# Elastıc search

### Giriş

Elasticsearch ile başlamak için aşağıda detaylı bir yol haritası bulunmaktadır. Bu yol haritası, Elasticsearch'in temel konularını anlamanıza ve pratiğe geçmenize yardımcı olacaktır.

### 1. \*\*Temel Kavramları Anlama:\*\*

- Elasticsearch nedir? Nedir ve ne işe yarar?

- İndeks, Shard, Node, Cluster kavramlarına hakim olma.

- JSON tabanlı belgeleri anlama ve temel sorguları yazabilme.

### 2. \*\*Elasticsearch Kurulumu:\*\*

- [Resmi Elasticsearch İndirme Sayfası](https://www.elastic.co/downloads/elasticsearch) üzerinden Elasticsearch'in en son sürümünü indirme.

- Elasticsearch'in nasıl kurulacağını öğrenme (Windows, Linux, veya Docker üzerinde).

### 3. \*\*Temel Elasticsearch Operasyonları:\*\*

- Elasticsearch kümesini başlatma, durdurma ve yönetme.

- Index oluşturma, güncelleme, silme işlemlerini öğrenme.

- Basit sorgular yazma ve sonuçları inceleme.

### 4. \*\*Mapping ve Index Yapısını Anlama:\*\*

- Mapping kavramını anlama ve belirli bir mapping yapısını tanımlama.

- Index template'leri ve dynamic mapping konularını öğrenme.

### 5. \*\*Query DSL ve Sorgu Optimizasyonu:\*\*

- Query DSL (Domain Specific Language) kullanarak karmaşık sorgular yazma.

- Sorgu optimizasyon tekniklerini anlama.

### 6. \*\*Indexing ve Analiz:\*\*

- Belge indeksleme süreçlerini anlama.

- Analiz (Analysis) konseptini kavrama ve analiz süreçlerini özelleştirme.

### 7. \*\*Gelişmiş Arama ve Filtreleme:\*\*

- Full-text arama tekniklerini öğrenme.

- Aggregation (Toplama) ve filtering konularını anlama.

### 8. \*\*Elasticsearch Kümeleri ve Güvenlik:\*\*

- Birden fazla Elasticsearch node'unu birleştirerek bir küme oluşturma.

- Temel güvenlik önlemlerini alma (SSL/TLS, kullanıcı yönetimi).

### 9. \*\*Loglama ve Monitörleme:\*\*

- Elasticsearch loglarına erişme ve anlama.

- Elasticsearch kümesini monitörleme yöntemlerini öğrenme.

### 10. \*\*Elasticsearch Ekosistemi ile Tanışma:\*\*

- Kibana, Logstash, Beats gibi Elastic Stack bileşenleri ile tanışma.

- Elasticsearch ile birlikte kullanılan diğer araçları öğrenme.

### 11. \*\*Pratik Projeler ve Senaryolar:\*\*

- Gerçek dünya senaryolarında Elasticsearch kullanarak uygulama geliştirme.

- Örnek projeler ve kullanım senaryoları üzerinde çalışma.

### 12. \*\*Elasticsearch Topluluğuna Katılma:\*\*

- Elasticsearch forumları, bloglar ve sosyal medya üzerindeki toplulukları takip etme.

- Sorularınızı sorma ve deneyimlerinizi paylaşma.

### Kaynaklar:

- [Elasticsearch Resmi Belge Sayfası](https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html)

- [Elasticsearch - Getting Started](https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html)

- [Elasticsearch - Udemy Kursu](https://www.udemy.com/course/elasticsearch-complete-guide/)

Bu yol haritası, Elasticsearch ile ilgili temel konulara hakim olmanıza ve gerçek dünyada kullanılabilecek bilgileri öğrenmenize yardımcı olacaktır. Başarılar dilerim!

### veri yönetimi(data management)

#### Define an Index that satisfies a given set of requirements

Elasticsearch, bir açık kaynaklı bir arama ve analiz motoru olarak bilinir ve genellikle büyük miktarda veri üzerinde hızlı arama, analiz ve görselleştirme işlemleri için kullanılır. Elasticsearch'te veriler belirli bir yapıda, belirli bir endeks içinde saklanır. Bir endeks, benzer türdeki belgelerin toplandığı ve bu belgelerin hızlı ve etkili bir şekilde aranabilmesini sağlayan bir veri deposudur.

"Define an index that satisfies a given set of requirements" ifadesi, belirli bir takım gereksinimleri karşılayan bir endeks oluşturulması anlamına gelir. Bu gereksinimler genellikle veri modeli, arama performansı, analiz ihtiyaçları ve diğer iş gereksinimlerini içerir. İşte bu süreci anlatan birkaç adım:

1. \*\*Veri Modelini Belirleme:\*\*

- Hangi türde verileri depolamak istediğinizi belirleyin. Her belge tipi için hangi alanları (fields) içereceğinizi düşünün.

2. \*\*Endeks Ayarlarını Belirleme:\*\*

- Endeksin adı, ayarları ve özellikleri gibi temel bilgileri tanımlayın.

- Belirli bir dil ve analiz ayarları belirleyin (örneğin, dil analizi, eşleme analizi).

3. \*\*Belge Türlerini Tanımlama (Mapping):\*\*

- Her belge tipi için belge haritalamasını (mapping) tanımlayın. Bu, belge türlerinin hangi alanlara sahip olduğunu ve her alanın veri tipini içerir.

4. \*\*Analiz ve Arama İhtiyaçlarını Tanımlama:\*\*

- Arama sorgularınıza uygun analiz ve dizinleme ayarlarını belirleyin. Özel analiz süreçleri, filtreleme veya eşleme özellikleri ekleyin.

5. \*\*Performans ve Ölçeklenebilirlik Gereksinimlerini Göz Önünde Bulundurma:\*\*

- Endeks boyutu, anlık talepler ve ölçeklenebilirlik ihtiyaçlarını düşünün. Belirli bir performans seviyesine ulaşmak için ilgili ölçeklenebilirlik stratejilerini uygulayın.

6. \*\*Güvenlik ve Erişim Kontrollerini Ayarlama:\*\*

- Endeks üzerindeki güvenlik gereksinimlerini belirleyin ve gerekirse kullanıcı erişim kontrollerini yapılandırın.

Bu adımlar, bir endeks oluştururken göz önünde bulundurmanız gereken temel konuları içerir. Elasticsearch, RESTful API üzerinden bu işlemleri gerçekleştirebileceğiniz bir dizi yöntem sağlar. Bu sayede, belirli bir kullanım senaryosuna ve iş gereksinimlerine uygun bir endeks oluşturabilirsiniz.

#### Define and use an index template for a given pattern that satisfies a given set of requirements

Elasticsearch'te dinamik şablonlar (dynamic templates), endeks oluşturulurken belgelerin alanlarının otomatik olarak eşleştirilmesini ve belirli gereksinimlere uygun bir yapı oluşturulmasını sağlayan güçlü bir araçtır. Dinamik şablonlar, belge alanlarının veri türleri, dizinleme seçenekleri ve analiz ayarları gibi özelliklerini belirlemek için kullanılır. Aşağıda, bir dinamik şablonun tanımlanması ve kullanılması için temel adımları içeren bir açıklama bulunmaktadır:

1. \*\*Dinamik Şablonun Tanımlanması:\*\*

- Dinamik bir şablon oluşturmak için, Elasticsearch'te uygun bir endeks içinde `\_template` API'sini kullanabilirsiniz.

- Bu şablon, belgelerin alanlarını eşleştirmek için bir dizi kural içerir. Her kural, bir alan adı deseni ve bu desenle eşleşen alanlar için belirlenmiş özellikleri içerir.

```json

PUT \_template/my\_dynamic\_template

{

"index\_patterns": ["\*"],

"mappings": {

"dynamic\_templates": [

{

"strings": {

"match\_mapping\_type": "string",

"mapping": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

}

}

},

{

"integers": {

"match\_mapping\_type": "long",

"mapping": {

"type": "integer"

}

}

}

// ... Diğer kurallar eklenebilir

]

}

}

```

Yukarıdaki örnek, dinamik bir şablon oluşturur ve bu şablon, bir endekste türü "string" olan alanları "text" olarak ve aynı zamanda "keyword" alt alanını ek olarak belirli bir eşleşme deseniyle indeksler. Ayrıca, türü "long" olan alanları "integer" olarak eşleştirir.

2. \*\*Dinamik Şablonun Kullanılması:\*\*

- Dinamik şablonlar, endeks oluşturulduğunda veya belirli bir tür eklendiğinde otomatik olarak uygulanır. Her belge eklenirken, şablona uygun kurala göre alanlar dinamik olarak eşleştirilir ve endekslenir.

Dinamik şablonlar, özellikle değişken ve büyük veri setlerinde alanların dinamik olarak tanımlanmasını kolaylaştırır. Bu, kullanıcıların el ile her alanı tanımlamak yerine genel kurallar belirleyerek esnek bir yapı oluşturmalarına olanak tanır.

#### Define and use a dynamic template that that satisfies a given set of requirements

Elasticsearch'te dinamik şablonlar, belgelerin endekslendiği sırada alanlarının otomatik olarak belirlenmesini sağlayan önemli bir özelliktir. Dinamik şablonlar, belirli bir set gereksinimi karşılayan esnek endeks yapıları oluşturmanıza yardımcı olur. Aşağıda, bu sürecin adım adım bir açıklamasını bulabilirsiniz:

### Adım 1: Elasticsearch'te Dinamik Şablon Oluşturma

Dinamik bir şablon oluşturmak için, `\_template` API'sini kullanabilirsiniz. Bu API, belirli bir endeks deseni için belge alanlarının eşleştirilmesini tanımlayan bir JSON belgesini kabul eder.

