1. 온보딩 프로세스

기반 기술

1.1. info

Owner	서정현

1.2. History

버전	변경 내용	작성자	일자
v0.1	초안 작성	서정현	2024년 1월 15일
v1.0	리뷰 의견 적용 및 내용 추가	서정현	2024년 1월 17일

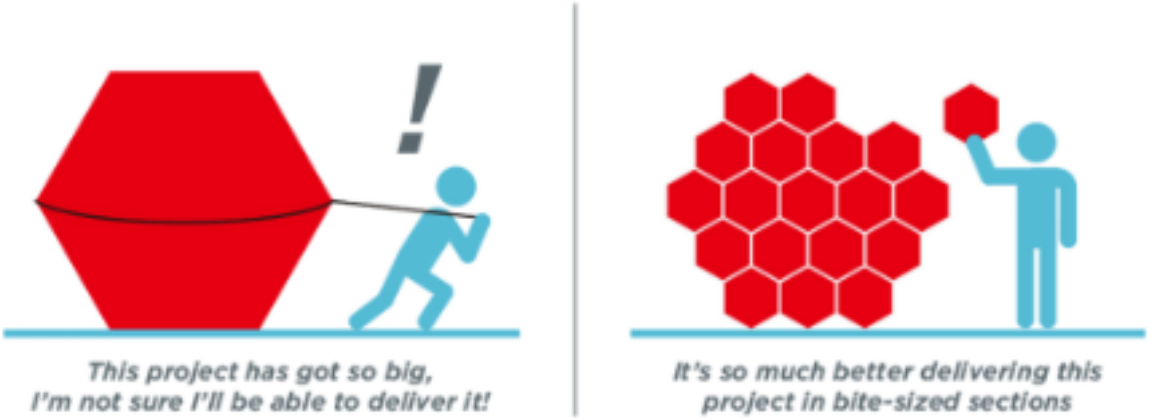
1.3. 문서의 목적

DWorks 를 처음 접하는 구성원이 DWorks를 이해하기 위한 기반 지식 습득 및 구성원간의 원활한 소통이 가능하도록 합니다.

2. 기반 지식

2.1. MSA (MicroServices Architecture)

MSA 하면 항상 함께 얘기되는게 Monolithic 아키텍처에요.



- Monolithic (EER)
    - 작은 코드 베이스에서 시작하여, 초기 개발단계에서 개발/테스트가 용이한 장점이 있지만, 어플리케이션 구성 단위가 커서 작은 변경에도 민감하다는 단점이 있어요!
- MSA (DWorks)
    - Monolithic 에 비해 복잡한 구성이라는 단점이 있지만, 서비스별 적합한 기술스택적용이 가능하고, 확장성, 배포용이성, 서비스의 독립성으로 인해 특정 서비스의 장애가 전체 서비스의 장애로는 전파되지 않는 장점이 있어요!

2.2. Spring Boot

Spring Framework 기반으로 한 어플리케이션을 빠르게 설정/개발 및 실행 할 수 있는 환경 제공해요

1. 복잡한 XML 대신, Java 기반의 설정을 선호해요
    a. @Configuration, @bean 등 Annotation 등을 사용하여 가독성이 좋은 코드들을 만들어요! 예를들어 작성한 무언가가 Bean으로써 동작을 하게 하고 싶다면, XML 설정파일을 찾아서 수정하는게 아니라, @Bean 을 추가하는것만으로 동일한 효과를 누리지요!

```Java
@Configuration // 해당 클래스가 설정 클래스임을 나타내요

public class DatabaseConfig {

  @Bean // 해당 메서드가 빈을 생성하는 메서드임을 나타내요

  public DataSource dataSource() {

    DriverManagerDataSource dataSource = new DriverManagerDataSource();

    return dataSource;

  }

}
```

