

# LLM(Large Language Model)을 사용한 AI 챗봇 연구

팀 명: ForPaw

부산대학교 정보컴퓨터공학부

202155595 이한홍

201924475 박재홍

202155590 이종일

지도교수: 김호원

# 목차

## 1. 과제 목표

## 2. 요구 사항 및 수정 사항

2.1 로그 관련 제약 사항

2.2 LLM 모델 제약 사항

2.3 데이터베이스 제약 사항

2.4 시스템 리소스 제약 사항

## 3. 설계 상세화

3.1 전체 구성도

3.2 Front-End 설계

3.2 Back-End 설계

3.2 데이터베이스 설계

## 4. 수정된 과제 추진 내역

## 5. 구성원별 진척 현황

## 6. 과제 수행 내역 및 중간 결과

# 1. 과제 목표

본 졸업 과제는 LLM 을 활용하여 로그 데이터를 주기적으로 분석하고, 비정상적인 활동이나 오류를 탐지하는 시스템을 개발하는 것이다. 이 시스템은 클라우드에서 실행되는 다양한 어플리케이션과 인프라의 로그 데이터를 수집, 저장, 분석하여 관리자에게 중요한 정보를 제공하며, 문제가 발생했을 때 빠르게 대응할 수 있도록 지원한다.

- 클라우드에서 실행 중인 Docker 컨테이너들의 로그를 Logstash, FileBeat, Elasticsearch 를 사용해서 로그를 주기적으로 수집한다. 예를 들어, 웹 서버를 운영 중이라면 Spring Boot, React, Nginx, MySQL 등의 어플리케이션이 실행 중인 있는 컨테이너에서 로그를 수집한다.
- Metricbeat, Elasticsearch 를 사용해서 클라우드에서 동작하고 있는 리눅스 인스턴스(EC2 인스턴스)의 성능 지표(CPU 사용량, 메모리 사용량, 디스크 I/O 등)를 주기적으로 수집한다.
- Elasticsearch 에 저장된 로그 데이터를 LLM 을 활용해 분석 및 요약한다. 로그 데이터에서 비정상적인 활동이나 오류를 탐지하며, 이 과정을 통해 시스템 운영 중 발생할 수 있는 문제를 빠르게 발견할 수 있다. LLM 은 로그에서 발생하는 패턴을 학습해 자동으로 중요 이벤트를 탐지하고 요약된 정보를 제공한다.
- 비정상적인 활동이나 오류가 탐지 됐을 때, Slack API 나 Google Mail API 를 사용해 관리자에게 알림을 전송하도록 개발한다.
- 분석/요약된 결과와 수집된 로그 데이터를 웹 페이지를 통해 조회할 수 있도록 개발한다. 사용자는 대시보드를 통해 로그 데이터를 시각적으로 확인할 수 있으며, 특정 컨테이너나 시간 범위에 대한 세부 로그 조회가 가능하다. 또한, 알림 내역과 로그 요약본을 웹 인터페이스에서 쉽게 접근할 수 있도록 한다.

## 2. 제약사항 및 수정사항

### 2.1 로그 관련 제약사항

#### (1) 로그 수집

- 각 Docker 컨테이너에서 생성된 로그를 중앙화된 장소로 전송해야 하지만, 다양한 로그 수집 도구(Logstash, Filebeat 등)를 사용할 때 어떤 기술이 데이터베이스와의 최적의 연동성을 제공할지 선택하는 데 어려움이 있다. 예를 들어, 클라우드 환경에서의 네트워크 성능과 로그 전송의 안정성을 동시에 확보할 수 있는 도구를 선택하는 것이 어렵다

- Docker 컨테이너에서 생성된 로그를 어떤 주기로 수집할지, 어떤 전처리를 거칠지, 그리고 어떤 형식으로 저장할지 결정하는 데 어려움이 있다. 로그 데이터는 실시간으로 수집될 필요가 있으며, 다양한 형식의 로그 데이터를 통일된 형식으로 전처리하여 저장하는 방식이 요구된다.

