Software Requirements Specification

Searchgoose

Team 9

201411296 이선명

201411312 장하나

201711375 권혁규

201711413 이유진

Table of Contents

1. Introduction

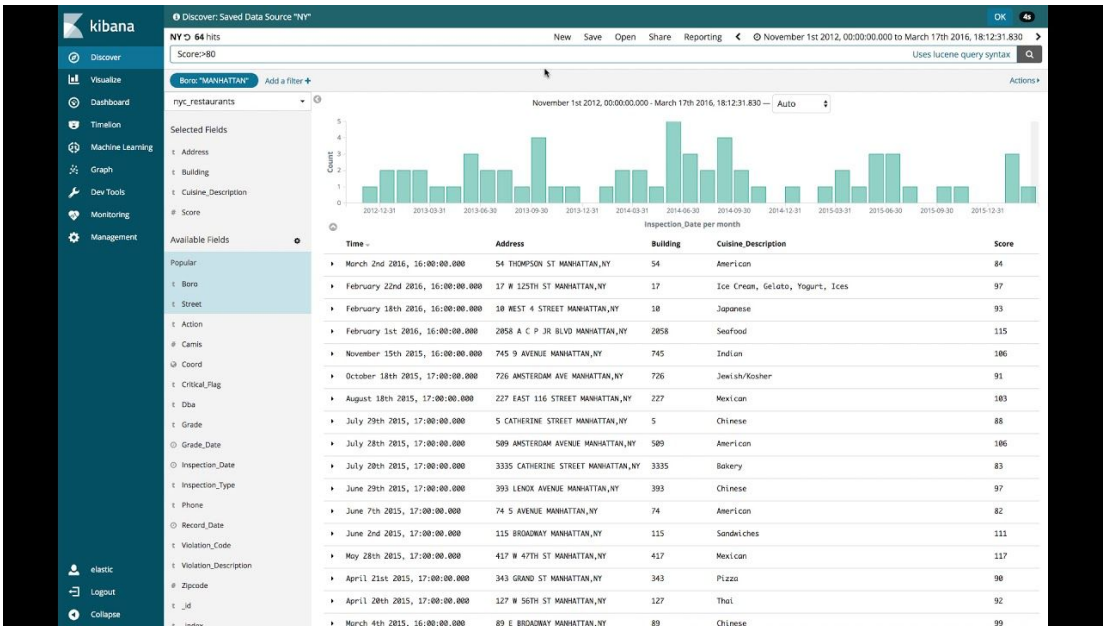
1.1 Purpose

이 문서의 목적은 분산 RESTful 검색엔진 시스템을 개발하는 내용을 담고 있다.
searchgoose가 무엇인지 궁금해하는 사람들에게 내용을 설명하기 위한 목적이다.

1.2 Scope

Searchgoose는 오픈소스 분산 RESTful 검색엔진이자 분석엔진이다. 데이터 인덱싱, 검색, 분석 기능을 제공한다. real-time search 기능을 제공하며 빠른 검색을 위해 structured text, unstructured text, numerical data를 효율적으로 저장, 인덱싱 한다. 이를 통해 사용자는 정보를 조회, 집계함으로써 데이터에서 트랜드, 패턴을 찾을 수 있다.
주요 사용자는 개발자이며, 기존 Relational Database가 갖는 문제점인 자유롭지 않은 샤딩과 클러스터링, 단순한 문자열 검색에서 벗어나서 문자열 형태소 분석, 역인덱싱, 샤딩과 클러스터링을 활용하여 속도와 데이터 핸들링에 대한 유연성을 제공하여 개발자는 만들고자 하는 애플리케이션의 비즈니스 로직에 집중할 수 있다. 아래와 같은 적용 사례가 예상된다.

- 앱이나 웹 사이트에 search box를 추가하고자 할 때
- Log나 metric, 혹은 security event data를 저장하고 분석하고자 할 때
- 머신 러닝을 사용하는 과정에서 실시간으로 데이터를 모델링 하고자 할 때
- Business workflow를 자동화하는 과정에서 storage engine이 필요할 때
- geographic information system(GIS)에서 공간 데이터를 관리 및 통합, 분석하고자 할 때
- Bioinformatics 분야에서 유전 정보를 가공하고 저장하고자 할 때



더 나아가 Restful 한 속성과 JSON 문서 기반의 통신을 지원하기에 HTTP 프로토콜을 이용해 시각화 클라이언트와도 손쉽게 연동이 가능하도록 개발한다. 검색, 그리고 aggregation의 집계 기능을 이용해 문서, 집계 결과 등을 불러와 웹 도구로 시각화를 할 수 있다.

1.3 Definitions, acronyms, and abbreviations

이 문서에서 사용될 용어, 약어 등의 설명

(https://www.elastic.co/guide/en/elasticsearch/reference/current/glossary.html)

클러스터 (Cluster)

클러스터는 하나 이상의 노드(서버)가 모인 것이며, 이를 통해 전체 데이터를 저장하고 모든 노드를 포괄하는 통합 색인화 및 검색 기능을 제공한다. 어떤 노드가 어느 클러스터에 포함되기 위해서는 이름에 의해 클러스터의 구성원이 되도록 설정된다. 클러스터에 하나의 노드만 있는 것은 유효하며 문제가 없다. 또한 각자 고유한 클러스터 이름을 가진 독립적인 클러스터를 여러 개 둘 수도 있다.

노드 (Node)

노드는 클러스터에 포함된 단일 서버로서 데이터를 저장하고 클러스터의 색인화 및 검색 기능에 참여한다. 노드는 클러스터처럼 이름으로 식별되는데, 기본 이름은 시작 시 노드에 지정되는 임의 UUID(Universally Unique IDentifier)이다.네트워크의 어떤 서버가 클러스터의 어떤 노드에 해당하는지 식별한다. 노드는 클러스터 이름을 통해 어떤 클러스터의 일부로 구성될 수 있으며, 하나의 클러스터에서 원하는 개수의 노드를 포함할 수 있다.

인덱스(Index)

색인은 다소 비슷한 특성을 가진 문서의 모음이다. 이를테면 고객 데이터에 대한 색인, 제품 카탈로그에 대한 색인, 주문 데이터에 대한 색인을 각각 둘 수 있다. 색인은 이름(모두 소문자여야 함)으로 식별되며, 이 이름은 색인에 포함된 문서에 대한 색인화, 검색, 업데이트, 삭제 작업에서 해당 색인을 가리키는 데 쓰인다. 단일 클러스터에서 원하는 개수의 색인을 정의할 수 있다.

도큐먼트 (Document)

문서는 색인화할 수 있는 기본 정보 단위이다. 예를 들어 어떤 단일 고객, 단일 제품, 단일 주문에 대한 문서가 각각 존재할 수 있다. 이 문서는 JSON(JavaScript Object Notation) 형식이다. 하나의 색인/유형에 원하는 개수의 문서를 저장할 수 있으며, 문서가 물리적으로는 어떤 색인 내에 있더라도 문서는 색인화되어 색인에 포함된 어떤 유형으로 지정되어야 한다.

필드 (Field)

문서에는 필드, 혹은 키-값 쌍들의 리스트를 가지고 있다. 이 값은 간단한 스칼라값일수도 있고 배열이나 오브젝트같은 구조체일수도 있다. 필드는 관계형 데이터베이스 테이블의 컬럼과 비슷하다. 각 필드의 매핑에는 필드 타입이 있어 저장될 수 있는 데이터의 종류를 나타낸다. 매핑은 어떻게 필드의 값이 분석되어야 하는지 정의할 수 있게 해준다.

ID

문서의 ID는 서로 다른 문서끼리 구분할 수 있게 해준다. 문서의 인덱스 혹은 ID는 유일해야 한다. 만약 ID가 제공되지 않았다면 자동으로 생성될 것이다.

샤드 (Shards) & 리플리카 (Replica)

색인은 방대한 양의 데이터를 저장할 수 있으며, 이 데이터가 단일 노드의 하드웨어 한도를 초과할 수도 있다. 예를 들어 10억 개의 문서로 구성된 하나의 색인에 1TB의 디스크 공간이 필요할 경우, 단일 노드의 디스크에서 수용하지 못하거나 단일 노드에서 검색 요청 처리 시 속도가 너무 느려질 수 있다.

이러한 문제를 해결하고자 색인을 이른바 샤드(shard)라는 조각으로 분할하는 기능을 제공한다. 색인을 생성할 때 원하는 샤드 수를 간단히 정의할 수 있다. 각 샤드는 그 자체가 온전한 기능을 가진 독립적인 "색인"이며, 클러스터의 어떤 노드에서도 호스팅할 수 있다.

샤딩은 무엇보다도 2가지 이유로 중요하다.

- 콘텐츠 볼륨의 수평 분할/확장이 가능해진다.
- 작업을 (어쩌면 여러 노드에 위치한) 여러 샤드에 분산 배치하고 병렬화함으로써 성능/처리량을 늘릴 수 있다.

샤드가 분산 배치되는 방식 및 그 문서가 다시 검색 요청으로 집계되는 방식의 메커니즘이 필요하며, 언제든 오류가 일어날 가능성이 있는 네트워크/클라우드 환경에서는 어떤 이유에서든 샤드/노드가 오프라인 상태가 되거나 사라지게 될 경우에 대비하여 페일오버 메커니즘을 마련하는 것이 매우 유익하고 바람직하다. 이러한 취지에서 색인의 샤드에 대해 하나 이상의 복사본을 생성할 수 있는데, 이를 리플리카 샤드(replica shard), 줄여서 리플리카라고 한다.

