Using Elasticsearch and OpenSearch for Content Indexing and Content-Based Retrievals (CBR)

The use of Elasticsearch and OpenSearch was first introduced as a preview capability in the 5.5.8 release. In the 5.5.12 release of FileNet Content Manager and IBM Content Foundation, Elasticsearch or OpenSearch can be used as a fully supported alternative to IBM Content Search Services (CSS) for content indexing and for content-based retrieval (CBR). This new capability can be used with traditional installations of Content Platform Engine and with containerized installations.

Elasticsearch or OpenSearch licenses must be purchased separately, they are not included in the FileNet Content Manager or IBM Content Foundation bundles. Refer to the appropriate software product compatibility report (SPCR) for information on the supported levels of Elasticsearch and OpenSearch. Reports can be generated here: http://www.ibm.com/software/reports/compatibility/clarity/index.html

Some components within Cloud Pak for Business Automation (CP4BA) embed Elasticsearch and OpenSearch. However, these embedded versions are not licensed for use with FileNet Content Manager or IBM Content Foundation. In addition, when installing the content pattern with the CP4BA operators, only CSS is available for content indexing and for content-based retrieval.

This new content-based search feature operates similarly to CSS. The Document, Annotation, Folder, and Custom Object classes, as well as the string properties of those classes can be enabled for full text indexing.

The CONTAINS clause in a full text search query is very similar for retrieving content using CSS, Elasticsearch, or OpenSearch; therefore, existing queries do not need to change if a decision is made to index content with Elasticsearch or OpenSearch rather than CSS.

There are differences in the results that might be returned by the different technologies as the stemming algorithms are different.

This white paper provides insight into the new content-based search feature and identifies the differences between the technologies used for content-based indexing and searching.

Important: In this white paper, as well as in the Administration Console for Content Engine (ACCE), the term Elasticsearch is used to refer to both Elasticsearch and OpenSearch.

Contents

Search Engine to Object Store Mapping

The search engine to object store mapping is as follows:

- The Content Platform Engine domain supports both the new Elasticsearch content-based search feature and CSS.
- A domain can support only one Elasticsearch content-based search cluster.
- An object store can support either Elasticsearch content-based search or CSS, but not both.

Figure 1 illustrates this mapping.

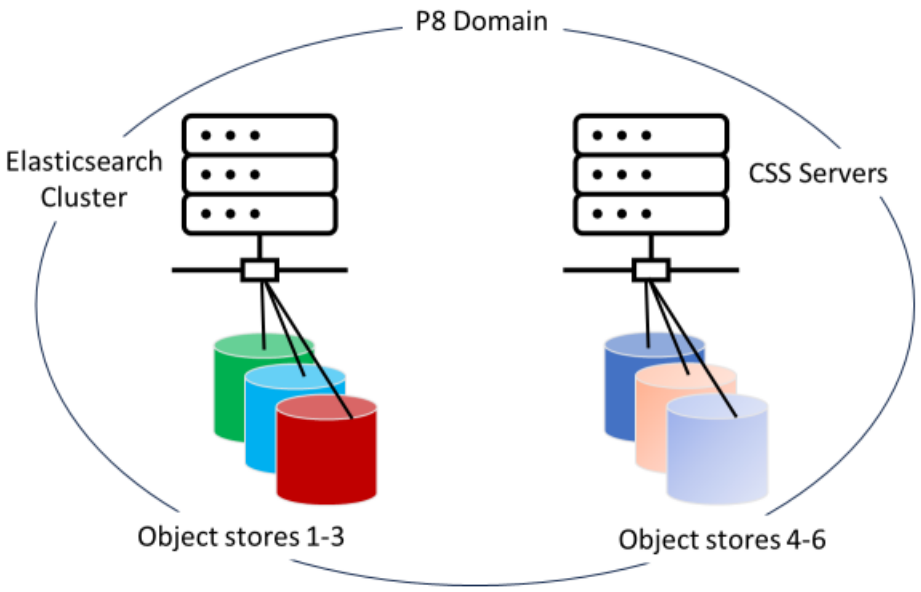


Figure 1: Search Engine to Object Store Mapping

Elasticsearch Index Areas

An Object Store can support one Elasticsearch index area. The index area contains one Elasticsearch index per enabled root class. For instance, one index for the Document class and all the subclasses of the Document class, and one index for the Custom Object class and all the subclasses of the Custom Object class. In comparison, CSS supports multiple index areas per object store, and multiple partitioned indexes per index area. The Elasticsearch index scale-out is based on index sharding, which is controlled by Elasticsearch.

Indexing is usually quicker when you have more shards, as each document needs to be stored only once per shard. If fast ingestion is the major concern, there should be at least one shard per data node.

For more information on shards and nodes, refer to this documentation:
https://www.elastic.co/guide/en/elasticsearch/reference/current/size-your-shards.html

Indexing Pipeline

The indexing pipeline used for Elasticsearch uses a queue sweep for indexing called the Elasticsearch Indexing Queue Sweep.

The Content Platform Engine object server code creates index queue requests for all newly created, updated, or deleted objects that are Content Based Retrieval (CBR) enabled. An Index Job creates queue entries when a class is first enabled for indexing, or when a re-index request is made. The index entries are processed by the indexing queue sweep using the Elasticsearch REST API.

Figure 2 illustrates the indexing pipeline for Elasticsearch and for CSS.

