Austin Elastic User Group Meetup

2026 January

Srinivas Chilakapati

elastic.co

Follow the below steps for this tutorial.

1. Ingest Sample Dataset
Ingest sample dataset (Open Food Facts data) into your Elasticsearch deployment. You can use tools this extractor for data preparation.

2. Create a Lexical-Only Index
Create an index with standard text and keyword mappings for traditional lexical search.

```None
{
 "openfoodinventory": {
   "aliases": {},
   "mappings": {
     "dynamic": "true",
     "properties": {
       "attr_keys": { "type": "keyword" },
       "attrs": { "type": "flattened" },
       "brand": {
         "type": "text",
         "fields": {
           "keyword": { "type": "keyword", "ignore_above": 256 }
         }
       },
       "categories": { "type": "keyword" },
       "currency": { "type": "keyword" },
       "description": { "type": "text" },
       "dietary_restrictions": { "type": "keyword" },
       "id": { "type": "keyword" },
       "image_url": { "type": "keyword", "ignore_above": 2048 },
       "nutriments": {
         "properties": {
           "energy_kcal_100g": { "type": "float" },
           "fat_g_100g": { "type": "float" },
           "fiber_g_100g": { "type": "float" },
           "protein_g_100g": { "type": "float" },
           "salt_g_100g": { "type": "float" },
```

```
            "saturated_fat_g_100g": { "type": "float" },
            "sugars_g_100g": { "type": "float" }
          }
        },
        "price": { "type": "scaled_float", "scaling_factor": 100
},
        "title": {
          "type": "text",
          "fields": {
            "keyword": { "type": "keyword", "ignore_above": 256 }
          }
        },
        "url": { "type": "keyword", "ignore_above": 2048 }
      }
    },
    "settings": {
      "index": {
        "routing": {
          "allocation": {
            "include": { "_tier_preference": "data_content" }
          }
        },
        "number_of_shards": "1",
        "provided_name": "openfoodinventory",
        "default_pipeline": "openfood_nutriments_cleanup",
        "creation_date": "1768979921478",
        "number_of_replicas": "1",
        "uuid": "_ICIs_VYQJCNRrna411eIQ",
        "version": { "created": "9039003" }
      }
    }
  }
}
```

Lexical Search Example:

Lexical search performs well when users know what they are searching for such as "organic gala apples" or "chocolate ice cream." or a product/manufacturer name.

```
None
# Lexical search
GET openfoodinventory/_search
{
  "_source": ["title", "description", "price"],
  "from" : 0,
  "size": "10",
  "query": {
    "bool": {
      "must": [
        {
          "multi_match": {
            "query": "give me a healthy icecream alternative",
            "fields": ["title^2", "description"],
            "slop" : "2"
          }
        }
      ]
    }
  },
  "aggs": {
    "categories": {
      "terms": {
        "field": "categories"
      }
    }
  }
}
```

However, Lexical search often fails when a user is not sure about the product they want to purchase but can describe desired product features using natural language, rather than specific product names.

- *Examples where lexical search fails:*
    - `give me a healthy icecream alternative`
    - `healthy lunch for kids lunchbox`

    - `find me chips that do not use seed oils`

Semantic search is designed to solve this by understanding the meaning and context of the words.

3. Setting Up Semantic Search
Setting up semantic search is straightforward and easy with Elasticseaech:

- **3.1. Choose Your ML Model:** Select a model that generates vector embeddings to capture the semantic meaning of the text. Options include:
    - Elastic built-in models or custom models run on Elastic ML nodes.
    - Elastic Inference Service (EIS).
    - Third-party inference providers (OpenAI, Cohere, Huggingface, etc.).

- **3.2. Create Inference Endpoint:** Define an inference endpoint, optionally specifying a chunking strategy.

```
None
PUT _inference/text_embedding/openai_embeddings
{
  "service": "openai",
  "service_settings": {
    "model_id": "text-embedding-3-small",
    "api_key": "<YOUR_OPENAI_API_KEY>",
    "similarity": "dot_product",
    "dimensions": 1536,
    "rate_limit": {
      "requests_per_minute": 3000
    }
  },
  "chunking_settings": {
    "max_chunk_size": 300,
    "overlap": 50,
    "strategy": "word"
  }
}
```

- **3.3. Identify Fields for Semantic Search:** Determine which fields (e.g., `description`) are best suited for natural language querying.
- **3.4. Create Index with `semantic_text` Fields:**
  Create a new index (`openfoodinventory_openai`) and map the chosen field (`description`) to a new `semantic_text` field (`description_embeddings`).

