# 정진묵 기술이력서

연락처 : 010-8010-3492

이메일 : jink060806@gmail.com / 깃허브 / 블로그

최종학력 : 경희대학교 기계공학과 졸업

## 경력사항

| 재직기간 | 회사명 | 역할 | 퇴사이유 |
|---|---|---|---|
| 2022.05 ~ 현재 | KT | 백엔드 엔지니어 | 서비스 개발 도전 |

## 교육이수

| 교육기간 | 교육 과정명 | 교육기관 |
|---|---|---|
| 2024.07 ~ 2024.08 | ATDD, 클린 코드 with Spring 9기 | NEXTSTEP |
| 2023.11 ~ 2023.12 | TDD, 클린 코드 with Java 17기 | NEXTSTEP |
| 2021.09 ~ 2022.03 | 엘리스 인공지능 AI 트랙 3기 | 엘리스 |

## 경험기술

- Java, Spring Boot, Spring MVC, Spring Data JPA, Spring for Apache Kafka, Spring Security, Spring Batch
- MySQL, PostgreSQL, ElasticSearch, Redis
- Apache Kafka, Apache Flink
- Kubernetes, Docker, Github Actions, Prometheus, Grafana
- Jira, Notion, Confluence Wiki, Slack, Intellij

# 업무역량

- Java와 Spring 을 활용하여 환경 측정기 디바이스 관제 시스템 API 개발을 담당했습니다.
- Apache Kafka, Redis, Spring for Apache Kafka 등을 활용하여 측정 장치 데이터 파이프라인 고도화 개발 및 운영을 담당했습니다.
- Apache Flink 를 활용하여 IoT 디바이스 측정 데이터에 대한 이벤트 처리 스트림 처리 시스템을 개발했습니다.
- 측정 장치 데이터 파이프라인 고도화 프로젝트를 진행하며 확장성과 원활한 운영에 대해 지속적으로 고민하고 있습니다.
    - 하나의 어플리케이션을 4개의 어플리케이션으로 분리하여 유연한 기능 추가와 확장성 있는 시스템을 구축했습니다.
    - 원활한 운영을 위해 Prometheus, Grafana 등의 운영 시스템을 구축하였습니다.
- 다양한 프로젝트를 경험 하면서 팀원들 간의 원활한 커뮤니케이션을 위해 노력을 기울이고 있습니다.
    - 데이터 파이프라인의 전체 과정이 4000줄 이상의 하나의 메서드로 되어 있던 레거시 시스템의 로직을 분석하여 Wiki에 정리 및 팀원들에게 공유하였습니다.
    - 새로운 개발자가 투입된 경우 시간을 따로 마련해 작성한 코드를 공유하는 시간을 가졌습니다.

# 기술 학습

- 외부 교육 수강
    - NEXTSTEP 에서 진행하는 TDD, 클린 코드 강의를 수강하고 업무에 적용하기 위해 노력하고 있습니다.
    - TDD 의 개념과 객체 지향적인 코드를 작성하는 방법을 배우고 여러 미션을 통해 익힐 수 있었습니다.
    - 인프런이나 다양한 강의 영상을 통해 Spring, Test Code, Spring Cloud 등의 기술 학습을 꾸준히 하고 있습니다.

- 꾸준한 개발 스터디 참여
    - 4명의 개발자가 모여 매 주마다 책을 읽고 내용을 공유하는 스터디를 1년 반 넘게 진행중입니다.
    - 데이터베이스, 아키텍처 설계, 클린 코드와 같은 주제를 다루는 책들을 읽고 학습했습니다.
    - 스터디를 통해 학습한 내용은 개인 노트 앱이나 블로그에 정리하고 있습니다.

- 개인 프로젝트를 통한 기술 습득
    - 새로운 기술을 단순히 책이나 강의를 통해 학습하고 끝나는 것이 아니라 개인 프로젝트를 통해 직접 사용해보고 있습니다.
    - ElasticSearch를 학습하고 적용하기 위해 스포츠 기사 조회 프로젝트를 진행했습니다.
    - Spring, JPA, AWS, CI/CD 개념에 대해 학습 후 적용하기 위해 결혼 초청장 서비스를 진행했습니다.

