실무에 활용하는 Elasticsearch 검색엔진 구축

5일차 : ElasticSearch 1

오늘의 아젠다

엘라스틱서치의 개요

엘라스틱서치 설치 실습

엘라스틱서치 데이터 CRUD 실습

엘라스틱서치 클러스터 관리

엘라스틱서치 매핑

1. 엘라스틱서치의 개요

엘라스틱서치의 개요

Shay Banon이 Lucene을 바탕으로 개발한 분산 검색엔진
2010년 2월 첫 버전이 공개
Apache2 Licence
설치와 서버 확장이 매우 편리
아파치 루씬 기반
높은 가용성 (HIGH AVAILABILITY)
멀티 테넌시 (MULTY TENANCY)
JSON DOCUMENT / RESTFUL API
실시간 검색 (Near Real Time)

엘라스틱서치 5.0 특징

루씬 6.2 기반

색인 성능

블록 KD 트리 데이터 구조 적용

lock 경합 완화

트랜스로그 fsync lock 조건 완화

“append-only + ID 자동 생성” 인덱스 구조의 경우

GET 요청 처리 방식 변경

색인 성능이 25% ~ 80% 정도 향상(디스크 사용량 , 메모리 사용량 감소)

기타

Ingest Node 추가

보안문제 강화

색인 성능

부트스트랩 검사(bootstrap check)

상용모드와 개발모드 (transport 프로토콜이 바인딩한 네트워크 인퍼페이스가 루프백(loop-back) 주소인지 아닌지에 따라 결정)

