JOHN BRYCE
Leading in IT Education
matrix company

UPDATED TO ELK V.8.3.3

Deep Dive: Advanced Indexing, Mappings, and Data Relationships

Provided by DevOpShift & Autherd by Yaniv Cohen
DevopShift CEO and Head of Devops JB

- INTRODUCTION -

- Index Settings -
A Deeper Dive

Index settings allow granular control over how data is stored, indexed, and searched. Advanced settings provide optimization and better resource management.

- Advanced Setting -

S

Advanced Setting

Refresh Rate

What is it?

- Controls how often Elasticsearch makes the newly added documents searchable.

Why it matters:

- Balancing between search latency and system performance.

Lower values: faster document availability.
Higher values: reduced system overhead.

```
PUT /index_name/_settings
{
   "refresh_interval": "5s"
}
```

S

Advanced Setting
Merge Settings

What are they?

- Controls the merging of segments, optimizing index storage and search performance.

Why it matters:

- Improves disk I/O and ensures better resource allocation during indexing.

Segments: Small, searchable index parts.

Merging: Combining small segments into larger ones.

```
PUT /index_name/_settings
{
   "index.merge.scheduler.max_thread_count": 3
}
```

S

Advanced Setting

Custom Analyzers

What are they?

- Custom-defined combinations of tokenizers and filters tailored for specific text-processing needs.

Why it matters:

- Standard analyzers may not be suitable for all textual content or languages.

Example Use-Case:

- Processing medical texts or domain-specific jargons.

```
PUT /index_name
{
  "settings": {
    "analysis": {
      "analyzer": {
        "custom_analyzer": {
          "type": "custom",
          "tokenizer": "whitespace"
        }
      }
    }
  }
}
```

Advanced Setting

Blocking Writes

```
PUT
/my-index-000001/_block/write
```

What is it?

- Temporarily blocks write operations to an index.

Why it matters:

- Useful for maintenance operations, backups, or to prevent data addition during certain periods.

S

Advanced Setting

Shard Allocation Filtering - Controlling Shard Placement

Key Points:

- Direct where shards of an index should or should not be allocated.
- Useful for data locality, hardware optimization, or segregating specific data.

```
PUT /index_name/_settings
{
   "index.routing.allocation.include.tag": "value",
   "index.routing.allocation.exclude.tag": "value"
}
```

S

Advanced Setting

Force Merge - Optimizing Segment Storage

Key Points:

What is Merging?

- Elasticsearch indices are divided into chunks called segments. As documents are indexed, more segments are created. Over time, to optimize storage and search performance, Elasticsearch will merge smaller segments into larger ones.

- Force Merge: Is a manual optimization process that merges segments to reduce the count.

Advanced Setting

Force Merge - Optimizing Segment Storage

Key Points:
Advantages:

- Reclaims storage space.
- Can improve search performance by reducing the number of segments checked.

Considerations:

- It's a resource-intensive operation. Ensure minimal cluster activity during force merge.
- It might degrade indexing speed for some time post-operation.
- Frequent force merges can reduce the lifespan of SSDs due to more write operations.

Advanced Setting

Force Merge - Optimizing Segment Storage

Configuration Example:

```
POST /index_name/_forcemerge?max_num_segments=1
```

Advanced Setting

Making an Index Read-Only - Protecting Index Data

Key Points:

- Prevents write operations to safeguard data.
- Useful in scenarios like data archival or ensuring data integrity during certain operations.

```
PUT /index_name/_settings
{
   "index.blocks.write": true
}
```

Advanced Setting

Cache Settings - Fine-Tuning Cache Behavior

Enhancing Query Performance and Resource Management - Key Points:

Query Cache:

- Remembers the results of queries. When the exact same query is called, Elasticsearch can skip the actual searching and just pull the result from the cache.

Tuning:

- Too large a cache can cause excessive memory usage. Too small, and it might not be effectively used.

Field Data Cache:

- Holds the field values for filter or aggregation operations.

Significance:

- Text fields aren't in the cache by default because they'd consume too much heap space. Instead, they use a structure called fielddata.

S

Advanced Setting

Cache Settings - Fine-Tuning Cache Behavior

Enhancing Query Performance and Resource Management - Key Points:
Shard Request Cache:

- Caches the local results of shard-level request results. Very effective for repeated identical queries over time-series data.

Considerations:

- **Evictions:** Elasticsearch will evict older cached data if the cache size grows beyond its configuration.