## (2) 로그 분석

- 어플리케이션마다 생성하는 로그 형식이 모두 다르기 때문에, 이러한 로그를 일괄적으로 분석하는 데 어려움이 있다. 로그 파일의 구조가 다르면 로그 데이터를 파싱하는 로직도 달라져야 하며, 이로 인해 분석 과정에서 복잡성이 증가한다.
- LLM 이 처리할 수 있는 토큰 수는 제한되어 있다. 따라서 로그 데이터의 양이 많아질 경우, LLM 의 토큰 수를 초과할 가능성이 있으며, 이때는 로그 데이터를 분할하거나 요약해야 하는 추가 작업이 필요하다. 대량의 로그 데이터를 효과적으로 분할하여 처리하는 방식에 대한 고려가 필요하다.

## 2.2 LLM 모델 제약 사항

### (1) Context-length

- 사용하려던 Llama3 - 8B 모델은 성능과 비용 면에서 적절하지만, context-length 가 8k 로 제한되어 있어 로그의 양이 많아질 경우 처리하기 어렵다. 이에 따라, context-length 가 128k 로 더 길고 성능도 우수한 Chat GPT 4o-mini 모델로 대체되었다.

### (2) 파인 튜닝

- 로그를 바탕으로 비효율적인 패턴을 탐지하기 위해 파인 튜닝을 계획했으나, Chat GPT 4o-mini 는 경우 이미 이러한 패턴들을 충분히 학습한 모델이므로 테스트 결과 효율적으로 패턴을 감지했다. 따라서 파인튜닝보다 프롬프트 엔지니어링에 집중하기로 계획을 변경하였다.

## 2.3 데이터 베이스 제약사항

### (1) RDBMS

- 기존에는 Logstash 를 통해 수집한 데이터를 가공하여 RDBMS 인 MySQL 에 저장하려 했으나, 방대한 로그 데이터를 저장하고 조회하는

과제의 성격과 맞지 않으며, Logstash 와의 연동성도 좋지 않아 로그 저장소를 분산형 검색 엔진인 Elasticsearch 로 대체하였다.

## 2.4 시스템 리소스 제약사항

### (1) AWS EC2

- 초기에는 하나의 EC2 인스턴스에서 어플리케이션 컨테이너와 로깅 기술들을 함께 운영하려 했으나, 기존 어플리케이션의 성능에 영향을 줄 수 있어 로깅 기술은 별도의 EC2 인스턴스에서 실행되도록 수정하였다.

