workshop

February 7, 2020

1 ElasticSearch Indices, Mapping y Operaciones CRUD

1.1 Maestría en Sistemas de Información

1.1.1 Almacenamiento masivo de datos Workshop indexación

El presente Workshop permitirá al estudiante familiarizarse con los conceptos de **indexación**, **maping** y las operaciones CRUD en **ElasticSearch** versión 7.5.2. Una vez conlcuido se espera del estudiante que esté en capacidad de:

- Conocer los conceptos de indexación y maping y aplicarlos en la configuración de bases customizables de ElasticSearch
- Utilizar los comandos de Kibana para las operaciones de: Create, Update, Read, Delete
- Utilizar los comandos de python de la librería de ElasticSearch para las operaciones de: Create, Update, Read, Delete

Para aplicar y afianzar los conocimientos, el estudiante realizará una aplicación Flask que permita Crear una base de datos de ElasticSearch, permita indexar información, actualizarla y borrarla.

1.2 Elasticsearch términos:

- **Documento:** Unidad básica de informacón indexable, recuperable. (**json**)
- **Índice:** Un índice de Elasticsearch (ES) es el nombre que recibe el espacio donde ES guarda los datos (documentos) . Está conformado por: una o más **shards**, que pueden tener cero o más **réplicas**.
- **Shard:** Son contenedores ubicados en un uno o múltiples nodos. Están conformados de segmentos de *Lucene*. Un índice se divide en uno o mas **shards**. Un **shard** es un Fragmento de un índice que contiene parte de los documentos
- **Replica:** Copia o duplicado de la información que existe en el **shard primario**.
- **Nodo:** Instancia de Elastic Search; Almacena todos los documentos.
- **Cluster:** Conjunto de uno o más nodos. Pueden existir diferentes índices dentro de un clúster.

- **Settings:** Configuración del índice: # de réplicas, # de shards a nivel de cluster. Diferentes indices pueden tener diferentes valores, mientras que todos los nodos del cluster tendrán los mismos valores.
- **Mappings:** Modelo de datos de un índice. Campos y Tipos de dato.

1.3 Indexación Invertida:

Concepto: * Es una estructura de datos que consiste en una lista de palabras únicas del documento empatadas con una tupla que identifica el documento y la posición en la que aparece el término. * Es un mapeo de **términos o palabras** a los **documentos**

1.4 Workshop:

1.4.1 Directrices

1. En las siguientes diapositivas se le presentará los comandos en **Kibana** para manipular *Elasticsearch **y los comandos de** Python** equivalentes para el mismo efecto.
2. Los índices creados con Kibana tendran la extensión **_ki**; como en: **empleados_ki** y las bases similares creadas con **Python** tendrán la extensión **_py**, como en: **empleados_py**
3. Recuerde activar el servicio elastic search con:

[2]:
```
!service elasticsearch start
```

4. Recuerde activar el servicio kibana con:

[3]:
```
!service kibana start
```

1.4.2 Creando un índice con Kibana

Para crear un indice con Kibana ejecutamos el código:

```
PUT estudiantes_maestria_ki
```

La respuesta obtenida es:

```
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "estudiantes_maestria_ki"
}
```

1.4.3 Creando un índice con Python

1. Importar la librería de Elasticsearch
2. Instanciar un objeto **es** de Elasticsearch con conexión al **localhost:9200**

[1]:
```
from elasticsearch import Elasticsearch
es = Elasticsearch("localhost:9200")
```

[5]:
```
resp = es.indices.create(index="estudiantes_maestria_py")
resp
```

[5]:
```
{'acknowledged': True,
 'shards_acknowledged': True,
 'index': 'estudiantes_maestria_py'}
```

1.4.4 Configuración de un índice

Por defecto elasticsearch genera un indice con 1 shard y 1 réplica. Para confirmarlo utilice el comando en Kibana:

```
GET estudiantes_maestria_ki/_settings?pretty
```

Que devuelve:

```
{
  "estudiantes_maestria_ki" : {
    "settings" : {
      "index" : {
        "creation_date" : "1581034801288",
        "number_of_shards" : "1",
        "number_of_replicas" : "1",
        "uuid" : "Au4DvS54Rr-gTrbA5ayN_A",
        "version" : {
          "created" : "7050299"
        },
        "provided_name" : "estudiantes_maestria_ki"
      }
    }
  }
}
```