검색과 집계관련

time range query 리팩토링을 통한 집계 성능을 항상

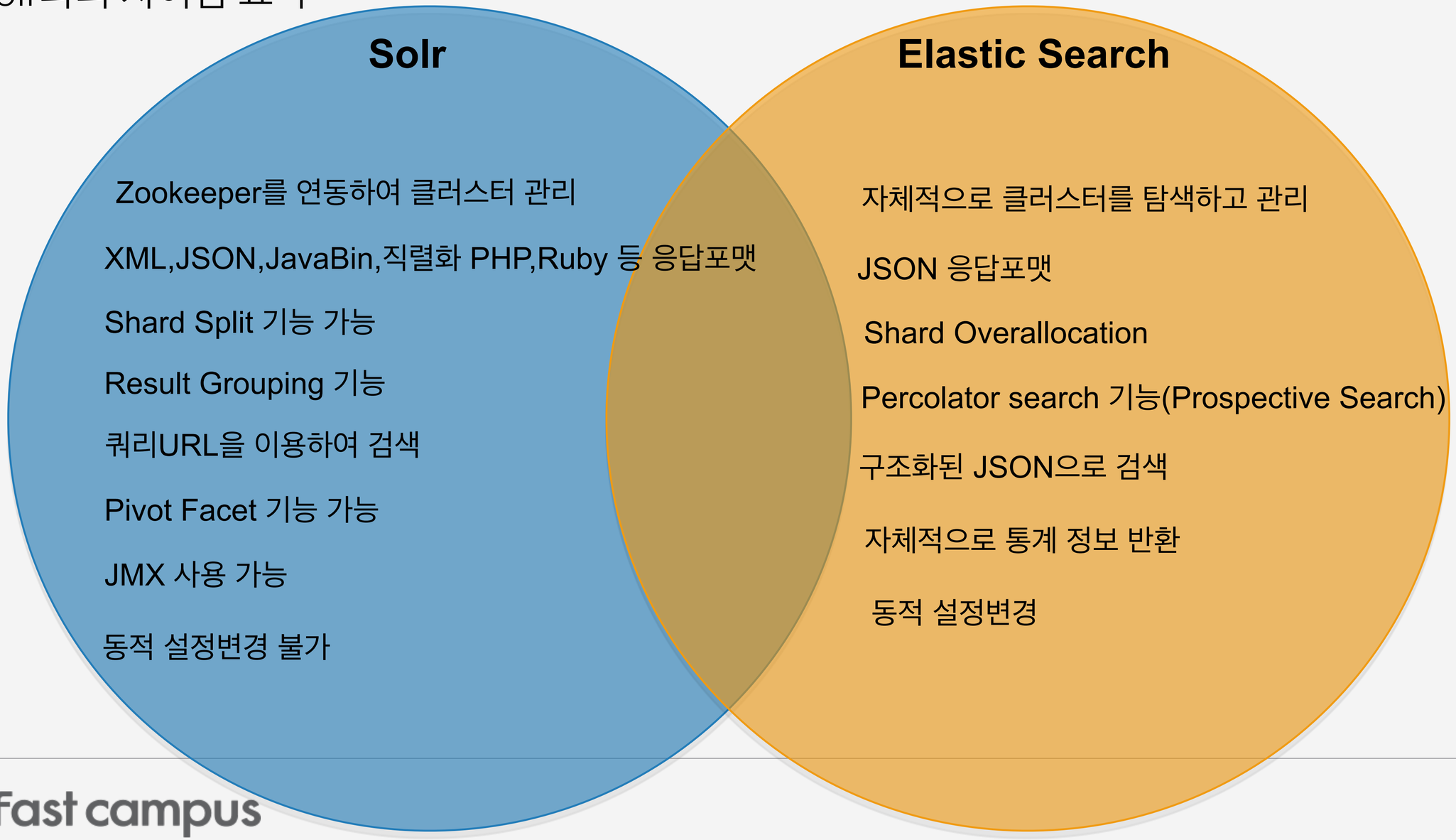
search_after 구문 추가로 페이징 속도 향상

BM25 스코어링 적용

릴리즈 노트 참조

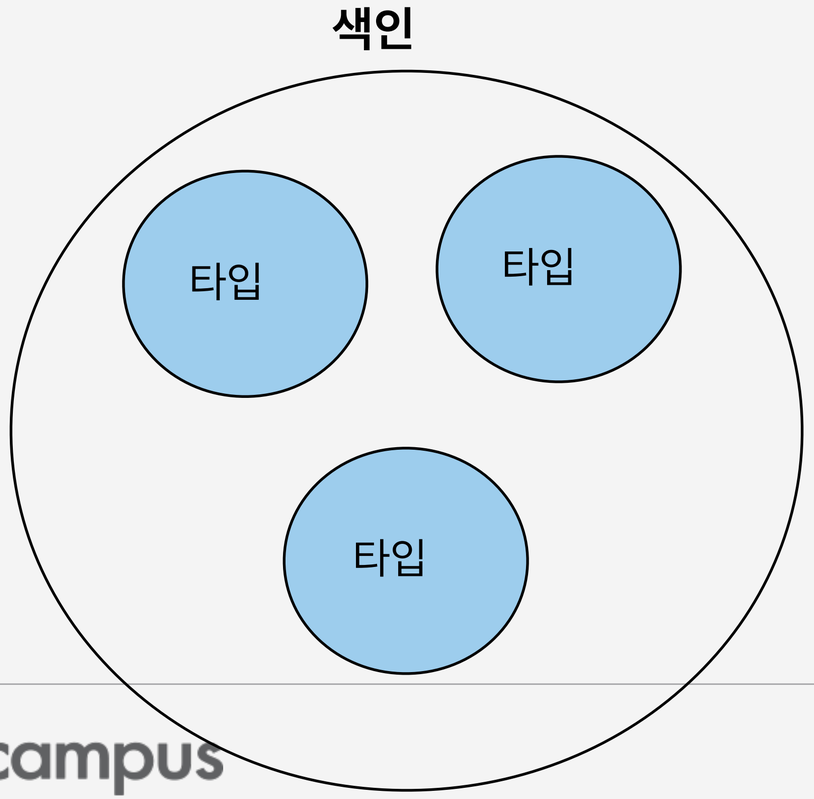
https://www.elastic.co/kr/blog/elasticsearch-5-0-0-released

Solr와의 차이점 요약



용어 정리 – 색인(index)이란?

검색엔진의 논리적 저장 단위

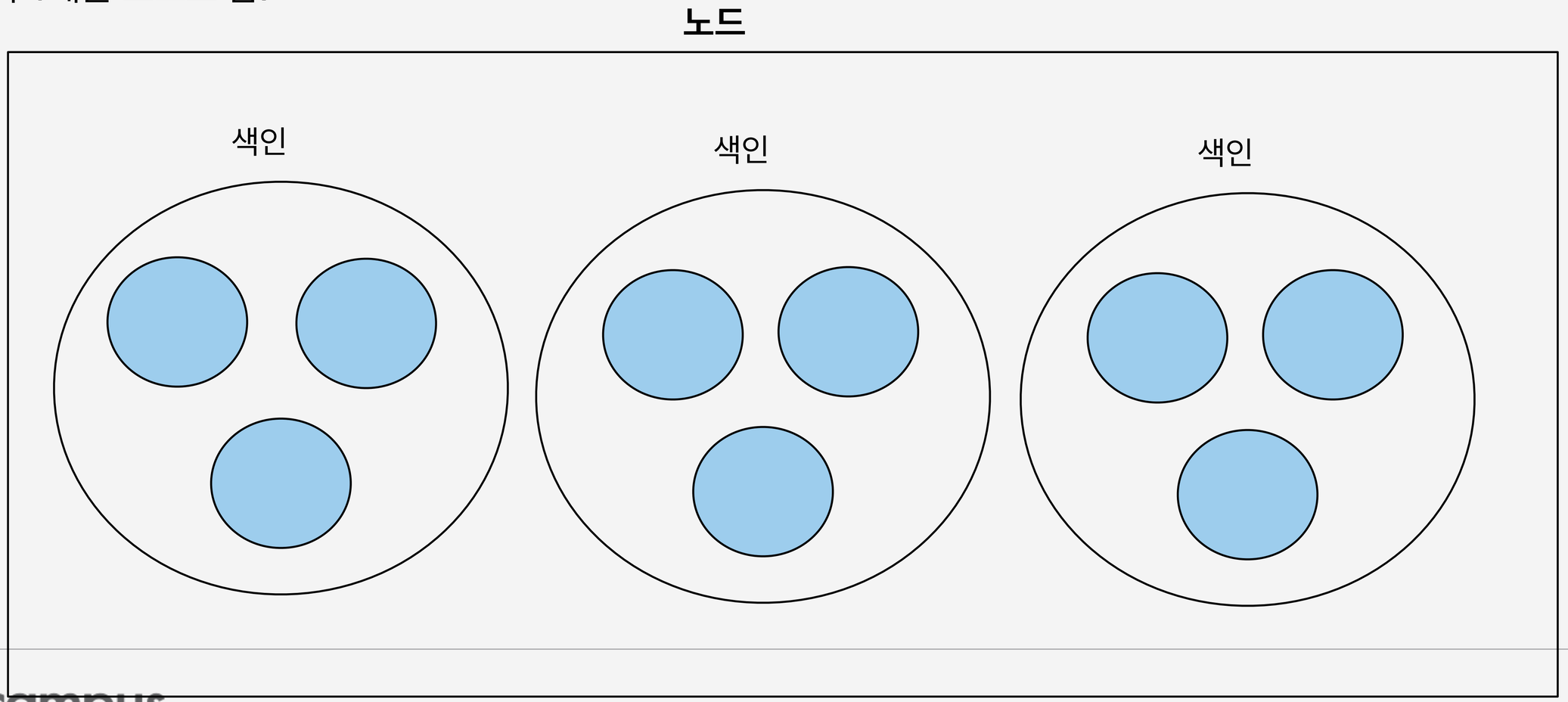



```
   367 2016-03-21 09:58 _0.cfe
391641 2016-03-21 09:58 _0.cfs
   256 2016-03-21 09:58 _0.si
   367 2016-03-21 09:58 _1.cfe
497025 2016-03-21 09:58 _1.cfs
   256 2016-03-21 09:58 _1.si
   367 2016-03-21 09:58 _2.cfe
385642 2016-03-21 09:58 _2.cfs
   256 2016-03-21 09:58 _2.si
   367 2016-03-21 09:59 _3.cfe
395658 2016-03-21 09:59 _3.cfs
   256 2016-03-21 09:59 _3.si
   367 2016-03-21 09:59 _4.cfe
421333 2016-03-21 09:59 _4.cfs
   256 2016-03-21 09:59 _4.si
   367 2016-03-21 09:59 _5.cfe
377152 2016-03-21 09:59 _5.cfs
   256 2016-03-21 09:59 _5.si
   367 2016-03-21 09:59 _6.cfe
399195 2016-03-21 09:59 _6.cfs
   256 2016-03-21 09:59 _6.si
   367 2016-03-21 09:59 _7.cfe
257893 2016-03-21 09:59 _7.cfs
   256 2016-03-21 09:59 _7.si
    36 2016-03-21 10:59 segments.gen
   471 2016-03-21 10:59 segments_3
     0 2016-03-21 09:58 write.lock
```