이처럼 리플리카를 만드는 복제는 무엇보다도 2가지 이유로 중요하다.

- 샤드/노드 오류가 발생하더라도 고가용성을 제공한다. 따라서 리플리카 샤드는 그 원본인 기본 샤드와 동일한 노드에 배정되지 않는다.
- 모든 리플리카에서 병렬 방식으로 검색을 실행할 수 있으므로 검색 볼륨/처리량을 확장할 수 있다.

요약하자면 각 색인은 여러 개의 샤드로 분할할 수 있으며, 하나의 색인은 복제하지 않거나(리플리카 없음) 1회 이상 복제할 수 있다. 복제되면 각 색인은 기본 샤드(복제 원본 샤드)와 리플리카 샤드(기본 샤드의 복사본)를 갖게되고, 샤드 및 리플리카의 수는 색인별로, 색인 생성 시점에 정의할 수 있다. 색인이 생성된 다음 언제라도 탄력적으로 리플리카의 수를 변경할 수 있으나, 샤드 수는 사후 변경이 불가하다.

1.4 References

- https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

- http://blevesearch.com/docs/Terminology/

- https://esbook.kimjmin.net/

1.5 Overview

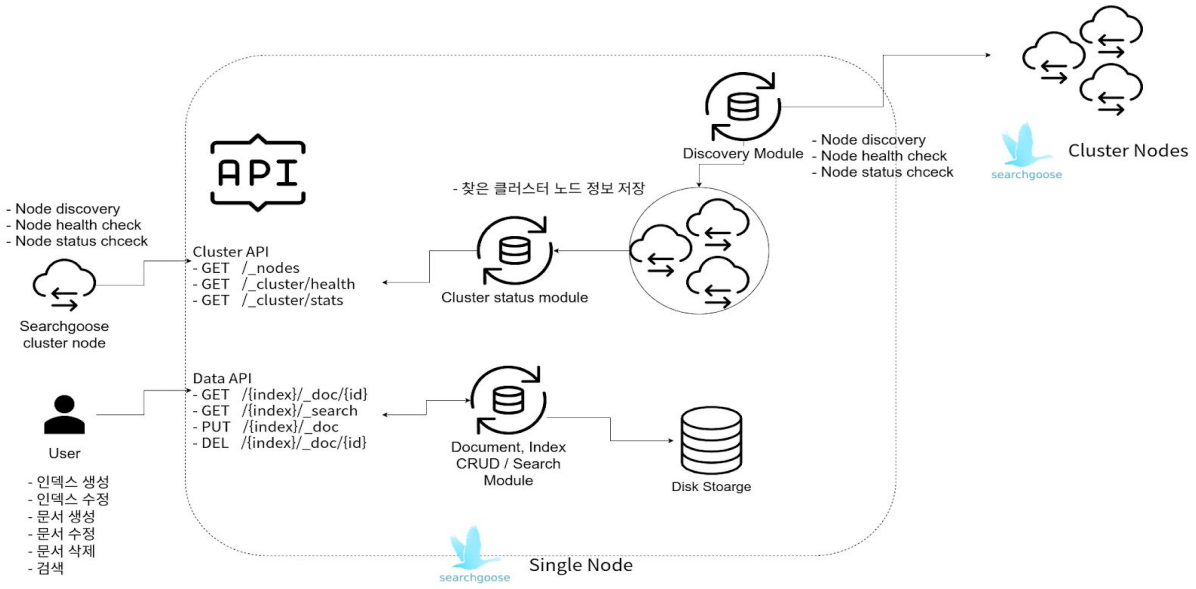
이 다음 챕터인 Overall Description 부분에서 본 제품의 기능에 대한 개요를 제공한다. 여기서 일반적 요구사항 및 기술적 요구사항에 대한 배경을 설명한다.
그리고 3번째 챕터인 Specific Requirements 부분에서 개발자가 이를 구현하기 위한 기능 상세를 기술용어들로 설명한다.

2. Overall description

2.1 Product perspective

Searchgoose는 개발자가 만들 application의 data layer component 역할을 담당하는 독립된 product로서 동작한다. 따라서 별도의 database를 필요로 하지 않는다. Linux 운영체제에서 실행될 것을 가정한다.

Single node 관점



Single node 관점에서 searchgoose 노드는 http api를 통해 아래와 같은 행위를 수행한다.

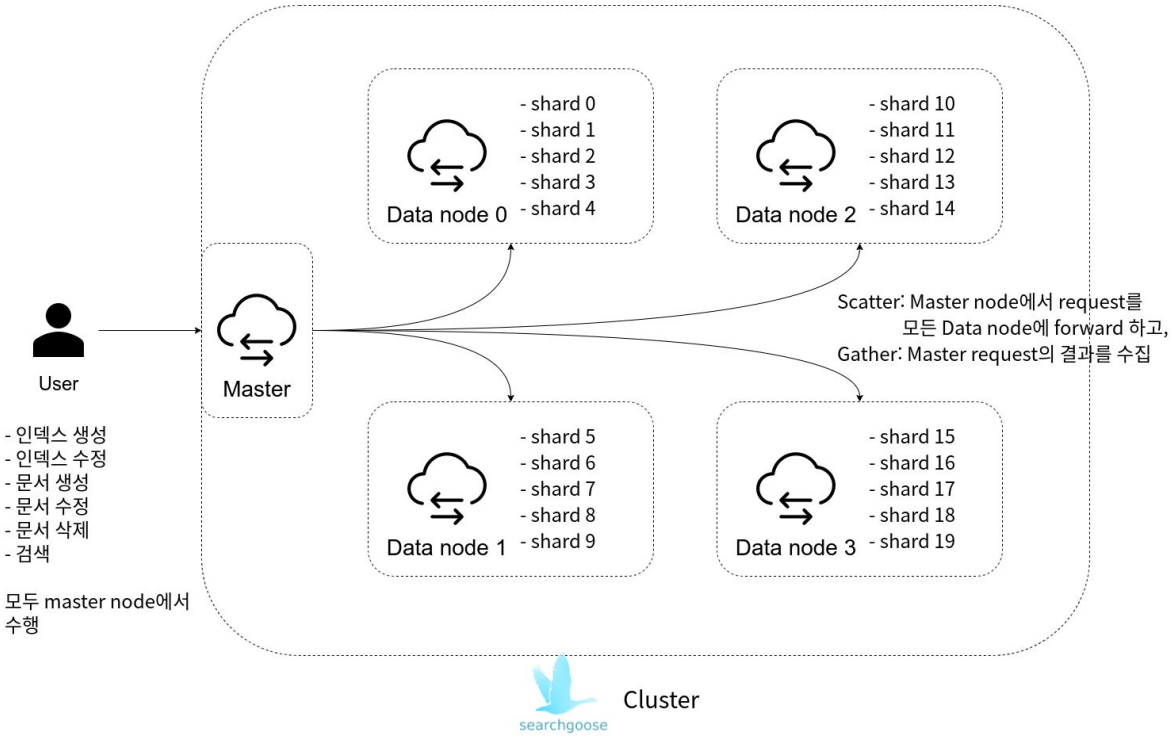
- 클러스터 서비스 디스커버리

Searchgoose는 복수의 노드가 협업할 수 있도록 설계한다. 그러므로 각 노드는 현재 클러스터에 참여하는 노드가 얼마나 있는지, 각 노드의 ip 주소와 port등의 정보를 알고 있어야 하고, TCP 연결을 맺을 수 있어야 한다. 나아가 후술할 Cluster 관점에서의 master node election에 참여하게 된다.

- 데이터 저장, 인덱싱

사용자가 문서를 저장하고, 조회할 수 있어야 한다. 따라서 CRUD, 검색, 집계 등의 데이터 관련 작업을 처리하는 기능을 제공한다. Searchgoose는 데이터 저장 및 인덱싱을 위해서 bleve (https://github.com/blevesearch/bleve)라는 오픈소스 라이브러리를 사용한다.

Cluster 관점



Cluster 관점에서 searchgoose 노드들은 master node과 data node로 나뉜다.

