architectural Analysis - Hive

Scalability:

Distributed Processing: Leverages the distributed processing capabilities of the underlying Hadoop infrastructure.

Support for Different Execution Engines: The choice of execution engine (MapReduce, Tez, or Spark) allows adapting to different scalability requirements.

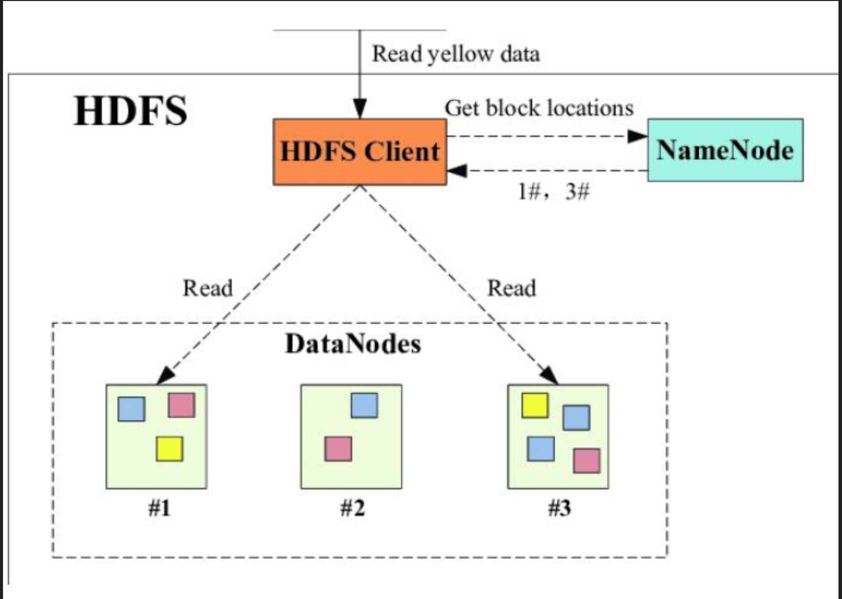
Fault Tolerance:

Metastore Redundancy: Metadata stored in the Metastore can be made fault-tolerant through replication or backup strategies.

Query Retry: If a task fails during execution, Hive can retry the task on another node.

Functionality and Features - Data Storage Methods

Hadoop Distributed File System (HDFS): The primary storage system used by Hadoop and Hive. It transfers data between nodes quickly to handle large amounts of data.

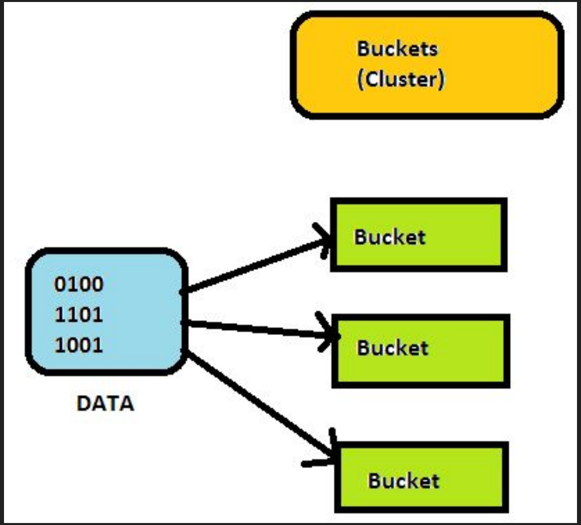


Hadoop Distributed File System (HDFS) structure and its reading process.

