**API**

노드 상태 점검

# Local
curl localhost:9200/\_cluster/nodes/\_local
# Address
curl localhost:9200/\_cluster/nodes/10.0.0.3,10.0.0.4
curl localhost:9200/\_cluster/nodes/10.0.0.\*
# Names
curl localhost:9200/\_cluster/nodes/node\_name\_goes\_here
curl localhost:9200/\_cluster/nodes/node\_name\_goes\_\*
# Attributes (set something like node.rack: 2 in the config)
curl localhost:9200/\_cluster/nodes/rack:2
curl localhost:9200/\_cluster/nodes/ra\*:2
curl localhost:9200/\_cluster/nodes/ra\*:2\*

결과를 보기 편하게

url 뒤에 ?pretty=true 를 붙이면 결과가 보기 편하게 출력된다. 테스트시에만 사용하길 권장

**Index API**

다음 예제는 JSON형식의 문서를 twitter 인덱스의 tweet 타입에 1번 id로 저장한다.

curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '{
 "user" : "kimchy",
 "post\_date" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
}'

**자동 인덱스 생성 (**Automatic Index Creation)

인덱스 작업은 자동으로 없는 인덱스를 생성한다. 문서의 형식을 맵핑이라고 하는듯 하며 형식은 자유롭다. 타입이 생성된적이 없을 경우 이 맵핑도 자동으로 생성된다.

자동 인덱스 생성 옵션은 설정파일에서 action.auto\_create\_index: false 옵션을 줘서 비활성화 할 수 있다.

맵핑 자동생성도 마찬가지로 설정파일에서 index.mapper.dynamic: false 옵션으로 비활성화 할 수 있다.

**버저닝(Versioning)**

모든 문서들은 버전번호가 주어지며, 이 버전정보는 인덱스API 호출시 결과로 전달된다.

인덱스API 호출시 옵션으로 버전을 명시할 수 있는데 이 경우 현재 버전이 옵션에서 명시한 버전과 일치할 경우에만 문서를 업데이트 한다. 트랜잭션처럼 문서를 읽은다음 그 결과를 업데이트 하기 전에 변경되지 않았음을 보장해 준다. (업데이트를 위해 읽을 경우 preference를 \_primary로 설정하기를 권장(?))

curl -XPUT 'localhost:9200/twitter/tweet/1?version=2' -d '{
 "message" : "elasticsearch now has versioning support, double cool!"
}'

버전정보를 명시하지 않을경우 버전을 체크하지 않는다.

기본적으로 버전은 업데이트시마다 1부터 자동으로 증가한다. 버전을 명시해서 저장할 수도있는데 이때는version\_type=external 옵션을 준다. 이 옵션을 사용하면 명시한 버전번호를 현재 버전과 비교하지 않고 현재 버전보다 큰지만을 체크한다. 현재 버전보다 클 경우 저장되나 그렇지 않을 경우 인덱스 작업은 실패하게 된다. 데이터베이스의 데이터를 인덱싱 할 경우 최종 결과만을 남겨놓고 싶을 경우 사용할 수 있다.

**작업 타입(Operation Type)**

인덱스 작업은 기본적으로 없을경우 문서를 등록하지만 이미 있을경우 업데이트 한다. op\_type 옵션을 create로 줄 경우 해당 문서가 없으면 생성하지만 이미 있을 경우 작업이 실패하게 된다.

curl -XPUT 'http://localhost:9200/twitter/tweet/1?op\_type=create' -d '{
 "user" : "kimchy",
 "post\_date" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
}'

다음과 같이 작성해도 된다.

curl -XPUT 'http://localhost:9200/twitter/tweet/1/\_create' -d '{
 "user" : "kimchy",
 "post\_date" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
}'

**자동 ID 생성 (Automatic ID Generation)**

인덱스 작업시 id를 명시하지 않을 경우 자동으로 id가 부여된다. 이 경우 PUT이 아니라 POST로 호출해야 한다.

curl -XPOST 'http://localhost:9200/twitter/tweet/?pretty=true' -d '{

 "user" : "kimchy",

 "post\_date" : "2009-11-15T14:12:12",

 "message" : "trying out Elastic Search"

}'

**라우팅 (Routing)**

기본적으로 샤드의 위치는 문서의 id의 해쉬값을 이용해 이루어진다. 이를 좀 더 명확히 조절하기 위해 해쉬함수에 제공될 값으로 routing 파라미터를 명시할 수 있다.

curl -XPOST 'http://localhost:9200/twitter/tweet?routing=kimchy' -d '{
 "user" : "kimchy",
 "post\_date" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
}'

이 예제의 “tweet” 문서는 routing파라미터의 값인 “kimchy”라는 값을 기준으로 샤드에 배치된다.

