[**https://plumbr.io/handbook/garbage-collection-algorithms-implementations/concurrent-mark-and-swee**](https://plumbr.io/handbook/garbage-collection-algorithms-implementations/concurrent-mark-and-sweep)**p**

[**https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/cms.html**](https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/cms.html)

**Starting a Concurrent Collection CycLE**

**With the serial collector a major collection occurs whenever the tenured generation becomes ful**l and all application threads are stopped while the collection is done. In contrast, **the start of a concurrent collection must be timed such that the collection can finish before the tenured generation becomes full; otherwise, the application would observe longer pauses due to concurrent mode failure. There are several ways to start a concurrent collection.**

Based on recent history, the CMS collector maintains estimates of the time remaining before the tenured generation will be exhausted and of the time needed for a concurrent collection cycle. Using these dynamic estimates, a concurrent collection cycle is started with the aim of completing the collection cycle before the tenured generation is exhausted. These estimates are padded for safety, because concurrent mode failure can be very costly.

A concurrent collection also starts if the occupancy of the tenured generation exceeds an initiating occupancy (a percentage of the tenured generation). The default value for this initiating occupancy threshold is approximately 92%, but the value is subject to change from release to release. This value can be manually adjusted using the command-line option -XX:CMSInitiatingOccupancyFraction=<N>, where <N> is an integral percentage (0 to 100) of the tenured generation size.

For elasticsearch - CMSInitiatingOccupancyFraction - 75% of tenured generation size

The concurrent collection cycle typically includes the following steps:

* Stop all application threads, identify the set of objects reachable from roots, and then resume all application threads.
* Concurrently trace the reachable object graph, using one or more processors, while the application threads are executing.
* Concurrently retrace sections of the object graph that were modified since the tracing in the previous step, using one processor.
* Stop all application threads and retrace sections of the roots and object graph that may have been modified since they were last examined, and then resume all application threads.
* Concurrently sweep up the unreachable objects to the free lists used for allocation, using one processor.
* Concurrently resize the heap and prepare the support data structures for the next collection cycle, using one processor.

**Concurrent Mode Failure**

The CMS collector uses one or more garbage collector threads that run simultaneously with the application threads with the goal of completing the collection of the tenured generation before it becomes full. As described previously, in normal operation, the CMS collector does most of its tracing and sweeping work with the application threads still running, so only brief pauses are seen by the application threads. However, if the CMS collector is unable to finish reclaiming the unreachable objects before the tenured generation fills up, or if an allocation cannot be satisfied with the available free space blocks in the tenured generation, then the application is paused and the collection is completed with all the application threads stopped. The inability to complete a collection concurrently is referred to as *concurrent mode failure* and indicates the need to adjust the CMS collector parameters. If a concurrent collection is interrupted by an explicit garbage collection (System.gc()) or for a garbage collection needed to provide information for diagnostic tools, then a concurrent mode interruption is reported.

**Elasticsearch properties**

The Concurrent Mark Sweep (CMS) collector is designed for applications that prefer shorter garbage collection pauses and that can afford to share processor resources with the garbage collector while the application is running. Typically applications that have a relatively large set of long-lived data (a large tenured generation) and run on machines with two or more processors tend to benefit from the use of this collector. However, this collector should be considered for any application with a low pause time requirement. The CMS collector is enabled with the command-line option -XX:+UseConcMarkSweepGC.

**Has a large tenured generation - Inverted index lot of objects in memory which will survive for very long - so CMS collector is required for efficient tenured generation cleaning. 😻😻😻😻🤑**

**Log rolling policies - Time based, Size Based**

Upsert Example Elasticsearch  😍😍😍😍😍

-------------------------------------------

AFAIK, what you put in "upsert" is done only if the document doesn't exist.

For example, if you simply want to add a new field to an existing doc, this

should work:

curl -XPOST localhost:9200/test/test/1/\_update -d '

{

"script": "ctx.\_source.field1 = "value1"",

}'

But that would fail if the document is missing. So just in case it's not

there, you can add an initial doc in the upsert section. For example:

curl -XPOST localhost:9200/test/test/3/\_update -d '

{

"script": "ctx.\_source.field1 = "value1"",

"upsert": {

"field1": "value1",

"field2": "value2"

}

}'

<https://www.elastic.co/guide/en/elasticsearch/reference/6.0/docs-bulk.html>

-----------------------------------------------------------------------------------

bulk format -  newline delimited json -   application/x-ndjson.

