Pune Institute of Computer Technology, Pune

DEPARTMENT OF COMPUTER ENGINEERING

LP2 Mini Project (AY 2022-23)

Cloud Controller for SaaS using HDFS

Class: **TE1**

Batch: **L1**

Sem: **VI**

SUBMITTED BY

31123 – Sushilkumar Dhamane

31127 – Shreyash Dwivedi

31132 – Shreyash Halge

Under The guidance of

Prof. Yogesh Handge

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to everyone who contributed to the successful completion of this project. Their support, guidance, and expertise were invaluable throughout the entire journey.

First and foremost, we would like to thank our project supervisor, Prof. Yogesh Handge, for their unwavering support, insightful guidance, and continuous encouragement. Their expertise and feedback were instrumental in shaping the project and ensuring its successful implementation.

We would also like to extend our appreciation to our fellow classmates and friends who aided and feedback during various stages of the project. Their constructive input and valuable insights helped us refine our ideas and improve the overall quality of the web application.

Furthermore, we would like to acknowledge the support and resources provided by PICT. The access to necessary infrastructure, software, and libraries greatly facilitated the development process.

Motivation

The motivation behind this report lies in the increasing demand for efficient and secure cloud-based solutions, specifically in laboratory environments. Laboratories often deal with large volumes of data and require seamless access to software applications for research, experimentation, and collaboration.

The motivation for developing a cloud controller for SaaS using HDFS in the laboratory environment stems from the following factors:

1. **Scalability and Flexibility:** Cloud computing offers unparalleled scalability, allowing laboratories to rapidly scale their computing resources up or down based on their specific needs. By implementing a cloud controller for SaaS, laboratories can harness the power of distributed storage and processing, enabling them to handle large datasets and accommodate fluctuating workloads effectively.

2. **Cost-Effectiveness:** The pay-as-you-go model of cloud computing eliminates the need for upfront investments in hardware and infrastructure. By leveraging cloud resources, laboratories can reduce their operational costs and allocate their budgets more efficiently towards research and innovation.

3. **Collaboration and Accessibility**: Cloud-based solutions enable seamless collaboration among researchers and facilitate remote access to software applications and datasets. By implementing a cloud controller for SaaS, laboratories can provide their researchers with on-demand access to shared applications and data, regardless of their physical location, fostering collaboration and enhancing productivity.

4. **Data Security and Privacy:** Laboratories deal with sensitive and proprietary data that require robust security measures. By incorporating encryption techniques and secure data handling practices, the cloud controller ensures the confidentiality and integrity of the data stored and transferred within the SaaS environment.

5. **Exploration of Open-Source Technologies:** Open-source technologies, such as Hadoop and HDFS, offer powerful and cost-effective solutions for distributed storage and processing. By implementing the cloud controller using these technologies, laboratories can leverage the vast community support, ongoing enhancements, and interoperability with other open-source tools and frameworks.

Introduction

In today's rapidly evolving technological landscape, cloud computing has emerged as a game-changing paradigm, enabling organizations to efficiently manage their software and infrastructure resources. Among the various cloud computing models, Software as a Service (SaaS) has gained significant popularity due to its ability to provide on-demand access to software applications over the internet, eliminating the need for local installations and maintenance.

The objective of this report is to present the implementation of a cloud controller for SaaS using the Hadoop Distributed File System (HDFS) within a laboratory environment. The cloud controller serves as the bridge between the client applications and the underlying cloud infrastructure, facilitating file management operations such as file segmentation, encryption, and secure uploading and downloading of files.

The choice of HDFS as the underlying distributed file system is driven by its scalability, fault tolerance, and compatibility with the Apache Hadoop ecosystem. By leveraging HDFS, the cloud controller enables efficient storage, retrieval, and processing of large volumes of data, making it an ideal choice for managing SaaS applications in a laboratory setting.

Abstract

The increasing demand for scalable and efficient cloud-based solutions has led to the development of a cloud controller for Software as a Service (SaaS) using the Hadoop Distributed File System (HDFS) in a laboratory environment. This report presents the implementation details, challenges faced, and insights gained during the development process.