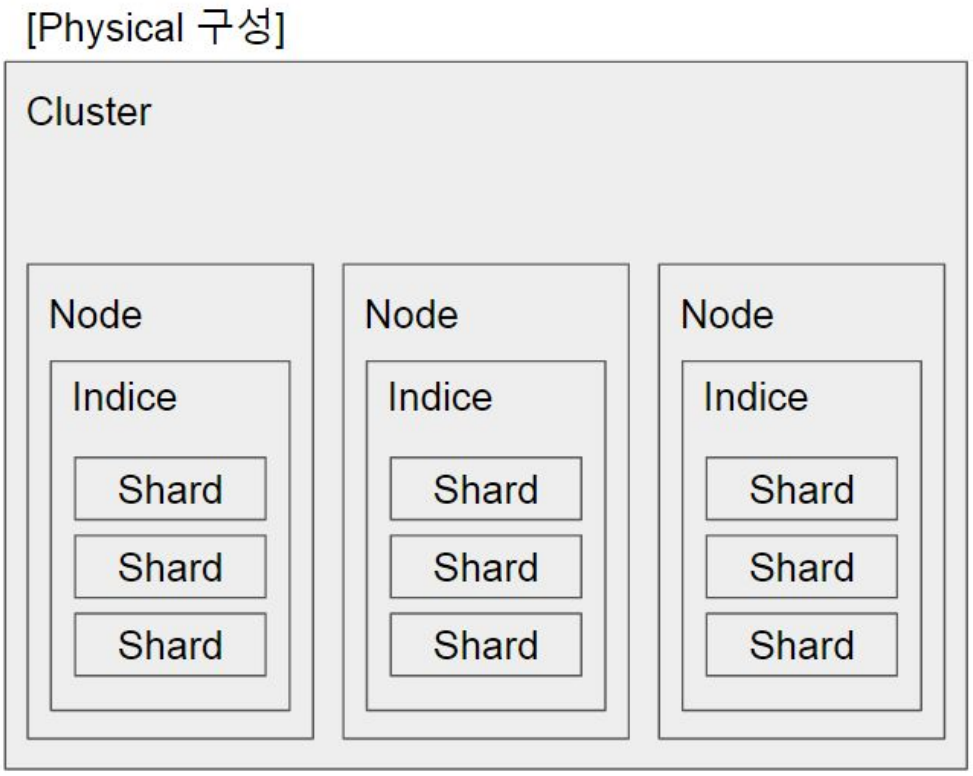
마스터 노드 (Master node)

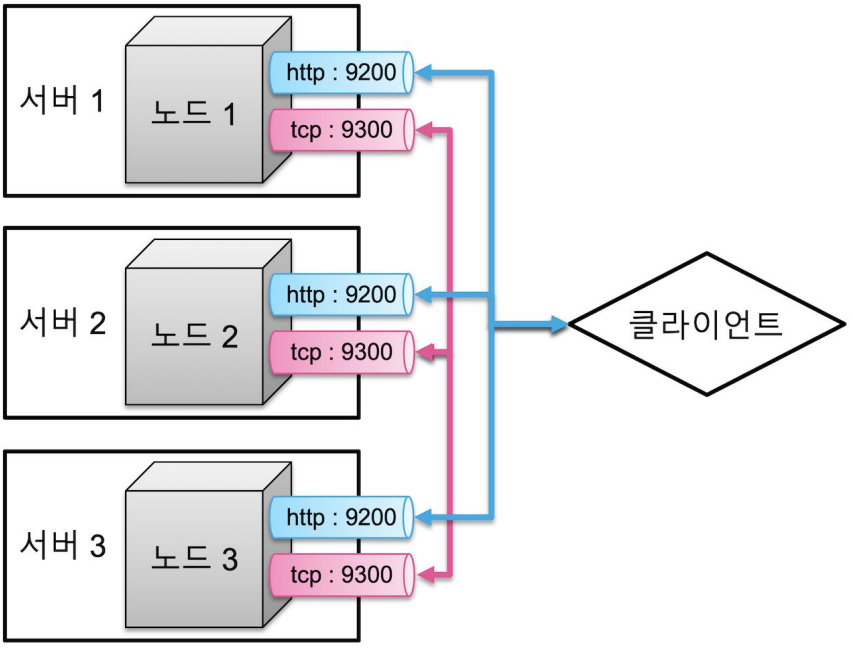
마스터 노드는 인덱스 생성 및 제거와 같이 가벼운 클러스터 대상의 작업들을 수행하며 어떤 노드가 클러스터의 일부인지 추적하고 어떠한 노드에 샤드를 할당할지 결정한다. 클러스터가 좋은 상태를 유지하기 위해서는 안정된 마스터 노드가 중요하다. 투표 전용 노드가 아닌 모든 마스터가 될 수 있는 노드들은 선거 절차를 통해 마스터 노드가 될 수 있다.

데이터 노드 (Data node)

데이터 노드는 사용자가 인덱싱을 완료한 문서들이 저장된 샤드들을 가지고 있다. 데이터 노드는 CRUD, 검색, 집계 등의 데이터 관련 작업을 관리한다. 이 작업들은 많은 IO, 메모리, CPU 자원을 필요로 한다. 이 자원들의 사용을 모니터링하고 과부하가 걸릴 때 더 많은 데이터 노드를 추가하는 것이 중요하다.

클러스터링(Clustering)





Searchgoose의 노드들은 클라이언트와의 통신을 위한 http 포트(9200~9299), 노드 간의 데이터 교환을 위한 tcp 포트 (9300~9399) 총 2개의 네트워크 통신을 열어두고 있다.
일반적으로 1개의 물리 서버마다 하나의 노드를 실행하는 것을 권장하고 있으며, 3개의 다른 물리 서버에서 각각 1개 씩의 노드를 실행하면 각 클러스터는 위와 같이 구성된다.

디스커버리(Clustering)

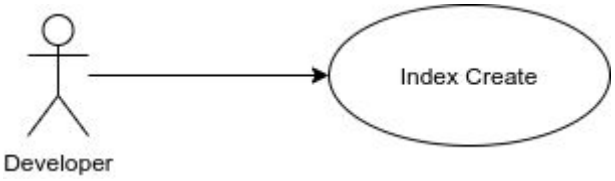
노드가 처음 실행 될 때 같은 서버, 또는설정된 네트워크 상의 다른 노드들을 찾아 하나의 클러스터로 바인딩 하는 과정을 디스커버리 라고 한다.

설정에 있는 주소 순서대로 노드가 있는지 여부를 확인하며, 노드가 존재하는 경우 클러스터의 이름을 확인하고 일치하면 같은 클러스터로 바인딩하고 종료하며, 일치하지 않는 경우 설정의 다음 주소를 확인하며 반복한다.
반면, 노드가 존재하지 않는 경우 설정의 다음 주소를 확인하며 반복한다. 그리고 설정에 존재하는 주소가 끝날 때 까지 노드를 찾지 못한 경우 스스로 새로운 클러스터 시작한다.

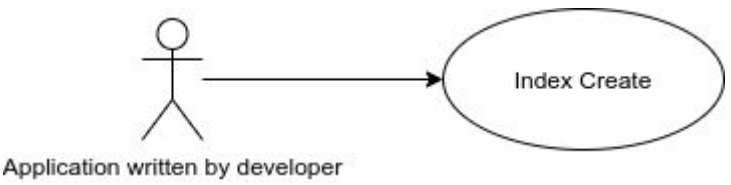
위에서 묘사한 searchgoose의 행위 및 기능들은 모두 HTTP RESTful api를 통해 제공된다. 따라서 searchgoose API를 사용하기 위해서는 인터넷이 안정적으로 연결되어 있어야하며 대량의 데이터를 저장하고 검색할 수 있도록 충분한 용량의 스토리지, 메모리가 필요하다.

2.2 Product functions

2.2.1 인덱스 생성



개발자는 searchgoose http api를 호출하여 인덱스라고 칭하는 Document 필드의 집합인 스키마 및 메타데이터를 정의해 둘 수 있다. Searchgoose는 이를 통해 후술할 데이터 삽입시 데이터가 배치될 샤드와 리플리카를 정의해 둘 수 있다.

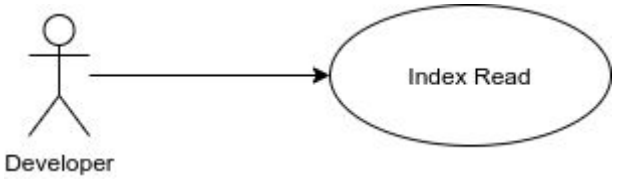


Application 도메인에 따라서 개발자가 만든 application이 인덱스 생성을 할 수도 있다. Searchgoose HTTP API 호출을 통해서 손쉽게 이를 달성 할 수 있다. 자동화된 data pipeline이 구축되어 있는 경우가 이 경우에 해당한다.

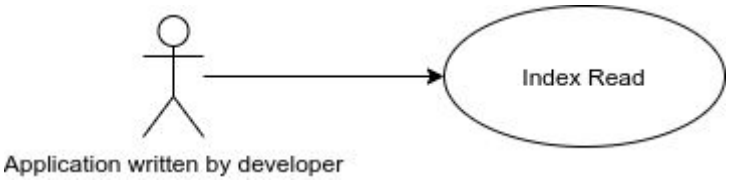
인덱스 생성 동작의 명세는 아래와 같다.

1. 개발자 또는 application이 searchgoose의 Index create API를 호출하여 생성할 인덱스의 정보를 전달한다.
2. 정보를 전달받은 Searchgoose node가 master node일 경우 적절한 샤드와 리플리카 수를 지정하고 클러스터의 임의의 샤드를 배치할 data node에게 index create request를 forward한다.. Data node인 경우 클러스터의 단일 master node에게 forward 한다.
3. 2번의 작업이 모두 끝나면 성공했음을 개발자 또는 application에게 알린다.

2.2.2 인덱스 조회

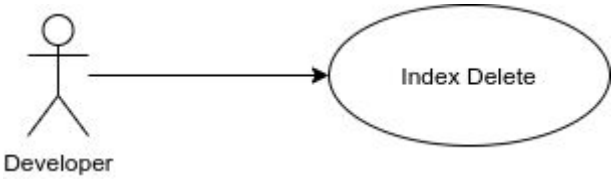


개발자는 searchgoose http api를 호출하여 이미 생성되어 있는 인덱스 정보를 조회할 수 있다. 주로 디버깅 목적으로 활용된다.

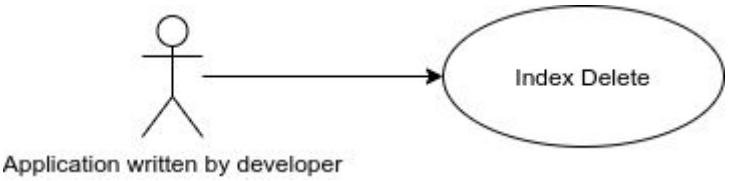


Application 도메인에 따라서 개발자가 만든 application이 인덱스 정보를 조회하는 경우가 있을 수 있다. 인덱스 정보에 따라 쿼리가 달라지는 Data visualization에서 흔하게 수행되는 경우라고 예상된다.