맵핑에 \_routing를 required로 설정하면 routing값이 없이는 인덱싱 작업이 실패하게 된다.

부모와 자식 (Parents & Children)

자식 문서는 인덱스 작업시 부모를 명시함으로서 이루어진다.

curl -XPUT localhost:9200/blogs/blog\_tag/1122?parent=1111 -d '{
 "tag" : "something"
}'

자식 문서를 인덱싱 할 때 routing 값을 명시적으로 제공하지 않으면 부모 문서의 값을 따른다.

타임스탬프 (Timestamp)

문서는 인덱싱 작업시 타임스탬프를 명시할 수 있다. 이는 timestamp파라미터를 통해 적용된다.

curl -XPUT localhost:9200/twitter/tweet/1?timestamp=2009-11-15T14%3A12%3A12 -d '{
 "user" : "kimchy",
 "message" : "trying out Elastic Search",
}'

TTL

문서는 인덱스 작업시 유효기간을 지정할 수 있으며 이 기간이 지나면 해당 문서는 자동으로 삭제된다. 이 유효기간은 timestamp를 기준으로 동작한다. 숫자만 등록할 시 milliseconds단위로 지정되며 다음과 같이 다양한 방법으로 지정할수도 있다.

curl -XPUT 'http://localhost:9200/twitter/tweet/1?ttl=86400000' -d '{
 "user": "kimchy",
 "message": "Trying out elasticsearch, so far so good?"
}'

curl -XPUT 'http://localhost:9200/twitter/tweet/1?ttl=1d' -d '{
 "user": "kimchy",
 "message": "Trying out elasticsearch, so far so good?"
}'

curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '{
 "\_ttl": "1d",
 "user": "kimchy",
 "message": "Trying out elasticsearch, so far so good?"
}'

**Delete API**

다음 예제는 twitter인덱스의 tweet타입의 1번 id를 가지는 문서를 삭제한다.

curl -XDELETE 'http://localhost:9200/twitter/tweet/1'

**버저닝 (Versioning)**

정확한 버전 관리를 위해 삭제시 버전 번호를 명시할 수 있다. 이 경우 현재 버전이 명시된 버전과 다를시 삭제되지 않는다.

**라우팅 (Routing)**

인덱스 작업시 routing파라미터를 명시했을 경우 삭제시에도 routing파라미터를 명시해야 한다. 만약 정확한 값을 입력하지 않을 경우 문서는 삭제되지 않는다.

curl -XDELETE 'http://localhost:9200/twitter/tweet/1?routing=kimchy'

삭제할 경우에 routing값을 알지 못하는 경우가 많은데 이때는 \_routing 맵핑을 required로 설정한다. 이럴 경우 삭제시 routing값을 명시하지 않으면 모든 샤드로 삭제명령이 브로드캐스트 된다.

**인덱스 자동생성 (Automatic for the Index)**

인덱스 API와 마찬가지로 삭제 API도 자동으로 해당 인덱스와 타입을 생성한다.

**Get API**

다음 예제는 twitter인덱스의 tweet타입의 1번 id를 가지는 문서를 조회한다.

curl -XGET 'http://localhost:9200/twitter/tweet/1'

문서의 존재여부만 조회할 시에는 HEAD를 사용해도 된다.

curl -XHEAD 'http://localhost:9200/twitter/tweet/1'.

**타입 옵션 (Optional Type)**

조회시 타입을 \_all로 설정하면 모든 타입들 중 하나의 문서가 반환된다.

curl –XGET localhost:9200/twitter/\_all/1

**필드 (Fields)**

다음과 같이 fields파라미터를 줌으로서 조회할 필드를 명시할 수도 있다.

curl -XGET 'http://localhost:9200/twitter/tweet/1?fields=title,content'

obj1.obj2 형식으로 하위 오브젝트를 가져올수도 있다.