# 업무 경험

## KT (2022.05 ~ 현재)

**[**수행 프로젝트**]**

**환경 측정 장치 데이터 파이프라인 고도화 개발 프로젝트**

**[**해결방안 및 사용기술**]**

- 기존 파이프라인 로직 정리 및 히스토리 파악 및 프로젝트 규칙 설정
    - Kafka를 이용한 데이터 파이프라인 구조에 대해 Confluence Wiki를 통해 관련 내용을 정리 하도록 팀 내 규정 및 History 작성 제안을 통해 서비스 로직에 대한 이해 및 개발 환경 개선
    - 모노 레포 및 멀티 모듈 프로젝트 구성 시 모듈 이름과 Kafka Topic 이름 설정에 대한 규칙 정리
- 기존 모노리틱 서비스를 **4**개의 **Java** 어플리케이션으로 구성된 형태의 파이프라인으로 구조 개선
    - 기존 Spring Boot 하나의 어플리케이션에서 진행하던 로직을 Kafka와 함께, 데이터 해석 및 그룹화 → 전처리 → 보정,QC → DB에 데이터 저장, 총 4개의 Spring Boot 어플리케이션으로 분리 개발
    - 전처리 로직에서 기존에 모두 if 문으로 분기 처리된 로직을 Interface 와 전략 패턴을 응용하여 유지 보수하기 쉬운 코드로 개선
    - 어플리케이션을 분리하고 이를 Kafka 를 통해 연결하는 구조였기 때문에 파이프라인과 관련 없는 기능이라면 새로운 어플리케이션을 개발하고 원하는 토픽에 연결함으로써 기능 확장에 유리하도록 설계
    - 기존에 측정 데이터를 단건마다 DB에 Insert 하는 로직을 Bulk Insert 로 변경하여 DB 저장 시간이 더 빨라지도록 구현
- **Kafka** 클러스터 **3**대**,** 각 어플리케이션은 **2**개씩 배포하여 고가용성 확보
    - 안정적인 운영을 위해 Grafana, Prometheus 등의 스택을 이용하여 Kafka 관련 대시보드 구축
    - 어플리케이션 로그를 편리하게 보기 위해 로그 Grafana-Promtail-Loki 시스템 구축

**[**수행 프로젝트**]**

**IoT 이벤트 엔진 설계 및 개발**

**[**해결방안 및 사용기술**]**

- 여러 **IoT** 장비 간 **CEP(Complex Event Processing)** 처리를 위한 **Apache Flink** 도입
    - Flink Stream API를 활용하여 데이터 처리 후 적재, 이벤트 판단 등 파이프라인 설계 및 개발
    - Flink Keyed State를 이용해 디바이스 측정 데이터가 정상적으로 올라오지 않는 장애 상황에 대한 이벤트 처리 기능 개발

- **Apache Flink**를 활용하여 스트리밍 단순 이벤트 처리 로직 구현
    - 여러 단계의 대소/동등 비교 등의 수식 처리를 Java Object로 파싱하여 판단하는 로직 구현
    - Java 인터페이스를 활용한 다형성과 재귀 로직을 이용하여 이벤트 판단 로직 구현
    - 핵심 로직인 수식 판단 로직의 도메인 클래스의 경우 TDD를 활용하여 엣지 케이스에 대한 버그를 최소화하며 개발 진행

- 이벤트 관리에 필요한 **API**를 제공하는 **Spring** 어플리케이션 개발
    - Spring Data JPA 를 기본으로 사용하고, 동적 쿼리는 QueryDSL을 이용하여 개발
    - 이벤트에 관한 CRUD 작업이 발생하면 Redis에 해당 내용을 업데이트 하고 Flink Job 에서 사용하도록 설계

# 개인 프로젝트

**[프로젝트명]**

**스포츠 기사 조회 서비스** (Github)

**[상세 설명]**

사용자가 키워드를 등록하면 해당 키워드로 스포츠 기사 제목, 본문을 검색해 결과를 알려주는 서비스입니다. 사용자가 등록한 키워드에 해당하는 기사가 새로 추가되는 경우 사용자에게 알람을 보내주어 관심있는 키워드에 대한 기사를 지속적으로 확인할 수 있습니다.

**[해결방안 및 사용기술]**

- **Spring Boot, Spring Data JPA, Spring batch, OracleDB, Redis cluster, ElasticSearch Cluster** 를 이용해 프로젝트 구성
- **Spring Batch** 를 이용한 네이버 스포츠 기사 크롤링
    - Spring Batch와 Kubernetes Cron Job 을 이용해서 10분마다 배치 작업을 실행해 스포츠 기사를 ElasticSearch와 Oracle Database 에 저장
    - 이때 Redis에 크롤링한 기사 주소를 저장하여 중복된 기사가 크롤링 되는 케이스를 막도록 설계
    - CI 과정에서 빌드 시간이 느려 특정 버전의 크롬 드라이버를 다운받은 베이스 이미지를 이용하여 이미지 빌드 시간 단축
- **ElasticSearch**의 **search** 쿼리를 이용한 스포츠 기사 조회 기능 구현
    - 2core 12GB 서버 1대, 1core 6GB 서버 2대를 이용해 ElasticSearch 클러스터 구성, 각 서버는 master, data 역할을 하는 노드로 구성
    - 한글 검색 키워드에 맞춘 검색을 지원하기 위해 한글 토크나이저인 analysis-nori 적용
    - 캐릭터 필터, 토크나이저, 토큰 필터를 조합한 커스텀 애널라이저를 사용
        - ElasticSearch에 데이터를 저장할 때 HTML 태그와 같이 저장되기 때문에 HTML 태그를 삭제하는 캐릭터 필터를 적용
        - 토크나이저로는 nori_tokenizer를 사용

        - 토큰 필터로 nori_part_of_speech 와 nori_readingform 필터를 적용
    - Search After 쿼리와 PIT(Point In Time)를 사용하여 무한 스크롤 기능 구현