The objective of the cloud controller is to provide a seamless interface between client applications and the underlying cloud infrastructure, enabling file management operations such as file segmentation, encryption, and secure uploading and downloading of files. By leveraging the scalability and fault tolerance of HDFS, the cloud controller ensures efficient storage, retrieval, and processing of large volumes of data, making it suitable for managing SaaS applications in laboratory settings.

The implementation of the cloud controller involves integrating open-source technologies, such as Java and Hadoop, to deliver the required functionality. Key components of the cloud controller system, including the client application, cloud controller, HDFS, encryption module, and file segmentation module, are discussed in detail.

Literature Survey

To develop an effective cloud controller for SaaS using HDFS, a comprehensive literature survey was conducted to explore existing research and industry practices. The survey focused on the following key areas:

1. **Cloud Computing and SaaS**: The literature survey revealed numerous studies and articles discussing the concepts and benefits of cloud computing, as well as the Software as a Service (SaaS) model. Various papers highlighted the advantages of SaaS, including scalability, cost-effectiveness, and ease of deployment. Additionally, research on the challenges and best practices for building cloud-based applications provided valuable insights for designing the cloud controller system.

2. **Hadoop and HDFS**: A significant portion of the literature survey was dedicated to understanding Hadoop and the Hadoop Distributed File System (HDFS). Numerous research papers and technical articles were explored to gain insights into the architecture, features, and usage patterns of Hadoop and HDFS. These resources covered topics such as data storage and processing, fault tolerance, scalability, and performance optimization techniques.

3. **File Segmentation and Distribution**: To tackle the file segmentation and distribution aspect of the cloud controller, studies related to file partitioning techniques were reviewed. This included research on data fragmentation, sharding, and distributed file systems. The literature survey provided insights into various segmentation algorithms, metadata management approaches, and data reconstruction techniques, which proved invaluable in designing an efficient file segmentation module.

4. **Encryption and Security**: A crucial aspect of the cloud controller implementation is data security. The literature survey encompassed research on encryption algorithms, secure data transmission, and key management in distributed systems. Various encryption techniques, such as symmetric and asymmetric encryption, were explored to identify the most suitable approach for securing files in the cloud controller system.

5. **Distributed Systems and Fault Tolerance**: Since the cloud controller operates in a distributed environment, understanding distributed systems' concepts and fault tolerance mechanisms was vital. The literature survey included studies on distributed systems architectures, consensus algorithms, data replication, and fault tolerance approaches. These resources provided insights into the design and implementation of fault-tolerant systems and influenced the error handling and fault tolerance aspects of the cloud controller.

Problem definition

The problem at hand is to develop a cloud controller for Software as a Service (SaaS) over an existing Local Area Network (LAN) in a laboratory environment. The cloud controller should facilitate basic operations such as file segmentation, encryption, and uploading/downloading files in an encrypted form. The aim is to build a robust and secure system that enables users to interact with the cloud controller through a client application, while leveraging open-source technologies and Hadoop Distributed File System (HDFS).

Scope

Selection of appropriate technologies and tools for building the cloud controller system. Description of the programming language and frameworks used, such as Java and Spring Boot for the client application. Detailed explanation of the steps involved in developing the cloud controller using Java. Description of the client application development, including the user interface implementation using HTML, CSS, and JavaScript. Overview of the backend logic development, covering user request handling and interaction with HDFS through the HDFS Java API. Integration of encryption libraries for encrypting and decrypting files before storage and retrieval from HDFS. Implementation of the cloud controller logic to handle file upload requests from the client application. Segmentation of the uploaded file, encryption of each segment, and storage in HDFS. Provision of download functionality to retrieve encrypted file segments from HDFS, decrypt them, and reconstruct the original file on the client side. Implementation of a file segmentation module to divide large files into smaller segments or blocks.

System Architecture

The cloud controller system consists of the following components:

- **Client Application:** A user interface for interacting with the cloud controller.
- **Cloud Controller:** The core component responsible for handling user requests and managing file operations.
- **HDFS:** The distributed file system used to store and retrieve files.
- **Encryption Module:** A module responsible for encrypting and decrypting files.
- **File Segmentation Module:** A module responsible for dividing files into segments or blocks.