Örneğin, varsayalım ki bir endeks adının öneki "log-" olan endeksler oluşturuyorsunuz ve metin tipindeki alanları "text" olarak, sayısal tipindeki alanları "integer" olarak tanımlamak istiyorsunuz:

```json

PUT \_template/my\_dynamic\_template

{

"index\_patterns": ["log-\*"],

"mappings": {

"dynamic\_templates": [

{

"strings": {

"match\_mapping\_type": "string",

"mapping": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

}

}

}

}

},

{

"integers": {

"match\_mapping\_type": "long",

"mapping": {

"type": "integer"

}

}

}

// ... Diğer kurallar eklenebilir

]

}

}

```

Bu örnekte, "log-\*" desenine uyan her endeks için metin tipindeki alanlar "text" olarak, sayısal tipindeki alanlar ise "integer" olarak eşleştirilir. Ek kurallar ekleyerek farklı alan türleri için özel eşleştirmeler tanımlanabilir.

### Adım 2: Dinamik Şablonun Kullanılması

Dinamik şablonlar, belge eklerken veya endeks oluştururken otomatik olarak uygulanır. Örneğin, aşağıdaki gibi bir belge ekleyelim:

```json

POST log-2024.01.19/\_doc/1

{

"message": "An example log message",

"severity": 3,

"timestamp": "2024-01-19T12:30:00"

}

```

Bu belge eklenirken, dinamik şablon belirli kurallara göre uygulanır. Örneğin, "message" alanı metin tipinde olduğu için "text" olarak eşleştirilir ve aynı zamanda "keyword" alt alanı eklenir. "severity" alanı sayısal bir alan olduğu için "integer" olarak eşleştirilir.

Bu sayede, dinamik şablonlar kullanılarak belgelerinize uygun endeks yapıları otomatik olarak oluşturulmuş olur. Dinamik şablonlar, veri modelinizdeki değişikliklere hızlı bir şekilde adapte olmanıza ve endeksleme işlemlerini esnek bir şekilde yönetmenize yardımcı olur.

#### Define an Index Lifecycle Management policy for a time-series index

Elasticsearch Index Lifecycle Management (ILM), belirli bir endeks için yaşam döngüsünü yönetmek için kullanılan bir mekanizmadır. Time-series (zaman serisi) verileri için ILM politikaları, genellikle belirli bir süre boyunca güncel olan ve daha sonra dondurulup arşivlenen veri türlerini yönetmek için kullanılır. Aşağıda, bir zaman serisi endeksi için ILM politikası tanımlama sürecini adım adım açıklamaktadır:

### Adım 1: ILM Politikası Oluşturma

ILM politikasını oluşturmak için Elasticsearch'te bir HTTP isteği göndermek gerekir. Aşağıda basit bir örnek bulunmaktadır:

```json

PUT \_ilm/policy/my\_time\_series\_policy

{

"policy": {

"phases": {

"hot": {

"actions": {

"rollover": {

"max\_size": "50GB",

"max\_age": "30d"

}

}

},

"warm": {

"min\_age": "30d",

"actions": {

"allocate": {

"require": {

"data": "warm"

}

}

}

},

"cold": {

"min\_age": "90d",

"actions": {

"freeze": {}

}

},

"delete": {

"min\_age": "365d",

"actions": {

"delete": {}

}

}

}

}

}

```

Bu örnek, bir "my\_time\_series\_policy" adlı ILM politikası oluşturur ve aşağıdaki aşamalardan geçer:

- \*\*Hot Phase:\*\* Bu aşamada endeks rollover (yeniden adlandırma) işlemi gerçekleştirilir. Endeks boyutu 50GB'yi geçerse veya endeks 30 gün yaşını doldurursa, rollover işlemi başlatılır.

- \*\*Warm Phase:\*\* Bu aşamada endeks, 30 gün yaşını doldurduktan sonra warm (ısınma) aşamasına geçer. Veri, "warm" data nodlarına taşınır.

- \*\*Cold Phase:\*\* Bu aşama, endeks 90 gün yaşını doldurduğunda devreye girer. Endeks dondurulur ve daha fazla güncellenemez.

- \*\*Delete Phase:\*\* Bu aşama, endeks 365 gün yaşını doldurduğunda devreye girer. Endeks otomatik olarak silinir.

### Adım 2: ILM Politikasının Endekse Bağlanması

Oluşturulan ILM politikasını belirli bir endekse bağlamak için, endeksi oluştururken veya varolan bir endeksi güncellerken `index.lifecycle.name` ve `index.lifecycle.rollover\_alias` ayarlarını kullanmanız gerekir.

Örneğin, yeni bir endeks oluştururken:

```json

PUT my\_time\_series\_index-000001

{

"aliases": {

"my\_time\_series\_alias": {

"is\_write\_index": true

}

},

"settings": {

"index.lifecycle.name": "my\_time\_series\_policy",

"index.lifecycle.rollover\_alias": "my\_time\_series\_alias"

}

}

```

Bu örnekte, `my\_time\_series\_index-000001` adlı endeks oluşturulur ve `my\_time\_series\_policy` adlı ILM politikası ile ilişkilendirilir. Aynı zamanda, `my\_time\_series\_alias` adlı bir alias (takma ad) oluşturulur ve bu alias üzerinden endekse yazma işlemleri gerçekleştirilir.

Bu adımlar, Elasticsearch'te zaman serisi verilerini yönetmek için ILM kullanımını anlatmaktadır. Bu yöntem, zamanla değişen ve büyüyen veri setleri için etkili bir yaşam döngüsü yönetimi sağlar.

#### Define an index template that creates a new data stream

Elasticsearch'te, bir veri akışı (data stream) oluşturan bir endeks şablonu (index template) tanımlamak, belirli bir veri modeli veya kullanım durumu için esnek bir yapı oluşturmanıza olanak tanır. Veri akışları, genellikle zaman serisi verileri gibi dinamik ve sürekli olarak gelen verileri daha iyi yönetmek için kullanılır. Aşağıda, yeni bir veri akışı oluşturan bir endeks şablonunu tanımlama sürecini adım adım açıklarım:

### Adım 1: Index Template Oluşturma

Elasticsearch'te bir endeks şablonu oluşturmak için `\_template` API'sini kullanırız. Öncelikle, yeni bir endeks şablonu tanımlayalım:

```json

PUT \_index\_template/my\_data\_stream\_template

{

"index\_patterns": ["my\_data\_stream-\*"],

"template": {

"settings": {

"number\_of\_shards": 1,

"number\_of\_replicas": 1

},

"mappings": {

"properties": {

"timestamp": {

"type": "date"

},

"value": {

"type": "float"

},

// Diğer alanlar eklenebilir

}

}

}

}

```

Bu örnekte, `my\_data\_stream\_template` adlı bir endeks şablonu tanımlanıyor. Bu şablon, `my\_data\_stream-\*` desenine uyan tüm endeksleri kapsayacak şekilde tasarlanmıştır.

- `settings`: Endeks ayarları belirtilir. Örnekte, endeksin birincil shard sayısı (`number\_of\_shards`) ve replika sayısı (`number\_of\_replicas`) belirlenmiştir.

- `mappings`: Endeksin alanları ve bunların veri tipleri belirlenir. Bu örnekte, bir tarih (`timestamp`) ve bir ondalık sayı (`value`) alanı tanımlanmıştır.

### Adım 2: Data Stream Oluşturma

Şablonu tanımladıktan sonra, bu şablonu kullanarak bir veri akışı oluşturabilirsiniz. Veri akışı adı, endeks adının başına eklenen özel bir ön ek olarak kullanılır. Örneğin:

```json

PUT /my\_data\_stream-000001

{

"aliases": {

"my\_data\_stream\_alias": {

"is\_write\_index": true

}

}

}

```

Bu örnekte, `my\_data\_stream-000001` adlı bir veri akışı oluşturulmuştur ve `my\_data\_stream\_alias` adlı bir alias eklenmiştir. Alias, veri akışı üzerinden yazma işlemlerini yönlendirmek için kullanılır.

### Adım 3: Veri Akışını Kullanma

Artık `my\_data\_stream\_alias` adlı alias üzerinden veri akışına yazabilir ve bu veri akışını sorgulayabilirsiniz. Yeni veriler geldikçe Elasticsearch, otomatik olarak yeni endeksler oluşturur ve verileri bu endekslerde saklar. Böylece, esnek bir yapıda veri akışları yönetilmiş olur.

Bu adımlar, Elasticsearch'te bir veri akışı oluşturmak için index template kullanımını açıklar. Bu yaklaşım, özellikle zaman serisi verileri gibi sürekli gelen verilerle çalışırken esneklik sağlar ve verilerin dinamik olarak yönetilmesine olanak tanır.

