What is a Web Server Log?

A web server log entry

127.0.0.1 - Jake [10/Oct/2021:13:55:36 -0700] "GET /monkey_pb.gif HTTP/1.0" 200 2326

In this illustration, a client with the IP address 127.0.0.1, presumably Jake, requested the file /monkey_pb.gif at a specific time and date.

The server's response was a 200 status code (reflecting success), and it transmitted a file of 2326 bytes.

Another example of a web server log's entry could be:

192.0.2.1 - - [07/Dec/2021:11:45:26 -0700] "GET /index.html HTTP/1.1" 200 4310

This excerpt shows a user, with the IP address 192.0.2.1, triggered a request on 7th December 2021 at 11:45:26.

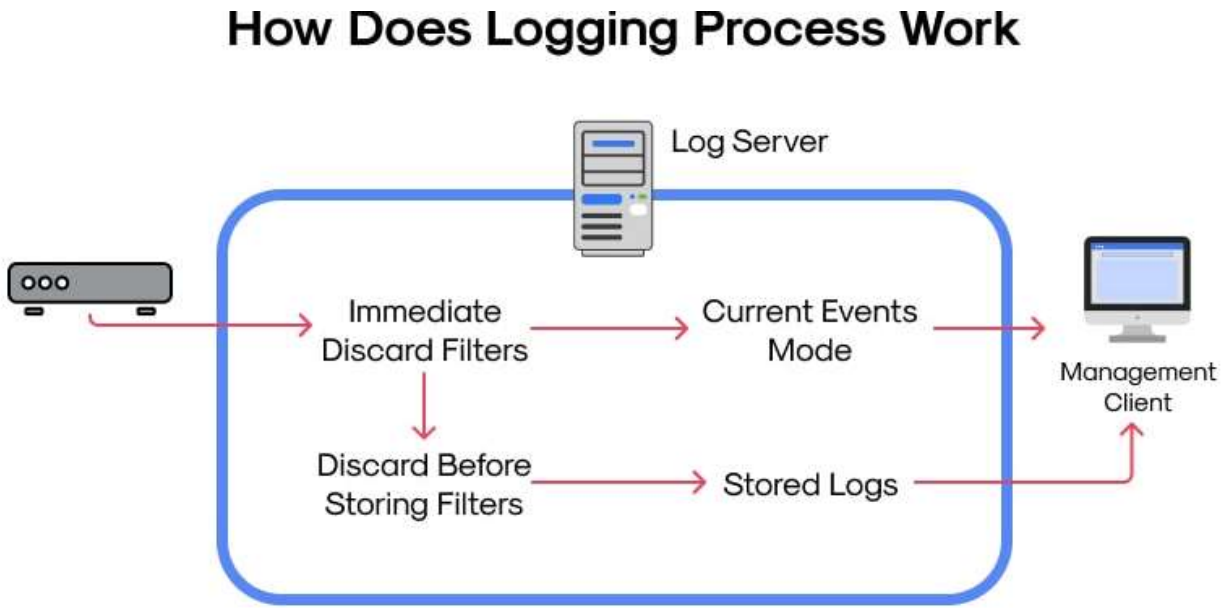
The requested page was 'index.html', using the HTTP/1.1 protocol.

The server successfully engaged with the request (status code: 200), responding with 4310 bytes.

Web server logs are generally maintained in an easily decipherable text manner. But their size can become extensive, particularly for heavily trafficked sites necessitating parsing and analyzing applications or tools.

What is a Web Server Log?

- This ledger is akin to the server's event history, documenting every request handled, every document delivered, and every glitch faced.
- **Access Logs**: Logging all the requests handled by the server. It reveals who accessed what, when, and in what manner.
- **Error Logs**: Logging encountered server errors. These can be key to diagnosing and identifying issues within the server or the site.
- **Security Logs**: Logging all security-associated incidents, like unsuccessful login attempts or doubtful activity. They are vital tools in upholding the server and the site's security robustness.



Web Server Log Analysis using Hadoop (MapReduce, YARN, and HDFS)

Example: A large e-commerce company runs hundreds of web servers that continuously generate logs containing user requests, IP addresses, timestamps, URLs accessed, and HTTP status codes. These logs need to be processed daily to:

- Detect abnormal traffic patterns (e.g., from suspicious IPs).
- Count failed login attempts.
- Identify pages with the highest 404 errors. (Someone tried to access a protected resource (e.g., /admin, /login) - resource not found)
- Archive processed logs for compliance.

Write a Hadoop-based solution for the following using the Hadoop ecosystem with some constraints .

- **Identify IP addresses with repeated *401 Unauthorized login attempts*.**
- **Use MapReduce to: Extract IP addresses with 401 status codes from each log entry.**
- **Count the number of such failed attempts per IP.**
- **Use YARN to manage resources and execute the MapReduce job on the cluster.**
- **Write the results back to HDFS in the path:/user/logs/output/failed_login_summary/ Format: IP_address <tab> count**

Constraints:

- **Minimum Record Volume**: The log file must contain at least 100,000 lines to simulate real-world batch processing. Test on a sufficiently large dataset.
- **Mapper and Reducer Optimization**:
  - Avoid using in-memory data structures that can cause memory overflow.
  - Your solution should be scalable and efficient.
- **Output Filtering**:
  Only write IPs with more than 5 failed login attempts to the final output.
- **Handling Malformed Logs**:
  Some log lines may be incomplete or corrupted. Your Mapper should skip malformed records without crashing.
- **Job Tracking**:
  Use Hadoop's counters or logs to print the total number of valid and skipped log entries (optional).