Technologies used…

The implementation of the cloud controller for SaaS using HDFS involves the integration of various open-source technologies to provide the necessary functionality. The key technologies used in the development process are as follows:

1. **Java:** Java is a widely used programming language known for its platform independence and extensive libraries. It is the primary language used for implementing the cloud controller and client application due to its robustness, scalability, and compatibility with the Hadoop ecosystem.

2. **Apache Hadoop:** Apache Hadoop is an open-source framework that provides a distributed computing and storage infrastructure for processing large datasets. It includes components such as Hadoop Distributed File System (HDFS) for distributed storage and MapReduce for distributed processing. Hadoop is utilized as the underlying infrastructure to support the storage and processing requirements of the cloud controller.

3. **Hadoop Distributed File System (HDFS):** HDFS is a distributed file system designed to store and manage large datasets across multiple machines. It provides fault tolerance, high throughput, and data locality. The cloud controller leverages HDFS to store the segmented files and ensure data redundancy and scalability.

4. **Encryption Libraries:** Various encryption libraries and algorithms are used to implement secure file encryption and decryption. Commonly used libraries include Java Cryptography Architecture (JCA) and Bouncy Castle. These libraries offer a wide range of encryption algorithms, such as AES (Advanced Encryption Standard), to ensure data security within the cloud controller.

5. **Apache Maven:** Maven is a build automation tool used for managing dependencies, building, and packaging Java projects. It simplifies the project setup process and provides a consistent build environment. Maven is employed to manage the dependencies of the cloud controller and ensure smooth integration with Hadoop and other required libraries.

6. **Integrated Development Environment (IDE):** An IDE, such as Eclipse or IntelliJ IDEA, is used to develop and debug the cloud controller and client application. These IDEs offer powerful tools and features that enhance development productivity, such as code editing, debugging, and project management capabilities.

7. **Testing Frameworks:** Testing frameworks, such as JUnit, are utilized to create comprehensive test cases and automate the testing process. These frameworks enable the validation of various functionalities, error handling, performance, and scalability of the cloud controller system.

8. **Operating System and Networking:** The implementation of the cloud controller is carried out on a Linux-based operating system, such as Fedora. The networking infrastructure is configured to ensure proper communication between the client application and the cloud controller over the laboratory LAN.

Implementation

I. Setting up HDFS:

1. Install Hadoop on a cluster of machines in the laboratory LAN.
2. Configure the Hadoop cluster by setting up the core-site.xml and hdfs-site.xml files to define the cluster settings and HDFS properties.
3. Start the HDFS daemons (NameNode and DataNodes) to make the distributed file system operational.

II. Cloud Controller Development:

1. Create a Java Spring Boot application to build the client application.
2. Implement the user interface using HTML, CSS, and JavaScript for file upload/download operations.
3. Develop the backend logic in Java for handling user requests and interacting with HDFS.
4. Use the HDFS Java API to connect to the HDFS cluster and perform file operations like uploading, downloading, and file segmentation.
5. Integrate encryption libraries (e.g., Bouncy Castle or Jasypt) to encrypt and decrypt files before storing and retrieving them from HDFS.

III. File Segmentation:

1. Implement a file segmentation module in Java to divide large files into smaller segments or blocks.
2. Define the segment size for file division, taking into account factors like HDFS block size and optimal file transfer performance.
3. Associate metadata with each segment, such as segment name, sequence number, and file mapping information.

IV. File Encryption:

1. Utilize encryption libraries (e.g., Bouncy Castle or Jasypt) to encrypt files before storing them in HDFS.
2. Implement an encryption module in Java that supports encryption algorithms like AES or RSA.
3. Define encryption keys and manage encryption-related metadata.

V. Upload/Download Operations:

1. Develop the cloud controller to handle file upload requests from the client application.
2. Implement logic to segment the uploaded file, encrypt each segment, and store them in HDFS.
3. Provide download functionality to retrieve the encrypted file segments from HDFS, decrypt them, and reconstruct the original file on the client side.
4. Testing and Deployment:
5. Perform thorough testing of the cloud controller by simulating various user scenarios.
6. Verify the proper functioning of file segmentation, encryption, and upload/download operations.
7. Optimize the system for performance and security.
8. Deploy the cloud controller.

