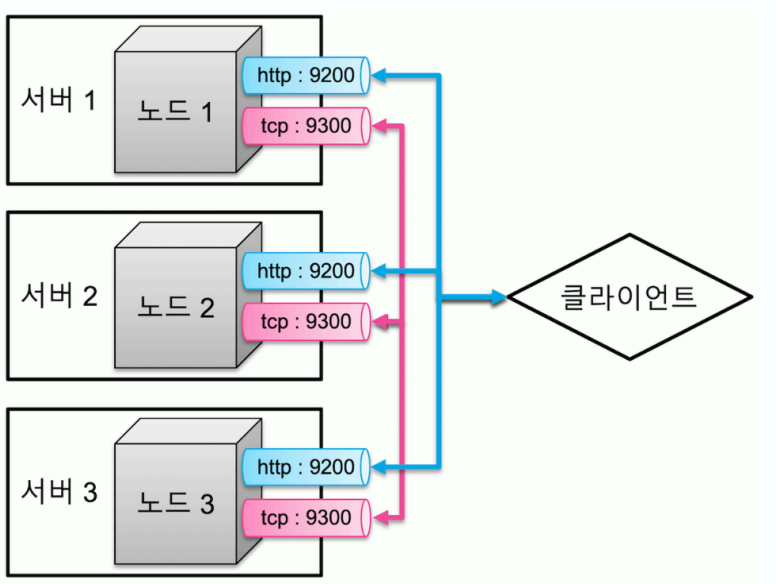
Elasticsearch 시스템구조

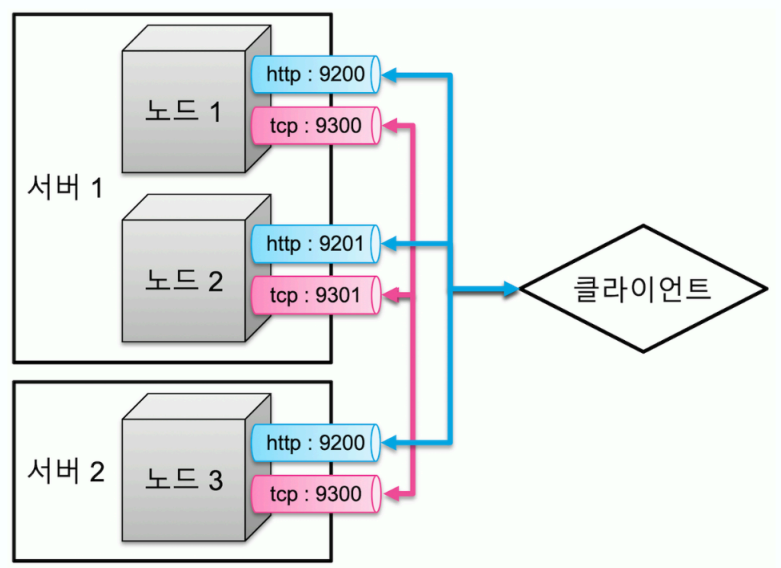
클러스터 구성

여러 서버에 하나의 클러스터로 실행

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



하나의 물리적인 서버 안에서 여러 개의 노드를 실행하는 것도 가능합니다. 이 경우에는 각 노드들은 차례대로 9200, 9201... 순으로 포트를 사용하게 됩니다. 클라이언트는 9200, 9201 등의 포트를 통해 원하는 노드와 통신을 할 수 있습니다. 만약에 서버1 에서 두개의 노드를 실행하고, 또 다른 서버에서 한개의 노드를 실행시키면 클러스터는 다음과 같이 구성됩니다.



서버1에는 두개의 노드가 있기 때문에 서버 1의 두번째 노드는 실행되는 http, tcp 포트가 각각 9201, 9301 로 실행이 됩니다.

물리적인 구성과 상관 없이 여러 노드가 하나의 클러스터로 묶이기 위해서는 클러스터명 `cluster.name` 설정이 묶여질 노드들 모두 동일해야 합니다. 같은 서버나 네트워크망 내부에 있다 하더라도 `cluster.name`이 동일하지 않으면 논리적으로 서로 다른 클러스터로 실행이 되고, 각각 별개의 시스템으로 인식이 됩니다.

