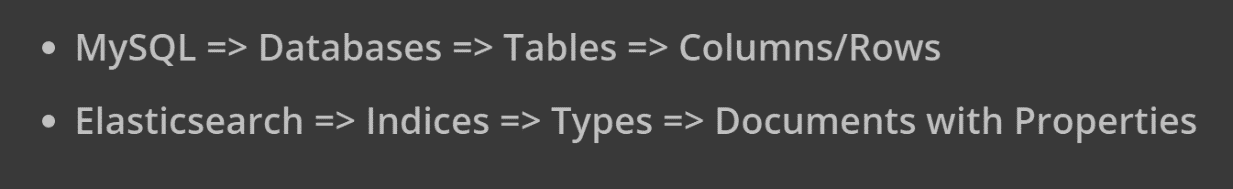
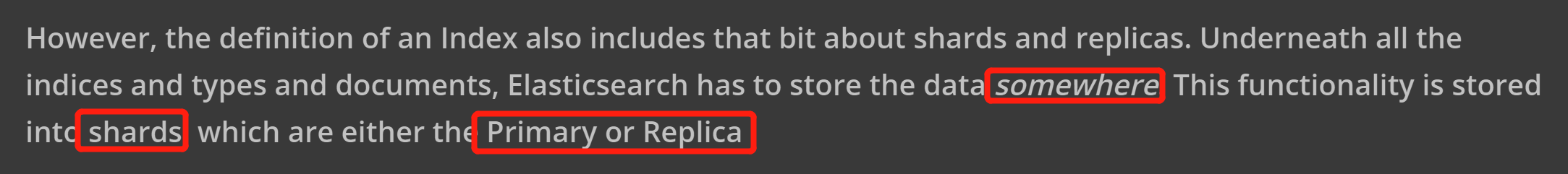
**ES fundamental**

# **What is ES index**

ES index is most important basic part of ES cluster. An **index** is like a ‘**database’** in a relational database. It has a mapping which defines **multiple types**. An index is a logical namespace which maps to **one or more primary shards** and can have **zero or more replica shards(For cluster mode)**.



Indices are fairly lightweight data organization mechanisms, so Elasticsearch will happily let you create hundreds of indices.

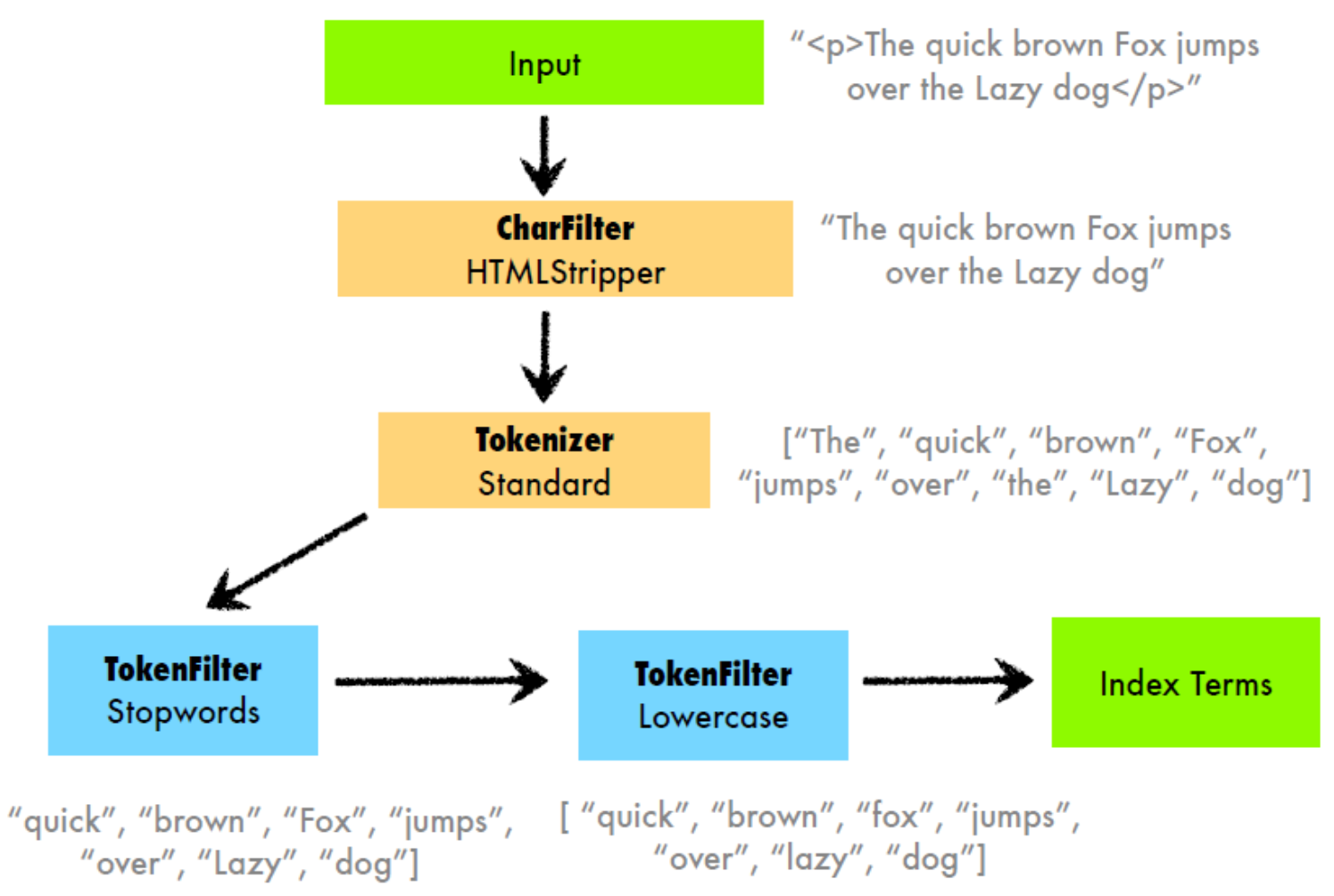


What ES index can do?