### Küme yönetimi(Cluster Management)

#### Diagnose shard issues and repair a cluester’s health

Elasticsearch kümesinin sağlığını teşhis etmek ve olası shard sorunlarını çözmek, genellikle Elasticsearch'in sağlam bir şekilde çalışmasını sürdürebilmenin önemli bir parçasıdır. Aşağıda, shard sorunlarını teşhis etme ve kümenin sağlığını düzeltebilme adımlarını içeren detaylı bir açıklama bulunmaktadır:

### Shard Sorunlarını Teşhis Etme:

1. \*\*Cluster Sağlığı Kontrolü:\*\*

- Elasticsearch kümenizin genel sağlığını kontrol etmek için `\_cat/health` API'sini kullanabilirsiniz. Bu API, kümenin genel durumu, shard sayıları ve durumları hakkında bilgi sağlar.

```bash

GET \_cat/health?v

```

2. \*\*Shard Durumunu Kontrol Etme:\*\*

- `\_cat/shards` API'sini kullanarak her bir endeksteki shard'ların durumunu kontrol edebilirsiniz. Özellikle "UNASSIGNED" durumundaki shard'lar sorunlu olabilir.

```bash

GET \_cat/shards?v

```

3. \*\*Node Durumunu Kontrol Etme:\*\*

- Elasticsearch kümesindeki her bir node'un durumunu kontrol etmek için `\_cat/nodes` API'sini kullanabilirsiniz.

```bash

GET \_cat/nodes?v

```

4. \*\*Shard Ayrıntılarını Kontrol Etme:\*\*

- Shard'lar hakkında daha fazla ayrıntıya ihtiyacınız varsa, `\_cluster/allocation/explain` API'sini kullanabilirsiniz.

```bash

GET \_cluster/allocation/explain?pretty

```

### Shard Sorunlarını Çözme:

1. \*\*UNASSIGNED Shard'ları İnceleme:\*\*

- Eğer shard'lar "UNASSIGNED" durumdaysa, bu shard'ları yeniden atamak için `\_cluster/reroute` API'sini kullanabilirsiniz.

```bash

POST /\_cluster/reroute

{

"commands": [

{

"allocate": {

"index": "your\_index",

"shard": 0,

"node": "desired\_node",

"allow\_primary": true

}

}

]

}

```

2. \*\*Shard Yeniden Dengeleme:\*\*

- Eğer küme dengesizse ve shard'lar düzgün bir şekilde dağılmamışsa, küme dengesini sağlamak için `\_cluster/reroute` API'sini kullanabilirsiniz.

```bash

POST /\_cluster/reroute?retry\_failed=true

```

3. \*\*Node Sorunlarını İnceleme:\*\*

- Eğer belirli bir node'da sorun varsa, node'un durumunu kontrol edip, gerekirse sorunlu node'u kapatıp açabilir veya ilgili node üzerindeki shard'ları başka bir node'a taşıyabilirsiniz.

```bash

GET /\_cat/nodes?v

```

4. \*\*Küme Sağlığını İnceleme:\*\*

- Küme sağlığını daha ayrıntılı incelemek için `\_cluster/health` API'sini kullanabilirsiniz.

```bash

GET \_cluster/health?v

```

Bu adımlar, Elasticsearch kümesinin sağlığını teşhis etme ve shard sorunlarını çözme konusunda yardımcı olabilir. Unutmayın ki her durum özeldir ve sorunları çözmek için spesifik bilgiler gerekebilir. Elasticsearch kılavuzları ve topluluk forumları, karmaşık sorunlara çözüm bulmak için değerli kaynaklar olabilir.

#### Backup and restore a cluster and/or specific indices

Elasticsearch kümenizin veya belirli endekslerin yedeklenmesi ve geri yüklenmesi, veri kaybını önlemek ve veri bütünlüğünü sağlamak açısından önemli bir süreçtir. Bu işlemi gerçekleştirmek için Elasticsearch, Snapshot and Restore API'sini sağlar. Aşağıda, bir kümenin veya belirli endekslerin yedeklenmesi ve geri yüklenmesi için adım adım bir açıklama bulunmaktadır:

### Kümenin Yedeklenmesi:

1. \*\*Snapshot Repository Tanımlama:\*\*

- İlk adım, yedekleri depolamak için bir "snapshot repository" tanımlamaktır. Bu genellikle uzak bir depolama alanı olabilir (örneğin, AWS S3 veya bir dosya sistemi).

```bash

PUT \_snapshot/my\_backup\_repository

{

"type": "fs",

"settings": {

"location": "/path/to/backups"

}

}

```

2. \*\*Snapshot Oluşturma:\*\*

- Ardından, belirli bir snapshot (yedek) oluşturabilirsiniz. Aşağıdaki örnekte, "my\_backup\_repository" adlı depoya "my\_snapshot" adında bir snapshot oluşturulmaktadır.

```bash

PUT /\_snapshot/my\_backup\_repository/my\_snapshot

{

"indices": "index1,index2",

"ignore\_unavailable": true,

"include\_global\_state": false

}

```

- `indices`: Yedeklenmesi istenen endeksleri belirtir.

- `ignore\_unavailable`: Eğer endeksler mevcut değilse, bu ayarın `true` olması durumunda hata almazsınız.

- `include\_global\_state`: Kümenin genel durumunu da yedeklemek istiyorsanız `true` olarak ayarlanmalıdır.

3. \*\*Snapshot Durumunu Kontrol Etme:\*\*

- Oluşturulan snapshot'ın durumunu kontrol etmek için `\_snapshot` API'sini kullanabilirsiniz.

```bash

GET /\_snapshot/my\_backup\_repository/my\_snapshot

```

### Kümenin veya Belirli Endekslerin Geri Yüklenmesi:

1. \*\*Kümenin Kapatılması (Opsiyonel):\*\*

- Kümenin yedeği geri yüklenmeden önce kümenin kapatılması önerilir.

```bash

POST /\_cluster/voting\_config\_exclusions/\_acknowledge

```

2. \*\*Kümenin Geri Yüklenmesi:\*\*

- Yedeklenen bir snapshot'ı kümenize geri yüklemek için `\_snapshot/restore` API'sini kullanabilirsiniz.

```bash

POST /\_snapshot/my\_backup\_repository/my\_snapshot/\_restore

{

"indices": "index1,index2",

"ignore\_unavailable": true,

"include\_global\_state": false

}

```

- `indices`: Geri yüklenmesi istenen endeksleri belirtir.

- `ignore\_unavailable`: Eğer endeksler mevcut değilse, bu ayarın `true` olması durumunda hata almazsınız.

- `include\_global\_state`: Kümenin genel durumunu geri yüklemek istiyorsanız `true` olarak ayarlanmalıdır.

3. \*\*Kümenin Açılması (Opsiyonel):\*\*

- Kümenin geri yüklemesi tamamlandıktan sonra kümenin açılması önerilir.

```bash

POST /\_cluster/voting\_config\_exclusions/\_reset

```

Bu adımlar, Elasticsearch kümenizin veya belirli endekslerin yedeklenmesi ve geri yüklenmesi için temel bir rehber sunar. Yedekleme ve geri yükleme işlemleri karmaşık olabilir, bu nedenle ilgili Elasticsearch belgelerini dikkatlice okumanız ve uygulama öncesi dikkatli bir plan yapmanız önemlidir.

#### Configure a snapshot to be searchable

Elasticsearch'te bir snapshot oluşturduktan sonra, bu snapshot'ı arama yetenekleriyle kullanabilmek için snapshot'ı "searchable" (arama yapılabilir) hale getirmeniz gerekebilir. Bu özellik, özellikle büyük veri setlerinde geçmiş bir durumu sorgulama veya arama ihtiyacı olduğunda kullanışlıdır. Elasticsearch'te bir snapshot'ı searchable hale getirmek için şu adımları takip edebilirsiniz:

### Adım 1: Snapshot Oluşturma

Önce, searchable bir snapshot oluşturmak için bir snapshot almanız gerekir. Snapshot oluşturma işlemi için `\_snapshot` API'sini kullanabilirsiniz.

Örneğin, `my\_backup\_repository` adlı bir depoya `my\_snapshot` adında bir snapshot oluşturmak için şu isteği kullanabilirsiniz:

```bash

PUT /\_snapshot/my\_backup\_repository/my\_snapshot

{

"indices": "index1,index2",

"ignore\_unavailable": true,

"include\_global\_state": false

}

```

- `indices`: Yedeklemek istediğiniz endeksleri belirtir.

- `ignore\_unavailable`: Eğer endeksler mevcut değilse, bu ayarı `true` olarak ayarlayabilirsiniz.

- `include\_global\_state`: Kümenin genel durumunu yedeklemek istiyorsanız, bu ayarı `true` olarak ayarlayabilirsiniz.

### Adım 2: Snapshot'ı Searchable Hale Getirme

Snapshot'ı searchable hale getirmek için, bu snapshot'ı restore ederken birkaç ek parametre eklememiz gerekiyor. Bu işlem, snapshot'ı geri yüklerken bir index ve index settings'ini de tanımlamayı içerir.

```bash

POST /\_snapshot/my\_backup\_repository/my\_snapshot/\_restore

{

"indices": "restored\_index",

"include\_global\_state": false,

"index\_settings": {

"index": {

"blocks": {

"read\_only": false

}

}

}

}

```

Bu örnekte, `my\_snapshot` adlı snapshot'ı kullanarak bir endeks olan `restored\_index` oluşturuluyor. `index\_settings` altında, bu endeksin "read\_only" bloğunu kaldırmak için bir ayar tanımlanıyor. Bu, snapshot'ı searchable hale getirir.

### Adım 3: Restore İşleminin Tamamlanması

Snapshot'ı searchable hale getirmek için yaptığınız restore işlemi tamamlandıktan sonra, bu endeksi kullanarak sorgular yapabilirsiniz.

```bash

GET /restored\_index/\_search

{

"query": {

"match\_all": {}

}

}

```

Bu sorgu, `restored\_index` adlı endeksi sorgular ve snapshot'ı searchable hale getirme işleminin başarıyla tamamlandığını kontrol etmenize yardımcı olabilir.

Unutmayın ki searchable snapshot'lar, orijinal verilerin bir anlık görüntüsünü temsil eder. Bu nedenle, bu snapshot'ın searchable hale getirilmiş hali, snapshot alındığı andaki veri durumunu yansıtacaktır.

#### Configure a cluster for cross-cluster search

Elasticsearch'te, çapraz küme arama (cross-cluster search) özelliği sayesinde farklı Elasticsearch küme ve endekslerini tek bir sorgu altında birleştirebilir ve arama yapabilirsiniz. Bu, farklı kümelere dağılmış verilere tek bir noktadan erişim sağlamak için kullanışlıdır. Çapraz küme arama özelliğini kullanabilmek için aşağıdaki adımları takip edebilirsiniz:

### Adım 1: Cross-Cluster Search İzinleri Ayarlama

Öncelikle, çapraz küme arama için gerekli izinleri ayarlamalısınız. Elasticsearch kullanıcılarının çapraz küme sorgularını yapabilmesi için gerekli izinlerin verilmiş olması gerekmektedir. Örnek bir rol ve kullanıcı oluşturabilirsiniz:

```bash

POST /\_security/role/cross\_cluster\_search\_role

{

"cluster": ["monitor"],

"indices": [

{

"names": ["\*"],

"privileges": ["read"]

}

]

}

POST /\_security/user/cross\_cluster\_search\_user

{

"password": "password",

"roles": ["cross\_cluster\_search\_role"]

}

```

Bu örnekte, `cross\_cluster\_search\_role` adlı bir rol oluşturuluyor ve bu role çapraz küme arama için gerekli `read` izinleri veriliyor. Daha sonra, bu role ait bir kullanıcı oluşturuluyor.