---------------------------------------------------------------------------

If you’re providing text file input to curl, you **must** use the --data-binary flag instead of plain -d. The latter doesn’t preserve newlines. Example:

A note on the format. The idea here is to make processing of this as fast as possible. As some of the actions will be redirected to other shards on other nodes, only action\_meta\_data is parsed on the receiving node side ?

Client libraries using this protocol should try and strive to do something similar on the client side, and reduce buffering as much as possible.

The response to a bulk action is a large JSON structure with the individual results of each action that was performed. The failure of a single action does not affect the remaining actions.

There is no "correct" number of actions to perform in a single bulk call. You should experiment with different settings to find the optimum size for your particular workload.

If using the HTTP API, make sure that the client does not send HTTP chunks, as this will slow things down.

------------------------------------------------------------------------------------------------------------------------------------------------

<https://www.elastic.co/blog/elasticsearch-sequence-ids-6-0>

Timeline

Description automatically generated

Global checkpoint is also for all shards. When a primary shard is down, new primary shard and all other active shards increment their global checkpoint, as well as their local checkpoints. When new primary comes up, different in global checkpoints is done and data replicated accordingly.

**The First Benefit: Faster Recovery**

We skipped over how the actual recovery process worked prior to Elasticsearch 6.0. When Elasticsearch recovers a replica after it has been offline, it has to make sure that that replica is identical with the active primary. Inactive shards have [**synced flush markers**](https://www.elastic.co/guide/en/elasticsearch/reference/6.0/indices-synced-flush.html) to quickly make this validation but shards with active indexing simply have no guarantees. If a shard goes offline while there is still active indexing, the new primary shard then copies Lucene segments (which are files on disk) across the network. This can be a heavy, time-consuming operation if those shards are large. This had to happen because we weren't keeping track of individual write operations (sequence numbers) until 6.0 and behind the scenes, **Lucene merges all the adds/updates/deletes into larger segments in a way that you can't recover the individual operations that made up the changes… that is, unless you keep the transaction log (or translog) around for a period of time. 😍😻😻😻😻**

**That's what we now do: we keep the translog until it grows "too large" or "too old" to warrant keeping any more of it. If a replica needs to be "brought up to date" we use the last global checkpoint known to that replica and just replay the relevant changes from the primary translog rather than an expensive large file copy.** **If the primary's translog was "too large" or "too old" to be able to re-play to the replica, then we fall back to the old file-based recovery.**

If you've been operating a large cluster that has real network disconnects, restarts, upgrades, etc, we expect this will make you significantly happier as you won't be waiting for long periods as shards recover.

**Optimistic concurrency control**[**edit**](https://github.com/elastic/elasticsearch/edit/master/docs/reference/docs/concurrency-control.asciidoc)

Elasticsearch is distributed. When documents are created, updated, or deleted, the new version of the document has to be replicated to other nodes in the cluster. ***Elasticsearch is also asynchronous and concurrent, meaning that these replication requests are sent in parallel, and may arrive at their destination out of sequence***. Elasticsearch needs a way of ensuring that an older version of a document never overwrites a newer version.

**Design an API which is asynchronous is nature..??**

To ensure an older version of a document doesn’t overwrite a newer version, every operation performed to a document is assigned a sequence number by the primary shard that coordinates that change. The sequence number is increased with each operation and thus newer operations are guaranteed to have a higher sequence number than older operations. Elasticsearch can then use the sequence number of operations to make sure a newer document version is never overridden by a change that has a smaller sequence number assigned to it.

For example, the following indexing command will create a document and assign it an initial sequence number and primary term:

PUT products/\_doc/1567{

"product" : "r2d2",

"details" : "A resourceful astromech droid"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/master/snippets/1506.console)

You can see the assigned sequence number and primary term in the \_seq\_no and \_primary\_term fields of the response:

{

"\_shards" : {

"total" : 2,

"failed" : 0,

"successful" : 1

},

"\_index" : "products",

"\_id" : "1567",

"\_version" : 1,

"\_seq\_no" : 362,

"\_primary\_term" : 2,

"result" : "created"}

**Elasticsearch keeps tracks of the sequence number and primary term of the last operation to have changed each of the documents it stores. The sequence number and primary term are returned in the \_seq\_no and \_primary\_term fields in the response of the**[**GET API**](https://www.elastic.co/guide/en/elasticsearch/reference/master/docs-get.html)**: 🤩🤩🤩**