For example, if we can build a chat website like wechat, we have to store users’ information into database, each user can have their own features like hobbies, friends, working information etc.(we can also write these info into relational database, but someone can have their own new features, if we just store to databases, we have to make many columns to represent these, also with some features just one people have, others are just **Non**), but we can build ES **index for each person(**each person can have their own features as key in json body, diversity**).**

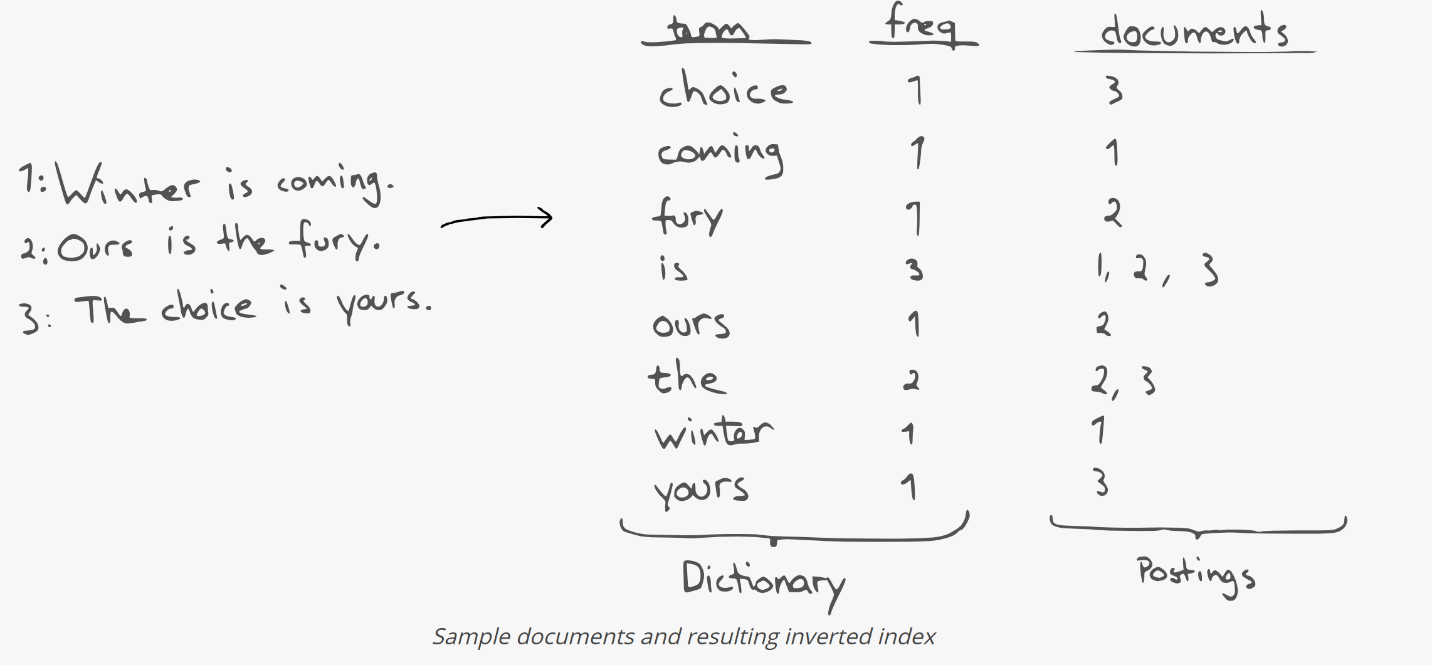
We can also use ES index to split data into different parts for cluster.

**General step for ES index:**



# **ES inverted index**

Let's say we have these three simple documents: "Winter is coming.", "Ours is the fury." and "The choice is yours.". After some simple **text processing** (lowercasing, removing punctuation and splitting words), we can construct the "**inverted index**" shown in the figure.



The inverted index maps ***terms*** to **documents** (and **possibly positions** in the documents) containing the term. Since the terms in the ***dictionary*** are **sorted**, we can quickly find a term, and **subsequently** its **occurrences** in the *postings*-structure.

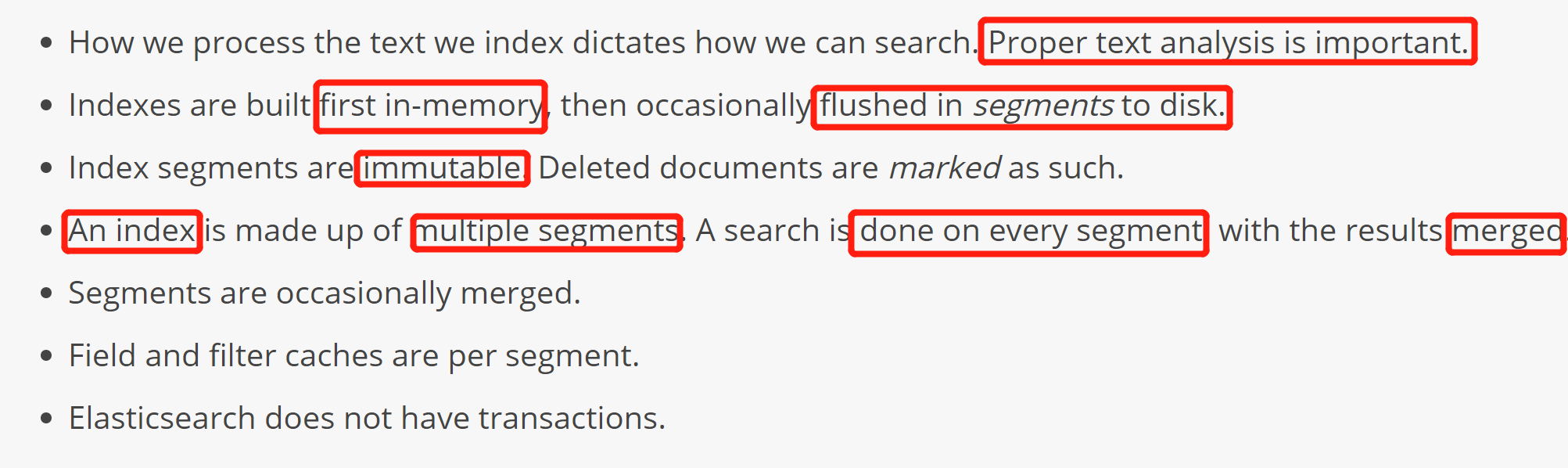
A simple search with **multiple terms** is then done by looking **up all the terms** and **their occurrences**, and take the **intersection** (for AND searches) or the **union** (for OR searches) of the sets of occurrences to get the **resulting list** of documents. More complex types of queries are obviously more elaborate, but the approach is the same: first, **operate** on the **dictionary** to **find candidate terms**, then on the **corresponding occurrences**, **positions**, etc.