- **ElasticSearch** 의 **ILM(index lifecycle management)** 기능과 **Alias** 기능을 이용한 인덱스 운영

    - sports_articles_${yyyyMMdd} 형식의 인덱스를 사용하고 관련된 인덱스를 articles 라는 Alias 를 통해 사용하도록 설정
    - 날짜별로 인덱스를 분리하여 이후 Index Lifecycle Management 기능을 이용해 60일 지난 인덱스는 삭제하도록 설정
    - 초기 기본 애널라이저를 사용하다 이후 커스텀 애널라이저를 사용하도록 변경. 이때 새로운 인덱스 생성 후 Reindex API로 기존 인덱스 데이터를 마이그레이션 한 후 Alias 를 이용해 다운타임 없이 인덱스 변경

- 기사 업데이트 알림을 위한 **Server Sent Event** 이용

    - Spring Batch 작업이 끝날 때 Redis Pub/Sub 에 새로 생성된 기사 id 목록을 전달함
    - Spring Application 에서는 해당 기사 id 목록을 Subscribe 하여 ElasticSearch 의 Multi term vectors API 를 이용해 역인덱스 내용과 사용자가 등록한 키워드를 비교하여 일치하는 사용자에게 Server Sent Event 를 발행하여 기사 업데이트를 알림

- **Redis**를 이용해 사용자 검색어 순위를 보여주는 기능 구현

    - Redis Sorted Set 을 이용하여 사용자가 검색할때 Sorted Set 에 해당하는 키워드의 score 를 1 증가
    - 이후 상위 10개의 목록을 가져와 현재 상위 10개의 목록과 비교해 변동이 있다면 Server Sent Event 를 발행하여 검색어 순위를 업데이트함
    - 사용자가 검색을 진행할 때 검색어에 변경이 있는 경우에만 이벤트를 발행하여 검색어 순위를 업데이트 하도록 로직 구현
        - 초기에는 Spring 의 EventListener 기능을 사용했지만 이후 여러 어플리케이션에서 해당 이벤트를 받아야 하므로 Redis Pub/Sub 으로 변경

- **GitHub Actions** 를 사용한 **CI/CD** 자동화 구성

    - 배포 서버만 존재했고 인원이 2명이라 main 브랜치와 하위 feature 브랜치로 개발 진행. 이후 main 브랜치에 Merge가 일어난 경우 새로 업데이트 진행되도록 설정

    - GitHub Actions의 단계는 크게 test, build, deploy 단계로 나눠서 작업되도록 설정

        - test 단계는 Pull Request 가 open 되거나 다시 reopen 되는 경우에 진행되도록 설정했으며 어플리케이션의 테스트를 진행합니다.
        - build 단계는 Merge가 성공하면 진행하도록 설정했으며, docker image를 빌드하고 빌드된 이미지를 GitHub Package에 저장하는 단계입니다.
        - 마지막 deploy 단계는 build 단계가 끝난 이후 저장된 이미지를 이용하여 Kubernetes API를 활용해 이미지 버전을 변경하여 롤링 업데이트가 진행되도록 설정했습니다.

**[회고]**

- 배운점
    - ElasticSearch의 다양한 기능들에 대해 배울 수 있었습니다.
        - Search API, Analyzer, ILM, Alias, Reindex 등
    - Redis Pub/Sub 에 대해 배울 수 있었습니다.
        - Redis Pub/Sub 에 대한 개념 및 활용 방안
        - Spring Data Redis 에서 어떻게 Pub/Sub 을 사용하는지 학습
    - 사용자에게 알람을 보내기 위해 사용한 Server Sent Event 에 대해 배울 수 있었습니다.
        - Spring 에서 Server Sent Event 를 사용할 때 어떤 설정들이 중요한지

- 느낀점
    - 직접 운영을 하기 위해서는 정말 많은 고민이 필요하다고 느꼈습니다.
        - ElasticSearch 의 인덱스 설계부터, ILM 설계 등 어떻게 확장성 있는 설계를 할 수 있을지 많은 고민이 필요하다고 느꼈습니다.
        - 실제 서비스라면 다운타임 없이 어떻게 인덱스를 업데이트 하고, 데이터는 어느 정도 기간 보관할 것인지 등 다양한 고민을 하게 되었습니다.
    - 특정 기술을 사용할 때 어떻게 설정을 할지 해당 설정이 어떤 역할을 하는지 더 자세히 알아야겠다고 느꼈습니다.
        - Server Sent Event 를 구현하면서 nginx의 proxy-read-timeout 설정을 제대로 해주지 않아서 원하던 방식으로 동작하지 않았습니다.
        - Spring 에서 SseEmitter 클래스의 동작 방식에 대해 학습한 후 원하는

동작대로 실행시킬 수 있었습니다.