하나의 서버에서 여러 클러스터 실행

하나의 물리 서버에 3개의 노드를 실행시킨다고 가정 해 보겠습니다. 노드들의 이름은 각각 **node-1**, **node-2**, **node-3** 이고 node-1과 node-2의 클러스터명은 **es-cluster-1**, node-3의 클러스터명은 **es-cluster-2**로 실행을 합니다. 설정은 config/elasticsearch.yml 파일에서 아래와 같이 입력합니다. 각 탭에서 노드의 설정을 확인합니다.

node-1

```
cluster.name: es-cluster-1
node.name: "node-1"
```

node-2

```
cluster.name: es-cluster-1
```

```
node.name: "node-2"
```

node-3

```
cluster.name: es-cluster-2
node.name: "node-3"
```

또는 다음과 같이 실행 커맨드를 이용해서도 설정이 가능합니다.

node-1

```
$ bin/elasticsearch -Ecluster.name=es-cluster-1 -Enode.name=node-1
```

node-2

```
$ bin/elasticsearch -Ecluster.name=es-cluster-1 -Enode.name=node-2
```

node-3

```
$ bin/elasticsearch -Ecluster.name=es-cluster-2 -Enode.name=node-3
```

node-1과 node-2는 하나의 클러스터로 묶여있기 때문에 데이터 교환이 일어납니다. node-1로 입력된 데이터는 node-2 에서도 읽을 수 있으며 그 반대도 가능합니다. 하지만 node-3 은 클러스터가 다르기 때문에 node-1, node-2에 입력된 데이터를 node-3 에서 읽을 수는 없습니다.

가장 먼저 node-1을 실행시켰을 때 나타나는 화면입니다.

node-1 실행

```
$ bin/elasticsearch -Ecluster.name=es-cluster-1 -Enode.name=node-1
[2019-08-27T05:22:07,254][INFO ][o.e.e.NodeEnvironment    ] [node-1] usi
ng [1] data paths, mounts [[/ (/dev/disk1s1)]], net usable_space [88.2gb], ne
t total_space [465.6gb], types [apfs]
[2019-08-27T05:22:07,256][INFO ][o.e.e.NodeEnvironment    ] [node-1] he
ap size [989.8mb], compressed ordinary object pointers [true]
[2019-08-27T05:22:07,261][INFO ][o.e.n.Node               ] [node-1] node nam
e [node-1], node ID [RYhEbLLjQKaoHzGNkSNo-g], cluster name [es-cluster
```

```
-1]
...
```

위 실행 화면에서 먼저 노드명이 `[node-1]`인 것을 확인할 수 있습니다. 실행 메시지를 계속 확인 하다 보면 다음과 같은 부분이 있습니다.

node-1 실행 화면

```
[2019-08-27T05:22:14,030][INFO ][o.e.t.TransportService   ] [node-1] publish_address {127.0.0.1:9300}, bound_addresses {[::1]:9300}, {127.0.0.1:9300}
...
[2019-08-27T05:22:17,307][INFO ][o.e.h.AbstractHttpServerTransport] [node-1] publish_address {127.0.0.1:9200}, bound_addresses {[::1]:9200}, {127.0.0.1:9200}
```

`[o.e.t.TransportService]` 에서 tcp 포트 **9300**, 그리고 `[o.e.h.AbstractHttpServerTransport]` 에서 http 포트 **9200**을 각각 확인할 수 있습니다.

모든 클러스터에는 반드시 하나의 마스터 노드가 존재합니다. 실행 메시지의 다음 부분을 살펴보겠습니다.

node-1 실행 화면

```
[2019-08-27T05:22:17,230][INFO ][o.e.c.s.MasterService    ] [node-1] elected-as-master ([1] nodes joined)[{node-1}{RYhEbLLjQKaoHzGNkSNo-g}{KZxmWwFVTPKU5QHDbmULfg}{127.0.0.1}{127.0.0.1:9300}{dim}{ml.machine_memory=17179869184, xpack.installed=true, ml.max_open_jobs=20} elect leader, _BECOME_MASTER_TASK_, _FINISH_ELECTION_], term: 1, version: 1, reason: master node changed {previous [], current [{node-1}{RYhEbLLjQKaoHzGNkSNo-g}{KZxmWwFVTPKU5QHDbmULfg}{127.0.0.1}{127.0.0.1:9300}{dim}{ml.machine_memory=17179869184, xpack.installed=true, ml.max_open_jobs=20}]}
...
```

`[o.e.c.s.MasterService] [node-1] elected-as-master`부분에서 현재 node-1 이 마스터 노드로 선출 것을 확인할 수 있습니다. 마스터 노드에 대한 자세한 설명은 뒤에서 계속 다루도록 하겠습니다.