# 3. 설계 상세화

## 3.1 전체 구성도

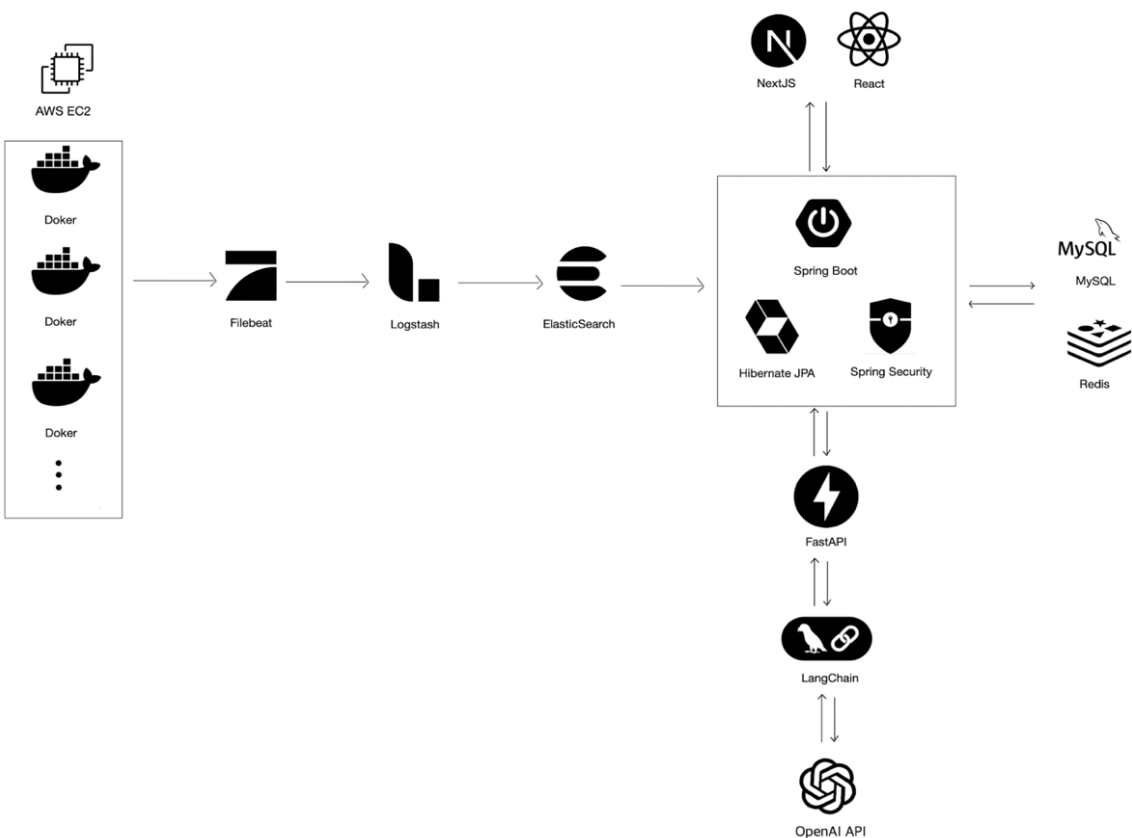


그림 1. 프로젝트 아키텍처

요소	내용
로그 수집	Filebeat 를 사용하여 컨테이너의 로그를 수집한 후, Logstash 가 수집된 로그를 전처리하여 ElasticSearch 에 저장한다. Filebeat 는 설정된 주기에 따라 로그를 수집하며, 로그는 JSON 형식으로 Logstash 에 전송된다.
백엔드 서버 (Spring)	ElasticSearch 에 저장된 로그들을 API 를 통해 클라이언트에 전달하며, 매 시간마다 Cron Job 이 실행되어 FastAPI 로 최근 로그 데이터를 전달한다.
백엔드 서버 (FastAPI)	Spring 서버에서 전달받은 로그 데이터를 LangChain 과 OpenAI API 를 사용해 분석/요약한다. 분석/요약된 데이터는 ElasticSearch 에 저장되며, 관리자 페이지에서 조회할 수 있다.
프롬프팅	OpenAI API 를 호출하기 전에 LangChain 을 사용해 프롬프트 값을 추가한다
DB (MySQL)	웹 페이지에서 사용할 사용자 인증 정보나 화면에 필요한 데이터를 저장한다.
DB (ElasticSearch)	로그 데이터를 저장한다
프론트엔드	React 와 NextJS 를 사용해 사용자에게 결과값을 웹 형식으로 보여준다. 사용자는 로그 데이터와 분석 결과를 대시보드를 통해 실시간으로 확인할 수 있으며, 필터링 및 검색 기능도 제공된다.

3.2 Front-End

- React 와 NextJS 를 사용하여 Front-End 를 구현한다.
- 대시보드를 통해 각 컨테이너의 상태 및 로그 데이터를 요약해 시각화하며, 주요 알림 및 오류 상태를 한눈에 확인할 수 있다. 실시간 업데이트를 통해 클라우드 인프라의 현재 상태를 확인할 수 있다.
- 로그 분석 요약 페이지에서는 각 컨테이너별로 수집된 로그 데이터를 LLM 을 통해 분석한 결과를 요약해 보여준다.
- 로그 상세 페이지에서는 선택한 컨테이너의 세부 로그를 시간 순서대로 보여주며, 검색 및 필터링 기능을 제공한다. 특정 시간대나 로그 레벨별로 필터링할 수 있도록 한다.
- 알림 내역 페이지에서는 시스템에서 탐지한 비정상적인 활동이나 오류에 대한 알림을 모아서 보여준다.

