searchgoose

Team 9
201411296 이선명
201411312 장하나
201711375 권혁규
201711413 이유진

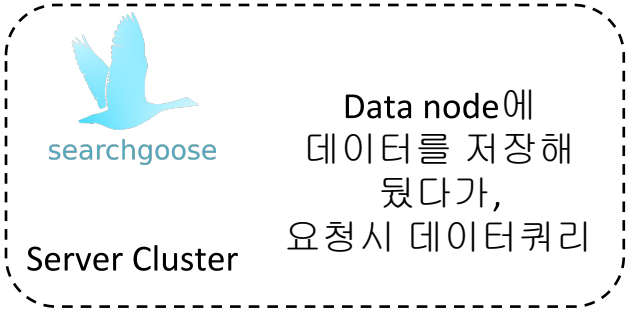
1차구현 데모

구현 - Overall

Data API
- GET /{index}/_doc/{id}
- POST /{index}/_doc
- DEL /{index}/_doc/{id}
- GET /{index}/_search

User

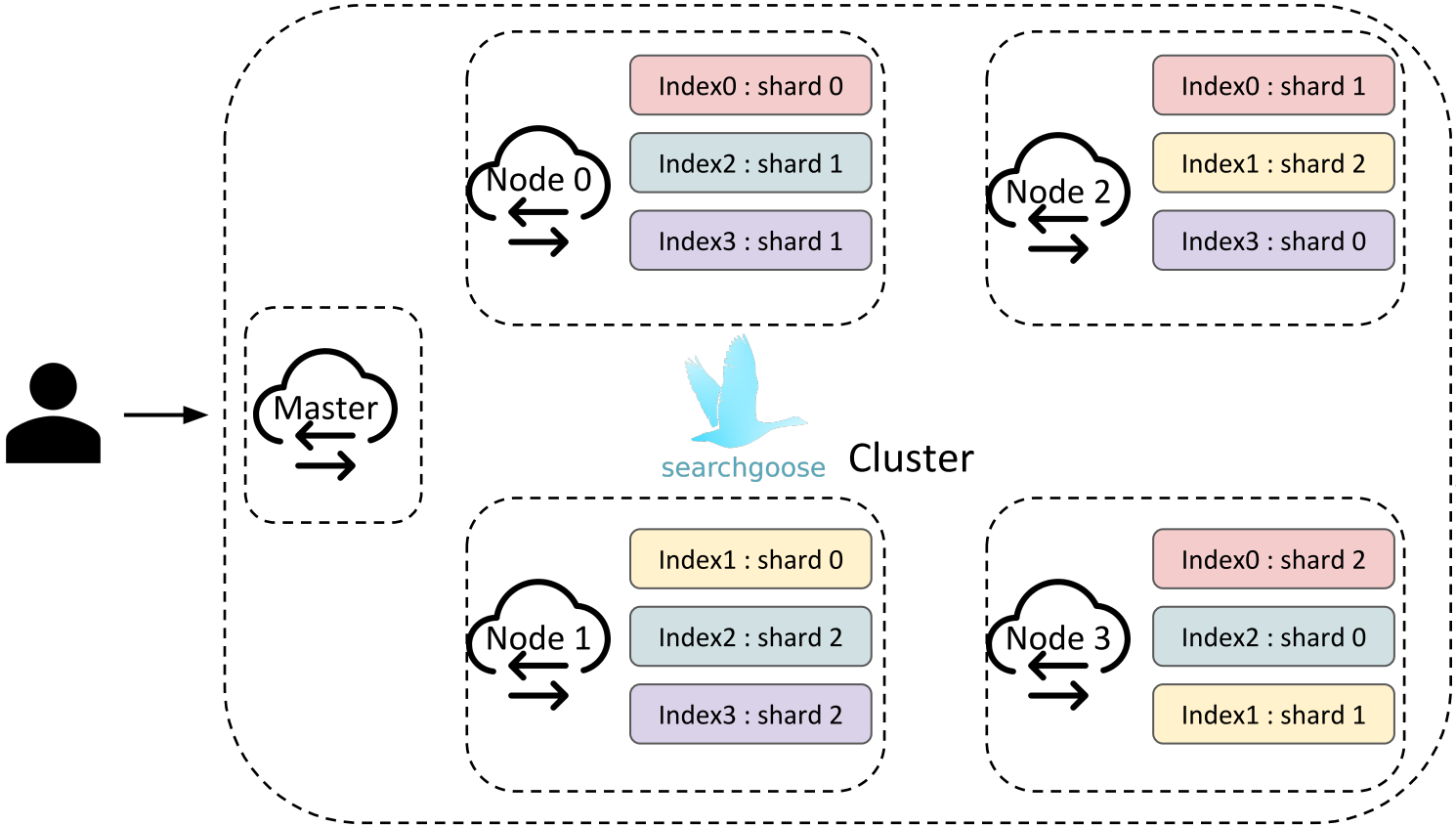
- 인덱스 생성
- 인덱스 조회
- 데이터 삽입
- 데이터 조회
- 데이터 삭제
- 검색



User 입장에서는 하나의 거대한 시스템에 데이터를 저장하고 쿼리한다.

실제 구현은 복수의 노드가 협업하여 하나의 거대한 시스템처럼 동작하게 한다.

