Lab 10

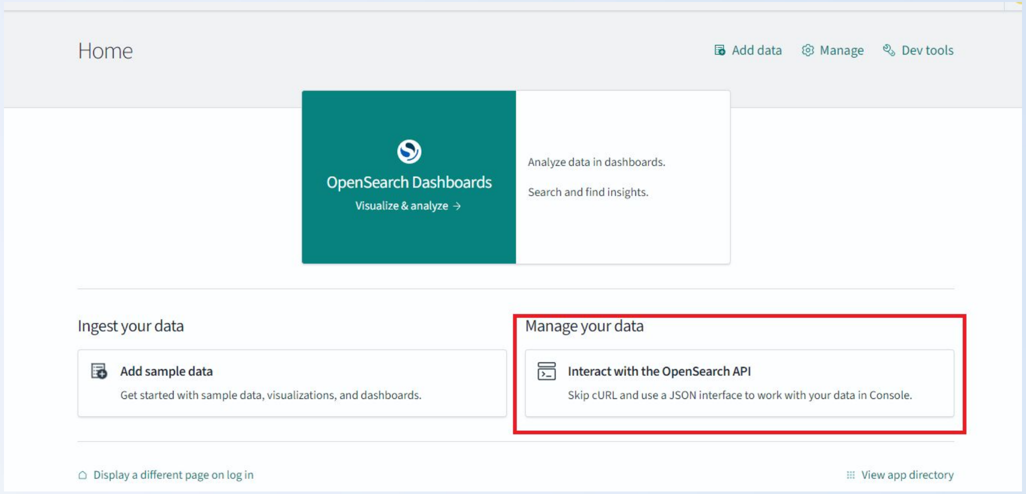
Creating a RAG Pipeline

Lab Objectives

- Verify the cluster is configured for conversation
- Create a connector/register the model
- Create a conversational flow agent
- Configure multiple Knowledge Bases

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



Analyze data in dashboards.

Search and find insights.

Ingest your data

Add sample data
Get started with sample data, visualizations, and dashboards.

Manage your data

Interact with the OpenSearch API
Skip cURL and use a JSON interface to work with your data in Console.

Prerequisites

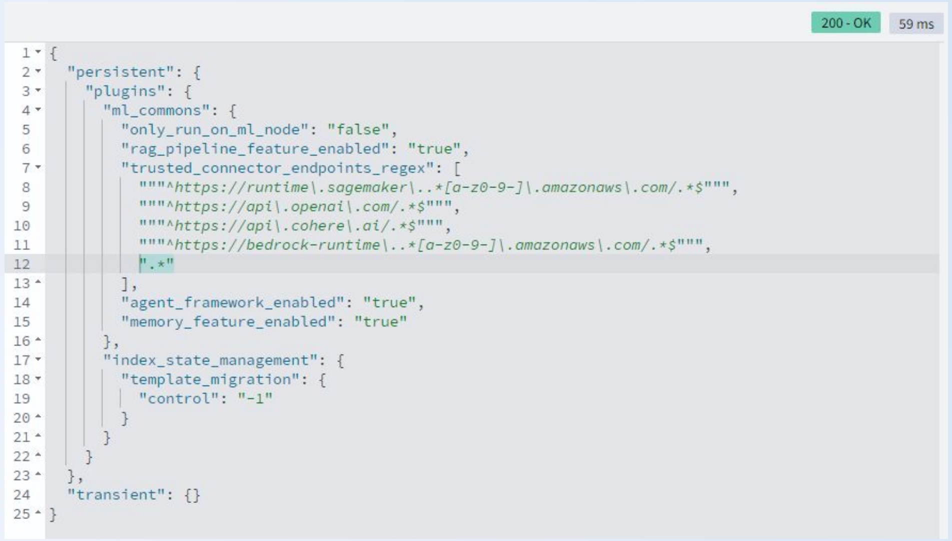
```
PUT /_cluster/settings
{
  "persistent": {
    "plugins.ml_commons.memory_feature_enabled": false,
    "plugins.ml_commons.rag_pipeline_feature_enabled": true,
    "plugins.ml_commons.agent_framework_enabled": true,
    "plugins.ml_commons.only_run_on_ml_node": false,
    "plugins.ml_commons.connector_access_control_enabled": true,
    "plugins.ml_commons.model_access_control_enabled": false,
    "plugins.ml_commons.native_memory_threshold": 99,
    "plugins.ml_commons.allow_registering_model_via_local_file": true,
    "plugins.ml_commons.allow_registering_model_via_url": true,
    "plugins.ml_commons.model_auto_redeploy.enable":true,
    "plugins.ml_commons.model_auto_redeploy.lifetime_retry_times": 10
  }
}

GET _cluster/settings
```