### Adım 2: Cross-Cluster Search Bağlantı Ayarları

Çapraz küme arama yapabilmeniz için, küme yapılandırma dosyanızda (örneğin, `elasticsearch.yml`) `remote.cluster\_name` ayarını eklemeniz gerekmektedir. Bu ayar, çapraz küme sorgularını yapmak istediğiniz hedef kümenin adını belirtir.

```yaml

cluster:

remote:

cluster\_one:

seeds:

- host: "remote\_cluster\_host"

port: 9300

```

Bu örnekte, `cluster\_one` adlı uzak kümenin konfigürasyonu yapılmıştır.

### Adım 3: Cross-Cluster Search Bağlantıyı Test Etme

Yapılandırmaları tamamladıktan sonra, çapraz küme arama bağlantısını test etmek için `\_remote/info` API'sini kullanabilirsiniz.

```bash

GET /\_remote/info

```

Bu sorgu, çapraz küme bağlantısının durumu ve konfigürasyonu hakkında bilgi sağlar.

### Adım 4: Cross-Cluster Search Sorgusu

Artık çapraz küme aramalarını yapabilirsiniz. Aşağıda basit bir örnek bulunmaktadır:

```bash

POST /index\_pattern/\_search

{

"query": {

"match": {

"field\_name": "search\_term"

}

}

}

```

Bu sorgu, `index\_pattern` adlı endeksler üzerinde `field\_name` alanında belirtilen `search\_term` değerini arar, ancak bu sorgu tüm çapraz küme endekslerini kapsar.

Çapraz küme arama, Elasticsearch küme yapılandırmanızı dikkatlice yapmayı ve güvenlik önlemlerini uygulamayı gerektirir. Özellikle güvenlik konularında dikkatli olmalısınız, çünkü farklı kümelere erişim verildiğinde bu durumu izlemek önemlidir.

#### Implement cross-cluester replication

Elasticsearch'te çapraz küme replikasyonu (cross-cluster replication - CCR), bir Elasticsearch kümesinden diğerine veri replikasyonu yapma yeteneğini ifade eder. Bu özellik, farklı coğrafi konumlardaki kümeler arasında veri replikasyonunu ve senkronizasyonunu sağlayarak yüksek erişilebilirlik ve felaket kurtarma senaryolarını destekler. Çapraz küme replikasyonunu uygulamak için aşağıdaki adımları takip edebilirsiniz:

### Adım 1: Çapraz Küme Replication Bağlantıları Ayarlama

Her iki küme arasında çapraz küme replikasyonunu kurabilmek için iki Elasticsearch kümesini yapılandırmanız gerekir. Bu, her iki küme arasında güvenli bir bağlantı oluşturmayı içerir.

Önce, her iki kümenin de `elasticsearch.yml` konfigürasyon dosyasına aşağıdaki gibi ayarları ekleyin:

```yaml

cluster:

remote:

cluster\_one:

seeds:

- host: "remote\_cluster\_host"

port: 9300

```

Bu örnekte, `cluster\_one` adlı uzak kümenin konfigürasyonu yapılmıştır. Aynı yapılandırmayı diğer küme için de gerçekleştirin.

### Adım 2: İndeks Replication Ayarlarını Belirleme

Replikasyon yapmak istediğiniz bir endeksi seçin ve bu endeks için replikasyon ayarlarını belirtin. Örneğin:

```bash

PUT /your\_index/\_ccr/follow

{

"remote\_cluster": "cluster\_one",

"leader\_index": "your\_index"

}

```

Bu, `your\_index` adlı endeksi `cluster\_one` adlı uzak küme üzerinden takip etmeye başlar.

### Adım 3: Replication Durumunu Kontrol Etme

Çapraz küme replikasyonunu başlatıp başlatmadığınızı kontrol etmek için aşağıdaki API'yi kullanabilirsiniz:

```bash

GET /your\_index/\_ccr/follow

```

Bu API, çapraz küme replikasyonunun durumu ve takip edilen endeks hakkında bilgi sağlar.

### Adım 4: Çapraz Küme Replication'ı Başlatma

Çapraz küme replikasyonunu başlatmak için, bir endeksi takip etme işlemini başlatmanız gerekir. Örneğin:

```bash

POST /your\_index/\_ccr/follow

```

Bu, belirtilen endeksin takip edilmeye başlandığını ve verilerin çapraz küme üzerinde replike edildiğini gösterir.

### Adım 5: Replication Durumunu İzleme

Replikasyon sürecini izlemek için aşağıdaki API'yi kullanabilirsiniz:

```bash

GET /your\_index/\_ccr/stats

```

Bu API, çapraz küme replikasyonunun istatistiklerini ve durumunu gösterir.

Çapraz küme replikasyonu, çeşitli kullanım senaryolarında (yüksek erişilebilirlik, felaket kurtarma vb.) kullanılan güçlü bir özelliktir. Ancak, çapraz küme replikasyonunu yapılandırırken ve kullanırken dikkatli olunmalıdır, çünkü yanlış yapılandırma veya kullanım, beklenmeyen sonuçlara neden olabilir. Dokümantasyonu dikkatlice okumak ve ihtiyacınıza uygun olarak yapılandırmak önemlidir.

#### Recover corrupted data

Elasticsearch kümesindeki bozuk (corrupted) verilerle başa çıkmak için birkaç farklı yaklaşım bulunmaktadır. Bu verileri kurtarmak için aşağıda birkaç genel yöntem açıklanmıştır:

### 1. Snapshot ve Restore Kullanma:

Elasticsearch'te düzenli olarak snapshot almak, olası veri kayıplarını önlemek için iyi bir uygulamadır. Eğer verileriniz bozulmuşsa ve daha önce bir snapshot almışsanız, bu snapshot'u kullanarak verilerinizi geri yükleyebilirsiniz.

- Önceki bir snapshot almak için:

```bash

PUT /\_snapshot/my\_backup\_repository/my\_snapshot

{

"indices": "your\_index",

"ignore\_unavailable": true,

"include\_global\_state": false

}

```

- Snapshot'tan geri yükleme işlemi:

```bash

POST /\_snapshot/my\_backup\_repository/my\_snapshot/\_restore

{

"indices": "your\_index",

"ignore\_unavailable": true,

"include\_global\_state": false

}

```

### 2. Index Recovery API Kullanma:

Eğer sadece belirli bir endeks bozulmuşsa, Index Recovery API'yi kullanarak endeksi kurtarabilirsiniz.

```bash

POST /your\_index/\_recovery

```

Bu, endeks üzerindeki shard'ları kurtarmak için bir talepte bulunur. Ancak, bu yöntem sadece endeksi kurtarır, bozuk belgeleri düzeltemez.

### 3. Log Larını İnceleme:

Elasticsearch, çalışma sırasında çeşitli logları kaydeder. Eğer bozuk verilerle ilgili daha fazla bilgiye ihtiyacınız varsa, Elasticsearch log dosyalarını inceleyebilirsiniz. Loglarda hata mesajları ve bozuk veri belirtileri olabilir.

### 4. Elasticsearch Checksum'larını Kullanma:

Elasticsearch, veritabanındaki her bir belge için bir checksum değeri hesaplar. Eğer bir belge bozulmuşsa, bu checksum değeri değişir. `checksum` alanını kullanarak bozuk belgeleri tespit edebilirsiniz.

```bash

GET /your\_index/\_search

{

"query": {

"bool": {

"must\_not": {

"checksum": {

"value": "previous\_checksum\_value"

}

}

}

}

}

```

Bu sorgu, belirli bir checksum değerine sahip olmayan belgeleri listeleyecektir.

Unutmayın ki bu yöntemlerin her biri, veri bozulması durumunu tamamen çözmeyebilir. Eğer bozuk veri kaynağı hala devam ediyorsa, bu sorunu ortadan kaldırmadan yeni veri oluşturulduğunda sorun devam edebilir. Veri bütünlüğünü sağlamak için, Elasticsearch kümesini dikkatlice izlemeniz ve gerektiğinde önleyici önlemler almanız önemlidir.

### Veri işleme(Data Processing)

#### Define a mapping that satisfies a given set of requirements

Elasticsearch'te bir endeks oluştururken veya var olan bir endekse bir belge eklerken, veri yapısını belirtmek için bir eşleme (mapping) tanımlamanız gerekmektedir. Bir eşleme, belirli bir endeksteki belgelerin alanlarının ve veri türlerinin nasıl tanımlandığını belirler. Bu, Elasticsearch'in belgeleri nasıl indeksleyeceği, sorgulayacağı ve arama yapılacağı konusunda önemli bir rol oynar. İşte bir eşleme tanımlama süreci ve temel kavramlar:

### Adım 1: Endeks Oluşturma

Önce, bir endeks oluşturmanız gerekir. Endeks, Elasticsearch'te belgeleri depolayan temel bir birimdir. Aşağıdaki örnekte, "my\_index" adlı bir endeks oluşturuyoruz:

```bash

PUT /my\_index

```

### Adım 2: Eşleme Tanımlama

Endeksi oluşturduktan sonra, bu endekse bir eşleme tanımlamanız gerekir. Eşleme, endeksteki belgelerin alanlarını ve veri tiplerini belirtir. Örneğin, "my\_index" endeksi için bir eşleme tanımlayalım:

```bash

PUT /my\_index/\_mapping

{

"properties": {

"name": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword"

}

}

},

"age": {

"type": "integer"

},

"is\_active": {

"type": "boolean"

},

"timestamp": {

"type": "date"

}

// Diğer alanlar eklenebilir

}

}

```

Yukarıdaki örnekte, "my\_index" endeksi için dört alan tanımladık:

- "name": Metin tipinde bir alan. Ayrıca, "keyword" adında bir alt alanı, tam eşleme aramalarda kullanılmak üzere tip olarak "keyword" olarak belirttik.

