Elasticsearch on fruits

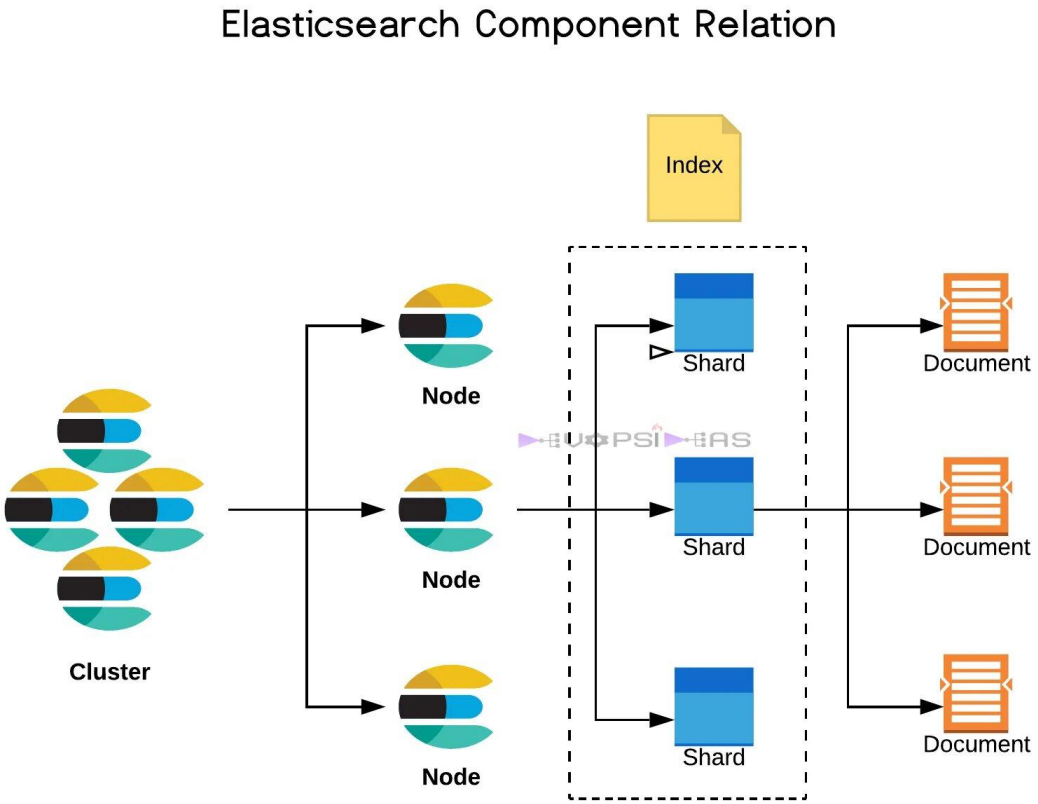
Agenda

- **Elasticsearch Architecture**
  (nodes, indexes, shards, replicas, routing, read/write/failover)
- **Index API**
- **Indexing Documents - Create, Read, Update, Delete**
- **Multi-Document API**
- **Datatypes & Mapping**
- **Text Analysis**
- **Query DSL - Search & Aggregation**
- **Understanding Relevance Score**
  (TF-IDF, Okapi BM25)

Architecture

- Nodes
- Index
- Shard
- Replica
- Routing
- Read / Write / Failover

Elasticsearch Architecture



Nodes

- **Node** - являє собою запущений **Elasticsearch** процес
- Можуть фізично знаходитись на різних серверах, або ж у контейнерах чи віртуальних машинах
- Усі **nodes of cluster** знають одна про одну та можуть обмінюватися даними.
- Основні види **nodes**:
  - *master (eligible)* – управляє індексами та метаданими кластера
  - *data (hot, warm, cold)* – виконують лише операції пов’язані з даними
  - *coordinating* – розумне балансування навантаження

Index

- Це колекція документів з однаковими або схожими полями.
- Може складатися із однієї або декількох шард, що допомагає розподілити документи між поміж **nodes**.
- Документи які потрапляють у індекс зберігаються повністю в оригіналі.
- Поля документів індексуються у додаткові структури даних щоб прискорювати пошук та інші операції.