En Python se obtiene la información con el comando:

[6]:
```
import json
resp = es.indices.get_settings(index="estudiantes_maestria_py")
resp
```

[6]:
```
{'estudiantes_maestria_py': {'settings': {'index': {'creation_date':
'1581087827919',
    'number_of_shards': '2',
    'number_of_replicas': '2',
    'uuid': 'Va-MmjrpRMOutnpPBzNZ-Q',
    'version': {'created': '7050299'},
    'provided_name': 'estudiantes_maestria_py'}}}}
```

1.4.5 Eliminar un índice

Para eliminar un índice ejecute el comando de Kibana

```
DELETE estudiantes_maestria_ki
```

Que devuelve:

```
{
  "acknowledged" : true
}
```

En python:

[3]:
```
resp = es.indices.delete(index="estudiantes_maestria_py")
resp
```

[3]:
```
{'acknowledged': True}
```

1.4.6 Settings

Si deseamos personalizar el número de shards y réplicas usamos un documento json declarando estos valores. La declaración en Kibana queda:

```
PUT estudiantes_maestria_ki
{
  "settings": {
    "number_of_replicas": 2,
    "number_of_shards": 2
  }
}
```

De forma similar en python esto se haría declarando un diccionario

[5]:
```
settings_dict = {"settings": {"number_of_replicas":2, "number_of_shards":2}}
resp = es.indices.create(index="estudiantes_maestria_py", body=settings_dict)
resp
```

[5]:
```
{'acknowledged': True,
 'shards_acknowledged': True,
 'index': 'estudiantes_maestria_py'}
```

1.4.7 Mappings

Es el proceso de definir cómo un documento y los campos que contiene se guardan e indexan. Un mapping contiene: 1. Meta-fields: son campos de meta datos como: _index, _type, _id y _source 2. Fields: lista de propiedades del documento. Cada campo (Field) tiene un tipo de dato que puede ser: * sencillos: text, keyword, date, long, double, boolean, ip * jerárquicos: object, nested * especializados: geo_point, geo_shape, completion

Nota 1: A partir de la versión 7, los mapping types están declarados obsoletos. **Nota 2:** No es necesaria la declaración de los campos y sus tipos de datos para crear el índice gracias al **dynamic_mapping**.

1.4.8 Mappings:

Suponga un documento estudiante con las propiedades siguientes: * nombre, * apellido * Cédula de identidad * email * edad * fecha de ingreso al curso

Declare el **explicit mapping** para el **index** *estudiantes_maestria_py*

En kibanna se procede:

```
PUT estudiantes_maestria_ki
{
  "mappings": {
    "properties": {
      "nombre": {"type": "text"},
      "apellido": {"type": "text"},
      "cedula_identidad": {"type": "keyword"},
      "email":{"type": "keyword"},
      "edad": {"type": "integer"},
      "fecha_ingreso":{"type": "date", "format": "date"}
    }
  }
}
```

En python se declara el diccionario o se lee el archivo json con la configuración:

[8]:
```
estudiante_mapping = {
                        "properties": {
                          "nombre": {"type": "text"},
                          "apellido": {"type": "text"},
                          "cedula_identidad": {"type": "keyword"},
                          "email":{"type": "keyword"},
                          "edad": {"type": "integer"},
                          "fecha_ingreso":{"type": "date", "format": "date"}
                        }
                      }
resp = es.indices.put_mapping(index="estudiantes_maestria_py",␣
 ↪body=estudiante_mapping)
resp
```

[8]:
```
{'acknowledged': True}
```

1.4.9 Indexación de Documentos

Escriba un documento json con sus datos personales e indexelos dentro del índice **estudiantes_maestria**.