2. 의존성 관리를 간편하게 할수 있어요
    a. 예를들어 웹 어플리케이션을 개발하고 싶다면, spring-boot-starter-web 디펜던시를 설정하는 것만으로도 필요한 라이브러리들을 자동으로 포함 할 수 있어요!

```Unset
<dependency>
      <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

3. Embedded Server를 지원해요
    a. 기본은 Tomcat 이고, 간단한 설정으로 Jetty, Undertow 등으로 변경 가능해요! 서버를 선택/설정하는 시간을 줄여주지요!

2.3. Spring Boot Actuator

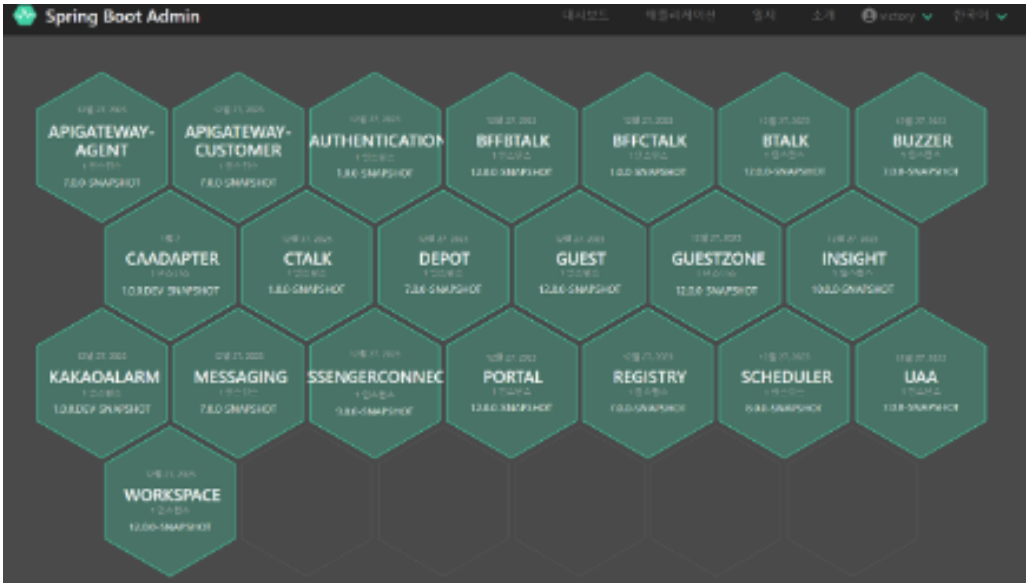
Spring Boot 기반 어플리케이션의 상태, 환경설정, 메모리 사용량, 쓰레드 풀 상태등 모니터링 및 관리를 쉽게하기위한 기능(엔드포인트)을 제공해요

예를들어 상태를 체크하고 싶다면 /actuator/health 엔드포인트를 이용하여 아래와 같은 응답내용으로 확인할 수 있어요

