Logging Best Practices and ELK Stack Integration

Amo Samuel

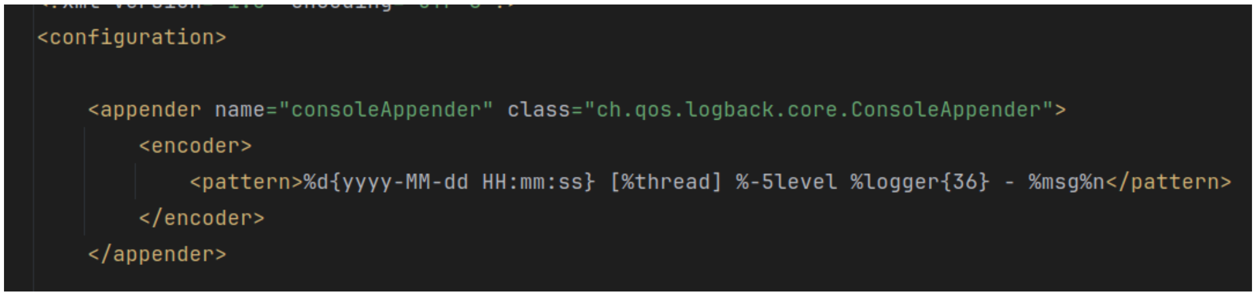
Introduction

In modern applications, especially when working with microservices or distributed systems, logging becomes critical for monitoring, debugging, and tracing system behavior. Effective logging practices can help identify issues faster and provide valuable insights into application performance. I integrated logging using **Logback** and connected it to the **ELK Stack** (Elasticsearch, Logstash, Kibana) for better log management. Here's how I approached the entire process, adhering to some important logging best practices along the way.

Logback Configuration

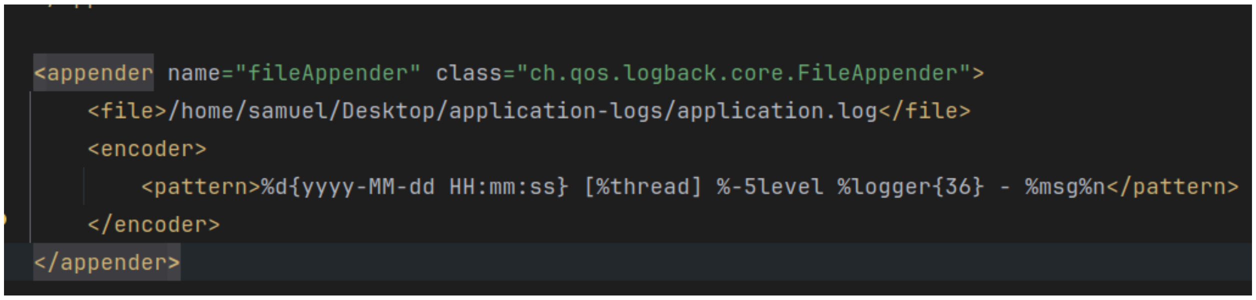
For logging in my application, I opted for **Logback** due to its flexibility and easy integration with Spring Boot. Below are the key aspects of my Logback setup.

Console Appender: I've configured a console appender to print logs to the console during development and troubleshooting. It helps me monitor the logs in real-time while the application is running. By default spring print logs to the console



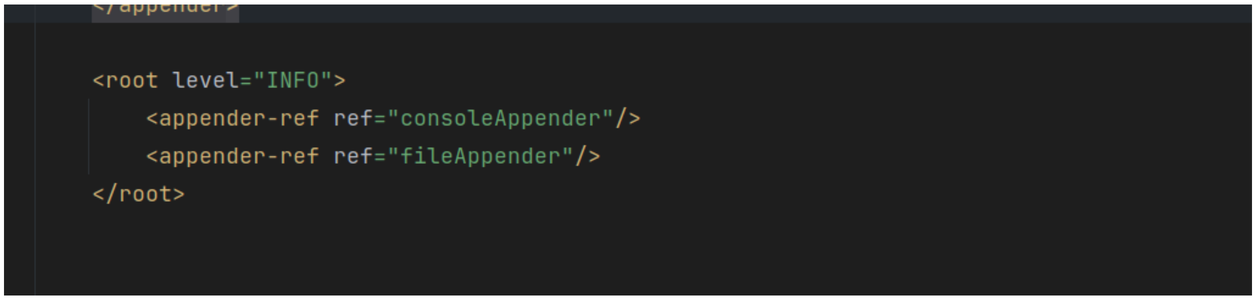
```
<configuration>

    <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>
```