2.2.3 인덱스 삭제

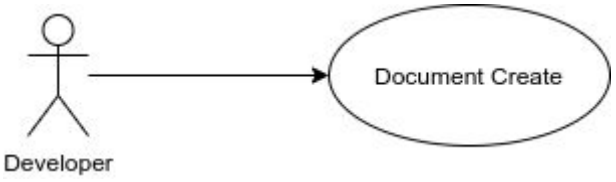


개발자는 searchgoose http api를 호출하여 이미 생성되어 있는 인덱스를 삭제할 수 있다. 이 경우 해당 인덱스에 속해있는 모든 문서들이 삭제된다.

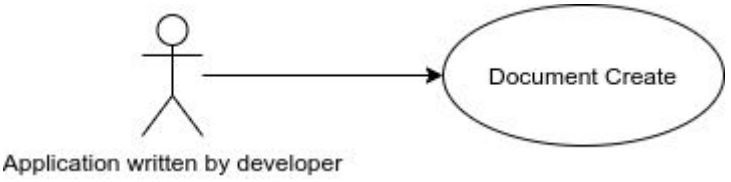


Application 도메인에 따라서 개발자가 만든 application이 인덱스를 삭제할 수 있다. 다만 인덱스 삭제는 곧 클러스터가 취급하던 데이터를 모두 삭제하는 것에 직결되고 관리가 어려워지므로 이는 권장되는 방법이 아니다.

2.2.4 문서 삽입



개발자는 searchgoose http api를 호출하여 특정 인덱스에 해당하는 데이터를 삽입할 수 있다. 이후 저장한 데이터를 개발자가 조회 또는 검색하거나, 개발자가 만든 application이 조회 또는 검색하게 된다.

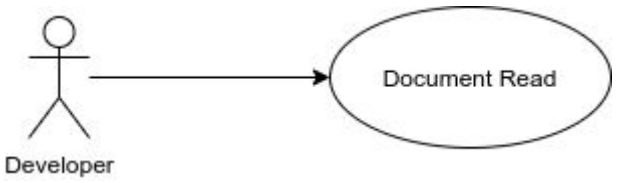


Application 도메인에 따라서 개발자가 만든 application이 데이터를 삽입할 수도 있다.
1만건을 넘는 대량의 데이터의 경우 보통 이 경우가 될 것이다.

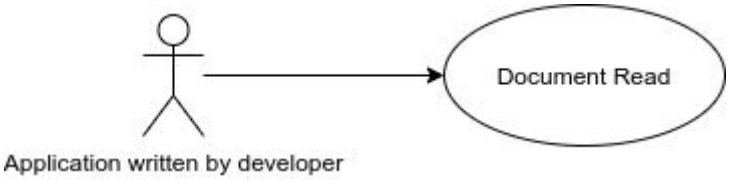
문서 삽입 동작의 정의는 아래와 같다.

1. HTTP 프로토콜을 사용한 API를 통해 searchgoose는 JSON 데이터를 입력받는다.
2. Seachgoose는 입력값에 대한 검증을 거친다. 입력값이 곧 Document의 field로 채워지게되는데, 이 field가 해당하는 인덱스의 field 스키마와 일치하는지 비교한다. 일치하지 않는다면 error 임을 알리고 동작을 종료한다.
3. 검증을 성공적으로 마쳤다면 document에 ID를 부여하고 해당 ID를 input으로 하는 hash function의 output 에 따라 샤드 번호를 계산한다. 최종적으로 해당 샤드를 갖는 데이터 노드에게 문서 삽입 요청을 forward 한다.
4. 문서 삽입 요청을 전달받은 데이터 노드는 자신의 디스크 저장소에 데이터를 저장 및 인덱싱한다.
5. 저장 및 인덱싱이 끝나면 해당 노드는 마지막으로 개발자 또는 application에게 성공적으로 동작이 완료되었음을 알린다.

2.2.5 문서 조회



개발자는 searchgoose http api를 호출하여 이미 생성되어 있는 문서를 조회할 수 있다. 주로 디버깅 목적에서 활용된다.



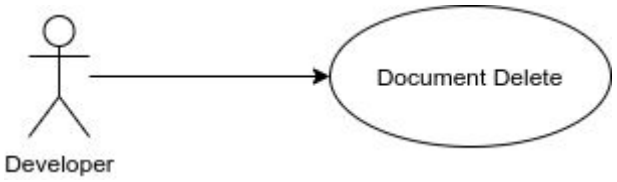
Application 도메인에 따라서 개발자가 만든 application이 문서를 조회할 수 있다. 일반적인 사용패턴이 될 것으로 예상된다.

문서 조회 동작의 정의는 아래와 같다.

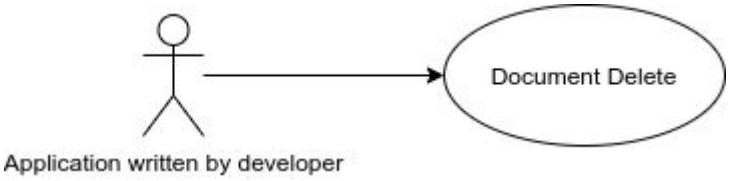
1. 개발자 또는 application은 문서 조회 API를 통해 조회할 문서의 ID를 전달한다.
2. searchgoose 노드는 전달받은 ID를 input으로 하는 hash function의 output에 따라 샤드 번호를 계산한다. 최종적으로 해당 샤드를 갖는 데이터 노드에게 문서 조회 요청을 forward한다.
3. 문서 조회 요청을 수행하여 해당 문서를 찾아낸다.

4. 성공적으로 완료되었음과 함께 찾은 문서를 response로 보낸다.

2.2.6 문서 삭제



개발자는 searchgoose http api를 호출하여 이미 생성되어 있는 문서를 삭제할 수 있다. 실수 혹은 application 버그로 인해 잘못 추가된 문서를 삭제하는 경우에 해당한다.

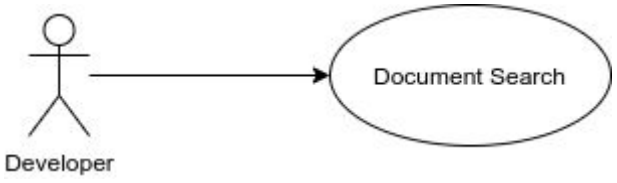


Application 도메인에 따라서 개발자가 만든 application이 문서를 삭제할 수 있다. 일반적인 사용패턴이 될 것으로 예상된다.

문서 삭제 동작의 정의는 아래와 같다.

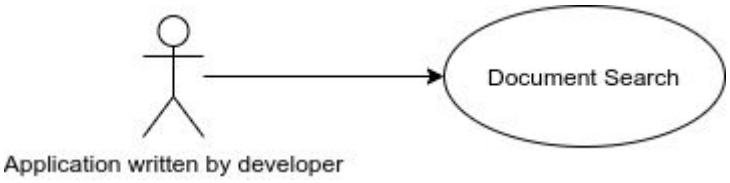
1. 개발자 또는 application은 문서 조회 API를 통해 삭제할 문서의 ID를 전달한다.
2. searchgoose 노드는 전달받은 ID를 input으로 하는 hash function의 output에 따라 샤드 번호를 계산한다. 최종적으로 해당 샤드를 갖는 데이터 노드에게 문서 삭제 요청을 forward한다.
3. 문서 삭제 요청을 수행하여 해당 문서를 삭제한다.
4. 성공적으로 완료되었음을 개발자 또는 application에게 response로 알린다.

2.2.7 문서 검색



개발자는 searchgoose http api를 활용하여 생성되어 있는 문서들 중 특정 keyword와

매칭되는 문서들을 검색할 수 있다.

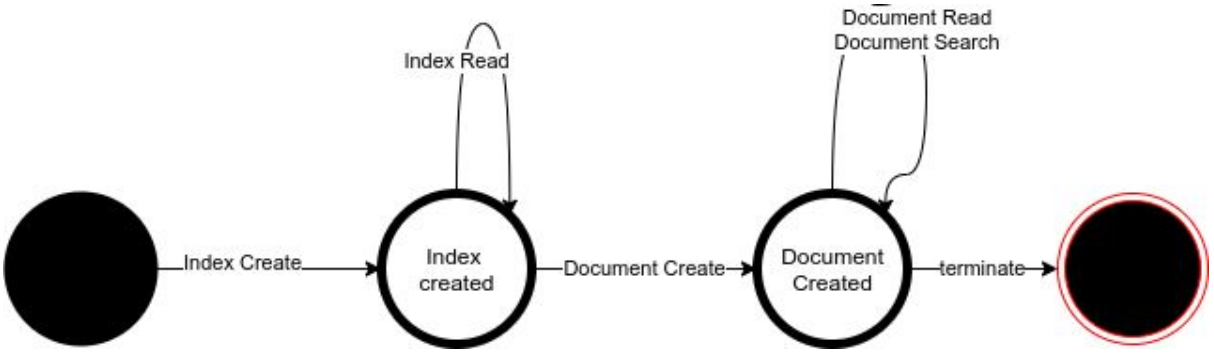


Application 도메인에 따라서 개발자가 만든 application이 문서를 검색할 수 있다. 검색엔진 또는 일반 business application의 사용패턴이 될 것으로 예상된다.