- **These are the settings we've used to prepare OpenSearch for this course**
- **They have been pre-applied already, so there is no need to run this again**
- **The next slide has explanations tabled out for each setting**

Setting	Value	Meaning
plugins.ml_commons.memory_feature_enabled	FALSE	Disables memory usage tracking feature for ML models.
plugins.ml_commons.rag_pipeline_feature_enabled	TRUE	Enables RAG (Retrieval-Augmented Generation) pipeline feature.
plugins.ml_commons.agent_framework_enabled	TRUE	Enables the agent framework (for multi-step LLM agents/workflows).
plugins.ml_commons.only_run_on_ml_node	FALSE	Allows ML tasks to run on non-ML nodes as well (useful for smaller clusters).
plugins.ml_commons.connector_access_control_enabled	TRUE	Enables access control for ML connectors (restricts use by permissions).
plugins.ml_commons.model_access_control_enabled	FALSE	Disables model-level access control (i.e., any user can use any model).
plugins.ml_commons.native_memory_threshold	99	Allows up to 99% of native memory to be used for ML models before throttling or eviction (risky, but performance-optimized).
plugins.ml_commons.allow_registering_model_via_local_file	TRUE	Enables registering models using local file paths (useful in air-gapped or dev environments).
plugins.ml_commons.allow_registering_model_via_url	TRUE	Enables registering models from external URLs (e.g., HuggingFace models).
plugins.ml_commons.model_auto_redeploy.enable	TRUE	Enables automatic redeployment of failed or removed models.
plugins.ml_commons.model_auto_redeploy.lifetime_retry_times	10	Sets the max number of auto-redeploy attempts per model to 10.

Prerequisites

- This step allows us to whitelist the IP for our hosted LMStudio box
    - We've been a bit lazy and just done a .* due to Azure's floating IP space.
- Once again, we've already ran these for you



```
{
  "persistent": {
    "plugins": {
      "ml_commons": {
        "only_run_on_ml_node": "false",
        "rag_pipeline_feature_enabled": "true",
        "trusted_connector_endpoints_regex": [
          """^https://runtime\.sagemaker\..*[a-z0-9-]\.amazonaws\.com/.*$""",
          """^https://api\.openai\.com/.*$""",
          """^https://api\.cohere\.ai/.*$""",
          """^https://bedrock-runtime\..*[a-z0-9-]\.amazonaws\.com/.*$""",
          ".*"
        ],
        "agent_framework_enabled": "true",
        "memory_feature_enabled": "true"
      },
      "index_state_management": {
        "template_migration": {
          "control": "-1"
        }
      }
    }
  },
  "transient": {}
}
```

Register a text embedding model

- POST /_plugins/_ml/models/_register
- {
-   "name": "huggingface/sentence-transformers/all-mpnet-base-v2",
-   "version": "1.0.2",
-   "model_format": "TORCH_SCRIPT"
- }

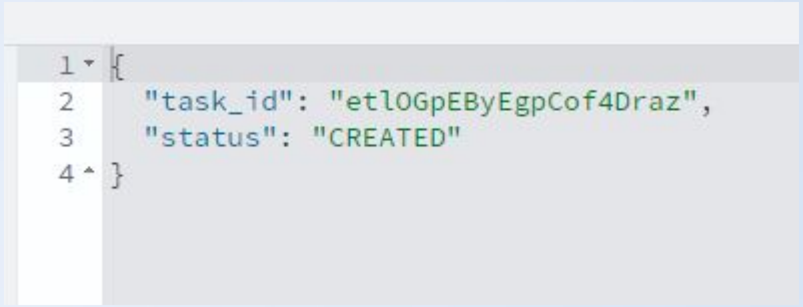
- GET /_plugins/_ml/tasks/<task id>

- POST /_plugins/_ml/models/your_text_embedding_model_id/_deploy

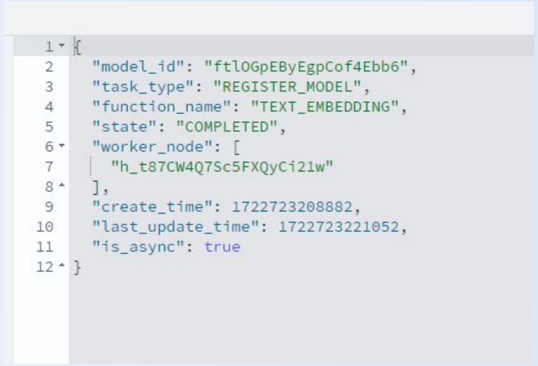
- `"Name"`
  - Specifies the model to register. In this case, it's the **all-mpnet-base-v2** sentence transformer from the Hugging Face `sentence-transformers` library. This model is used for **semantic text embeddings**.
- `"Version"`
  - The version you're assigning **within OpenSearch**, not necessarily related to Hugging Face's versioning.
- `"Model_format"`
  - This tells OpenSearch how the model is packaged. `TORCH_SCRIPT` means it's a **TorchScript-serialized** PyTorch model. TorchScript is a format that allows PyTorch models to run outside of Python environments.
-