**라우팅 (Routing)**

Routing 파라미터를 사용해 인덱싱 했다면 조회시에도 동일한 값을 입력해야만 조회가 가능하다.

curl -XGET 'http://localhost:9200/twitter/tweet/1?routing=kimchy'

**Multi Get API**

인덱스, 타입(옵션), id(및 라우팅)을 기반으로 여러 개의 문서를 조회한다.

curl 'localhost:9200/\_mget' -d '{
 "docs" : [
 {
 "\_index" : "test",
 "\_type" : "type",
 "\_id" : "1"
 },
 {
 "\_index" : "test",
 "\_type" : "type",
 "\_id" : "2"
 }
 ]
}'

하나의 인덱스에 대해서만 조회도 가능하다. 이 경우에는 body에 인덱스를 명시하지 않는다.

curl 'localhost:9200/test/\_mget' -d '{
 "docs" : [
 {
 "\_type" : "type",
 "\_id" : "1"
 },
 {
 "\_type" : "type",
 "\_id" : "2"
 }
 ]
}'

타입도 마찬가지

curl 'localhost:9200/test/type/\_mget' -d '{
 "docs" : [
 {
 "\_id" : "1"
 },
 {
 "\_id" : "2"
 }
 ]
}'

이 경우는 다음과 같이 간단히 사용할수도 있다.

curl 'localhost:9200/test/type/\_mget' -d '{
 "ids" : ["1", "2"]
}'

**필드 (Fields)**

문서별로 조회할 필드를 명시할 수 있다.

curl 'localhost:9200/\_mget' -d '{
 "docs" : [
 {
 "\_index" : "test",
 "\_type" : "type",
 "\_id" : "1",
 "fields" : ["field1", "field2"]
 },
 {
 "\_index" : "test",
 "\_type" : "type",
 "\_id" : "2",
 "fields" : ["field3", "field4"]
 }
 ]
}'

**Update API**

입력된 스크립트를 기반으로 문서를 업데이트 한다. 이 동작은 인덱스에서 문서를 조회하고, 스크립트를 실행하고, 결과를 재인덱싱 한다. 조회 작업과 재인덱싱 도중 문서에 변화가 없는지 체크하기 위해 versioning을 사용한다.

이 동작은 여전히 문서의 full reindexing을 의미한다. 단지 네트웍상의 데이터 왕복이 없고 조회와 재인덱싱 사이의 버전 충돌 가능성을 줄일 뿐이다. 이 기능이 동작하기 위해선 \_source필드가 사용 가능해야 한다.

첫번째로 문서를 인덱싱한다.

curl -XPUT localhost:9200/test/type1/1 -d '{
 "counter" : 1,
 "tags" : ["red"]
}'

다음으로 카운터를 증가시키는 스크립트를 실행한다.

curl -XPOST 'localhost:9200/test/type1/1/\_update' -d '{
 "script" : "ctx.\_source.counter += count",
 "params" : {
 "count" : 4
 }
}'

태그 목록에 태그를 추가할 수도 있다.

curl -XPOST 'localhost:9200/test/type1/1/\_update' -d '{
 "script" : "ctx.\_source.tags += tag",
 "params" : {
 "tag" : "blue"
 }
}'

새로운 필드를 추가할수도 있다.

curl -XPOST 'localhost:9200/test/type1/1/\_update' -d '{
 "script" : "ctx.\_source.text = \"some text\""
}'

필드를 삭제할수도 있다.

curl -XPOST 'localhost:9200/test/type1/1/\_update' -d '{
 "script" : "ctx.\_source.remove(\"text\")"
}'

특정 태그가 포함되면 문서를 삭제하고 그렇지 않을경우에는 무시한다.

curl -XPOST 'localhost:9200/test/type1/1/\_update' -d '{
 "script" : "ctx.\_source.tags.contains(tag) ? ctx.op = \"delete\" : ctx.op = \"none\"",
 "params" : {
 "tag" : "blue"
 }
}'

업데이트는 Index API와 유사하게 다음 파라미터들을 지원한다.

routing – 적절한 샤드의 위치를 찾기 위해 라우팅값을 설정한다.

parent – 경로를 설정한다.

timeout – 샤드가 사용가능해질때까지 대기시간

replication – 삭제, 인덱스 작업시 리플리케이션 방법 (sync와 async)

consistency – 삭제, 인덱스 작업시 쓰기 일관성

percolate - ?

refresh – 업데이트된 내용이 검색 결과에 바로 반영되도록 인덱싱 결과를 즉시 반영

추가로 조회/인덱싱(삭제) 사이의 버전 충돌에 대비해 재시도 횟수를 설정이가능하다. retry\_on\_conflict 기본값은 0

Timestamp와 ttl도 ctx.\_ttl와 ctx.\_timestamp를 통해 업데이트 가능하다.