문서 검색 동작의 정의는 아래와 같다.

1. 개발자 또는 application은 문서 조회 API를 통해 검색 조건이 될 인덱스 및 인덱스의 field, 그리고 keyword를 전달한다.
2. Searchgoose는 전달받은 인덱스를 통해 해당 인덱스의 샤드를 갖고 있는 모든 데이터 노드들에게 문서 검색 요청을 forward 한다.
3. 각 데이터 노드들은 각자의 샤드안에서 해당 field에서 keyword와 매칭되는 문서를 모두 찾아서 forward 한 노드에게 전달한다.
4. 각 데이터 노드들로부터 전달받은 문서들을 취합해서 최종적으로 개발자 또는 application에 응답한다.

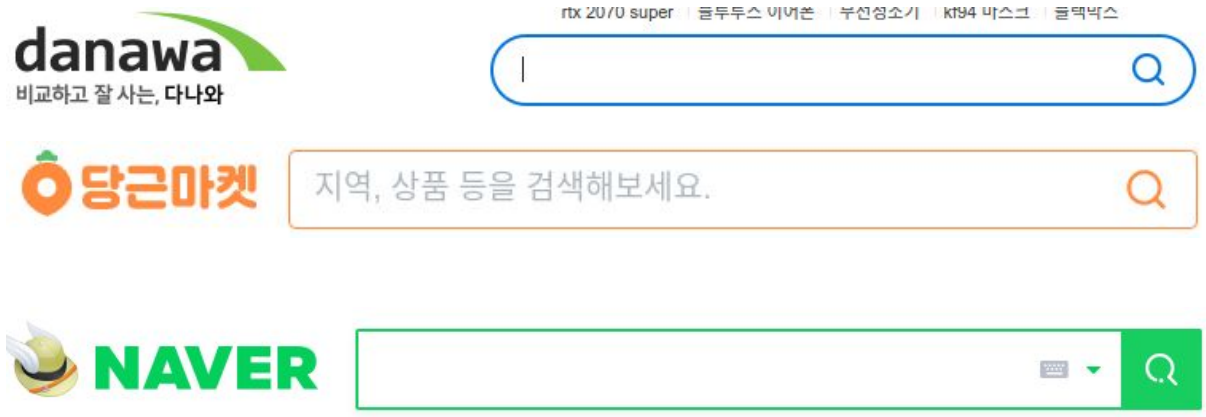
이렇게 인덱스 생성/조회/삭제, 문서 생성/조회/삭제/검색 등의 기본적인 기능들을 알아보았다.



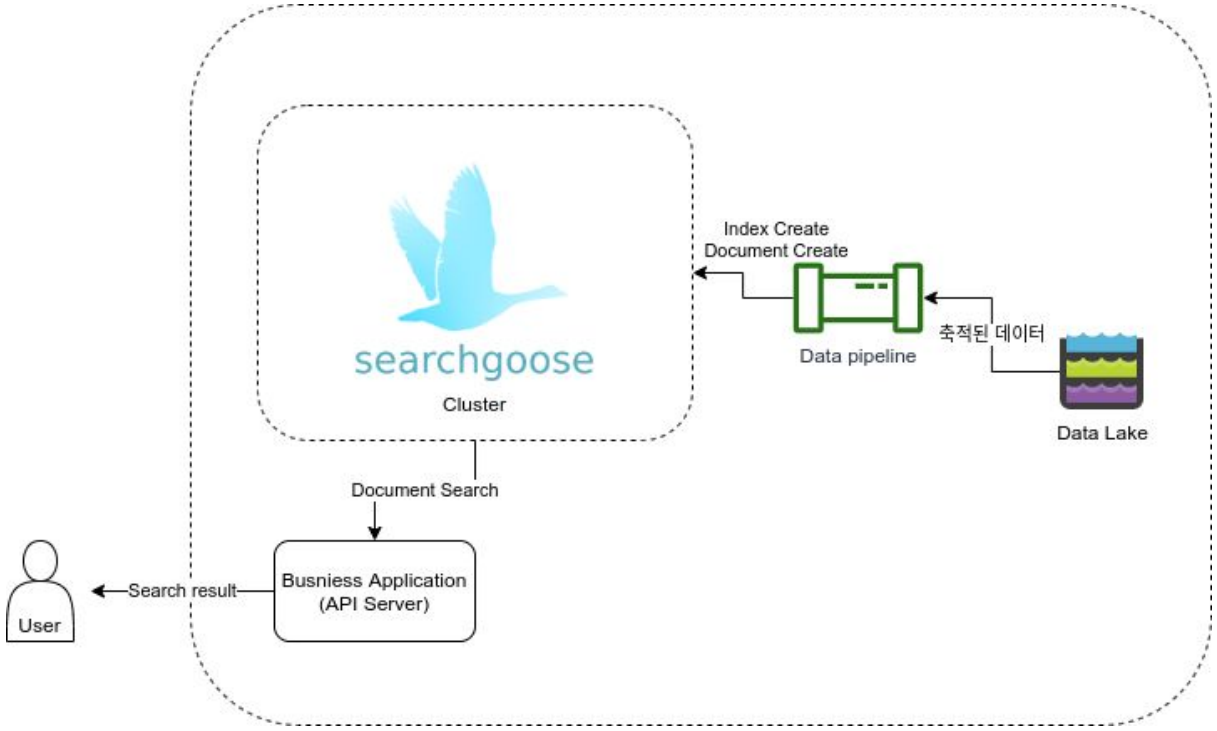
최종적으로 위와같은 finite state machine이 적용되는 것을 알 수 있다.

아래부터의 설명들은 searchgoose를 활용하여 개발자들이 개발할 수 있을 business application 및 searchgoose 활용 사례를 나열한다.

2.2.8 검색엔진



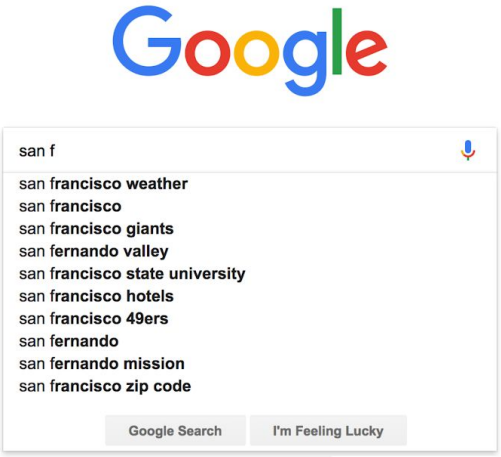
(Business application 예시이미지, 사진은 각각 다나와, 당근마켓, 네이버)



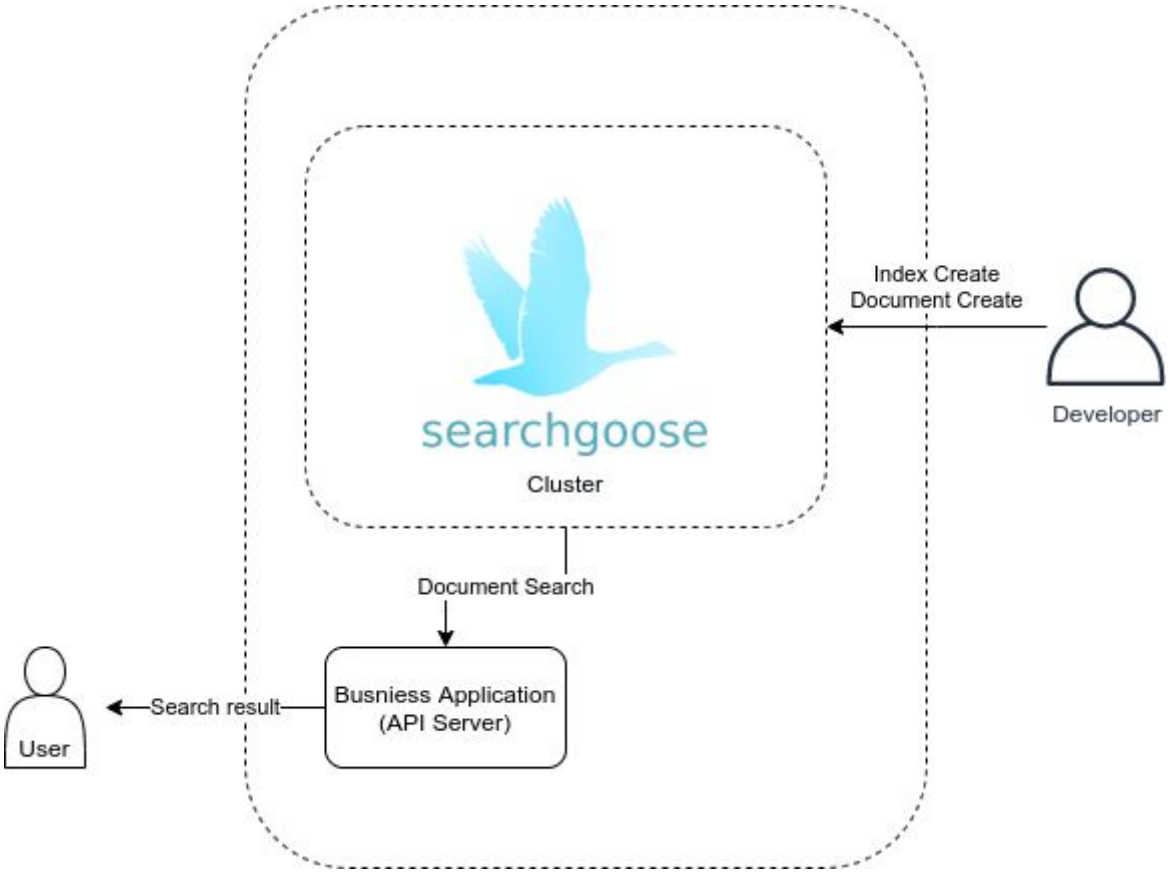
(Diagram)

각 business application은 서비스를 지속함에 따라 축적되는 유저 또는 상품 데이터가 존재한다. 이를 활용하기 위한 가장 첫 단계는 검색이다. 데이터와 searchgoose 기능을 활용하기 위해 축적된 데이터를 searchgoose에 삽입하는 과정이 필요하다. 따라서 searchgoose를 활용하고 싶은 개발자들은 자동화된 data pipeline을 통해 인덱스 생성과 문서 생성 과정을 거친다. 그리고 유저에게 노출되는 business application에서 문서 검색기능을 활용함으로써 검색엔진을 구현할 수 있게 된다.