- **Clear Cache:** It's possible to manually clear caches, but this is usually unnecessary as ES manages cache quite efficiently.

S

Advanced Setting

Cache Settings - Fine-Tuning Cache Behavior

Configuration Example: Disable Query Cache:

```
PUT /index_name/_settings
{
   "index.queries.cache.enabled": false
}
```

Control Field Data Cache Size:

```
PUT /index_name/_settings
{
   "index.fielddata.cache.size": "60%"
}
```

- Index Templates -

T

Index Templates - An Overview

Streamlining Index Configuration and Mappings

Key Points:

Definition:

- Index templates allow for pre-defined settings and mappings to be automatically applied to new indices upon creation.

Significance:

- Ensures consistency, simplifies management, and avoids repetitive configurations.

T

Index Templates

Why Use Index Templates?

Consistency:

- Every index created matching a pattern will have the same settings and mappings.

Automation:

- Reduces manual overhead by applying desired configurations automatically.

Flexibility:

- Multiple templates with different patterns can be defined to cater to varying needs.

T

Index Templates

The Structure of an Index Template

Template Name:
- The unique identifier.

Index Patterns:
- The wildcard patterns for index names this template should apply to.

Order (optional):
- In cases where multiple templates match, the order determines priority.

Settings:
- Desired index settings.

Mappings:
- Field mappings for the index.

Aliases (optional):
- Any index aliases to be applied upon index creation.

Index Templates in Action

Configuration Example

Endpoint PUT _index_template/template_1: This is the HTTP method and the API endpoint we're calling. Here, we're using the PUT method to create or update an index template. template_1 is the unique name we're giving to this template.

index_patterns: This defines which indices the template will be applied to.
["pattern-*"]: This means that the template will be applied to any new index created with a name starting with "pattern-". For instance, pattern-1, pattern-2022, pattern-jan and so on.

settings: Here, we define specific configurations we want for the index.
number_of_shards: The template specifies that any index matching the given pattern should have 2 primary shards. This affects how the data is distributed across the cluster.

mappings: This defines the structure of the data that's going to be indexed.
properties: Within properties, we're setting up the structure of the indexed documents.
field1: we're specifying a field named "field1".
type: "text": The type for this field is "text". This means Elasticsearch will treat any data in this field as textual data, which will be fully searchable and will support full-text search capabilities.

```
PUT _index_template/template1
{
 "index_patterns": ["pattern-*"],
 "template": {
 "settings": {
   "number_of_shards": 2
 },
 "mappings": {
   "properties": {
     "field1": { "type": "text" }
   }
 }
}
}
```

Index Templates in Action

Wildcards and Priority

Wildcards:

- Broadly match multiple indices. E.g., logs-* matches logs-2022, logs-jan, etc.

Priority:

- If logs-* and logs-2022* templates exist, set a higher order on the more specific one to ensure it takes precedence.

T

Index Templates in Action

Best Practices

Version Control:
- Keep track of templates, especially in larger teams.

Specificity:
- Be precise with patterns to avoid unintentional matches.

Review:
- Regularly review and prune outdated or unnecessary templates.

- Advanced Mapping Configurations -

Fine-Tuning Data Representations in Elasticsearch

Advanced Mapping Configurations
Field Data Types

Advanced Data Types: Elasticsearch offers data types beyond the basics to cater to specific data.

Full list: https://www.elastic.co/guide/en/elasticsearch/reference/current/sql-data-types.html

Advanced Mapping Configurations

Geo_Point - For geographical locations, allowing for distance calculations and geo queries.

One of those special data type is: Geo_Point

Usage as: Latitude and Longitude Pairs

Definition: Used to store geographical locations as latitude and longitude pairs.

Example Mapping:

```
{
  "properties": {
    "location": {
      "type": "geo_point"
    }
  }
}
```

Example Data Insertion:

```
{
  "location": {
    "lat": 40.7128,
    "lon": -74.0060
  }
}
```

T

Advanced Mapping Configurations

Geo_Point - Geospatial Searches as an example

Distance Query: Find points within a specified distance from a central point.

```
{
 "query": {
  "bool": {
   "filter": {
    "geo_distance": {
     "distance": "20km",
     "location": {
      "lat": 40.7128,
      "lon": -74.0060
     }
    }
   }
  }
 }
}
```

- Dynamic Mapping - An Introduction

Fine-Tuning Data Representations in Elasticsearch

Dynamic Mapping - An Introduction

Automatic Field Detection

- Elasticsearch can automatically detect the data type of fields during indexing.
- It offers convenience but sometimes might lead to undesired mappings.
- Using dynamic mapping templates, we can define custom rules for mapping new fields.