Register a text embedding model

- When you add the text embedding model the server will return a task ID
- Run the GET command with the task ID to retrieve the **model id**
- Use the **model id** to deploy the model



```
{
  "task_id": "etlOGpEByEgpCof4Draz",
  "status": "CREATED"
}
```



```
{
  "model_id": "ftlOGpEByEgpCof4Ebb6",
  "task_type": "REGISTER_MODEL",
  "function_name": "TEXT_EMBEDDING",
  "state": "COMPLETED",
  "worker_node": [
    "h_t87CW4Q7Sc5FXQyCi21w"
  ],
  "create_time": 1722723208882,
  "last_update_time": 1722723221052,
  "is_async": true
}
```

Test the model

- POST /_plugins/_ml/models/your_text_embedding_model_id/_predict
- {
- "text_docs":["today is sunny"],
- "return_number": true,
- "target_response": ["sentence_embedding"]
- }

Request Body (in JSON format):

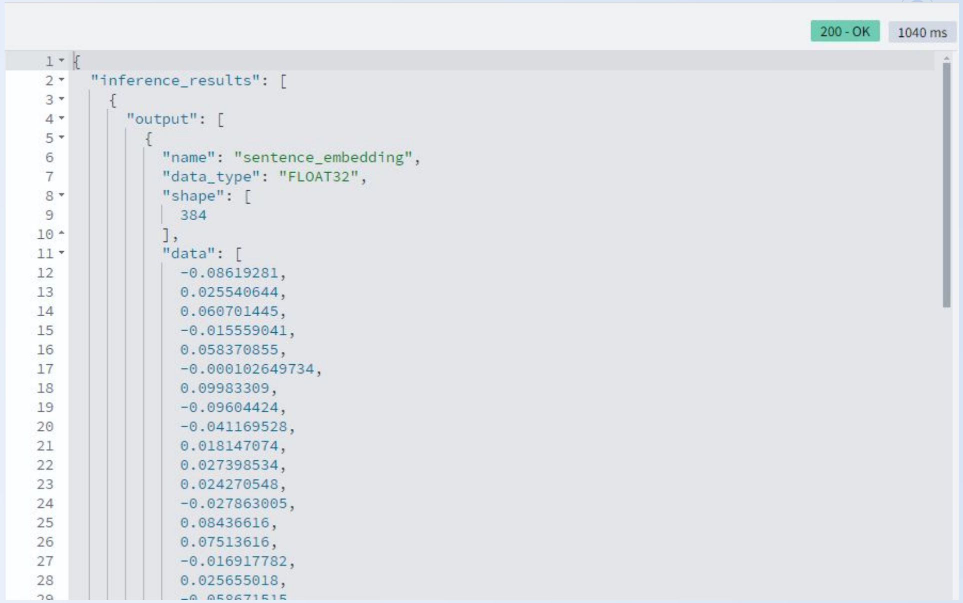
- **text_docs**:
  - An array of text documents to be processed, in this case, `["today is sunny"]`
- **return_number**:
  - A boolean value, `true`, indicating that the response should include numerical data (e.g., embeddings)
- **target_response**:
  - An array specifying the desired output format, here `["sentence_embedding"]`, indicating that the model should return sentence embeddings

Purpose:

- **text_docs**: Provides the input text to be embedded by the model
- **return_number**: Directs the model to return numerical results, likely the embeddings
- **target_response**: Specifies that the response should include sentence embeddings for the provided text documents

Test the model

- Running a predict request will allow us to test the model
- You will see inference results return and it will be a series of numbers which means the model is working properly


200 - OK 1040 ms

```
{
  "inference_results": [
    {
      "output": [
        {
          "name": "sentence_embedding",
          "data_type": "FLOAT32",
          "shape": [
            384
          ],
          "data": [
            -0.08619281,
            0.025540644,
            0.060701445,
            -0.015559041,
            0.058370855,
            -0.000102649734,
            0.09983309,
            -0.09604424,
            -0.041169528,
            0.018147074,
            0.027398534,
            0.024270548,
            -0.027863005,
            0.08436616,
            0.07513616,
            -0.016917782,
            0.025655018,
```