2.2.9 검색어 자동완성



(Business application 예시이미지, 사진은 Google 검색엔진)

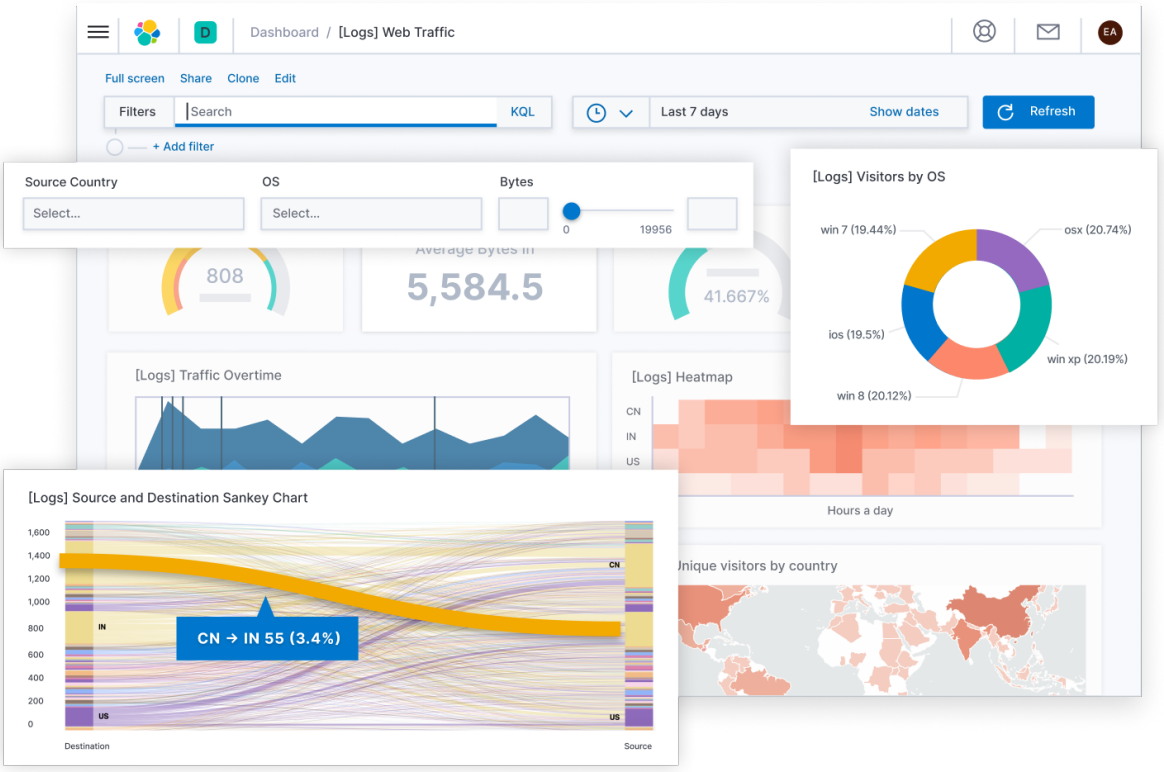


(Diagram)

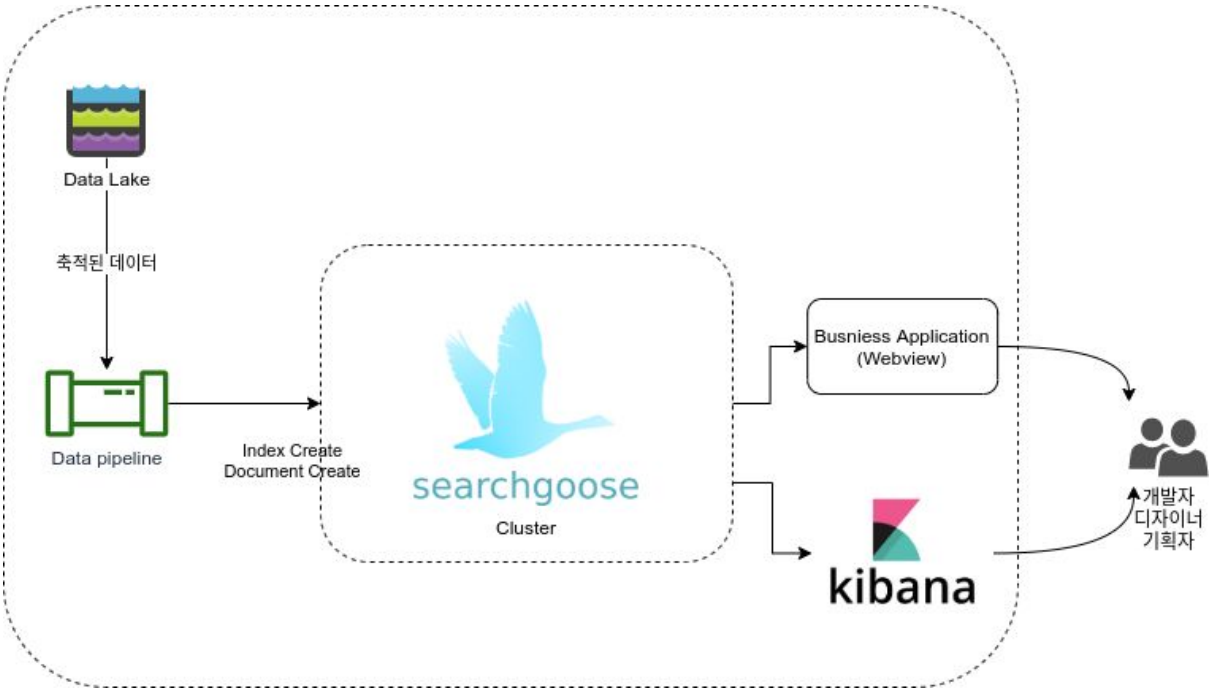
검색엔진에 있어서 검색어 자동완성에도 searchgoose의 기능을 활용할 수 있다. 단어와 사전

데이터, 유저가 많이 검색하는 키워드를 모두 특정 인덱스에 저장하고, 유저가 검색하려고 할때마다 business application이 인덱스에 검색 요청을 함으로써 자동완성 기능을 제공할 수 있다. 사전 등의 데이터는 그리 많지 않을 것으로 예상된다. 따라서 개발자가 직접, 혹은 application을 개발해서 인덱스 생성 및 문서 생성을 해낼 수 있을 것이다.

2.2.10 데이터 시각화 (Data visualization)



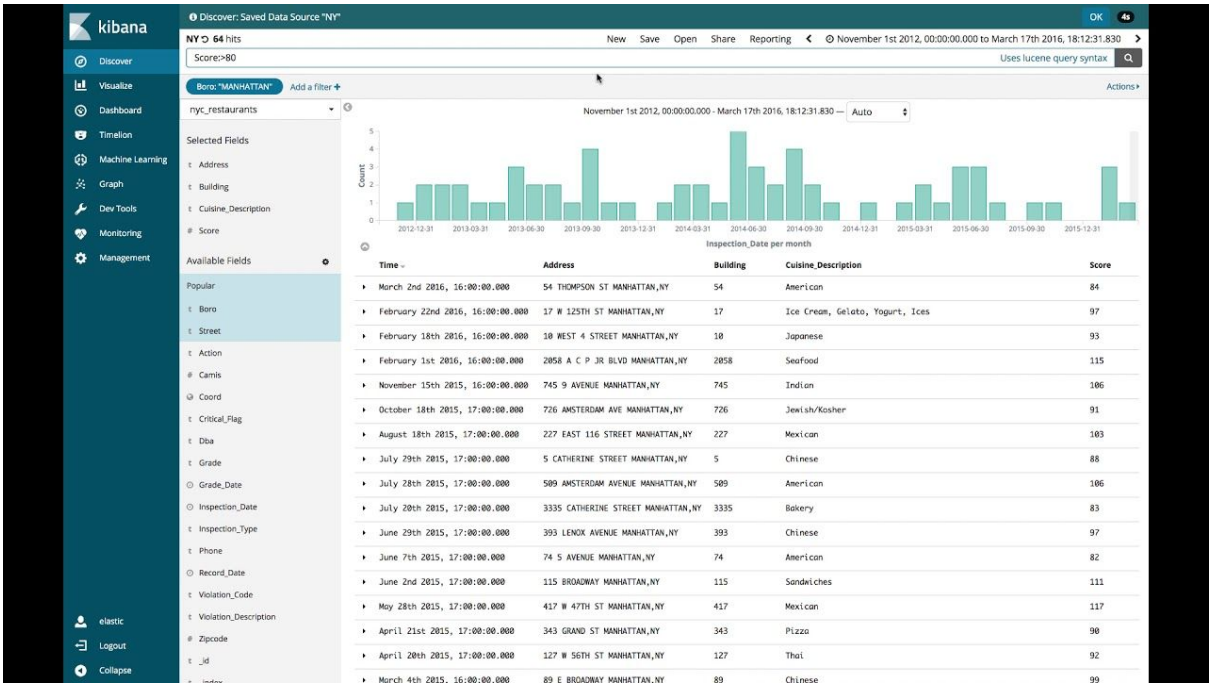
(Business application 예시 이미지, 사진은 kibana)