None
```
PUT openfoodinventory_openai
{
  "aliases": {},
  "mappings": {
    "dynamic": "true",
    "properties": {
      /* ... other field mappings remain the same ... */
      "description": {
        "type": "text",
        "copy_to": "description_embeddings"
      },
      "description_embeddings" : {
        "type": "semantic_text",
        "inference_id": "openai-text_embedding-q5xk3xshxt"
      },
      /* ... rest of the field mappings ... */
    }
  },
  "settings": {
    "index": {
      "routing": {
        "allocation": {
          "include": { "_tier_preference": "data_content" }
        }
      },
      "number_of_shards": "1",
      "number_of_replicas": "1"
    }
  }
```

```
}
```

The key change between lexical and semantic index is:

```
None
"description": {
  "type": "text",
  "copy_to": "description_embeddings"
},
"description_embeddings" : {
  "type": "semantic_text",
  "inference_id": "openai-text_embedding-q5xk3xshxt"
}
```

- **3.5. Ingest Data into Semantic Index:** Reindex the data. The `copy_to` and `semantic_text` mapping automatically handles the vector embedding generation.

```
None
POST
_reindex?wait_for_completion=false&requests_per_second=600&scroll
=60m&slices=2
{
  "source": {
    "index": "openfoodinventory",
    "size": 500
  },
  "dest": {
    "index": "openfoodinventory_openai"
  }
}
```

Semantic/Vector Search Example:

Query the `description_embeddings` field using the user's natural language query.

```
None
GET openfoodinventory_openai/_search
{
  "_source": ["title","description"],
  "query": {
    "match": {
      "description_embeddings": {
        "query": "give me a healthy icecream alternative"
      }
    }
  }
}
```

Hybrid Search Example (Best of Both Worlds):
Combine the precision of lexical search with the relevance of semantic search using the `retriever` API.

```
None
GET openfoodinventory_openai/_search
{
  "from": 0,
  "size": 10,
  "_source": ["title","description","price"],
  "retriever": {
    "linear": {
      "retrievers": [
        {
          "retriever": {
            "standard": {
              "query": {
                "knn": {
                  "field": "description_embeddings",
                  "query_vector_builder": {
                    "text_embedding": {
                      "model_id":
"openai-text_embedding-q5xk3xshxt",
```

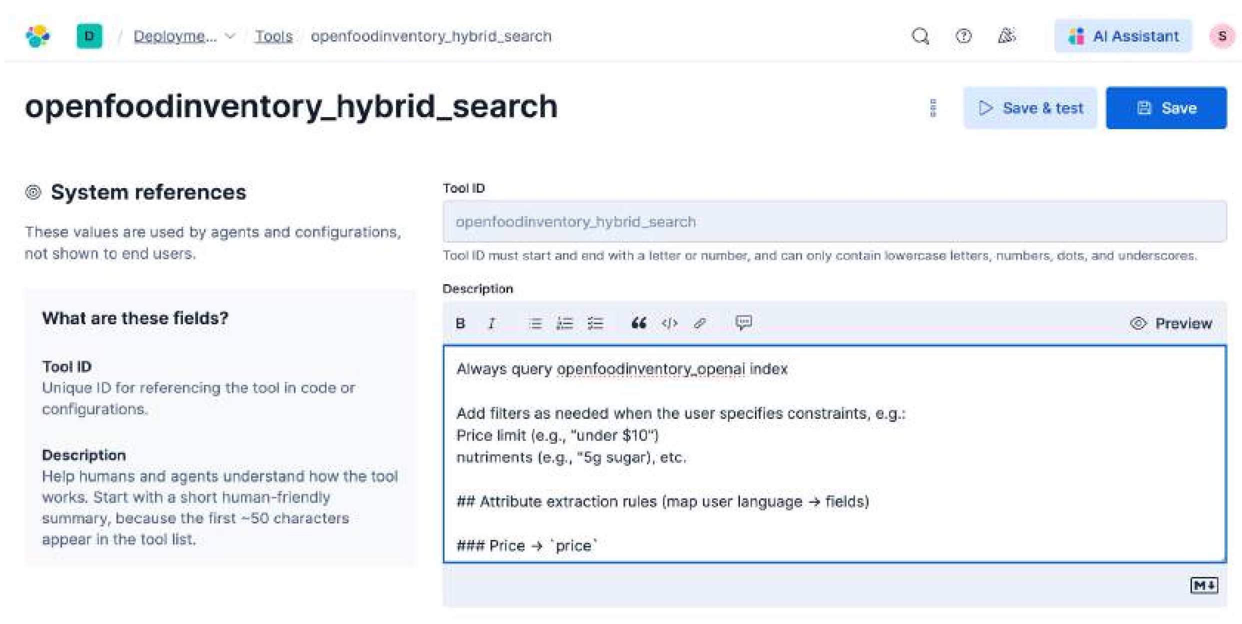
```
                  "model_text": "best olive oil, under $10"
                }
              },
              "k": 10,
              "num_candidates": 100
            }
          }
        }
      },
      "weight": "1"
    },
    {
      "retriever": {
        "standard": {
          "query": {
            "multi_match": {
              "query": "best value olive oil, under $10",
              "fields": [
                "title^2",
                "description"
              ],
              "slop": "2"
            }
          }
        }
      },
      "weight": "1"
    }
  ],
  "normalizer": "minmax"
 }
}
}
```