File Appender: To store the logs persistently, I configured a file appender that writes logs to a file on my local machine. This log file is crucial for post-mortem analysis when an issue occurs, and I need to review the logs later.



```
<appender name="fileAppender" class="ch.qos.logback.core.FileAppender">
    <file>/home/samuel/Desktop/application-logs/application.log</file>
    <encoder>
        <pattern>%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
</appender>
```

Root Logger: The root logger has been set to INFO level, I also linked both the console and file appenders to capture logs in real-time and store them



```
    </appender>

    <root level="INFO">
        <appender-ref ref="consoleAppender"/>
        <appender-ref ref="fileAppender"/>
    </root>
```

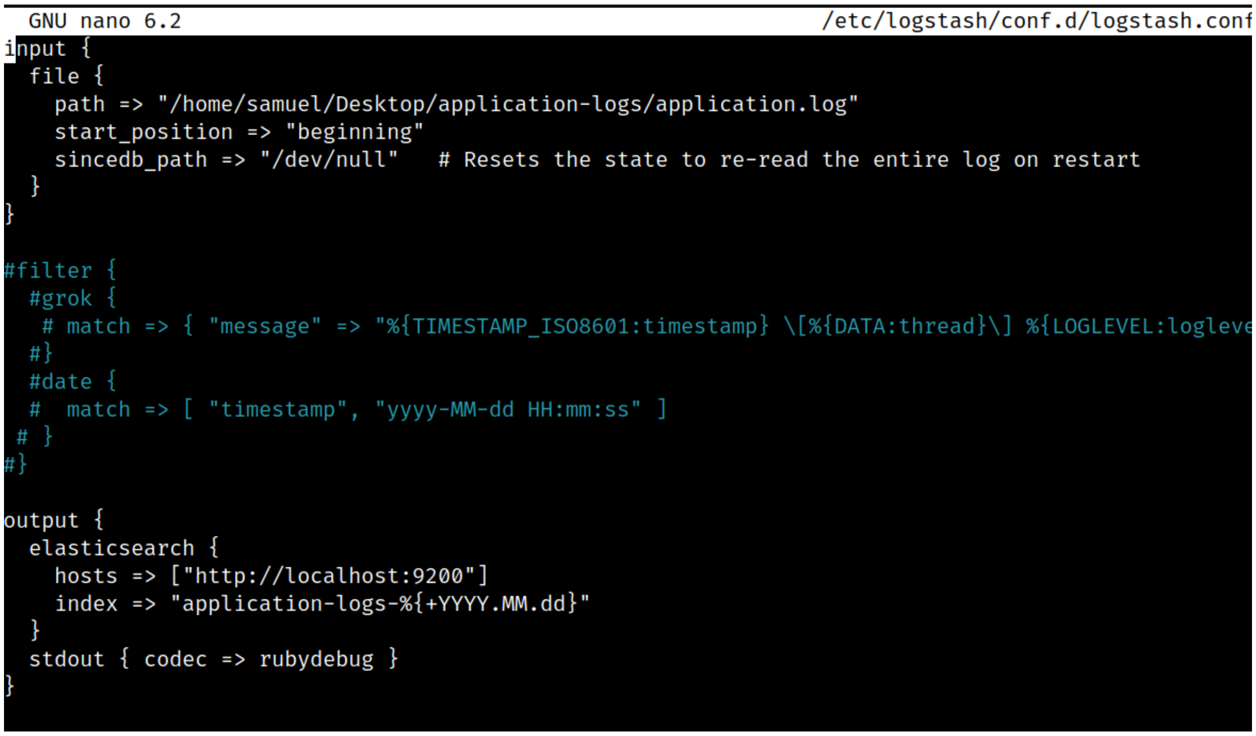
ELK Stack Integration

For centralized log management and real-time monitoring, I integrated my Logback logs with the **ELK Stack**. This integration allows me to store, analyze, and visualize logs effectively.

Logstash Configuration

I configured Logstash to read logs from the file generated by Logback. The input is from the log file, and the output is sent to **Elasticsearch**. A key part of this setup is the creation of daily log indexes in Elasticsearch.