Functionality and Features - Query Optimization Techniques

Partitioning & Bucketing

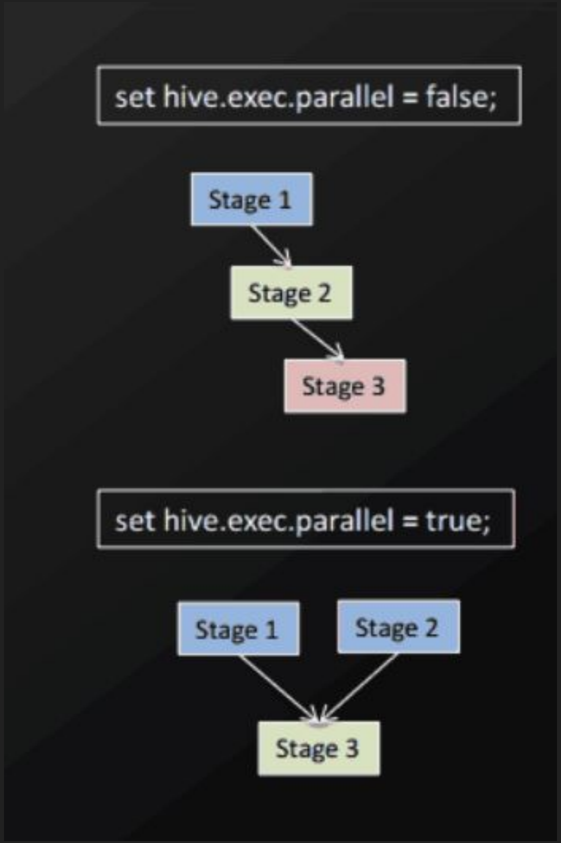
- By partitioning, it splits tables into more digestible pieces based on their columns.
- This helps query speed as it requires less data to be searched. While with bucketing, data is separated into buckets placed into different files in the Hadoop Distributed File System (HDFS; Hadoop's primary storage system).
- This allows more efficient querying of large sets of data, such as ones with JOIN operations and aggregations.



Functionality and Features - Query Optimization Techniques

Parallel Execution

- Hadoop can run MapReduce processes in parallel (In the Mapping step, data is split between parallel processing tasks)
- Single sophisticated Hive queries are frequently converted into many MapReduce tasks, which are then run in the order in which they were created. They can then take advantage of unused capacity on a cluster to increase cluster utilization while lowering query execution times overall
- toggling a single flag - SET hive.exec.parallel=true

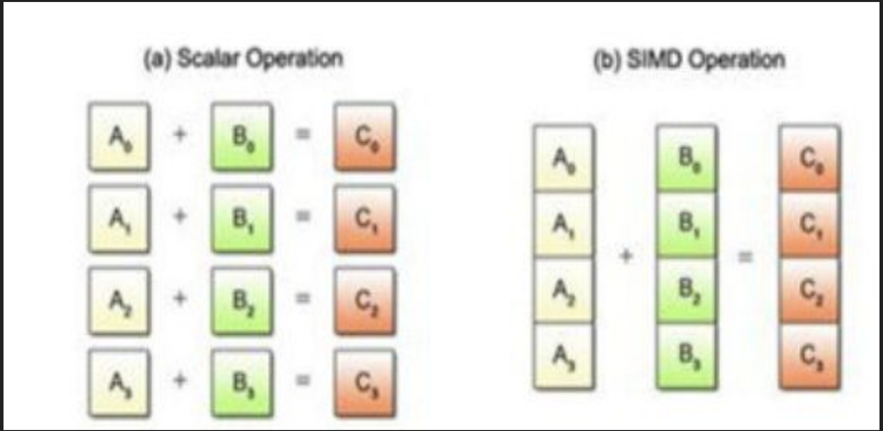

set hive.exec.parallel = false;
Stage 1
Stage 2
Stage 3
set hive.exec.parallel = true;
Stage 1
Stage 2
Stage 3

Functionality and Features - Query Optimization Techniques

Vectorization

- Used to cut CPU usage considerably where scan, aggregates, and join operations are used. This is done by processing a block consisting of 1024 rows at a time rather than one row at a time. Each block's column is stored as a vector and then operations such as arithmetic and comparison operations are by iterating through a loop quicker than normal, as there will be no function called during the loop. This method of query optimization uses the CPU pipeline and cache memory to help speed up the operation. Vectorization leads to cutting down slow queries by using the CPU architecture more.