- "age": Tamsayı tipinde bir alan.

- "is\_active": Mantıksal (boolean) tipinde bir alan.

- "timestamp": Tarih ve saat tipinde bir alan.

### Eşleme Tanımında Kullanılan Bazı Temel Kavramlar:

- \*\*type\*\*: Alanın veri tipini belirtir (örneğin, text, keyword, integer, boolean, date).

- \*\*fields\*\*: İlave özellikler içeren bir alan. Örneğin, metin alanı için keyword alt alanını tanımlamak gibi.

- \*\*index\*\*: Bir alanın dizine eklenip eklenmeyeceğini belirtir. Örneğin, "timestamp" alanı için varsayılan olarak dizine eklenir.

- \*\*analyzer\*\*: Metin alanları için kullanılan dil analizini belirtir. Örneğin, metin arama sorgularında kullanılmak üzere analiz edilmiş bir metin alanı oluşturabilir.

Eşleme tanımını belirledikten sonra, belirli gereksinimleri karşılayacak şekilde uygun tür ve seçenekleri seçmelisiniz. Eşleme, belirli bir veri modelini veya kullanım durumunu karşılamak üzere esnek bir şekilde yapılandırılabilir.

#### Define and use a custom analizer that satisfies a given set of requirements

Elasticsearch'te özel bir analizör (analyzer) tanımlamak, metin verilerini dizine eklerken kullanılan dil analiz kurallarını belirlemenizi sağlar. Analizörler, belgelerin metin içeriğini parçalamak, normalleştirmek ve dizine eklemek için kullanılır. Özel bir analizör tanımlayarak, belirli gereksinimleri karşılamak veya özel metin işleme kuralları uygulamak mümkündür.

Aşağıda, özel bir analizör tanımlama ve kullanma adımları yer almaktadır:

### Adım 1: Özel Analizör Tanımlama

Öncelikle, bir analizör tanımlamanız gerekir. Aşağıdaki örnekte, "custom\_analyzer" adında bir analizör tanımlanmaktadır. Bu analizör, özel bir tokenizer ve filtre seti içermektedir.

```bash

PUT /\_analyze

{

"tokenizer": "standard",

"filter": ["lowercase", "asciifolding"],

"char\_filter": ["html\_strip"],

"text": "Bu, özel bir analizör örneğidir."

}

```

Bu örnekte:

- \*\*tokenizer\*\*: Metni parçalayan bir tokenizer belirlenir. "standard" tokenizer, birçok dilde kullanılan varsayılan bir tokenizer'dır.

\*\*filter\*\*: Parçalanan metni normalleştiren ve düzenleyen filtreler belirlenir. "lowercase" filtresi metinleri küçük harfe dönüştürür, "asciifolding" filtresi ise ASCII dışındaki karakterleri dönüştürür.

- \*\*char\_filter\*\*: Metin içinde özel karakterleri temizleyen karakter filtreleri belirlenir. "html\_strip" filtresi HTML etiketlerini temizler.

### Adım 2: Özel Analizörü Endekse Eklemek

Tanımladığınız özel analizörü, bir endeks veya alan için kullanmak istiyorsanız, endeksin veya alanın eşlemesinde bu analizörü belirtmeniz gerekir. Aşağıdaki örnekte, "custom\_analyzer" adındaki özel analizörün "my\_index" adlı bir endeks için kullanılması gösterilmektedir:

```bash

PUT /my\_index

{

"mappings": {

"properties": {

"content": {

"type": "text",

"analyzer": "custom\_analyzer"

}

}

}

}

```

Bu örnekte, "content" adlı bir metin alanı tanımlanmış ve bu alan için özel analizör olan "custom\_analyzer" kullanılmıştır.

### Adım 3: Özel Analizörü Kullanma

Artık belgeleri eklerken veya sorgularken bu özel analizörü kullanabilirsiniz. Örneğin, bir belge eklerken:

```bash

POST /my\_index/\_doc/1

{

"content": "Bu, özel analizörü kullanarak eklenen bir belgedir."

}

```

Veya bir sorgu yaparken:

```bash

GET /my\_index/\_search

{

"query": {

"match": {

"content": "örnek"

}

}

}

```

Bu sorgu, "content" alanındaki metinleri "custom\_analyzer" analizörü ile işleyecektir.

Özel analizörler, belirli dil ihtiyaçlarına, özel karakter temizleme gereksinimlerine veya benzeri durumlara göre özelleştirilebilir. Elasticsearch'in sağladığı bir dizi dahili analizör ve filtre bulunsa da, özel ihtiyaçlarınıza uyacak şekilde kendi analizörünüzü tanımlayabilirsiniz.

#### Define and use multi-fields with different data types and/or analyzers

Elasticsearch'te çoklu alanlar (multi-fields), bir alanın aynı veriyi farklı veri türleri veya analiz yöntemleriyle indekslemesine olanak tanıyan bir özelliktir. Bu, aynı alanın hem anahtar kelime (keyword) hem de tam metin (full-text) aramalara uygun hale getirilmesine veya aynı alanın farklı analizörlerle işlenmesine olanak tanır. Çoklu alanlar, genellikle bir alanın birden çok kullanım senaryosunu desteklemek için kullanılır.

Aşağıda, çoklu alanlar tanımlama ve kullanma adımları yer almaktadır:

### Adım 1: Çoklu Alanları Tanımlama

Çoklu alanları tanımlamak için eşleme (mapping) sırasında `fields` özelliğini kullanabilirsiniz. Aşağıdaki örnekte, "my\_field" adlı bir alanın hem anahtar kelime hem de tam metin indekslemesi için iki farklı alt alan tanımlanmaktadır:

```bash

PUT /my\_index

{

"mappings": {

"properties": {

"my\_field": {

"type": "text",

"fields": {

"keyword": {

"type": "keyword",

"ignore\_above": 256

},

"fulltext": {

"type": "text",

"analyzer": "standard"

}

}

}

}

}

}

```

Bu örnekte:

- "my\_field" adlı ana alan bir metin alanıdır (`type: "text"`).

- "keyword" adlı alt alan, tam metin aramalara uygun olmayan, sadece anahtar kelime tabanlı aramalara uygun bir `keyword` tipinde alandır.

- "fulltext" adlı diğer alt alan, tam metin aramalara uygun olacak şekilde "standard" analizörü ile işlenen bir metin alanıdır.

### Adım 2: Veri Ekleme

Belirtilen çoklu alanlar tanımlandıktan sonra, belgeler eklerken her iki alt alanı da doldurabilirsiniz:

```bash

POST /my\_index/\_doc/1

{

"my\_field": "Bu, çoklu alanlarla çalışan bir örnektir."

}

```

### Adım 3: Sorgu Yapma

Her iki alt alanı da sorgularınızda kullanabilirsiniz. Örneğin:

```bash

GET /my\_index/\_search

{

"query": {

"match": {

"my\_field.keyword": "örnek"

}

}

}

```

Bu sorgu, "my\_field.keyword" adlı alt alanı kullanarak "örnek" kelimesini içeren belgeleri arayacaktır.

### Çoklu Alanlarda Dikkat Edilmesi Gerekenler:

- Çoklu alanlar, endeksleme ve depolama maliyetini artırabilir. Bu nedenle, sadece ihtiyaç duyulan durumlarda kullanılmalıdır.

- Çoklu alanlar, belirli bir alanın farklı kullanım senaryolarını desteklemek için özelleştirilmiş analizörlerle birleştirilebilir.

Çoklu alanlar, Elasticsearch'te esnek ve özelleştirilebilir bir veri modeli oluşturmanıza yardımcı olabilir. İhtiyacınıza uygun analizörleri ve veri tiplerini kullanarak, belgelerinizi daha etkili bir şekilde indeksleyebilir ve arayabilirsiniz.

#### Use the reindex API and Update By Query API to reindex and/or update documents

Elasticsearch'te, belirli durumlar için belgeleri tekrar indekslemek veya güncellemek amacıyla Reindex API ve Update By Query API kullanılır. İşte her iki API'nın kullanımı hakkında detaylı bilgi:

### Reindex API:

Reindex API, bir endeksten diğerine belgeleri kopyalamak için kullanılır. Bu, endeks şemasını değiştirmek veya verileri temizlemek gibi senaryolarda oldukça kullanışlıdır.

#### Temel Kullanım:

Aşağıda, bir endeksten diğerine belgeleri kopyalamak için temel Reindex API kullanımını gösteren bir örnek bulunmaktadır:

```bash

POST \_reindex

{

"source": {

"index": "eski\_endeks"

},

"dest": {

"index": "yeni\_endeks"

}

}

```

Bu örnekte, "eski\_endeks" adlı bir endeksten belgeleri alır ve bunları "yeni\_endeks" adlı başka bir endekse kopyalar.

### Update By Query API:

Update By Query API, belirli bir sorguya uyan belgeleri güncellemek için kullanılır. Bu API, belirli bir koşulu sağlayan belgeleri seçer ve bu belgelere bir dizi güncelleme uygular.

#### Temel Kullanım:

Aşağıda, Update By Query API'nın temel kullanımını gösteren bir örnek bulunmaktadır:

```bash

POST eski\_endeks/\_update\_by\_query

{

"script": {

"source": "ctx.\_source.my\_field = 'updated\_value'",

"lang": "painless"

},

"query": {

"match": {

"my\_field.keyword": "eski\_değer"

}

}

}

```

Bu örnekte, "eski\_endeks" adlı endeksteki belgeleri seçer ve "my\_field" adlı alanı "eski\_değer" olan belgeleri bulur. Daha sonra, bu belgelerin "my\_field" alanını "updated\_value" ile günceller.