How Dynamic Mapping Works?

- When a new document is indexed and Elasticsearch encounters an unknown field, it uses the field's name and its value to guess the field type.

Examples:

- "age": 26 would be mapped as an integer.
- "is_active": true would be mapped as a boolean.

Dynamic Mapping

Controlling Dynamic Mapping Behavior - Making Dynamic Mapping Work for we

Dynamic Setting: Control the behavior using the dynamic setting.

- **true (default)**: Automatically maps new fields.
- **false**: Ignores new fields.
- **strict**: Throws an exception if an unknown field is detected.

Dynamic Templates - Why?

- Define custom rules to better control the automatic mapping of fields.
- For instance, we might want string fields with the name *_text to always be mapped as text, regardless of their actual content.

T

Setting Up Dynamic Templates

Controlling Dynamic Mapping Behavior - Making Dynamic Mapping Work for we

Example:

This example ensures that any new field ending with _text is mapped as a text type, even if its content might suggest another type.

```
{
 "mappings": {
  "dynamic_index_templates": [
   {
    "strings_as_text": {
     "match_mapping_type": "string",
     "match": "*_text",
     "mapping": {
      "type": "text"
     }
    }
   }
  ]
 }
}
```

T

Best Practices for Dynamic Templates

Controlling Dynamic Mapping Behavior - Making Dynamic Mapping Work for we

Be Explicit:

- Whenever possible, provide explicit mappings. Rely on dynamic mappings for unforeseen fields.

Limit Wildcards:

- Be cautious with wildcards. Overuse can lead to unintended mappings.

Regularly Review Mappings:

- Even with templates, periodically review your index mappings to catch any undesired automatic mappings.

Use a Naming Convention:

- Consistent naming patterns for fields can make dynamic templates more effective and predictable.

```
{
 "mappings": {
  "dynamic_index_templates": [
   {
    "strings_as_text": {
     "match_mapping_type": "string",
     "match": "*_text",
     "mapping": {
      "type": "text"
     }
    }
   }
  ]
 }
}
```

- LABS -

Fine-Tuning Data Representations in Elasticsearch

L

Dynamic Mapping Labs

Lab 1: Exploring Default Dynamic Mapping

Objective:

- Discover how Elasticsearch handles unknown fields through dynamic mapping.

Instructions:

- Create a new index: dynamic_test.
- Index a document with fields that are not predefined in mappings.

Retrieve the mappings for dynamic_test and observe the data types Elasticsearch has chosen.

```
{
  "username": "john_doe",
  "last_login": "2023-05-10T12:00:00Z"
}
```

L

Dynamic Mapping Labs

Lab 2: Handling Unknown Fields

Objective:

- Modify dynamic mapping behavior and observe the effects.

Instructions:

- Create a new index, dynamic_strict, with the dynamic setting set to strict.
- Attempt to index a document with an unknown field.
- Observe the error returned by Elasticsearch.
- Discuss: When might a strict dynamic setting be useful?

L

Index Templates Labs:

Lab 1: Creating an Index Template

Objective:

- Set up an index template and observe its effects on new indices.

Instructions:

- Create an index template that applies to indices named template_test_*.
- Define mappings in the template for a text field named description.
- Create a new index, template_test_1, and retrieve its mappings.
- Answer: How can index templates streamline index creation?

L

Index Templates Labs:

Lab 2: Dynamic Template Creation

Objective:

- Define dynamic templates within an index template.

Instructions:

- Extend the template from Lab 1 to include a dynamic template that maps any string ending with _text as a text type.
- Create an index, template_test_2, and index a document with a field named summary_text.
- Retrieve mappings for template_test_2 and observe the data type of summary_text.

L

Index Advanced Setting Labs:

Lab 1: Manipulating Shards and Replicas

Objective:

- Understand the implications of adjusting shard and replica settings.

Instructions:

- Create an index, shard_test, with 1 shard and 0 replicas.
- Observe its health status.
- Update the index to have 2 replicas.
- Observe the updated health status and discuss the changes.

L

Index Advanced Setting Labs:

Lab 2: Exploring the Force Merge

Objective:

- Observe the effects of the force merge operation on an index.