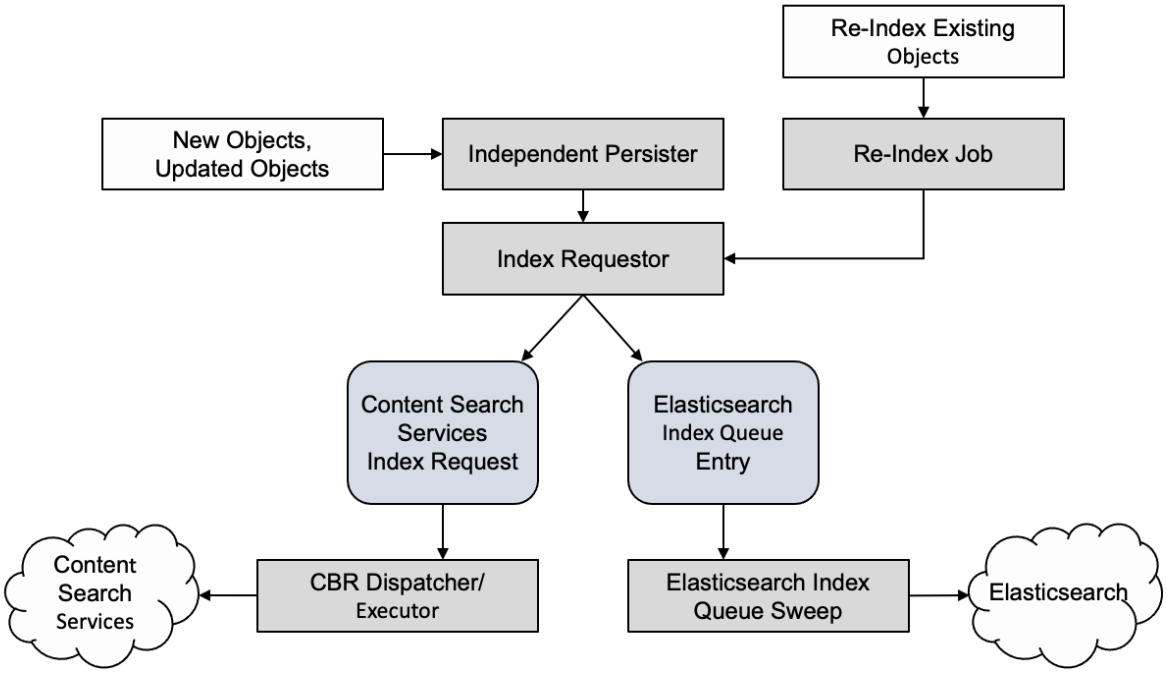


Figure 2: The indexing pipeline for Elasticsearch and for CSS

Configuring an Environment to use Elasticsearch

Supported Elasticsearch and OpenSearch Releases

Refer to the appropriate software product compatibility report for supported Elasticsearch and OpenSearch versions and for any additional caveats. Use the following website to generate a report: http://www.ibm.com/software/reports/compatibility/clarity/index.html

Configure an Elasticsearch Cluster to Allow Content Platform Engine Access

You must configure the Elasticsearch cluster to allow access by the Content Platform Engine. The Content Platform Engine supports both IP-based security and username/password-based security.

For IP-based security, access to the Elasticsearch cluster is controlled by the client IP addresses that can access the cluster (normally within a VPN). As a result, the Content Platform Engine Elasticsearch cluster object is created without specifying a username or a password.

For username/password-based security, create credentials on the Elasticsearch cluster that allow access by the Content Platform Engine. Set the specified credentials as the username/password properties on the Content Platform Engine Elasticsearch cluster object.

Required Permissions

The Elasticsearch users and roles configured for the Content Platform Engine Elasticsearch cluster must have the following access rights:

- Read access to the cluster base URL to test connectivity to the cluster.
- Full control access to indexes with the prefix name of *'fncm_*'*.

Solid-State Storage Device

Document indexing is a disk-intensive activity. In production environments, use a solid-state storage device (SSD) for the cluster storage to limit the possibility of disk I/O being a performance bottleneck.

Configure Content Platform Engine

Use ACCE to configure the Content Platform Engine to use Elasticsearch:

1. Create an Elasticsearch cluster at the domain level. Select Domain > Global Configuration > Administration > Elasticsearch Clusters.

   There can be only one Elasticsearch cluster per domain. However, the cluster can be shared by multiple object stores.

   Supply either the name of the Elasticsearch load balancer or a list of host names.

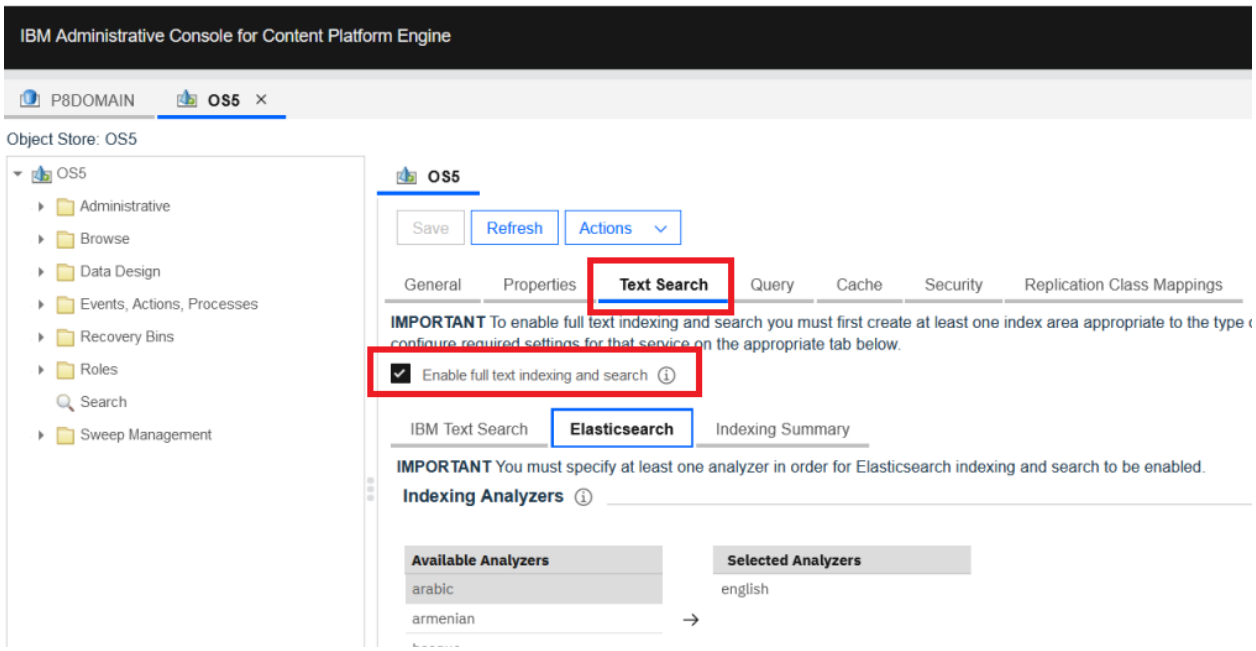
   If you provide a list of host names, the Content Platform Engine manages requests across the set of hosts in a round-robin basis.

   You must also provide a port for accessing the cluster. The CPE does not supply a default port. Append the port number to each host name; for instance, MyElasticSearchCluster:9200 or MyOpenSearchCluster:443.

2. Create an OpenSearch index area at the object store level. Expand the appropriate object store node, then select Administrative > Index Areas.

   It is important to consider the number of shards and replicas per index when you create the index area. For performance reasons, consider using the same number of shards and nodes as a starting configuration. Once content is indexed, changing the number of shards or replicas requires content to be re-indexed. Refer to the following section in this document for additional best practice information: Best Practices.