GET products/\_doc/1567

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/master/snippets/1507.console)

returns:

{

"\_index" : "products",

"\_id" : "1567",

"\_version" : 1,

"\_seq\_no" : 362,

"\_primary\_term" : 2,

"found": true,

"\_source" : {

"product" : "r2d2",

"details" : "A resourceful astromech droid"

}}

Note: The [Search API](https://www.elastic.co/guide/en/elasticsearch/reference/master/search-search.html) can return the \_seq\_no and \_primary\_term for each search hit by setting [seq\_no\_primary\_term parameter](https://www.elastic.co/guide/en/elasticsearch/reference/master/search-request-body.html#request-body-search-seq-no-primary-term).

The sequence number and the primary term uniquely identify a change. By noting down the sequence number and primary term returned, you can make sure to only change the document if no other change was made to it since you retrieved it. This is done by setting the if\_seq\_no and if\_primary\_term parameters of either the [Index API](https://www.elastic.co/guide/en/elasticsearch/reference/master/docs-index_.html) or the [Delete API](https://www.elastic.co/guide/en/elasticsearch/reference/master/docs-delete.html).

For example, the following indexing call will make sure to add a tag to the document without losing any potential change to the description or an addition of another tag by another API:

PUT products/\_doc/1567?if\_seq\_no=362&if\_primary\_term=2{

"product" : "r2d2",

"details" : "A resourceful astromech droid",

"tags": ["droid"]}

Elasticsearch for storing trie and a quad tree, different shaped grids such as quarters in circles, polygons. Distributed data structure in built support.

<https://blog.burntsushi.net/transducers/#fst-construction>

Geo shape - data points

---------------------------

Square grids, tesellated triangular mesh

<https://www.elastic.co/guide/en/elasticsearch/reference/current/geo-shape.html>

query geometry ==>

<https://www.baeldung.com/elasticsearch-geo-spatial>

**N-gram tokenizer**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/ngram-tokenizer.asciidoc)

The ngram tokenizer first breaks text down into words whenever it encounters one of a list of specified characters, then it emits [N-grams](https://en.wikipedia.org/wiki/N-gram) of each word of the specified length.

N-grams are like a sliding window that moves across the word - a continuous sequence of characters of the specified length. They are useful for querying languages that don’t use spaces or that have long compound words, like German.

**Example output**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/ngram-tokenizer.asciidoc)

With the default settings, the ngram tokenizer treats the initial text as a single token and produces N-grams with minimum length 1 and maximum length 2:

POST \_analyze{

"tokenizer": "ngram",

"text": "Quick Fox"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/800.console)

The above sentence would produce the following terms:

[ Q, Qu, u, ui, i, ic, c, ck, k, "k ", " ", " F", F, Fo, o, ox, x ]

**Configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/ngram-tokenizer.asciidoc)

The ngram tokenizer accepts the following parameters:

|  |  |
| --- | --- |
| min\_gram | Minimum length of characters in a gram. Defaults to 1. |
| max\_gram | Maximum length of characters in a gram. Defaults to 2. |
| token\_chars | Character classes that should be included in a token. Elasticsearch will split on characters that don’t belong to the classes specified. Defaults to [] (keep all characters).  Character classes may be any of the following:   * letter —  for example a, b, ï or 京 * digit —  for example 3 or 7 * whitespace —  for example " " or "\n" * punctuation — for example ! or " * symbol —  for example $ or √ * custom —  custom characters which need to be set using the custom\_token\_chars setting. |
| custom\_token\_chars | Custom characters that should be treated as part of a token. For example, setting this to +-\_ will make the tokenizer treat the plus, minus and underscore sign as part of a token. |

It usually makes sense to set min\_gram and max\_gram to the same value. The smaller the length, the more documents will match but the lower the quality of the matches. The longer the length, the more specific the matches. A tri-gram (length 3) is a good place to start.

The index level setting index.max\_ngram\_diff controls the maximum allowed difference between max\_gram and min\_gram.