Instructions:

- Index multiple versions of a document into an index named merge_test (UPDATE...)
- Observe the segment count.
- Use the force merge API to reduce the number of segments to 1.
- Observe the changes in segment count and discuss its implications.

Comprehensive Lab: Dynamic Mapping, Templates & Settings

Combine the concepts of dynamic mapping, index templates, and advanced settings.

Instructions:

- Create an index template for indices named comprehensive_* with a dynamic template for fields ending with _keyword and settings for 3 shards and 1 replica.
- Create an index, comprehensive_test, and index documents with a mix of predefined fields and unknown fields ending with _keyword.
- Retrieve the mappings and settings for comprehensive_test.
- Discuss the benefits of combining these Elasticsearch features in real-world use cases.

- Mapping parameters -

Mapping Parameters - Overview

Controlling Document Analysis and Storage

Description: Mapping parameters allow you to define how a field should be indexed and searched. They provide granular control over the indexing process.

M

Mapping Parameters

The copy_to Parameter

Description:
The **copy_to** parameter allows you to copy the values of one field to another. Useful for creating custom full-text search fields or consolidating data.

Use Case: Searching for a full name using a combined field instead of querying individual first and last name fields.

```
{
 "properties": {
  "first_name": {
   "type": "text",
   "copy_to": "full_name"
  },
  "last_name": {
   "type": "text",
   "copy_to": "full_name"
  },
  "full_name": {
   "type": "text"
  }
 }
}
```

M

Mapping Parameters

copy_to Parameter - Expected Outcome

Given data:

```
{
  "first_name": "John",
  "last_name": "Doe"
}
```

Result of a Search on full_name:

```
{
  "hits": {
    "hits": [{
      "_source": {
        "first_name": "John",
        "last_name": "Doe",
        "full_name": "John Doe"
      }
    }]
  }
}
```

M

Mapping Parameters

The boost Parameter

Description:
The boost parameter allows you to make one field more relevant than another, influencing scoring in search results

Use Case: Making sure that matches in the title field are considered more relevant than matches in the description field.

```
{
 "properties": {
  "title": {
   "type": "text",
   "boost": 2
  },
  "description": {
   "type": "text"
  }
 }
}
```

M

Mapping Parameters

The boost Parameter

Given data:
Indexed two documents:

1. {"title": "Elasticsearch Guide", "description": "A comprehensive guide to Elasticsearch"}
2. {"title": "Database Systems", "description": "This book includes a chapter on Elasticsearch"}

Result of a Search with the term "Elasticsearch":
Document with "Elasticsearch Guide" in the title will rank higher in search results than the second one, even if both documents contain the term "Elasticsearch".

Observation: The boost value of 2 on the title field makes matches in the title more relevant than in the description.

M

The null_value Parameter

Handling NULLs

Description:
The null_value parameter provides a way to replace null values with a specified default when indexing.

Use Case: Instead of ignoring or rejecting a document with a null status field, it will index it with the value "NOT_AVAILABLE".

```
{
  "properties": {
    "status": {
      "type": "keyword",
      "null_value": "NOT_AVAILABLE"
    }
  }
}
```

M

Mapping Parameters

Please review other Mapping parameters:
https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html

- Custom Field Mappings -

M

Custom Field Mappings - Introduction

Defining Your Field’s Characteristics

Description:
Custom field mappings give you control over how fields in your index are interpreted, indexed, and then searched.

Why Use Custom Field Mappings?

- Precise Control: Define how each field should be analyzed.
- Optimized Searches: Customize field properties for specific search requirements.
- Consistent Results: Enforce certain data types and formats for fields.

M

Custom Field Mappings

How to Define Custom Field Mappings

Description:

- Field Type: Decide the type of the field (e.g., text, date, keyword, etc.).
- Analyzer: Define or choose an analyzer for text processing.
- Special Properties: Set properties like boost, doc_values, or custom formats.

Custom Field Mapping - Example

Objective: Index a date field with a custom date format.

Output Usage: This allows for indexing dates like "12-05-2021" or as epoch milliseconds.

```
PUT /custom_date_index
{
  "mappings": {`
    "properties": {
      "custom_date": {
        "type": "date",
        "format": "dd-MM-yyyy||epoch_millis"
      }
    }
  }
}
```

M

Custom Field Mappings

Testing Custom Field Mappings

Description:
Use the _analyze endpoint or index sample data to see how custom field mappings affect indexing and searching. For the date example:

```
POST /custom_date_index/_doc/1
{
  "custom_date": "12-05-2021"
}
```

Expected Output:

The date is stored internally as a Unix timestamp, but when retrieved, it's presented in the provided format ("12-05-2021").