세그먼트

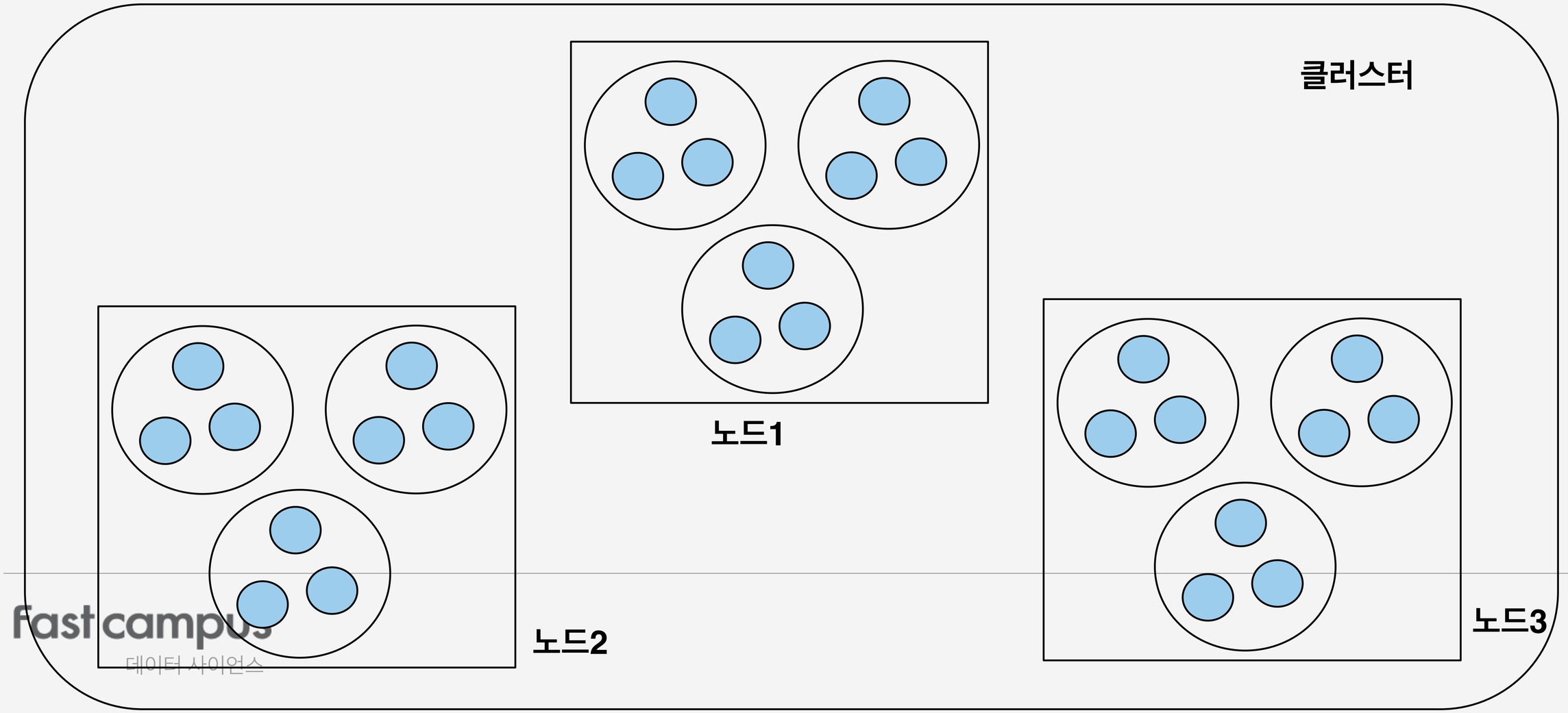
용어정리 - 노드란?

실행중인 엘라스틱 서치 인스턴스 하나를 지칭
통상 서버 1대를 노드로 봄.



용어정리 - 클러스터란?

- 엘라스틱서치의 가장 큰 시스템 단위
- 하나의 클러스터는 여러 개의 노드로 이루어짐



용어정리 - 샤드와 리플리카란?

분산 검색엔진에서 shard란 일종의 파티션과 같은 의미이며,
데이터를 저장할 때 나누어진 하나의 조각에 대한 단위
데이터의 횡적 분할 단위

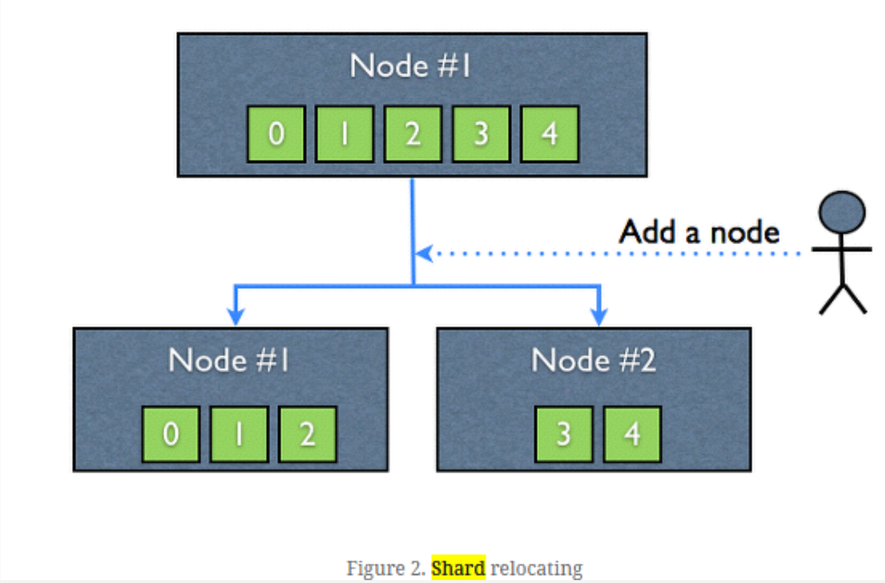
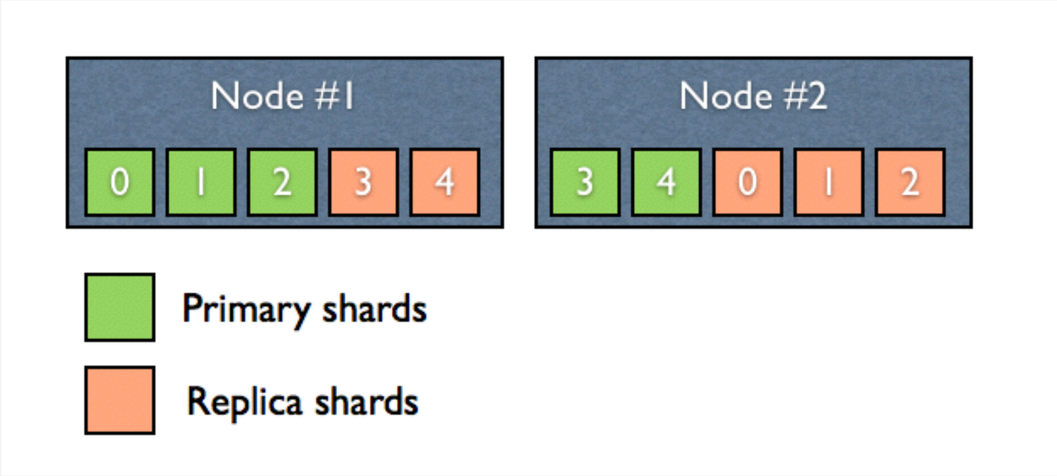