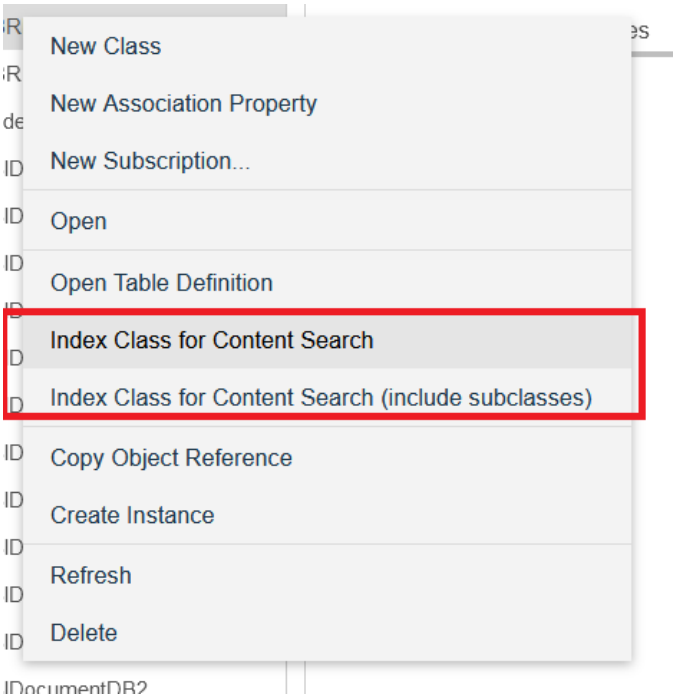
3. Enable Elasticsearch at the object store level. Select the appropriate object store node, then select the Text Search tab and check the Enable full text indexing and search option.



   On the Elasticsearch tab, select the indexing analyzers you want to use. There are two nonlanguage analyzers: simple and fncm_email_analyzer.

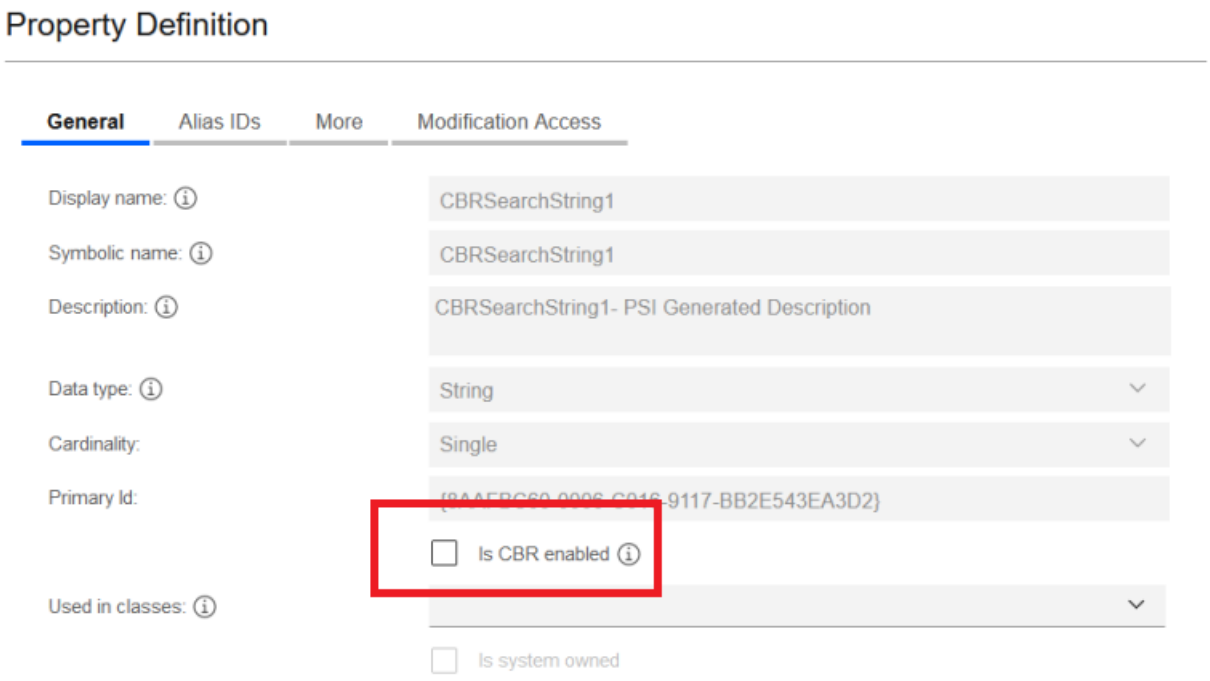
   Analyzers are applied when objects are first indexed, and if the analyzer list is changed, the objects must be reindexed. More information on selecting analyzers is provided in the Best Practices section of this document.

4. Enable CBR on classes and properties. Once a class is enabled for CBR, newly created objects of the class are automatically indexed. If there are existing objects for classes that are CBR enabled, you can index those objects by selecting one of the Index Class for Content Search options.

