Indexa tu información con OpenSearch y Python

José Luis Ariza Cabrera

¿Quién soy?

- Ingeniero electrónico de profesión
- Master en Project Management.
- Desarrollador de software desde el 2015.
- Organizador grupo Python Cali.
- Amante de los libros, el cine, las series, los videojuegos y el anime.

¿Qué es OpenSearch?

Es el fork Open Source de ElasticSearch y Kibana. Fue lanzado en Julio de 2021 luego de que ElasticSearch anunciara, en Enero de ese mismo año, que no se distribuiría más usando la licencia Apache 2.0; dejando de ser Open Source.

** Al ser un fork de ElasticSearch, es posible **migrar** de ElasticSearch a OpenSearch y viceversa*

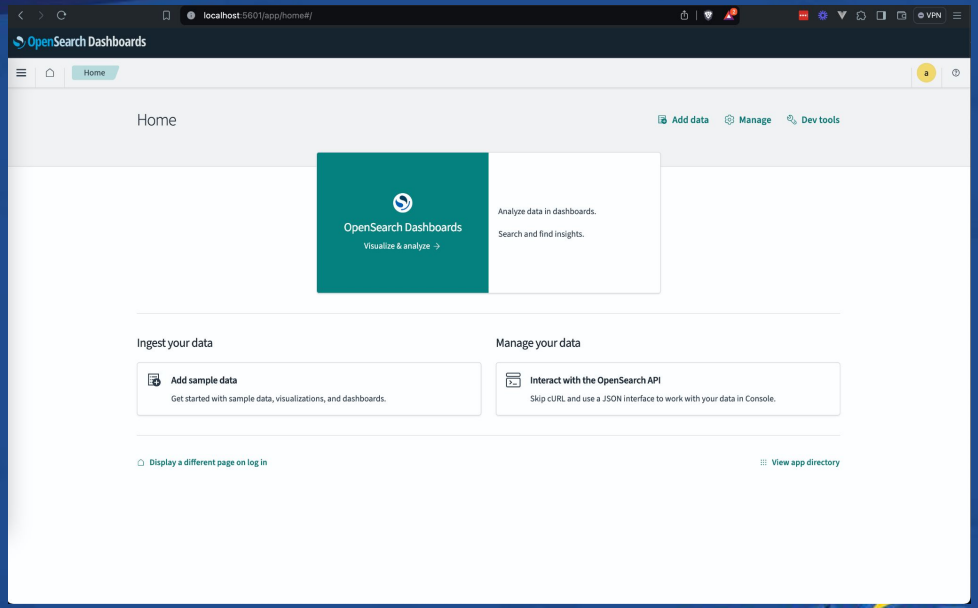
¿Qué es OpenSearch?

Consta de tres componentes:

1. OpenSearch - Motor de búsqueda
2. OpenSearch Dashboards - Interfaz de usuario
3. Data Prepper - Colector de datos

¿Qué es OpenSearch?

1. Ve a opensearch-container y corre `docker compose up`
2. Ve a http://localhost:5601/. Usa admin/admin para ingresar



Revisión de los componentes del proyecto base

- opensearch-container
  - compose.yml
    - Configuración de opensearch y opensearch dashboards usando docker

- django_project
  - local.yml
    - Configuración requerimientos del proyecto Django usando docker
  - opensearch_workshop/pokemons/
    - App Django que usaremos en el workshop.
      - Modelos:
        - Pokemon
        - PokeMove
        - PokeAbility
        - PokeType

Configurar lista base (sin OpenSearch)

1. Crear SearchForm
2. Crear ListView
   a. Crear template
   b. Registrar url
3. Integrar SearchForm con ListView
4. Quick Demo

Configuración django-opensearch-dsl

1. Añadir requerimiento `django-opensearch-dsl==0.6.2`
2. Agregar `django_opensearch_dsl` a las installed apps
3. Agregar settings de OpenSearch

```
# OPEN_SEARCH_CONFIGURATION
OPEN_SEARCH_USER = "admin"
OPEN_SEARCH_PASSWORD = "admin"
OPENSEARCH_DSL_AUTO_REFRESH = True
OPEN_SEARCH_HOST = "<YOUR LOCAL IP>"
OPENSEARCH_DSL = {
    "default": {
        "hosts": [{"host": OPEN_SEARCH_HOST, "port": 9200}],
        "http_compress": False,
        "http_auth": (OPEN_SEARCH_USER, OPEN_SEARCH_PASSWORD),
        "use_ssl": True,
        "verify_certs": False,
        "ssl_assert_hostname": False,
        "ssl_show_warn": False,
    },
}
```

Configuración django-opensearch-dsl

4. Crear OpenSearch Document para modelo `Pokemon`

```python
from django_opensearch_dsl import Document
from django_opensearch_dsl.registries import registry
from .models import Pokemon


@registry.register_document
class PokemonDocument(Document):
    class Django:
        model = Pokemon
        fields = ["pokemon_id","name"]

    class Index:
        name = "pokemons"  # Name of the Opensearch
index
```