Shard (1)

- Це будівельні блоки які утворюють індекс та відповідають за аналіз, збереження та пошук документів.
- Типовий індекс складається із 1 або більше **shards**. У останньому випадку кожен **shard** міститиме лише порцію даних індекса.
- Реалізований як Lucene Index і по суті є повноцінним двигуном пошуку над даними.
- Коли документи потрапляють у шард, вони періодично пишуться у фіксовані Lucene сегменти на диску і стають доступними для пошуку. Цей процес відомий як **refresh**.
- Коли кількість сегментів зростає, відбувається їх об’єднання (**merge**). І так як сегменти є незмінні, новий сегмент утворюється копіюванням старих і заміною старих на один новий. Це спричиняє навантаження на IO та Disk Memory ресурси.

Shard (2)

- Оскільки сегменти незмінні, при оновленні документа, старий буде відмічений як deleted, а новий буде збережено.
- Те ж саме із видаленням, deleted документи все ще фізично зберігатимуться доки не відбудеться новий merge.
- Розмір не обмежений, але не рекомендується мати шарди що займають більше **50Gb**.
- За наявності кількох shard запити виконуються паралельно і результати об’єднуються.
- За замовчуванням **refresh** відбувається кожну секунду (near real time), але це значення можна змінювати.
- Кількість **primary shard** фіксована і задається при створенні індекса. Після чого її не можна змінювати.

Replica

- **Shards** поділяють на два типи **primary** та **replica** для підтримки *data redundancy* та *high availability*.
- **Replica shards** це по суті копії **primary shard**.
- Розміщуются на **nodes of cluster** інших ніж їхній **primary shard**.
- У випадку якщо нода що містить **primary shard** стає недоступна, репліка займає її місце і змінює роль.
- Репліки також беруть участь у виконанні пошукових запитів. Тож іноді збільшення реплік може покращити продуктивність.
- Спочатку документи зберігаються у **primary shard**, а уже потім паралельно копіюються у репліки.

Routing

- Коли відбувається індексування, оновлення, пошук чи видалення документа, **routing** указує до якої із **shard** відноситься дана операція.
- **Shard** визначається за формулою
  shard_num = hash(_routing) % num_of_primary_shards
- За замовчуванням для **_routing** використовується **_id** документа, але це значення можна змінити на інше. За умови, що його завжди передаватимуть для усіх операцій.
- Можливість зміни **routing** значення дозволяє згрупувати документи які логічно пов’язані і має сенс зберігати їх у одному **shard**-і.
- Дуже важливо щоб документи розподілялися рівномірно по шардах, інакше продуктивність операцій знизиться.
- Пошук використовує адаптивний роутинг який намагається підібрати **primary or replica shard** здатну оптимально виконати операцію.

Read / Write / Failover (1)

- **Primary shard** разом із **replica shards** утворюють групу що називається **replica set**.
- Як уже згадувалося, зміна даних **(write/update/delete)** відбувається спершу на **primary shard** а потім поширюється на **replica shards**.
- Операція вважається завершеною після того як **primary shard** переконається, що всі **replica shards** записали документ.
- Якщо ж при запису якась репліка стала недоступна, **primary shard** попросить **master node** прибрати цю репліку із списку **replica set**.
- Таким чином операції пошуку можуть без проблем виконуватись на **primary** та **replica shards** із активної групи **replica set**.
- Для відновлення після ситуацій у яких **primary shard** дає збій і її місце у **replica set** займає **replica shard** були додані лічильники **primary term** (*_primary_term*) та **sequence number** (*_seq_no*) які додаються до кожної операції що вносить зміни.