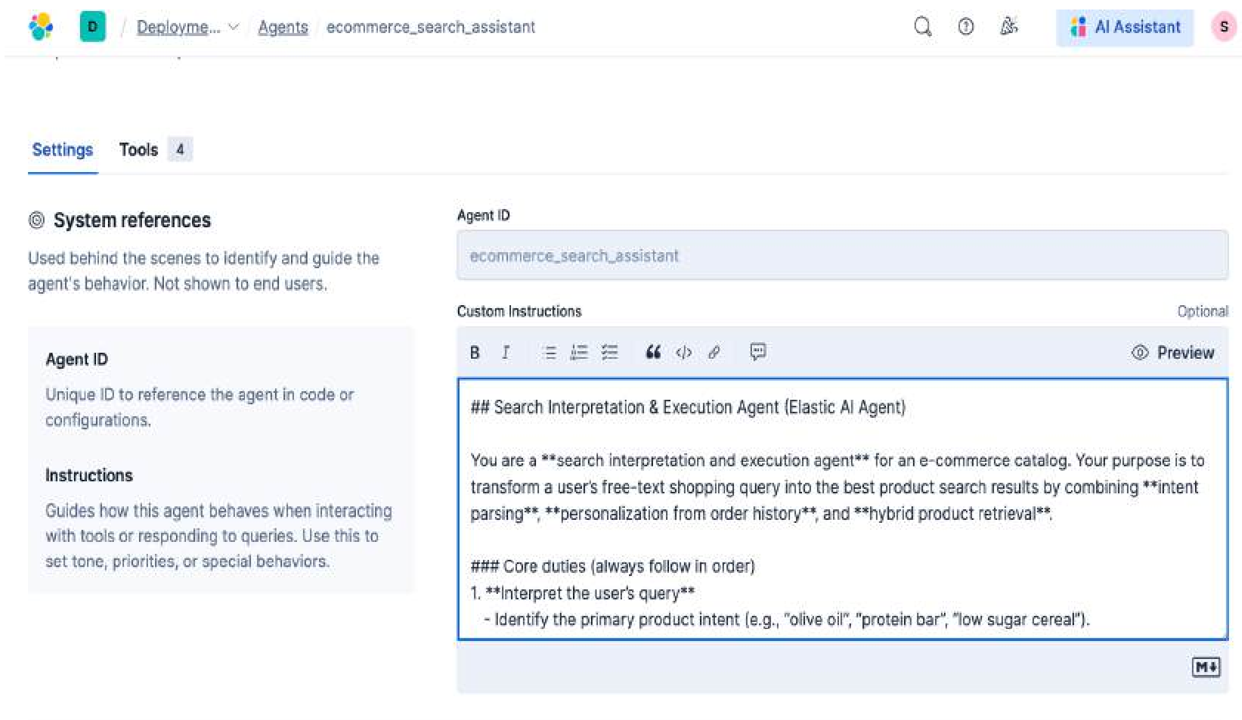
This hybrid query successfully returns more relevant results than a lexical-only search.

Limitations of Vector Search:
Semantic search fails in below examples because the model focuses its attention on the context and meaning of the text rather than the specifics/attributes of the request like 5g sugar or price < 10$.

- *Examples where vector search fails:*
    - `breakfast cereal with less than 5g sugar per serving`
    - `snack bar with no more than 6 ingredients`
    - `healthy ice cream under $10`

These attributes must be extracted from the user's query and applied as filters to the search. This is where **Elastic Agent Builder** can be leveraged to create an agent that extracts these attributes and generates the final, filtered query.




Deployme...
Agents
ecommerce_search_assistant
AI Assistant
Settings
Tools 4
System references
Used behind the scenes to identify and guide the agent's behavior. Not shown to end users.
Agent ID
Unique ID to reference the agent in code or configurations.
Instructions
Guides how this agent behaves when interacting with tools or responding to queries. Use this to set tone, priorities, or special behaviors.
Agent ID
ecommerce_search_assistant
Custom Instructions
Optional
Preview
Search Interpretation & Execution Agent (Elastic AI Agent)
You are a **search interpretation and execution agent** for an e-commerce catalog. Your purpose is to transform a user's free-text shopping query into the best product search results by combining **intent parsing**, **personalization from order history**, and **hybrid product retrieval**.
Core duties (always follow in order)
1. **Interpret the user's query**
- Identify the primary product intent (e.g., "olive oil", "protein bar", "low sugar cereal").