- 개선할 점
    - 사용자 검색어 순위를 보여줄 때 현재는 Redis Sorted Set 의 score 가 꾸준히 증가하므로 해당 시간대에 맞는 검색어 순위를 제대로 보여주지 못해서 로직 개선이 필요합니다.
        - 로직 개선을 위해 주기적으로 상위 10개의 키워드 score를 전부 1 ~ 10 으로 세팅을 한 후 나머지 검색어는 삭제하는 방법을 생각하고 있습니다.
    - Redis Pub/Sub 을 사용하다보니 특정 어플리케이션이 배포될 때 Redis 채널에 이벤트가 발생하게 되면 해당 어플리케이션에서는 이벤트를 인지하지 못하는 문제가 있습니다.
        - Kafka 와 같이 이벤트를 저장할 수 있는 메시지 큐를 사용하면 해결할 수 있을것 같습니다.
    - 개인 프로젝트이다 보니 실제 배포 환경만 있고 개발, 스테이징 환경은 존재하지 않는데 각 환경마다 브랜치 전략과 배포 전략을 구분할 수 있다면 더 좋았을것 같습니다.
        - Pull Request 생성 이후 코드 수정 후 다시 Push를 하는경우 테스트가 진행되지 않는 문제가 있어서 좀 더 깔끔하게 개선할 수 있을것 같습니다.

**[프로젝트명]**

**결혼 초청장 및 관리 서비스** (Github)

**[상세 설명]**

예비 신랑 신부가 결혼식 청첩장을 만들고 공유할 수 있는 서비스입니다. 청첩장을 보낼 때 웨딩 사진과 자신이 받고 싶은 선물 목록을 같이 보낼 수 있습니다.

**[해결방안 및 사용기술]**

- **Spring Boot, Spring Data JPA, Spring Security, RDS, S3**를 이용해 개발 및 배
- **Spring Security, Oauth2.0** 및 **JWT**를 이용한 인증**,** 인가 **API** 구현
    - 서비스에 필요한 정보를 담기 위해 Oauth2.0 인증 과정에서 얻은 정보만 사용하고 서비스에서 자체 토큰 생성
- **Naver** 쇼핑 페이지 특정 **URL** 입력 시 해당 상품 정보 크롤링 **API** 구현
- **AWS S3**를 이용한 웨딩 사진 업로드 및 공개 **API** 구현
    - 사진 업로드 시 사진 URL 조회는 public 으로 공개한 후 해당 주소만 DB에 저장하도록 로직 구현
    - S3 업로드 부분과 DB 저장부분의 트랜잭션을 분리하여 DB Connection 시간을 줄이도록 구현
- 청첩장을 받은 사용자가 해당 청첩장에만 접근할 수 있도록 **API** 로직 구현
    - 청첩장 도메인 주소에 UUID를 2개 사용하여 다른 청첩장을 최대한 보지 못하도록 설정
    - 처음 청첩장에 접속 후 이후 API 호출 시 DB에 사용자 체크하는 로직을 반복하지 않도록 특정 header 값을 사용
- 단위 테스트 작성**, mockito** 를 이용한 테스트 혹은 **@SpringBootTest**를 이용한 통합 테스트 코드 작성
- **GitHub Actions** 를 사용한 **CI/CD** 자동화 구성
    - 초기 Jenkins 를 이용하여 구성 했지만, 관리 포인트 증가로 인해 GitHub Actions로 변경

**[회고]**

- 배운점
    - Spring 관련 기술을 사용한 첫 프로젝트였기 때문에 관련 기술들을 학습하고 직접 적용할 수 있었습니다.
    - AWS를 이용해 직접 인프라를 처음 세팅하면서 다양한 관련 개념들을 알게 되었습니다.
        - AWS VPC, Application Load Balancer, S3 등 다양한 인프라 관련 기능들에 대해 학습할 수 있었습니다.
    - CI/CD 자동화에 대해 처음 접하고 직접 세팅 하며 관련 기술을 습득했습니다.
        - API 개발 외에 처음으로 DevOps 관련 기술을 직접 이용하면서 새로운 기술에 대해 두려움 없이 적용할 수 있다는 자신감을 얻을 수 있었습니다.

- 느낀점
    - 테스트 코드 작성을 통해 기존보다 좀 더 자신감 있는 배포를 할 수 있다고 느꼈습니다.
        - 직접 테스트 코드를 작성하며 엣지 케이스에 대해 테스트를 하고 배포를 하다보니 테스트 코드가 없던 코드에 비해 버그가 덜 발생할거라는 자신감이 생겼습니다.
        - 이후 테스트 코드에 대한 관심이 생겨 외부 교육을 수강하게 된 계기가 되었습니다.
    - CI/CD 자동화를 통해 좀 더 개발에 집중할 수 있다고 느꼈습니다.
        - 기존 업무에서는 직접 파일을 이동시키고 배포를 수동으로 진행했는데 이러한 과정을 자동으로 진행하다보니 코드 구현에 집중할 수 있었습니다.

- 개선할 점
    - 동시성에 대한 처리가 부족한 로직이 있어 개선이 필요합니다.
        - 초대를 받은 사람들에게 후원을 받으면 총 후원 금액에 업데이트가 되는데 해당 로직에 동시성 이유가 있을 수 있습니다.
        - 하나의 데이터에 대해 검색 후 후원 금액 컬럼을 업데이트하는 로직이기 때문에 비관적 락을 이용하여 처리하면 좋을것 같습니다.