```
Unset
{
 "status": "UP"
 ...
}
```

2.4. Spring Boot Admin

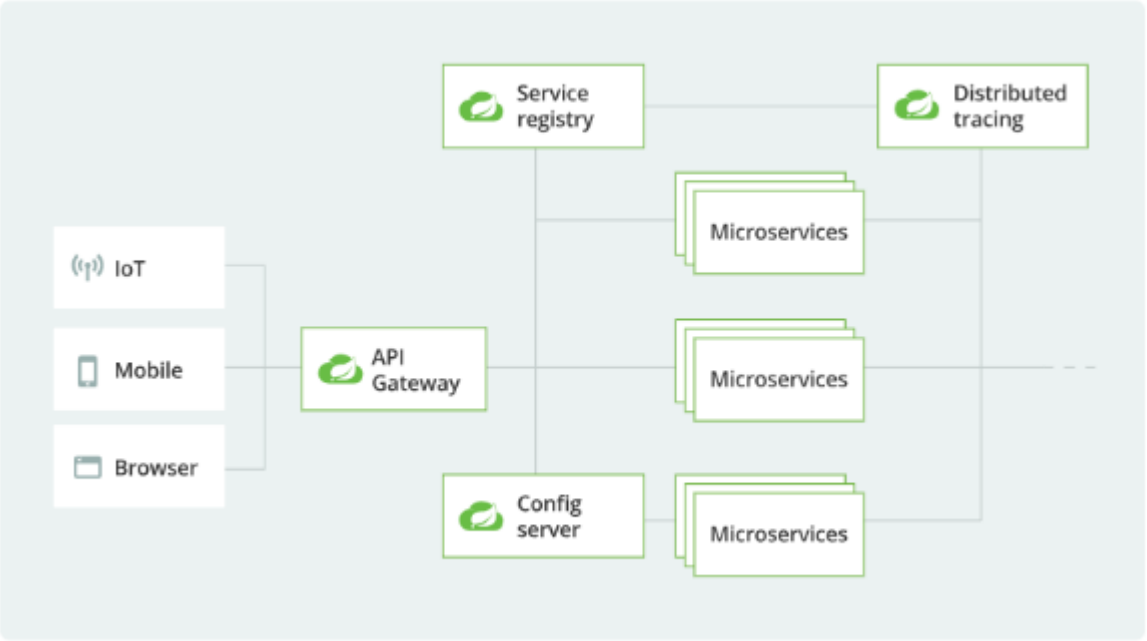
Spring Boot Actuator 엔드포인트들을 기반으로 시각화 및 모니터링기능의 웹 어플리케이션이에요, 그 외에도 Actuator 엔드포인트를 통한 설정 변경등의 작업이 가능해요



2.5. Spring Cloud

MSA를 구현하는데 필요한 도구/기능을 제공해요

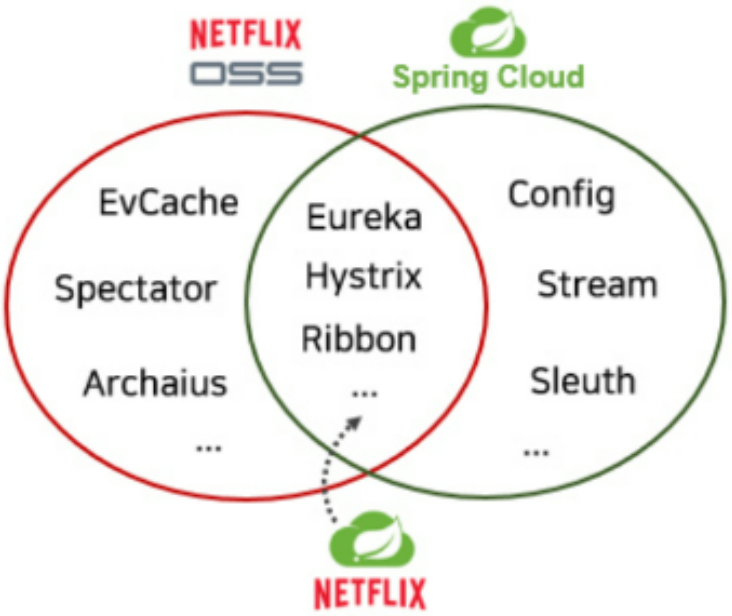
주로 SpringBoot를 기반으로 Config Server, Service Registry/Discovery, Load Balancing, Circuit Breaker, API Gateway 등을 지원하고, Spring Data (feat. JPA), Spring Cloud Security 등도 함께할 수 있어요.



2.6. Netflix OSS (Open Source Software)

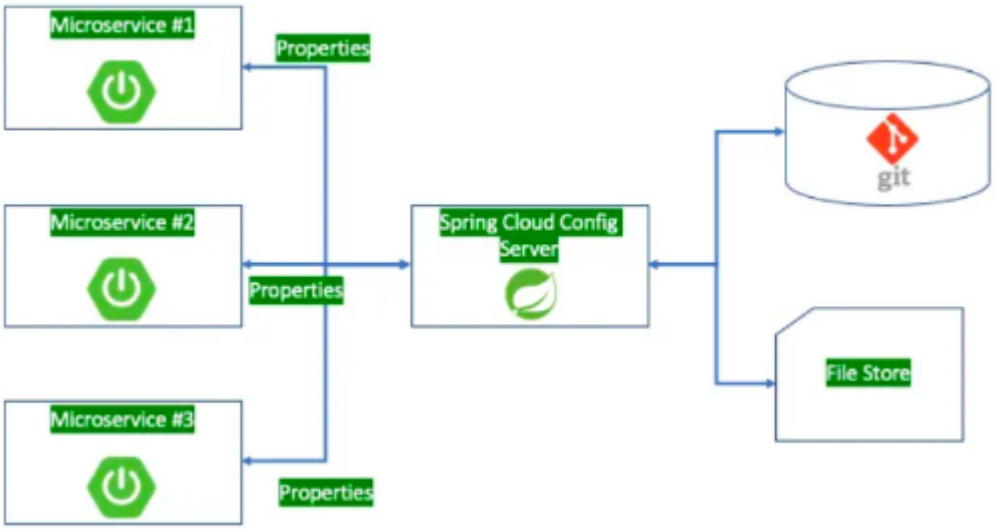
분산 시스템환경에서, 안정성, 확장성, 유연성을 보장하기위한 도구 모음집이에요, Eureka, Zuul, Ribbon, Hystrix, Feign 등으로 구성되어있답니다.