Create an ingest pipeline

```
PUT /_ingest/pipeline/lab10_conversational_pipeline
{
   "description": "text embedding pipeline",
   "processors": [
      {
         "text_embedding": {
            "model_id": "your_text_embedding_model_id",
            "field_map": {
               "population_description":
"population_description_embedding"
            }
         }
      }
   ]
}
```

Request Body (in JSON format):

- **description**: `"text embedding pipeline"`
  - Provides a description of the pipeline
- **processors**:
  - An array of processors to be applied in the pipeline. In this case, it contains one processor:
    - **Text_embedding**:
    - **model_id**: `"your_text_embedding_model_id"`
      - Specifies the ID of the text embedding model to use
    - **Field_map**:
    - **population_description**: `"population_description_embedding"`
    - Maps the input field `"population_description"` to the output field `"population_description_embedding"`, where the embedding will be stored

Purpose:

- This allows any data that gets sent to the knn index to be vectorized for retrieval by the embedding model.

Use existing LLM

- We will use the LLM we configured in lab 6 so please look back in your notes to grab the **model id**
- We will test the LLM first to verify the agent is working properly before proceeding
- If you cannot find the **model id** refer to the OpenSearch commands document to find the command to search for model ids and look for yours

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

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"`)

Register the Model

- POST /_plugins/_ml/$CONNECTORID/_register
- {
-   "name": "abuseipdb-knn-rag",
-   "function_name": "remote",
-   "description": "openvoid Model on Development LMStudio Host",
-   "connector_id": "AoJReJgBlaNTCsEI__pE"
- }
-
-
-
-
- POST /_plugins/_ml/models/b4LtXpgBlaNTCsEIo-ma/_deploy

POST /_plugins/_ml/$CONNECTORID/_register:
Registers the recently created connector

- **"Name"**
  - **The name you're assigning to this model inside OpenSearch.**
- **"function_name"**
  - **Set to "remote", meaning the model is hosted externally and accessed via an HTTP API (e.g., LM Studio, Ollama, etc.).**
- **"Description"**
  - **A human-readable description — here, describing it as hosted on a development LMStudio instance running a model from openvoid.**
- **"Connector_id"**
  - **The ID of the previously registered ML connector (from your earlier query). This connector knows how to talk to the model endpoint (e.g., what URL to call, what parameters to send).**

POST /_plugins/_ml/$CONNECTORID/_deploy:
Deploys the recently created connector

Register RAG Tool agent

- This request is very large command and we opted to put it in the **Day2-OpenSearch-Commands.txt** document on the Day 2 git repo
- Once you run the command then proceed to the next slide to explain the parameters

Register conversational flow agent

Request Body (in JSON format):

- **name**: `"Test_Agent_For_RAG"`
  - Specifies the name of the agent being registered
- **type**: `"conversational_flow"`
  - Indicates the type of agent, which handles conversational interactions
- **description**: `"this is a test agent"`
  - Provides a description of the agent
- **app_type**: `"rag"`
  - Specifies the application type, in this case, "retrieval-augmented generation" (RAG)
- **Memory**:
- **type**: `"conversation_index"`
- Defines the memory type used by the agent for storing conversation history
- **Tools**:
- An array of tools used by the agent:
- **Tool 1**:
  - **type**: `"VectorDBTool"`
    - Indicates a tool for querying a vector database
  - **Parameters**:
    - **model_id**: `"uCSF6pABd2qN_ubfPaSt"`
      - Specifies the ID of the model used for embedding or searching.
    - **index**: `"my_test_data"`
      - Refers to the index in the vector database
    - **embedding_field**: `"embedding"`
      - The field containing embeddings in the vector database
    - **source_field**: `["text"]`
      - Specifies the source field to retrieve text from.
    - **input**: `"${parameters.question}"`
      - Placeholder for dynamic input, which is the user's question