구현 - Cluster

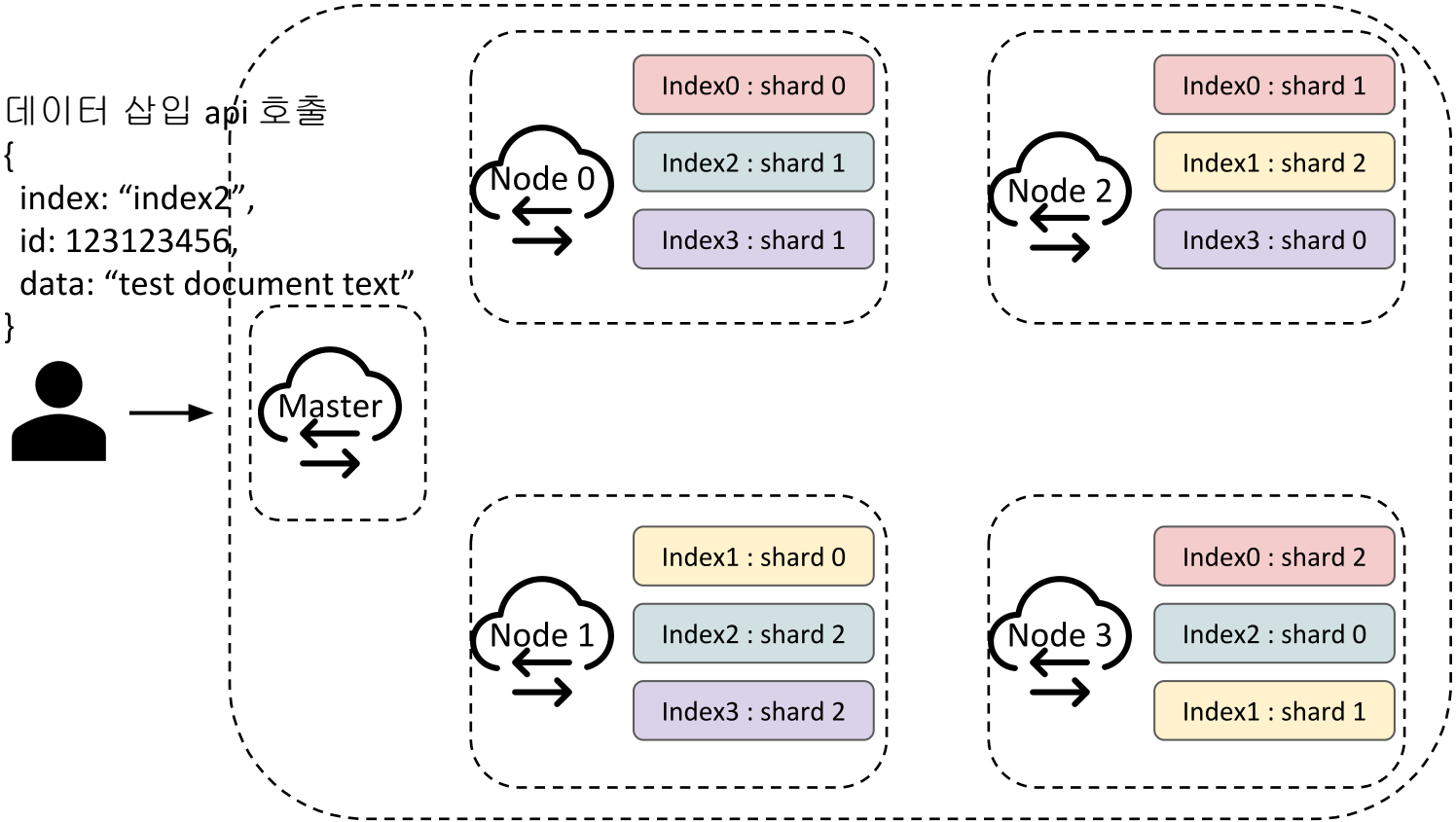


인덱스라는 분류단위
마다
샤드라는 개념이 존재.
샤드를 각 데이터노드에
분산하여 저장.
요청시 데이터쿼리.

구현 - Cluster example

데이터 삽입 api 호출

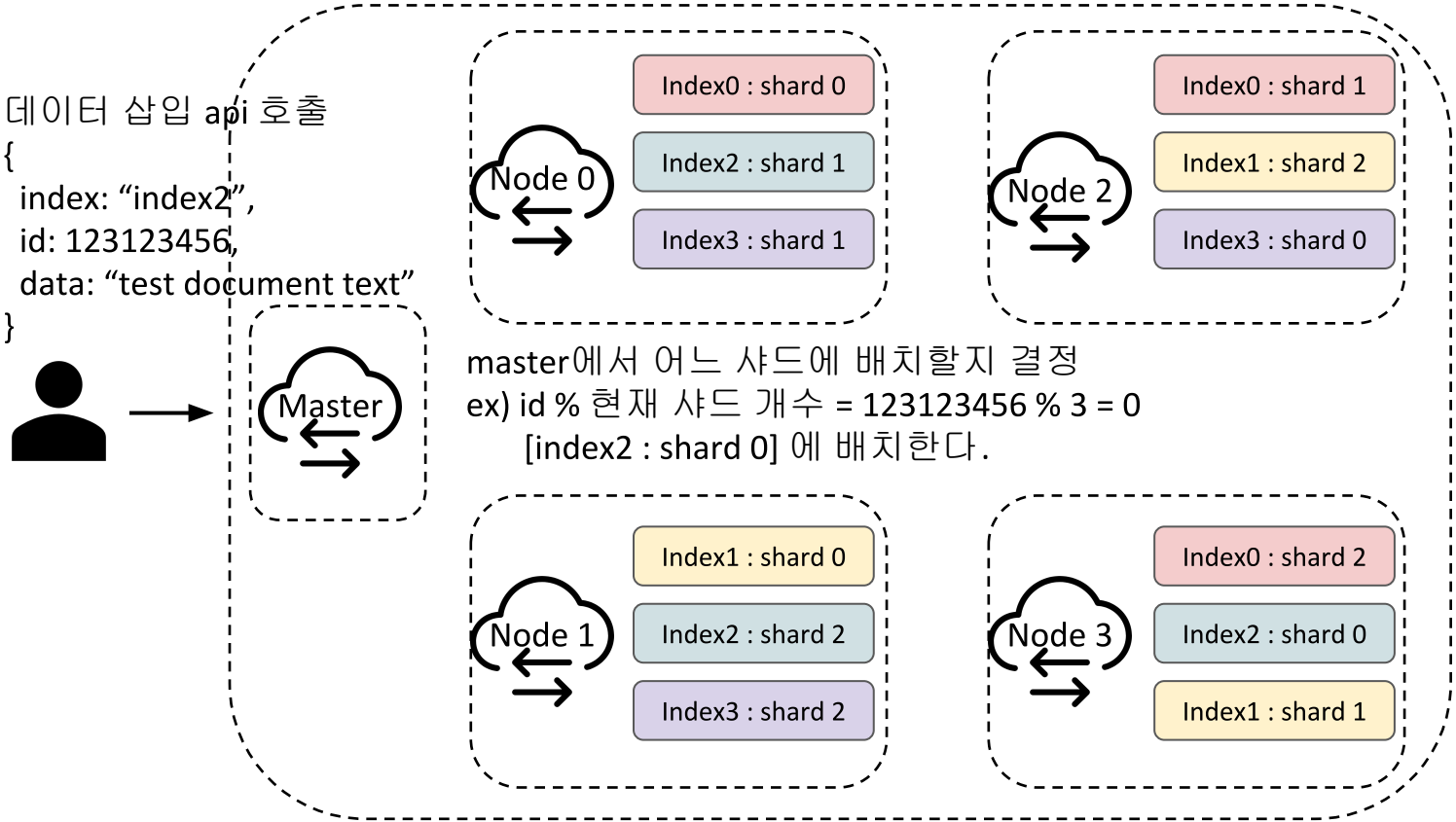
```
{
  index: “index2”,
  id: 123123456,
  data: “test document text”
}
```