Figure 2. Shard relocating

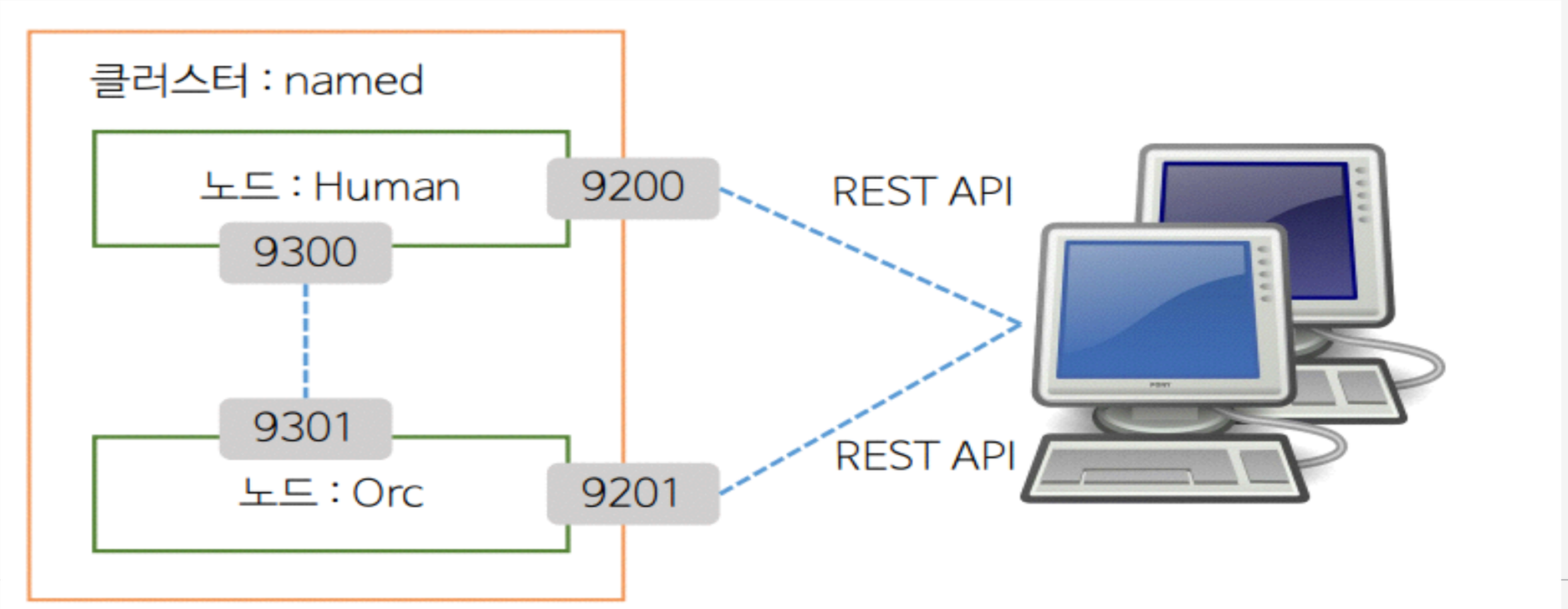


리플리카란 샤드의 복제본
리플리카는 운영중에 그 수를 변경할수 있으나
샤드는 수를 조정 할수 없다.

용어정리 - 바인딩이란?