In other words, we can efficiently find things given term ***prefixes***. When all we have is an inverted index, we want everything to look like a **string prefix** problem.

For example:

(1). To find everything *ending* with "tastic", we can index the ***reverse*** (e.g. **"fantastic" → "citsatnaf"**) and search for everything starting with "citsat".

(2) inding substrings often involves splitting terms into smaller terms called "**n-grams**". For example, "yours" can be split into "^yo", "you", "our", "urs", "rs$", which means we would get occurrences of "ours" by searching for "our" and "urs"(3-grams).

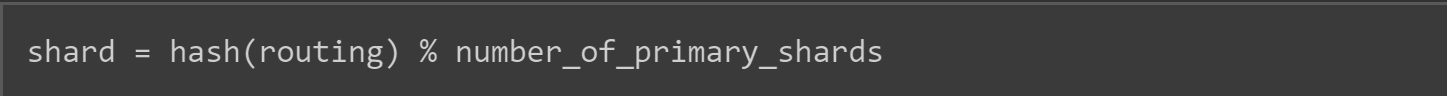


# **Distributed Document store**

ES is a cluster based system for document retrieved, so we can scale the cluster horizontally and can store as many documents in this distributed system.

## **3.1.Route Document to shard**

When you index a document, it is stored on a **single primary shard**. How does Elasticsearch know which **shard** a document belongs to? When we create a new document, how does it know whether it should store that document on **shard 1 or shard 2**? The process **can’t be random**, since we may need to **retrieve** the document in the future. In fact, it is determined by a simple **formula**:



The **routing** value is an **arbitrary string(document string name or custom value)**, which defaults to the document’s  \_id but can also be set to a custom value. This routing string is passed through a **hashing function** to **generate** **a number**, which is **divided** by the **number of primary shards(How many primary shards needed)** in the index to return the ***remainder***. The remainder will always be in the **range 0 to number\_of\_primary\_shards - 1**, and gives us the number of the shard **where a particular document lives**.

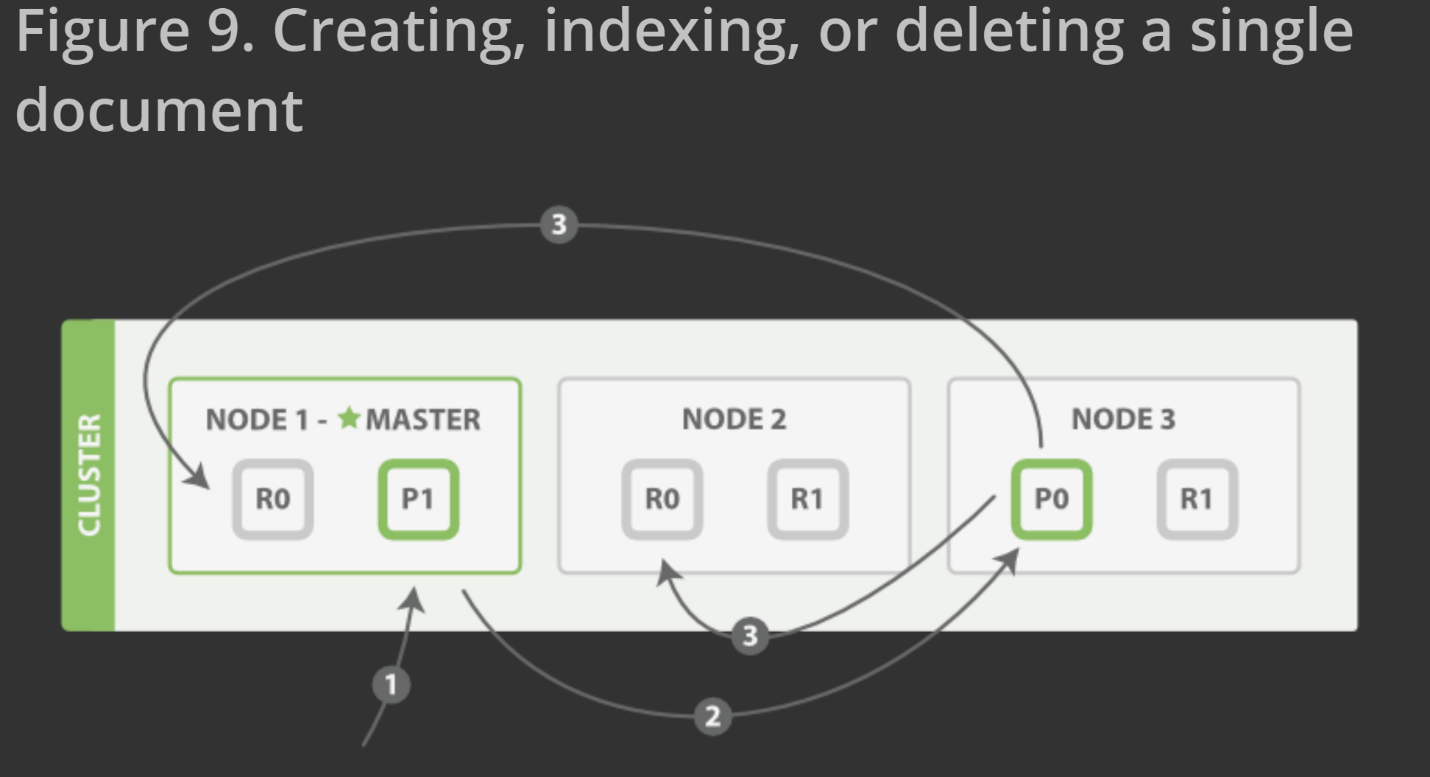
## **3.2.Primary and Replica shards interact**