### 3.1 Back-End

- Spring Boot 3.2 와 FastAPI 0.1 을 사용한다.
- Spring Boot 와 FastAPI 간의 통신 및 프론트엔드와의 통신은 모두 REST API 방식을 사용하여 효율적으로 데이터를 주고받는다.
- 빠른 배포 및 테스트를 위해 백엔드 애플리케이션은 Docker 컨테이너 형태로 배포된다.
- Hibernate JPA 를 사용해 Elasticsearch 와 MySQL 에 접근하여 데이터를 처리한다. MySQL 은 주로 사용자 인증 정보 및 웹 애플리케이션에서 필요한 데이터를 저장하는 데 사용되며, Elasticsearch 는 로그 데이터와 분석 결과를 저장하는 데 사용된다.
- Redis 는 스트리밍 기능 및 세션 관리를 위한 캐시 저장소로 사용되며, 성능 향상을 위해 활용된다. 로그 분석의 실시간 처리를 위해 빠른 캐시 역할을 하며, 사용자 세션 관리를 통해 로그인을 유지하거나 실시간 데이터 처리에 사용한다.

### 3.1 데이터베이스 설계 (RDBMS)

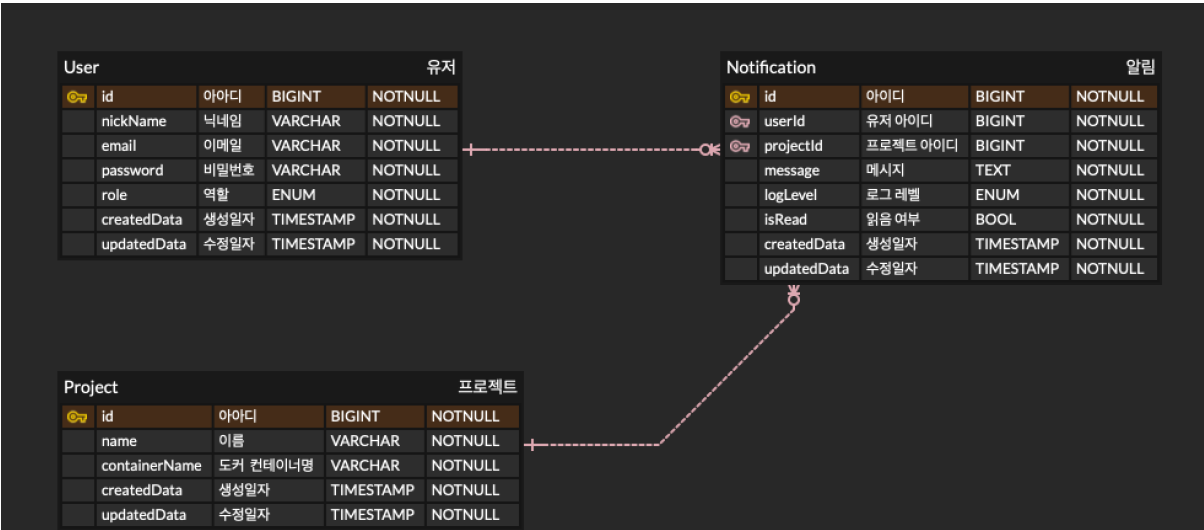


그림 2. MySQL 스키마

4. 수정된 과제 추진 내역

6 월					7 월				8 월					9 월			
1	2	3	4	5	1	2	3	4	1	2	3	4	5	1	2	3	4
요구 사항 분석 후 초기 기획 및 아키텍처 설계																	
		Docker 학습															
				ELK 스택 학습													
				LangChain 학습													
				FastAPI 학습													
									ELK 파이프라인 구축								
									LangChain 구축								
									스프링 부트 API 개발								
									FastAPI API 개발								
												UI 개발					
												프롬프팅 진행					
																배포	
																발표 준비	



