**Project blackbucks**

**Real-time Log Analysis Using Hadoop and Spark**

Abstract

Log analysis is a critical aspect of monitoring and maintaining modern infrastructure. This project focuses on real-time log analysis using Apache Hadoop and Apache Spark, leveraging their distributed processing capabilities to analyse large-scale log data efficiently. Using Spark Streaming, logs will be processed dynamically, extracting insights such as error detection, user behaviour patterns, and security threats. The project aims to demonstrate the efficiency of big data frameworks in handling and analysing logs in near real-time.

Strategic Intentions

* Implementing a scalable and distributed architecture for log analysis.
* Utilizing Spark Streaming for real-time data processing.
* Store and manage log data efficiently using HDFS (Hadoop Distributed File System).
* Perform log parsing, filtering, and anomaly detection using advanced analytics.

Dataset

Imported from Kaggle

Data set name - Web Server Logs

I am using Google Colab for Real-time Log Analysis with Hadoop and Spark

* **Free and Easy Setup:** Google Colab provides a free cloud-based environment with no installation required, so you can quickly start working with big data tools like Spark without worrying about local system configurations.
* **Pre-installed Libraries:** Colab supports Python and allows easy installation of packages (like PySpark), making it convenient to experiment with Hadoop and Spark frameworks.
* **Powerful Cloud Resources:** Colab offers access to CPUs and GPUs on Google’s servers, which can speed up processing compared to some local machines.
* **Integration with Google Drive:** You can easily upload, store, and manage your datasets using Google Drive, simplifying data handling.

Technical Framework

* Apache Hadoop – Distributed storage and processing.
* Apache Spark – Fast, scalable big data computation framework.
* Spark Streaming – Real-time stream processing module.
* HDFS – File storage for structured and unstructured logs.

Source Code

First install the java because it is Required for both Hadoop & Spark.

* Install Java

!apt-get install openjdk-11-jdk-headless -qq > /dev/null

* Download Spark 3.3.2 with Hadoop 3

!wget -O spark-3.3.2-bin-hadoop3.tgz <https://archive.apache.org/dist/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz>

* Extract Spark

!tar -xzf spark-3.3.2-bin-hadoop3.tgz

* Install findspark to integrate with Colab Python

!pip install -q findspark

import os

import findspark

* Set environment variables

os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"

os.environ["SPARK\_HOME"] = "/content/spark-3.3.2-bin-hadoop3"

* Initialize findspark

findspark.init()

* Create Spark session

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("LogAnalysis").getOrCreate()

print("✅ Spark session started successfully")

output :

Spark session started successfully

* uploading the data set imported from the Kaggle

from google.colab import files

uploaded = files.upload()

* zip file (dataset) path

"C:\Users\aashi\Downloads\archive (2).zip"

import zipfile

import os

* Update with actual uploaded filename

zip\_path = "/content/archive (2).zip"

extract\_path = "/content/logdata"

* Unzip the file

with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:

    zip\_ref.extractall(extract\_path)

* List extracted files

for root, dirs, files in os.walk(extract\_path):

    for file in files:

        print(os.path.join(root, file))

* Output :

/content/logdata/web-server-access-logs\_10k.log

* Download Spark 3.5.5

!wget https://downloads.apache.org/spark/spark-3.5.5/spark-3.5.5-bin-hadoop3.tgz

* Extract the downloaded file

!tar xf spark-3.5.5-bin-hadoop3.tgz

* Install OpenJDK 8

!apt-get install openjdk-8-jdk-headless -qq > /dev/null

* Install findspark for Spark integration

!pip install -q findspark

import os

* Set environment variables for Java and Spark

os.environ["JAVA\_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK\_HOME"] = "/content/spark-3.5.5-bin-hadoop3"

import findspark

findspark.init()

from pyspark.sql import SparkSession

* Initialize Spark session

spark = SparkSession.builder \

    .appName("RealTimeLogAnalysis") \

    .getOrCreate()

import os

input\_file = "/content/logdata/web-server-access-logs\_10k.log"

stream\_folder = "/content/logdata\_stream"

os.makedirs(stream\_folder, exist\_ok=True)

chunk\_size =

with open(input\_file, "r") as infile:

    lines = infile.readlines()

for i in range(0, len(lines), chunk\_size):

    chunk = lines[i:i+chunk\_size]

    with open(f"{stream\_folder}/log\_part\_{i//chunk\_size}.log", "w") as outfile:

        outfile.writelines(chunk)

from pyspark.sql.functions import regexp\_extract, col, when

* Read new files as streaming text lines

logs\_stream = spark.readStream.text(stream\_folder)

* Regex pattern for parsing Apache log format

pattern = r'(\S+) (\S+) (\S+) \[(.\*?)\] "(.\*?)" (\d{3}) (\S+)'