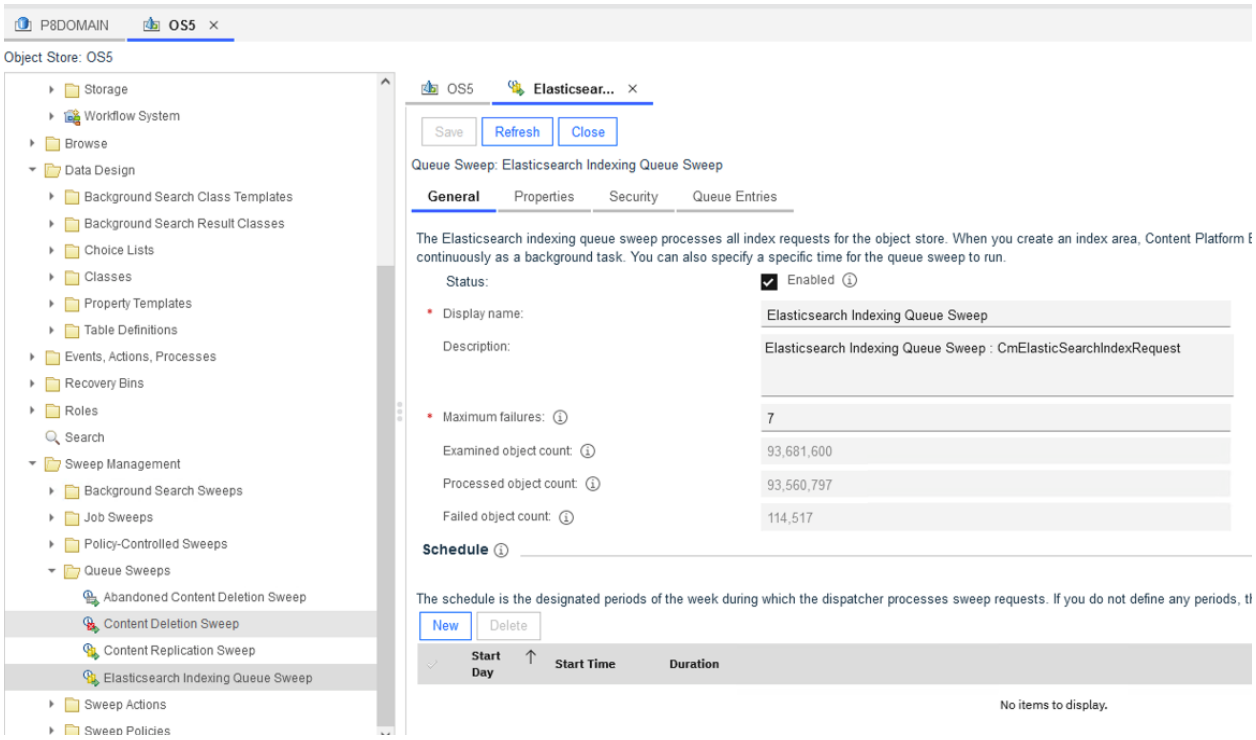


The option to enable CBR on a class is on the General tab of the class definition.

Use the Enable CBR checkbox on a string property to index the content of string properties. Navigate to the property definition in the appropriate classes, then check the Enable CBR option. The Enable CBR checkbox is on the General tab of the property definition.



5. Monitor the Elasticsearch indexing queue sweep process using ACCE.



Additional Step for Windows Environments

On CPE Windows servers, install the Microsoft 2013 and 2019 Redistributable files if the files are not already on the server. The files are needed for CBR indexing and searching, as well as for generating thumbnails.

Reindexing

The Elasticsearch feature does not support the Index Resync capability that is available for CSS. The Index Resync mechanism, is the recommended best practice for resynchronizing indexes as a fine-grained filter expression can be used to reindex a small set of objects. For more information, refer to described in https://www.ibm.com/docs/en/filenet-p8-platform/5.5.12?topic=failures-fixing-out-sync-full-text-index.

The two options for Elasticsearch reindexing are:

- Reindex all the objects in an entire class (Document, Annotation, Custom Object, or Folder)
- Use the Full Text Reindex Job Sweep to reindex specific objects

The Full Text Reindex Job Sweep provides a mechanism to submit CBR-enabled Documents, Annotations, Custom Objects, or Folders for reindexing, based on a filter expression. (See *Sweep filter conditions* https://www.ibm.com/docs/en/filenet-p8-platform/5.5.12?topic=sweeps-sweep-filter-conditions). The primary use case for the Full Text Reindex Job Sweep is for reindexing CBR-enabled objects that are indexed under Elasticsearch.

A Full Text Reindex Job Sweep operates on objects selected by a filter expression and a CBR-enabled class (Document, Annotation, Custom Object, or Folder). The sweep can act on a single class, or the sweep can also process the objects in the subclasses. Initiate the sweep from ACCE. Navigate to the appropriate object store and then to Sweep Management > Job Sweeps > Full Text Reindex Jobs.

The Job Sweep processes objects that satisfy the filter expression and class conditions, and then inserts the index requests into the indexing queue to be processed by the indexer. Monitor the progress and results of a Full Text Reindex Job Sweep using ACCE.

The default settings for the Full Text Reindex Job Sweep are

- Inter-Batch Delay: 1,000 milliseconds
- Sweep Batch Size: 200
- Target class: Document
- Effective start and end dates: Blank

The Sweep Job inserts rows into the indexing queue very quickly, which might overload the queue, and prevent newly added objects from being indexed in a timely fashion. Use the Inter-batch delay to limit the processing rate of the Sweep Job.

Full Text Query Syntax

Refer to the following IBM Documentation topic for details on how the CONTAINS clause is used by CSS:

https://www.ibm.com/docs/en/filenet-p8-platform/5.5.12?topic=reference-cbr-queries

The syntax of the CONTAINS clause for Elasticsearch is very similar. There is one advanced feature that applies to Elasticsearch, which is the optional ability to supply the analyzer name to be used for the search. Normally all analyzers are used. For example, to search the document titles for the term 'arches' using only the English analyzer, supply the analyzer in the CONTAINS clause as follows:

```
CONTAINS(d.*,'documenttitle.english:arches')
```

The query syntax used within the CONTAINS clause is Elasticsearch specific, with some minor differences from the CSS query syntax. The Content Platform Engine passes the value supplied in the CONTAINS clause to Elasticsearch as the query string value. For more information, refer to the following documentation https://opensearch.org/docs/latest/query-dsl/full-text/index/.

Word Stemming

Stemming is the process of reducing words to their base or root form. CSS and Elasticsearch use different methodologies to derive the stems of words. The word stemming determines what documents are found with a CBR search.

CSS uses dictionary stemming, while Elasticsearch uses algorithmic stemming. The two stemming methods can produce different results. For instance, non-dictionary words are stemmed by Elasticsearch, but not stemmed by CSS.

By default, Elasticsearch uses Porter algorithmic stemming. For more information, refer to the following link: https://snowballstem.org/algorithms/porter/stemmer.html

Best Practices

This section covers the following topics:

- Selecting the appropriate number of shards for an index
- Selecting the appropriate language analyzers
- Setting the maximum number of indexing queue sweep workers
- Setting the reindexing sweep job parameters
- Tuning CBR indexing when there is a high work load
- Backup and recovery

Selecting the Number of Shards for an Index Area

The Content Platform Engine takes advantage of Elasticsearch index sharding to achieve index scale-out. It is important to use enough shards when you create the index area, as the number of shards cannot be changed without performing a full reindex. The OpenSearch administrator needs to determine the correct number of shards for the index area based on predicted ingestion volume, number of shards the cluster can manage, and so on. The number of replicas needs to be considered when you create the index area, since the replicas provide high availability for the indexed data.

Indexing is usually quicker when you have more shards, as each document needs to be stored only once per shard. If fast ingestion is the major concern, you should have at least one shard per data node.

The ratio of shards to nodes is very important. Indexing work is sent to the shards in a round robin fashion. To ensure the best performance, the number of shards should be a multiple of the number of nodes. For instance, if there are three data nodes, configure three, six, or nine shards. If the number of shards is not a multiple of the number of nodes, performance degrades because the workload is not spread out evenly over the nodes.

Selecting the Language Analyzers for an Object Store

Elasticsearch uses analyzers for indexing and for searching. The analyzers used by the Content Platform Engine are set at the object store level and are applied to all CBR-enabled classes in the object store.

If the analyzer list changes a reindex is required. The recommendation is to use the simple analyzer and one language analyzer for each of the languages in which documents ingested into the object store might be written.

