ELK Stack Setup with Spring Boot Microservices (Docker Based)

This document provides a comprehensive guide to:

- **Setting up the ELK (Elasticsearch, Logstash, Kibana) stack using Docker**
- **Configuring Spring Boot microservices to send logs to Logstash**
- **Viewing and filtering logs in Kibana**
- **Visualizing API hit counts per microservice**

PART 1: ELK Stack Docker Setup

1. Docker Compose File

Create a file named docker-compose.yml:

```
description: ELK stack setup using Docker Compose
version: '3.7'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.13
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - ES_JAVA_OPTS=-Xms512m -Xmx512m
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
      - "9200:9200"
    networks:
      - elk

  logstash:
    image: docker.elastic.co/logstash/logstash:7.17.13
```

```
    container_name: logstash
    volumes:
      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
    ports:
      - "5000:5000"
    networks:
      - elk

  kibana:
    image: docker.elastic.co/kibana/kibana:7.17.13
    container_name: kibana
    ports:
      - "5601:5601"
    networks:
      - elk
    environment:
      ELASTICSEARCH_HOSTS: http://elasticsearch:9200

networks:
  elk:
    driver: bridge
```

2. Logstash Configuration (logstash.conf)

```
input {
  tcp {
    port => 5000
    codec => json_lines
  }
}

filter {
  json {
    source => "message"
  }
}

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

```
}
```

3. Run the ELK Stack

```
docker-compose up -d
```

PART 2: Spring Boot Configuration

1. Add Logstash Encoder Dependency in pom.xml

```
<dependency>
  <groupId>net.logstash.logback</groupId>
  <artifactId>logstash-logback-encoder</artifactId>
  <version>7.4</version>
</dependency>
```

2. Configure logback-spring.xml

```
<configuration>
  <appender name="LOGSTASH"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
    <destination>localhost:5000</destination>
    <encoder class="net.logstash.logback.encoder.LogstashEncoder">
      <customFields>{"service":"micro1"}</customFields>
    </encoder>
  </appender>

  <root level="INFO">
    <appender-ref ref="LOGSTASH"/>
  </root>
</configuration>
```

Repeat for each service by changing "service":"micro1" to "micro2", "micro3", etc.

3. Example Log Statement in Controller

```
private static final Logger logger =
LoggerFactory.getLogger(MyController.class);

@GetMapping("/api/hello")
public String hello() {
```

```
    logger.info("API Hit: /api/hello");
    return "Hello";
}
```

PART 3: Kibana UI Setup

1. Access Kibana

Open browser: http://localhost:5601

2. Create Index Pattern

- **Go to Stack Management > Index Patterns**
- **Create new index pattern: springboot-logs-***
- **Select @timestamp as the time field**

3. Discover Logs per Microservice

- **Go to Discover**
- **Filter by service: "micro1", "micro2", etc.**
- **Save each filter as a saved search**

4. Visualize API Hit Count

- **Go to Visualize > Create Visualization > Line Chart**
- **Y-Axis: Count**
- **X-Axis: @timestamp (Date Histogram)**
- **Split Series: Field = service**
- **Save as API Hit Count Per Service**

5. Create Dashboard

- **Go to Dashboard > Create**
- **Add:**
    - **Saved searches per microservice**
    - **Visualizations (e.g., API hits, errors by service)**

OPTIONAL: API Hit Monitoring using Custom Fields

Log structured info like this in each request:

```
logger.info("{\"method\":\"GET\",\"endpoint\":\"/api/hello\",\"status\":200,\"timeMs\":34}");
```

You can then parse these fields in Logstash and use them in Kibana visualizations (e.g., endpoint frequency, response times, status code distribution).

Let me know if you want a PDF or more advanced monitoring examples (like alerts or anomaly detection).