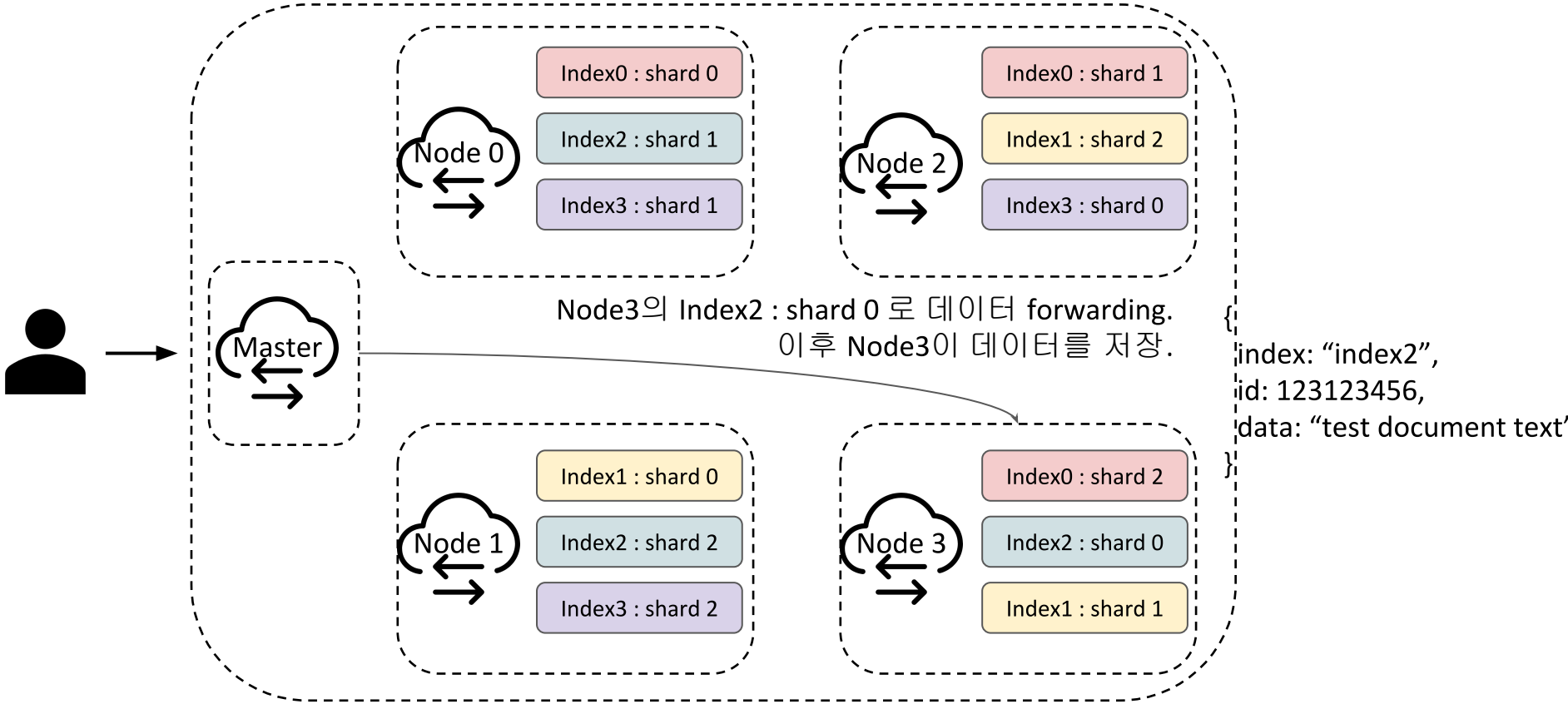
구현 - Cluster example

데이터 삽입 api 호출

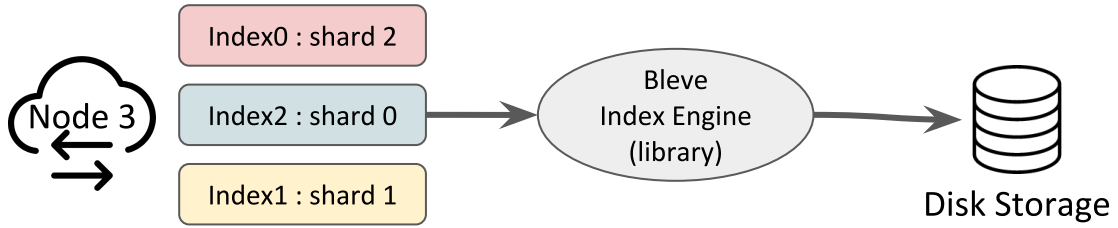
```
{
  index: “index2”,
  id: 123123456,
  data: “test document text”
}
```



구현 - Cluster example



구현 - Data node



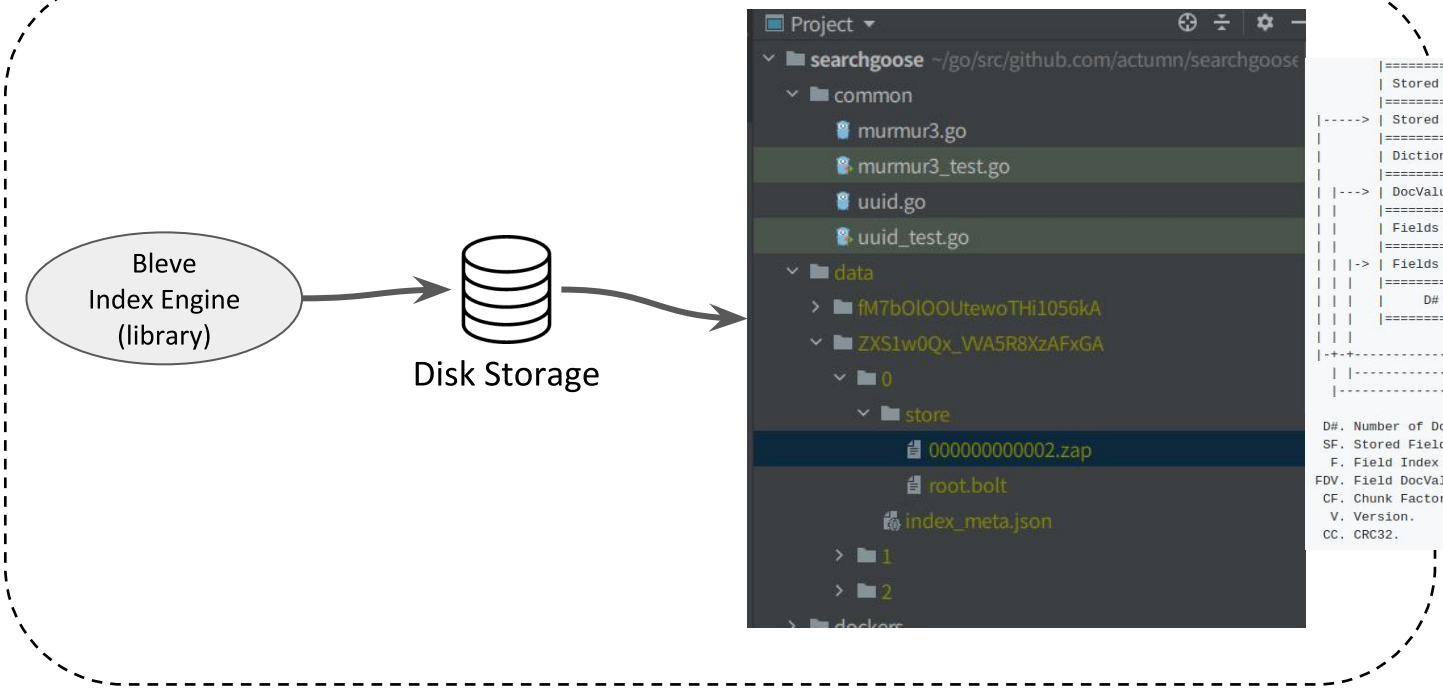
Bleve Index Engine
- 데이터 삽입
- 데이터 조회
- 데이터 삭제
- 검색

bleve 라이브러리의 도움을 받아서 disk storage에 저장 및 인덱싱.
이후 데이터 쿼리

Bleve?

Text indexing libarary.
디스크에 데이터를 정해진 포맷의 파일로 저장한다.
Full text search를 지원한다.

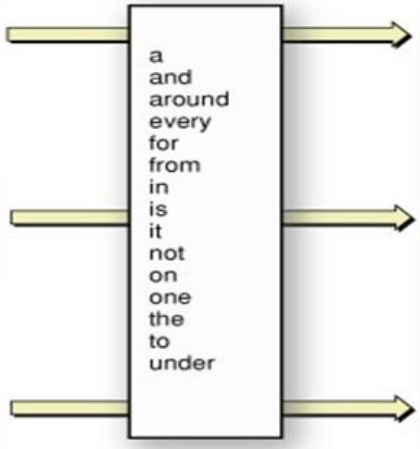
./data/{index-uuid}/{shard-number}/...
데이터를 인덱싱해서 저장한다.



Bleve - Inverted index for search

각 document를 단어마다 분리
단어마다 document 인덱싱을 통해
검색 단어에 가장 어울리는 document를 빠르게 찾아낼 수 있음.