Configuración django-opensearch-dsl

5. Crear índice de OpenSearch e indexar los datos

```
docker compose -f local.yml run --rm django python manage.py opensearch index create
docker compose -f local.yml run --rm django python manage.py opensearch document index
```

6. Crear índice en OpenSearch
7. Revisar datos indexados

Usando la consola de OpenSearch para traer resultados

```
GET pokemons/_search
{
   "query": {
     "bool": {
       "filter": [
         {
           "match": {
             "name": {
               "query": "pikach",
               "fuzziness": "AUTO"
             }
           }
         }
       ]
     }
   }
}
```

```
GET pokemons/_search
{
   "query": {
     "bool": {
       "filter": [
         {
           "query_string": {
             "query": "*saur*"
           }
         }
       ]
     }
   }
}
```

Usando el Document para traer resultados de OpenSearch

```
from opensearch_workshop.pokemons.documents import PokemonDocument
type(PokemonDocument.search()) #django_opensearch_dsl.search.Search
query = {
   "query": {
       "bool": {
           "filter": [{"match": {"name": {"query": "pikach", "fuzziness": "AUTO"}}}]
       }
   }
}
search = PokemonDocument.search().from_dict(query)
for hit in search:
   print(hit.pokemon_id, hit.name)
```

Usando el Document para traer resultados de OpenSearch

```
from opensearch_workshop.pokemons.documents import
PokemonDocument
query = {"query": {"bool": {"filter":
[{"query_string": {"query": "*saur*"}}]}}}

search = PokemonDocument.search().from_dict(query)

for hit in search:
   print(hit.pokemon_id, hit.name)

3 venusaur
1 bulbasaur
2 ivysaur
```

```
from opensearch_workshop.pokemons.documents import
PokemonDocument
query = {"query": {"bool": {"filter":
[{"query_string": {"query": "*saur*"}}]}}}

search = PokemonDocument.search().from_dict(
   query).sort("pokemon_id")

for hit in search:
   print(hit.pokemon_id, hit.name)

1 bulbasaur
2 ivysaur
3 venusaur
```

Usando el Document para traer resultados de OpenSearch

```
search = (
   PokemonDocument.search()
   .filter("query_string", query="*saur*")
   .filter("range", pokemon_id={"gt": 1})
   .sort("pokemon_id")
)
for hit in search:
   print(hit.pokemon_id, hit.name)

2 ivysaur
3 venusaur

search.to_queryset()
<QuerySet [<Pokemon: 2-ivysaur>, <Pokemon:
3-venusaur>]>
```

Integración básica del Document a interfaz de búsqueda

```python
...
from .documents import PokemonDocument

class PokemonListView(FormMixin, ListView):
    ...
    def get_queryset(self) -> QuerySet[Any]:
        form = SearchPokemonForm(self.request.GET)
        queryset = super().get_queryset()
        if form.is_valid():
            data = form.cleaned_data
            name = data["name"]
            os_query = (
                PokemonDocument.search()
                .filter("query_string",query=f"*{name}*",fields=["name",],)
                .sort("pokemon_id")
                  # los resultados de opensearch se deben paginar
                  # pero está fuera del scope de este taller
                .extra(size=1000, track_total_hits=True)
            )
            queryset = os_query.to_queryset()
        return queryset
```

Integración avanzada del Document a interfaz de búsqueda

- OpenSearch permite representar relaciones:
  - NestedField
    - ManyToMany
    - ManyToOne
  - ObjectField
    - ForeignKey
    - OneToOne

```python
class PokemonDocument(Document):
    types = fields.NestedField(
        properties={
            "id": fields.IntegerField(),
            "name": fields.TextField(),
        }
    )
    moves = fields.NestedField(
        properties={
            "id": fields.IntegerField(),
            "name": fields.TextField(),
        }
    )
    abilities = fields.NestedField(
        properties={
            "id": fields.IntegerField(),
            "name": fields.TextField(),
        }
    )
    ...
```

* Al actualizar el Document, debes correr `manage.py opensearch index create` y `manage.py opensearch document index` para actualizar la información en OpenSearch

Integración avanzada del Document a interfaz de búsqueda

- Al usar NestedField u ObjectField, podemos usar los campos de los objetos relacionados en la búsqueda.
  - Para objectField, podemos usar “<object>.<property>” directamente en el search
  - Para NestedField, debemos construir la query usando nested queries

```
query = Q(
    "bool",
    should=[
        Q("query_string", query=f"*{name}*", fields=["name"]),
        Q("nested", path="types", query=Q("query_string", query=f"*{name}*",
fields=["types.name"])),
        Q("nested", path="moves", query=Q("query_string", query=f"*{name}*",
fields=["moves.name"])),
        Q("nested", path="abilities", query=Q("query_string", query=f"*{name}*",
fields=["abilities.name"])),
    ],
    minimum_should_match=1,
)
os_query = (
    PokemonDocument.search()
    .query(query)
    .sort("pokemon_id")
    # los resultados de opensearch se deben paginar
    # pero está fuera del scope de este taller
    .extra(size=1000, track_total_hits=True)
)
```

Uso De OpenSearch fuera de Django

- Se usa el cliente python oficial
  - *django-opensearch-dsl* es un wrapper para Django basado en dicho cliente
- Su uso permite usar OpenSearch directamente a través de Query DSL. Sin embargo, requiere más definiciones así como requiere que los procesos de indexado sean explícitos.
- Al ser más general su uso, es más versátil.