**[프로젝트명]**

**인공지능을 활용한 레시피 추천 서비스** (Github)

**[상세 설명]**

재료 사진을 업로드 하면 해당 재료로 만들 수 있는 레시피를 제공해 줍니다. 재료로 검색하면 해당 재료로 만들 수 있는 레시피를 제공합니다. 자신만의 레시피를 등록할 수 있습니다.

**[해결방안 및 사용기술]**

- **Docker Compose**를 이용하여 개발환경 세팅
    - 프론트와 백엔드 개발자가 각자 개인 노트북에서 개발을 진행하다보니 서로 API 연동 테스트의 어려움이 있어 Docker를 학습하여 Docker Compose 를 이용해 각자 노트북에 개발환경을 세팅하여 편하게 개발을 진행할 수 있도록 세팅
- 프론트 서버와 연동하여 소셜 로그인 기능 구현
    - 프론트 서버에서 인가 코드를 발급 받고 해당 인가 코드를 백엔드 서버로 보내고 이후 access_token 을 발급 받고 로그인 처리는 백엔드 서버에서 진행하도록 구현
- 레시피 작성과 관련된 **API** 개발 진행

**[회고]**

- 배운점
    - Docker 라는 컨테이너 기술을 스스로 학습하고 처음으로 프로젝트에 사용하면서 개발환경과 운영환경과의 차이 없이 진행할 수 있다는 장점이 있다고 느꼈습니다.
    - 특히 어플리케이션을 배포할 때 Docker Image에 문제가 없으면 손쉽게 배포할 수 있다는 장점이 있는것 같습니다.

- 느낀점
    - 처음으로 개발부터 배포까지 진행한 프로젝트이다보니 힘들더라도 마무리를 했다는 부분에서 성취감을 느낄 수 있었습니다.
    - 여러 사람과 같이 프로젝트를 진행하다보니 대화의 중요성을 느낄 수 있었습니다.
        - 특히 같은 주제로 얘기를 하더라도 서로 이해하는 부분이 다를 수 있다는 사실을 많이 느꼈습니다.

- 개선할 점
    - 문서화를 잘 해야겠다고 생각했습니다.
        - 서로 서비스 요구사항에 대해 회의한 내용에 대해 제대로 정리를 하지 않았다보니 나중에 서로 다른 요구사항을 이해해서 다시 커뮤니케이션을 하는 경우가 많았습니다.
        - 회의를 하면 항상 회의록을 잘 작성하고 서로 공유하는 팀 그라운드 룰이 있으면 더 효율적이게 프로젝트를 진행할 수 있었을 것 같습니다.

# 자기소개서

**[**프로젝트 경험**]**

**1. Java** 와 **Spring Framework** 기반의 서비스 개발 경험

데이터 파이프라인 고도화 프로젝트를 진행했습니다. 해당 프로젝트는 Java, Spring Boot, Spring for Apache Kafka를 사용하여 진행했습니다. 파이프라인 로직 중 Kafka에 전달된 데이터를 애플리케이션에서 사용하는 DTO 형식에 맞도록 전처리하는 기능이 존재했습니다. 이때 문제는 Kafka에 크게 실내 측정기, 실외 측정기, 환기장치, 산소발생기 4개의 디바이스의 데이터가 들어오는데, 해당 디바이스 종류별로 전처리 과정이 조금씩 다른 문제가 있었습니다.

이때 Java의 인터페이스를 활용하여 다형성을 이용해 유연하게 코드를 작성하도록 구현했습니다. 기존 로직의 경우, if 문을 사용하여 4가지 디바이스 경우에 대해 나눠서 처리를 하였지만 데이터 태그들이 한두 개가 아닌 몇십 개의 태그들이 존재했기 때문에 코드의 가독성과 유지보수성이 좋지 않다고 느꼈습니다. 따라서 데이터를 전처리하는 인터페이스를 하나 정의하고, 각각의 디바이스에 대한 전처리 구현체 4개를 따로 만들어서 사용했습니다. 이때 스프링의 빈 주입을 통해 4개의 구현체를 Map 형식으로 주입받고 Key 값을 디바이스 종류로 설정하였습니다. 이후 Kafka로 들어오는 Key 값에 어떤 종류의 디바이스 장치인지 정보가 있었기 때문에 해당 정보를 이용하여 들어오는 데이터마다 Map에서 구현체를 선택해 전처리를 진행하도록 구현했습니다.

이렇게 구현하여 이후 특정 디바이스의 전처리 로직에 수정이 필요한 경우 해당하는 구현체만 수정하면 되기 때문에 유지보수에 유용한 구조가 되었고, 만약 새로운 디바이스가 추가된다면 해당 디바이스에 맞는 구현체를 구현하고 빈으로 추가 등록만 하면 되기 때문에 객체지향 원칙 중 개방-폐쇄 원칙을 지키는 구조를 가져갈 수 있었습니다.