### Güvenlik Uyarısı:

Update By Query API, belgeleri güncellemek için tasarlanmış olsa da, büyük veri setlerinde performans sorunlarına neden olabilir ve dikkatlice kullanılmalıdır. Her iki API'yi de dikkatlice kullanmalısınız ve uygulamanızın gereksinimlerine uygun olarak yapılandırmalısınız. Ayrıca, Elasticsearch sürümünüzde mevcut olan güncel belgeleri inceleyerek ve belgelere zarar vermeden önce yedekleme yaparak bu API'ları kullanmalısınız.

Bu API'leri kullanmadan önce belgelerinizin yedeğini almak, güvenlik ve geri dönüşü kolaylaştırmak adına iyi bir uygulamadır.

#### Define and use an angest pipeline that satisfies a given set of requirements including the use of painless to modify documents

İngest pipeline, Elasticsearch'te belgelerin indekslenmeden önce işlemlerden geçirilmesini sağlayan bir mekanizmadır. Bu, belgelerin analiz edilmesi, dönüştürülmesi veya zenginleştirilmesi gibi işlemleri içerebilir. İşte bir ingest pipeline tanımlama ve kullanma adımları:

### Adım 1: İngest Pipeline Tanımlama

İngest pipeline tanımlamak için aşağıdaki gibi bir API çağrısı yapabilirsiniz:

```bash

PUT /\_ingest/pipeline/my\_pipeline

{

"description" : "My custom pipeline",

"processors" : [

{

"set" : {

"field": "new\_field",

"value": "default\_value"

}

},

{

"grok" : {

"field": "log\_message",

"patterns": ["%{COMBINEDAPACHELOG}"]

}

},

{

"uppercase" : {

"field" : "new\_field"

}

}

]

}

```

Yukarıdaki örnekte, "my\_pipeline" adında bir ingest pipeline tanımlanmıştır. Bu pipeline, belgeler üzerinde üç işlem gerçekleştirir:

1. `set`: Yeni bir alan olan "new\_field" oluşturur ve değerini "default\_value" olarak ayarlar.

2. `grok`: "log\_message" alanındaki metni Apache log formatına göre analiz eder.

3. `uppercase`: "new\_field" alanındaki metni büyük harfe çevirir.

### Adım 2: İngest Pipeline'ı Kullanma

Şimdi, bu ingest pipeline'ını kullanarak belgeleri indeksleyebilirsiniz. İngest pipeline'ını bir indeksle ilişkilendirmek için `index` parametresini kullanabilirsiniz:

```bash

POST /my\_index/\_doc?pipeline=my\_pipeline

{

"log\_message": "192.168.1.1 - - [22/Mar/2022:10:00:00 +0000] \"GET /index.html\" 200 1234"

}

```

Bu örnekte, "my\_index" adlı bir endekse bir belge eklenirken, "my\_pipeline" adlı ingest pipeline'ı kullanılır.

İngest pipeline'ları, belgeleri indekslemeden önce verileri dönüştürmek veya zenginleştirmek için güçlü bir araçtır. Painless (Elasticsearch'te kullanılan gömülü bir dil) gibi ifade dilleri ile çeşitli işlemler gerçekleştirebilirsiniz. Tanımladığınız pipeline'lar, belirli veri düzenleme gereksinimlerinizi karşılamak üzere özelleştirilebilir.

#### Define runtime fileds to retrieve custom values using painless scripting

Elasticsearch'te runtime alanlar (runtime fields), belgelerin indekslenmesi sırasında değil, sorgulama sırasında dinamik olarak hesaplanan ve alınan değerleri içeren özel alanlardır. Runtime alanlar, özellikle belirli bir sorgulama ihtiyacını karşılamak veya belge içeriğini dönüştürmek için kullanışlıdır. Painless, Elasticsearch'te kullanılan gömülü bir dil olduğu için, runtime alanlarını tanımlamak için Painless dilini kullanabilirsiniz.

Aşağıda, bir runtime alanı tanımlama adımlarını içeren bir örnek bulunmaktadır:

### Adım 1: Runtime Alanı Tanımlama

Runtime alanlarını tanımlamak için, bir endeks üzerinde bir mapping tanımlamanız gerekir. Aşağıdaki örnekte, "my\_index" adlı bir endekse bir runtime alanı ekliyoruz:

```bash

PUT /my\_index

{

"mappings": {

"runtime": {

"my\_runtime\_field": {

"type": "keyword",

"script": {

"source": "emit(doc['existing\_field'].value.toUpperCase())"

}

}

}

}

}

```

Bu örnekte, "my\_index" adlı endeks için "my\_runtime\_field" adlı bir runtime alanı tanımlanmıştır. Bu alan, "existing\_field" adlı bir alanın değerini büyük harfe çevirir.

### Adım 2: Sorgu Yapma

Tanımladığınız runtime alanını kullanmak için bir sorgu yapabilirsiniz. Aşağıdaki örnekte, "my\_runtime\_field" adlı runtime alanını kullanarak belgeleri sorguluyoruz:

```bash

GET /my\_index/\_search

{

"query": {

"match": {

"my\_runtime\_field": "MERGED\_VALUE"

}

}

}

```

Bu sorgu, "my\_runtime\_field" adlı runtime alanındaki değeri kullanarak belgeleri filtreler.

### Dikkat Edilmesi Gerekenler:

1. Runtime alanları, sorgu sırasında hesaplandığı için, belgeleri daha hızlı almak için önceden hesaplanmış ve indekslenmiş alanlar kadar performanslı olmayabilir.

2. Painless dilini kullanırken güvenlik önlemlerini göz önünde bulundurmalısınız. Scripting güvenliği, Elasticsearch konfigürasyonunda kontrol edilebilir.

3. Runtime alanlarının kullanımı, belirli bir ihtiyaca yönelik olarak dikkatlice planlanmalıdır, aksi takdirde gereksiz karmaşıklıklara neden olabilir.

Runtime alanları, belirli bir endeksi sorgularken dinamik olarak değerler oluşturmak ve kullanmak için güçlü bir araçtır. Bu, özellikle belirli bir sorgu senaryosuna özgü ek bilgiler eklemek veya belge içeriğini dönüştürmek istediğiniz durumlarda kullanışlıdır.

### Veri Arama(Searching Data)

#### Write and execute a search query for terms and/or phrases şn one or more fields of an index

Elasticsearch'te arama sorguları, belirli bir endeks üzerindeki bir veya daha fazla alanı içeren belgeleri bulmak için kullanılır. Sorgular, basit kelime eşleştirmelerinden karmaşık filtreleme ve analiz işlemlerine kadar bir dizi senaryoyu destekler. İşte bir terim ve/veya ifade arama sorgusu oluşturma ve yürütme adımları:

### Adım 1: Basit Terim Arama Sorgusu

Basit bir terim arama sorgusu oluşturmak için, `match` veya `term` sorgu türlerinden birini kullanabilirsiniz. Örneğin, bir "my\_field" adlı bir alan içinde "search\_term" terimini arayan bir sorgu şu şekilde olabilir:

```bash

GET /my\_index/\_search

{

"query": {

"match": {

"my\_field": "search\_term"

}

}

}

```

Bu sorgu, "my\_index" adlı endeks içindeki belgeler arasında "my\_field" alanında "search\_term" terimini arar.

### Adım 2: Frazeleri İçeren Terim Arama

Eğer belirli bir kelime öbeği veya ifadeyi içeren belgeleri arıyorsanız, `match\_phrase` veya `match\_phrase\_prefix` sorgu türlerini kullanabilirsiniz. Örneğin:

```bash

GET /my\_index/\_search

{

"query": {

"match\_phrase": {

"my\_field": "search phrase"

}

}

}

```

Bu sorgu, "my\_index" adlı endeks içindeki belgeler arasında "my\_field" alanında "search phrase" ifadesini içeren belgeleri arar.

### Adım 3: Birden Fazla Alanı İçeren Terim Arama

Eğer birden fazla alanı içeren belgeleri aramak istiyorsanız, `multi\_match` sorgu türünü kullanabilirsiniz. Örneğin:

```bash

GET /my\_index/\_search

{

"query": {

"multi\_match": {

"query": "search\_term",

"fields": ["field1", "field2"]

}

}

}

```

Bu sorgu, "my\_index" adlı endeks içindeki belgeler arasında "field1" veya "field2" alanlarında "search\_term" terimini arar.

### Adım 4: Analiz Edilmemiş Terim Arama

Eğer bir alanın içindeki terimi analiz etmeden aramak istiyorsanız, `term` sorgu türünü kullanabilirsiniz. Analiz edilmemiş terim arama şu şekildedir:

```bash

GET /my\_index/\_search

{

"query": {

"term": {

"my\_field.keyword": "search\_term"

}

}

}

```

Bu sorgu, "my\_index" adlı endeks içindeki belgeler arasında analiz edilmemiş "my\_field.keyword" alanında "search\_term" terimini arar.

Bu örnekler, Elasticsearch'te basit terim ve ifade arama sorgularını nasıl oluşturacağınızı göstermektedir. Gerçek senaryolarınıza uyacak şekilde sorguları özelleştirebilir ve belirli arama ihtiyaçlarınıza göre ayarlayabilirsiniz.