Read / Write / Failover (2)

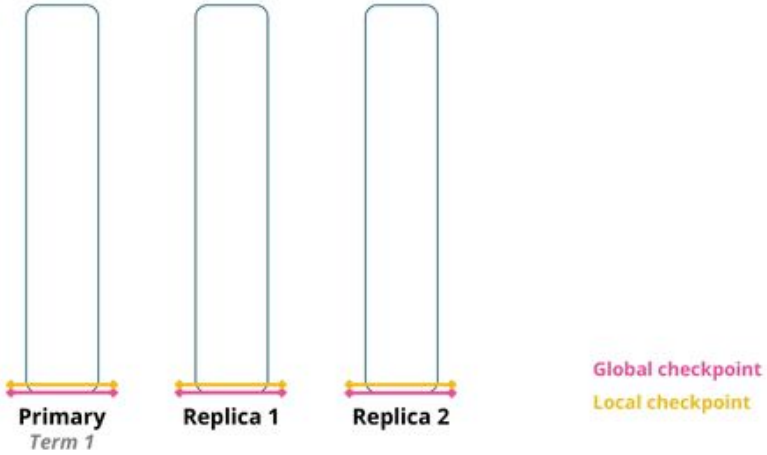
- **Primary term** лічильник збільшується кожного разу коли для **replica set** змінюється **primary shard**. Та зберігається у стані кластера на **master node**. Таким чином коли **primary shard** яка була зламана повертається у **replica set** вона все ще певний час вважає себе основною і може відправляти операції які не були поширені на інші репліки. Такі операції відразу ж ідентифікуються тому що цей лічильник буде застарілий і вони будуть відкинуті.
- **Sequence number** лічильник який **primary shard** присвоює кожній новій операції яка змінює документи, також він зберігається разом з документом який змінювався.
- Таким чином комбінація чисел **primary term** та **sequence number** унікально визначає версію документа та дозволяє реалізувати **Optimistic Locking** механізм. Але це ще не все :)

Read / Write / Failover (3)

- **Global checkpoint** це **sequence number** значення, яким управляє **primary shard**.
- Усі операції які мають **sequence number** менший ніж **global checkpoint** це уже ті операції які гарантовано були виконані на **primary** та **replica shards** у **replica set**.
- **Global checkpoint** має важливе значення коли необхідно дуже швидко порівняти операції щоб синхронізувати консистентність між усіма шардами у **replica set**, якщо відбувалися якісь помилки та заміна **primary shard**.
- Кожен **shard** у **replica set** також підтримує **local checkpoint**, це той же **sequence number**, але його значення указує на те що усі операції із меншим **sequence number** уже були успішно виконані для цієї **primary or replica shard**.

Read / Write / Failover (4)

- Коли якась **replica** виконала операцію від **primary**, у відповідь вона відправляє **local checkpoint**. Таким чином **primary** знає коли можна оновити **global checkpoint**, а при наступній операції це оновлене значення буде передане усім **replica shards**.



Index API

- ➔ Create / Get / List Indexes
- ➔ Get / Update Mapping
- ➔ Analyze Disk Usage
- ➔ Get / Update Settings
- ➔ Get Index Stats

Index - create

PUT /{index-name}

```
{
    “settings”: {
        “index”: {
            “number_of_shards”: 1,
            “number_of_replicas”: 0
        }
    },
    “mappings”: {
        “properties”: {
            “name” : {“type”: “keyword”}
        }
    }
}
```

Index - get

GET /{index-name}

Дозволяє переглянути деталі про індекс:

- ➔ Aliases
- ➔ Mappings
- ➔ Settings

Index - list

GET /_cat/indices?format=JSON

Response:


```
[{
   "health": "green",
   "status": "open",
   "index": "news",
   "uuid": "4yowodBsSeOUkAaxlP9-8Q",
   "pri": "1",
   "rep": "0",
   "docs.count": "10",
   "docs.deleted": "0",
   "store.size": "70.3kb",
   "pri.store.size": "70.3kb"
 }]
```