## 5. 구성원별 진척상황

이한홍	<b>설계</b> <ul style="list-style-type: none"><li>- API 설계</li><li>- DB 스키마 설계</li></ul> <b>로깅 시스템 구축</b> <ul style="list-style-type: none"><li>- Filebeat =&gt; Logstash =&gt; ElasticSearch 로 이어지는 파이프라인 구축</li><li>- 로그 데이터 전처리/후처리 로직 구현</li><li>- 로그 조회를 위한 ElasticSearch 와 Spring Boot 연동</li><li>- 로깅을 위한 EC2 인스턴스와 어플리케이션 EC2 인스턴스 연동</li></ul> <b>BackEnd (Spring Boot)</b> <ul style="list-style-type: none"><li>- 유저 인증을 위한 Spring Security 구현</li><li>- ElasticSearch 에 저장된 로그를 조회하기 위한 JPA 쿼리 메서드 구현</li><li>- API 를 통한 Spring Boot 와 FastAPI 의 통신 구현</li></ul> <b>BackEnd (FastAPI)</b> <ul style="list-style-type: none"><li>- Langchain 연동</li><li>- OpenAI API 연동</li></ul> <b>인프라</b> <ul style="list-style-type: none"><li>- AWS EC2 생성 및 Docker 설치</li><li>- Jenkins 를 활용한 CI/CD 파이프라인 구축</li></ul>
	<b>박재홍</b> <b>프롬프트</b> <ul style="list-style-type: none"><li>- 로그 데이터의 프롬프트 연구</li></ul>
	<b>이종일</b> <b>FrontEnd</b> <ul style="list-style-type: none"><li>- 화면 설계</li></ul>

## 6. 과제 수행 내역 및 중간 결과

### 6.1 로그 수집 파이프라인 구축

#### 6.1.1 구축 과정

- filebeat.yml 을 통해 filebeat 설정하기
- logstash.conf 를 통해 logstash 설정하기
- docker-compose.yml 을 통해 ELK 파이프라인 실행

#### 6.1.1 Filebeat 설정

```
filebeat.inputs:
- type: container
  paths:
    - /var/lib/docker/containers/**/*.log # 수집할 로그 파일의 경로를 지정
  processors: # 수집된 로그를 처리하는데 사용되는 프로세서들을 정의
    - add_docker_metadata: ~ # 이 프로세서는 수집된 로그에 해당 로그가 속한 Docker 컨테이너의 메타데이터(컨테이너 이름, ID 등)를 자동으로 추가

# ERROR와 WARNING 레벨 로그만 Logstash로 전송
processors:
- drop_event.when.not:
  or:
    - equals:
        log.level: "ERROR"
    - equals:
        log.level: "WARNING"

# 수집된 로그 데이터를 어디로 전송할지 정의
output.logstash:
  hosts: ["logstash:5044"]
```

그림 3. filebeat.yml 파일

#### 6.1.2 Logstash 설정

```

input {
  beats { # Beats로부터 데이터를 수신
    port => 5044 # Logstash가 Beats로부터 로그를 수신할 포트를 지정
  }
}

# 입력된 데이터를 처리하고 필터링하는 블록
filter {
  # 컨테이너 이름이 "spring" 또는 "mongo"인 경우에만 데이터를 처리
  if [container][name] =~ /spring|mongo/ {
    # 필요 없는 필드 제거
    mutate {
      remove_field => [
        "input", "@version", "tags", "log",
        "[container][labels]", "[container][image]", "[container][id]",
        "host", "ecs", "agent", "stream"
      ]
    }
  } else {
    drop { } # 해당되지 않는 데이터는 버림
  }
}

output {
  elasticsearch { # 데이터를 Elasticsearch로 출력하는 플러그인
    hosts => ["http://elasticsearch:9200"] # 데이터를 전송할 Elasticsearch의 호스트 주소를 지정
    index => "docker-logs-%{+YYYY.MM.dd}" # 데이터를 저장할 Elasticsearch 인덱스를 지정
  }
  stdout {
    codec => json # 로그를 JSON 형식으로 출력
  } # 로그를 콘솔에도 출력 (디버깅용)
}