**Search API**

검색 API는 검색 쿼리를 실행해 결과를 반환한다. 검색은 인덱스와 타입에 상관 없이 실행 가능하다. 검색 쿼리는 파라미터로 입력할 수도 있고 request body로 입력할 수도 있다.

**라우팅 (Routing)**

검색을 실행할 경우 해당 인덱스를 가진 모든 샤드에 broadcast된다.(레플리카 간에는 Round Robin) routing파라미터를 입력하면 어느 샤드를 검색할지 지정할 수 있다.

다음과 같이 routing을 명시해서 인덱싱을 했다고 생각해보자.

curl -XPOST 'http://localhost:9200/twitter/tweet?routing=kimchy' -d '{
 "user" : "kimchy",
 "postDate" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
}'

이런 경우 특정 사용자의 tweet만 검색한다고 하면 routing파라미터를 입력하여 관련된 샤드만 검색하게 된다.

$ curl -XGET 'http://localhost:9200/twitter/tweet/\_search?routing=kimchy' -d '{
 "query": {
 "filtered" : {
 "query" : {
 "query\_string" : {
 "query" : "some query string here"
 }
 },
 "filter" : {
 "term" : { "user" : "kimchy" }
 }
 }
 }
}'

routing파라미터를 여러 개 입력하고 싶을때는 comma로 구분해서 표현한다.

**Search API – Request Body**

예제

$ curl -XGET 'http://localhost:9200/twitter/tweet/\_search' -d '{
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}'

결과

{
 "\_shards":{
 "total" : 5,
 "successful" : 5,
 "failed" : 0
 },
 "hits":{
 "total" : 1,
 "hits" : [
 {
 "\_index" : "twitter",
 "\_type" : "tweet",
 "\_id" : "1",
 "\_source" : {
 "user" : "kimchy",
 "postDate" : "2009-11-15T14:12:12",
 "message" : "trying out Elastic Search"
 }
 }
 ]
 }
}

**파라미터 (Parameters)**

timeout – 시간 제한. 검색 도중 시간이 만료되면 그때까지의 결과만을 가지고 반환된다. 디폴트는 시간제한이 없음

from - 검색 결과 중 반환을 시작할 번호. 기본값은 0

size – 반환할 결과의 수. 기본값은 10

search\_type – 수행할 검색의 방법. dfs\_query\_then\_fetch, dfs\_query\_and\_fetch, query\_then\_fetch, query\_and\_fetch가 있다. 기본값은 query\_then\_fetch

위의 파라미터 중 search\_type은 request body로 요청할 수 없고 파라미터로만 입력해야 한다.

검색은 HTTP GET/POST모두로 요청이 가능하지만 POST와는 다르게 GET에서 body를 전송하는 방법은 지원하는 클라이언트가 많이 없다.

**Search API – From / Size**

파라미터로 설정할 수 있고 request body에 설정할 수도 있다. 기본값은 from이 0, size가 10이다.

{
 "from" : 0, "size" : 10,
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

**Search API – Indices And Types**

검색 API는 하나의 인덱스 안에서 여러 개의 타입을 검색할 수 있고, 여러 개의 인덱스를 검색할 수도 있다.

twitter인덱스의 모든 문서 검색

curl -XGET 'http://localhost:9200/twitter/\_search?q=user:kimchy'

특정 타입만 검색할 수도 있다.

curl -XGET 'http://localhost:9200/twitter/tweet,user/\_search?q=user:kimchy'

여러 인덱스에서 모든 tweet타입중 특정 태그를 가진 문서를 검색

curl -XGET 'http://localhost:9200/kimchy,elasticsearch/tweet/\_search?q=tag:wow'

\_all을 이용해 모든 인덱스에서 tweet타입을 검색

curl - XGET 'http://localhost:9200/\_all/tweet/\_search?q=tag:wow'

모든 인덱스의 모든 타입을 검색

curl -XGET 'http://localhost:9200/\_search?q=tag:wow'

**Search API – Sort**

특정 필드에 하나 이상의 정렬방법을 설정할 수 있으며 역순으로 정렬도 가능하다. 점수로 정렬할 시에는 \_score라는 특수필드를 사용한다.

