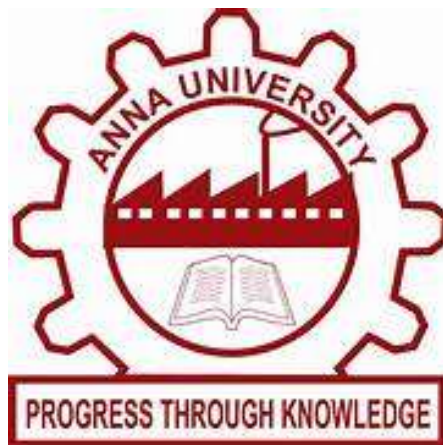


MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY, CHENNAI – 600044.

IMPLEMENTATION OF SIMPLE FEATURES OF TOOLS

A Project Report



Submitted by

Name: Gopika S

Register Number: 2022503505

Year/Semester: Final Year – VII Sem

Department: Computer Science and Engineering

Batch: 2022 – 2026

Subject Name: Cloud Computing

Subject Code: CS6006

Date of Submission: 31.10.2025

1) TITLE

Implementation of a Simple App.js Web Application using Docker

INTRODUCTION

Docker is an **open-source containerization platform** that allows developers to build, deploy, and run applications in isolated environments called containers. It ensures consistency across different systems by packaging code with its dependencies and libraries.

Docker simplifies deployment, testing, and scaling of applications, making it a core tool for DevOps and cloud computing. It runs on a client-server model where the Docker Engine handles container operations, while Docker Hub serves as a centralized image repository.

PURPOSE OF THE TOOL

The main purpose of Docker is to enable portability, efficiency, and isolation in application deployment.

It allows developers to:

- Build once and run anywhere.
- Minimize configuration issues between environments.
- Reduce resource usage compared to traditional virtual machines.
- Quickly deploy updates and maintain version control through Docker images.

Docker is particularly useful in cloud environments where applications must scale easily and remain consistent across nodes.

DEPLOYMENT

Objective:

To create and deploy a simple App.js web application using Docker and display the message *“Hello from Docker on Windows from Gopika!”*

Step 1: Verify Docker Installation

1. Download Docker from the official website and install it on your system.



2. Run the following command to check the version:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell
PS C:\Users\Gopika S\OneDrive\Desktop\cloud-docker-app> docker --version
Docker version 28.5.1, build e180ab8
```

3. Run “docker run hello-world” to verify Docker running:

```
PS C:\Users\Gopika S\OneDrive\Desktop\cloud-docker-app> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

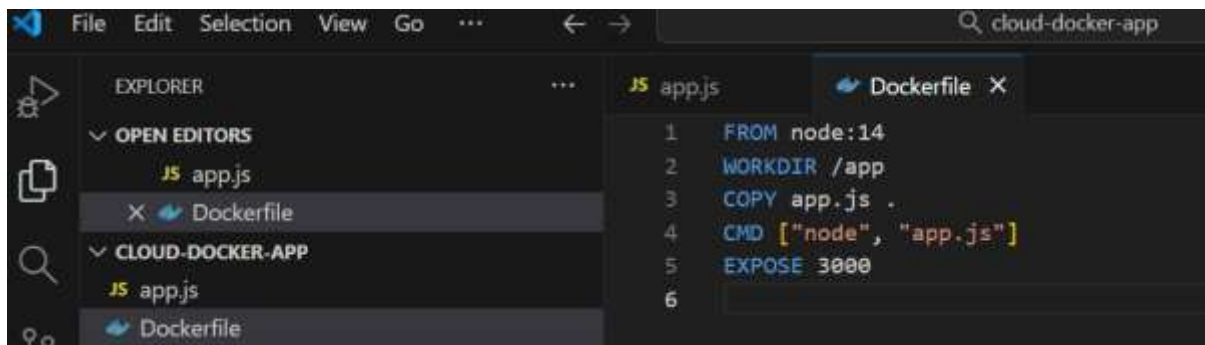
Step 2: Create Application Files

1. Create a working directory cloud-docker-app and add two files:

a) app.js

```
1 const http = require('http');
2 const port = 3000;
3 const server = http.createServer((req, res) => {
4   res.statusCode = 200;
5   res.setHeader('Content-Type', 'text/plain');
6   res.end('Hello from Docker on Windows from gopika!');
7 });
8 server.listen(port, () => {
9   console.log(`Server running at http://localhost:${port}/`);
10 });
11 |
```