The simple analyzer breaks tokens on punctuation. Without the simple analyzer, sentences that lack spaces between the punctuation are not tokenized as expected.

However, using the simple analyzer can cause problems with searches not finding strings with numbers. For example, ‘PO3025721’ is tokenized as just ‘po’. This can cause the search results to match far more documents than expected.

The fncm_email_analyzer is a custom analyzer designed to handle information in emails.

For more information about Analyzers, refer to the following blog: https://opensearch.org/docs/latest/

Setting the Maximum Workers for the Indexing Queue Sweep

The indexing queue sweep is created automatically during Elasticsearch indexing. The sweep is defined with a default of eight workers. In some cases, you might need to increase the number of workers to improve indexing throughput. It can be difficult to estimate if an increase in the indexing queue workers impacts indexing throughput, or if the increase in the workers has a negative impact on other Content Platform Engine operations. If you are considering increasing the number of workers, do so incrementally while you monitor the system by using the System Dashboard or with other tools that can monitor the Content Platform Engine system resources (CPU, memory, and other resources). Refer to the section on Monitoring Performance for more information on tracking system performance.

Setting the Reindexing Sweep Job Parameters

The Reindexing Sweep Job can insert rows into the indexing queue very quickly, which might overload the queue, and prevent newly added objects from being indexed in a timely fashion. Use the Inter-batch delay to limit the processing rate of the Sweep Job. For example, assume the sweep uses the following settings:

- Workers: 1
- Batch size: 200
- Inter-batch delay: One second (1,000 milliseconds)

With these settings the sweep rate, that is the rate that rows are inserted into indexing queue, cannot exceed 200 per second. As a best practice, set the Inter-batch delay to 1 second (1,000 milliseconds) or higher, set workers to 1, and set the batch size to 200.

Tuning Content Based Retrieval Indexing with Elasticsearch for High Volume Work Loads

For high volume ingestion, migration, or re-indexing scenarios, the default configuration for the Elasticsearch Indexing Queue Sweep might not be sufficient. A high ingest rate can cause the number of indexing requests to build up. In turn, this can cause a significant delay in documents being included in search results. Making the following changes can help improve indexing throughput rate:

- Increasing the number of sweep workers

  This is the recommended method of tuning indexing performance. To change the maximum number of sweep workers, go to Object Store > Sweep Management > Queue Sweeps > Elasticsearch Indexing Queue Sweep > Properties > Maximum Sweep Workers.

- Changing the bulk batch size

  The indexing rate can be affected by tuning the BulkAPI batch size. The BulkAPI batch size controls the size of batches that Content Platform Engine submits for indexing to the Elasticsearch cluster. The default BulkAPI batch size is 40 documents. Alter the setting using the following JVM argument:

  ```
  ecm.elasticsearch.bulkapi.max.batch.size
  ```

Backup and Recovery

Recommendations for backup and recovery:

- Use shard replicas for high availability. The shard replicas enable Elasticsearch to automatically recover corrupted shards.
- Take periodic Elasticsearch index snapshots in case a manual recovery is needed.
- If an index needs to be recovered, use a reindex sweep job to synchronize the index with the object store data. For example, if an index is recovered to the current time minus four hours, run a reindex sweep job using a filter expression that processes objects with a last modified date of the current time minus four hours.

Monitoring Performance

This section provides details of the information provided in the

- PCH and log file counters
- Trace logs when CBR tracing is enabled. There are different entries when indexing content and when searching for content.

PCH and Log File Counters

The CBR layer provides a rich set of high level PCH counters, many apply to both CSS and Elasticsearch. In addition, there is the following set of Elasticsearch-specific PCH counters:

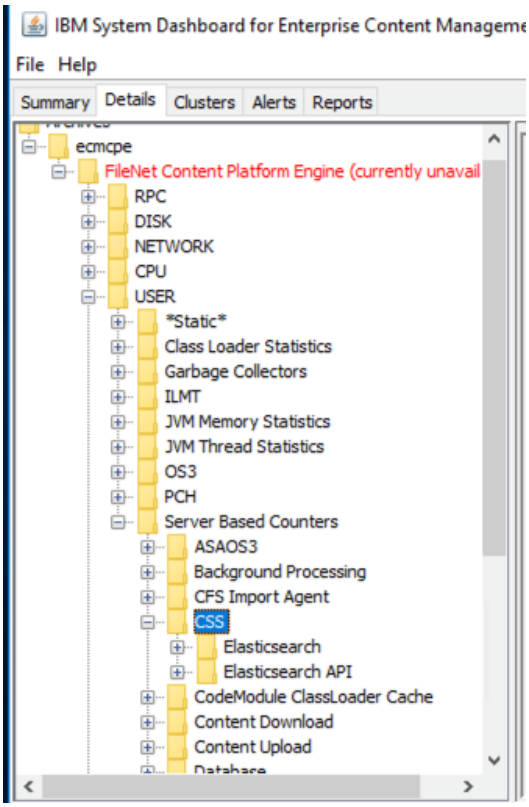
Counter	Description
IndexReadSource	Time spent reading source content to perform indexing
IndexJSONPrep	Time spent preparing extracted text (as JSON) for indexing
Query	The number of times Elasticsearch queries are run
QueryHits	The number of results from the Elasticsearch queries
ScalarQuery	Number of queries based on a set of object IDs
ScalarQueryHits	The number of results found by the scalar queries
CountQuery	A count of the hits found by the queries (using a Select COUNT query)
CountQueryHits	The number of times a COUNT query is run
GetObjectById	The number of queries that search for a single object
SubmitBulkBatch	The number of times a call is made in a batch to create, update, or delete index entries
SubmitBulkBatchItems	Total number of items processed by a bulk batch submission
SubmitBulkBatchFailures	The number of times the batch submission fails
BulkObjectsCreated	Within the batch, the number of index entries created
BulkObjectsUpdated	Within the batch, the number of index entries updated
BulkObjectsDeleted	Within the batch, the number of index entries deleted
BulkObjectsIgnored	Within the batch, the number of index entries that don't need to be updated
BulkObjectsFailed	Within the batch, the number of index requests that failed

The counters are periodically written to the log file if CBR Summary trace is enabled and there is indexing or search activity. Look for *ElasticCounters* in the logs. Following is a sample entry:

```
2022-02-01T18:13:43.062 0000007A CBR  FNRCE0000D - DEBUG
ElasticCounters.objects_indexed: 742, indexed_data_size: 23861273,
objects_deleted: 0, batches_submitted: 27, batch_items_submitted:
742, queries: 0, query_hits: 0, scalar_queries: 0,
scalar_query_hits: 0, count_queries: 0, count_query_hits: 0,
get_object_by_id: 742, initiate_pit: 0
```