```

그림 4. filebeat.yml 파일

### 6.1.3 docker compose 설정

```

version: '3'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.3
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - ES_JAVA_OPTS=-Xms512m -Xmx512m -Duser.timezone=Asia/Seoul
    ports:
      - "9200:9200"
    volumes:
      - es_data:/usr/share/elasticsearch/data

  logstash:
    image: docker.elastic.co/logstash/logstash:7.17.3
    container_name: logstash
    volumes:
      - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
    ports:
      - "5044:5044" # Logstash가 로그를 수신할 포트
    depends_on:
      - elasticsearch

  filebeat:
    image: docker.elastic.co/beats/filebeat:7.17.3
    container_name: filebeat
    user: root
    volumes:
      - /var/lib/docker/containers:/var/lib/docker/containers:ro
      - /var/run/docker.sock:/var/run/docker.sock
      - ./filebeat.yml:/usr/share/filebeat/filebeat.yml
    depends_on:
      - logstash

volumes:
  es_data:

```

그림 5. docker-compose.yml 파일

## 6.1.4 결과

```

elasticsearch
docker.elastic.co/elasticsearch/elasticsearch:7.17.3
0740ff0e13b
9200:9200

STATUS
Running (15 seconds ago)

Logs Inspect Bind mounts Exec Files Stats
Open in external terminal

{
  "_source": {
    "container": {
      "name": "spring"
    },
    "@timestamp": "2024-08-22T12:45:31.794Z",
    "message": "2024-08-22T12:45:31.794+09:00 INFO 1 --- [MessageBroker-1] o.s.w.s.c.WebSocketMessageBrokerStats : WebSocketSession[0 current WS(0)-H
    ttpStream(0)-HttpPoll(0), 0 total, 0 closed abnormally (0 connect failure, 0 send limit, 0 transport error)], stompSubProtocol[processed CONNECT(0)-CONNECTED(0)
    -DISCONNECT(0)], stompBrokerRelay[1 sessions, ReactorNettyTcpClient[reactor.netty.tcp.TcpClientConnect@4b05a635] (available), processed CONNECT(1)-CONNECTED(1)
    -DISCONNECT(0)], inboundChannel[pool size = 0, active threads = 0, queued tasks = 0, completed tasks = 0], outboundChannel[pool size = 0, active threads = 0, que
    ued tasks = 0, completed tasks = 0], sockJsScheduler[pool size = 6, active threads = 1, queued tasks = 10, completed tasks = 0]"
  },
  "_index": "docker-logs-2024.08.22",
  "_type": "_doc",
  "_id": "HesfepEB0wLipFC8LiQ",
  "_score": 1.0,
  "_ignored": [
    "message.keyword"
  ],
  "_source": {
    "container": {
      "name": "mongo"
    },
    "@timestamp": "2024-08-22T12:45:24.641Z",
    "message": "{\"t\":{\"$date\":\"2024-08-22T12:45:24.641+00:00\"},\"s\":{\"I\":\"\", \"c\":\"\\\"WTCHKPT\\\"\", \"id\":\"22430\", \"ctx\":\"\\\"CheckpointInter\\\"\", \"msg\":\"\\
    \"WiredTiger message\\\", \"attr\":{\"message\":{\"ts_sec\":1724330724, \"ts_usec\":641026, \"thread\":\"1:0xfffffa1cdebc0\", \"session_name\":\"\\\"WT_SESSION.checkpoint\\\"
    \", \"category\":\"\\\"WT_VERB_CHECKPOINT_PROGRESS\\\"\", \"category_id\":6, \"verbose_level\":\"\\\"DEBUG_1\\\"\", \"verbose_level_id\":1, \"msg\":\"\\\"saving checkpoint snapshot min: 3
    , snapshot max: 3 snapshot count: 0, oldest timestamp: (0, 0), meta checkpoint timestamp: (0, 0) base write gen: 6340\\\"}}}"
  },
  "_index": "docker-logs-2024.08.22",
  "_type": "_doc",
  "_id": "HusfepEB0wLipFC8iLG",
  "_score": 1.0,
  "_ignored": [