{
 "sort" : [
 { "post\_date" : {"order" : "asc"} },
 "user",
 { "name" : "desc" },
 { "age" : "desc" },
 "\_score"
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

JSON 파서가 배열을 지원하지 않으면 다음과 같이 입력해도 된다.

{
 "sort" : {
 { "post\_date" : {"order" : "asc"} },
 "user" : { },
 "\_score" : { }
 },
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

**정렬 값 (Sort Value)**

각 문서에서 정렬에 사용된 값도 결과에 포함되어 반환한다.

**값이 없는 필드 (Missing Value)**

숫자 필드는 값이 없는 필드를 다룰 방법을 지정할 수 있다. Missing 값은 \_last 또는 \_first를 넣거나 특정한 값(문서 정렬에서 사용될)을 넣을 수 있다.

{
 "sort" : [
 { "price" : {"missing" : "\_last"} },
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

**맵핑되지 않은 필드 무시 (Ignoring Unmapped Fields)**

기본적으로 필드와 맵핑되지 않았을 경우엔 검색에 실패한다. Ignore\_unmapped옵션은 맵핑이 없는 필드는 무시하고 그 값으로 정렬하지 않는다.

{
 "sort" : [
 { "price" : {"ignore\_unmapped" : true} },
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

**거리로 정렬 (Geo Distance Sorting)**

{
 "sort" : [
 {
 "\_geo\_distance" : {
 "pin.location" : [-70, 40],
 "order" : "asc",
 "unit" : "km"
 }
 }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

다음 형식으로 위치정보를 제공해도 된다.

Lat Lon as Property

{
 "sort" : [
 {
 "\_geo\_distance" : {
 "pin.location" : {
 "lat" : 40,
 "lon", -70
 }
 "order" : "asc",
 "unit" : "km"
 }
 }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

Lat Lon as String

{
 "sort" : [
 {
 "\_geo\_distance" : {
 "pin.location" : "-70,40",
 "order" : "asc",
 "unit" : "km"
 }
 }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

Geohash

{
 "sort" : [
 {
 "\_geo\_distance" : {
 "pin.location" : "drm3btev3e86",
 "order" : "asc",
 "unit" : "km"
 }
 }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

Lat Lon as Array

{
 "sort" : [
 {
 "\_geo\_distance" : {
 "pin.location" : [-70, 40],
 "order" : "asc",
 "unit" : "km"
 }
 }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

스크립트 기반 정렬 (Script Based Sorting)

{
 "query" : {
 ....
 },
 "sort" : {
 "\_script" : {
 "script" : "doc['field\_name'].value \* factor",
 "type" : "number",
 "params" : {
 "factor" : 1.1
 },
 "order" : "asc"
 }
 }
}

Note, it is recommended, for single custom based script based sorting, to use custom\_score query instead as sorting based on score is faster. (?)

**스코어 계산 (Track Score)**

필드를 기준으로 정렬할 때 score는 계산되지 않는다. track\_scores를 true로 줘서 score를 계산할 수 있다.

{
 "track\_scores": true,
 "sort" : [
 { "post\_date" : {"reverse" : true} },
 { "name" : "desc" },
 { "age" : "desc" }
 ],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

Search API – Fields

검색된 문서에서 읽어들일 필드를 선택할 수 있다. 디폴트는 \_source필드이다.

{
 "fields" : ["user", "postDate"],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

fields는 자동으로 저장된 모든 필드들을 로드한다.(store 맵핑이 yes일 경우) 만약 저장되지 않은 경우에는 \_source를 읽어들여 추출하게 된다.(중첩된 문서객체를 반환할 수 있도록)

\*를 사용하면 저장된 모든 필드를 반환한다.

빈 배열을 넘기면 검색 결과에 대해서 \_id와 \_type만을 반환한다.

{
 "fields" : [],
 "query" : {
 "term" : { "user" : "kimchy" }
 }
}

Script 필드도 자동으로 감지되어 필드처럼 사용할 수 있다. \_source.obj1.obj2로 사용할 수 있으며, 권장하진 않지만 obj1.obj2 형식도 동작한다.