- Set `hive.vectorized.execution = true`
- Set `hive.vectorized.execution.enabled=true`



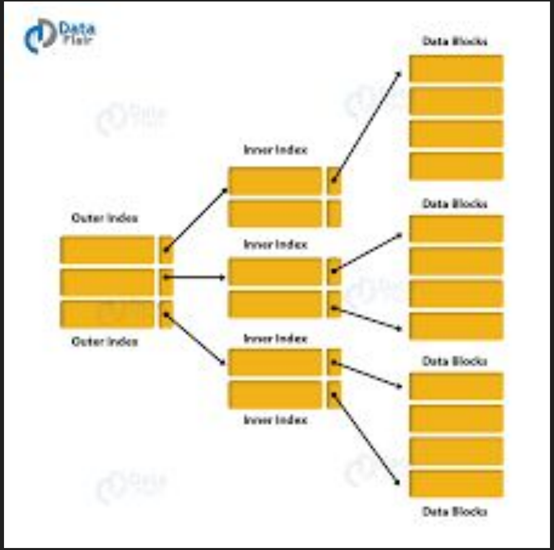
Functionality and Features - Query Optimization Techniques

Hive Indexing

- the original table use of indexing will create a separate called index table which acts as a reference
- no need for the query to scan all the rows in the table while we perform a query on a table that has an index

Cost-Based Optimization

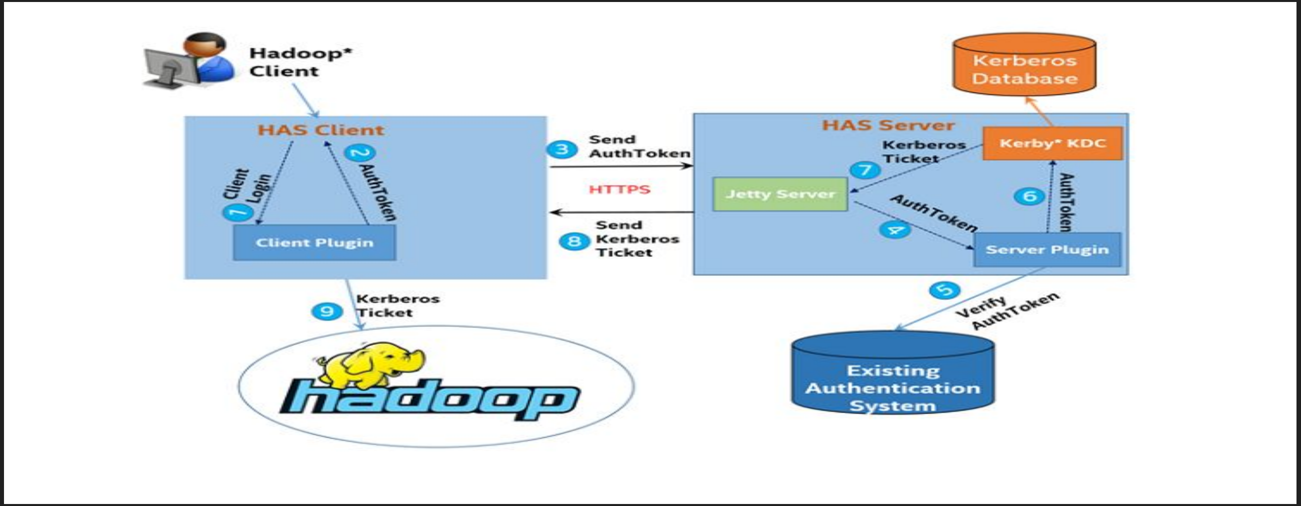
- Scanning the query, CBO generates an effective execution plan
- Queries them into an operator tree, assigns costs to operators,
- chooses the most cost-effective strategy