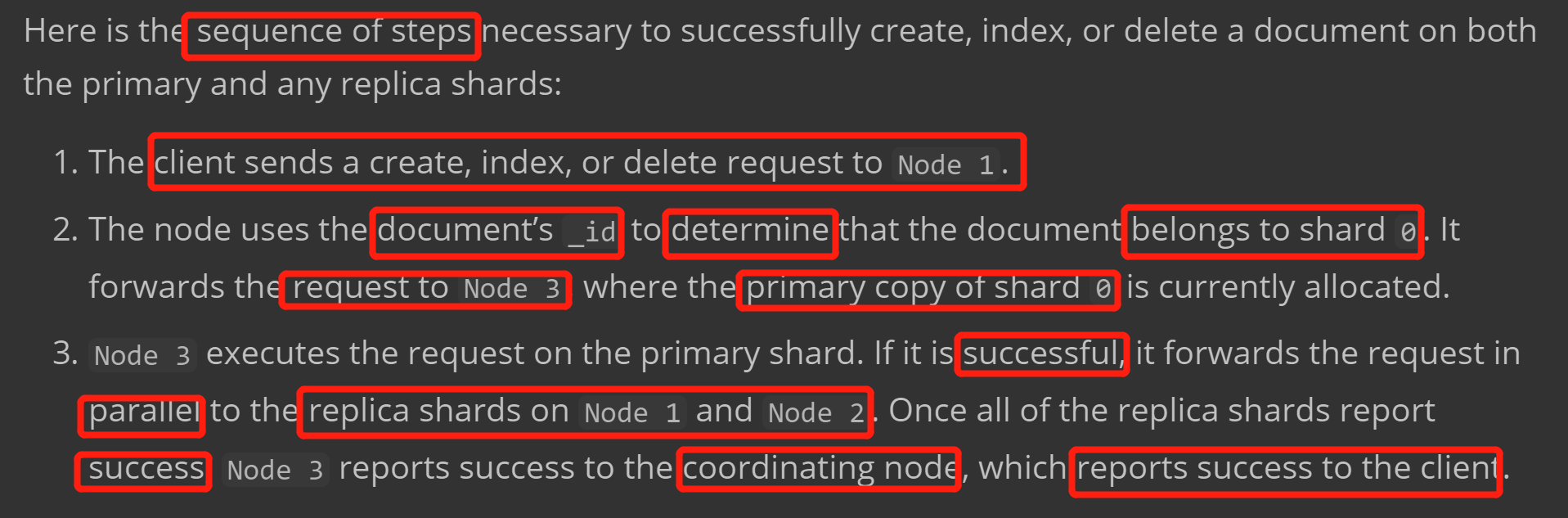
For explanation purposes, let’s imagine that we have a cluster consisting of **three nodes**. It contains **one index** called blogs that has **two primary shards**. Each primary shard **has two replicas**:



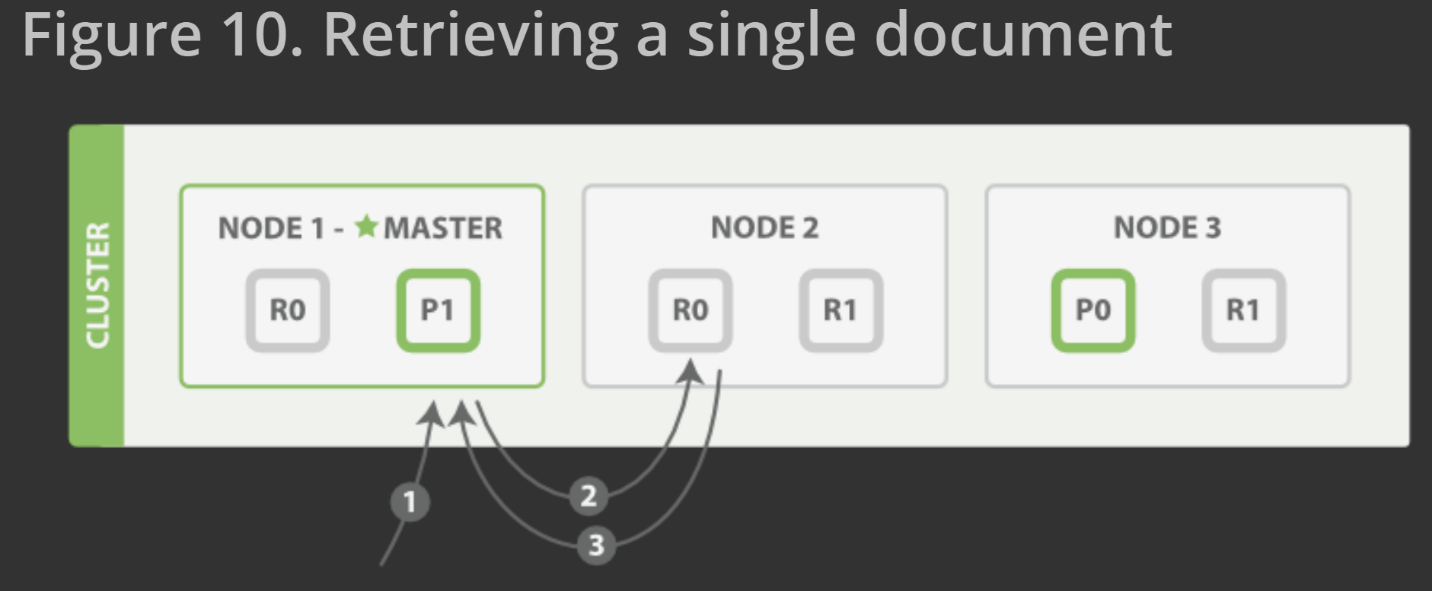
We can send our **requests** to **any node** in the cluster. **Every node** is fully capable of **serving any request**. **Every node** knows the **location of every document** in the cluster and so can forward requests **directly** to the **required** node. In the following examples, we will send **all** of our **requests to Node 1**, which we will refer to as the ***coordinating node***.