Component	Role
HDFS	Stores input and output log files
MapReduce	Filters and counts failed login attempts by IP
YARN	Executes the job across the cluster

Solution

Upload Log File to HDFS

Assume your local file is access.log. **First, upload it to HDFS**.

```
# Create input directory in HDFS (if not already created)
hdfs dfs -mkdir -p /user/logs/input
# Upload the log file to HDFS
hdfs dfs -put access.log /user/logs/input/
```

Write the Mapper Script — mapper.py

```
import sys # When writing MapReduce scripts in Python for Hadoop Streaming, we use sys.stdin and sys.stdout to read and write data.
# Process each line from standard input (provided by Hadoop Streaming)
for line in sys.stdin:
    parts = line.strip().split()
    # Skip malformed lines (do not follow the expected format )
    if len(parts) < 9:
        continue
    ip = parts[0]            # First column is IP address
    status = parts[-2]       # Second-to-last column is HTTP status code
    # Filter: only emit IPs with 401 Unauthorized
    if status == "401":
        print(f"{ip}\t1")    # Emit key-value pair: (IP, 1)
```

Save as: mapper.py

Make it executable: chmod +x mapper.py

Write the Reducer Script — reducer.py

```
import sys
current_ip = None
failed_count = 0
# Read each key-value pair from standard input
for line in sys.stdin:
    ip, count = line.strip().split("\t")
    count = int(count)
    # If we are still on the same IP, accumulate the count
    if ip == current_ip:
        failed_count += count
    else:
        # Output previous IP if failed attempts > 5
        if current_ip and failed_count > 5:
            print(f"{current_ip}\t{failed_count}")
        # Reset counters for the new IP
        current_ip = ip
        failed_count = count
# Output the last IP
if current_ip and failed_count > 5:
    print(f"{current_ip}\t{failed_count}")
```

Save as: reducer.py

Make it executable: chmod +x reducer.py

chmod: stands for change mode, used to modify file permissions.

+x: adds execute permission.

reducer.py: target file you're modifying.

Run MapReduce Job Using Hadoop Streaming

Use the following command to launch your job on YARN with Hadoop Streaming:

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
    -D mapreduce.job.name="FailedLoginIPCounter" \
    -D mapreduce.job.reduces=1 \
    -input /user/logs/input/access.log \
    -output /user/logs/output/failed_login_summary \
    -mapper mapper.py \
    -reducer reducer.py \
    -file mapper.py \
    -file reducer.py
```

Make sure the output directory doesn't already exist, or delete it:

```
hdfs dfs -rm -r /user/logs/output/failed_login_summary
```

Check Output in HDFS

View output file content

```
hdfs dfs -cat
/user/logs/output/failed_login_summary/part-00000
```

Sample Input and Output

Input (access.log)

```
192.168.1.10 - - [24/Apr/2025:13:55:36] "POST /login HTTP/1.1" 401 -
192.168.1.10 - - [24/Apr/2025:13:56:36] "POST /login HTTP/1.1" 401 -
192.168.1.10 - - [24/Apr/2025:13:57:36] "POST /login HTTP/1.1" 401 -
192.168.1.10 - - [24/Apr/2025:13:58:36] "POST /login HTTP/1.1" 401 -
192.168.1.10 - - [24/Apr/2025:13:59:36] "POST /login HTTP/1.1" 401 -
192.168.1.10 - - [24/Apr/2025:14:00:36] "POST /login HTTP/1.1" 401 -
10.0.0.1 - - [24/Apr/2025:14:01:36] "GET /admin HTTP/1.1" 404 -
```

Output:

```
192.168.1.10 6
```

Component	Role
HDFS	Stores input and output log files
MapReduce	Filters and counts failed login attempts by IP
YARN	Executes the job across the cluster

Project Structure or pipeline

web_log_analysis/

- **mapper.py**
- **reducer.py**
- **README.txt**
- **access.log (sample file for testing)**
- **run_job.sh**

---

1. mapper.py

(As already discussed in pervious slides — **extracts IPs with status 401**)

2. reducer.py

(As already discussed in pervious slides — **aggregates counts per IP, filters > 5**)

3. README.txt

Web Server Log Security Analysis using Hadoop

Goal: Analyze Apache web logs to detect IP addresses with more than 5 failed login attempts (HTTP 401).

Instructions:

1. **Upload** access.log to HDFS: *hdfs dfs -mkdir -p /user/logs/input*
   hdfs dfs -put access.log /user/logs/input/
2. **Run the job** using the provided shell script:
   ./run_job.sh
3. **Check output:**
   hdfs dfs -cat /user/logs/output/failed_login_summary/part-00000

4. access.log

A sample log file like:

```
192.168.1.10 - - [24/Apr/2025:13:55:36] "POST /login HTTP/1.1" 401 -
172.16.0.5 - - [24/Apr/2025:13:56:01] "GET /home HTTP/1.1" 200 -
192.168.1.10 - - [24/Apr/2025:13:56:40] "POST /login HTTP/1.1" 401 -
10.0.0.3 - - [24/Apr/2025:13:57:12] "GET /admin HTTP/1.1" 404 –
```

5. **run_job.sh**

```
# Hadoop Streaming job using YARN
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
  -D mapreduce.job.name="FailedLoginDetection" \
  -files mapper.py,reducer.py \
  -mapper mapper.py \
  -reducer reducer.py \
  -input /user/logs/input/access.log \
  -output /user/logs/output/failed_login_summary
echo "Job submitted. Output:"
hdfs dfs -cat /user/logs/output/failed_login_summary/part-00000
```

Make it executable:

chmod +x run_job.sh (a Bash script in this case)