Challenges Faced…

1. **System Integration:** Integrating and coordinating multiple components, such as the client application, cloud controller, HDFS, encryption module, and file segmentation module, can be complex. Ensuring seamless communication and data flow between these components might require careful design and implementation.

2. **Hadoop and HDFS Configuration:** Configuring Hadoop and HDFS correctly can be challenging, especially for newcomers to the technology stack. Issues related to networking, cluster setup, and configuration parameters may arise and need to be resolved for the proper functioning of the system.

3. **File Segmentation and Reconstruction:** Implementing an efficient file segmentation module and reconstructing the original file from the segments can be intricate. Considerations such as segment size, metadata management, and maintaining the order of segments during reconstruction require careful attention.

4. **Encryption and Security:** Implementing secure encryption algorithms and managing encryption keys can be challenging. Ensuring the confidentiality and integrity of the files while optimizing the encryption process for performance is a crucial aspect of the implementation.

5. **Performance Optimization**: Working with large files and distributed storage introduces performance considerations. Optimizing file transfer, encryption/decryption processes, and leveraging parallelism in HDFS for efficient data processing and retrieval can be challenging but necessary for a high-performing system.

6. **Error Handling and Fault Tolerance:** Implementing robust error handling and fault tolerance mechanisms is crucial in distributed systems. Addressing issues such as network failures, data corruption, and handling exceptions during file operations require careful consideration and implementation.

7. **Testing and Scalability:** Testing the cloud controller for various scenarios, such as concurrent user interactions, large file uploads/downloads, and fault tolerance, is essential to ensure its reliability and scalability. Creating comprehensive test cases and simulating realistic usage patterns can be time-consuming and complex.

8. **Deployment and Infrastructure Setup:** Deploying the cloud controller on the laboratory LAN may require setting up and configuring the necessary infrastructure, including network configurations, hardware requirements, and cluster management. Coordinating these activities with existing laboratory resources might present logistical challenges.

Conclusion

In this report, we presented the implementation of a cloud controller for Software as a Service (SaaS) using the Hadoop Distributed File System (HDFS) in a laboratory environment. The cloud controller serves as a vital component in managing file operations, such as file segmentation, encryption, and secure uploading and downloading of files.

Through the integration of open-source technologies, including Java, Hadoop, and HDFS, we have successfully developed a robust and scalable cloud controller system. The implementation of the cloud controller addresses the key objectives of scalability, cost-effectiveness, collaboration, and data security in the laboratory setting.

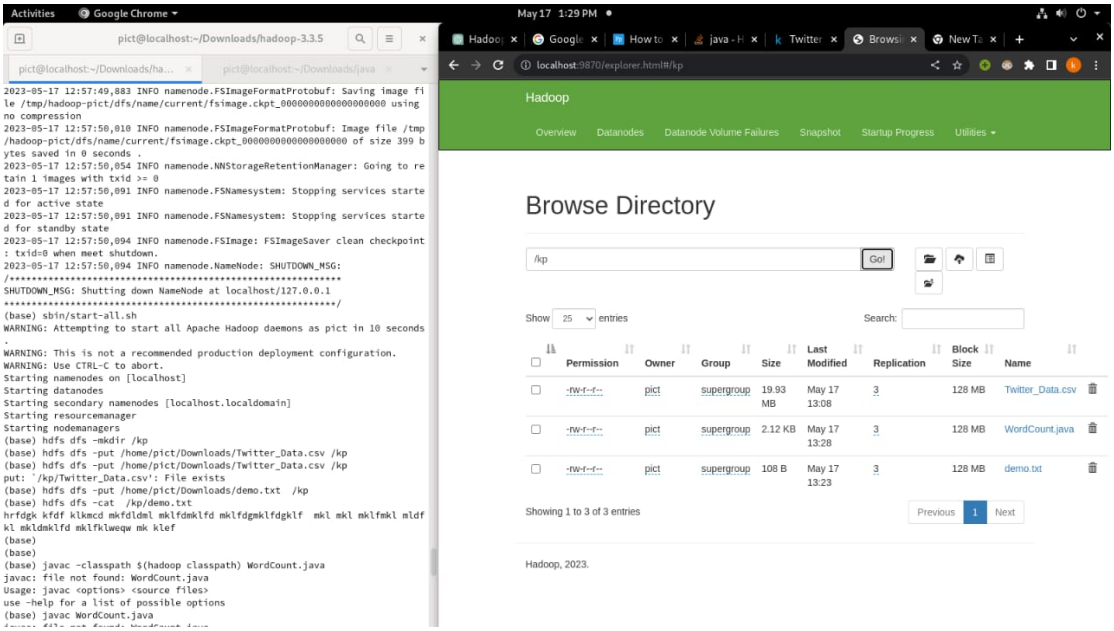
The literature survey conducted during the project provided valuable insights into cloud computing, Hadoop, file segmentation, encryption, fault tolerance, and performance optimization. These insights guided the design decisions and best practices adopted throughout the implementation process, ensuring the system leverages industry standards and addresses the challenges faced in the laboratory environment.