Custom Field Mappings

Implementing Custom Field Mappings

Description:
When defining an index, include custom field mappings within the "mappings" section. This can be done at index creation or modified later using the _reindex functionality (if changing existing data).

- Multi-fields - An Introduction -

Getting More Out of Single Fields

Multi-fields - An Introduction

Getting More Out of Single Fields

- In Elasticsearch, a field can be indexed in multiple ways.
- **Multi-fields** allow you to define secondary fields, meaning a single piece of text can be indexed as both a full-text text field and an exact value keyword field.

Why Multi-fields?

- **Flexibility**: Search the same piece of data in various ways.
- **Optimization:** Depending on the query's nature, different representations of the data can lead to more efficient searches.
- **Analysis:** Run different analyses on the same text. For example, a field could be tokenized for full-text search and not tokenized for sorting/aggregation.

- Before we continue… -

text vs. keyword - The Basics

L

Multi-fields - An Introduction

text vs. keyword - The Basics

- Elasticsearch offers various data types for text fields. The two primary ones are **text** and **keyword**.
- While both deal with textual data, they serve different purposes and have unique characteristics.

L

Multi-fields - An Introduction

Text - Optimized for Full-text Search

- **text fields** are designed for full-text searches, such as searching within articles, descriptions, and comments.
- **They are analyzed upon indexing**, which means the field's content is processed by an analyzer to produce tokens. These tokens are what's actually indexed.

Characteristics:

- **Tokenization**: Breaks down text into individual tokens (typically words).
- **Lowercasing**: Converts all characters to lowercase for case-insensitive search.
- **Stop Words**: Common words like "and", "or", and "the" might be removed to improve search efficiency.
- **Not Suitable for Sorting/Aggregations: Because of analysis.**

L

Multi-fields - An Introduction

keyword Field Type - Optimized for Exact Matches

- keyword fields are designed for exact value searches, like filtering by tags, email addresses, or status values.
- They are not analyzed, which means the content is indexed as-is.

Characteristics:

- **No Tokenization**: Indexed as a whole.
- **Case-Sensitive**: "BLUE" is different from "blue".
- **Perfect for Sorting & Aggregations**: As they hold the exact data.
- **Ideal for Structured Content**: Like IDs, status codes, email addresses, and more.

L

Multi-fields - An Introduction

keyword vs text - Why Does It Matter?

Use-cases & Implications

- **Full-Text Search:** If you want to find documents containing a certain word or phrase, use a text field.
- **Exact Matching:** Filtering by specific tags, statuses, or other exact values? Use a keyword.
- **Aggregations:** Running aggregations (like counts, sums, or averages) on textual data? You'll need a keyword field type.
- **Sorting:** Sorting documents based on a textual field? Again, a keyword is your go-to

L

Multi-fields - An Introduction

keyword vs text - Practical Implication

Imagine a dataset of products with a "description" field:

- Using a text type, users can search for products containing specific words or phrases in their description.

- Using a keyword (like description.keyword), users can aggregate data based on exact descriptions or sort products by their exact description.

- Back to Multi-Fields-

M

Multi-fields

Setting Up Multi-fields

This configuration indexes the name field as full-text and allows for exact value searches using name.raw.

```
{
 "properties": {
  "name": {
   "type": "text",
   "fields": {
    "raw": {
     "type": "keyword"
    }
   }
  }
 }
}
```

M

Multi-fields
Quick lab - Querying Multi-fields

Write a query that Use the primary and secondary fields using:

- Full-text search: name: "John Doe"
- Exact value search (e.g., for sorting/aggregation): name.raw: "John Doe"

```
{
 "properties": {
  "name": {
   "type": "text",
   "fields": {
    "raw": {
     "type": "keyword"
    }
   }
  }
 }
}
```

M

Multi-fields

Real-world Scenario - Blog Posts

Imagine an application with blog posts:

- **Title**: Text that should be searchable but also available for exact matching, sorting, and aggregations.
- **Content**: Mainly for full-text search, but might need a non-analyzed version for specific use-cases.