**2. ElasticSearch 8.11.1** 버전 인덱스 설계 경험

처음 인덱스를 설계할 때 문자열 프로퍼티에서 어떤 필드를 “text” 타입으로 두고 어떤 필드를 “keyword” 타입으로 설계하는지에 대해 고민이 있었습니다. “text” 타입은 전문 검색에 적합하고 “keyword” 타입은 단순 완전 일치 검색에 적합하기 때문에, 어떤 필드로 검색할지를 먼저 고려했습니다. 서비스 요구사항으로는 제목과 본문으로 검색을 하기 위해 제목과 본문 필드는 “text” 타입으로 설정하고, 다른 대부분의 필드는 “keyword”로 설정했습니다. 이때 이후에 저널리스트 이름으로도 검색할 수 있을 것이라는 서비스 확장을 염두에 두고 추가될 만한 필드는 “text” 타입으로 설정했습니다.

다음으로 “text” 타입으로 설정한 필드의 애널라이저는 저장된 데이터 형식에 맞게 캐릭터 필터, 토크나이저, 토큰 필터를 조합한 커스텀 애널라이저를 사용했습니다. ElasticSearch에 저장된 기사 내용과 제목이 HTML 형식으로 저장되어 있었기 때문에 HTML 태그를 삭제해주는 html_strip 캐릭터 필터를 적용하였고, 토크나이저는 한글 분석을 위해 nori_tokenizer를 사용했습니다. 마지막으로 토큰 필터는 nori_part_of_speech와 nori_readingform 필터를 적용하여 검색 성능을 높이기 위해 사용했습니다.

다음으로 주기적으로 데이터를 삭제하기 위해 ILM(Index Lifecycle Management) 기능을 적용하여 60일이 지난 인덱스는 삭제하도록 설계하였습니다. 이를 위해 동일한 인덱스 설정을 갖도록 인덱스 템플릿을 정의하고 날짜에 따라 인덱스를 생성하도록 했습니다. 이후 ElasticSearch의 Alias 기능을 이용해 날짜별 인덱스를 하나의 인덱스 이름 별칭으로 조회하도록 인덱스를 설계하였습니다.

**3.** 새로운 기술 스택을 학습한 경험

IoT 이벤트 엔진 설계 및 개발 프로젝트를 진행할 때 Apache Flink를 도입하여 사용했습니다. Flink를 도입한 이유로는, 우선 IoT 디바이스에서 Kafka Topic에 측정 데이터가 올라오는데, 해당 데이터를 실시간으로 스트림 처리하여 데이터베이스에 저장해야 하는 스트리밍 처리가 필요했습니다.

다음으로 강력한 상태 저장 처리가 필요했습니다. 특히 특정 디바이스에서 주기적으로 데이터가 올라오지 않는 경우, 해당 상황을 인지하고 디바이스에 문제가 있다고 판단하기 위해 디바이스별로 Keyed State를 이용한 상태 처리가 필요했으며, Flink가 이러한 상태 처리에 강점이 있다고 판단했습니다.

마지막으로 CEP(Complex Event Processing) 기능을 이용한 이벤트 처리 요구사항이 있었는데, Flink에서는 API로 이를 제공했기 때문에 요구사항에 필요한 기능이 적절하게 갖춰져 있다고 판단하여 해당 기술을 선택했습니다.

처음 Flink라는 기술을 학습하기 위해 가장 먼저 공식 문서를 이용했습니다. 강의나 책을 보는 것도 좋지만 상대적으로 자료가 부족했기 때문에 공식 문서가 가장 좋은 자료라고 판단했습니다. 기본적인 Flink의 개념을 이해하기 위해 공식 문서를 읽고 개인적으로 간단한 데모 프로젝트를 구현하여 Flink에서 제공하는 여러 API를 테스트한 후, 원하는 요구사항을 구현할 수 있겠다고 판단하여 기술을 선택했습니다.

다음으로 Kafka와 ElasticSearch를 처음 학습할 때는 공식 문서뿐만 아니라 여러 좋은 책이나 강의가 많았기 때문에 외부 자료들을 많이 이용했습니다. Kafka의 경우 “실전 카프카: 개발부터 운영까지”라는 책으로 학습을 했고, ElasticSearch는 “엘라스틱서치 바이블”이라는 책으로 학습했습니다.

학습 이후에는 반드시 개인적으로 간단한 프로젝트라도 만들어서 학습한 내용을 직접 코드로 작성하는 시간을 갖기 위해 노력합니다. Kafka의 경우 Spring 환경에서 Spring for Apache Kafka를 이용해 사용했는데, 관련된 설정을 이해하기 위해 Spring 공식 문서를 확인하며 테스트를 진행했습니다. 컨슈머의 싱글 모드 혹은 배치 모드 설정이나 자동 커밋, 프로듀서의 중복 없는 전송, 정확히 한 번 전송 등의 설정을 어떻게 하는지 직접 테스트하며 확인했습니다.

ElasticSearch의 경우 직접 프로젝트를 진행하며 책에서 읽었던 Alias, Reindex, 커스텀 애널라이저 기능 등을 직접 적용하면서 작동 방식을 이해할 수 있었습니다.