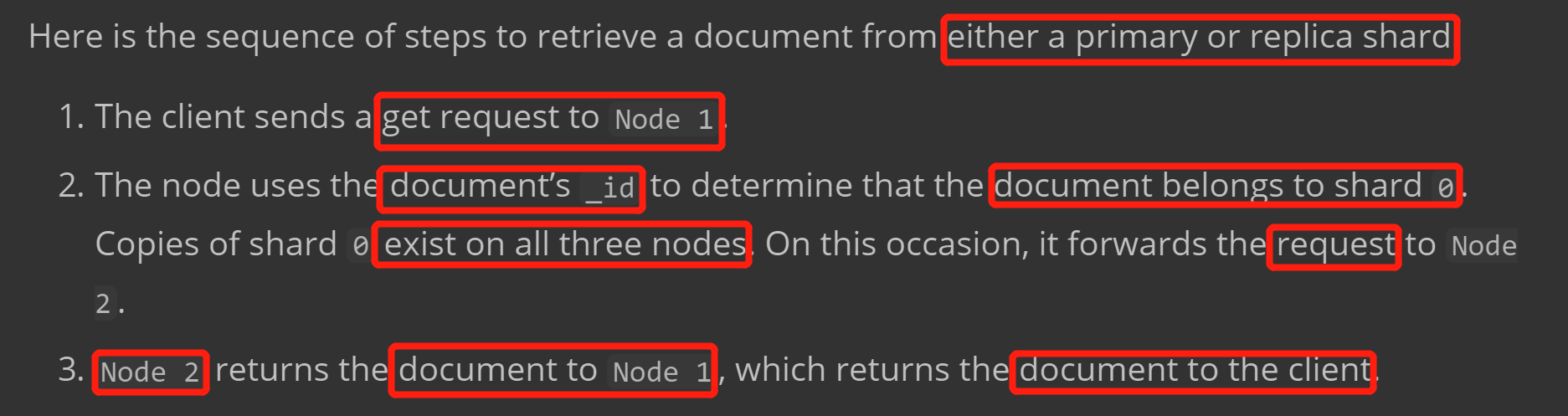
## **3.3.Document index, delete**





## **3.4.Retrieve one Document**



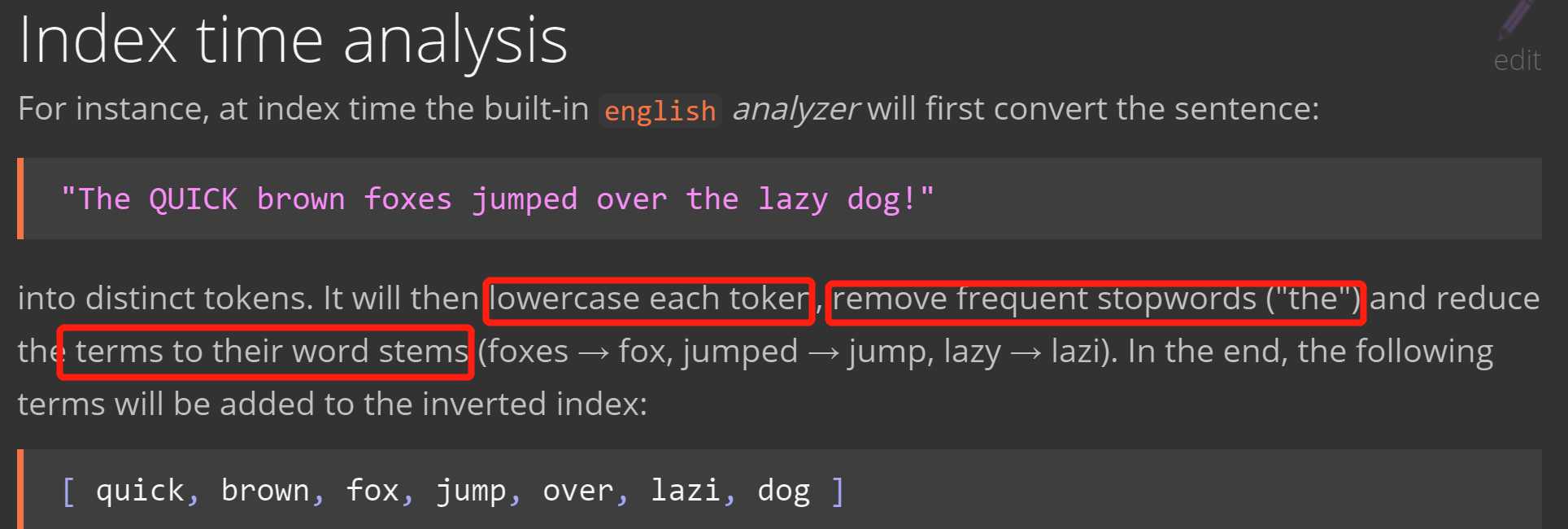


# **4.Anaylsis**

## **4.1.Fundamental content**

Analysis is performed by an [*analyzer*](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html) which can be either a **built-in analyzer** or a [**custom**](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-custom-analyzer.html)**analyzer** defined per index.

Basic example:



An *analyzer*  — whether built-in or custom — is just a package which contains three lower-level building blocks: ***character filters***, ***tokenizers***, and ***token filters***.