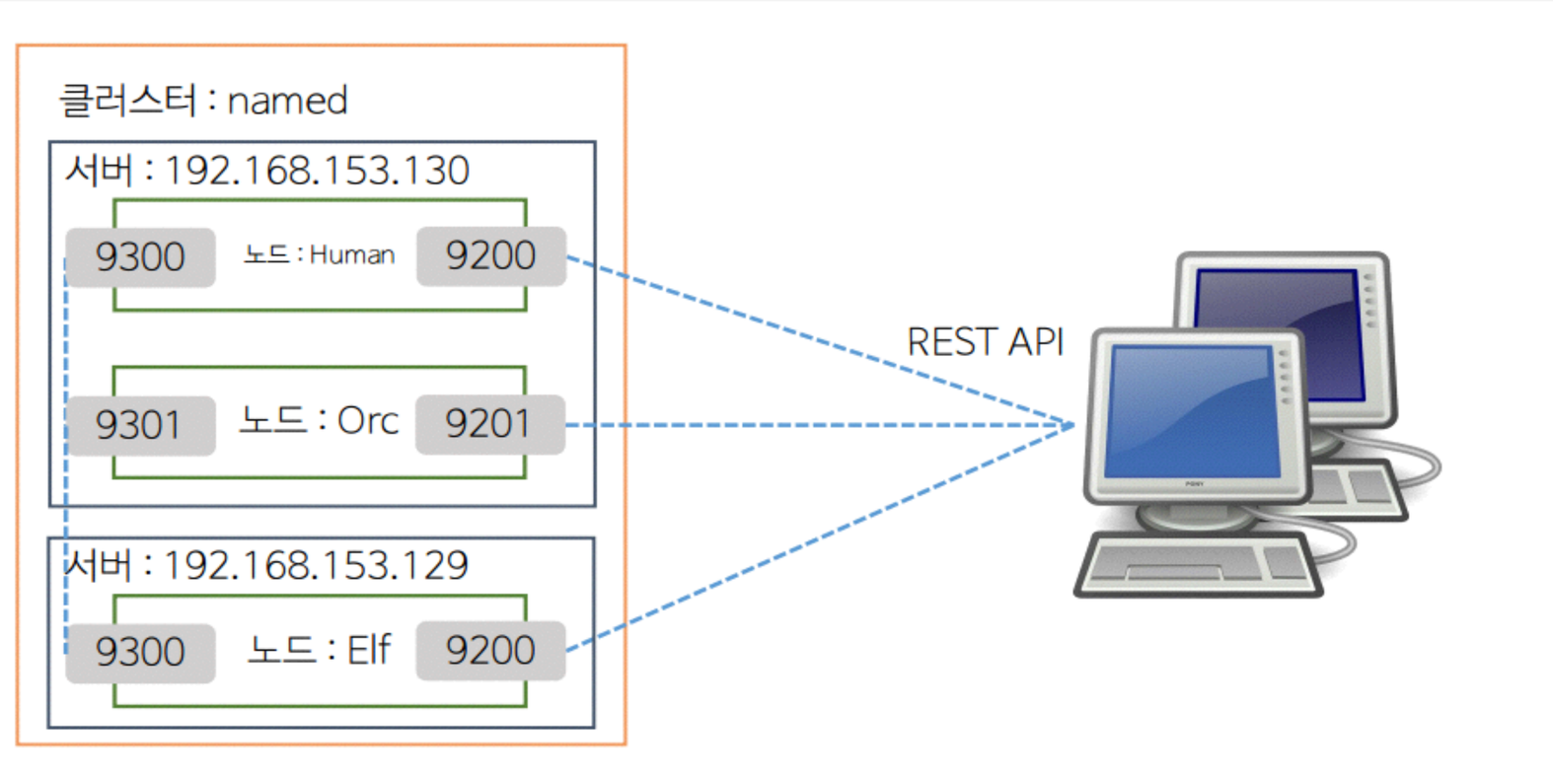
같은 클러스터 이름을 가지고 실행된 노드는 자동으로 묶는 행위

- 9200번부터 REST API를 위한 HTTP 통신 포트가 할당
- 9300번 부터 노드간 바인딩을 위한 포트로 할당



용어정리 - 디스커버리란?

- 효율적인 스케일아웃을 위해 네트워크에 있는 다른 서버의 노드와도 바인딩 가능
- 네트워크 바인딩을 위해 젠 디스커버리(ZEN DISCOVERY) 기능 내장
- 멀티캐스트와 유니캐스트 방식을 모두 지원
- 공식 운영 그룹에서는 유니캐스트의 사용을 권장
- 반드시 두 엘라스틱서치의 버전은 동일해야 함



용어 정리 - 마스터 노드

Data 노드

데이터가 저장되는 노드
인덱싱 및 검색 작업 수행
transport 모듈로 통신하기 때문에 http 모듈은 비활성화

설정 :
node.data : true
node.master : false

Non-Data 노드

마스터 전용 노드 (dedicated master node)

클러스터 - 노드 - 샤드의 맵핑 정보를 담고있는 노드
클러스터 관리 명령 수행하는 노드 (가벼움)
런타임 시 클러스터 멤버 중 마스터 노드로 적합한 노드 자동 선출
한 노드가 데이터 노드와 마스터 노드 역할을 모두 수행하면 클러스터 불안정성 증가
마스터 전용 노드에게는 인덱싱 및 검색 요청을 보내지 않음으로써 클러스터 불안정성 감소

설정 :
node.data : false
node.master : true

클라이언트 노드 (client node)

노드들의 앞단에서 로드 밸런서 역할
HTTP 쿼리 파싱, 검색 요청에 대한 scatter/gatther, 네트워크 부하 담당
데이터 저장, 클러스터 관리 명령은 수행하지 않음
data node는 인덱싱 및 검색 작업에만 집중 가능

설정 :
node.data : false
node.master : false

노드 탐색 메카니즘

탐색

1. 노드 시작
2. 클러스터 이름이 동일한 마스터 노드 탐색
3. 찾으면 클러스터 합류 (마스터 자질이 있는 노드로 합류 마스터는 아님)
4. 못찾으면 자신이 마스터 선택

탐색하는 타입

멀티 캐스트

- 멀티캐스트 메세지에 응답하는 모든 노드 탐색
-네트워크가 멀티캐스트 지원해야 함

유니 캐스트

-호스트를 명시적으로 설정, 각 호스트에 접속시도
- 보안성 우수

게이트웨이와 복구모듈(Recovery)

게이트웨이 모듈 : 클러스터 자료와 메타데이터를 영속적으로 저장
클러스터 복구 과정에서 게이트웨이 모듈로 부터 정보를 읽어옴
gateway.type : local(default)
플러그인 설치시 아마존 S3를 이용할수 있음

복구 제어
복구: 모든 샤드와 리플리카를 초기화 하는 과정. 트랜잭션 로그에서 모든 자료를 읽어 샤드에 적용
gateway.expected_nodes : 구성된 노드수를 알림. 지정된 숫자만큼 노드가 연결되면 복구시작
gateway.recover_after_nodes : 복구 프로세스를 시작할 노드 갯수
gateway.recover_after_time: 복구 프로세스를 시작할 시간 딜레이

2. 엘라스틱서치 설치 실습

- elasticsearch 설치
- head 설치
- 은전한닢 설치

힙메모리 설정

엘라스틱서치 추천

Give (less than) Half Your memory to Lucene

Don’t Cross 32GB!

java -Xmx32600m (java 1.7에서는 disable)

java -Xmx32766m (java 1.8에서는 enable)

시스템에서 스왑기능 꺼라

sudo swapoff -a

E/S 설정에서도 꺼라

bootstrap.mlockall: true

GC 튜닝 전략

GC튜닝 전략의 핵심은 Full GC 시간 줄이기 (Stop the world)

기본적으로 확인해야 하는 옵션

구분	옵션	설명
힙(heap) 영역 크기	-Xms	JVM 시작 시 힙 영역 크기
	-Xmx	최대 힙 영역 크기
New 영역의 크기	-XX:NewRatio	New영역과 Old 영역의 비율
	-XX:NewSize	New영역의 크기
	-XX:SurvivorRatio	Eden 영역과 Survivor 영역의 비율