이제 node-1이 실행중인 상태에서 두번째 노드인 node-2를 실행시키면 다음과 같은 실행 메시지들을 확인할 수 있습니다.

node-2 실행

```
$ bin/elasticsearch -Ecluster.name=es-cluster-1 -Enode.name=node-2
...
[2019-08-27T07:50:54,308][INFO ][o.e.n.Node               ] [node-2] starting
...
[2019-08-27T07:50:54,479][INFO ][o.e.t.TransportService   ] [node-2] publish_address {127.0.0.1:9301}, bound_addresses {[::1]:9301}, {127.0.0.1:9301}
...
[2019-08-27T07:50:54,488][WARN ][o.e.b.BootstrapChecks    ] [node-2] the default discovery settings are unsuitable for production use; at least one of [discovery.seed_hosts, discovery.seed_providers, cluster.initial_master_nodes] must be configured
[2019-08-27T07:50:54,503][INFO ][o.e.c.c.ClusterBootstrapService] [node-2] no discovery configuration found, will perform best-effort cluster bootstrapping after [3s] unless existing master is discovered
[2019-08-27T07:50:54,778][INFO ][o.e.c.s.ClusterApplierService] [node-2] master node changed {previous [], current [{node-1}{RYhEbLLjQKaoHzGNkSNo-g}{KZxmWwFVTPKU5QHDbmULfg}{127.0.0.1}{127.0.0.1:9300}{dim}{ml.machine_memory=17179869184, ml.max_open_jobs=20, xpack.installed=true}]}, added {{node-1}{RYhEbLLjQKaoHzGNkSNo-g}{KZxmWwFVTPKU5QHDbmULfg}{127.0.0.1}{127.0.0.1:9300}{dim}{ml.machine_memory=17179869184, ml.max_open_jobs=20, xpack.installed=true},}, term: 1, version: 16, reason: ApplyCommitRequest{term=1, version=16, sourceNode={node-1}{RYhEbLLjQKaoHzGNkSNo-g}{KZxmWwFVTPKU5QHDbmULfg}{127.0.0.1}{127.0.0.1:9300}{dim}{ml.machine_memory=17179869184, ml.max_open_jobs=20, xpack.installed=true}}
...
[2019-08-27T07:50:55,216][INFO ][o.e.h.AbstractHttpServerTransport] [node-2] publish_address {127.0.0.1:9201}, bound_addresses {[::1]:9201}, {127.0.0.1:9201}
...
```

node-2의 경우 tcp 포트가 **9301**, http 포트가 **9201** 로 잡힌 것을 확인할 수 있습니다.

`[o.e.c.s.ClusterApplierService] [node-2] master node changed {previous [], current [{node-1} ... added {{node-1} ...` 부분을 보면 같은 클러스터에 이미 실행중인 마스터 노드 **node-1**이 있기 때문에 node-2 는 node-1이 마스터로 있는 클러스터에 묶인 것이 확인됩니다.

node-2 를 실행하고 난 뒤 다시 **node-1** 의 콘솔 메시지를 확인하면 마찬가지로 node-2 가 클러스터에 추가되었다는 메시지를 확인할 수 있습니다.

node-1 실행 화면

```
[2019-08-27T07:50:54,736][INFO ][o.e.c.s.MasterService    ] [node-1] node
-join[{node-2}{1EQ3a93iRMqppD49aQoTzg}{4CRm0xbHT36e38r2udKx0g}
{127.0.0.1}{127.0.0.1:9301}{dim}{ml.machine_memory=17179869184, ml.max
_open_jobs=20, xpack.installed=true} join existing leader], term: 1, version:
16, reason: added {{node-2}{1EQ3a93iRMqppD49aQoTzg}{4CRm0xbHT36
e38r2udKx0g}{127.0.0.1}{127.0.0.1:9301}{dim}{ml.machine_memory=171798
69184, ml.max_open_jobs=20, xpack.installed=true},}
[2019-08-27T07:50:55,200][INFO ][o.e.c.s.ClusterApplierService] [node-1]
added {{node-2}{1EQ3a93iRMqppD49aQoTzg}{4CRm0xbHT36e38r2udKx
0g}{127.0.0.1}{127.0.0.1:9301}{dim}{ml.machine_memory=17179869184, ml.
max_open_jobs=20, xpack.installed=true},}, term: 1, version: 16, reason: Pu
blication{term=1, version=16}
```