Disk Storage



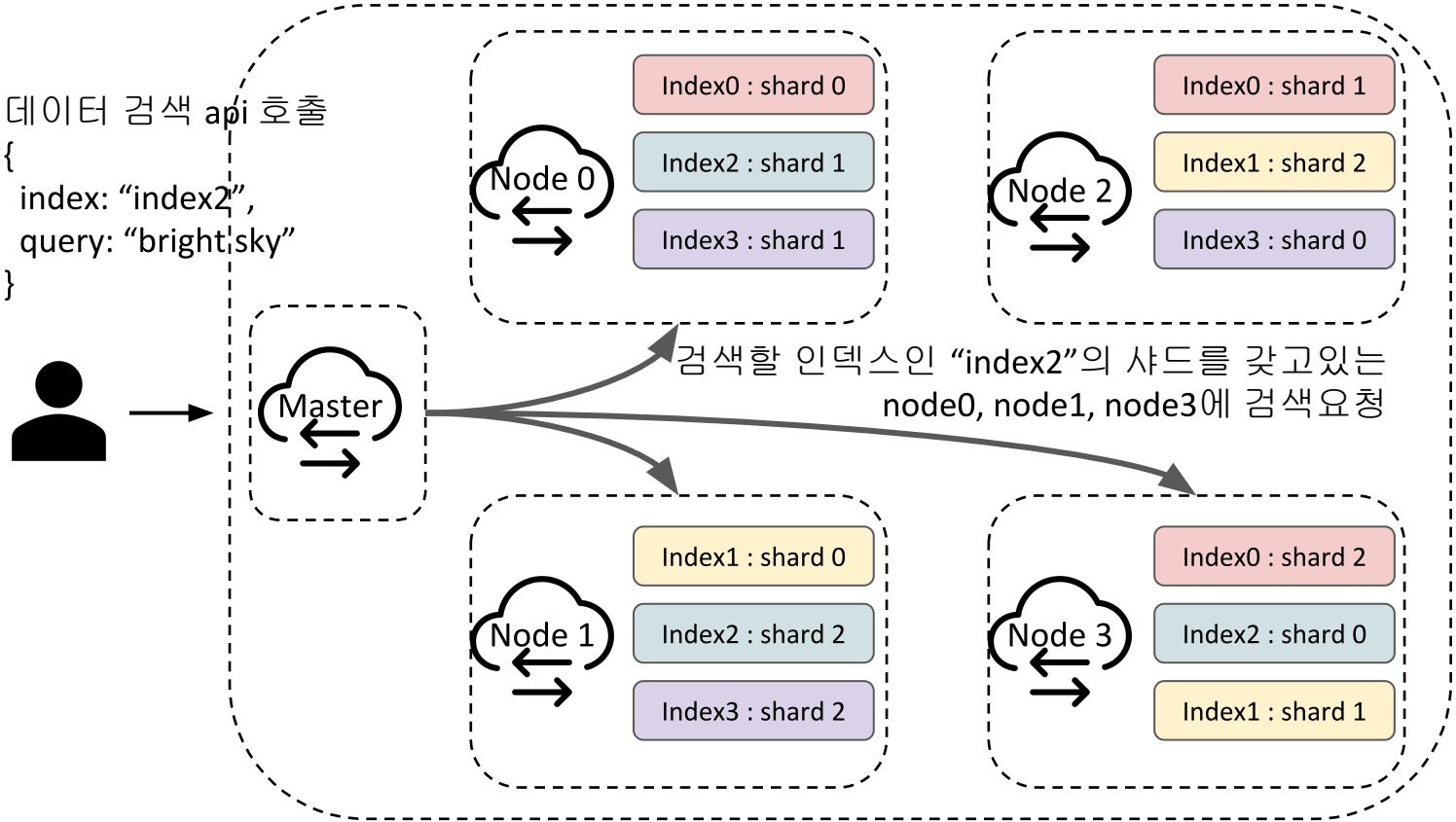
Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

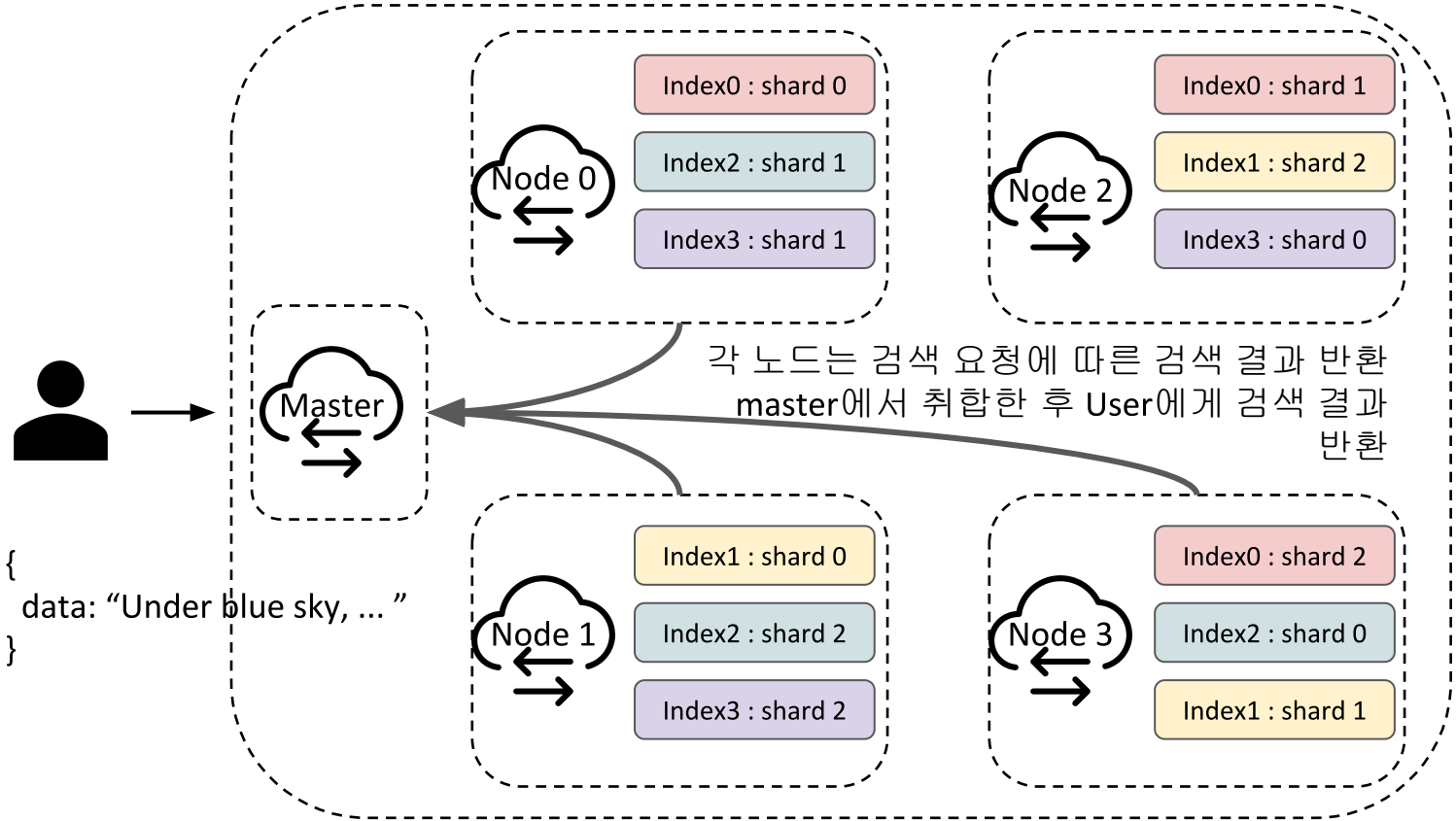
ex) “bright sky” 검색시
3. bright -> 1, **3**
12. sky -> 2, **3**
계산 결과 3번 document가 점수가 가장 높게 나올댜