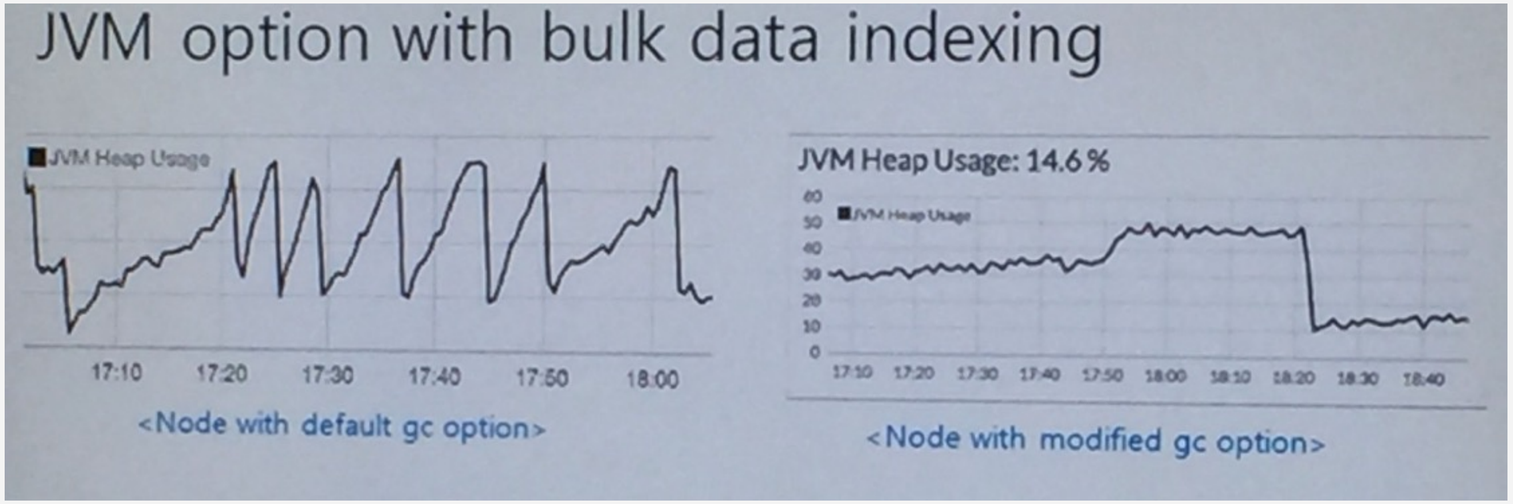
GC 튜닝 전략

GC방식에 따른 옵션

구분	옵션
Serial GC	-XX:+UseSerialGC
Parallel GC	-XX:+UseParallelGC -XX:ParallelGCThreads=value
Parallel Compacting GC	-XX:+UseParallelOldGC
CMS GC	-XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+CMSParallelRemarkEnabled -XX:CMSInitiatingOccupancyFraction=value -XX:+UseCMSInitiatingOccupancyOnly
G1	-XX:+UnlockExperimentalVMOptions -XX:+UseG1GC

GC 튜닝 전략

참고자료



-XX:MaxTenuringThreshold=15 (신세대의 객체가 얼마만큼의 시간이 지나야 구세대로 승진되는지 지정

-XX:NewRatio=7 (전체 힙 크기 중 New 크기의 비율)

-XX:SurvivorRatio=3 (객체가 구세대로 승진되기 전에 사용해야 하는 생존 공간 heap의 희망 비율)

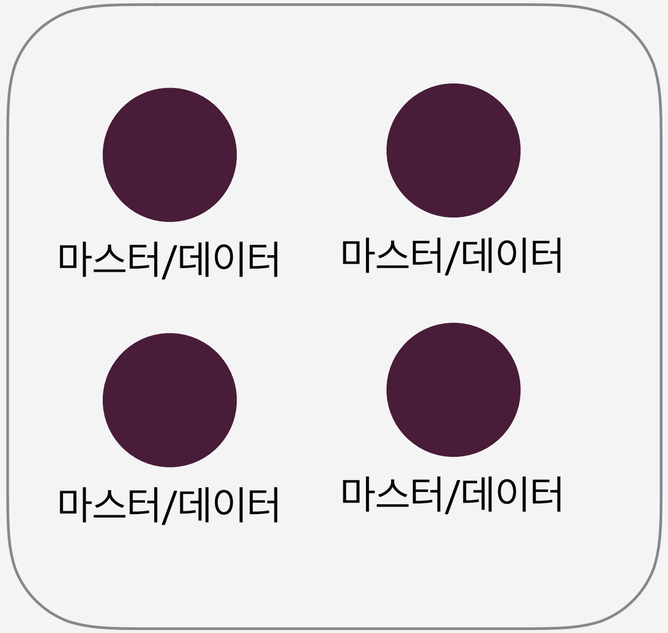
-XX:-UseAdaptiveSizePolicy (New Generation의 크기가 Survivor Space의 크기를 변경할 것인지의 여부)

샤드 갯수에 따른 성능 변화

참고자료

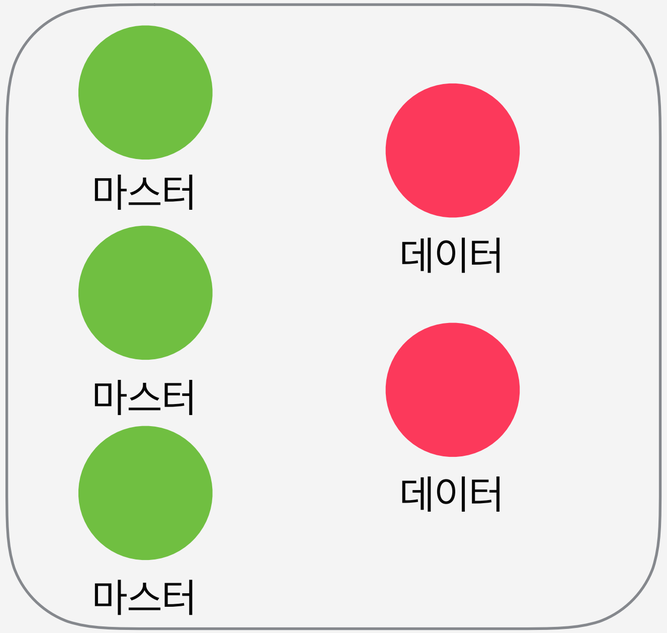
샤드갯수	사이즈	문서량	쿼리1응답평균	쿼리2응답평균	쿼리3응답평균
5	17gb	40M	0.6524	0.7728	0.8876
10	8.5gb	20M	0.5328	0.5554	0.4526
20	4.2gb	10M	0.8972	0.5044	0.5578