이제 node-1, node-2 가 실행 중인 상태에서 node-3 을 추가로 실행 해 보겠습니다.

node-3 실행 화면

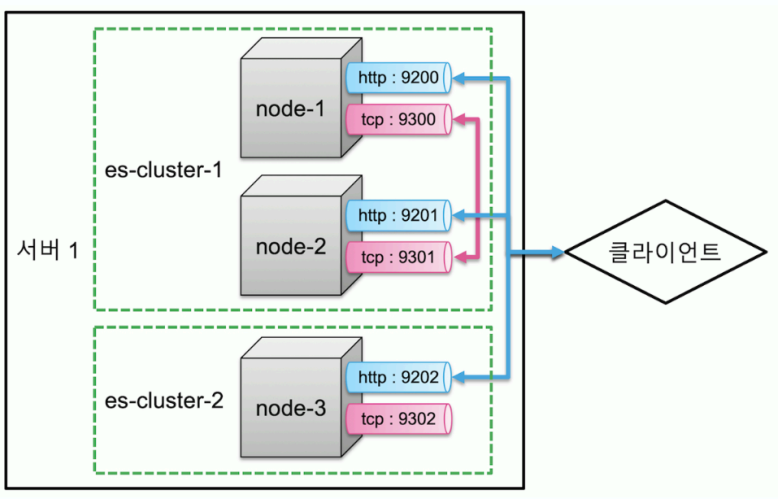
```
$ bin/elasticsearch -Ecluster.name=es-cluster-2 -Enode.name=node-3
...
[2019-08-27T08:01:37,474][INFO ][o.e.t.TransportService   ] [node-3] publi
sh_address {127.0.0.1:9302}, bound_addresses {[::1]:9302}, {127.0.0.1:9302}
...
[2019-08-27T08:01:37,640][WARN ][o.e.d.HandshakingTransportAddressC
onnector] [node-3] handshake failed for [connectToRemoteMasterNode[[::
1]:9300]]
...
[2019-08-27T08:01:40,671][INFO ][o.e.c.s.MasterService    ] [node-3] elect
ed-as-master ([1] nodes joined)[{node-3}{XPFkVAjKQfaVoWkd4Hqv5A}{8Y
3wZO41R_CmlMV-JJhoPg}{127.0.0.1}{127.0.0.1:9302}{dim}{ml.machine_me
mory=17179869184, xpack.installed=true, ml.max_open_jobs=20} elect lea
der, _BECOME_MASTER_TASK_, _FINISH_ELECTION_], term: 1, version: 1, re
```

```
ason: master node changed {previous [], current [{node-3}{XPFkVAjKQfaVoWkd4Hqv5A}{8Y3wZO41R_CmIMV-JJhoPg}{127.0.0.1}{127.0.0.1:9302}{dim}{ml.machine_memory=17179869184, xpack.installed=true, ml.max_open_jobs=20}]}
...
[2019-08-27T08:01:40,797][INFO ][o.e.h.AbstractHttpServerTransport] [node-3] publish_address {127.0.0.1:9202}, bound_addresses {[::1]:9202}, {127.0.0.1:9202}
```

먼저 node-3 의 http, tcp 포트 설정은 **9202**, **9302** 를 사용하도록 설정되었습니다.

`[o.e.d.HandshakingTransportAddressConnector] [node-3] ... handshake failed for ...`
부분을 보면 같은 서버에서 실행중인 node-1, node-2 를 찾았지만 클러스터명이 `es-cluster-2` 로 다르기 때문에 node-3은 node-1, node-2 와 같은 클러스터로 바인딩 되지 않았습니다. 그리고 `[o.e.c.s.MasterService ] [node-3] elected-as-master ...` 부분에서 **node-3** 스스로 `es-cluster-2` 클러스터의 마스터 노드로 선출 된 것을 확인할 수 있습니다.

지금까지 실행한 노드들의 클러스터 구조를 그림으로 나타내면 아래 그림과 같습니다.