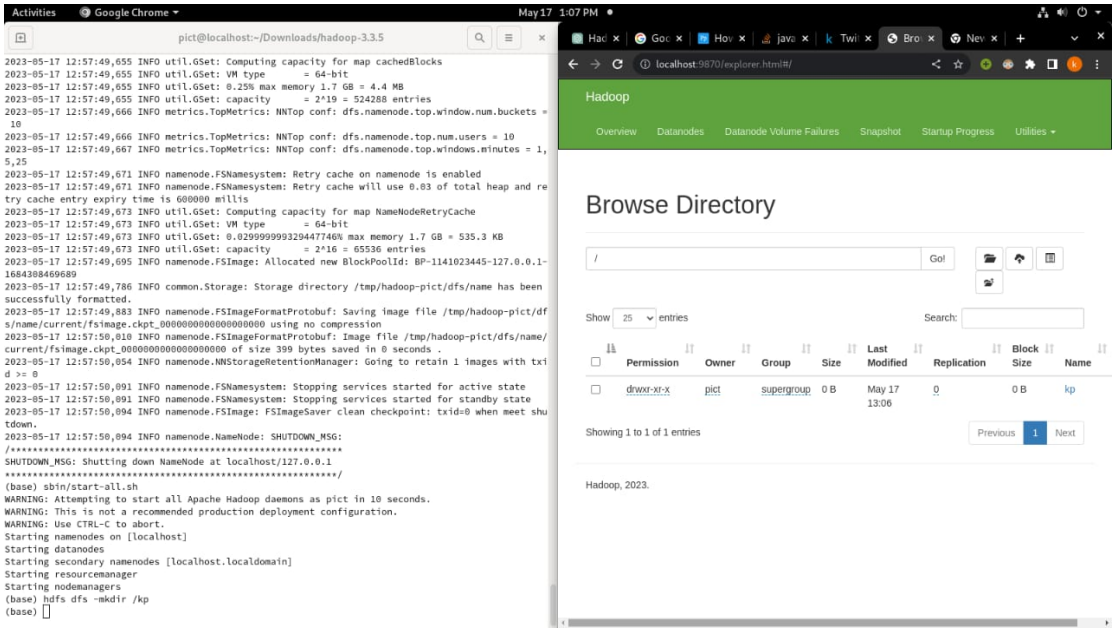
Challenges encountered during the implementation, such as system integration, Hadoop configuration, file segmentation, encryption, and performance optimization, were addressed through careful analysis and appropriate solutions. The testing framework developed during the project enabled comprehensive testing of the cloud controller system, validating its reliability and scalability.

By implementing the cloud controller for SaaS using HDFS, laboratories can leverage the scalability, cost-effectiveness, and accessibility offered by cloud computing. Researchers can seamlessly access shared software applications, collaborate effectively, and process large volumes of data in a secure manner. The cloud controller empowers laboratories to focus more on research and innovation, while the underlying infrastructure handles the complexities of file management and data processing.