**Example configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/ngram-tokenizer.asciidoc)

In this example, we configure the ngram tokenizer to treat letters and digits as tokens, and to produce tri-grams (grams of length 3):

PUT my\_index{

"settings": {

"analysis": {

"analyzer": {

"my\_analyzer": {

"tokenizer": "my\_tokenizer"

}

},

"tokenizer": {

"my\_tokenizer": {

"type": "ngram",

"min\_gram": 3,

"max\_gram": 3,

"token\_chars": [

"letter",

"digit"

]

}

}

}

}}

POST my\_index/\_analyze{

"analyzer": "my\_analyzer",

"text": "2 Quick Foxes."}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/801.console)

The above example produces the following terms:

[ Qui, uic, ick, Fox, oxe, xes ]

**Uses a sliding window, creates ngrams of 3 length, 2 is included as a word but it is not possible to generate a trigram out of it .. so trigrams does not contain  digit,** **tokens and terms means the same, sentence gets tokenized into terms 😍😍**

**Edge n-gram tokenizer**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/edgengram-tokenizer.asciidoc)

The edge\_ngram tokenizer first breaks text down into words whenever it encounters one of a list of specified characters, then it emits [N-grams](https://en.wikipedia.org/wiki/N-gram) of each word where the start of the N-gram is anchored to the beginning of the word.

Edge N-Grams are useful for *search-as-you-type* queries.

When you need *search-as-you-type* for text which has a widely known order, such as movie or song titles, the [completion suggester](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-suggesters.html#completion-suggester) is a much more efficient choice than edge N-grams. Edge N-grams have the advantage when trying to autocomplete words that can appear in any order.

**Example output**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/edgengram-tokenizer.asciidoc)

With the default settings, the edge\_ngram tokenizer treats the initial text as a single token and produces N-grams with minimum length 1 and maximum length 2:

POST \_analyze{

"tokenizer": "edge\_ngram",

"text": "Quick Fox"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/794.console)

The above sentence would produce the following terms:

[ Q, Qu ]

These default gram lengths are almost entirely useless. You need to configure the edge\_ngram before using it.

**Configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/edgengram-tokenizer.asciidoc)

The edge\_ngram tokenizer accepts the following parameters:

**min\_gram**

Minimum length of characters in a gram. Defaults to 1.

**max\_gram**

Maximum length of characters in a gram. Defaults to 2.

See [Limitations of the max\_gram parameter](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-edgengram-tokenizer.html#max-gram-limits).

**token\_chars**

Character classes that should be included in a token. Elasticsearch will split on characters that don’t belong to the classes specified. Defaults to [] (keep all characters).

Character classes may be any of the following:

* letter —  for example a, b, ï or 京
* digit —  for example 3 or 7
* whitespace —  for example " " or "\n"
* punctuation — for example ! or "
* symbol —  for example $ or √
* custom —  custom characters which need to be set using the custom\_token\_chars setting.

**custom\_token\_chars**

Custom characters that should be treated as part of a token. For example, setting this to +-\_ will make the tokenizer treat the plus, minus and underscore sign as part of a token.

**Limitations of the max\_gram parameter**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/edgengram-tokenizer.asciidoc)

The edge\_ngram tokenizer’s max\_gram value limits the character length of tokens. When the edge\_ngram tokenizer is used with an index analyzer, this means search terms longer than the max\_gram length may not match any indexed terms.

For example, if the max\_gram is 3, searches for apple won’t match the indexed term app.

To account for this, you can use the [truncate](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-truncate-tokenfilter.html) token filter with a search analyzer to shorten search terms to the max\_gram character length. However, this could return irrelevant results.

For example, if the max\_gram is 3 and search terms are truncated to three characters, the search term apple is shortened to app. This means searches for apple return any indexed terms matching app, such as apply, snapped, and apple.

We recommend testing both approaches to see which best fits your use case and desired search experience.

**Example configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/edgengram-tokenizer.asciidoc)

In this example, we configure the edge\_ngram tokenizer to treat letters and digits as tokens, and to produce grams with minimum length 2 and maximum length 10:

PUT my\_index{

"settings": {

"analysis": {

"analyzer": {

"my\_analyzer": {

"tokenizer": "my\_tokenizer"

}

},

"tokenizer": {

"my\_tokenizer": {

"type": "edge\_ngram",

"min\_gram": 2,

"max\_gram": 10,

"token\_chars": [

"letter",

"digit"

]

}

}

}

}}

POST my\_index/\_analyze{

"analyzer": "my\_analyzer",

"text": "2 Quick Foxes."}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/795.console)

The above example produces the following terms:

[ Qu, Qui, Quic, Quick, Fo, Fox, Foxe, Foxes ]

Usually we recommend using the same analyzer at index time and at search time. In the case of the edge\_ngram tokenizer, the advice is different. It only makes sense to use the edge\_ngram tokenizer at index time, to ensure that partial words are available for matching in the index. At search time, just search for the terms the user has typed in, for instance: Quick Fo.