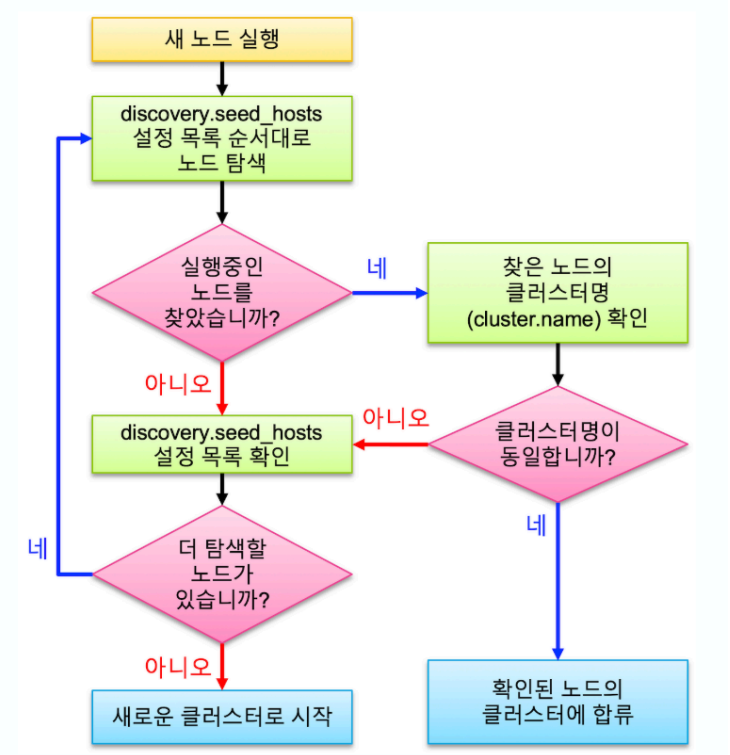
하나의 물리 서버에서 서로 다른 두 개의 클러스터 실행

디스커버리(Discovery)

노드가 처음 실행 될 때 같은 서버, 또는 `discovery.seed_hosts: [ ]` 에 설정된 네트워크 상의 다른 노드들을 찾아 하나의 클러스터로 바인딩 하는 과정을 **디스커버리** 라고 합니다. 디스커버리는 다음과 같은 순서로 이루어집니다.

1. `discovery.seed_hosts` 설정에 있는 주소 순서대로 노드가 있는지 여부를 확인
    - 노드가 존재하는 경우 > `cluster.name` 확인

    - 일치하는 경우 > 같은 클러스터로 바인딩 > 종료
    - 일치하지 않는 경우 > 1로 돌아가서 다음 주소 확인 반복
  - 노드가 존재하지 않는 경우 > 1로 돌아가서 다음 주소 확인 반복
2. 주소가 끝날 때 까지 노드를 찾지 못한 경우
  - 스스로 새로운 클러스터 시작



클러스터에 노드가 무수히 많아도 보통 `discovery.seed_hosts` 설정에는 처음에 탐색할 노드 3~5 개 정도만 설정 하면 큰 문제 없이 클러스터가 바인딩 됩니다. 보통은 마스터 후보 노드들을 지정하게 되며 처음 탐색하는 대상 노드는 반드시 먼저 가동중이어야 합니다.

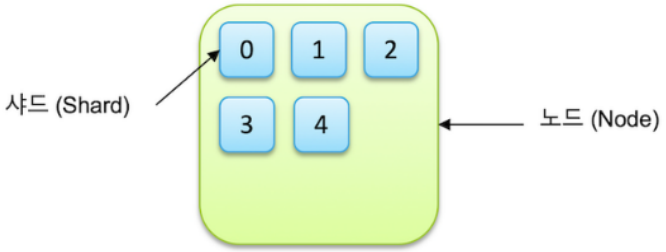
인덱스, 샤드, 모니터링

도큐먼트(document): Elasticsearch 에서의 단일 데이터 단위

인덱스(Index): 이 도큐먼트를 모아놓은 집합. RDB에서 테이블같은 개념.

💡 인덱스라는 단어가 여러 뜻으로 사용되기 때문에 데이터 저장 단위인 인덱스는 **인디시즈(indices)** 라고 표현하기도 한다.

여기서는 데이터를 Elasticsearch에 저장하는 행위는 **색인**, 그리고 도큐먼트의 집합 단위는 **인덱스** 라고 하겠다. 인덱스는 하나로 저장되는게 아니라 엘라스틱서치에서는 샤드라는 개념으로 쪼개서 저장된다.(여러 개의 노드에 흩어져서 저장됨)



하나의 인덱스가 5개의 샤드로 저장되도록 설정한 예

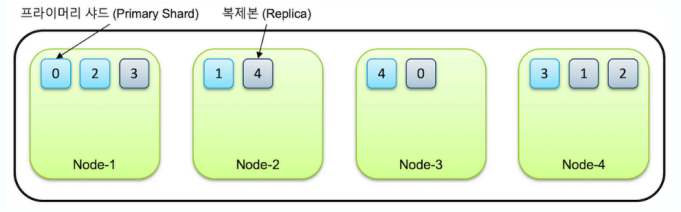
인덱스는 기본적으로 **샤드(shard)**라는 단위로 분리되고 각 노드에 분산되어 저장이 된다. 샤드는 루씬의 단일 검색 인스턴스이다.