이처럼 새로운 기술 스택을 학습해야 하는 경우, 가장 먼저 공식 문서나 관련된 강의 혹은 책을 통해 대략적으로 어떤 기능을 하는지 확인하는 시간을 갖습니다. 그 후 반드시 직접 코드로 구현하며 테스트해보고 학습한 기능들을 제 것으로 만들기 위해 노력합니다.

**4.** 비효율적인 시스템 구조를 개선한 경험

데이터 파이프라인 고도화 프로젝트에서는 기존 하나의 애플리케이션으로 이루어진 Spring 애플리케이션을 크게 4개의 Spring 애플리케이션과 Kafka를 이용해 분리하는 과정을 거쳤습니다. 각 애플리케이션의 역할은 다음과 같습니다. 먼저, Avro 형식의 데이터를 코드에서 사용할 DTO 형식으로 변환하는 역할, 다음으로 해당 DTO의 값을 전처리하여 DB에 저장할 칼럼 형식으로 변환하는 역할, 데이터 보정을 진행하는 역할, 마지막으로 DB에 데이터를 저장하는 역할이 있습니다.

가장 먼저 기존 파이프라인의 로직을 파악하는 것이 중요했습니다. 메서드 하나로 이루어진 로직이었는데, 해당 로직에 대해 정확하게 알고 있는 사람이 없었기 때문에 전달받은 코드를 하나하나 분석했습니다. 이러한 과정에서 시퀀스 다이어그램을 작성하고 로직을 파악하기 위한 작업들을 진행했으며, 요구사항을 정리했습니다. 예를 들어, 기존 코드에는 실외, 실내 공기에 대한 보정 과정이 있었는데, 해당 과정은 사실 실외 공기에 대한 보정만 진행하면 되는 요구사항이었습니다. 이를 팀장님과 공유하고 다른 부서 사람들과의 회의를 통해 요구사항을 좀 더 명확하게 확립할 수 있었습니다.

다음으로, 좀 더 유연하고 확장성 있는 구조에 대해 고민했습니다. 데이터는 크게 실내 측정기, 실외 측정기, 환기 장치, 산소 발생기 4개의 종류가 있었습니다. 단일 애플리케이션을 4개의 애플리케이션으로 나눈 이유는 중간에 Kafka를 이용해 데이터를 넘겨주는데, 해당 토픽에 추가적인 기능을 갖는 애플리케이션이 추가될 수 있기 때문에 확장성을 염두에 두고 설계했습니다. 실제로 중간 토픽에서 외부에 전달하는 데이터를 따로 필터링해서 전달하는 애플리케이션도 나중에 유연하게 확장할 수 있었습니다.

결국, 기존 하나의 메서드로 이루어져 있던 파이프라인 과정을 애플리케이션 4개와 Kafka를 이용한 형식의 파이프라인으로 고도화했습니다. 저는 해당 모듈에서 DTO를 DB에 저장할 칼럼 형식으로 변환하는 모듈과 보정하는 모듈의 개발을 주로 맡아서 진행했습니다. 이후 정상적인 운영을 위한 모니터링 관련 정책과 배포 작업 정책을 수립하여 몇 달간의 테스트 이후에 큰 문제 없이 배포할 수 있었습니다.

**[**업무 강점 및 약점**]**

**[**업무 강점**]**

**1.** 커뮤니케이션

개발자에게는 개발 능력뿐만 아니라 다양한 사람들과 협업을 진행하기 때문에 원활하게 협업할 수 있는 능력 또한 굉장히 중요한 요소 중 하나라고 생각합니다. 과거 사내에서 진행했던 프로젝트를 통해 어떻게 해야 일을 효율적으로, 불필요한 의사소통으로 인한 자원 낭비 없이 협업을 진행할 수 있는지에 대해 고민하고 그러한 고민들을 실행으로 옮겨 이후에 새로운 신입분들이 입사했을 때 빠르게 팀에 적응할 수 있도록 도움이 되었던 경험이 있습니다.

처음에 데이터 파이프라인 고도화 프로젝트를 시작했을 때, 팀에서 아무도 해당 프로젝트 코드들이 어떻게 작성되었는지 모르는 상태였고, 제대로 된 로직도 잘 모르는 상태였습니다. 이러한 상황에서 제가 먼저 프로젝트에 투입되어 기존 코드를 분석하고 있었는데, 매우 많은 로직이 있어 제대로 정리가 안 되고, 오래 본 저는 괜찮지만 이를 새로 보게 되는 분들은 이해하기 매우 어려운 코드로 구성되어 있었습니다. 이에 따라 새로운 기능이 추가될 때마다 서로 물어보게 되고, 이해하는 것이 달랐다고 느꼈습니다. 그래서 저는 모든 로직을 서비스 시퀀스와 로직에 대한 설명을 적어 팀에 공유하였습니다.