Here's the Logstash configuration that I used:



```
  GNU nano 6.2                                                  /etc/logstash/conf.d/logstash.conf
input {
  file {
    path => "/home/samuel/Desktop/application-logs/application.log"
    start_position => "beginning"
    sincedb_path => "/dev/null"   # Resets the state to re-read the entire log on restart
  }
}

#filter {
  #grok {
   # match => { "message" => "%{TIMESTAMP_ISO8601:timestamp} \[%{DATA:thread}\] %{LOGLEVEL:logleve
  #}
  #date {
  #  match => [ "timestamp", "yyyy-MM-dd HH:mm:ss" ]
 # }
#}

output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "application-logs-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug }
}
```

- **File Input**: I specified the path to my log file and set the start_position to beginning, ensuring that Logstash processes all existing logs on startup.
- **Elasticsearch Output**: The logs are sent to my Elasticsearch instance running on localhost:9200. I used a dynamic naming convention for the log **index** by including a date pattern in the index name (application-logs-%{+YYYY.MM.dd}). This configuration ensures that a new index is created daily, which keeps my logs organized and makes it easier to retrieve logs for specific dates.
    - **Why Date-based Indexing?** Organizing logs in date-based indexes is crucial for scalability and performance. When I want to retrieve logs, especially for debugging or performance monitoring, I can easily target the relevant index for a specific day. It prevents bloating a single index with too much data, which could slow down search operations in Elasticsearch. This approach also allows for the easy deletion of old logs by simply removing outdated indexes.
- **Standard Output**: For debugging purposes, I added a stdout output with a rubydebug codec to visualize the log structure when needed.

Kibana for Visualization

Once the logs are indexed in Elasticsearch, I use **Kibana** to create visualizations and dashboards for real-time log analysis. By organizing logs into daily indexes, it becomes straightforward to query logs for specific periods and generate dashboards that show daily trends or spikes in errors.

Logging Best Practices

As I set up my logging system, I followed a few important best practices to ensure logs are both meaningful and manageable:

Structured and Consistent Logging:

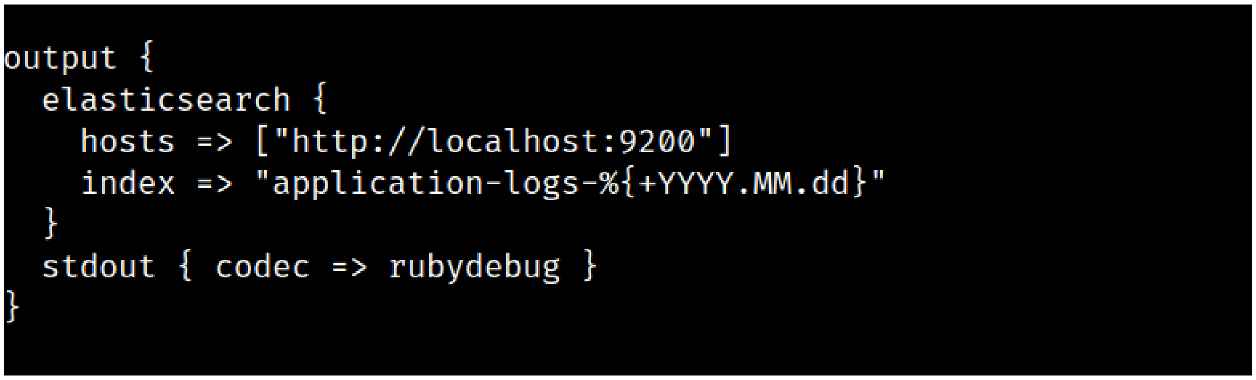
I ensured that all logs follow a consistent pattern. The log pattern includes the timestamp, thread information, log level, logger name, and the actual message. Structured logs make it easier to search and analyze logs later, especially when using tools like **Elasticsearch** or **Kibana**.

Log Levels:

I used appropriate log levels (INFO, DEBUG, ERROR, etc.) to ensure that I'm not logging unnecessary details during normal operation. This reduces noise and keeps logs meaningful. For example, using INFO for routine operations and ERROR for significant issues helps me filter logs more effectively when troubleshooting.

Log Rotation and Indexing:

By creating date-based indexes in **Elasticsearch**, I ensured that logs are segmented by day. This makes it easy to manage large volumes of log data and retrieve logs for specific time periods without performance degradation. It also simplifies the process of archiving or deleting older logs. Each day's logs are stored in a new index, named with a dynamic pattern:



```
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "application-logs-%{+YYYY.MM.dd}"
  }
  stdout { codec => rubydebug }
}
```

Conclusion

Integrating the **ELK Stack** with **Logback** logging has significantly improved my log management capabilities. With the addition of dynamic, date-based indexes, I can manage logs more effectively, ensuring smooth retrieval of logs by day. The indexed logs can be visualized in **Kibana**, providing valuable insights into system health and performance. This approach not only makes the logging process scalable but also keeps the system performant when handling large volumes of log data.

By adhering to logging best practices like **structured and consistent logging**, **appropriate use of log levels**, and **log rotation/indexing**, I ensure that my application is easier to maintain and troubleshoot.