Index - delete

DELETE /{index-name}

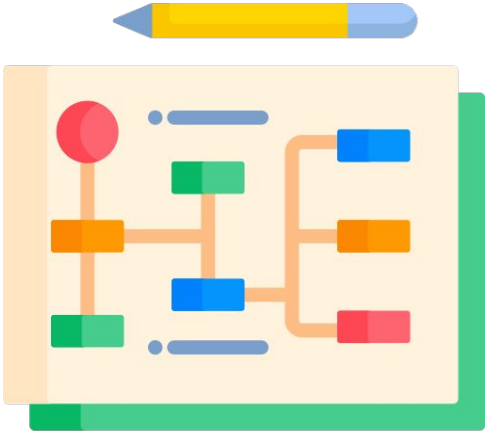
Видаляє індекс разом із документами та метаданими.

Index - get / update mapping

GET /{index-name}/_mapping

PUT /{index-name}/_mapping

```
{"index-name": {
    "mappings": {
        "properties": {
            "created": { "type": "date"}
        }}
}
```



Index - analyze disk usage

POST /{index-name}/_disk_usage*?run_expensive_tasks=true*

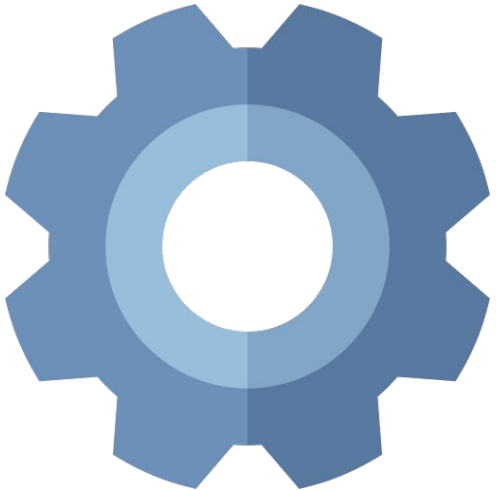
Дозволяє переглянути скільки пам’яті займають індексовані поля документів та загальну інформації про використання дискової пам’яті індексом.

Index - get / update settings

GET /{index-name}/_settings

PUT /{index-name}/_settings

- ➔ Дозволяє переглянути поточну конфігурацію індекса.
- ➔ Дозволяє оновити динамічні параметри конфігурації індекса.
- ➔ Dynamic Index Settings



Index - get statistics

GET /{index-name}/_stats

Відображає детальну статистику використання індекса.

Indexing Documents

- ➔ Create
- ➔ Read
- ➔ Update
- ➔ Delete

Create Document

PUT /{index-name}/_doc/[_id]?[params]

Зберігає новий документ, або повністю оновлює, якщо вказано **_id** та документ уже присутній у індексі. Деякі цікаві параметри:

- **if_primary_term**={int-primary-term} та **if_seq_no**={int-sequence-num} Операція буде виконана за умови що документ має задані значення.
- **refresh**={true|false} говорить що документ слід відразу включити у пошук.
- **routing**={value} значення яке буде використовуватися для вибору **shard**.
- **timeout**={1m} максимальна тривалість операції.
- **wait_for_active_shards**={int-value} число **shard** які мають бути доступні перед початком операції.