Now, you can run full-text searches on title and content, but also sort blog posts by title.exact or run aggregations on content.verbatim.

```
{
  "properties": {
    "title": {
      "type": "text",
      "fields": {
        "exact": {
          "type": "keyword"
        }
      }
    },
    "content": {
      "type": "text",
      "fields": {
        "verbatim": {
          "type": "keyword"
        }
      }
    }
  }
}
```

- Custom Analyzers -
Tailoring Text Analysis

A

Custom Analyzers - Tailoring Text Analysis

Beyond Built-in Analyzers - recap

- **While** Elasticsearch offers a rich set of built-in analyzers, there are use-cases where a custom touch is required.

- **Custom** analyzers provide granular control over the text analysis process.

A

Why Custom Analyzers?

Catering to Specific Needs

- **Language-Specific Processing**: While Elasticsearch supports many languages, your specific requirements might necessitate custom tweaks.
- **Domain-Specific Content**: For specialized content like medical journals or legal documents.
- **Optimized Search Experience**: Improve search relevance by indexing content in ways that match user query patterns.

A

Anatomy of an Analyzer

Catering to Specific Needs

- **Tokenizer**: Breaks text into tokens (like words or terms).
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html

- **Char Filters**: Pre-process the text, e.g., removing HTML tags or converting symbols.
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-charfilters.html

- **Token Filters**: Post-process the tokens, e.g., lowercasing or stemming.
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html

- EXAMPLES -

A

Tokenizers - The whitespace Tokenizer

Breaking Text on Whitespace

This tokenizer divides text at whitespace characters.

```
POST _analyze
{
  "tokenizer": "whitespace",
  "text": "Elasticsearch is fun!"
}
```

Output

```
{
  "tokens": [
    {"token": "Elasticsearch"},
    {"token": "is"},
    {"token": "fun!"}
  ]
}
```

A

Tokenizers - The ngram Tokenizer

Generating Small Text Fragments

This tokenizer can break text into fragments with min and max lengths - Great for autocomplete (TBD)

```
POST _analyze
{
  "tokenizer": {
    "type": "ngram",
    "min_gram": 2,
    "max_gram": 3,
    "token_chars": ["letter", "digit"]
  },
  "text": "ABCD"
}
```

Output

```
{
  "tokens": [
    {"token": "AB"},
    {"token": "ABC"},
    {"token": "BC"},
    {"token": "BCD"},
    {"token": "CD"}
  ]
}
```

A

Char Filters - The mapping Char Filter

Replacing Custom Patterns

This char filter replaces specified patterns

```
POST _analyze
{
 "char_filter": [{
   "type": "mapping",
   "mappings": ["&=>and"]
 }],
 "text": "Fish & Chips"
}
```

Output

```
Fish and Chips
```

A

Char Filters - The html_strip Char Filter

This char filter removes HTML tags from the text.

This char filter removes HTML tags from the text

```
POST _analyze
{
  "char_filter": ["html_strip"],
  "text": "<b>Elasticsearch</b> is <i>fun</i>!"
}
```

Output

```
Elasticsearch is fun!
```

A

Token Filters - The lowercase Token Filter

Making Tokens Lowercase

This filter converts tokens to lowercase

```
POST _analyze
{
  "tokenizer": "whitespace",
  "filter": ["lowercase"],
  "text": "ElasticSearch Is Fun!"
}
```

Output

```
{
  "tokens": [
    {"token": "elasticsearch"},
    {"token": "is"},
    {"token": "fun!"}
  ]
}
```

A

Token Filters - The stemmer Token Filter

Reducing Tokens to Their Root Form

This filter uses an algorithm to transform words to their root form.

```
POST _analyze
{
  "tokenizer": "whitespace",
  "filter": [{"type": "stemmer", "name": "english"}],
  "text": "running runner runs"
}
```

Output

```
{
  "tokens": [
    {"token": "run"},
    {"token": "runner"},
    {"token": "run"}
  ]
}
```

- Building a Custom Analyzer -

A

Building Our Own Custom Analyzer

```
{
  "settings": {
    "analysis": {
      "tokenizer": {
        "my_custom_tokenizer": {
          "type": "standard",
          "max_token_length": 5
        }
      },
      "char_filter": {
        "my_custom_char_filter": {
          "type": "mapping",
          "mappings": ["& => and"]
        }
      },
      "analyzer": {
        "my_custom_analyzer": {
          "type": "custom",
          "tokenizer": "my_custom_tokenizer",
          "char_filter": ["my_custom_char_filter"],
          "filter": ["lowercase"]
        }
      }
    }
  }
}
```

This example creates a custom analyzer named my_custom_analyzer.