* Parse log lines into structured columns

logs\_parsed = logs\_stream.select(

    regexp\_extract("value", pattern, 1).alias("ip"),

    regexp\_extract("value", pattern, 2).alias("client"),

    regexp\_extract("value", pattern, 3).alias("user"),

    regexp\_extract("value", pattern, 4).alias("timestamp"),

    regexp\_extract("value", pattern, 5).alias("request"),

    regexp\_extract("value", pattern, 6).cast("integer").alias("status"),

    regexp\_extract("value", pattern, 7).alias("size")

)

* Convert 'size' column from string to integer, handle '-'

logs\_parsed = logs\_parsed.withColumn(

    "size",

    when(col("size") == "-", 0).otherwise(col("size").cast("integer"))

)

status\_counts = logs\_parsed.groupBy("status").count()

import shutil

shutil.rmtree("/content/logdata\_stream")

os.makedirs("/content/logdata\_stream", exist\_ok=True)

query = status\_counts.writeStream \

    .outputMode("complete") \

    .format("console") \

    .start()

import time

import shutil

input\_file = "/content/logdata/web-server-access-logs\_10k.log"

stream\_folder = "/content/logdata\_stream"

chunk\_size = 500

with open(input\_file, "r") as infile:

    lines = infile.readlines()

for i in range(0, len(lines), chunk\_size):

    chunk = lines[i:i+chunk\_size]

    file\_path = f"{stream\_folder}/log\_part\_{i//chunk\_size}.log"

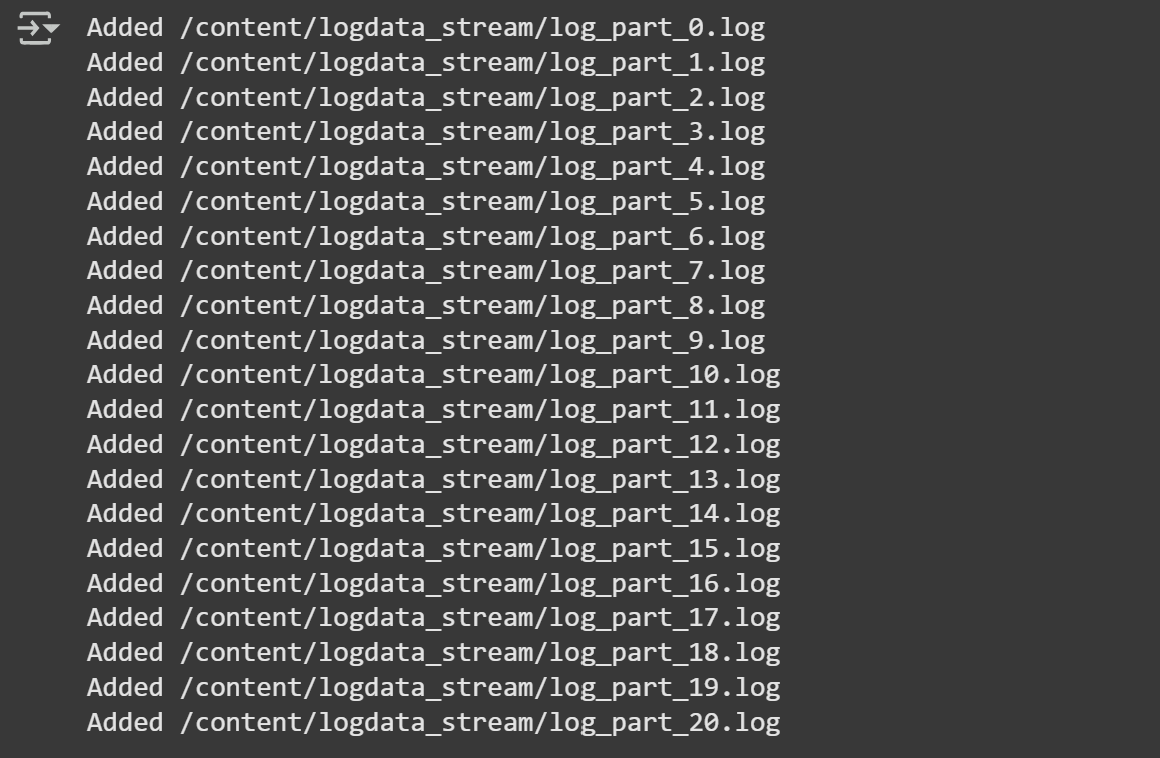
    with open(file\_path, "w") as outfile:

        outfile.writelines(chunk)

    print(f"Added {file\_path}")

    time.sleep(5)

* wait 5 seconds before adding next chunk
* Output:



query.stop()

import matplotlib.pyplot as plt

status\_codes = [200, 404, 500]

counts = [1450, 50, 11]

plt.bar(status\_codes, counts, color=['green', 'orange', 'red'])