Create Document Response

_index: "fruits"

_id: "3"

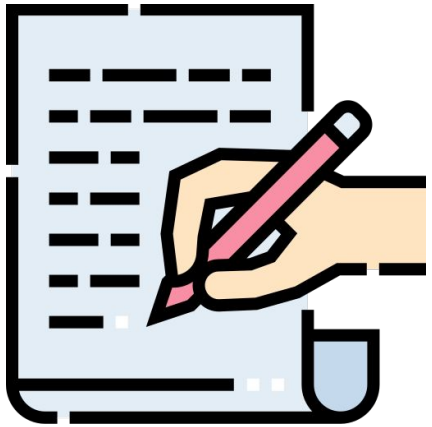
_version: 1

result: "created" | “updated”

_shards: { "total": 2, "successful": 1, "failed": 0 }

_seq_no: 4,

_primary_term: 1



Read Document

GET /{index-name}/_doc/{_id}

GET /{index-name}/_source/{_id}

HEAD /{index-name}/_doc/{_id}

Цікаві параметри:

- routing
- _source, _source_[includes | excludes]
- preference
- version

Read Document Response

_index: "fruits",

_id: "3",

_version: 1,

_seq_no: 4,

_primary_term: 1,

found: true,

_source: { "name": "kiwi" }

Update Document

POST /{index-name}/_update/{_id}

For **scripted** update:

```
{“script”: {
    “source”: “ctx._source.counter += params.count”,
    “lang”: “painless”,
    “params”: {“count”: 4}}
}
```

For **partial** update:

```
{
  “doc”: {“count”: 20}
}
```

Update Document Response

_index: "fruits",

_id: "3",

_version: 2,

result: "updated" | “noop”,

_shards: { "total": 2, "successful": 1, "failed": 0 },

_seq_no: 6,

_primary_term: 1

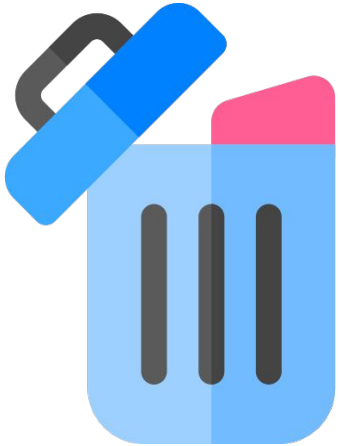
Delete Document

DELETE /{index-name}/_doc/{_id}

Позначить документ як видалений і прибере його із результатів пошуку.

Цікаві параметри:

- if_seq_no , if_primary_term
- refresh
- routing
- timeout
- version
- wait_for_active_shards



Delete Document Response

_index:	"fruits",
_id:	"2",
_version:	4,
result:	"deleted",
_shards:	{ total": 2, "successful": 1, "failed": 0},
_seq_no:	7,
_primary_term:	1

Multi-Document API

- ➔ BULK
- ➔ UPDATE BY QUERY
- ➔ DELETE BY QUERY
- ➔ REINDEX

Bulk Operations

POST /_bulk

POST /{index-name}/_bulk

{ "index": { "_index": "fruits", "_id": "1" }}

{ "name": "Orange" }

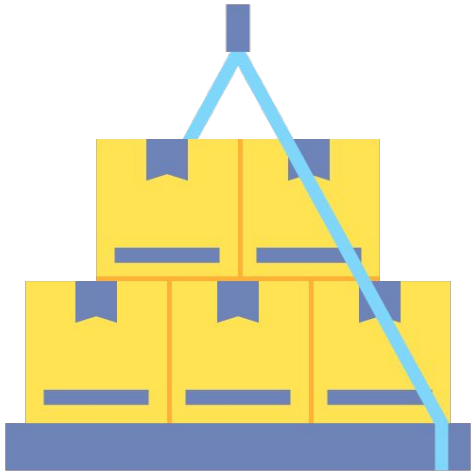
{ "create": { "_index":"fruits", "_id": "2" }}

{"name": "Apple"}

{"update": { "_index": "fruits", "_id": "3" }}

{"doc": { "count": 1 }}

{"delete": { "_index": "fruits", "_id": "4" }}



NOTE: **Content-Type: application/x-ndjson**

Bulk params

Деякі цікаві параметри:

- **refresh** [true | false] - робить зміни видимими для пошуку
- **routing** - перевизначає значення для вибору shard
- **_source** [true | false] - дозволяє виключити контент документів з відповіді
- **timeout** [1m] - ліміт на створення індексів, оновлення динамічних мапінгів, очікування активних shard
- **wait_for_active_shards** - задає мінімум shard що мають бути активні перед виконанням операції

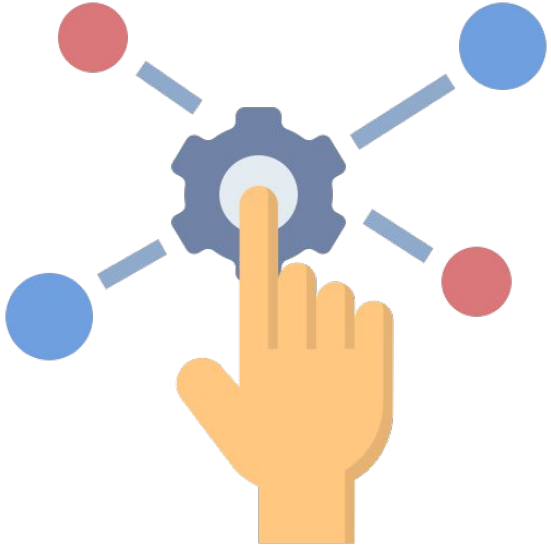
Bulk actions

- **create**
  Зберігає документ, якщо його не було.

- **delete**
  Видаляє документ з таким **_id**.

- **update**
  Виконує часткове оновлення документу.

- **index**
  Індексує документ, якщо такий уже є, замінює його на нову версію.



Bulk Operations Response

took:	227
errors:	false
items:	
[index, create, update, delete]:	
_index:	fruits
_id:	“1”
_version:	1
result:	[created | deleted | updated]
_shards:	{total: 2, successful: 2, failed: 0}
_seq_no:	25
_primary_term:	3
status:	200 | 201 | 404
error:	{type, reason, shard, index, index_uuid}

Update by Query

POST /{index-name}/_update_by_query

```
{
  “script”: { “source”: “ctx._source.in_stock--” },
  “query”: { “match_all”: { } }
}
```

Оновлює документи які відповідають заданому критерію, також дозволяє оновити індекс після зміни **mapping** (не змінюючи **_source**).

Як це працює:

- Робиться знімок поточного стану, щоб зафіксувати версії.
- Документи що відповідають критеріям об’єднуються в батчі та оновлюються разом.
- Якщо версія документу не відповідає знімку, уся операція переривається.
- Конфлікти версій можна ігнорувати.
- Якщо операція провалилася, зміни які встигли застосуватися – не будуть відмінені.

Update by Query params

- **conflicts** [abort | proceed]
  описує як діяти якщо зустрівся конфлікт версій.
- **preference**
  дозволяє задати node або shard для операції.
- **refresh** [true | false]
  дозволяє відразу зробити зміни видимими для пошуку.
- **requests_per_second**
  дозволяє обмежити кількість запитів на оновлення.
- **scroll**
  час на зберігання пошукового контексту для scrolling-а.
- **scroll_size** [1000]
  розмір scroll запитів.
- **search_type** [query_then_fetch | dfs_query_then_fetch]
  як виконувати пошук.
- **slices**
  оновлення можна поділити на декілька задач.

Update by Query Response

took: 6,

timed_out: false,

total: 1,

updated: 1,

deleted: 0,

batches: 1,

version_conflicts: 0,

noops: 0,

retries: { "bulk": 0, "search": 0},

throttled_millis: 0,

requests_per_second: -1,

throttled_until_millis: 0,

failures: []

Delete by Query

POST /{index-name}/_delete_by_query

```
{“query”: {
    “match”: { “name”: “Apple” }
}}
```

Видаляє документи які відповідають заданим критеріям пошуку.