Using System Dashboard to Monitor Elasticsearch

System Dashboard displays the CBR counters for Elasticsearch under the Server-Based Counters. For CSS counters are displayed under the FileNet Content Platform Engine node under the object store



In Figure 3, the Bulk API Objects Created counter can be used to check on the CBR indexing rate

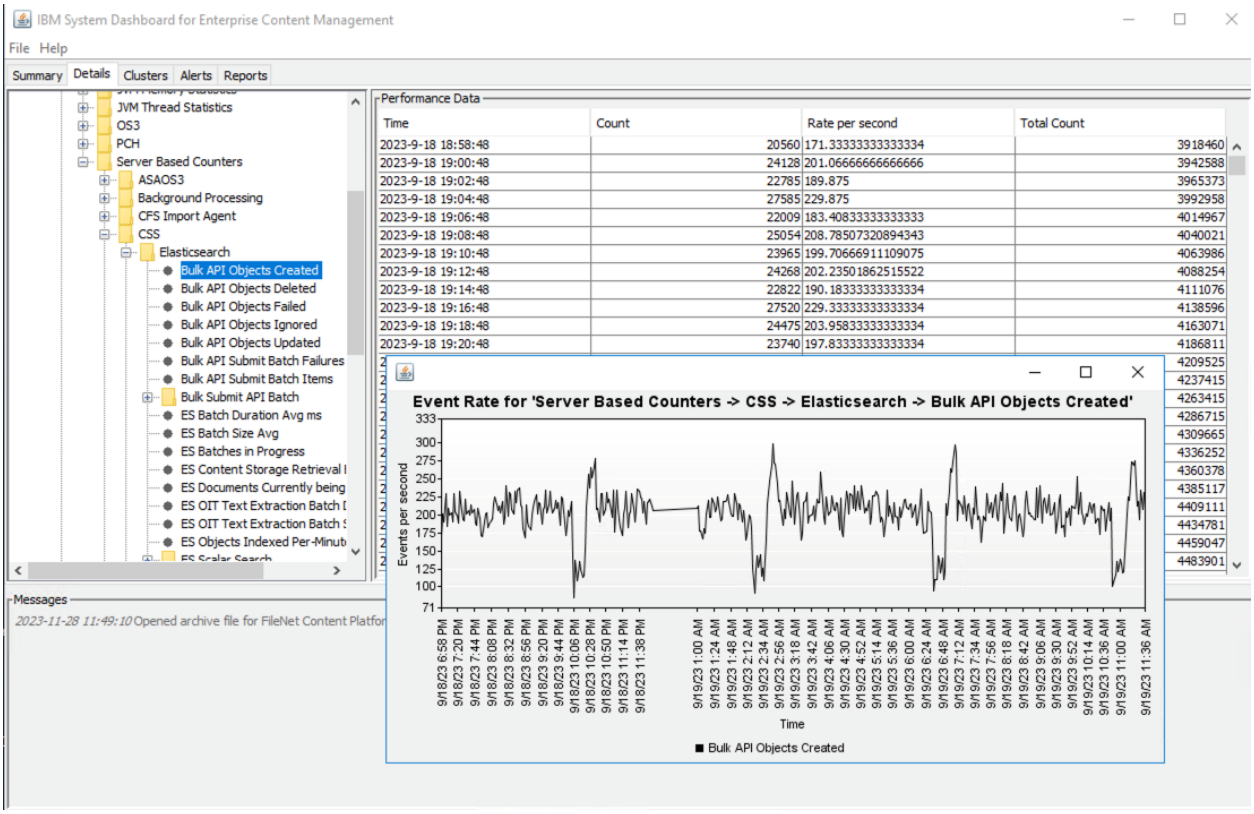


Figure 3: Viewing indexing rate

Figure 4 shows the number of items processed by a sweep process.

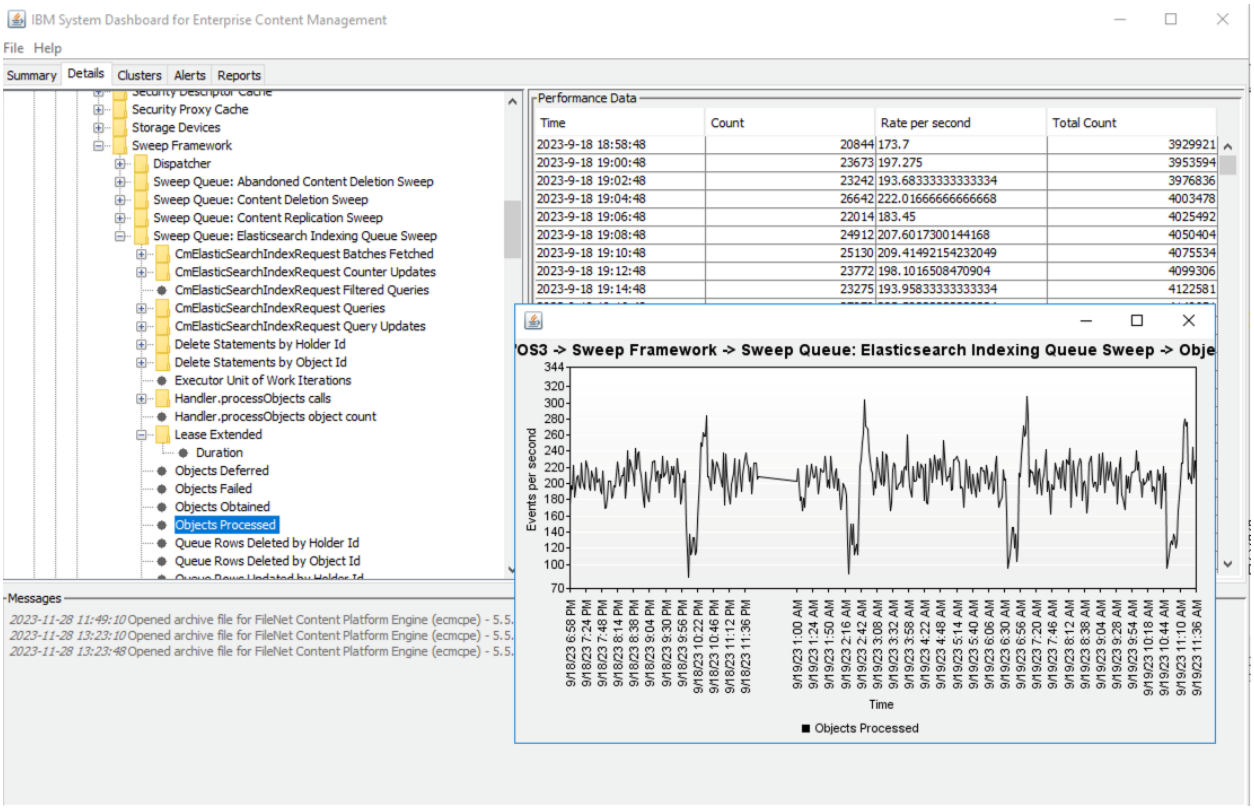


Figure 4: Objects processed by a sweep process

Indexing Logging

This section explains the log entries that occur when the different logging levels are enabled for CBR.

