



Logging & Log Management



Objective

- Understand the importance of logs in troubleshooting, security, and system monitoring.
- Learn about centralized log management, log rotation, and disk space optimization.
- Explore the ELK stack for collecting, storing, and visualizing logs.
- Implement structured logging and understand log levels for application monitoring.
- Parse and analyze logs using Logstash, Fluentd, and Kibana for event correlation.
- Understand distributed logging and tracing across microservices.





**Explaining why logs are
critical for
troubleshooting,
auditing, security
monitoring, and
performance analysis.**

Let's see

1. Troubleshooting

- Root cause identification
- Step-by-step tracing
- Error replication

2. Auditing

- Accountability
- Change tracking
- Regulatory compliance



Let's see

3. Security Monitoring

- Threat detection
- Forensics
- Intrusion detection systems (IDS)

4. Performance Analysis

- Bottleneck identification
- Usage trends
- Service level monitoring





Discussing different types of logs

Let's discuss

- **System Logs:** Capture events from the kernel, operating system, and services, like startup messages, crashes, or hardware issues.
- **Application Logs:** Record application-specific events such as errors, user requests, and transactions to track app behavior and issues.
- **Access Logs:** Track user activity, security events, and API traffic, helping monitor usage patterns and detect unauthorized access.





**Explaining why
centralized logging is
essential for scalability
and troubleshooting.**

Let's see

Centralized logging is essential for scalability and troubleshooting because it consolidates logs from multiple systems, applications, and environments into a single platform.

- This unified view simplifies log management as systems grow, allowing teams to efficiently handle large volumes of log data.
- It also accelerates troubleshooting by enabling quick searches, correlation of events across services, and detection of root causes, reducing downtime and improving system reliability.





**Discussing challenges of
managing logs in
distributed environments
(log rotation, disk bloat,
retrieval issues).**

Let's discuss

Managing logs in distributed environments is challenging due to:

- **Log rotation:** Ensuring old logs are archived or deleted to prevent overflow.
- **Disk bloat:** Logs can quickly consume storage across multiple nodes if not managed.
- **Retrieval issues:** Accessing and correlating logs across systems can be complex and time-consuming without centralization.





**Introducing log rotation
tools (Logrotate) to
prevent excessive disk
usage.**

Log rotation tools

- Log rotation tools like Logrotate are used to manage log files by automatically rotating, compressing, and removing old logs.
- This prevents logs from growing indefinitely and consuming excessive disk space, helping maintain system performance and reliability.





Explaining the log processing pipeline in the ELK stack.

Let's see

The ELK stack processes logs through this pipeline:

- **Logstash:** Collects logs from various sources, parses, filters, and transforms them into structured data.
- **Elasticsearch:** Stores and indexes the structured logs, enabling fast search and retrieval.
- **Kibana:** Visualizes and analyzes the logs through dashboards and queries, helping with monitoring and troubleshooting.





**Providing a quick
overview of setting up
ELK for log management.**

Let's see

Here's a quick overview of setting up the ELK stack for log management:

- Install Elasticsearch
- Install Logstash
- Install Kibana
- Configure Inputs and Filters
- Secure and Monitor





Take A 5-Minute Break!



- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes





Explaining logging best practices in applications.

Let's see

Logging best practices in applications include:

- Log meaningful messages
- Use log levels
- Avoid sensitive data
- Structure logs
- Include timestamps and IDs
- Handle exceptions properly
- Avoid excessive logging





Discussing log levels and when to use them

Let's discuss

Log levels help categorize log messages:

- **DEBUG:** Detailed logs for developers during development.
- **INFO:** High-level logs for normal operations.
- **WARN:** Logs for non-critical issues that don't affect functionality.
- **ERROR:** Logs for critical failures that impact the system's functionality.





**Demonstrating
structured logging using
JSON/XML formats for
easier parsing**

Let's do it

Structured logging can be done using formats like JSON or XML for easier parsing:

- JSON Example:

```
{  
    "timestamp": "2025-04-29T10:00:00Z",  
    "level": "ERROR",  
    "message": "Database connection failed",  
    "user_id": 12345,  
    "request_id": "abc-123"  
}
```



Let's do it

- XML Example:

```
<log>
  <timestamp>2025-04-29T10:00:00Z</timestamp>
  <level>ERROR</level>
  <message>Database connection failed</message>
  <user_id>12345</user_id>
  <request_id>abc-123</request_id>
</log>
```



Explaining how to parse unstructured logs for analysis

Let's see

Parsing unstructured logs for analysis involves extracting meaningful information from raw, free-form log entries. Here's how you can do it:

- Identify Patterns
- Use Regular Expressions (Regex)
- Leverage Tools
- Define Filters
- Test & Refine





**Demonstrating using
Logstash or Fluentd for
log parsing.**

Let's do it

Using Logstash for log parsing:

1. Install Logstash and create a configuration file (logstash.conf):

```
input {
  file {
    path => "/path/to/logfile.log"
    start_position => "beginning"
  }
}

filter {
  grok {
    match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:level} %{GREEDYDATA:log_message}" }
  }
}

output {
  stdout { codec => rubydebug }
}
```



2. Run Logstash with the config:

```
bin/logstash -f logstash.conf
```

Using Fluentd for log parsing:

1. Install Fluentd and configure the input/output:

```
<source>
  @type tail
  path /path/to/logfile.log
  pos_file /var/log/td-agent/td-agent.log.pos
  tag my.logs
</source>

<filter my.logs>
  @type parser
  format /(?<timestamp>.+) (?<level>\w+) (?<log_message>.+)/
</filter>

<match my.logs>
  @type stdout
</match>
```

2. Run Fluentd:

```
td-agent -c fluentd.conf
```





**Introducing Kibana's
filtering and searching
capabilities for log
analysis.**

Let's see

Kibana offers powerful filtering and searching for log analysis:

- **Search Bar:** Allows querying logs using Lucene or KQL syntax.
- **Filters:** Apply filters to narrow results by fields like log level or user ID.
- **Time Range:** Filter logs by specific time frames.
- **Visualizations:** Create charts and graphs to analyze log data trends.
- **Saved Searches:** Save frequent queries for quick access.





Explaining challenges of logging in microservices (multiple services, different log formats).

Let's see

Logging in microservices faces challenges like:

- **Multiple Services:** Each service generates its own logs, making it difficult to track and correlate events across services.
- **Different Log Formats:** Each microservice may use different log formats, complicating centralized log management and analysis.
- **Distributed Nature:** Logs from different services must be aggregated and synchronized for effective troubleshooting and monitoring.
- **Performance:** Excessive logging across many services can impact performance.





**Discussing correlation
IDs for tracking requests
across distributed
systems.**

Let's discuss

- Correlation IDs are unique identifiers used to track requests across multiple services in a distributed system.
- They are passed along with each request, allowing you to trace its journey through various services.
- By logging these IDs in each service, you can correlate logs, debug issues, and monitor performance across the entire system, making it easier to troubleshoot and optimize.





**Introducing
Jaeger/Zipkin for
distributed tracing to
track request lifecycles.**

Let's see

- Jaeger and Zipkin are popular tools for distributed tracing, helping track request lifecycles across microservices.
- They capture trace data, showing the path of a request through services, and measure latency between them.
- By visualizing these traces, you can identify bottlenecks, debug performance issues, and understand the flow of requests in complex distributed systems.





Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistants



Thanks



!