Uso De OpenSearch fuera de Django

```python
from opensearchpy import OpenSearch
from opensearchpy.helpers.search import Search

class OpenSearchConnector (object):

    def __init__ (self, *args, **kwargs):
        self.client = OpenSearch (
            hosts=[{"host": "192.168.64.1" , "port": 9200}],
            http_compress =False,
            http_auth =("admin" , "admin"),
            use_ssl =True,
            verify_certs =False,
            ssl_assert_hostname =False,
            ssl_show_warn =False,
        )

    def create_index (self, index_name , index_body ):
        response = self.client.indices.create (index_name , body=index_body )
        return response

    def delete_index (self, index_name ):
        response = self.client.indices.delete (index_name )
        return response
```

```python
class OpenSearchConnector(object):
...
    def index_document(self, index_name, document, refresh=True):
        response = self.client.index(index=index_name, body=document,
refresh=refresh)
        return response

    def delete_document(self, index_name, document_id, refresh=True):
        response = self.client.delete(index=index_name, id=document_id,
refresh=refresh)
        return response

    def search(self, index_name, query):
        search = Search(
            using=self.client,
            index=index_name,
        )
        response = search.query(query).execute()
        return response
```

Uso De OpenSearch fuera de Django

```python
from opensearch_integration  import OpenSearchConnector
connector  = OpenSearchConnector ()
connector .create_index (
   "books",
   {
       "mappings" : {
           "properties" : {
               "title" : {"type" : "text"},
               "author" : {"type" : "text"},
           }
       },
   },
)
books  = [
   {"title" : "It Ends With Us" , "author" : "Colleen Hoover" },
   {"title" : "Atomic Habits" , "author" : "James Clear" },
   {"title" : "The 48 Laws of Power" , "author" : "Robert Greene and Joost Elffers"  },
   {"title" : "The Subtle Art of Not Giving a F*ck"  , "author" : "Mark Manson" },
   {"title" : "Harry Potter and the Philosopher's Stone"  , "author" : "J. K.
Rowling" },
]
# Esto puede ser hecho en un solo request usando una operación `bulk`
for book in books :
   connector. index_document ("books" , book)
```

```python
from opensearchpy.helpers.query import Q
connector.search(
   "books",
   {
       "query_string": {
           "query": "*harry*",
           "fields": ["title", "author"]
       }
   }
)
connector.search(
   "books",
   Q(
       "query_string",
       query=f"*harry*", fields=["title", "author"]
   )
)
```

5.1-pure-python-opensearch-integration

Uso De OpenSearch fuera de Django

```python
from flask import Flask, render_template, request
from opensearch_integration import OpenSearchConnector
from opensearchpy.helpers.query import Q
app = Flask(__name__)

@app.route("/")
def pokemons():
    search = request.args.get("search", "")
    connector = OpenSearchConnector()
    query = ...
    pokemons = connector.search("pokemons", query)
    return render_template("search.html",
**{"pokemons": pokemons, "search": search})
```

```python
    query = Q(
        "bool",
        should=[
            Q("query_string" , query=f"*{search}*", fields=["name"]),
            Q(
                "nested",
                path="types",
                query=Q("query_string" , query=f"*{search}*",
fields=["types.name"]),
            ),
            Q(
                "nested",
                path="moves",
                query=Q("query_string" , query=f"*{search}*",
fields=["moves.name"]),
            ),
            Q(
                "nested",
                path="abilities",
                query=Q("query_string" , query=f"*{search}*",
fields=["abilities.name"]),
            ),
        ],
        minimum_should_match=1,
    )
```

5.1-pure-python-opensearch-integration

Conclusiones y Consideraciones

- Usar un cliente para OpenSearch simplifica y agiliza su uso frente al uso de su REST API.
- Gracias al poder y la versatilidad de python, los datos obtenidos de OpenSearch pueden ser procesados por diversos sistemas y para diversos usos: Web, DS, ML, etc.
- Aunque el uso de OpenSearch puede mejorar el proceso de búsqueda, es una dependencia adicional que hay que considerar en el proceso.

Conclusiones y Consideraciones

- Go live:
  - Usando Docker
  - Usando Distros basadas en Debian
    - Usando el playbook de ansible oficial
  - Usando AWS OpenSearch Service

pending

¿Preguntas?

Referencias

- OpenSearch
  - https://opensearch.org/docs/latest/about/
- Django-opensearch-dsl
  - https://github.com/Codoc-os/django-opensearch-dsl
- Opensearch-py
  - https://github.com/opensearch-project/opensearch-py
- PokeApi
  - https://pokeapi.co/

Slides Theme: Prezentr