#### Write an execute a search query that is a boolean combination of multiple queries and filters

Elasticsearch'te boolean kombinasyonları kullanarak birden çok sorgu ve filtreleme koşulunu birleştirmek mümkündür. Bu, karmaşık arama senaryolarını yönetmek için kullanışlıdır. Elasticsearch, `bool` sorgu türünü kullanarak boolean kombinasyonları oluşturmanıza izin verir. İşte birkaç örnek:

### Örnek 1: Must, Should, ve Must Not Kullanımı

Bu örnekte, `must`, `should`, ve `must\_not` kısıtlamalarını kullanarak birden çok koşulu birleştiren bir sorgu örneği gösterilmektedir:

```bash

GET /my\_index/\_search

{

"query": {

"bool": {

"must": [

{ "match": { "field1": "value1" }},

{ "range": { "field2": { "gte": 10, "lte": 20 }}}

],

"should": [

{ "term": { "field3": "value3" }},

{ "term": { "field4": "value4" }}

],

"must\_not": [

{ "exists": { "field": "field5" }}

]

}

}

}

```

Bu sorgu, aşağıdaki koşulları içermektedir:

- `field1` alanında "value1" terimini içeren belgeler.

- `field2` alanında 10 ile 20 arasında bir değere sahip belgeler.

- `field3` veya `field4` alanlarında "value3" veya "value4" terimini içeren belgeler (en az biri).

- `field5` alanı olmayan belgeler.

### Örnek 2: Bool Sorgularını Gömülü Olarak Kullanma

Bool sorgularını gömülü olarak kullanabilir ve karmaşık sorguları basitleştirebilirsiniz. Örneğin:

```bash

GET /my\_index/\_search

{

"query": {

"bool": {

"must": [

{ "match": { "field1": "value1" }},

{

"bool": {

"should": [

{ "term": { "field2": "value2" }},

{ "range": { "field3": { "gte": 5 }}}

]

}

}

]

}

}

}

```

Bu örnekte, `field1` alanında "value1" terimini içeren belgeler ve ya `field2` alanında "value2" terimini içeren veya `field3` alanı 5'ten büyük olan belgeleri içeren bir sorgu örneği bulunmaktadır.

Bool sorguları, birden çok koşulu birleştirme ve istediğiniz karmaşıklıkta aramalar gerçekleştirme yeteneği sunar. Bu örnekler, temel bool sorgularını anlamanıza yardımcı olabilir ve özelleştirmeler için başlangıç noktası sağlayabilir.

#### Write an asynchronous search

Elasticsearch 7.7.0 sürümü itibariyle, Asenkron Arama (Asynchronous Search) özelliği kullanıma sunulmuştur. Bu özellik, büyük veri setleri üzerinde gerçekleştirilen uzun süreli sorgulamaların, arama sonuçlarının alındığı bir araçtır. Asenkron arama, geleneksel senkron sorgulamalardan farklı olarak, bir arama sorgusunu gönderdikten sonra sonuçları almak için bir süre beklemenize gerek olmadan, sorgunun çalıştığı arka planda işlemesi anlamına gelir.

İşte Elasticsearch'te asenkron arama kullanmanın temel adımları:

### Adım 1: Asenkron Arama Sorgusu Gönderme

Asenkron arama sorgusu göndermek için `\_search` endpoint'ine bir POST isteği gönderilir ve `"size"` parametresiyle sonuçların ne kadarının alınacağı belirlenir. `"size"` parametresi, bir sayı veya `"s"` (small), `"m"` (medium), `"l"` (large) gibi önceden belirlenmiş boyutlardan biri olabilir.

```bash

POST /my\_index/\_search

{

"size": "l",

"query": {

"match": {

"field": "value"

}

}

}

```

Bu örnekte, `"size"` parametresi `"l"` (large) olarak ayarlanmıştır, bu da büyük bir sonuç kümesi beklediğimizi gösterir.

### Adım 2: Asenkron Arama İstek ID'sini Alma

Asenkron arama isteği gönderildikten sonra, Elasticsearch bir `task` belgesi içinde bir `task` ID'si döner. Bu ID, arama sonuçlarını almak için kullanılacaktır.

```json

{

"task": "XYZ123456789"

}

```

### Adım 3: Asenkron Arama Sonuçlarını Alma

Elasticsearch, asenkron arama işlemini gerçekleştirirken, belirtilen `task` ID'si ile sonuçları almak için `\_tasks` endpoint'ini kullanmanıza olanak tanır.

```bash

GET /\_tasks/XYZ123456789

```

Bu istek sonucunda, asenkron arama işleminin durumu ve sonuçları alınabilir. Durum "completed" olduğunda, sonuçlar kullanılabilir hale gelir.

Asenkron arama, özellikle büyük veri setleri üzerinde karmaşık sorguların gerçekleştirilmesi durumunda performans ve kullanıcı deneyimi açısından önemlidir. Bu özellik, sorguların uzun süre çalışması durumunda kullanıcılara daha hızlı geri dönüşler sağlar.

### Arama Uygulamaları Geliştirme(Developing Search Applications)

#### Highlight the search terms in the response of a query

Elasticsearch'te, arama sorgularının sonuçlarında belirli terimleri vurgulamak için "highlighting" özelliği kullanılabilir. Bu özellik, kullanıcının sorgulanan terimlerin nerede bulunduğunu daha belirgin bir şekilde görmesine yardımcı olur. İşte Elasticsearch'te arama sonuçlarında terimleri vurgulamak için kullanılan `highlight` özelliği ile ilgili detaylı bir açıklama:

### Adım 1: Highlight Özelliğini Kullanma

Highlight özelliğini kullanmak için, `\_search` endpoint'ine bir POST isteği gönderirken `highlight` parametresini kullanmalısınız. Örneğin:

```bash

POST /my\_index/\_search

{

"query": {

"match": {

"field": "search\_term"

}

},

"highlight": {

"fields": {

"field": {} // Vurgulama yapılacak alan

}

}

}

```

Bu örnekte, `"field"` adlı bir alanın içindeki belgelerde "search\_term" terimini arar ve bu terimleri vurgular. `highlight` içindeki `"fields"` parametresi, hangi alanların vurgulanacağını belirtir.

### Adım 2: Response İçinde Highlight Sonuçlarını İnceleme

Arama sonuçları geldiğinde, her belge içinde `highlight` alanında vurgulama bilgilerini bulabilirsiniz. Örnek bir response şu şekildedir:

```json

{

"took": 10,

"hits": {

"total": {

"value": 1,

"relation": "eq"

},

"hits": [

{

"\_source": {

"field": "Bu bir örnek metin içerir."

},

"highlight": {

"field": ["Bu bir <em>örnek</em> metin içerir."]

}

}

]

}

}

```

Bu örnekte, `field` alanındaki belgenin içinde "örnek" terimleri vurgulanmıştır. Vurgulama, HTML etiketleri (`<em>`) kullanılarak gerçekleştirilir, ancak bu konfigürasyon değiştirilebilir.

### Dikkat Edilmesi Gerekenler:

1. Vurgulama bilgileri, arama sorgusunun tam olarak eşleşen terimleri içerir ve analiz işlemleri dikkate alınır.

2. Highlighting özelliği, özellikle tam metin aramalarında kullanışlıdır ve kullanıcıların belirli terimleri içeren belgeleri hızlı bir şekilde görmelerini sağlar.

3. Vurgulama konfigürasyonları özelleştirilebilir. Örneğin, etiketlerin türü, fragment uzunluğu, vb. belirlenebilir.

Highlighting özelliği, kullanıcı dostu arama sonuçları sunmak ve belirli terimleri vurgulamak için kullanışlı bir araçtır.

#### Sort the results of a query by a given set of requirements

Elasticsearch'te sorgu sonuçlarını belirli bir dizi gereksinime göre sıralamak için "sort" özelliği kullanılır. Bu özellik, belirli bir veya birden fazla alanı ve sıralama yöntemini belirterek sonuçları düzenlemenize olanak tanır. İşte bir sorguyu belirli bir set of gereksinimlere göre sıralamanın detaylı açıklaması:

### Adım 1: Temel Sıralama

Temel sıralama, belirli bir alanın değerlerine göre sonuçları artan (ascending) veya azalan (descending) şekilde sıralamak için kullanılır. Sıralama, `"sort"` parametresi ile sorgu gövdesinde belirtilir.

```bash

POST /my\_index/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"field1.keyword": {

"order": "asc"

}

}

]

}

```

Bu örnekte, "my\_index" adlı endeks içindeki belgeler, "field1.keyword" adlı alanın değerlerine göre artan sırayla sıralanır.

### Adım 2: Birden Fazla Alanla Sıralama

Eğer birden fazla alanı kullanarak sıralama yapmak istiyorsanız, sıralama alanlarını içeren bir dizi kullanabilirsiniz.

```bash

POST /my\_index/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"field1.keyword": {

"order": "asc"

}

},

{

"field2.keyword": {

"order": "desc"

}

}

]

}

```

Bu örnekte, "field1.keyword" adlı alanın değerlerine göre artan sırayla sıralama yapılır ve eşit değerlere sahip belgeler, "field2.keyword" adlı alanın değerlerine göre azalan sırayla sıralanır.

### Adım 3: Fonksiyon Bazlı Sıralama

Elasticsearch'te, belirli bir fonksiyonu kullanarak özel sıralama stratejileri de uygulayabilirsiniz.

```bash

POST /my\_index/\_search

{

"query": {

"match\_all": {}

},

"sort": [

{

"\_geo\_distance": {

"location": {

"lat": 40.7128,

"lon": -74.0060

},

"order": "asc",

"unit": "km"

}

}

]

}

```

Bu örnekte, "location" adlı bir geo\_point alanını kullanarak, belgeleri belirli bir coğrafi konumdan uzaklıklarına göre artan sırayla sıralama yapılır.

Sıralama, Elasticsearch'te arama sonuçlarını organize etmek ve belirli gereksinimlere göre düzenlemek için güçlü bir araçtır. Kullanılacak sıralama alanlarına ve sıralama stratejilerine bağlı olarak, sıralama parametrelerini özelleştirebilirsiniz.

#### Implement pagination of the results of a serarch query

Elasticsearch'te sorgu sonuçlarının sayfalama (pagination) işlemi, büyük veri setleri üzerinde çalışırken sadece belirli bir sayıdaki sonuçları almanızı sağlayan önemli bir özelliktir. Sayfalama, sorgu sonuçlarını daha etkili bir şekilde işlemek için kullanılır ve kullanıcıların belirli bir sayfa içindeki belirli bir sonuç kümesine odaklanmalarını sağlar.

