Ingest-attachment plugin

The `ingest-attachment` plugin enables OpenSearch to extract content and other information from files using the Apache text extraction library Tika. Supported document formats include PPT, PDF, RTF, ODF, and many more Tika (Supported Document Formats).

The input field must be a base64-encoded binary.

Installing the plugin

Install the `ingest-attachment` plugin using the following command:

```
./bin/opensearch-plugin install ingest-attachment
```

Attachment processor options

Name	Required	Default	Description
`field`	Yes	N/A	The field from which to get the base64-encoded binary.
`target_field`	No	Attachment	The field that stores the attachment information.
`properties`	No	All properties	An array of properties that should be stored. Can be `content`, `language`, `date`, `title`, `author`, `keywords`, `content_type`, or `content_length`.
`indexed_chars`	No	`100_000`	The number of characters used for extraction to prevent fields from becoming too large. Use `-1` for no limit.
`indexed_chars_field`	No	`null`	The field name used to overwrite the number of chars being used for extraction, for example, `indexed_chars`.
`ignore_missing`	No	`false`	When `true`, the processor exits without modifying the document

Name	Required	Default	Description
			when the specified field doesn't exist.

Example

The following steps show you how to get started with the `ingest-attachment` plugin.

Step 1: Create an index for storing your attachments

The following command creates an index for storing your attachments:

```
PUT /example-attachment-index
{
  "mappings": {
    "properties": {}
  }
}
```

Step 2: Create a pipeline

The following command creates a pipeline containing the attachment processor:

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    {
      "attachment" : {
        "field" : "data"
      }
    }
  ]
}
```

Step 3: Store an attachment

Convert the attachment to a base64 string to pass it as `data`. In this example the `base64` command converts the file `lorem.rtf`:

```
base64 lorem.rtf
```

Alternatively, you can use Node.js to read the file to `base64`, as shown in the following commands:

```
import * as fs from "node:fs/promises";
import path from "node:path";

const filePath = path.join(import.meta.dirname, "lorem.rtf");
const base64File = await fs.readFile(filePath, { encoding: "base64" });

console.log(base64File);
```

The `.rtf` file contains the following base64 text:

```
Lorem ipsum dolor sit amet:
e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0=.
```

```
PUT example-attachment-index/_doc/lorem_rtf?pipeline=attachment
{
  "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0="
}
```

Query results

With the attachment processed, you can now search through the data using search queries, as shown in the following example:

```
POST example-attachment-index/_search
{
  "query": {
    "match": {
      "attachment.content": "ipsum"
    }
  }
}
```

OpenSearch responds with the following:

```
{
  "took": 5,
  "timed_out": false,
```

```
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 1.1724279,
    "hits": [
      {
        "_index": "example-attachment-index",
        "_id": "lorem_rtf",
        "_score": 1.1724279,
        "_source": {
          "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0=",
          "attachment": {
            "content_type": "application/rtf",
            "language": "pt",
            "content": "Lorem ipsum dolor sit amet",
            "content_length": 28
          }
        }
      }
    ]
  }
}
```


Extracted information

The following fields can be extracted using the plugin:

- content
- language
- date
- title
- author
- keywords
- content_type
- content_length

To extract only a subset of these fields, define them in the `properties` of the pipeline processor, as shown in the following example:

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    {
      "attachment" : {
        "field" : "data",
        "properties": ["content", "title", "author"]
      }
    }
  ]
}
```

Limit the extracted content

To prevent extracting too many characters and overloading the node memory, the default limit is `100_000`. You can change this value using the setting `indexed_chars`. For example, you can use `-1` for unlimited characters, but you need to make sure you have enough HEAP space on your OpenSearch node to extract the content of large documents.

You can also define this limit per document using the `indexed_chars_field` request field. If a document contains `indexed_chars_field`, it will overwrite the `indexed_chars` setting, as shown in the following example:

```
PUT _ingest/pipeline/attachment
{
  "description" : "Extract attachment information",
  "processors" : [
    {
      "attachment" : {
        "field" : "data",
        "indexed_chars" : 10,
        "indexed_chars_field" : "max_chars",
      }
    }
  ]
}
```

With the attachment pipeline configured, you can extract the default `10` characters with specifying `max_chars` in the request, as shown in the following example:

```
PUT example-attachment-index/_doc/lorem_rtf?pipeline=attachment
{
  "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0="
}
```

Alternatively, you can change the `max_char` per document in order to extract up to `15` characters, as shown in the following example:

```
PUT example-attachment-index/_doc/lorem_rtf?pipeline=attachment
{
  "data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0=",
  "max_chars": 15
}
```