노드 구성 아키텍쳐



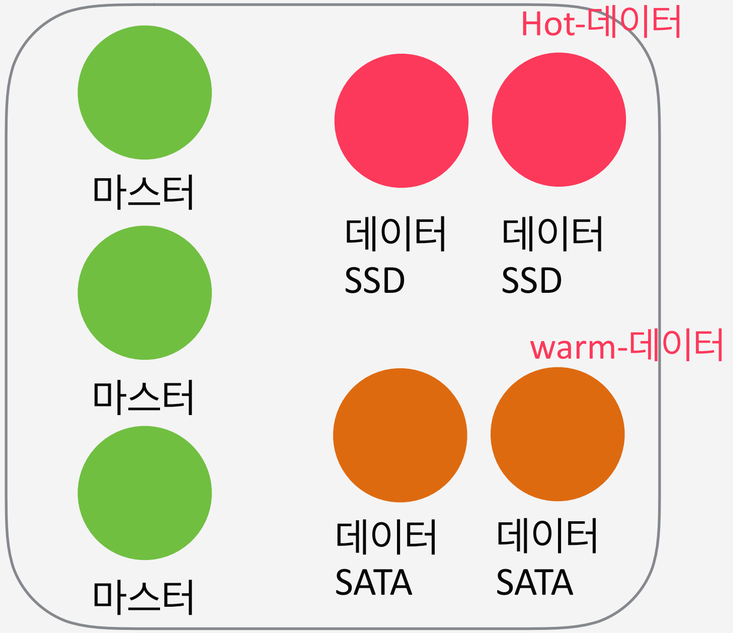
가장 기본적인 모델
split brain 문제 고려해야함

master와 data를 구분



가장 널리 쓰이는 모델
마스터를 최소 3대로 구성

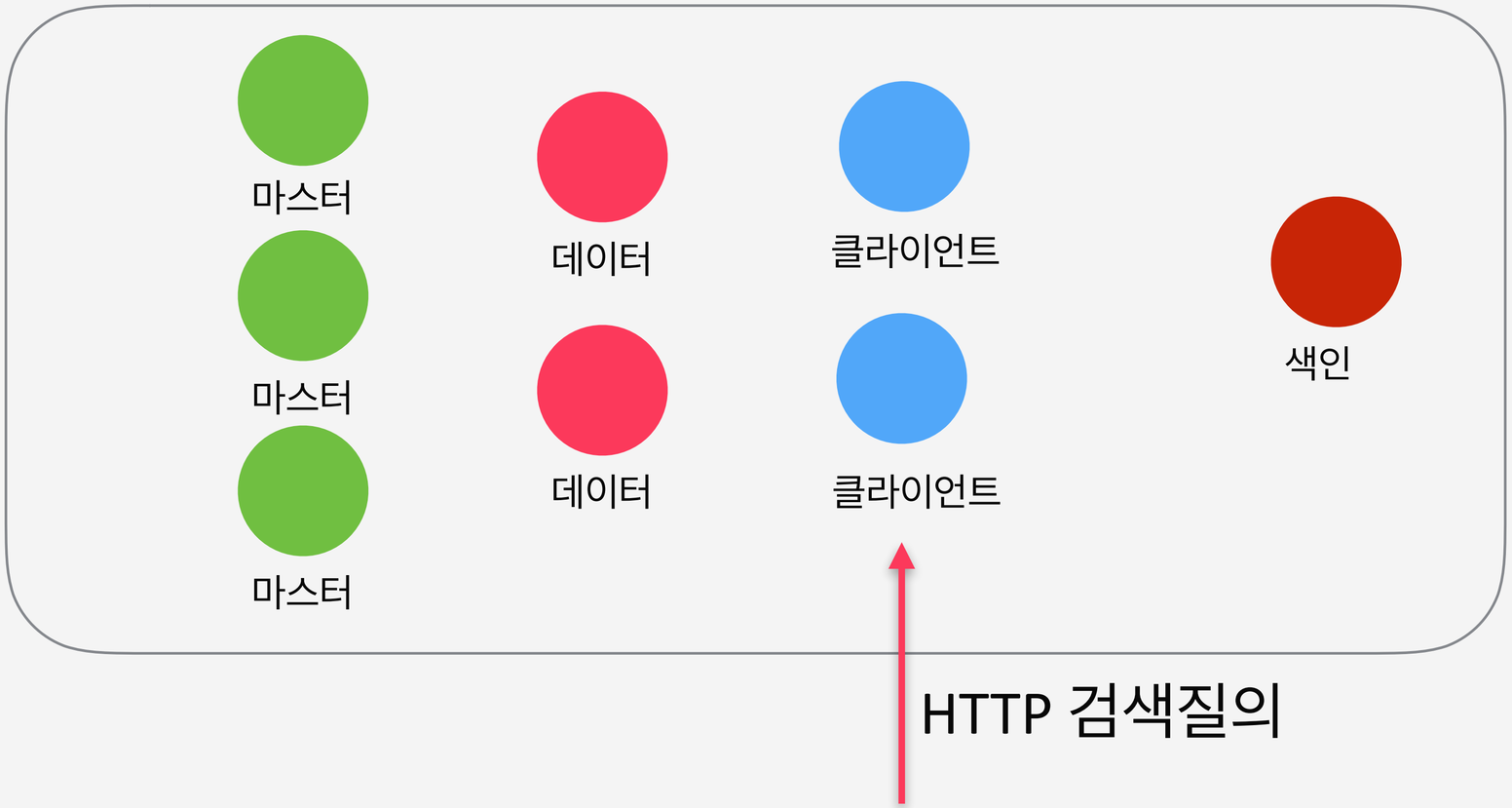
Hot-warm 아키텍쳐



(https://www.elastic.co/blog/hot-warm-architecture)

Elastic쪽에서 제안한 모델

노드 구성 아키텍쳐



2. 엘라스틱서치 설치 실습

- elasticsearch 설치
- head 설치
- 은전한닢 설치

3. 엘라스틱서치 데이터 처리 실습

Elasticsearch 데이터 처리

- 엘라스틱서치는 주로 REST API를 통하여 데이터를 처리
- HTTP METHOD는 GET, POST, PUT, DELETE, HEAD 등이 있음
- 엘라스틱서치의 REST API를 이용하기 위한 형태는 아래와 같음

curl –X{METHOD} http://host:port/{INDEX}/{TYPE}/{ID} –d '{DATA}'

HTTP METHOD, CRUD, SQL 비교

HTTP METHOD	CRUD	SQL
GET	READ	SELECT
PUT	UPDATE	UPDATE
POST	CREATE	INSERT
DELETE	DELETE	DELETE

RESTFul이란?