```

그림 6. 로그 파이프라인을 통해 저장된 spring, mongo 컨테이너의 로그

## 6.2 OpenAI API 연동과 Langchain 구축

### 6.2.1 구축 과정

- FastAPI 와 OpenAI API 연동
- FastAPI 에 Langchain 구축
- 로그 분석 요청에 대해 응답 해주는 API 구현

6.2.2 OpenAI API 와 Langchain 을 활용한 API 구축

```
async def generate_llm_response(instruction):  
    prompt = (  
        "Please analyze log data\n"  
        f"Log: {instruction}\n"  
    )  
  
    chatmodel = ChatOpenAI(  
        model="gpt-4o-mini",  
        temperature=0.3,  
        max_tokens=750,  
        openai_api_key = settings.OPENAI_API_KEY  
    )  
  
    prompt_template = PromptTemplate(input_variables=["prompt"], template="{prompt}")  
    formatted_prompt = prompt_template.format(prompt=prompt)  
  
    # response.content 사용하여 메시지 내용 추출  
    response = chatmodel.invoke(formatted_prompt)  
    response_text = response.content
```

그림 7. FastAPI API 코드

6.2.3 결과



그림 8. OpenAI API 를 통해 얻어온 LLM 결과값

## 6.3 향후 진행 계획

- (1) Spring Boot 를 활용해 Elasticsearch 와 통합된 API 를 더 고도화할 예정이다. 이를 통해 대량의 로그 데이터를 효율적으로 처리하고, 로그 검색 및 분석 기능을 최적화할 계획이다. 또한, 다양한 검색 조건 및 필터링 기능을 제공해 관리자가 로그 데이터를 쉽게 조회하고 분석할 수 있도록 할 것이다.
- (2) LLM 을 사용해 로그를 효과적으로 분석할 수 있도록 프롬프트를 작성하고, LangChain 을 활용한 프롬프트 엔지니어링을 고도화할 예정이다. 이를 통해 LLM 이 보다 정확하게 로그 패턴을 이해하고, 중요한 이벤트를 감지할 수 있도록 프롬프트 설계를 최적화할 것이다.
- (3) React 와 Next.js 를 사용해 프론트엔드 개발을 시작할 예정이다. UI 를 직관적이고 간단하게 설계해서, 대시보드, 로그 분석 페이지, 알림 페이지 등의 주요 기능을 구현할 계획이다.
- (4) 로그 수집 파이프라인에서 로그 데이터를 전처리하는 과정을 고도화할 예정이다.
- (5) 개발된 로그 분석 시스템을 실제 운영 중인 서비스에 연동하여 테스트할 예정이다. 이를 통해 실시간으로 수집된 로그 데이터를 분석하고, 시스템의 안정성과 성능을 검증할 것이다.
- (6) CloudWatch 를 사용해서 AWS 서비스들의 로그들도 수집해 볼 예정이다.
- (7) AWS SDK 를 활용해 특정 로그 이벤트가 발생했을 때 자동으로 대응하는 액션을 구현해 볼 예정이다. 예를 들어, 심각한 오류가 탐지되었을 때 자동으로 EC2 인스턴스를 재시작하거나, 특정 이벤트 발생 시 알림을 발송하는 등의 자동화 기능을 추가할 계획이다.