DWorks 는 Zuul → Spring Cloud Gateway, Feign → Spring Cloud OpenFeign, Ribbon → Spring Cloud Loadbalancer 으로 대체하여 사용하고있어요.



2.7. Spring Cloud Config (Config Server)

각 서비스에서 필요로 하는 설정을 Config Server 에 저장하고, 동적으로 로드 할 수 있도록 지원해요



MSA 구조에서 특정 서비스의 설정을 변경해야한다면 각각의 환경에서 설정을 변경해야 할까요?
환경이 추가되어 서비스를 하나 더 추가 구성해야 한다면?
너무나 귀찮고 복잡한 작업이에요!

Spring Cloud Config 는 여러 서비스에서 사용가능한 서버/클라이언트를 제공하여, 각 서비스는 Config Server 에서 자신의 설정 정보를 로드할 수 있답니다! (feat. Auto Scaling)

DWorks 에서는 Registry 서비스가 Config Server 역할을 담당해요!

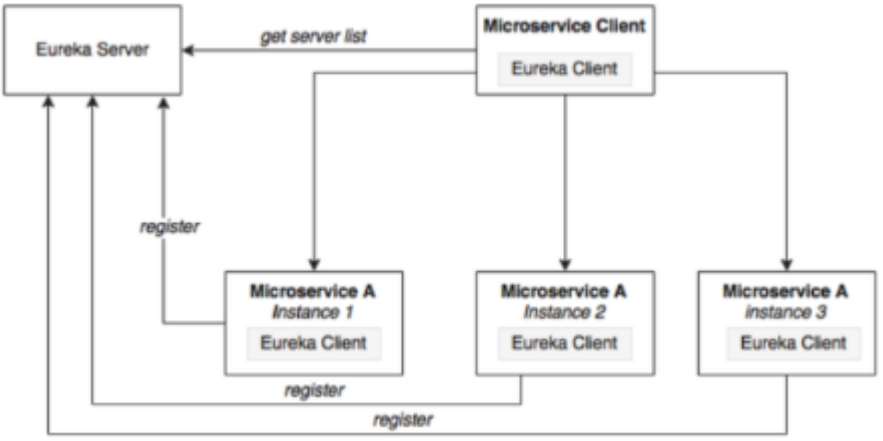
2.8. Eureka (Service Registry/Discovery)

Eureka 는 Service Registry/Discovery 의 구현체랍니다

각 서비스들의 위치 및 상태 정보를 관리할 수 있어요