구현 - Search example

데이터 검색 api 호출
{
index: “index2”,
query: “bright sky”
}



구현 - Search example




```
{
  data: “Under blue sky, … ”
}
```

API - 인덱스 생성

PUT /test-index
{,
 settings: {
  “shard 개수”: 3
 }, ...
}

Request

Method: PUT
Request URL: http://localhost:8080/test15

Parameters

Headers | Body

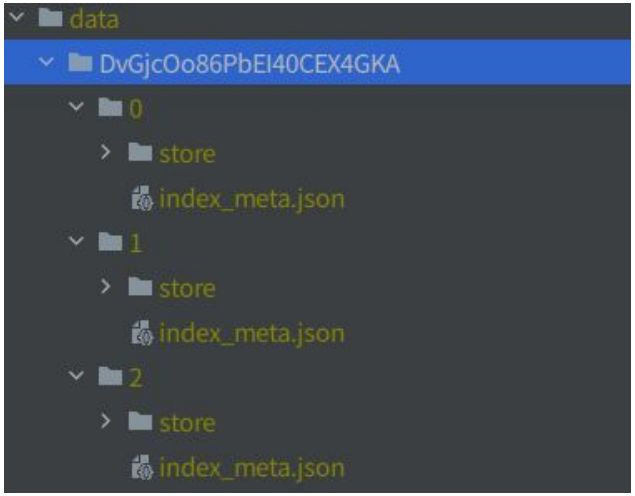
Body content type: application/json
Editor view: Raw input

FORMAT JSON MINIFY JSON

```
{
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 2
  },
  "mappings": {
    "properties": {
      "field1": {
        "type": "text"
      }
    }
  }
}
```

200 OK 22.29 ms

```
{
    "acknowledged": true,
    "index": "test15",
    "shards_acknowledged": true
}
```



disk에 해당 인덱스 metadata 저장

LOG

```
2020/09/08 18:52:38 PUT /test15
2020/09/08 18:52:38 Create index - index name: test15, mapping: {"properties":{"field1":{"type":"text"}}}
2020/09/08 18:52:38 publish new state to node[EtcqUjPQqHMrUnuMs_WXog]
2020/09/08 18:52:38 accept new state
2020/09/08 18:52:38 Create new index shard - index uuid: DvGjcOo86PbEI40CEX4GKA, shard number: 0
2020/09/08 18:52:38 Create new index shard - index uuid: DvGjcOo86PbEI40CEX4GKA, shard number: 1
2020/09/08 18:52:38 Create new index shard - index uuid: DvGjcOo86PbEI40CEX4GKA, shard number: 2
2020/09/08 18:52:38 publish ended successfully
```

API - 데이터 삽입

```
PUT /test-index/_doc/2
{
  “field1”: “test”,
  “field2”: “test2
}
```

Method: PUT
Request URL: http://localhost:8080/test15/_doc/2

Parameters

Headers | Body

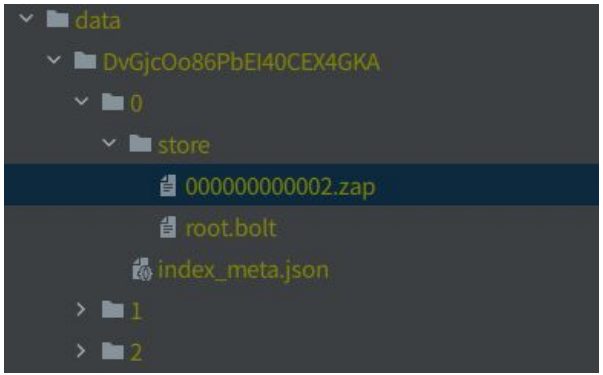
Body content type: application/json
Editor view: Raw input

FORMAT JSON MINIFY JSON

```
{
  "field1": "test",
  "field2": "test2"
}
```

200 OK 18.76 ms

```
{
    "_id": "2",
    "_index": "test15",
    "_primary_term": 1,
    "_seq_no": 0,
  - "_shards": {
        "failed": 0,
        "successful": 1,
```

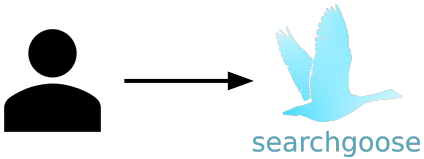


disk에 해당 데이터 저장

```
2020/09/08 19:08:06 PUT /test15/_doc/2
```

API - 데이터 조회

GET /test-index/_doc/2




```
{
  “field1”: “test”,
  “field2”: “test2
}
```

Request

Method: GET

Request URL: http://localhost:8080/test15/_doc/2

Parameters

Headers

Toggle source mode Insert headers set

ADD HEADER

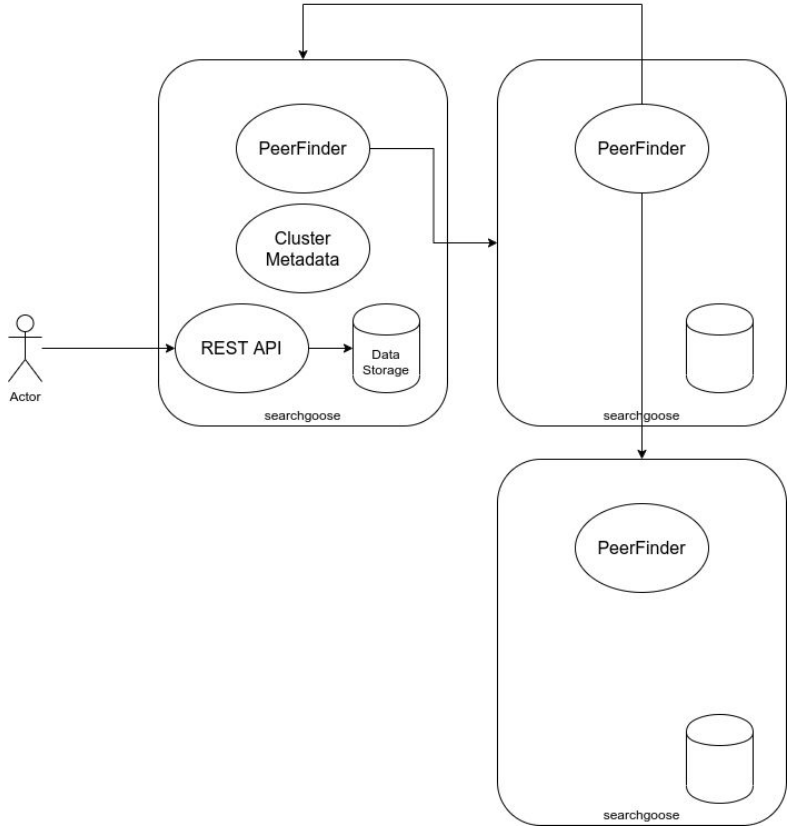
Headers are valid

200 OK 13.11 ms


```
{
    "_id": "2",
    "_index": "test15",
    "_primary_term": 1,
    "_seq_no": 0,
  - "_source": {
        "field1": "test",
        "field2": "test2"
    },
    "_type": "_doc",
    "_version": 1,
    "found": true
}
```