A

Building Our Own Custom Analyzer

Testing Custom Analyzers

```
POST _analyze
{
  "analyzer": "my_custom_analyzer",
  "text": "The QUICK Brown Fox & Jumped"
}
```

Observe how Elasticsearch tokenizes and analyzes the given text using your custom analyzer - Run it

A

Building Our Own Custom Analyzer

Implementing in Index/Mapping

When defining mappings, specify the custom analyzer:

```
{
  "mappings": {
    "properties": {
      "content": {
        "type": "text",
        "analyzer": "my_custom_analyzer"
      }
    }
  }
}
```

Real-World Scenario - Medical Documents

Consider a medical document repository:

- Tokenize medical terms without splitting them (e.g., "T-cell").
- Convert common abbreviations (e.g., "Dr." to "Doctor").
- A custom analyzer can ensure accurate and relevant search results.

Building Our Own Custom Analyzer

Best Practices

- **Test Extensively**: Ensure your custom analyzer behaves as expected.
- **Monitor Performance**: Custom analyzers can affect indexing speed.
- **Version Control**: As your requirements evolve, your analyzers might too. Track changes.

- Summary lab -

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON

Objective:

- To familiarize yourself with the creation and utilization of a multi-field combined with a custom analyzer in Elasticsearch. By the end of this lab, you will have an index with specialized text processing capabilities and will execute searches to observe the effects of your setup.

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Create an Index with Multi-Field & Custom Analyzer:
Instructions:

1. Begin by defining a custom analyzer. This analyzer should:
    a. Replace ampersands (&) with the word "and".
    b. Use a whitespace tokenizer that operates on tokens of a maximum length of 5 characters.
    c. Apply a lowercase filter to tokens.

2. Once your custom analyzer is ready, create an index mapping for a field named title.
   This field should:
    a. Use the custom analyzer you defined.
    b. Additionally, introduce a multi-field named raw of type keyword.

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Index a Sample Movie
Instructions:

1. Index a movie with the title "Fish & Chips: A British Tale" into the index you created.
2. Think about the transformations this title might undergo due to your custom analyzer.

Execute Searches
Instructions:

1. Perform a search using the term "Fish and Chips". Consider how the custom analyzer might affect your search results.
2. Execute an aggregation on the title.raw field. Reflect on what you expect to see in the aggregation results given the multi-field setup.

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Reflection Points (Post-Lab Discussion):

1. How did the custom analyzer influence the indexed data and your search results?

2. Why might combining both text and keyword types in one field (using multi-fields) be beneficial in certain scenarios?

- Solutions -

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Solution: Create an Index with Multi-Field & Custom Analyzer

```
PUT /movies_custom
{
    "settings": {
        "analysis": {
            "char_filter": { "replace_ampersands": { "type": "mapping", "mappings": ["&=>and"] } },
            "tokenizer": { "custom_whitespace": { "type": "whitespace", "max_token_length": 5 } },
            "analyzer": { "custom_analyzer": { "type": "custom", "tokenizer": "custom_whitespace", "char_filter": ["replace_ampersands"], "filter": ["lowercase"]
} }
        }
    },
    "mappings": { "properties": { "title": { "type": "text", "analyzer": "custom_analyzer", "fields": { "raw": { "type": "keyword" } } } } }
}
```

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

This setup has a custom analyzer that replaces ampersands, uses a custom whitespace tokenizer, and applies a lowercase filter. The "title" field has both a text and keyword type through multi-fields.

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Solution: Index a Sample Movie

```
POST /movies_custom/_doc/1
{
  "title": "Fish & Chips: A British Tale"
}
```

Solution: Index a Sample Movie

Search for the term "Fish and Chips" and observe the match.

```
GET /movies_custom/_search
{
  "query": {
    "match": {
      "title": "Fish and Chips"
    }
  }
}
```

A

Lab Instruction - Crafting a Multi-Field with Custom Analyzer

FULL HANDS-ON -

Aggregate on the title.raw field to see exact values

```
GET /movies_custom/_search
{
  "aggs": {
    "distinct_titles": {
      "terms": {
        "field": "title.raw"
      }
    }
  }
}
```