Functionality and Features - Security Measures

Authentication and Access Control

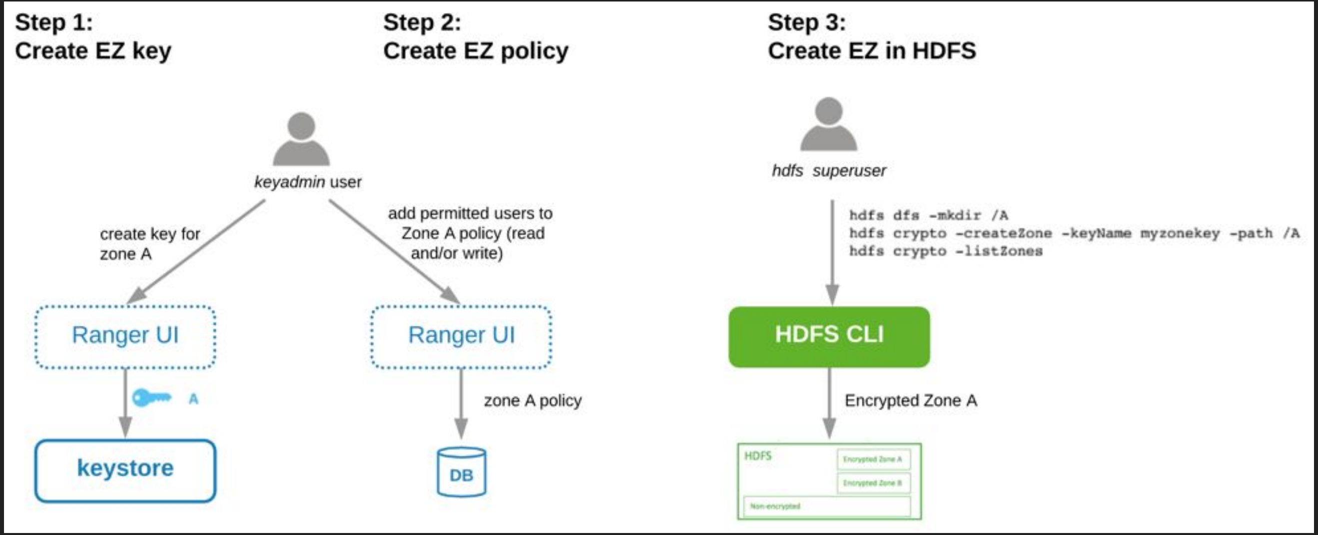
- Needs to validate users identities and can allow different levels of access to certain users using the Kerberos technique
- Guarantees that only authorized users can access the database



Functionality and Features - Security Measures

Data encryption:

- Data is protected via encryption by transforming the data into an unreadable format
- Unauthorized access will not be able to understand the data in the database

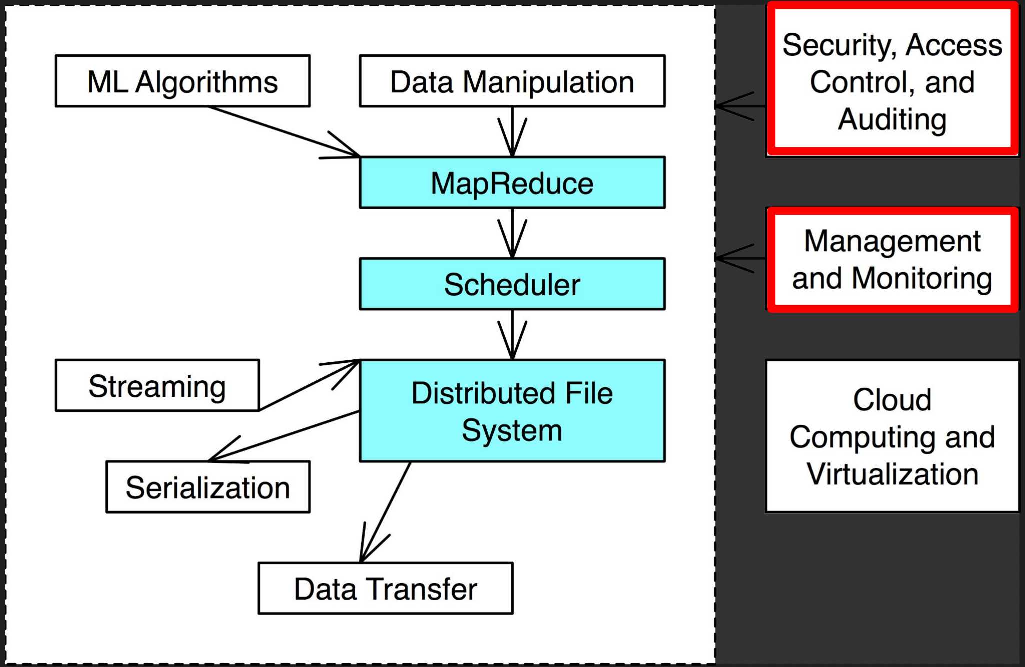


```
hdfs dfs -mkdir /A
hdfs crypto -createZone -keyName myzonekey -path /A
hdfs crypto -listZones
```