disk로부터 해당 데이터 추출

구현 상세 - Basic

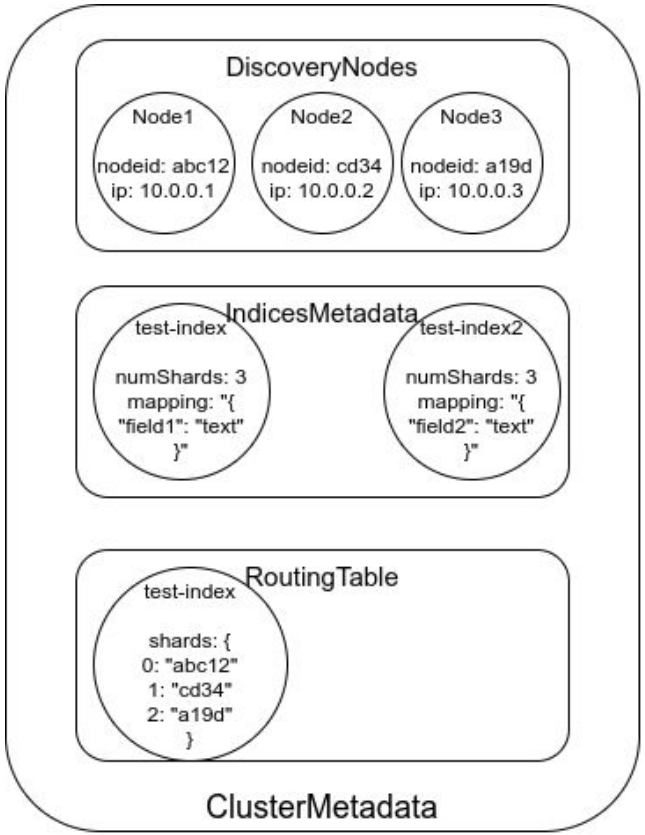


- REST API:
  Actor는 REST API 호출을 통해 disk에 데이터 저장 / 저장된 데이터 조회 / 검색 질의

- PeerFinder:
  각 서버 노드는 백그라운드에서 클러스터링 구축을 위해

- ClusterMetadata:
  1. 현재 참여하고 있는 노드 정보,
  2. 현재 클러스터에 있는 인덱스 정보,
  3. 각 인덱스 샤드별 관리하는 노드 정보
  등 클러스터 노드 모두가 합의하고 있는 단일 메타데이터.

구현 상세 - ClusterMetadata



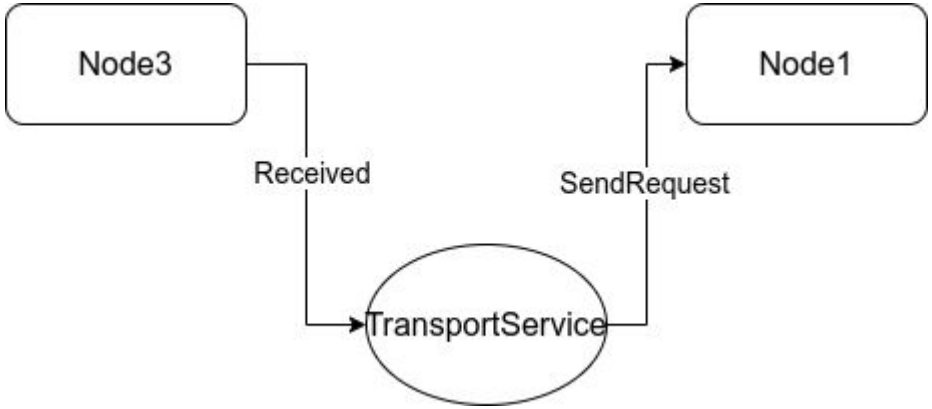
클러스터 노드 모두가 합의하고 있는 단일 메타데이터.

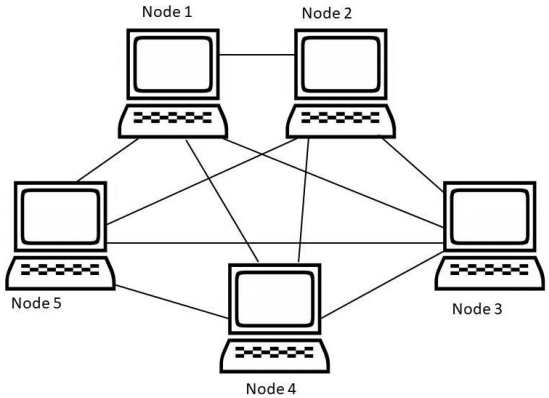
- DiscoveryNodes
  현재 클러스터에 참여하고 있는 노드 및 ip를 나타낸다.
  이를 통해 실제 노드와 connection을 맺고 데이터 전송/수신이 가능

- IndicesMetadata
  인덱스 별 샤드 개수 등의 setting, mapping 정보 등을 저장

- RoutingTable
  인덱스 샤드 별 관장하고 있는 노드 id 저장
  ex) test-index의 1번 샤드를 관리하는 노드의 id는 "cd34"
  이를 통해 cd34 -> ip 10.0.0.2 에 해당하는 노드에 데이터 저장/질의를 하게 된다.

구현 상세 - Transport



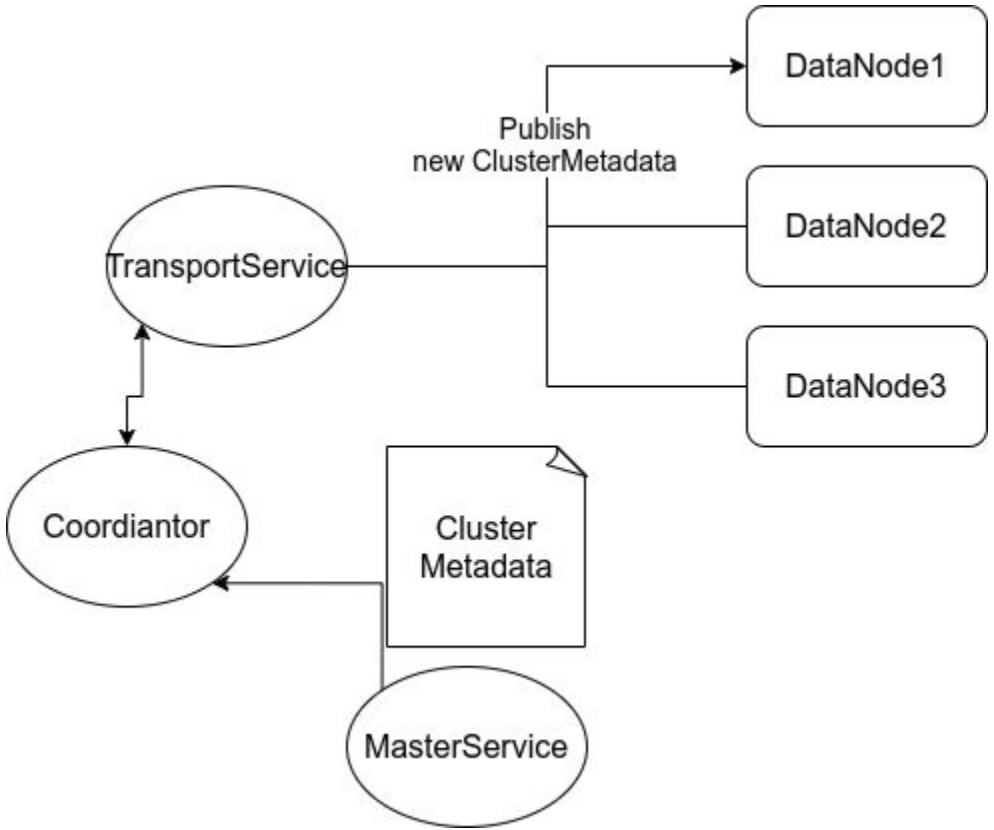


remote 노드와 통신하는 것을 담당한다.

TCP로 구현한다.
노드 별 connection 유지

searchgoose가 구성하게 되는
클러스터 아키텍쳐는
Fully Connected Mesh Topology가 된다.