Character filters eg: convert (012) to (onetowthree) …

Tokenizer eg: convert ‘hello world’ to terms [hello, world] …

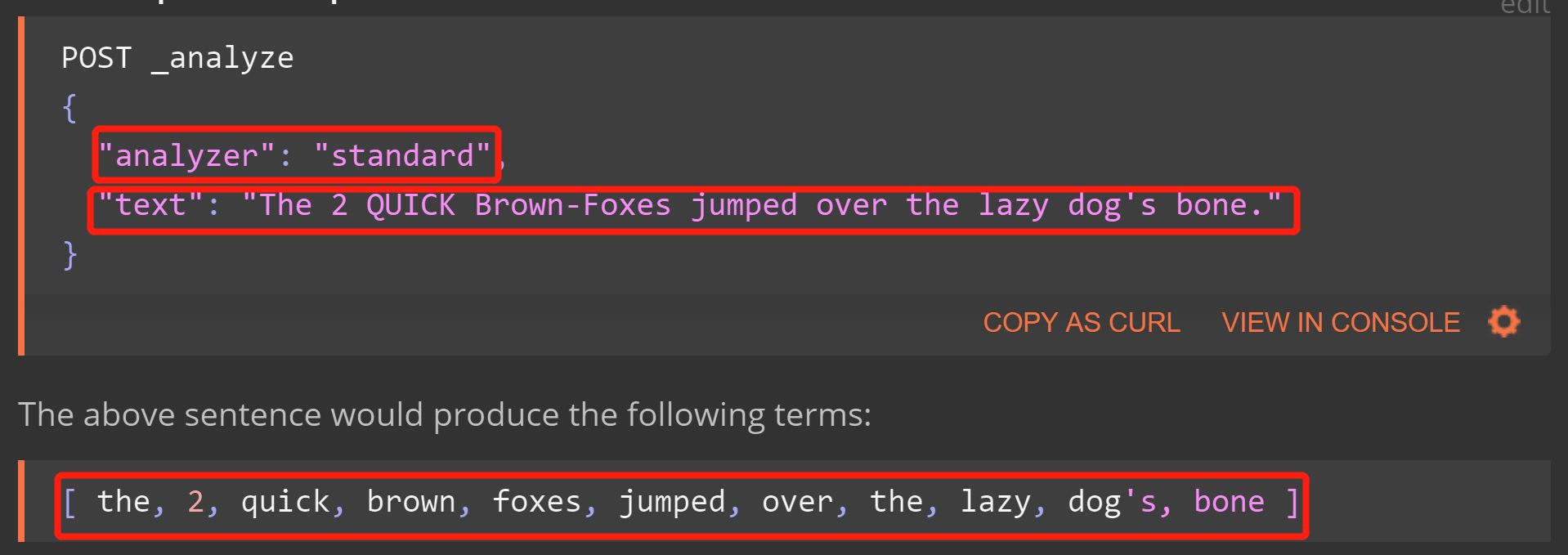
Token filters eg: convert ‘Hello World’ to ‘hello world’ or ‘the best person’ to ‘best person’ …

There are so many build-in analyzers that we can use to convert, filter documents. Link can be found in Appendix.

## **4.2.Use case Example**

**Build-in analysis** use case eg:

By using **Standard Analyzer**(remove stopwords, split sentence, lowercase words..), just write the configure json file(like source code **setting.json**).



There are also other types analyzer like: Configuring built-in analyzers, Standard Analyzer, Simple Analyzer, Whitespace Analyzer, Stop Analyzer, Keyword Analyzer, Pattern Analyzer, Language Analyzers, Fingerprint Analyzer(Can be found bellow link).