이러한 산출물로 인해 다음에 투입되는 분들은 조금 더 쉽게 업무에 적응할 수 있었고, 서로 비즈니스 로직에 대한 이해가 달라 발생하는 문제들을 최소화하며 로직 개발에 좀 더 집중할 수 있는 환경으로 변화할 수 있었습니다.

이러한 경험을 통해 불편함을 인지하고 작은 일이지만 먼저 노력한다면 팀에 큰 도움이 될 수 있다는 것을 느끼게 되었습니다. 이러한 마음을 가지고 입사하여 팀원들의 불편함을 찾으려 노력하고 먼저 나서서 사소하더라도 조금이나마 편하게 할 수 있도록 서포트한다면 좋은 시너지가 날 것이고, 좀 더 개발에 집중할 수 있는 팀으로 나아갈 수 있다고 생각합니다. 이처럼 사소한 일이라도 팀에 좋은 영향을 미칠 것으로 판단되면 먼저 나서서 해결하며, 팀원들이 저와 같이 협업하기 좋은 개발자가 되고 싶습니다.

**2.** 팀원들과 함께 성장하기 위한 기술공유

새로 알게 된 기술을 정리하고 동료들과 공유하는 것을 좋아합니다. 최근에 진행한 프로젝트를 Kubernetes 환경에서 운영했습니다. 이때 팀원들이 모두 Kubernetes 환경에 대해 잘 알고 있는 상태가 아니었습니다. 직접적으로 Kubernetes를 사용하지 않더라도 개발하면서 만나게 될 이슈 사항들이나 트러블슈팅을 할 때 Kubernetes를 전혀 모르는 상태에서 진행하면 어려울 것으로 판단하였습니다. 이때 기존에 개인적으로 Kubernetes를 학습하고 정리한 내용을 바탕으로 팀원들에게 공유하는 자리를 가졌습니다.

모든 내용을 단시간에 공유하기는 불가능하다고 판단하여 개발에 필요한 개념들을 추려서 공유했습니다. Pod, Service, Deployment, Label, ConfigMap 정도의 개념을 정리해서 공유했습니다. 이후 팀원들도 관련된 YAML 파일들을 보며 개념에 익숙해졌고, Kubernetes 환경에서 운영하는 데 많은 도움이 되었다는 피드백을 받을 수 있었습니다.

외부 교육을 통해 알게 된 내용 중 개발에 많은 도움이 될 것 같은 내용들도 틈틈이 팀원들과 얘기하며 공유하려고 합니다. NEXTSTEP에서 TDD 교육을 들었을 때, 객체 지향적인 코드를 위한 일급 컬렉션 클래스와 밸류 객체에 대한 내용이 있었습니다. 당시 교육을 듣고 미션을 진행하며 실제로 훨씬 안정적이고 객체지향적인 코드를 작성할 수 있다고 판단하고, 간단한 내용이기 때문에 팀원들과 공유하면 좋겠다고 판단했습니다. 이후 팀원들과 커피를 마시면서 교육에서 들은 내용을 공유했습니다. 이처럼 단순히 저 혼자만을 위한 성장이 아니라 주변 팀원들도 같이 성장할 수 있는 내용이라면 최대한 공유하려고 노력하고, 공유를 하는 과정을 통해 저 또한 한 번 더 복습하며 익힐 수 있다고 느껴서 좋은 습관이라고 생각합니다.

**[**업무 단점**]**

**1.** 스스로 해결해야 한다는 생각

학창 시절부터 문제가 생기면 항상 혼자 고민하고 공부하며 스스로 답을 찾으려는 스타일이었습니다. 이러한 습관이 아직도 남아있는지, 처음 개발을 시작하고 막히는 코드나 버그가 있을 때 어떻게든 혼자 디버깅하며 원인을 찾으려고 노력했습니다. 이때 디버깅에 집중하다 보면 특정 문제에 몇 시간씩 시간을 허비하는 경우도 종종 발생하곤 했습니다. 이러한 스타일이 개인적인 공부를 할 때는 좋은 습관이 될 수도 있다고 느꼈지만, 업무에서는 좋지 않은 습관이라는 생각이 들었습니다. 특히 마감 일정이 있는 업무를 진행할 때는 혼자 고민하는 시간이 어느 정도는 필요하지만, 특정 시간이 지나면 업무 효율이 떨어진다고 느꼈습니다.

따라서 스스로 업무에 있어서 혼자 고민하는 시간은 최대 1시간이라는 규칙을 정하고, 그 시간 안에

해결하지 못하는 문제는 주변 동료들과 대화를 통해 같이 고민하고 도움을 요청하는 방법을 사용했습니다. 물론 주변 동료도 고민이 필요한 문제들이 있지만, 같이 얘기하다 보면 실마리를 찾게 되는 경우도 있고, 이미 한 번 경험한 동료가 있다면 바로 해결되는 경우도 있었습니다.

이처럼 업무에서는 무엇보다 일정 관리가 중요한데, 너무 혼자서만 고민하지 말고 주변 동료들과 같이 고민하며, 마찬가지로 누군가 저에게 도움을 요청할 때 주저 없이 도와줘야겠다고 생각했습니다.

끝