As a future recommendation, additional features and enhancements can be considered, such as advanced file segmentation algorithms, integration with other cloud services, and further optimization for performance and fault tolerance. These enhancements would further enhance the capabilities and effectiveness of the cloud controller system in meeting the evolving needs of laboratories.


Activities
Terminal
May 17 12:57 PM
pict@localhost:~/Downloads/hadoop-3.3.5
(base) hdfs
bash: hdfs: command not found...
(base) ls
container-executor hdfs mapred.cmd yarn
hadoop hdfs.cmd oom-listener yarn.cmd
hadoop.cmd mapred test-container-executor
(base) cd ..
(base) bin/hdfs namenode -format
ERROR: Cannot execute /Downloads/hadoop-3.3.5/libexec/hdfs-config.sh.
(base) sbin/start-all.sh
ERROR: Cannot execute /Downloads/hadoop-3.3.5/libexec/hadoop-config.sh.
(base) ls -l libexec/hadoop-config.sh
-rwxr-xr-x. 1 pict pict 5457 Mar 15 21:27 libexec/hadoop-config.sh
(base) chmod +x libexec/hadoop-config.sh
(base) $HADOOP_HOME/bin/hadoop version
bash: /Downloads/hadoop-3.3.5/bin/hadoop: No such file or directory
(base) $HADOOP_HOME/bin/hadoop -version
bash: /Downloads/hadoop-3.3.5/bin/hadoop: No such file or directory
(base)
(base)
(base)
(base) ls -l libexec/hdfs-config.sh
-rwxr-xr-x. 1 pict pict 3735 Mar 15 21:32 libexec/hdfs-config.sh
(base)
(base)
(base) chmod +x libexec/hdfs-config.sh
(base)
(base)
(base) nano ~/.bashrc
(base) source ~/.bashrc
(base) $HADOOP_HOME/bin/hdfs version
Hadoop 3.3.5
Source code repository https://github.com/apache/hadoop.git -r 706d88266abcee09ed78fbaa0ad5f74d818ab0e9
Compiled by stevel on 2023-03-15T15:56Z
Compiled with protoc 3.7.1
From source with checksum 6bbd9afcf4838a0eb12a5f189e9bd7
This command was run using /home/pict/Downloads/hadoop-3.3.5/share/hadoop/common/hadoop-common-3.3.5.jar
(base) $HADOOP_HOME/bin/hadoop version
Hadoop 3.3.5
Source code repository https://github.com/apache/hadoop.git -r 706d88266abcee09ed78fbaa0ad5f74d818ab0e9
Compiled by stevel on 2023-03-15T15:56Z
Compiled with protoc 3.7.1
From source with checksum 6bbd9afcf4838a0eb12a5f189e9bd7
This command was run using /home/pict/Downloads/hadoop-3.3.5/share/hadoop/common/hadoop-common-3.3.5.jar
(base)


Activities
Google Chrome
May 17 1:29 PM
pict@localhost:~/Downloads/hadoop-3.3.5
2023-05-17 12:57:49,883 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-pict/dfs/name/current/fsimage.ckpt_0000000000000000000 using no compression
2023-05-17 12:57:50,010 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-pict/dfs/name/current/fsimage.ckpt_0000000000000000000 of size 399 bytes saved in 0 seconds .
2023-05-17 12:57:50,054 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-17 12:57:50,091 INFO namenode.FSNamesystem: Stopping services started for active state
2023-05-17 12:57:50,091 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-05-17 12:57:50,094 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-05-17 12:57:50,094 INFO namenode.NameNode: SHUTDOWN_MSG:
/**
SHUTDOWN_MSG: Shutting down NameNode at localhost/127.0.0.1
**/
(base) sbin/start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as pict in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [localhost.localdomain]
Starting resourcemanager
Starting nodemanagers
(base) hdfs dfs -mkdir /kp
(base) hdfs dfs -put /home/pict/Downloads/Twitter_Data.csv /kp
(base) hdfs dfs -put /home/pict/Downloads/Twitter_Data.csv /kp
put: `/kp/Twitter_Data.csv': File exists
(base) hdfs dfs -put /home/pict/Downloads/demo.txt /kp
(base) hdfs dfs -cat /kp/demo.txt
hrfdgk kfdf klkmcd mkfdldml mklfdmklfd mklfdgmklfdgklf mkl mkl mklfmkl mldfkl mkldmklfd mklfklweqw mk klef
(base)
(base)
(base) javac -classpath $(hadoop classpath) WordCount.java
javac: file not found: WordCount.java
Usage: javac <options> <source files>
use -help for a list of possible options
(base) javac WordCount.java
localhost:9870/explorer.html#/kp
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/kp
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
-rw-r--r--
pict
supergroup
19.93 MB
May 17 13:08
3
128 MB
Twitter_Data.csv
-rw-r--r--
pict
supergroup
2.12 KB
May 17 13:28
3
128 MB
WordCount.java
-rw-r--r--
pict
supergroup
108 B
May 17 13:23
3
128 MB
demo.txt
Showing 1 to 3 of 3 entries
Previous
1
Next
Hadoop, 2023.