Below is an example of how to set up a field for *search-as-you-type*.

Note that the max\_gram value for the index analyzer is 10, which limits indexed terms to 10 characters. Search terms are not truncated, meaning that search terms longer than 10 characters may not match any indexed terms.

PUT my\_index{

"settings": {

"analysis": {

"analyzer": {

"autocomplete": {

"tokenizer": "autocomplete",

"filter": [

"lowercase"

]

},

"autocomplete\_search": {

"tokenizer": "lowercase"

}

},

"tokenizer": {

"autocomplete": {

"type": "edge\_ngram",

"min\_gram": 2,

"max\_gram": 10,

"token\_chars": [

"letter"

]

}

}

}

},

"mappings": {

"properties": {

"title": {

"type": "text",

"analyzer": "autocomplete",

"search\_analyzer": "autocomplete\_search"

}

}

}}

PUT my\_index/\_doc/1{

"title": "Quick Foxes" }

POST my\_index/\_refresh

GET my\_index/\_search{

"query": {

"match": {

"title": {

"query": "Quick Fo",

"operator": "and"

}

}

}}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/796.console)

|  |  |
| --- | --- |
|  | The autocomplete analyzer indexes the terms [qu, qui, quic, quick, fo, fox, foxe, foxes]. |
|  | The autocomplete\_search analyzer searches for the terms [quick, fo], both of which appear in the index. |

**Operator AND  Operator OR** - **Default is OR**

Defined a custom analyzer and tokenizer. Autocomplete search, Autocomplete analyzer.

Analyzer consisting of char filter, tokenizer and token filter

Edge gram tokenizer autocomplete.

**"mappings":** **{**

**"properties":** **{**

**"title":** **{**

**"type":** **"text",**

**"analyzer":** **"autocomplete",**

**"search\_analyzer":** **"autocomplete\_search"**

**For field title - uses a autocomplete analyzer FOR indexing and search\_analyzer FOR  searching**

**Corresponding index query and search query uses respective analyzers. 😍😍😍**

[« N-gram tokenizer](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-ngram-tokenizer.html)[Path Hierarchy Tokenizer Examples »](https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-pathhierarchy-tokenizer-examples.html)

**Path Hierarchy Tokenizer**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/pathhierarchy-tokenizer.asciidoc)

The path\_hierarchy tokenizer takes a hierarchical value like a filesystem path, splits on the path separator, and emits a term for each component in the tree.

**Example output**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/pathhierarchy-tokenizer.asciidoc)

POST \_analyze{

"tokenizer": "path\_hierarchy",

"text": "/one/two/three"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/802.console)

The above text would produce the following terms:

[ /one, /one/two, /one/two/three ]

**Configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/pathhierarchy-tokenizer.asciidoc)

The path\_hierarchy tokenizer accepts the following parameters:

|  |  |
| --- | --- |
| delimiter | The character to use as the path separator. Defaults to /. |
| replacement | An optional replacement character to use for the delimiter. Defaults to the delimiter. |
| buffer\_size | The number of characters read into the term buffer in a single pass. Defaults to 1024. The term buffer will grow by this size until all the text has been consumed. It is advisable not to change this setting. |
| reverse | If set to true, emits the tokens in reverse order. Defaults to false. |
| skip | The number of initial tokens to skip. Defaults to 0. |

**Example configuration**[**edit**](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/pathhierarchy-tokenizer.asciidoc)

In this example, we configure the path\_hierarchy tokenizer to split on - characters, and to replace them with /. The first two tokens are skipped:

PUT my\_index{

"settings": {

"analysis": {

"analyzer": {

"my\_analyzer": {

"tokenizer": "my\_tokenizer"

}

},

"tokenizer": {

"my\_tokenizer": {

"type": "path\_hierarchy",

**"delimiter":** **"-", 😍😍😍**

**"replacement":** **"/", 😎😎😎**

"skip": 2

}

}

}

}}

POST my\_index/\_analyze{

"analyzer": "my\_analyzer",

"text": "one-two-three-four-five"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/803.console)

The above example produces the following terms:

[ /three, /three/four, /three/four/five ]

If we were to set reverse to true, it would produce the following:

[ one/two/three/, two/three/, three/ ]

}

}

}}

**For finding documents in a specific folder hierarchy, specific category hierarchy (For example IAB category hierarchy) Documents matching upto a specific level path in a tree (All paths in the tree are indexed)**

**Path Hierarchy Tokenizer Examples**

**----------------------------------------------------**

[edit](https://github.com/elastic/elasticsearch/edit/7.6/docs/reference/analysis/tokenizers/pathhierarchy-tokenizer-examples.asciidoc)

A common use-case for the path\_hierarchy tokenizer is filtering results by file paths. If indexing a file path along with the data, the use of the path\_hierarchy tokenizer to analyze the path allows filtering the results by different parts of the file path string.