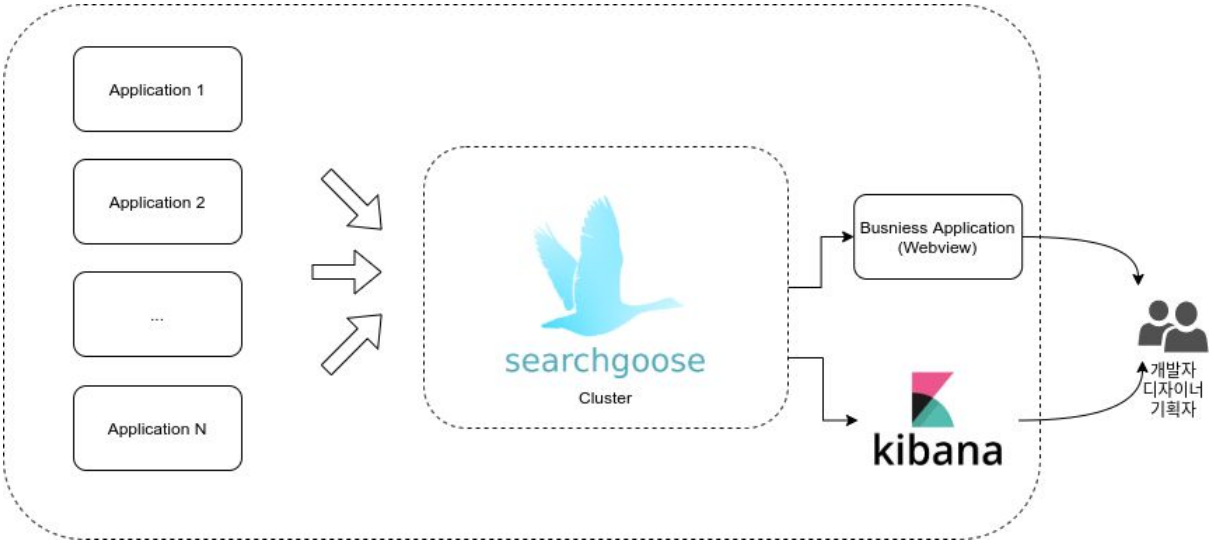
(Diagram)

각 business application은 서비스를 지속함에 따라 축적되는 유저 또는 상품 데이터가 존재한다. 개발자, 디자이너, 혹은 기획자는 이 축적된 데이터 속에서 유의미한 데이터를 꺼내서 유의미한 통계를 원할 것이다. 유의미한 통계를 위해 우선 축적된 데이터를 searchgoose에 삽입하는 과정이 필요하다. 따라서 searchgoose를 활용하고 싶은 개발자들은 자동화된 data pipeline을 통해 인덱스 생성과 문서 생성 과정을 거친다. 이후 문서 조회 기능과 검색기능을 활용해서 찾아낸 데이터를 시각화 할 수 있다. 시각화하기 위해 필요한 도구를 개발자가 직접 만들수도 있겠지만, 데이터 시각화 도구인 kibana를 사용하여 유의미한 통계를 만들 수도 있을 것이다.

2.2.11 중앙 집중형 로그 시스템 (Centralized logging system)



(Business application 예시 이미지, 사진은 kibana)



(Diagram)

Business application을 운영하다 보면 수많은 application log를 만나게 된다. 그리고 이 application log를 통해 개발자는 운영을 위해 디버깅, 트랜잭션 상태, 성능등을 측정하게 된다. 이를 위해서 로그를 분류 또는 검색을 하게 되는데, application log는 unstructured text로 저장되는 경우가 많다. 개발자는 searchgoose의 인덱스 생성, 문서 생성 및 문서 검색기능을 활용하여 손쉽게 분류, 검색을 할 수 있고 손쉬운 devops가 가능해진다.

2.3 User characteristics

기대되는 사용자는 일반적인 application 개발에 필요한 개발 지식이 충분한 사람으로써, 본 제품을 이용하여 자신이 만드는 application에 데이터 검색 기능, 데이터 분석 기능을 추가하고 싶은 사람으로 예상된다. Relational Database와 NoSQL의 차이를 명확하게 이해하고 있고, 분산 시스템에 대한 기초 지식을 갖고 있는 개발자들을 칭한다.

2.4 Assumptions and dependencies

인터넷이 연결되어 있고 Searchgoose 서버와 통신 가능한 모든 곳에서 API 사용 가능, 500GB 이상의 디스크 스토리지와 16GB 이상의 메인메모리, 4 core 이상의 CPU를 보유한 컴퓨터에 설치되는 것으로 가정한다.

3. Specific requirements

3.1 External interfaces

Searchgoose를 사용하기 위해서는 REST API를 이용해야 한다. 다양한 명령들을 실행하는데 이때 JSON Over HTTP를 사용해 JSON으로 인코딩된 데이터를 인풋과 아웃풋으로 주고받는다. 또한 도큐먼트 저장에도 JSON을 이용한다. API를 사용할 때에는 지정된 포맷의 URL을 써야 한다.

3.2 Functions

Searchgoose에서는 단일 도큐먼트 별로 고유한 URL을 갖으며, 도큐먼트에 접근하는 URL은 다음과 같다.

http://<호스트>:<포트>/<인덱스>/_doc/<도큐먼트 id>

1. **Document**

도큐먼트 API는 Index, Get, Delete, Update로 구성된 CRUD 작업을 수행한다.

유저는 해당 URL을 통해 Searchgoose의 리소스에 접근하며, 성격 및 기능에 따라 유저에게 제공되는 인터페이스를 HTTP EndPoint로 제공하고 있다.

a. **Index**

	name	**interfaces**	**descriptions**
1	데이터 입력	PUT my_index/_doc/1	my_index 인덱스에 도큐먼트 id가 1인 데이터 입력
2	데이터 생성	POST my_index/_doc	my_index 인덱스에 도큐먼트 id를 임의로 데이터 생성
3	데이터 생성 후 입력	PUT my_index/_create/1	my_index 인덱스에 도큐먼트 id가 1인 데이터가 없으면 생성한 뒤 데이터 입력
4	데이터 생성 후 입력	POST my_index/_create/1	3과 동일한 작업 수행

b. **Get**

	name	**interfaces**	**descriptions**
1	데이터 조회	GET my_index/_doc/1	my_index 인덱스에 id가 1인 도큐먼트의 데이터 조회
2	존재하는 데이터 조회	HEAD my_index/_doc/1	my_index 인덱스에 id가 1인 도큐먼트가 있는지 확인 후 데이터 조회

c. Delete

	name	interfaces	descriptions
1	데이터 삭제	DELETE my_index/_doc/1	도큐먼트 또는 인덱스 단위의 데이터 삭제

d. Update

	name	interfaces	descriptions
1	데이터 갱신	POST my_index/_update/1	id가 1인 도큐먼트의 내용을 업데이트

- 데이터 입력

my_index 인덱스에 도큐먼트 id가 1인 데이터 입력

```
PUT my_index/_doc/1

{
    "name":"admin",
    "message":"Hello, Searchgoose!"
}
```

- 데이터 생성

my_index 인덱스에 도큐먼트 id를 임의로 데이터 생성

```
POST my_index/_doc

{
    "name":"admin",
    "message":"Hello, Searchgoose!"
}
```

- 데이터 조회

id가 1인 도큐먼트의 데이터 조회

```
GET my_index/_doc/1
```

- 데이터 삭제

도큐먼트 또는 인덱스 단위의 데이터 삭제

```
DELETE my_index/_doc/1
```

- 데이터 갱신

id가 1인 도큐먼트의 내용 갱신

```
POST my_index/_update/1

{
    "doc": {
        "name": "new_name"
    }
}
```

2. Bulk

여러 명령을 배치로 수행하기 위해 사용된다.

	name	**interfaces**	**descriptions**
5	복수 명령 수행	POST _bulk	위의 다수의 CRUD endpoint들을 배치로 수행

- 복수 명령 수행

위의 다수의 CRUD endpoint들을 배치로 수행

```
POST _bulk

{"index":{"_index":"test", "_id":"1"}}
{"field":"value one"}
{"index":{"_index":"test", "_id":"2"}}
{"field":"value two"}
{"delete":{"_index":"test", "_id":"2"}}
{"create":{"_index":"test", "_id":"3"}}
{"field":"value three"}
{"update":{"_index":"test", "_id":"1"}}
{"doc":{"field":"value two"}}
```

3. Search

인덱스 단위로 검색이 이루어지며, 다양한 옵션을 통해 검색을 세분화 할 수 있다.