CBR summary trace level

Following is an example of the trace entry produced by the Elasticsearch indexing queue sweep processing (ElasticSearchIndexer.java) after each batch is processed. Note that the queue sweep uses multiple workers, which means that the output from a single entry does not indicate overall throughput.

```
2022-02-23T11:42:44.047 00000191 CBR  FNRCE0000D - DEBUG
Elasticsearch index batch: 200 items in 40876 ms,
create_update_in_batch 200, delete_in_batch 0, not_in_batch 0
(initialize 219 ms, extract_text 1469 ms, extract_text_cleanup 156
ms, index 38954 ms, update_index_counters 0 ms,
propagate_failure_codes 78 ms, content_bytes 2564998,
largest_content
406443)>>200,40876,200,0,0,219,1469,156,38954,0,78,2564998,406443
```

The elements in the entry are described in the following table. The integers after the double broken brackets (>>) marker are the values from the summary trace line in an easy to parse format. All duration values are in milliseconds.

Component	Meaning
index batch	The number of items in the batch. As each item is prescreened before being inserted into the indexing queue, each item has some form of processing applied to it, even if the processing simply determines that the text cannot be extracted.
items in ...	The overall batch processing duration.
create_update_in_batch	The number of create and update requests contained in the batch sent to Elasticsearch.
delete_in_batch	The number of delete requests contained in the batch sent to Elasticsearch.
not_in_batch	The number of queue sweep entries not included in the batch. For instance, if a delete references a non-existent Elasticsearch object, this is indicated in the not_in_batch value.
initialize	Overhead for setting up the batch for indexing.
extract text	Time spent extracting text using Oracle Outside In Technology (OIT).
extract text cleanup	Time spent deleting temporary files, and so on.
index	Time spent indexing the batch. This time is used to make calls to Elasticsearch.
update index counters	Time spent updating the Index Area API object with the current Elasticsearch index counters: number of documents and size of indexes.
propagate failure codes	Time spent updating the last failure code on objects. This action occurs only if the indexing failure recording level on the object store is set to “propagate to source”. If an error occurs, the failure code is written to the index_failure_code property on the document that has the issues. If propagate to source is not set, the error is written to the CE log. 
content_bytes	Total amount of content indexed in the batch.
largest_content	Within the batch, the largest document.

CBR moderate trace level

Following is an example of the details of each bulk API request submitted to Elasticsearch. The entry is produced by the Elasticsearch layer (ElasticBulkAPIBatch.java) that handles submitting the bulk API requests. For each indexing queue sweep batch there are normally multiple bulk API requests. The requests break the larger queue sweep batch into smaller bulk API batches.

```
2022-02-01T18:23:23.934 000006E4 CBR  FNRCE0000D - DEBUG
ElasticBulkAPIBatch.Summary elapsed: 211 ms (took 115), 40 created,
0 updated, 0 deleted, 0 ignored, errors: false, 0 failed
```

The elements in the entry are described in the following table. All duration values are in milliseconds.

Component	Meaning
elapsed	Elapsed time measured from the Content Platform Engine side of the request.
took	The elapsed time reported by Elasticsearch.
created	The number of new index items created.
updated	The number of existing index items updated.
deleted	The number of existing index items deleted.
ignored	The number of items that are not submitted due to some condition; for example, an update of a non-existent index item.
errors	True if any errors are reported by Elasticsearch for any item in the batch; otherwise, false.
failed	The number of items in the batch that failed.

CBR Detail trace level

Following is an example of a detail trace entry. The entry shows the number of documents in each batch sent to Elasticsearch. By default, the batch size is 40. The maxRequestPayloadSize is the maximum size of the batch that can be sent to Elasticsearch in bytes.

```
2022-02-23T11:20:43.858 00000181 CBR  FNRCE0000D - DEBUG
ElasticBulkAPIBatch.construction, bulkAPIenabled=true,
maxRequestsPerBatch=40, maxRequestPayloadSize=25165824
```

Search Logging

ElasticAPI full-text query trace

Following is an example of a moderate search trace entry for an ElasticAPI.full-text query:

```
2021-06-22T21:43:07.905 000000BC CBR  FNRCE0000D - DEBUG
ElasticAPI.full-text query returned 2000 hits in 402 ms (parsing 20
ms) [took=371, timed_out=false] query=[{"track_scores": true,
"_source": "object_id", "size" : 2000, "query": { "query_string" :
{  "fields" :
["fncm_content","fncm_content.english","fncm_content.french"],
"default_operator": "and", "analyze_wildcard" : true, "query" :
"kirk OR storage"}}, "search_after": [4.2998805,"7A2F1160-000E-CFC5-
A480-9DBCE048DB5F"], "sort": [{"_score": "desc", "object_id":
"asc"}]}]
```

Component	Meaning
returned x hits in y ms	Provides the number of hits returned by the search and how long the search took in milliseconds.
parsing	The time spent parsing the JSON results returned from Elasticsearch; that is, the time to convert the JSON to results consumable by the Content Platform Engine.
took	The time Elasticsearch spent processing the query. Elasticsearch supplies this value as part of the search response. The difference between the complete duration minus the took time plus the parsing time, is the amount of time the search was in transit between the Content Platform Engine and Elasticsearch; that is, the send/receive time.
Timed_out	False if the search completed in a timely manner.

CSEElasticQuery summary

If there is search activity, the following kinds of trace lines are produced once every two minutes. There is one line for each of the three types of searches:

- Normal
- Scalar
- Count (estimation).