**Customer** **analysis** eg:

Character Filter

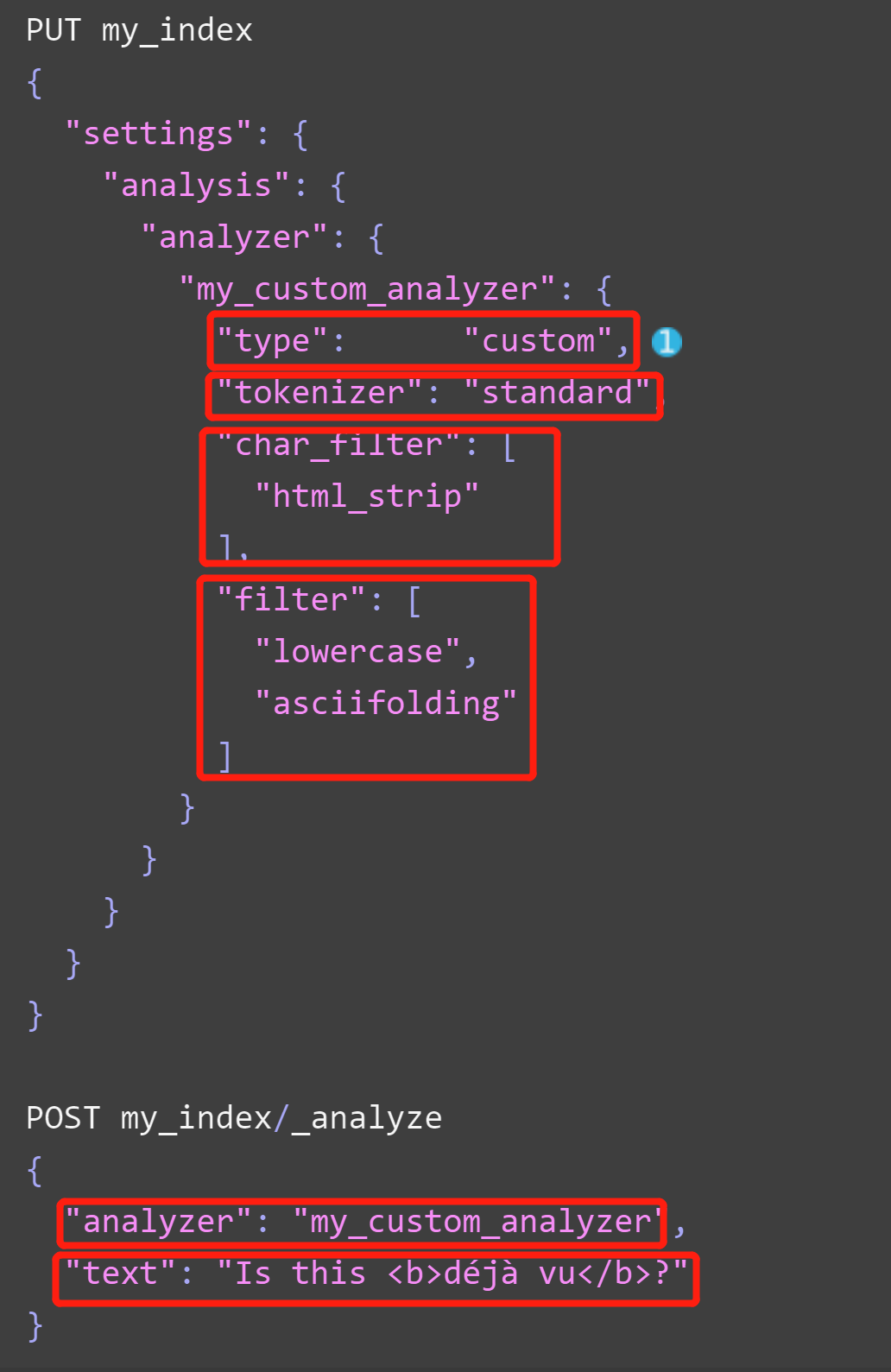
* HTML Strip Character Filter

Tokenizer

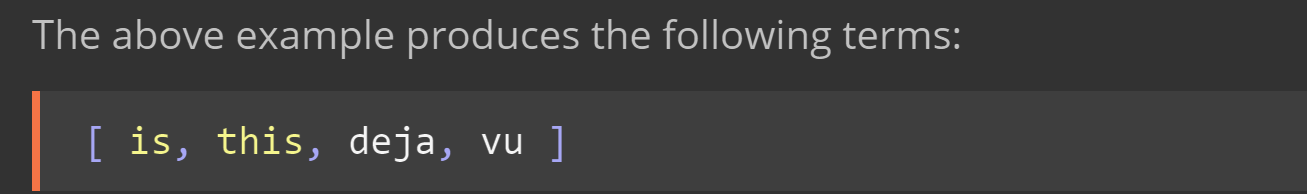
* Standard Tokenizer

Token Filters

* Lowercase Token Filter
* ASCII-Folding Token Filter



Returned result:

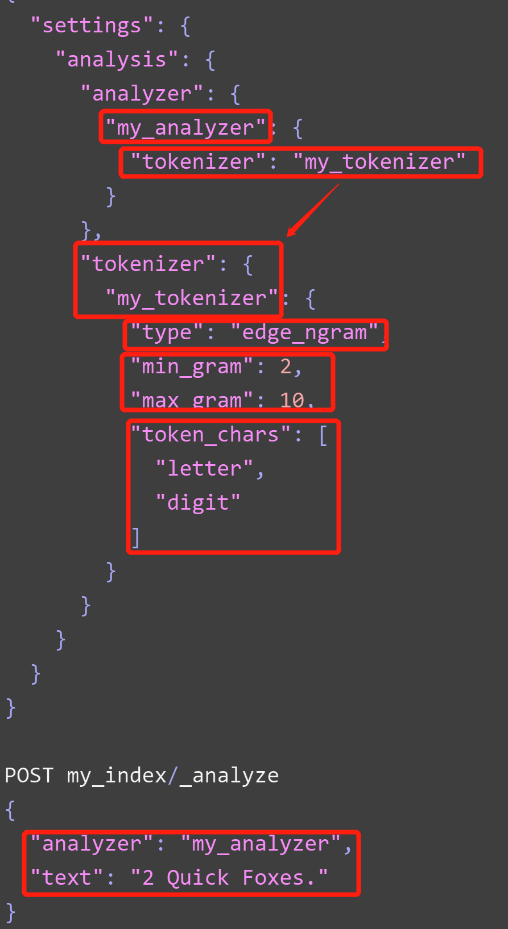


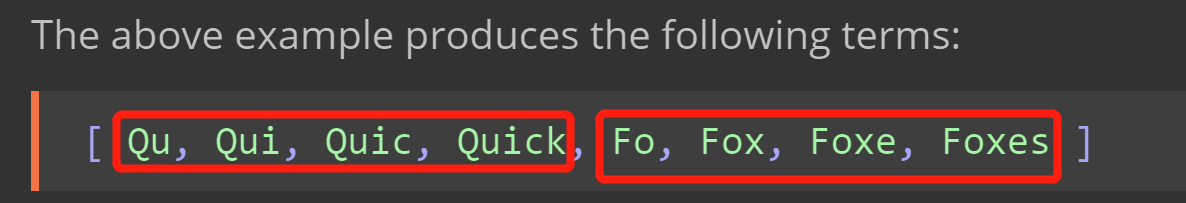
## **4.3.Tokenizer, Tokenizer filter & Character filter**

### **4.3.1.Tokenizer**

There are also many tokenizer can be used like Letter Tokenizer, N-gram Tokenizer, keyword Tokenizer etc. (link: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html>)

Edge-gram eg:

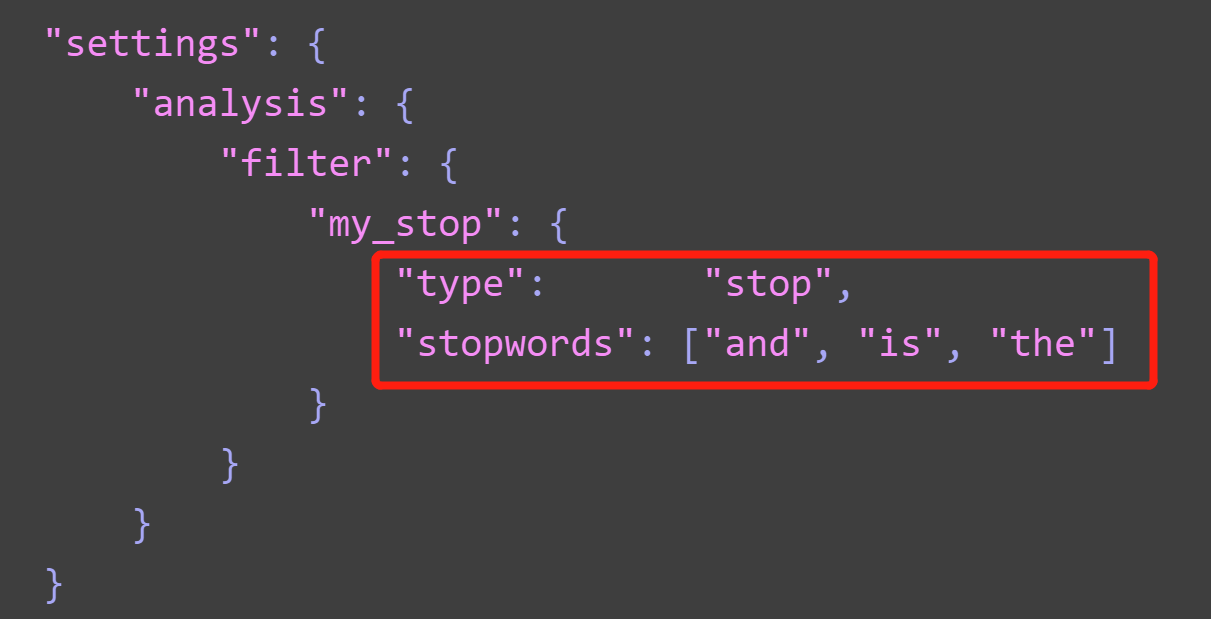


Return result:  


### **4.3.2.Tokenizer filters**

Tokenizer filters mainly focus on changing token words(lower, upper..), stop words remover, stemming token, etc.

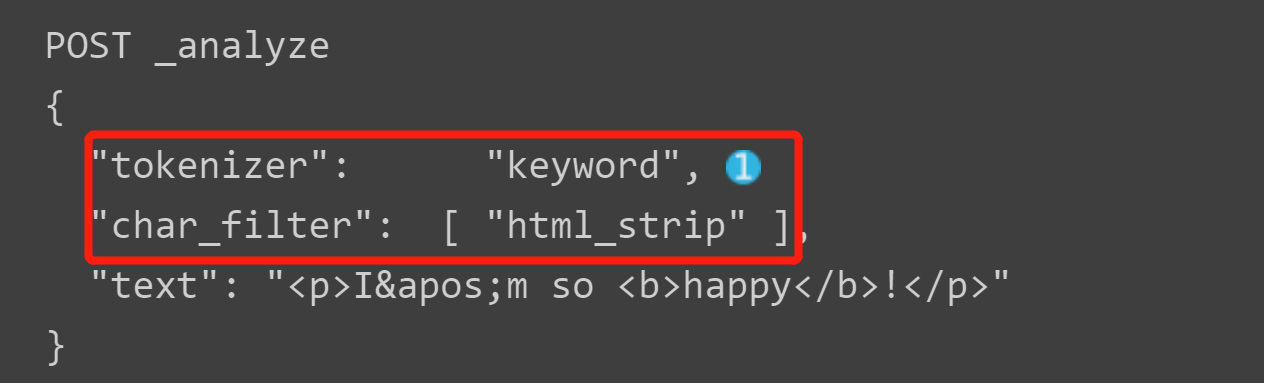
Stop words token filter eg:



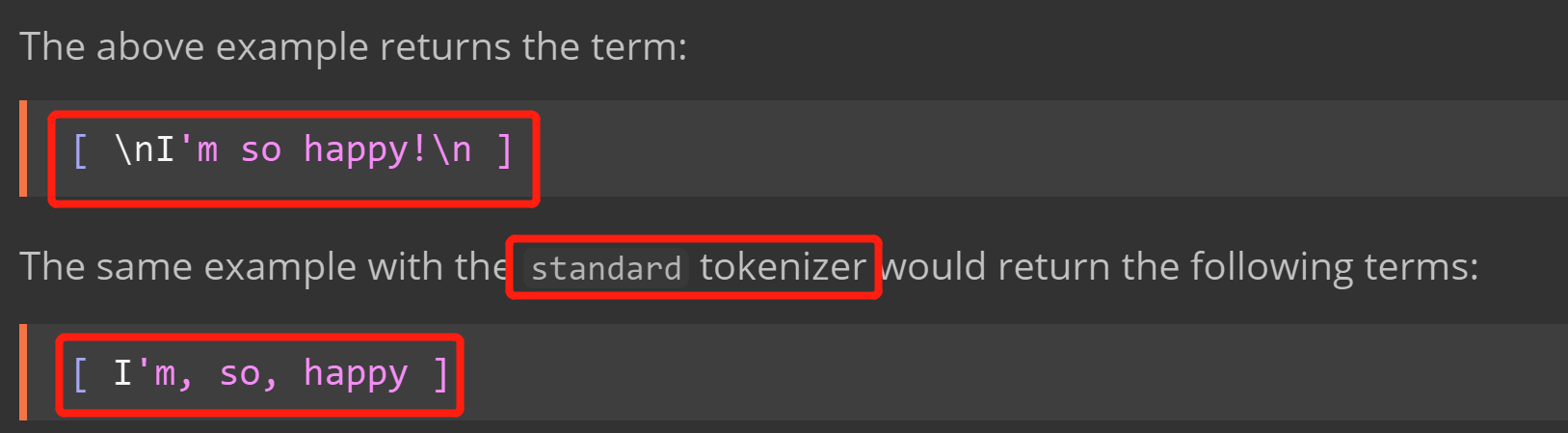
### **4.2.3.Character filters**

There are 3 main build-in character filter: HTML Strip Char Filter,Mapping Char Filter,Pattern Replace Char Filter.

HTML strip char eg:



Returned result:



**Appendix**:

From bottom to up: <https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up#inverted-indexes-and-index-terms>

More insight for ES: <https://www.elastic.co/blog/found-elasticsearch-top-down>

What is index: <https://www.elastic.co/blog/what-is-an-elasticsearch-index>

Distributed Document store: <https://www.elastic.co/guide/en/elasticsearch/guide/current/distributed-docs.html>

Analysis ES document: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html>