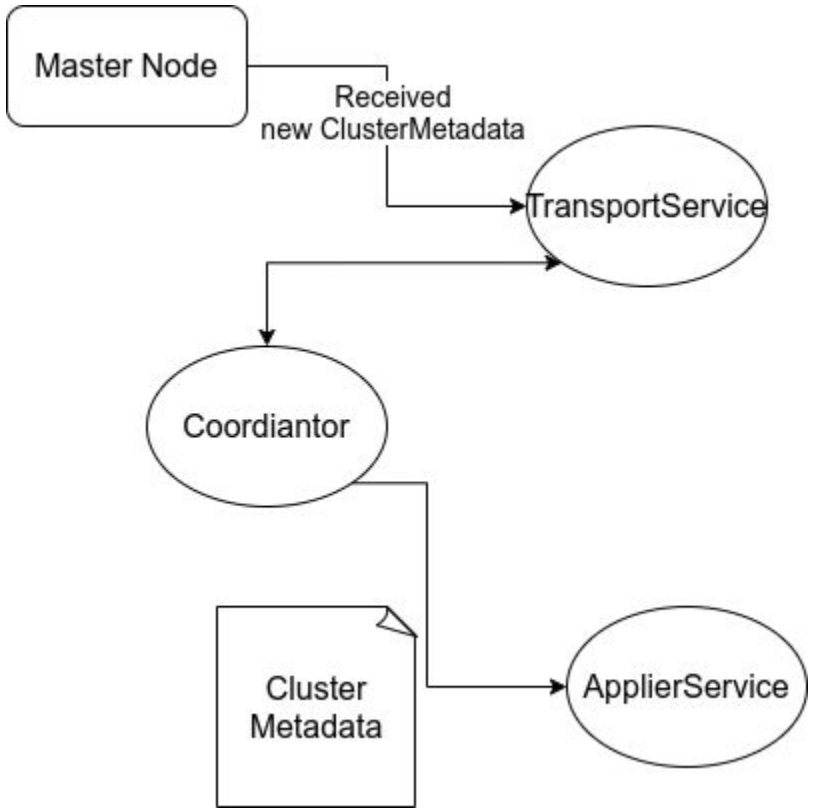
구현 상세 - ClusterMetadata publish



ClusterMetadata는 Master node에 의해 빈번히 변경될 수 있다.
(새로운 data node 참여, 새로운 index 생성 등)

- MasterService 는 ClusterMetadata가 변경되는 일이 있다면 이를 클러스터에 참여하고 있는 모든 노드에게 전파시키는 역할을 담당한다.
- Coordinator는 분산시스템에서 일어나는 일들을 담당한다. (Master node election, 클러스터 모든 노드에 ClusterMetadata 송신하는 함수 제공, ClusterMetadata를 수신하는 함수 제공 등)
- TransportService는 개별 노드별 ClusterMetadata 송신을 책임진다.

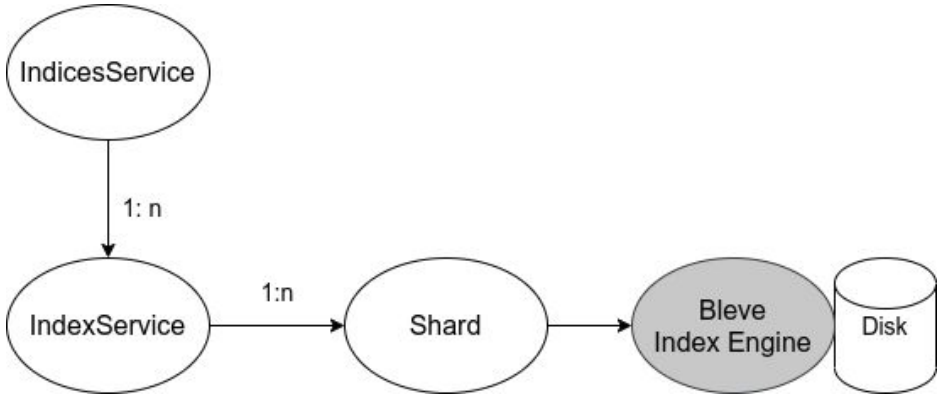
구현 상세 - ClusterMetadata apply



ClusterMetadata는 Master node에 의해 빈번히 변경될 수 있고, Data node는 Master로 부터 전달받은 새로운 ClusterMetadata를 자신에게도 적용시켜야 한다.

- TransportService는 마스터가 보내는 ClusterMetadata를 수신을 책임진다.
- Coordinator는 ClusterMetadata 를 수신받았을 때 처리하는 함수를 제공한다.
- ApplierService는 새로운 ClusterMetadata를 수신받았다면 자신에게 적용시킨다.
  (새로운 노드와 connection을 맺음 또는 새로운 index 별 shard 생성 및 디스크에 저장)

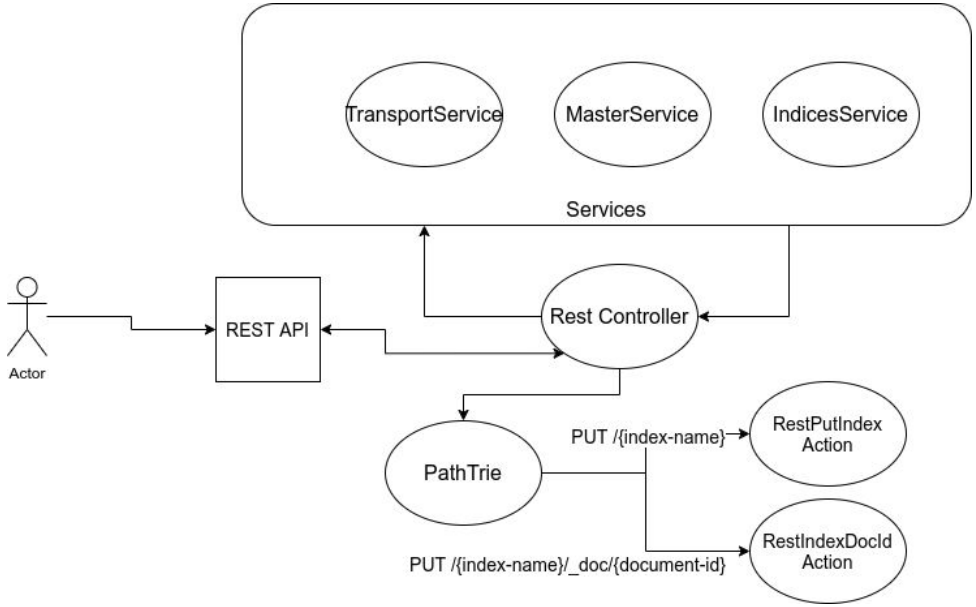
구현 상세 - Index, document



- IndiceService는 여러 IndexService를 관리한다.
- IndexService는 하나의 인덱스를 관리하기 위해 여러 shard를 갖는다.
- 각 shard는 bleve index engine을 통해 디스크 스토리지에서 정해진 path에 데이터를 저장한다.

.