Each line shows the total hits and duration for the given search type. For normal queries, the number of hits consumed by Content Platform Engine is also shown (get next result count).

```
2020-10-21T16:40:45.352 00000142 CBR  FNRCE0000D - DEBUG
CSEElasticQuery.summary: 7 queries returned 5567 hits in 5602 ms,
get next result count 4377
```

```
2020-10-21T16:40:45.352 00000142 CBR  FNRCE0000D - DEBUG
CSEElasticQuery.summary: 0 scalar queries returned 0 hits in 0 ms
```

```
2020-10-21T16:40:45.352 00000142 CBR  FNRCE0000D - DEBUG
CSEElasticQuery.summary: 0 estimation queries returned 0 hits in 0
ms
```

Tuning

Elasticsearch tuning is described in the following article. There are no specific tuning recommendations for the Content Platform Engine.

https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-search-speed.html

Known Issues

Read Time-out

In some cases, Elasticsearch-related read time out error messages might appear in the Content Platform Engine logs. The errors occur during indexing, when it takes longer than the default read timeout value of 90 seconds to complete an operation on the Elasticsearch side. The read timeout value can be changed by setting the JVM on all Content Platform Engine servers. The following example, changes the timeout to 120 seconds:

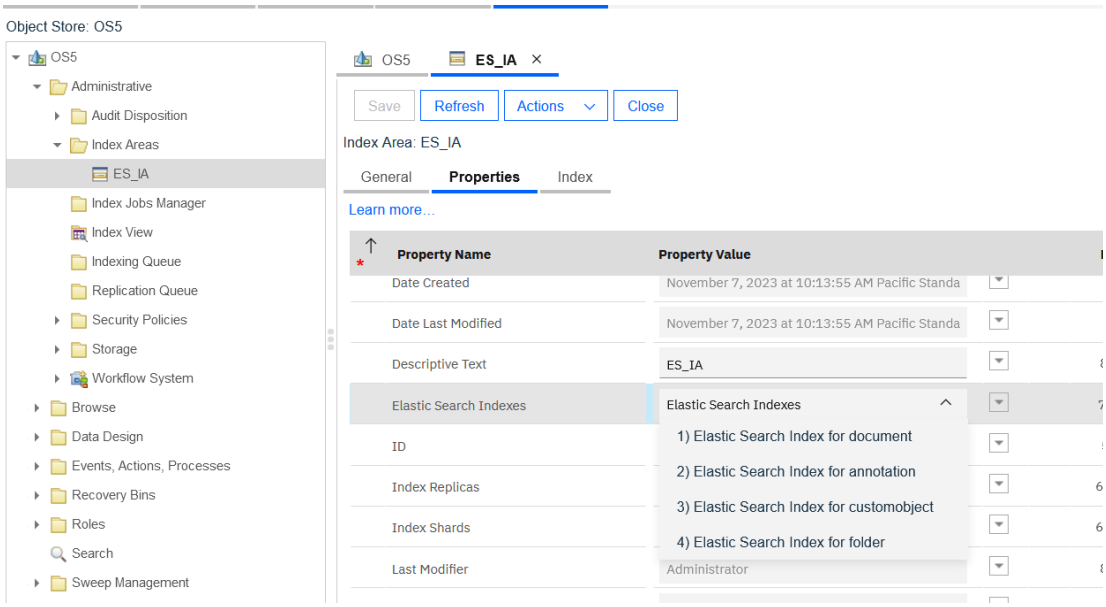
```
-Decm.elasticsearch.read.timeout.ms=120000
```

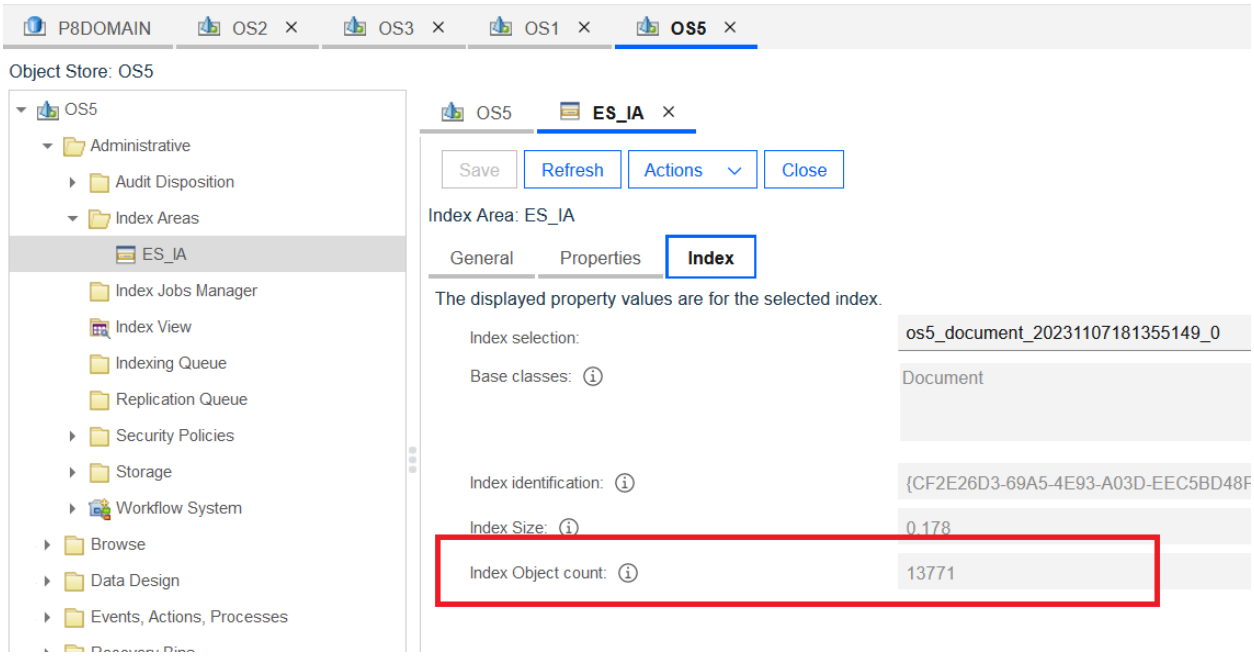
Index Statistics

You can retrieve the count of objects in an Elasticsearch index in ACCE by selecting the Properties Tab under Object Store > Administrative > Index Area > Index. Edit the ***Elasticsearch Indexes*** property and select an index from the list

- Option 1 for documents
- Option 2 for annotations
- Option 3 for custom objects
- Option 4 for folders

You can now find the count of objects under the **Indexed Object Count** property on the Index tab.





Alternatively, use the OpenSearch REST API to retrieve the count of objects in an Elasticsearch index. For example, query the /_cat/indices endpoint.

The indexing queue sweep counters (such as objects examined, and objects processed) cannot be used to determine the exact number of objects that are indexed. This is because of how a queue sweep handles batches, failures, retries, ignored documents, and so on.

Content Truncation

The Content Platform Engine limits the size of an indexing request sent to Elasticsearch by truncating the extracted content if needed. This is done to avoid exceeding the Elasticsearch maximum payload size (set with the http.max_content_length parameter). The default maximum size for Content Platform Engine properties and content is 77,000,000 characters. The maximum size can be altered using a JVM parameter. For example, the following statement limits to the size to 5,000,000:

```
-Decm.elasticsearch.content.payload.max.chars=5000000
```