This example configures an index to have two custom analyzers and applies those analyzers to multifields of the file\_path text field that will store filenames. One of the two analyzers uses reverse tokenization. Some sample documents are then indexed to represent some file paths for photos inside photo folders of two different users.

PUT file-path-test{

"settings": {

"analysis": {

"analyzer": {

"custom\_path\_tree": {

"tokenizer": "custom\_hierarchy"

},

"custom\_path\_tree\_reversed": {

"tokenizer": "custom\_hierarchy\_reversed"

}

},

"tokenizer": {

"custom\_hierarchy": {

"type": "path\_hierarchy",

"delimiter": "/"

},

"custom\_hierarchy\_reversed": {

"type": "path\_hierarchy",

"delimiter": "/",

"reverse": "true"

}

}

}

},

"mappings": {

"properties": {

"file\_path": {

"type": "text",

"fields": {

"tree": {

"type": "text",

"analyzer": "custom\_path\_tree"

},

"tree\_reversed": {

"type": "text",

"analyzer": "custom\_path\_tree\_reversed"

}

}

}

}

}}

POST file-path-test/\_doc/1{

"file\_path": "/User/alice/photos/2017/05/16/my\_photo1.jpg"} 😍😍

POST file-path-test/\_doc/2{

"file\_path": "/User/alice/photos/2017/05/16/my\_photo2.jpg"} 🤩🤩

POST file-path-test/\_doc/3{

"file\_path": "/User/alice/photos/2017/05/16/my\_photo3.jpg"} 😻😻

POST file-path-test/\_doc/4{

"file\_path": "/User/alice/photos/2017/05/15/my\_photo1.jpg"} 😍😍

POST file-path-test/\_doc/5{

"file\_path": "/User/bob/photos/2017/05/16/my\_photo1.jpg"} 😍😍

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/804.console)

A search for a particular file path string against the text field matches all the example documents, with Bob’s documents ranking highest due to bob also being one of the terms created by the standard analyzer boosting relevance for Bob’s documents.

GET file-path-test/\_search{

"query": {

"match": {

"file\_path": "/User/bob/photos/2017/05"

}

}}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/805.console)

It’s simple to match or filter documents with file paths that exist within a particular directory using the file\_path.tree field.

GET file-path-test/\_search{

"query": {

"term": {

"file\_path.tree": "/User/alice/photos/2017/05/16"

}

}}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/806.console)

With the reverse parameter for this tokenizer, it’s also possible to match from the other end of the file path, such as individual file names or a deep level subdirectory. The following example shows a search for all files named my\_photo1.jpg within any directory via the file\_path.tree\_reversed field configured to use the reverse parameter in the mapping.

GET file-path-test/\_search{

"query": {

"term": {

"file\_path.tree\_reversed": {

"value": "my\_photo1.jpg"

}

}

}}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/807.console)

Viewing the tokens generated with both forward and reverse is instructive in showing the tokens created for the same file path value.

POST file-path-test/\_analyze{

"analyzer": "custom\_path\_tree",

"text": "/User/alice/photos/2017/05/16/my\_photo1.jpg"}

POST file-path-test/\_analyze{

"analyzer": "custom\_path\_tree\_reversed",

"text": "/User/alice/photos/2017/05/16/my\_photo1.jpg"}

**Copy as cURL**[**View in Console**](http://localhost:5601/app/kibana#/dev_tools/console?load_from=https://www.elastic.co/guide/en/elasticsearch/reference/current/snippets/808.console)

It’s also useful to be able to filter with file paths when combined with other types of searches, such as this example looking for any files paths with 16 that also must be in Alice’s photo directory.

GET file-path-test/\_search{

"query": {

"bool" : {

"must" : {

"match" : { "file\_path" : "16" }

},

"filter": {

"term" : { "file\_path.tree" : "/User/alice" }

}

}

}}