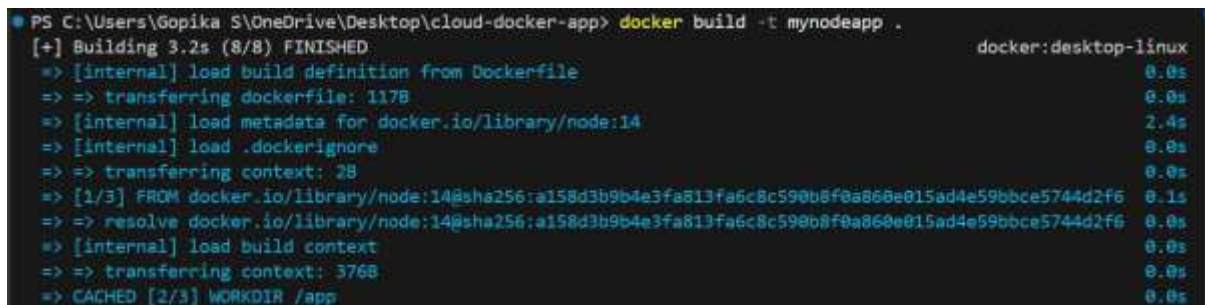
b) Dockerfile



Step 3: Build Docker Image

1. Execute the following command:

`docker build -t mynodeapp.`



Step 4: Run the Container

1. Use the following command to run the app:

`docker run -p 3001:3001 mynodeapp`



Step 5: View the Output

1. Open any web browser and visit

<http://localhost:3000>



Step 6: Check and Manage Containers

1. To list active containers:

```
PS C:\Users\Gopika S\OneDrive\Desktop\cloud-docker-app> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
486ef40bf789	mynodeapp	"docker-entrypoint.s..."	About a minute ago	Up 59 seconds	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
96a5082433b2	d3c1c5c7336e	"docker-entrypoint.s..."	9 minutes ago	Up 9 minutes	0.0.0.0:3001->3001/tcp, [::]:3001->3001/tcp

2. To stop a container:

```
PS C:\Users\Gopika S\OneDrive\Desktop\cloud-docker-app> docker stop 96a5082433b2
96a5082433b2
PS C:\Users\Gopika S\OneDrive\Desktop\cloud-docker-app> docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
486ef40bf789	mynodeapp	"docker-entrypoint.s..."	About a minute ago	Up About a minute	0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
96a5082433b2	d3c1c5c7336e	"docker-entrypoint.s..."	9 minutes ago	Exited (0) 9 minutes ago	

CONCLUSION

Through this experiment, Docker was successfully installed, configured, and used to deploy a simple App.js web application. The message *"Hello from Docker on Windows from Gopika!"* was displayed in the browser, proving that the containerized environment works as expected.

This experiment demonstrates Docker's efficiency in creating isolated, consistent, and portable application environments suitable for modern cloud-based development.

2) TITLE:

Implementation of Word Count Program using Hadoop MapReduce.

INTRODUCTION

Hadoop is an open-source framework for distributed storage and parallel processing of large-scale datasets across clusters of computers using simple programming models.

It includes two main components - **HDFS (Hadoop Distributed File System)** for storage and **MapReduce** for computation. This experiment demonstrates the **Word Count** program using Hadoop, which counts the occurrence of each word in a text file - a fundamental example of distributed computing.

PURPOSE OF THE TOOL

The purpose of using Hadoop is to process and analyze massive amounts of data efficiently in a distributed environment.

It provides:

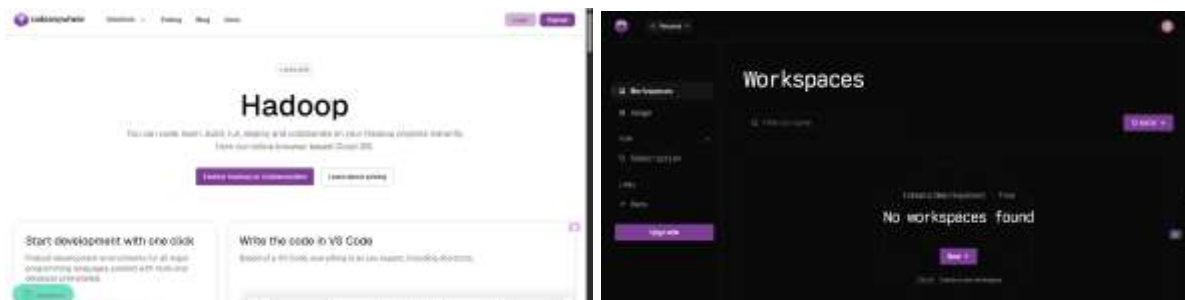
- **Scalability:** Data can be processed across multiple nodes.
- **Fault Tolerance:** Replicates data to handle node failures.
- **Parallel Processing:** Uses the MapReduce model for distributed computation.
- **Cost Efficiency:** Runs on commodity hardware.

The **Word Count** program demonstrates Hadoop's core functionality - splitting data, mapping it into key-value pairs, and reducing results to generate output.