검색의 방식으로는 쿼리스트링을 사용하는 방식과 검색 쿼리를 데이터 본문으로 입력하는 방식인 데이터 본문(data body) 검색 방식이 있다.

	name	interfaces	descriptions
6	데이터 검색	GET my_index/_search?q=value	my_index 인덱스에서 “value”라는 값을 검색
7	데이터 검색	GET my_index/_search?q=field:value	“field” 필드에서 “value”라는 값을 검색
8	데이터 검색	GET my_index/_search	“field” 필드에서 “value”라는 값을 검색

- 데이터 검색(쿼리)

my_index 인덱스에서 “value”라는 값을 검색

```
GET test/_search?q=value
```

- 데이터 검색(쿼리)

“field” 필드에서 “value”라는 값을 검색

```
GET my_index/_search?q=field:value
```

- 데이터 검색(본문검색)

“field” 필드에서 “value”라는 값을 검색

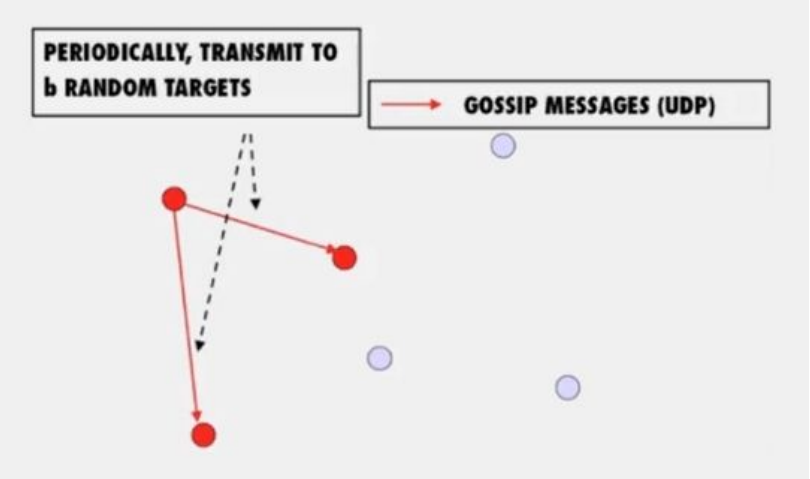
```
GET test/_search

{
    "query": {
        "match": {
            "field": "value"
        }
    }
}
```

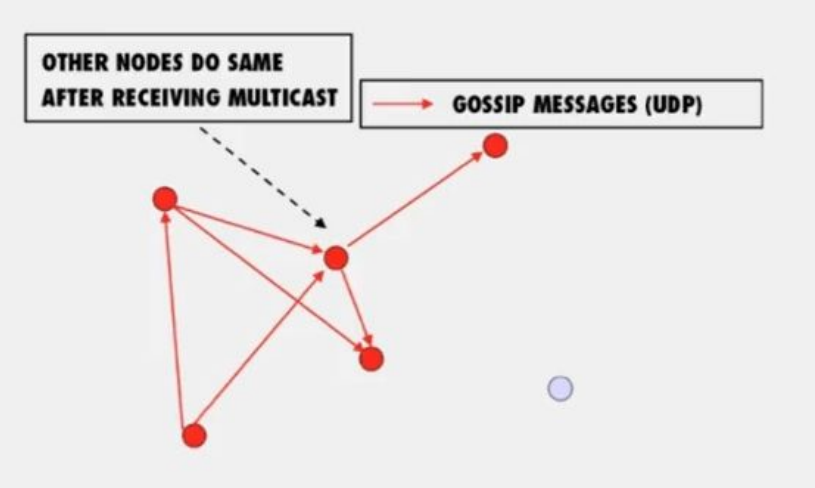
3.3 Non-functional requirements

Searchgoose는 한 클러스터 안의 여러 노드가 협업하여 대량의 연산을 처리하는 분산시스템이다. 이를 위해 복수의 노드 사이의 통신이 필요하다. Service discovery를 위한 여러 방법 중 아래에서 소개하는 gossip 방법을 통해 해당 클러스터에 참여하는 노드와 연결을 맺고 주기적으로 자신이 갖고 있는 정보와 상대 노드가 갖고 있는 정보를 교환한다. 클러스터의 모든 노드를 탐색했다면 후술할 quorum-based raft 방법을 통해 마스터 노드를

선출한다. 이후 마스터 노드와 데이터 노드가 각자의 역할을 담당하게 만든다.

Gossip Protocol





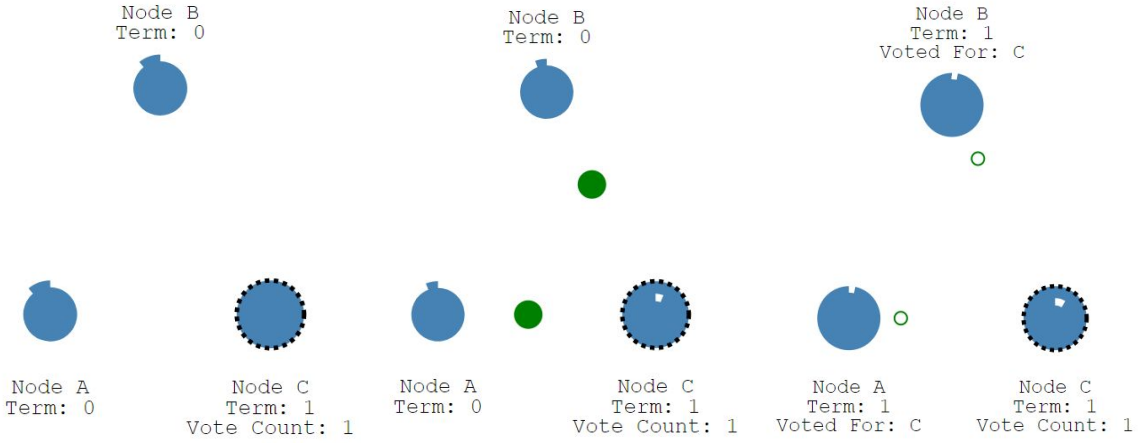
가십 프로토콜은 분산된 노드 네트워크에서 동작하며, centralized coordinator가 없는 대신 각 노드가 주기적으로 정보를 주고 받는다. 이 때 주기적으로 랜덤한 노드들을 선출해서 gossip message를 전송하고, 해당 메세지를 받은(infected) 다른 노드들도 역시 해당 메세지를 다른 노드들에게 전달하게 된다. 모든 메세지 전달이 완료된 후에 네트워크에 참여하고 있는 노드들은 gossip message를 통해 변경된 혹은 추가된 사항을 전달받고 동일한 상태의 정보를 갖게 되며, 그 정보를 바탕으로 새로운 노드와 기존 노드들끼리의 연결이 결정된다.

Raft

분산 시스템에 참여하고 있는 전체 노드들 사이의 연결이 결정되었다면, 이 시스템에서 중심적인 일을 수행하는 노드를 선출(election)해야 하는데 이 때 사용하는 알고리즘을

Raft라고 한다.

Raft의 node는 Follower, Candidate, Leader라는 3가지 state를 가진다. Leader는 Leader로 선출된 상태를 의미하며, Follower는 Leader가 선출된 이후 Leader로부터 변경 내역이 담긴 메세지를 받아 처리하는 상태이다. 이러한 heartbeat 메세지를 받게 된 node들은 자신의 election timeout 을 리셋하게 된다. 그러나, Leader로 부터 heartbeat 메세지를 받지 못하여 설정된 election timeout을 초과하게 된 node들은 자신의 state를 Candidate로 바꾸고 스스로에게 한 표를 준 뒤 네트워크에 있는 노드들에게 vote message를 전송한다. 그리고 해당 vote message를 받은 노드들은 자신의 election timeout을 초기화하고 자신에게 먼저 도달한 vote message를 보낸 node에게 투표하게 된다.



그렇게 모든 투표가 끝나고 과반 이상의 득표를 한 node는 Leader state가 되고, 자신이 당선되었다는 heartbeat message를 다른 노드들에게 보낸다. 이렇게 결정된 Leader node는 마스터 노드로서 모든 의사 결정권을 갖게 되어 client와 통신하게 된다.

Quorum & Split-brain

네트워크 이상 및 단절이 발생하여 cluster를 구성하는 node간 통신이 두절되고 cluster가 sub-cluster로 분리되는 현상을 **split-brain**이라고 한다. 만약, 분리된 두 sub-cluster에서 각각의 Leader node를 선출하게 되면 하나의 cluster 내에 두 개의 Leader node가 존재하게 되어 각각 따로 운영 되기 때문에 데이터 비동기 문제가 발생한다.

Quorum이란 정족수로, 합의체가 의사를 진행시키거나 의결을 하는 데 필요한 최소한도의 인원수를 뜻한다. 즉, 위의 문제 상황에서 하나의 노드가 Leader로 선출되기 위한 최소한의 득표수를 정의하고, 해당 수의 Quorum을 충족하지 못할 경우 Leader로 선출되지 못하도록 한다. Quorum based election algorithm에서는 Quorum을 cluster를 구성하는 노드 수의 절반으로 정의한다. 따라서 sub-cluster를 구성하는 노드들의 수가 전체 cluster의 절반보다 적게 되면 해당 sub-cluster에서는 Leader를 선출할 수 없게 되는 알고리즘이다.

한편, 전체 cluster의 node가 짝수개이면 같은 수의 sub-cluster로 split-brain이 발생했을 때 Quorum은 그 의미를 수행할 수 없게 된다. 따라서 cluster의 node의 수가 홀수개로

유지되어야 한다.