En Kibana:

```
PUT estudiantes_maestria_ki/_doc/1
{
  "nombre": "Carl",
  "apellido": "Sagan",
  "cedula_identidad": 1234567890,
```

```
  "email":"carl.sagan@alien.com",
  "edad": 99,
  "fecha_ingreso":"1934-11-09"
}
```

En python:

[9]:
```
mis_datos = { "nombre": "Carl",
              "apellido": "Sagan",
              "cedula_identidad": 1234567890,
              "email":"carl.sagan@alien.com",
              "edad": 99,
              "fecha_ingreso":"1934-11-09"}

resp = es.index(index="estudiantes_maestria_py", body=mis_datos, id=1)
resp
```

[9]:
```
{'_index': 'estudiantes_maestria_py',
 '_type': '_doc',
 '_id': '1',
 '_version': 1,
 'result': 'created',
 '_shards': {'total': 3, 'successful': 1, 'failed': 0},
 '_seq_no': 0,
 '_primary_term': 1}
```

- **Nota 1:** Si no existiera el índice, este se crea al utilizar el comando **index** en python o PUT en la consola de Kibana, por lo que no es necesario crear de antemano el índice. Además la declaración de los campos y sus tipos puede ser actualizada.
- **Nota 2:** Si el campo id no es declarado, elasticsearch atribuye genera una cláve id automáticamente. Como ejercicio, ingrese otro estudiante pero esta vez no declare el valo de id.
- **Nota 3:** Si cambia los datos pero los indexa en el mismo **id**, el dato actual será actualizado. Elasticsearch en la operación de **GET** devuelve un metadato llamdo *version* que cuenta los cambios que recibe el respectivo documento.

[10]:
```
mis_datos = { "nombre": "Richard",
              "apellido": "Dawkins",
              "cedula_identidad": 9078563412,
              "email":"richard.dawkins@soyateo.com",
              "edad": 99,
              "fecha_ingreso":"1941-03-26"}

resp = es.index(index="estudiantes_maestria_py", body=mis_datos)
resp
```

[10]:
```
{'_index': 'estudiantes_maestria_py',
 '_type': '_doc',
 '_id': '_8KvIXAB7_oskeqkbHtF',
 '_version': 1,
```

```
 'result': 'created',
 '_shards': {'total': 3, 'successful': 1, 'failed': 0},
 '_seq_no': 1,
 '_primary_term': 1}
```

1.4.10 Ejercicio de Aplicación:

Dada la lista de documentos de estudiantes candidatos a la maestria, denominada "candidatos.json", escriba un código python de indexación

1.4.11 Solución:

[2]:
```
import json
with open('json_docs/candidatos.json', 'r') as f:
    candidatos_dict = json.load(f)

for i, candidato in enumerate(candidatos_dict):
    resp = es.index(index='master_candidates', body=candidato, id=i)
    print('Candidato {} ha sido {}'.format(i, resp['result']))
```

```
Candidato 0 ha sido created
Candidato 1 ha sido created
Candidato 2 ha sido created
Candidato 3 ha sido created
Candidato 4 ha sido created
Candidato 5 ha sido created
```

1.4.12 Indexación por Bulk

Bulk se utiliza para hacer múltiples operaciones de indexado, eliminación o actualización en una sola llamada para reducir el **overhead** en la indexación e incrementar la velocidad.

Esta operación requiere que el documento se escriba en formato NDJSON

```
POST _bulk
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_id" : "2" } }
{ "create" : { "_index" : "test", "_id" : "3" } }
{ "field1" : "value3" }
{ "update" : {"_id" : "1", "_index" : "test"} }
{ "doc" : {"field2" : "value2"} }
```

1.5 Utiliza Bulk en la consola de Kibana para ingresar los datos del ejercicio anterior

Para el ejericio anterior, se puede ingresar los candidatos en la consola de Kibana de la siguiente manera:

```
POST _bulk
{"index":{"_index": "master_candidates_kibana", "_id":1}}
{ "nombre": "Carl", "apellido": "Sagan", "cedula_identidad": 1234567890, "email":"carl.sagan@al
{"index":{"_index": "master_candidates_kibana", "_id":2}}
{ "nombre": "Richard", "apellido": "Dawkins", "cedula_identidad": 9078563412, "email":"richard
{"index":{"_index": "master_candidates_kibana", "_id":3}}
{ "nombre": "Stephen", "apellido": "Hawking", "cedula_identidad": 1234567890, "email":"stephen
{"index":{"_index": "master_candidates_kibana", "_id":4}}
{ "nombre": "Geoffrey", "apellido": "Hinton", "cedula_identidad": 4559879077, "email":"geoffrey
{"index":{"_index": "master_candidates_kibana", "_id":5}}
{"nombre": "Alan","apellido": "Turing", "cedula_identidad": 8803373682, "email":"alan.turing@tl
```