DEPLOYMENT

Step 1: Opening the IDE (CodeAnywhere)

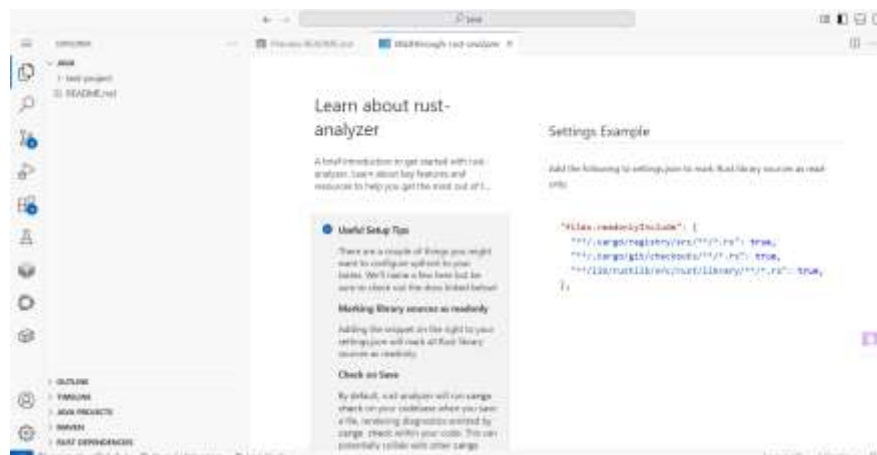
1. Login to your CodeAnywhere account.



2. Open a new **workspace** named java.



3. Launch the terminal inside the workspace and checking Ubuntu.



```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL CODEANYWHERE PORTS

codeany →/workspaces/java (main) $ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 22.04.5 LTS
Release:      22.04
Codename:     jammy
```

Step 2: Installing JDK 8

1. Install Java Development Kit (JDK 8), required for Hadoop:

```
codeany →/workspaces/java (main) $ sudo apt update
sudo apt install openjdk-8-jdk -y
Hit:1 https://download.docker.com/linux/ubuntu jammy InRelease
Hit:2 https://dl.yarnpkg.com/debian stable InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:6 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Step 3: Installing Hadoop

1. Download and extract Hadoop:

```
codeany →/workspaces/java (main) $ wget https://downloads.apache.org/hadoop/common/hadoop-3.4.2/hadoop-3.4.2.tar.gz
tar -xvzf hadoop-3.4.2.tar.gz
sudo mv hadoop-3.4.2 /usr/local/hadoop
```

2. Extracting the tar.gz file:

```
codeany →/workspaces/java (main) $ tar -xvzf hadoop-3.4.2.tar.gz
```

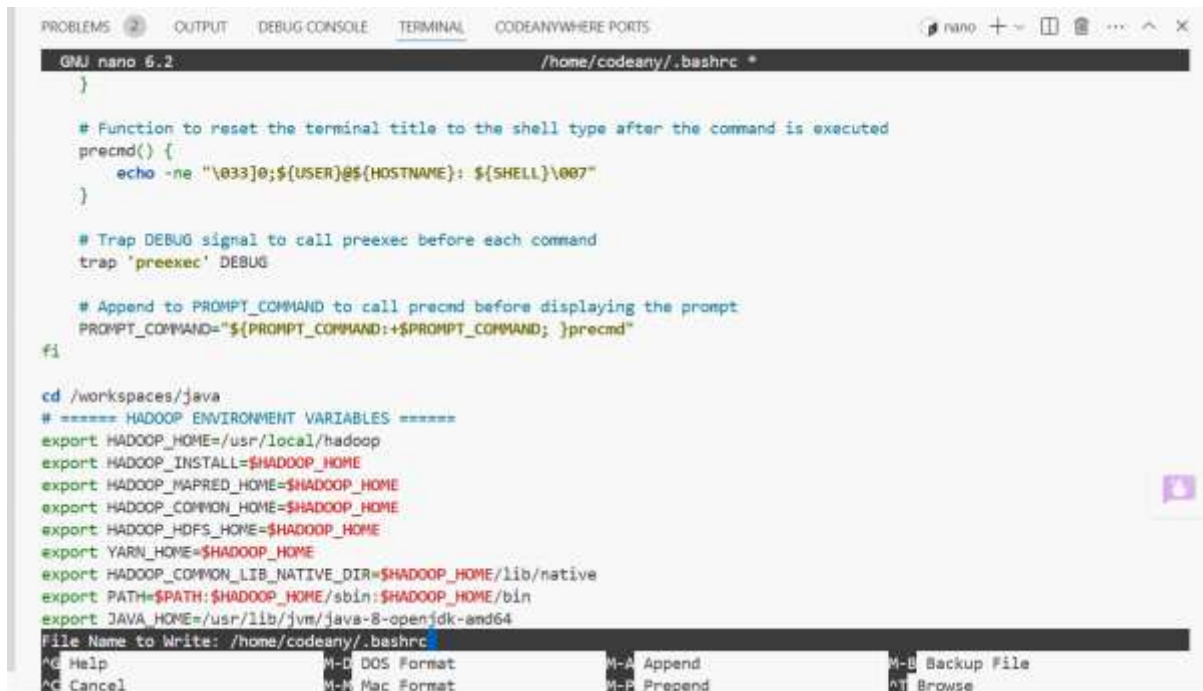

3. Moving the extracted file to Hadoop folder in local storage:

```
codeany → /workspaces/java (main) $ sudo mv hadoop-3.4.2 /usr/local/hadoop
```

Step 4: Setting Environment Variables

```
codeany → /workspaces/java (main) $ nano ~/.bashrc
```

1. Add the following lines:



```
GNU nano 6.2 /home/codeany/.bashrc *
}

# Function to reset the terminal title to the shell type after the command is executed
precmd() {
    echo -ne "\033]0;${USER}@${HOSTNAME}: ${SHELL}\007"
}

# Trap DEBUG signal to call preexec before each command
trap 'preexec' DEBUG

# Append to PROMPT_COMMAND to call precmd before displaying the prompt
PROMPT_COMMAND="${PROMPT_COMMAND:+$PROMPT_COMMAND; }precmd"
fi

cd /workspaces/java
# ===== HADOOP ENVIRONMENT VARIABLES =====
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
File Name to Write: /home/codeany/.bashrc
M-D DOS Format M-A Append M-B Backup File
M-C Cancel M-M Mac Format M-P Prepend M-T Browse
```

```
codeany → /workspaces/java (main) $ source ~/.bashrc
```

2. Check Hadoop version:

```
codeany → /workspaces/java (main) $ hadoop version
Hadoop 3.4.2
Source code repository https://github.com/apache/hadoop.git -r 84e8b89ee2ebe6923691205b9e171badde7a495c
Compiled by ahmarsu on 2025-08-20T10:30Z
Compiled on platform linux-x86_64
Compiled with protoc 3.23.4
From source with checksum fa94c67d4b4be021b9e9515c9b0f7b6
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.4.2.jar
```

Step 5: Creating Input Directory and File

1. Create a new input folder and text file:

```
codeany → /workspaces/java (main) $ mkdir ~/input
codeany → /workspaces/java (main) $ echo "Hello Hadoop Hello World from Gopika" > ~/input/file.txt
```


Step 6: Writing the Word Count Program

1. Create WordCount.java:

```
codeany → /workspaces/java (main) $ nano WordCount.java
```

2. Code:



```
GNU nano 6.2
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

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

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
        extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.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);
    }
}
```

Step 7: Compiling and Creating JAR File

1. Compile and package the program:

```
codeany →/workspaces/java (main) $ javac -source 1.8 -target 1.8 -classpath $(hadoop classpath) -d . WordCount.java
warning: [options] bootstrap class path not set in conjunction with -source 8
warning: [options] source value 8 is obsolete and will be removed in a future release
warning: [options] target value 8 is obsolete and will be removed in a future release
warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
4 warnings
```

2. Create the JAR file:

```
codeany →/workspaces/java (main) $ jar cf wc.jar WordCount*.class
```

Step 8: Executing the Word Count Program

1. Run Hadoop:

```
codeany →/workspaces/java (main) $ hadoop jar wc.jar WordCount ~/input ~/output
2025-10-28 03:26:21,876 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java class
e applicable
2025-10-28 03:26:22,324 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool inte
d-execute your application with ToolRunner to remedy this.
2025-10-28 03:26:22,471 INFO input.FileInputFormat: Total input files to process : 1
2025-10-28 03:26:22,532 INFO mapreduce.JobSubmitter: number of splits:1
2025-10-28 03:26:22,719 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local2137223162_0001
2025-10-28 03:26:22,720 INFO mapreduce.JobSubmitter: Executing with tokens: []
2025-10-28 03:26:23,226 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2025-10-28 03:26:23,227 INFO mapreduce.Job: Running job: job_local2137223162_0001
2025-10-28 03:26:23,229 INFO mapred.LocalJobRunner: OutputCommitter set in config null
```

Step 9: Viewing the Output

```
codeany →/workspaces/java (main) $ hdfs dfs -cat ~/output/part-r-00000
2025-10-28 03:27:49,866 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
e applicable
Gopika 1
Hadoop 1
Hello 2
World 1
from 1
```

CONCLUSION

The **Hadoop Word Count** program was successfully executed in **CodeAnywhere** after installing **JDK 8** and **Hadoop 3.4.2**. This experiment demonstrated the Hadoop MapReduce architecture by processing input text and counting the frequency of each word.

It provided hands-on understanding of distributed computation, job submission, and result collection, laying the foundation for advanced big data processing concepts.