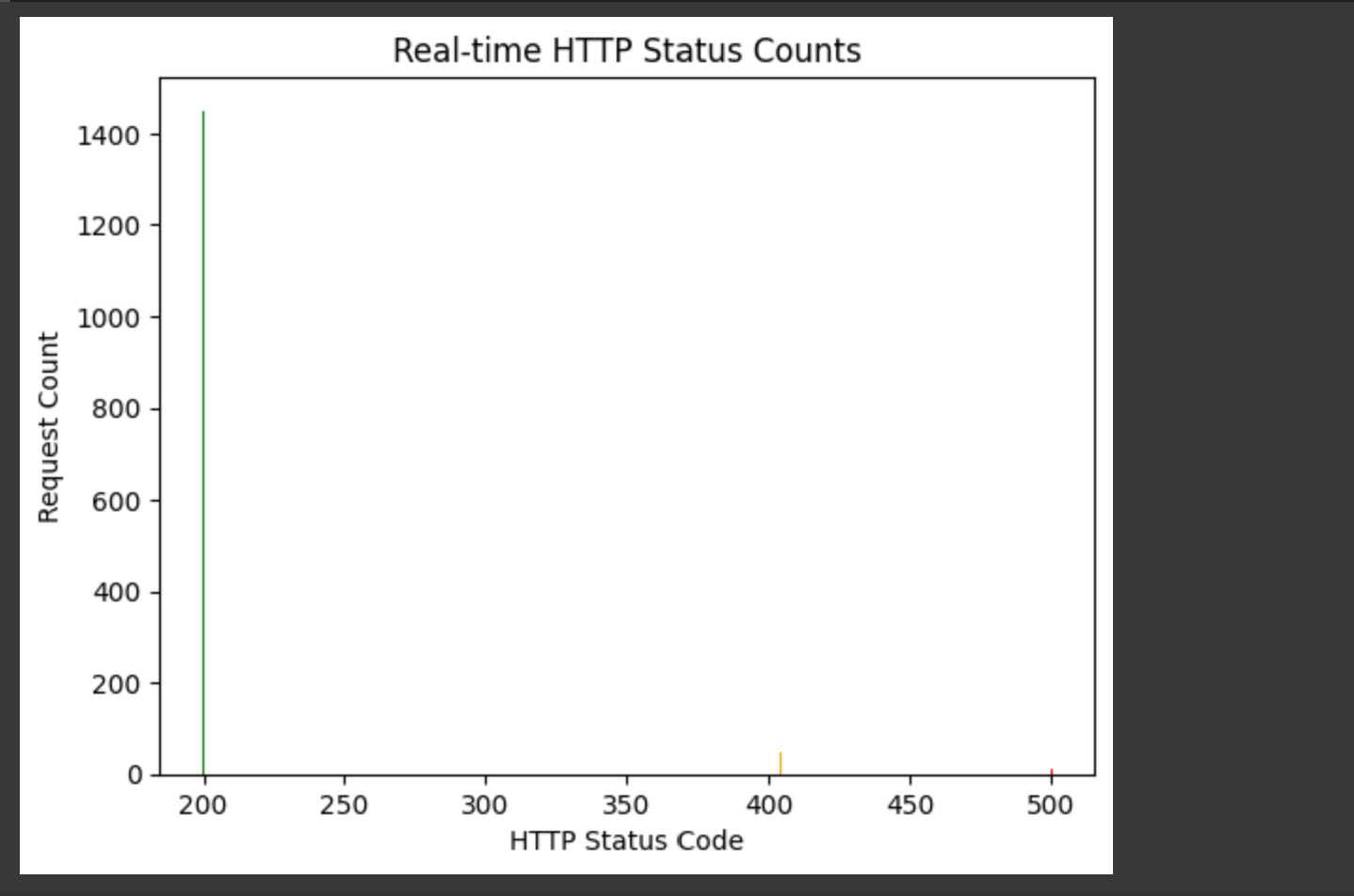
plt.xlabel('HTTP Status Code')

plt.ylabel('Request Count')

plt.title('Real-time HTTP Status Counts')

plt.show()

* Output:



It shows visually how many responses were successful (200).

Core Capabilities

* Real-time Log Processing – Instant analysis and insights.
* Distributed Computing – Handles large datasets efficiently.
* Scalable Framework – Easily adaptable for various log types.
* Error & Anomaly Detection – Identifies patterns and security threats.
* User-friendly Visualizations – Supports integration with Grafana/Kibana for monitoring.

Snapshot

Imagine a company’s website is getting millions of visits per day. Some users are legit, some are bots, and some are potential hackers trying to breach security. The system generates huge logs records of every activity, every click, every error. But manually sifting through all that data? **Impossible.**

That’s where **Real-time Log Analysis Using Hadoop and Spark** **project** comes in.

This project a real-time log analysis system powered by Hadoop and Spark, the rockstars of big data processing. The system takes in endless logs, filters out the noise, detects unusual behaviour, and provides actionable insights—all in real time.

* Hadoop stores vast amounts of log data efficiently.
* Spark Streaming processes logs dynamically, flagging errors, unusual spikes, and security threats instantly.
* You can integrate Kafka to continuously ingest logs or use visualization tools like Kibana to make sense of your findings.

This project is a time saver and good for companies handling massive amounts of data. IT admins, security experts, and data analysts could use it to spot issues before they turn into disasters—like cyberattacks, system failures, or user traffic anomalies.

**Thankyou…!!**

Aashish Bagi

22BCE9776