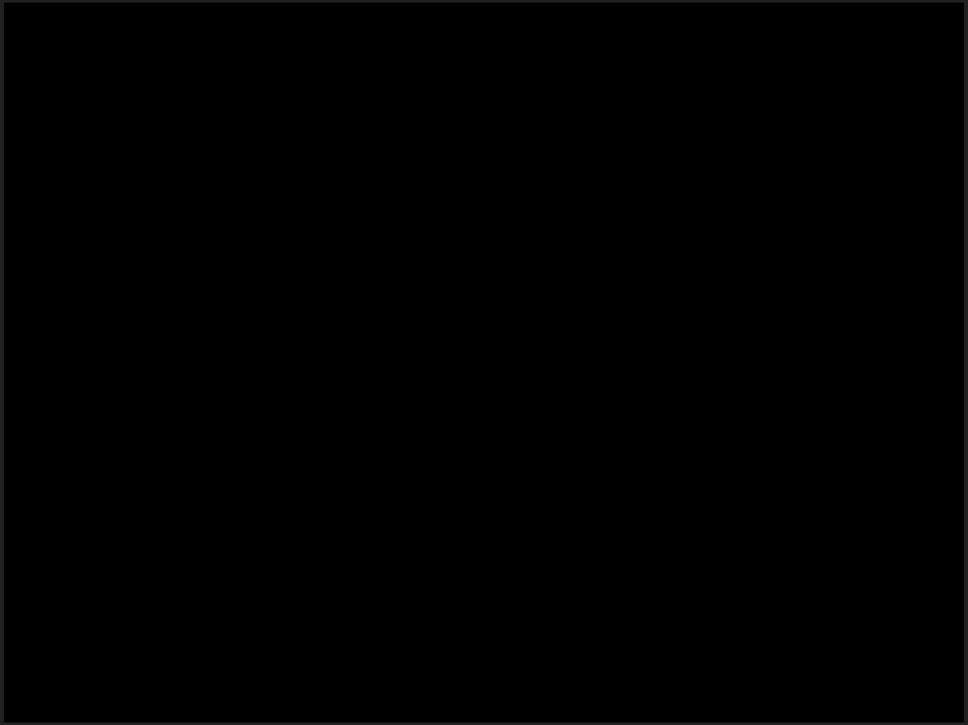
Functionality and Features - Security Measures

Audits and Monitoring

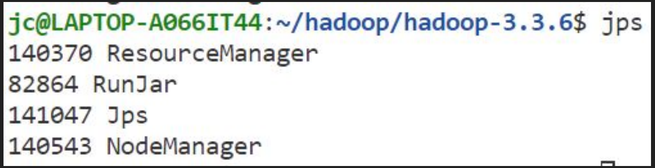
- Monitors the Hadoop ecosystem to detect unusual behaviour or potential security breaches
- Logging and auditing tools can allow for quick discovery of security incidents



Demo



Code



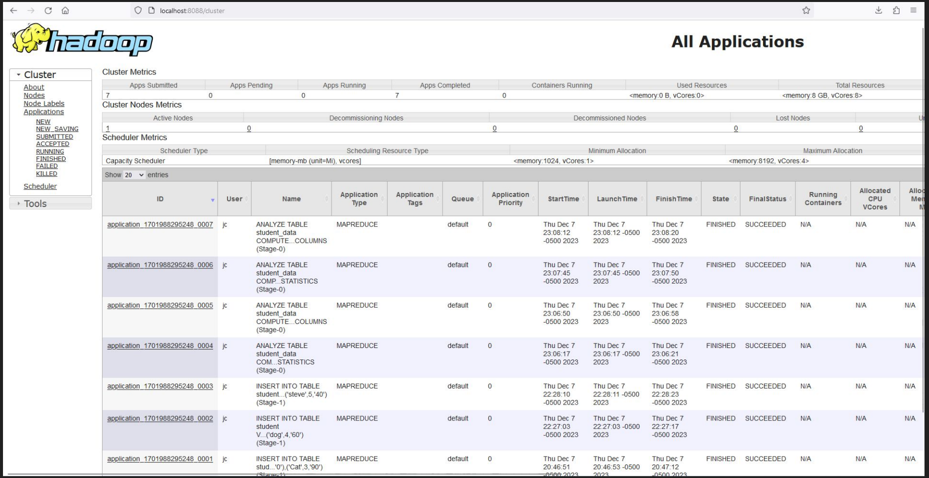
```
jc@LAPTOP-A066IT44:~/hadoop/hadoop-3.3.6$ jps
140370 ResourceManager
82864 RunJar
141047 Jps
140543 NodeManager
```