Primary Shard 와 Replica(복제본)

인덱스를 생성할 때 별도의 설정을 하지 않으면 7 버전부터는 디폴트로 1개의 샤드로 인덱스가 구성된다.

클러스터에 노드를 추가하게 되면 샤드들이 각 노드들로 분산되고 디폴트로 1개의 복제본을 생성한다.

프라이머리 샤드(Primary Shard): 처음 생성된 샤드



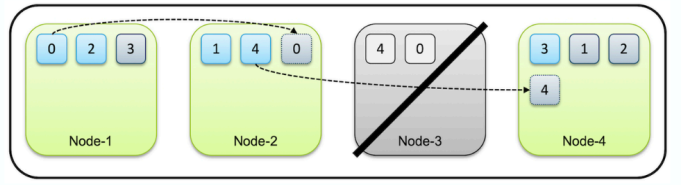
5개의 프라이머리 샤드와 복제본이 4개의 노드에 분산되어 저장된 예

예를 들어 한 인덱스가 5개의 샤드로 구성어 있고, 클러스터가 4개의 노드로 구성되어 있다고 가정하면 각각 5개의 프라이머리 샤드와 복제본, 총 10개의 샤드들이 전체 노드에 골고루 분배되어 저장된다.

노드가 1개만 있는 경우 프라이머리 샤드만 존재하고 복제본은 생성되지 않는. Elasticsearch 는 아무리 작은 클러스터라도 데이터 가용성과 무결성을 위해 최소 3개의 노드로 구성 할 것을 권장하고 있다.

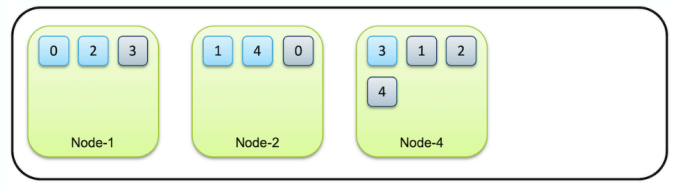
같은 샤드와 복제본은 동일한 데이터를 담고 있으며 **반드시 서로 다른 노드에 저장이 된다.**

만약에 위 그림에서 Node-3 노드가 시스템 다운이나 네트워크 단절등으로 사라지면 이 클러스터는 Node-3 에 있던 0번과 4번 샤드들을 유실하게 된다. 하지만 아직 다른 노드들 Node-1, Node-2 에 0번, 4번 샤드가 남아있으므로 여전히 전체 데이터는 유실이 없이 사용이 가능하다.



Node-3 노드가 유실되어 0번, 4번 샤드가 다른 노드에 복제본을 새로 생성한 예

처음에 클러스터는 먼저 유실된 노드가 복구 되기를 기다린다. 하지만 타임아웃이 지나 더 유실된 노드가 복구되지 않는다고 판단이 되면 Elasticsearch는 복제본이 사라져 1개만 남은 0번, 4번 샤드들의 복제를 시작한다. 처음에 4개였던 노드가 3개로 줄어도 복제가 끝나면 0~4번 까지의 프라이머리 샤드, 복제본이 각각 5개씩 총 10개의 데이터로 유지된다.



노드가 3개로 줄었을 때도 전체 데이터 유지

이렇게 프라이머리 샤드와 리플리카를 통해 Elasticsearch는 운영 중에 노드가 유실 되어도 데이터를 잃어버리지 않고 데이터의 가용성과 무결성을 보장한다.

> 💡 프라이머리 샤드가 유실된 경우에는 새로 프라이머리 샤드가 생성되는 것이 아니라, 남아있던 복제본이 먼저 프라이머리 샤드로 승격이 되고 다른 노드에 새로 복제본을 생성하게 된다.

