Lab 09

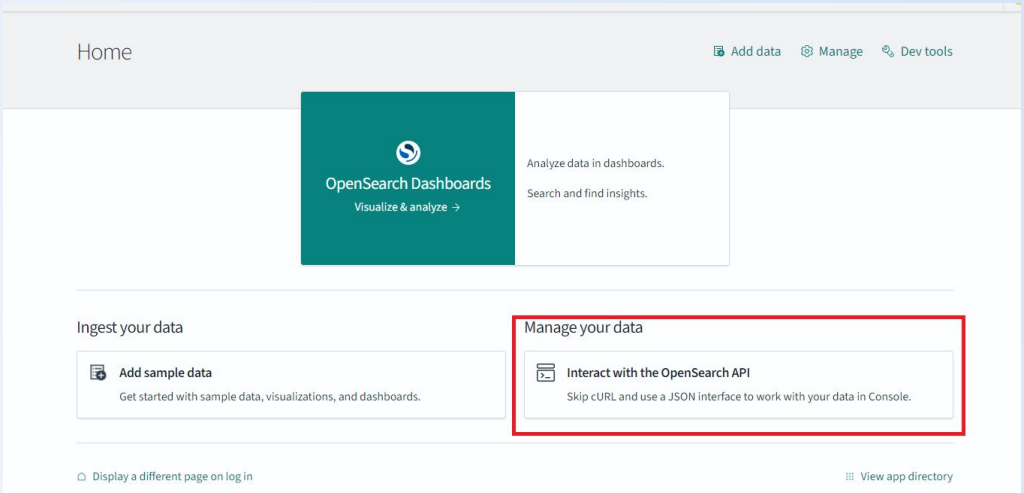
Creating Your First Agent and Model in OpenSearch

Lab Objectives

- Add a connector to our externally hosted model
- Configure the model in a search pipeline
- Create request to test inference against our pipeline and model
- Review all the changes

Log into OpenSearch

- Log into OpenSearch with the credentials that were distributed to you in class
  - https://20.106.177.39.c.hossted.app
- While on the homepage navigate to the API servers which will be located under the developer tools



Add a connector

```
POST /_plugins/_ml/connectors/_create
{
  "name": "LMStudio Connector - AbuseIPDB KNN",
  "description": "Production LMStudio Host",
  "version": "2",
  "protocol": "http",
  "parameters": {
    "endpoint": "http://20.106.179.227:5000",
    "model": "openvoid/prox-7b-dpo-gguf",
    "max tokens": 2000,
    "temperature": 0.5
  },
  "credential": {
    "openAI_key": "fakeapikey"
    },
  "client_config": {
    "connection_timeout": 1600,
    "read_timeout": 1800,
    "retry_backoff_millis": 1800000,
    "retry_timeout_seconds": 1800
  },
  "actions": [
    {
      "action_type": "predict",
      "method": "POST",
      "url": "${parameters.endpoint}/v1/chat/completions",
      "request_body": """{ "model": "${parameters.model}", "messages":
${parameters.messages}, "temperature": ${parameters.temperature} }"""
    }
  ]
}
```

POST /_plugins/_ml/connectors/_create: Creates a new connector

- **name**: The name of the connector (`"OpenAI Chat Connector"`)
- **description**: A brief description of the connector (`"LMStudio Connector for lab 9"`)
- **version**: Version number of the connector (`"2"`)
- **protocol**: Communication protocol used (`"http"`)
- **parameters**: Contains the configuration parameters for the connector
  - **endpoint**: The API endpoint URL (`"http://20.106.179.227:5000"`)
  - **model**: The model to be used (`"openvoid/prox-7b-dpo-gguf"`)
  - **temperature**: The temperature setting for the model (`0.5`)
- **actions**: Defines the actions the connector can perform
  - **action_type**: The type of action (`"predict"`)
  - **method**: HTTP method to be used (`"POST"`)
  - **url**: URL template for making predictions (`"${parameters.endpoint}/v1/chat/completions"`)

Add a connector

- We will pay close attention to the request body
- This is structured in a particular way so that our Mistral model can accept the parameters
  - Here is the example code pulled from the OpenAPI Server of what the model is expecting
- When changes models or OpenAPI servers you must make sure that you modify your requests to fit the formatting to fit what it expected
- **Make sure to save the connector_id output from this step**

URL: `http://localhost:5000/v1/chat/completions`
HTTP Method: POST
Headers:

- `Content-Type: application/json`

Data (JSON payload):

- **model**: `"openvoid/prox-7b-dpo-gguf"`
- **messages**:
  - **message 1**:
    - **role**: `"system"`
    - **content**: `"Always answer in rhymes."`
  - **message 2**:
    - **role**: `"user"`
    - **content**: `"Introduce yourself."`
- **temperature**: 0.7
- **max_tokens**: -1 (likely means no limit on the number of tokens

Registering the Model

Using the connector ID from the last step, we can register a model. Utilizing this JSON, a model_ID should be generated. **Ensure that this ModelID is saved.**

```
POST /_plugins/_ml/models/_register
{
  "name": "abuseipdb-test-rag-Development",
  "function_name": "remote",
  "description": "Mistral Model on Development
LMStudio Host",
  "connector_id": "$CONNECTOR_ID"
}
```

Deploying the Model

Now that we have a model ID- this command will officially deploy the model's connector and allow OpenSearch to query the LLM.

```
POST /_plugins/_ml/models/Vd3lSJEBfzbu3jp_pq6p/_deploy
```

Execute a query against the model

```
POST /_plugins/_ml/models/nv405ZABdV33CmeAYYpy/_predict
{
  "parameters": {
    "messages": [
      {
        "role": "assistant",
        "content": "look for SQLi in the index"
      }
    ],
    "temperature": 0.5
  }
}
```

Request Body (in JSON format):

- **parameters**:
  - **messages**:
    - **role**: `"assistant"`
    - **content**: `"look for SQLi in the index"`
  - **temperature**: `0.5`

Purpose:

- **messages**: Contains an array of message objects for the model. In this case, it includes a single message with the role of `"assistant"` and a request to "look for SQLi in the index"
- **temperature**: Controls the randomness of the model's responses. A value of `0.5` indicates a balance between deterministic and random responses

Execute a query against the model

- You will see a result similar to the screenshot at the right
- If you do not then work backwards through the steps and verify that you specified the correct **model id**

Final notes

- You are welcome to attempt to modify the query to see what results you get!
- Keep in mind that there may be a queue for the model to respond as other students may be also querying at the same time as you–be patient!

Lab End