1.5.1 Utilizando bulk en Python para indexar

Se puede utilizar la información que se dispone en el archivo **candidatos.json** para constuir **on the fly** la estructura **NDJSON** y así indexar en **bulk** con python. Para esto cree una nueva base de datos **master_candidates_bulk_py**

[3]:
```python
from elasticsearch.helpers import bulk

actions = []
resp = {'response': 'Oops'}
for i, candidato in enumerate(candidatos_dict):
    actions.append({"_op_type": "index",
                    "_index": "master_candidates_bulk_py",
                    "_id": i+1,
                    "_source": candidato})
    result_tmp = bulk(es, actions=actions)
    resp = {'response': result_tmp[0]}
    print(resp)
```

```
{'response': 1}
{'response': 2}
{'response': 3}
{'response': 4}
{'response': 5}
{'response': 6}
```

1.6 Utiliza el add-on Elasticsearch Head de Google Chrome y verifica las indexaciones realizadas

1.6.1 Operación de Lectura

La operación de lectura permite recuperar un documento indexado.

En Kibana

```
GET master_candidates/_doc/3
```

En Python

[26]:
```
index_name = "master_candidates"
id_number = 8
try:
    resp = es.get(index=index_name, id=id_number)
except Exception as messenger:
    resp = {'response': "no posible leer id {} del index {}, debido a {}".
 ↪format(id_number, index_name, messenger)}
resp
```

[26]:
```
{'response': 'no posible leer id 8 del index master_candidates, debido a
NotFoundError(404,
\'{"_index":"master_candidates","_type":"_doc","_id":"8","found":false}\')'}
```

1.6.2 Operación de Actualización

Permite ejecutar operaciones para actualizar o eliminar un documento en un índice.
Para el ejercicio realizado añadir el campo "articulo_principal" en el $ID = 5$

Kibana

```
POST master_candidates_kibana/_update/5
{
  "doc":{
    "articulo_principal": "Entscheidungsproblem"
  }
}
```

Python

[29]:
```
doc = {"doc":{"articulo_principal": "Entscheidungsproblem"}}
resp = es.update(index="master_candidates", id=4, body=doc)
resp
```

[29]:
```
{'_index': 'master_candidates',
 '_type': '_doc',
 '_id': '4',
 '_version': 2,
 'result': 'updated',
 '_shards': {'total': 2, 'successful': 1, 'failed': 0},
 '_seq_no': 5,
 '_primary_term': 1}
```

1.6.3 Eliminación de Documentos

Kibana

```
DELETE master_candidates_kibana/_doc/6
```

Python

[5]:
```
index_name = "master_candidates_bulk_py"
id_number = 6
try:
    resp = es.delete(index=index_name, id=id_number)
except Exception as ex:
    print("No se pudo eliminar id={} en indice: {} debido a {}".
 ↪format(index_name, id_number, ex))
resp
```

[5]:
```
{'_index': 'master_candidates_bulk_py',
 '_type': '_doc',
 '_id': '6',
 '_version': 2,
 'result': 'deleted',
 '_shards': {'total': 2, 'successful': 1, 'failed': 0},
 '_seq_no': 21,
 '_primary_term': 1}
```

1.7 Taller de Evaluación:

Realizar una aplicación WEB con Flask para aplicar las operaciones de Create, Read, Index, Update y Delete. Desarrolle una página web para cada funcionalidad.

Para esto utilice los datos disponibles en formato json en: https://swapi.co/api/people/ u otra referencia de su preferencia.

Puede considerar como guía el código disponible en:

- https://tryolabs.com/blog/2015/02/17/python-elasticsearch-first-steps/
- https://github.com/LeninGF/flask_elasticsearch_7