인덱스의 Settings 설정에서 샤드 갯수 지정

샤드의 개수는 인덱스를 처음 생성할 때 지정할 수 있다. 프라이머리 샤드 수는 인덱스를 처음 생성할 때 지정하며, **인덱스를 재색인 하지 않는 이상 바꿀 수 없다**. 복제본의 개수는 나중에 변경이 가능하다. 아래는 curl 명령을 통해 REST API로 샤드가 5개, 복제본은 1개인 books 라는 이름의 인덱스를 생성하는 예제이다. REST API에 대해서는 다음 장에서 더 자세히 다루도록 하겠다.

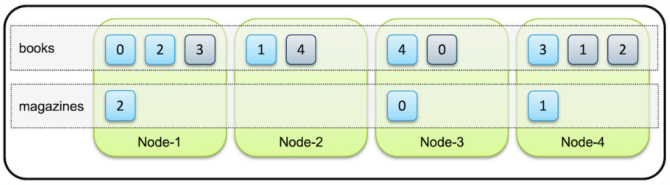
```
# 프라이머리 샤드 5, 복제본 1 인 books 인덱스 생성

$ curl -XPUT "http://localhost:9200/books" -H 'Content-Type: application/json' -d'
{
  "settings": {
    "number_of_shards": 5,
    "number_of_replicas": 1
  }
}'
```

books 인덱스의 복제본 수를 0으로 변경하려면 아래 명령으로 업데이트가 가능하다.

```
# books 인덱스의 복제본 개수를 0 으로 변경
```

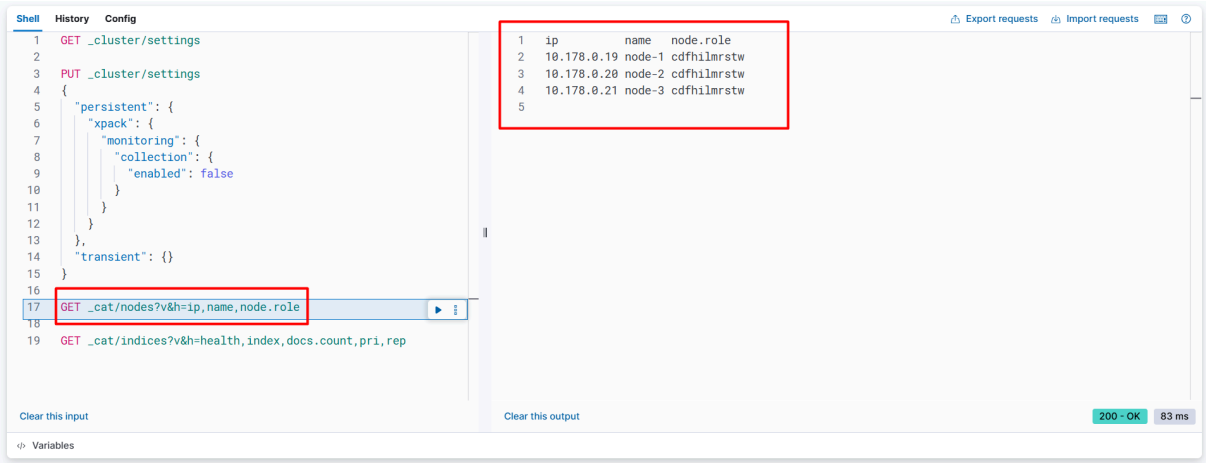
```
$ curl -XPUT "http://localhost:9200/books/_settings" -H 'Content-Type: ap
plication/json' -d'
{
  "number_of_replicas": 0
}'
```