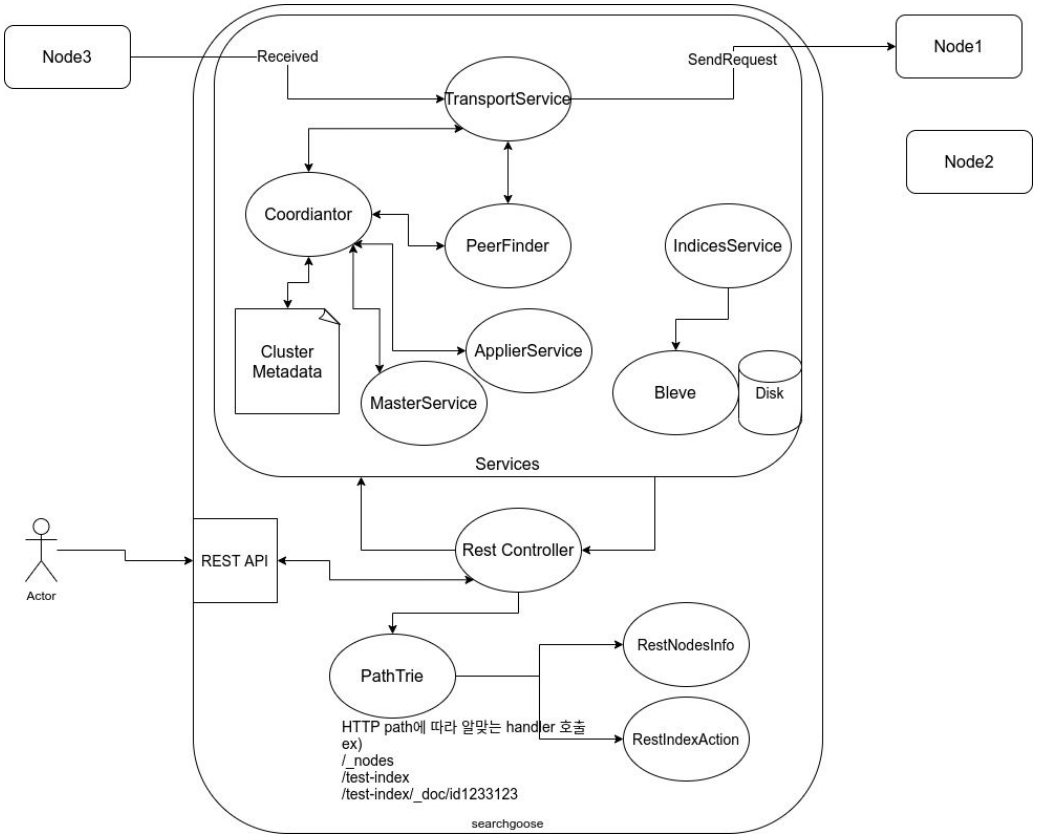
구현 상세 - REST API



User에게 노출되기 위한 HTTP API 담당 객체들

- RestController는 http request 및 http response를 책임지는 함수를 제공한다.
- 각 path마다 해당하는 handler가 존재한다.
  ex) PUT /{index-name} -> PutIndex handler
- path마다 알맞은 handler를 찾아내기 위해 trie 자료구조 사용
- 각 handler는 이전에 명세했던 IndicesService, TransportService, MasterService 등을 적절히 사용한다.

구현 상세 - Overall



Test cases

구현 완료

일부 구현 / 진행중

미구현

No	Name	Description
1.1	Cluster node healtch check api	Cluster 노드 상태 확인
1.2	Clustster stats api	Cluster 노드 인덱스 상태 확인
2.1	Index Create api	인덱스 설정 생성 확인
2.2	Index Read api	인덱스 설정 조회 확인
2.4	Index Delete api	인덱스 설정 삭제 확인
3.1	Doucmnet Create api	문서 생성 확인
3.2	Document Read api	문서 조회 확인
3.4	Document Delete api	문서 삭제 확인
4.1	Search api	문서 검색 확인

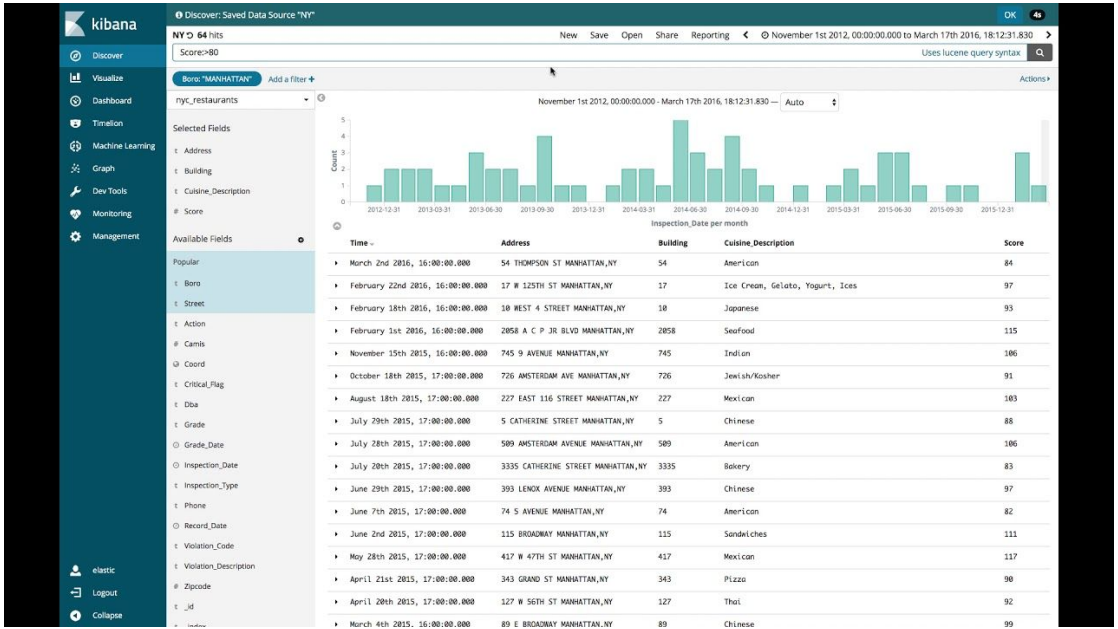
Test cases

No	Name	Description
5.1	Master node - index create	인덱스 생성 시 샤드 생성, 알맞는 Data node에 샤드 분산
5.2	Master node - index read	인덱스 정보 조회 확인
5.3	Master node - index delete	인덱스 삭제, 각 Data node에 인덱스 삭제 request
5.4	Master node - Document create	알맞은 data node에 문서 생성 request를 forward, 이후 reply
5.5	Master node - Document read	알맞은 data node에 문서 조회 request를 forward, 이후 reply
5.6	Master node - Document delete	알맞은 data node에 문서 삭제 request를 forward, 이후 reply
5.7	Master node - Search	각 Data node에 search request를 forward, 이후 모아서 집계
6.1	Data node - Document create	해당 Data node 에서 문서 생성
6.2	Data node - Document read	해당 Data node 에서 해당 문서 조회
6.3	Data node - Document delete	해당 Data node 에서 해당 문서 삭제
6.4	Data node - Search	해당 Data node에서 갖고있는 데이터 조회
7.1	Self-service-discovery	클러스터 내 노드간 서로 탐색하는 것을 확인
7.2	Self-consensus	Master node election 확인

Plan: 2nd Iterations

No	Name	Description
4.1	Search api	문서 검색 확인
5.3	Master node - index delete	인덱스 삭제, 각 Data node에 인덱스 삭제 request
5.4	Master node - Document create	알맞은 data node에 문서 생성 request를 forward, 이후 reply
5.5	Master node - Document read	알맞은 data node에 문서 조회 request를 forward, 이후 reply
5.6	Master node - Document delete	알맞은 data node에 문서 삭제 request를 forward, 이후 reply
5.7	Master node - Search	각 Data node에 search request를 forward, 이후 모아서 집계
6.1	Data node - Document create	해당 Data node 에서 문서 생성
6.2	Data node - Document read	해당 Data node 에서 해당 문서 조회
6.3	Data node - Document delete	해당 Data node 에서 해당 문서 삭제
6.4	Data node - Search	해당 Data node에서 갖고있는 데이터 조회
7.1	Self-service-discovery	클러스터 내 노드간 서로 탐색하는 것을 확인
7.2	Self-consensus	Master node election 확인

Plan: Final - Webview



Webview (Kibana)

kibana

→

API

Data API

-

GET /{index}/_doc/{id}
- GET /{index}/_search
- PUT /{index}/_doc
- DEL /{index}/_doc/{id}

←

searchgoose

kibana: 웹뷰 오픈소스 소프트웨어
kibana가 사용하는 프로토콜을 이용하여 구현해서
kibana가 제공하는 웹뷰 기능을 활용할 수 있게 한다.