Managing indexes

You index data using the OpenSearch REST API. Two APIs exist: the Index API and the `_bulk` API.

For situations in which new data arrives incrementally (for example, customer orders from a small business), you might use the Index API to add documents individually as they arrive. For situations in which the data flow is less frequent (for example, weekly updates to a marketing website), you might prefer to generate a file and send it to the `_bulk` API. For large numbers of documents, lumping requests together and using the `_bulk` API offers

superior performance. If your documents are exceptionally large, however, you might need to index them individually.

When indexing documents, the document `_id` must be 512 bytes or less in size.

Introduction to indexing

Before you can search data, you must *index* it. Indexing is the method by which search engines organize data for fast retrieval. The resulting structure is called, fittingly, an index.

In OpenSearch, the basic unit of data is a JSON *document*. Within an index, OpenSearch identifies each document using a unique ID.

A request sent to the Index API appears as follows:

```
PUT {index}/_doc/{id}
{ "A JSON": "document" }
```

A request to the `_bulk` API looks a little different, because you specify the index and ID in the bulk data:

```
POST _bulk
{ "index": { "_index": "<index>", "_id": "<id>" } }
{ "A JSON": "document" }
```

Bulk data must conform to a specific format, which requires a newline character (`\n`) at the end of every line, including the last line. This is the basic format:

```
Action and metadata\n
Optional document\n
Action and metadata\n
Optional document\n
```

> The document is optional, because `delete` actions don't require a document. The other actions (`index`, `create`, and `update`) all require a

> document. If you specifically want the action to fail if the document already exists, use the `create` action instead of the `index` action.

To index bulk data using the `curl` command, navigate to the folder where you have your file saved and run the following command:

```
curl -H "Content-Type: application/x-ndjson" -POST https://localhost:9200/
```

If any one of the actions in the `_bulk` API fail, OpenSearch continues to execute the other actions. Examine the `items` array in the response to figure out what went wrong. The entries in the `items` array are in the same order as the actions specified in the request.

OpenSearch automatically creates an index when you add a document to an index that doesn't already exist. It also automatically generates an ID if you don't specify an ID in the request. This simple example automatically creates the movies index, indexes the document, and assigns it a unique ID:

```
POST movies/_doc
{ "title": "Spirited Away" }
```

Automatic ID generation has a clear downside: because the indexing request didn't specify a document ID, you can't easily update the document at a later time. Also, if you run this request 10 times, OpenSearch indexes this document as 10 different documents with unique IDs. To specify an ID of 1, use the following request (note the use of PUT instead of POST):

```
PUT movies/_doc/1
{ "title": "Spirited Away" }
```

Because you must specify an ID, if you run this command 10 times, you still have only one document indexed with the `_version` field incremented to 10.

Indexes default to one primary shard and one replica. If you want to specify non-default settings, create the index before adding documents:

```
PUT more-movies
{ "settings": { "number_of_shards": 6, "number_of_replicas": 2 } }
```

Naming restrictions for indexes

OpenSearch indexes have the following naming restrictions:

- All letters must be lowercase.
- Index names can't begin with underscores (`_`) or hyphens (`-`).
- Index names can't contain spaces, commas, or the following characters:

  `:`, `"`, `*`, `+`, `/`, `\`, `|`, `?`, `#`, `>`, or `<`

Read data

After you index a document, you can retrieve it by sending a GET request to the same endpoint that you used for indexing:

```
GET movies/_doc/1

{
  "_index" : "movies",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "title" : "Spirited Away"
  }
}
```

You can see the document in the `_source` object. If the document is not found, the `found` key is `false` and the `_source` object is not part of the response.

To retrieve multiple documents with a single command, use the `_mget` operation. The format for retrieving multiple documents is similar to the `_bulk` operation, where you must specify the index and ID in the request body:

```
GET _mget
{
  "docs": [
    {
      "_index": "<index>",
      "_id": "<id>"
    },
    {
      "_index": "<index>",
      "_id": "<id>"
    }
  ]
}
```

To only return specific fields in a document:

```
GET _mget
{
  "docs": [
    {
      "_index": "<index>",
      "_id": "<id>",
      "_source": "field1"
    },
    {
      "_index": "<index>",
      "_id": "<id>",
      "_source": "field2"
    }
  ]
}
```

To check if a document exists:

```
HEAD movies/_doc/{doc-id}
```

If the document exists, you get back a `200 OK` response, and if it doesn't, you get back a `404 - Not Found` error.

Update data

To update existing fields or to add new fields, send a POST request to the `_update` operation with your changes in a `doc` object:

```
POST movies/_update/1
{
  "doc": {
    "title": "Castle in the Sky",
    "genre": ["Animation", "Fantasy"]
  }
}
```

Note the updated `title` field and new `genre` field:

```
GET movies/_doc/1

{
  "_index" : "movies",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 2,
  "_seq_no" : 1,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "title" : "Castle in the Sky",
    "genre" : [
      "Animation",
      "Fantasy"
    ]
  }
}
```

The document also has an incremented `_version` field. Use this field to keep track of how many times a document is updated.

POST requests make partial updates to documents. To altogether replace a document, use a PUT request:

```
PUT movies/_doc/1
{
```

```
  "title": "Spirited Away"
}
```

The document with ID of 1 will contain only the `title` field, because the entire document will be replaced with the document indexed in this PUT request.

Use the `upsert` object to conditionally update documents based on whether they already exist. Here, if the document exists, its `title` field changes to `Castle in the Sky`. If it doesn't, OpenSearch indexes the document in the `upsert` object.

```
POST movies/_update/2
{
  "doc": {
    "title": "Castle in the Sky"
  },
  "upsert": {
    "title": "Only Yesterday",
    "genre": ["Animation", "Fantasy"],
    "date": 1993
  }
}
```

Example response

```
{
  "_index" : "movies",
  "_type" : "_doc",
  "_id" : "2",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 3,
  "_primary_term" : 1
}
```

Each update operation for a document has a unique combination of the `_seq_no` and `_primary_term` values.

OpenSearch first writes your updates to the primary shard and then sends this change to all the replica shards. An uncommon issue can occur if multiple users of your OpenSearch-based application make updates to existing documents in the same index. In this situation, another user can read and update a document from a replica before it receives your update from the primary shard. Your update operation then ends up updating an older version of the document. In the best case, you and the other user make the same changes, and the document remains accurate. In the worst case, the document now contains out-of-date information.

To prevent this situation, use the `_seq_no` and `_primary_term` values in the request header:

```
POST movies/_update/2?if_seq_no=3&if_primary_term=1
{
  "doc": {
    "title": "Castle in the Sky",
    "genre": ["Animation", "Fantasy"]
  }
}
```

If the document is updated after we retrieved it, the `_seq_no` and `_primary_term` values are different and our update operation fails with a `409 – Conflict` error.

When using the `_bulk` API, specify the `_seq_no` and `_primary_term` values within the action metadata.

Delete data

To delete a document from an index, use a DELETE request:

```
DELETE movies/_doc/1
```

The DELETE operation increments the `_version` field. If you add the document back to the same ID, the `_version` field increments again. This

behavior occurs because OpenSearch deletes the document `_source`, but retains its metadata.

Next steps

- The Index Management (IM) plugin lets you automate recurring index management activities and reduce storage costs. For more information, see Index State Management.
- For instructions on how to reindex data, see Reindex data.