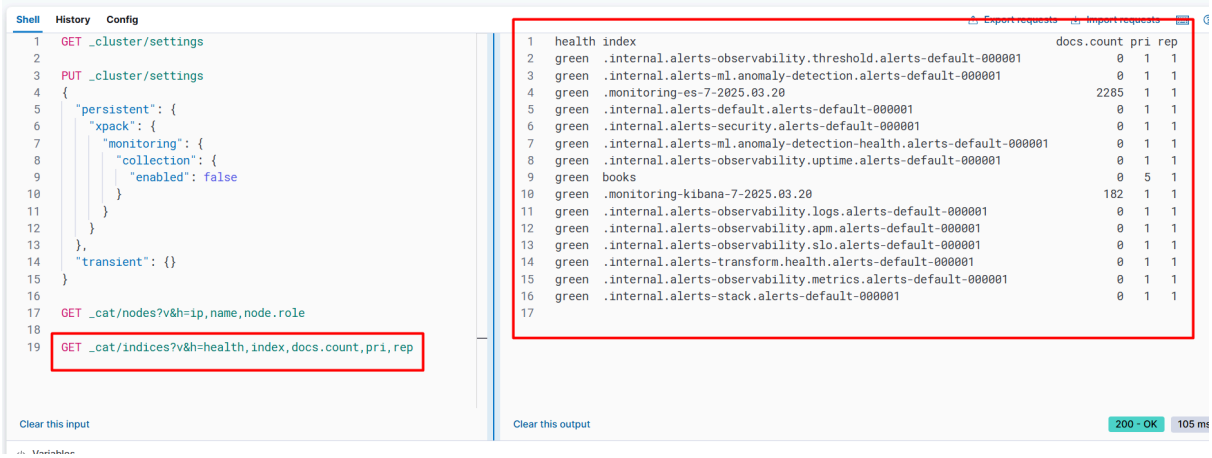
만약에 4개의 노드를 가진 클러스터에 프라이머리 샤드 5개, 복제본 1개인 books 인덱스, 그리고 프라이머리 샤드 3개 복제본 0개인 magazines 인덱스가 있다고 하면 전체 샤드들은 아래와 같은 모양으로 배치될 수 있다.



books 인덱스와 magazines 인덱스

_cat/shards API를 이용한 샤드 상태 조회





모니터링 도구를 이용한 클러스터 모니터링

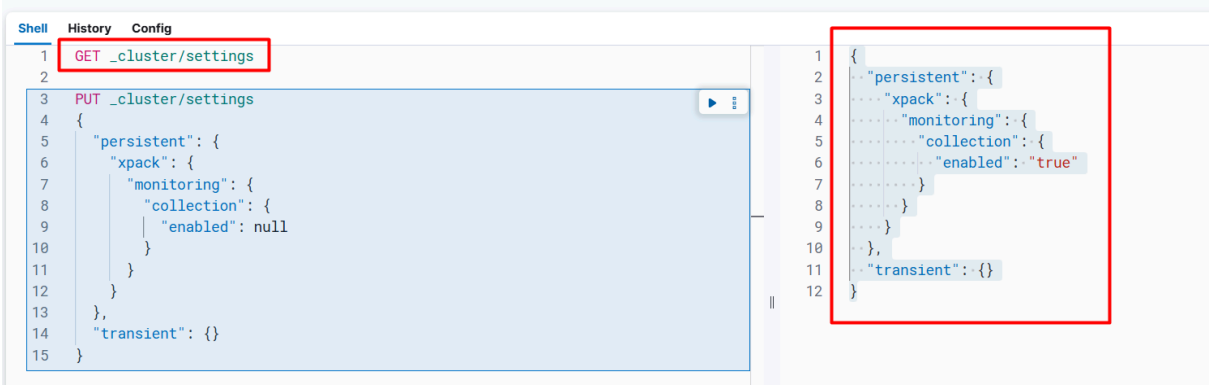
Kibana의 모니터링 도구 실행 및 확인

Management > Stack Monitoring 에서 확인할 수 있다

metricbeats 를 기본으로 권장한다. 하지만 일단은 설치도 따로 해야하고 설정도 해야하기 때문에 패스한다. 밑에 예전 방식으로 모니터링을 할 수 있다.

모니터링을 시작하면 엘라스틱이 수집한 모니터링 데이터를 엘라스틱 클러스터 내부에 저장한다. 그렇기에 모니터링 데이터 용량도 많이 나간다.

_cluster/settings API를 이용한 모니터링 실행/중지



kibana에 모니터링은 끄는 기능이 없다. elasticsearch 와 rest 통신을 하면서 주고 받기 때문에 kibana에서 없는 기능도 api 호출로 다른 기능을 사용할 수 있는 경우가 있다.

persistent 세팅은 클러스터를 전부 내렸다 올려도 살아있는 세팅이다.

transient 세팅은 내렸다 올리면 사라지는 세팅이다.

위 이미지에서 persistent의 collection 은 엘라스틱서치 노드들이 모니터링 데이터를 계속 스스로 수집을 하라는 설정이다. 이걸 api 로 끌 수 있다. (`false, null(세팅 자체가 사라짐)`)

kibana에 필요한 명령어(기능)이 없는 경우는 elasticsearch rest api 호출로 해결이 될 수 있다. 대부분의 기능은 elasticsearch 기반으로 작동한다.

마스터 노드와 데이터 노드