Delete by Query Response

took: 147,

timed_out: false,

total: 119,

deleted: 119,

batches: 1,

version_conflicts: 0,

noops: 0,

retries: { "bulk": 0, "search": 0 },

throttled_millis: 0,

requests_per_second: -1.0,

throttled_until_millis: 0,

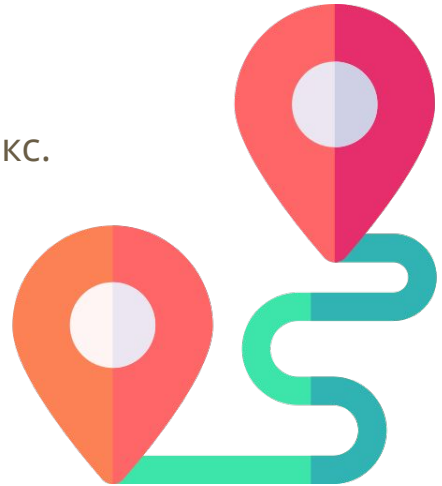
failures: [ ]

Reindex

POST /_reindex

```
{
    “source”: { “index”: “{src-index-name}” },
    “dest”: { “index”: “{dest-index-name}” }
}
```

Копіює та заново індексує документи зі старого у новий індекс.



Data Types & Mapping

- Alias
- Array
- Boolean
- Numeric
  - *byte, short, integer, long*
  - *float, double*
- Date
- Keyword
- Text
- Object
- Flattened
- Nested

Mapping

```
{ “mappings”: {
    “properties”: {
        “age”: { “type”: “integer” },
        “email”: { “type”: “keyword” }
        “name”: { “type”: “text” }
    }}}
```

Mapping – це схема документу яка дає вказівку **Elasticsearch** як індексувати поля документу. Мапінг може бути статичний і динамічний, але якщо вам заздалегідь відома структура документу і ви бажаєте мати більше контролю над індексуванням полів - використовуйте статичний мапінг.

Як видно із даного JSON-а, кожному полю присвоюється якийсь тип. Про ці типи ми і поговоримо далі.

Alias

```
“properties”: {
        “username”: { “type”: “text” },
        “login”: { “type”: “alias”,  “path”: “username” }
}
```

Задає альтернативне ім’я для існуючого поля, яке потім можна використовувати у пошукових запитах.

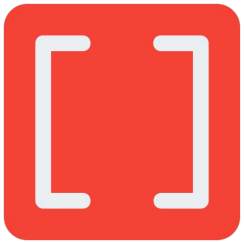
Array

Для масивів не існує окремо відведеного типу даних, за замовчуванням кожне поле може містити нуль або більше значень.

Проте є умова що елементи масивів мають мати одинаковий тип.

Деякі приклади:

- ["one", "two"]
- [1, 2]
- [1, [2, 3]] => [1, 2, 3]
- “arr”: [{ "name": "Mary", "age": 12 }, { "name": "John", "age": 10 }]
  =>
  {
  “arr.name”: [“Mary”, “John”],
  “arr.age”: [12, 10]
  }



Boolean

“properties”: {

“hidden”: { ***“type”: “boolean”*** }

}

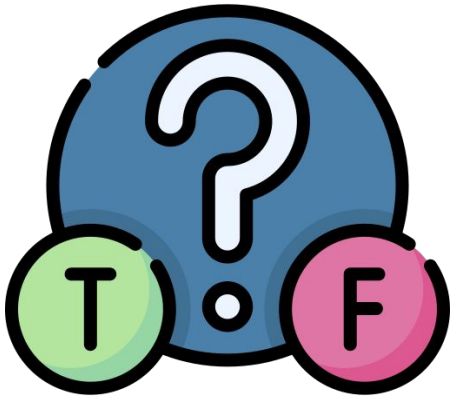
Приймає класичні два значення, деякі строки також може класифікувати у цей тип.