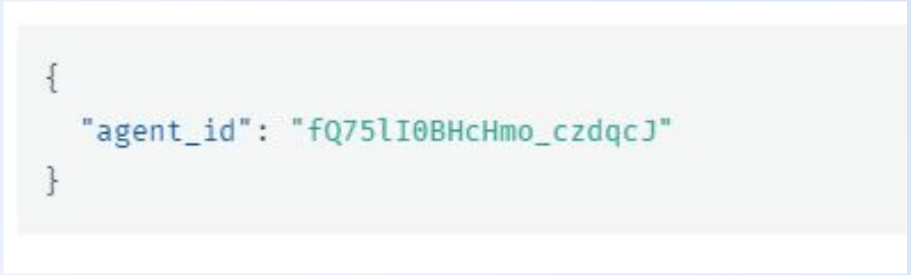
- **type**: `"MLModelTool"`
  - Indicates a tool for interacting with an ML model
- **description**: `"A general tool to answer any question"`
  - Provides a description of the tool
- **Parameters**:
  - **model_id**: `"1v4E65ABdV33CmeA85O-"`
    - Specifies the ID of the ML model used for generating responses
  - **response_filter**: `"$..choices[0].message.content"`
    - JSONPath to extract the content of the response from the model
  - **Messages**:
    - An array defining the context and role of messages:
      - **message 1**:
      - **role**: `"assistant"`
      - **content**: A detailed prompt instructing the assistant to answer questions based on the given context, with placeholders for context and history
  - **temperature**: `0.5`
    - Controls the randomness of the model’s responses

Purpose:

- **name**: Names the agent for identification
- **type**: Defines the agent’s function as handling conversational flows
- **description**: Describes the agent’s purpose
- **app_type**: Indicates the specific application of the agent (RAG)
- **memory**: Specifies how conversation history is stored
- **tools**: Lists and configures the tools the agent uses for embedding and answering questions

Register conversational flow agent

- OpenSearch will return with an **agent_id**
- This is **id** that we need to start our conversation



```
{
  "agent_id": "fQ75lI0BHcHmo_czdqcJ"
}
```

Run the agent

- POST /_plugins/_ml/agents/your_agent_id/_execute
- {
-   "parameters": {
-     "question": "Are there root login attempts?"
-   }
- }

Request Body (in JSON format):

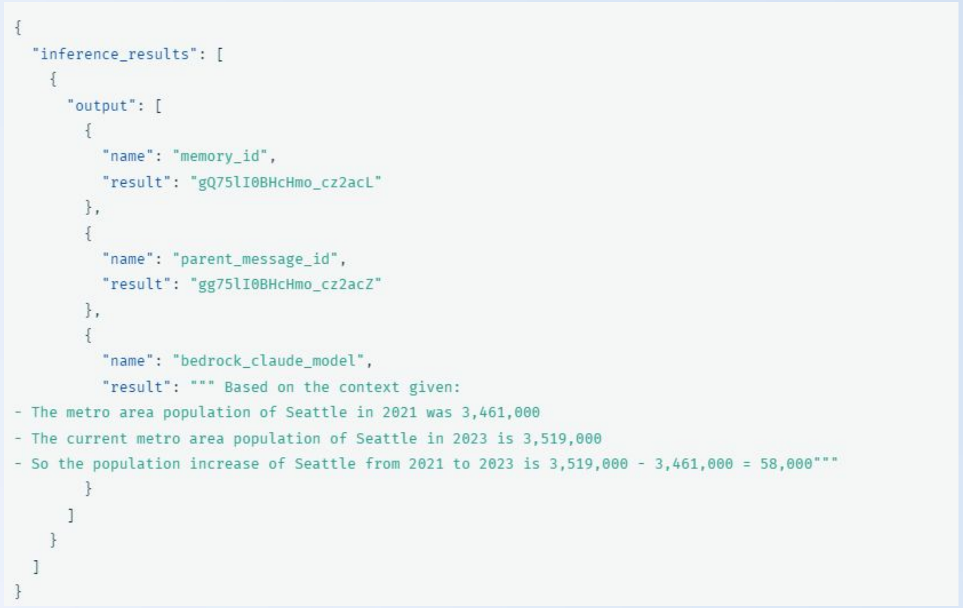
- **parameters**:
  - **question**: `"what's the population increase of Seattle from 2021 to 2023?"`
    - Contains the query or question that the agent is expected to answer

Purpose:

- **parameters**: Provides the input question to the agent for processing
- **question**: The specific query the agent needs to address, in this case, about the population increase in Seattle between 2021 and 2023

Run the agent

- Running a predict request will allow us to test the model
- You will see inference results return and it will be a series of numbers which means the model is working properly



```
{
  "inference_results": [
    {
      "output": [
        {
          "name": "memory_id",
          "result": "gQ75lI0BHcHmo_cz2acL"
        },
        {
          "name": "parent_message_id",
          "result": "gg75lI0BHcHmo_cz2acZ"
        },
        {
          "name": "bedrock_claude_model",
          "result": """ Based on the context given:
- The metro area population of Seattle in 2021 was 3,461,000
- The current metro area population of Seattle in 2023 is 3,519,000
- So the population increase of Seattle from 2021 to 2023 is 3,519,000 - 3,461,000 = 58,000"""
        }
      ]
    }
  ]
}
```

Lab End