- 각 서비스는 실행하면서 Eureka Server 에 자신의 위치와 상태를 등록해요
- 각 서비스에서 다른 서비스에 대해 통신이 필요하다면 Eureka Client 를 통해 Eureka Server 에서 해당 서비스의 위치를 찾아서 통신해요
    - 우리는 어떤 서비스로의 요청이 필요하지만 결정하고, 해당 서비스의 위치가 어디인지, 알 필요가 없어요!
      MSA 구조에서 서비스간의 결합을 최소화하고, 독립적으로 확장/변경이 가능하도록 하는 핵심 아이디어중에 하나가 이것이에요!

DWorks 에서는 Registry 서비스가 Eureka Server 역할을 담당해요!

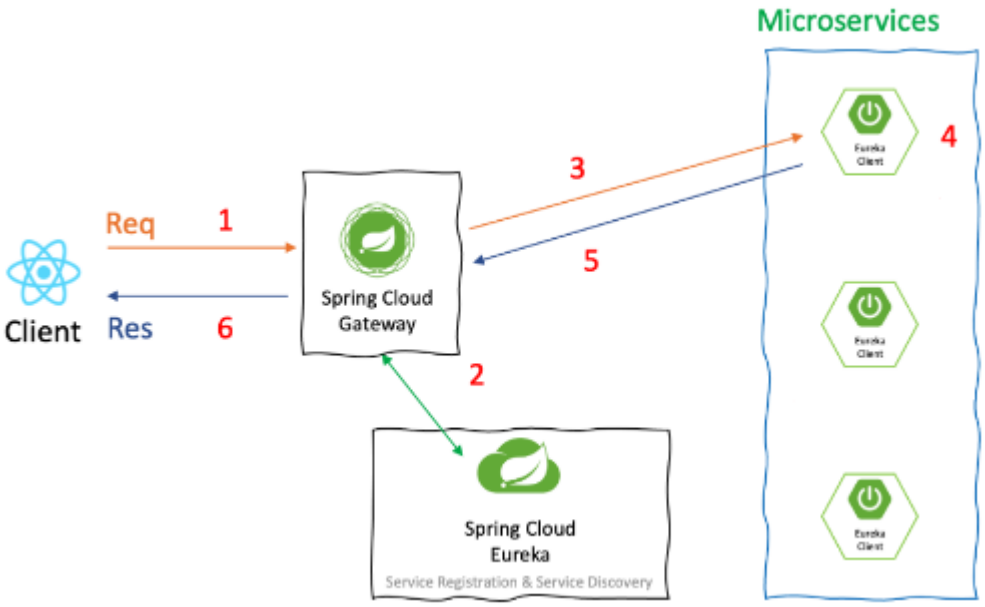


2.9. Spring Cloud Gateway (API Gateway)

Spring Cloud Gateway 는 API Gateway 의 구현체랍니다

다른 서비스에 대한 통신이 필요시 Eureka 를 통해 위치를 확인하고,
라우팅/필터링/보안처리등을 담당한답니다.

DWorks 에서는 ApiGateway-Agent, ApiGateway-Customer 서비스가 APIGateway 역할을 담당해요!



2.10. Open Feign

서비스간의 통신을 간편하게 작성할수 있도록 하는 도구에요