False values

`false`, `"false"`, `""` (empty string)

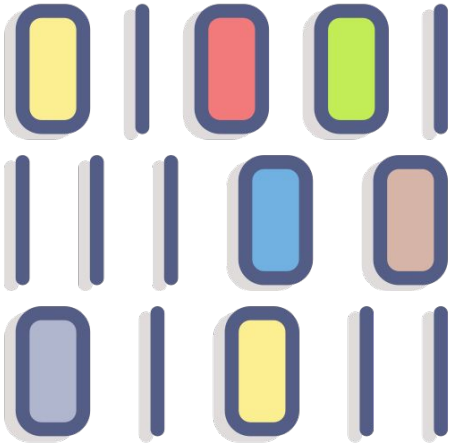
True values

`true`, `"true"`



Numeric

Type Name	Note	Min	Max
integer	32 bit integer	-2^{31} - 2 147 483 648	$+2^{31}-1$ + 2 147 483 647
long	64 bit integer	-2^{63} - 9 223 372 036 854 775 808	$+2^{63}-1$ + 9 223 372 036 854 775 807
float	32 bit floating point		
double	64 bit floating point		
scaled_float	backed by 64 bit long with double scaling factor		



Date

```
“properties”: {
        “createdAt”: {
                “type”: “date”,
                “format”: “yyyy-MM-dd’T’HH:mm:ss”
        }
}
```

У JSON документі не існує окремого типу для Date, тому такий тип можуть представляти:

- строки що містять форматовані значення, e.g. "2015-01-01" чи "2015/01/01 12:10:30"
- число яке вказує скільки мілісекунд минуло від Unix Epoch (1970-01-01T00:00:00)
- число яке вказує скільки секунд минуло від Unix Epoch (1970-01-01T00:00:00)

Keyword

```
“properties”: {
        “tag”: { “type”: “keyword” }
}
```

Використовують для зберігання структурованих полів, такий текст не аналізується а зберігається як є. Добре підходить для збереження таких даних як:

- IDs
- Email addresses
- Status codes
- ZIP codes
- Hashtags
- etc.

Text

```
“properties”: {
    “description”: { “type”: “text” }
}
```

Тип який дозволяє побудувати інвертований індекс для повнотекстового пошуку. Контент такого поля аналізується (розбивається на токени) перед збереженням у індексі. Потім під час пошуку, запит також аналізується і тоді кожен токен шукається окремо.

За допомогою опцій **analyzer** та **search_analyzer** можна явно вказувати як аналізувати текст перед збереженням та під час пошуку. Якщо вказано тільки перший варіант, то він буде використовуватися для обох випадків.

Object

```
“properties”: {
   “user”: {
            “properties”: {
                     “id”: { “type”: “integer” }
                     “firstName”: { “type”: “keyword” }
                     “lastName”: {“type”: “keyword”}
            }
   }
}
```

Дозволяє окремо індексувати поля для JSON об’єкта. Сам Elasticsearch їх бачить як:

- **user.id**
- **user.firstName**
- **user.lastName**

Flattened

```
“properties”: {
    “label”: { “type”: “flattened” }
}
```

Як альтернатива для типу Object, розкладає leaf поля JSON об’єкта та індексує як одне поле із типом keyword.

Nested (1)

```
“properties”: {
        “authors”: {
                “type”: “nested”,
                “properties”: {
                        “id”: { “type”: “integer” },
                        “name”: { “type”: “text” }
                }
        }
}
```

Nested (2)

У ситуаціях коли ми маємо JSON документ, на кшталт:

```
{ authors: [
        { id: 101, name: “George Orwell” },
        { id: 102, name: “Ernest Hemingway” }
        ]
}
```

Elasticsearch розкладає його як

```
{
        authors.id: [ 101, 102 ],
        authors.name: [ “George Orwell”, “Ernest Hemingway” ]
}
```

Потім під час пошуку зв’язок між полями об’єктів буде втрачено. Якщо вам необхідно зберегти цей зв’язок - використовуйте **nested** тип та відповідний синтаксис у пошукових запитах.

Thanks for Watching