Apache Hadoop:

```
start-all.sh
start-dfs.sh                # this will start namenode, datanode and namenode
start-yarn.sh               # this will start node manager and resource manager
Jps                         # check running daemons
```

Namenode Health: http://localhost:9870
Hadoop UI: http://localhost:8088/cluster/nodes

Stop:
Stop-yarn.sh, stop-dfs.sh; alternatively stop-all.sh



localhost:8088/cluster

All Applications

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	Used Resources	Total Resources
7	0	0	7	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	U
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allo Me
application_1701988295248_0007	jc	ANALYZE TABLE student_data COMPUTE...COLUMNS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:08:12 -0500 2023	Thu Dec 7 23:08:12 -0500 2023	Thu Dec 7 23:08:20 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0006	jc	ANALYZE TABLE student_data COMP...STATISTICS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:07:45 -0500 2023	Thu Dec 7 23:07:45 -0500 2023	Thu Dec 7 23:07:50 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0005	jc	ANALYZE TABLE student_data COMPUTE...COLUMNS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:06:50 -0500 2023	Thu Dec 7 23:06:50 -0500 2023	Thu Dec 7 23:06:58 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0004	jc	ANALYZE TABLE student_data COM...STATISTICS (Stage-0)	MAPREDUCE		default	0	Thu Dec 7 23:06:17 -0500 2023	Thu Dec 7 23:06:17 -0500 2023	Thu Dec 7 23:06:21 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0003	jc	INSERT INTO TABLE student...('steve',5,'40') (Stage-1)	MAPREDUCE		default	0	Thu Dec 7 22:28:10 -0500 2023	Thu Dec 7 22:28:11 -0500 2023	Thu Dec 7 22:28:23 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0002	jc	INSERT INTO TABLE student V...('dog',4,'60') (Stage-1)	MAPREDUCE		default	0	Thu Dec 7 22:27:03 -0500 2023	Thu Dec 7 22:27:03 -0500 2023	Thu Dec 7 22:27:17 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A
application_1701988295248_0001	jc	INSERT INTO TABLE stud...'0'),('Cat',3,'90') (Stage-1)	MAPREDUCE		default	0	Thu Dec 7 20:46:51 -0500 2023	Thu Dec 7 20:46:53 -0500 2023	Thu Dec 7 20:47:12 -0500 2023	FINISHED	SUCCEEDED	N/A	N/A	N/A

Code

Apache Hive:

Creating Hive warehouse:

```
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -chmod g+w /user/hive/warehouse

Initiate the Derby database to store metadata
$HIVE_HOME/bin/schematool -dbType derby -initSchema

cd $HIVE_HOME/bin
hive
```

Key Insights & Findings

Here is how our study found how Hadoop & Hive relate to the requirements of modern data frameworks:

Streaming Analytics:

- Hive streaming API allows data to be pumped continuously into Hive
- Incoming data is continuously committed into small batches of records in an existing Hive partitioning or table

Fault Tolerance & Data Protection

- Hadoop displayed logs that we were able to see the queries made onto our tables in the databases we managed

Key Insights & Findings

Velocity: Hadoop & Hive were both quick to take in our inserts and queries and were able to process them quickly

Volume: We found Hadoop to be quite scalable as it could manage multiple nodes at once and display their usage and statistics. You would be able to manage the different systems modifying the databases and scale it larger to manage greater amounts of data.

Cost-effective: We found it cheap to run, as it ran quite well on our computers

Challenges Faced

Complexity: Hadoop is complex, requires lots of tuning. Using a GUI would have made it easier to use.

We had many challenges in getting Apache Hadoop and Hive working, and creating enough data to test the larger features of the softwares. The first issue was installation. There was a way to install it on Windows, but issues on configuring it led to many errors. We then installed it on WSL (Windows Subsystem for Linux) and managed to get it running.

After that the next challenge was getting large datasets to test the major features of Hadoop and Hive. We were not able to get a large dataset to test as Hadoop deals with thousand and millions of rows of data. We tested both softwares with small databases of a few tables and rows of data. It was enough to see Hadoop's Mapper and Reducer on each query implemented onto the tables in our databases.