Sayfalama işlemi için `from` ve `size` parametrelerini kullanarak sorgu gönderirsiniz.

İşte bir sorgu sonuçlarını sayfalama işlemi için kullanılan temel bir örnek:

```bash

POST /my\_index/\_search

{

"query": {

"match\_all": {}

},

"from": 0, // Başlangıç belgesi (0'dan başlar)

"size": 10 // Her sayfa için belge sayısı

}

```

Bu örnekte:

- `"from"` parametresi, başlangıç belgesini belirtir. Sayfa numaralandırma 0'dan başlar.

- `"size"` parametresi, her sayfa için alınacak belge sayısını belirtir.

Örnekte, ilk sayfadan başlayarak 10 belge alınmaktadır. Sayfa numarasını ve her sayfa için alınacak belge sayısını ihtiyacınıza göre ayarlayabilirsiniz.

Sayfalama, büyük veri setleri üzerinde performansı artırmak ve kullanıcıların daha küçük ve yönetilebilir sonuç kümeleri üzerinde çalışmasını sağlamak için yaygın olarak kullanılan bir yöntemdir.

#### Define and use inex aliases

Elasticsearch'te index alias'ları, bir veya daha fazla indeksi temsil eden, değişmeyen ve sürekli bir adlandırmadır. Alias'lar, endeksleri daha dinamik ve yönetilebilir hale getirmenize yardımcı olabilir. Alias'lar, endekslerin üzerinde yapılan sorguları genişletmek ve sorguların daha esnek ve yönetilebilir olmasını sağlamak amacıyla kullanılır.

İşte index alias'larını tanımlama ve kullanma sürecini anlatan temel adımlar:

### Adım 1: Index Alias'ı Tanımlama

Alias tanımlamak için `\_aliases` endpoint'ini kullanabilirsiniz. Aşağıda, bir alias oluşturma örneği bulunmaktadır:

```bash

POST /\_aliases

{

"actions": [

{

"add": {

"index": "my\_index\_v1",

"alias": "my\_index"

}

}

]

}

```

Bu örnekte, "my\_index\_v1" adlı bir endeksi, "my\_index" adlı bir alias ile eşleştiriyoruz. Artık sorgular "my\_index" alias'ını kullanarak bu endeksi hedefleyebilir.

### Adım 2: Sorguları Alias Üzerinden Gerçekleştirme

Alias'lar, sorguları yönlendirmek ve indekslere dinamik olarak bağlantılar sağlamak için kullanılır. Bir sorgu gerçekleştirirken, alias adını hedef olarak kullanabilirsiniz:

```bash

GET /my\_index/\_search

{

"query": {

"match\_all": {}

}

}

```

Bu sorgu, aslında "my\_index\_v1" adlı endeksi hedefler, ancak alias aracılığıyla gerçekleştirilir.

### Adım 3: Alias'ları Kullanarak İndeks Ekleme ve Kaldırma

Alias'ları güncellemek, yeni bir indeks eklemek veya bir indeksi kaldırmak için kullanabilirsiniz. Aşağıda, alias'e yeni bir indeks ekleme örneği bulunmaktadır:

```bash

POST /\_aliases

{

"actions": [

{

"add": {

"index": "my\_index\_v2",

"alias": "my\_index"

}

}

]

}

```

Bu işlem, "my\_index\_v2" adlı yeni bir endeksi, "my\_index" alias'ına ekler. Artık sorgular otomatik olarak bu yeni endeksi de hedefleyecektir.

Alias'ları kullanmak, endekslerin güncellenmesi veya değiştirilmesi gerektiğinde daha esnek bir yapı sunar. Ayrıca, uygulamanızdaki indeks adlarını değiştirmeden sorgularınızı yönlendirmenizi sağlar. Alias'lar, indeks yönetimi ve sorguların esnekliği açısından Elasticsearch'te güçlü bir araçtır.

#### Define and use a serach template

Elasticsearch'te "search template" kullanarak, önceden tanımlanmış bir sorgu şablonunu kullanabilir ve bu şablona parametreler ekleyerek dinamik sorgular oluşturabilirsiniz. Search template'leri, sık kullanılan sorguları standartlaştırmak, sorgu karmaşıklığını azaltmak ve sorgu yapılarını daha okunabilir ve yönetilebilir hale getirmek için kullanılır.

İşte bir search template kullanma süreci:

### Adım 1: Sorgu Şablonunu Tanımlama

Search template tanımlamak için `\_scripts` veya `\_template` endpoint'ini kullanabilirsiniz. Bu örnek, bir sorgu şablonu tanımlar:

```bash

POST /\_scripts/my\_search\_template

{

"script": {

"lang": "mustache",

"source": {

"query": {

"match": {

"{{field}}": "{{value}}"

}

}

}

}

}

```

Bu örnekte, `my\_search\_template` adlı bir sorgu şablonu tanımlanmıştır. Şablonda `{{field}}` ve `{{value}}` gibi parametre yer tutucuları kullanılarak dinamik değerler alabilir.

### Adım 2: Sorgu Şablonunu Kullanma

Tanımlanan sorgu şablonunu kullanmak için, `\_search/template` endpoint'ini ve `id` parametresini kullanabilirsiniz:

```bash

POST /\_search/template/my\_search\_template

{

"params": {

"field": "title",

"value": "Elasticsearch"

}

}

```

Bu örnekte, `my\_search\_template` adlı sorgu şablonu kullanılarak, `"field": "title"` ve `"value": "Elasticsearch"` parametreleriyle doldurulmuş bir sorgu gönderilmiştir.

### Adım 3: Sonuçları İnceleme

Sorgu şablonu kullanıldığında, Elasticsearch size sorgunun sonuçlarını dönecektir.

Search template'ler, özellikle karmaşık veya sıkça kullanılan sorguların standartlaştırılması gerektiği durumlarda, sorguların yönetimini ve okunabilirliğini artırmak için kullanışlıdır. Parametreleri kullanarak sorgu şablonlarını özelleştirebilir ve aynı şablonu farklı koşullar altında kullanabilirsiniz. Bu, Elasticsearch üzerinde daha etkili ve esnek bir sorgulama deneyimi sunmanıza yardımcı olabilir.

### Teşhis ve Hata Ayıklama(Diagnostics and debugging)

Elasticsearch'te tanı ve hata ayıklama (diagnostics and debugging) işlemleri, sistemin durumunu izlemek, performans sorunlarını tespit etmek ve hataları çözmek için önemlidir. Bu işlemler, Elasticsearch küme yönetimi ve performans optimizasyonu açısından kritik öneme sahiptir. İşte Elasticsearch'te tanı ve hata ayıklama süreçleri için kullanılan bazı yöntemler:

### 1. \*\*Cluster ve Node Durumu İzleme:\*\*

Elasticsearch kümenizin durumunu izlemek için `\_cluster/health` endpoint'ini kullanabilirsiniz. Bu, kümenin genel sağlığını, aktif ve inaktif node'ları, atılmış shard'ları ve diğer önemli bilgileri sağlar.

```bash

GET /\_cluster/health

```

### 2. \*\*Node İstatistikleri ve Durumu:\*\*

Bir node'un durumunu ve istatistiklerini görmek için `\_nodes/stats` ve `\_nodes/<node\_id>/stats` endpoint'lerini kullanabilirsiniz.

```bash

GET /\_nodes/stats

GET /\_nodes/<node\_id>/stats

```

### 3. \*\*Shard Durumu Kontrolü:\*\*

Kümedeki shard'ların durumunu kontrol etmek için `\_cat/shards` endpoint'ini kullanabilirsiniz. Bu, shard'ların her biri için ayrıntılı bilgileri içerir.

```bash

GET /\_cat/shards

```

### 4. \*\*Arama Açıklamaları:\*\*

Elasticsearch'te gerçekleştirdiğiniz sorguların ayrıntılı açıklamalarını almak için `\_search/explain` endpoint'ini kullanabilirsiniz.

```bash

GET /index\_name/\_search/explain

{

"query": {

"match": {

"field": "value"

}

}

}

```

### 5. \*\*Arama Profillemesi:\*\*

Arama sorgularınızın performansını değerlendirmek için `\_search/profile` endpoint'ini kullanabilirsiniz. Bu, sorguların ne kadar süre aldığını ve hangi aşamalarda performans sorunları yaşandığını gösterir.

```bash

POST /index\_name/\_search

{

"profile": true,

"query": {

"match": {

"field": "value"

}

}

}

```

### 6. \*\*Indexing İstatistikleri:\*\*

Indexing işlemlerinin durumu ve istatistikleri için `\_cat/indices` endpoint'ini kullanabilirsiniz.

```bash

GET /\_cat/indices

```

### 7. \*\*Log Kayıtları İnceleme:\*\*

Elasticsearch log kayıtları, sistemdeki önemli olayları ve hataları içerir. Log dosyalarını inceleyerek sorunların kaynağını bulabilirsiniz. Varsayılan olarak, log dosyaları genellikle `/var/log/elasticsearch/` dizininde bulunur.

### 8. \*\*Explain API Kullanımı:\*\*

Bir sorgunun sonuçlarını ve eşleşen dokümanların skorlarını ayrıntılı bir şekilde anlamak için `\_search/explain` endpoint'ini kullanabilirsiniz.

```bash

GET /index\_name/\_search/explain

{

"query": {

"match": {

"field": "value"

}

}

}

```

Bu yöntemler, Elasticsearch kümenizin sağlığını izlemek, performans sorunlarını belirlemek ve hataları çözmek için kullanılabilir. İhtiyaca göre bu yöntemleri bir arada kullanarak daha kapsamlı bir tanı ve hata ayıklama süreci oluşturabilirsiniz.