REST는 웹의 창시자(HTTP) 중의 한 사람인 Roy Fielding의 2000년 논문에 의해서 소개
현재의 아키텍쳐가 웹의 본래 설계의 우수성을 많이 사용하지 못하고 있다고 판단
웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍쳐를 소개
그것이 바로 Representational state transfer (REST)

REST는 요소로는 크게 리소스,메서드,메세지

HTTP POST , http://myweb/users/
{ Method Resource (URI)
"users":{
"name":"terry"
} Message
}

메서드	의미	Idempotent
POST	Create	No
GET	Select	Yes
PUT	Update	Yes
DELETE	Delete	Yes

CRUD를 메서드로 표현

3. 클러스터 관리 (실습)

4. 매핑 (실습)

Mapping

ElasticSearch 별도스키마 설정이 없이도 자동으로 자료구조를 이해하고 저장
특정 인덱스의 명시적 구조를 정의
한 인덱스에 매핑을 여러개 만들수 있으나필드명은 중복 불가
매핑은 필드의 추가는 가능, 수정 삭제는 불가능
기존 매핑과 충돌의 경우 무시하는 옵션이 있었으나 Deprecated (~~ignore_conflicts~~)
매핑 삭제 옵션은 제공X (~~2.0부터 DELETE mapping API 없음~~)

```
$ curl -XPUT 'http://localhost:9200/books/book/_mapping' -d '
{
    "book" : {
        "properties" : {
            "projectName" : {"type" : "string", "index" : "not_analyzed"},
            “bookName" : {"type" : “string", "index" : "analyzed“,”index_analyzer”:” analysis-korean”},
            “bookContent" : {"type" : "string", "index" : "not_analyzed"},
            “regDate" : {"type" : "date"},
            "host" : {"type" : "string", "index" : "not_analyzed"},
            "body" : {"type" : "string"},
        }
    }
}
'
```

Mapping

매핑 타입 종류

Meta-fields : 매핑을 설정하지 않아도 시스템에서 자동으로 필수로 만드는 필드
_index, _type, _id, and _source

Fields 혹은 Properties : 사용자가 정의해서 만드는 필드

필드 타입 종류

Simple type : text,keyword,date,long,double,boolean,ip

Hierarchical type : object or nested.

etc ; geo_point, geo_shape, or completion

매핑관련 세팅값
index.mapping.total_fields.limit
index.mapping.depth.limit
index.mapping.nested_fields.limit

Mapping (text 타입 가능한 옵션)

analyzer : 분석기 지정 (Default standard)
boost : 필드별 가중치
fielddata : 집계 정렬을 위해 필드데이터를 메모리에 올릴것인지 여부
fields : 집계나 정렬을 위해 같은 값을 다른 분석기로 분석할지 여부
index : 색인 할것인지 여부(default false)
index_options : 색인시 텀,빈도,포지션,오프셋 값을 저장할것인지 여부
norms : 스코어링에서 문서의 길이를 적용시킬것인지
store : 데이터를 저장 할것인지 여부 (_source 와 별도)
search_analyzer : 검색시 분석기를 별도로 지정할 것인지 여부
term_vector : 텀벡터를 저장할 것인지 여부
copy_to : 필드의 원본을 복사하여 다른 필드로 전송

Mapping (array vs object vs nested)

```
{
 "title": "Nest eggs",
 "body":  "Making your money work...",
 "tags":  [ "cash", "shares" ],
 "comments": [
  {
   "name":    "John Smith",
   "comment": "Great article",
   "age":     28,
   "stars":   4,
   "date":    "2014-09-01"
  },
  {
   "name":    "Alice White",
   "comment": "More like this please",
   "age":     31,
   "stars":   5,
   "date":    "2014-10-22"
  }
 ]
}
```

검색

```
GET /_search
{
 "query": {
  "bool": {
   "must": [
    { "match": { "name": "Alice" }},
    { "match": { "age":  28     }}
   ]
  }
 }
}
```

```
{
 "title":            [ eggs, nest ],
 "body":             [ making, money, work, your ],
 "tags":             [ cash, shares ],
 "comments.name":    [ alice, john, smith, white ],
 "comments.comment": [ article, great, like, more, please, this ],
 "comments.age":     [ 28, 31 ],
 "comments.stars":   [ 4, 5 ],
 "comments.date":    [ 2014-09-01, 2014-10-22 ]
}
```

cross-object 매칭 발생

Mapping (array vs nested)

이러한 문제를 풀기 위해 nested objects가 디자인

단 내포된 오브젝트 만으로 검색 결과로 리턴될 수 없음

```
{ (1)
  "comments.name":    [ john, smith ],
  "comments.comment": [ article, great ],
  "comments.age":     [ 28 ],
  "comments.stars":   [ 4 ],
  "comments.date":    [ 2014-09-01 ]
}
{ (2)
  "comments.name":    [ alice, white ],
  "comments.comment": [ like, more, please, this ],
  "comments.age":     [ 31 ],
  "comments.stars":   [ 5 ],
  "comments.date":    [ 2014-10-22 ]
}
{ (3)
  "title":          [ eggs, nest ],
  "body":           [ making, money, work, your ],
  "tags":           [ cash, shares ]
}
```

```
PUT /my_index
{
 "mappings": {
  "blogpost": {
   "properties": {
    "comments": {
     "type": "nested",
     "properties": {
      "name":    { "type": "string"  },
      "comment": { "type": "string"  },
      "age":     { "type": "short"   },
      "stars":   { "type": "short"   },
      "date":    { "type": "date"    }
     }
    }
   }
  }
 }
}
```

nested query 사용

```
GET /my_index/blogpost/_search
{
  "query": {
   "bool": {
    "must": [
     { "match": { "title": "eggs" }}, (1)
     {
      "nested": {
       "path": "comments", (2)
       "query": {
        "bool": {
         "must": [
          { "match": { "comments.name": "john" }},(3)
          { "match": { "comments.age":  28     }}
         ]
       }}}}
    ]
}}}
```

감사합니다.

다음 주제 : 5일차 ElasticSearch Chapter 2
텍스트 데이터 분석
검색 질의