```java
@FeignClient(
    name = "scheduler",
    configuration = {FeignConfiguration.class}
)
public interface EurekaFeignAdapter {
    @GetMapping("/kafka/job")
    KafkaJobRdoListRdo findAll();
}
```

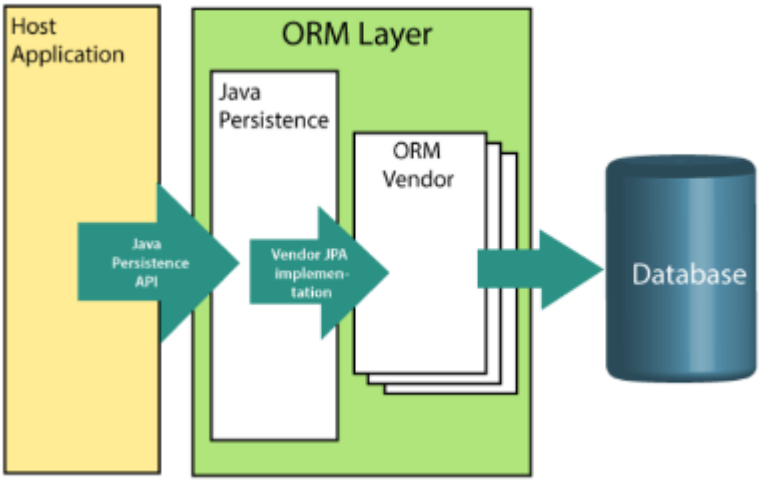
2.11. JPA (Java Persistence API)

JPA 는 Java 에서 제공하는 데이터베이스 표준 인터페이스에요

객체와 데이터베이스의 매핑을 담당하여 관계형 데이터베이스(RDBMS)와의 불일치를 해결해요

JPA의 추상화를 제공하는 Spring Data JPA 모듈을 통해 Spring 환경에서 쉽고, 유연하게 JPA를 사용할 수 있어요

실제로는 Hibernate 와 같은 Spring Data JPA 구현체를 통해 데이터베이스와 상호작용을 해요



2.12. Java 8

1. Lambda Expressions
    a. 함수형 프로그래밍 개념을 도입하였으며, 함수를 변수처럼 사용할수 있어요, 가장 큰 변화중 하나라고 할 수 있어요!