Activities
Google Chrome
May 17 1:07 PM
pict@localhost:~/Downloads/hadoop-3.3.5
2023-05-17 12:57:49,655 INFO util.GSet: Computing capacity for map cachedBlocks
2023-05-17 12:57:49,655 INFO util.GSet: VM type = 64-bit
2023-05-17 12:57:49,655 INFO util.GSet: 0.25% max memory 1.7 GB = 4.4 MB
2023-05-17 12:57:49,655 INFO util.GSet: capacity = 2^19 = 524288 entries
2023-05-17 12:57:49,666 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2023-05-17 12:57:49,666 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2023-05-17 12:57:49,667 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2023-05-17 12:57:49,671 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2023-05-17 12:57:49,671 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
2023-05-17 12:57:49,673 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2023-05-17 12:57:49,673 INFO util.GSet: VM type = 64-bit
2023-05-17 12:57:49,673 INFO util.GSet: 0.029999999329447746% max memory 1.7 GB = 535.3 KB
2023-05-17 12:57:49,673 INFO util.GSet: capacity = 2^16 = 65536 entries
2023-05-17 12:57:49,695 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1141023445-127.0.0.1-1684308469689
2023-05-17 12:57:49,786 INFO common.Storage: Storage directory /tmp/hadoop-pict/dfs/name has been successfully formatted.
2023-05-17 12:57:49,883 INFO namenode.FSImageFormatProtobuf: Saving image file /tmp/hadoop-pict/dfs/name/current/fsimage.ckpt_0000000000000000000 using no compression
2023-05-17 12:57:50,010 INFO namenode.FSImageFormatProtobuf: Image file /tmp/hadoop-pict/dfs/name/current/fsimage.ckpt_0000000000000000000 of size 399 bytes saved in 0 seconds .
2023-05-17 12:57:50,054 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-05-17 12:57:50,091 INFO namenode.FSNamesystem: Stopping services started for active state
2023-05-17 12:57:50,091 INFO namenode.FSNamesystem: Stopping services started for standby state
2023-05-17 12:57:50,094 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2023-05-17 12:57:50,094 INFO namenode.NameNode: SHUTDOWN_MSG:
/**
SHUTDOWN_MSG: Shutting down NameNode at localhost/127.0.0.1
**/
(base) sbin/start-all.sh
WARNING: Attempting to start all Apache Hadoop daemons as pict in 10 seconds.
WARNING: This is not a recommended production deployment configuration.
WARNING: Use CTRL-C to abort.
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [localhost.localdomain]
Starting resourcemanager
Starting nodemanagers
(base) hdfs dfs -mkdir /kp
(base)
localhost:9870/explorer.html#/
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities
Browse Directory
/
Go!
Show 25 entries
Search:
Permission
Owner
Group
Size
Last Modified
Replication
Block Size
Name
drwxr-xr-x
pict
supergroup
0 B
May 17 13:06
0
0 B
kp
Showing 1 to 1 of 1 entries
Previous
1
Next
Hadoop, 2023.