```java
// 기존의 익명 내부 클래스

Runnable runnable = new Runnable() {

  @Override

  public void run() {

    System.out.println("Hello World!");

  }

};

// 람다 표현식으로 간결하게 표현

Runnable runnableLambda = () -> System.out.println("Hello World!");
```

2. Stream API
    a. Collection 을 효율적이고, 간결한 코드로 사용할수 있도록 지원해요

```java
List<String> strings = Arrays.asList("apple", "banana",
"orange");
strings.stream()
    .filter(s -> s.startsWith("a"))
    .map(String::toUpperCase)
    .forEach(System.out::println);
```

3. Optional
    a. NullPointerException 을 방지하여 안정성을 높이기위해 사용해요
    b. 일반적으로 메서드의 반환 유형이나 컬렉션에서 사용해요, Optional 을 남용하면 코드의 가독성을 저해하고, 복잡성을 증가시킬수 있기때문이에요!

```java
Optional<String> nonEmptyOptional = Optional.of("Hello");
Optional<String> emptyOptional = Optional.ofNullable(null);
```

3. 인프라

3.1. Kafka (메시지 브로커)

분산처리 환경에서 대용량의 메시지(데이터)를 안전하게 처리하기 위한 분산 스트리밍 도구에요

분산 아키텍처, 확장성, 신뢰성있는 데이터 전달등의 특징을 기반으로, 대규모 로그 수집, 이벤트 소싱등에서 사용된답니다.

Kafka 를 사용함으로써 서비스 간의 결합도를 낮추고, 서비스의 독립성과 확장성을 향상시킬 수 있어요!

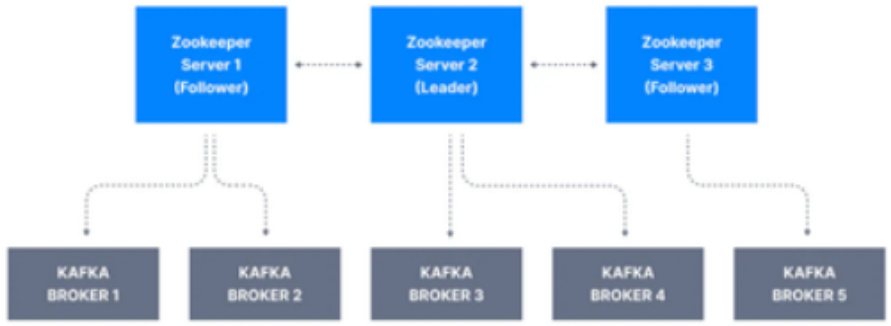
구성

1. Broker
    a. 클러스터를 구성하는 하나의 노드(서버)를 Broker라고 해요
2. Producer
    a. 메시지를 생성하는 주체이며, 메시지는 특정 Topic 으로 보내져요
    b. Kafka 클러스터중 하나의 Broker 에게 보내져요
    c. Topic 은 메시지의 주소라고 생각하면 됩니다.
3. Consumer
    a. 메시지를 소비하는 주체이고, 특정 Topic 의 메시지를 가져와서 처리합니다.

b. Consumer Group 에 속할 수 있고, 여러 Consumer 가 동시에 처리 할 수도 있어요

3.2. ZooKeeper (Coordination System)

Kafka 클러스터 구성 정보, Broker의 상태(Leader/Follower 선출) 및 Topic 의 구성등을 저장하고 관리해요



Zookeeper in Kafka

3.3. Elasticsearch (검색엔진)

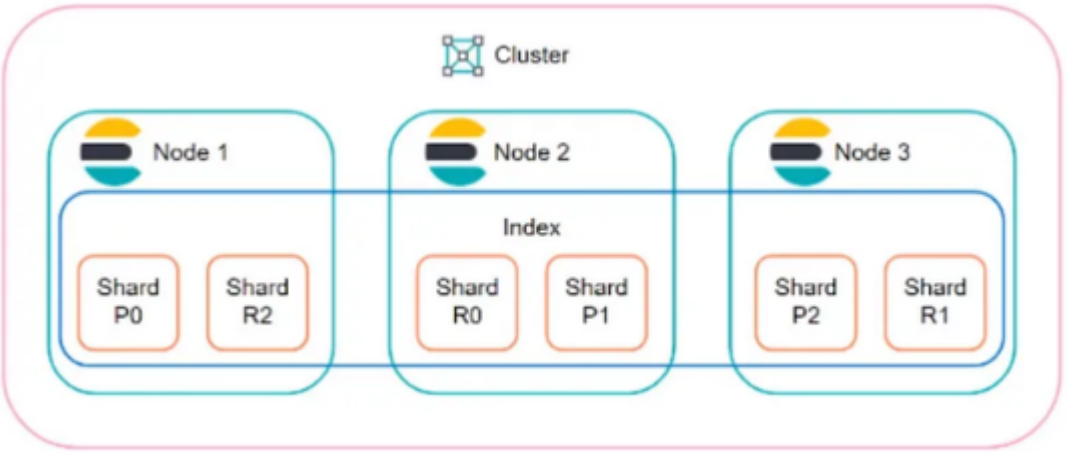
Apache Lucene 기반의 분산 검색 엔진이에요

아래와 같은 특징을 가지고 있어요.

1. 분산환경에서 여러 노드에 대량의 데이터를 분산저장 및 처리 할수 있어요
2. 실시간으로 데이터를 색인하고 검색하는데 강점을 가지고 있어요
3. RESTful API 를 통해 데이터를 저장/색인/분석/조회 할수있어요

구성

1. index
    a. 데이터를 저장하고 검색하기위한 논리적인 공간이에요, message, ticket, user 같은 것들이죠
    b. 하나 이상의 shard로 구성이 되요
2. shard
    a. 데이터를 분산하여 저장하고, 검색작업을 병렬로 처리하기위한 기본 단위에요, 클러스터내의 물리적인 노드들에 분산되어 저장된답니다
    b. Primary shard, Replica Shard 로 구성되며, 예를들어 Primary shard 가 3, Replica shard 가 1로 설정된다면 데이터가 3개로 나뉘어져 저장되고, 각각 1개씩의 Replica shard 로 구성되요



3.4. Hazelcast (Cache, IMDG- In-Memory Data Grid)

분산형 메모리 스토어에요

DWorks 는 Spring Cache 저장소로 사용하고 있어요

Spring Cache 는 분산 캐싱을 추상화하여 제공하며, @Cacheable, @CacheEvict, @CachePut 과 같은 Annotation 을 사용하여 캐싱기능을 쉽게 구현 가능하게해줘요

영구적인 데이터 저장소는 DB/Elasticsearch 를 이용하고, 성능향상을 위한 캐싱처리는 Hazelcast 를 이용해요.

C팀에서 준비한 자료도 함께 보아주세요!

Part_02_기술요소.docx