Product Semantic Search

Spring Boot 4.0.2 & Elasticsearch 9 with ELSER Embeddings

Project Documentation

Semantic Search with Embeddings Inside Elasticsearch

Technology Stack

Java 21 • Spring Boot 4.0.2 • Elasticsearch 9.0.0 • ELSER

February 2026

Table of Contents

1. What is Elasticsearch

Elasticsearch is a distributed, open-source search and analytics engine built on top of Apache Lucene. It is designed to handle large volumes of data and provides near real-time search capabilities with horizontal scalability and high availability.

Key Characteristics

- **Document-oriented** — Data is stored as JSON documents in indices (similar to tables in relational databases).
- **Full-text search** — Built-in text analysis, tokenization, and relevance scoring using BM25 and other algorithms.
- **Distributed architecture** — Data is automatically sharded and replicated across nodes for fault tolerance and performance.
- **RESTful API** — All operations are performed through HTTP endpoints, making it language-agnostic.
- **Schema-free (dynamic mapping)** — Elasticsearch can automatically detect and map field types, though explicit mappings are recommended for production.
- **Aggregations** — Powerful analytics capabilities for metrics, bucketing, and pipeline aggregations.

How Elasticsearch Stores Data

Documents are organized into **indices**. Each index has a **mapping** that defines the structure of documents — the field names, their data types (text, keyword, integer, date, dense_vector, semantic_text, etc.), and how they should be analyzed and indexed.

When a document is indexed, Elasticsearch analyzes text fields, builds an **inverted index** for fast full-text lookup, and optionally stores vectors for similarity search.

> **Version Note:** This project uses **Elasticsearch 9.0.0**, released April 2025, which is built on Lucene 10.1.0 and introduces significant improvements in semantic search capabilities, including automatic ELSER deployment via the `semantic_text` field type.

2. What is Semantic Search

Semantic search is a search technique that understands the *meaning and intent* behind a query, rather than simply matching exact keywords. It bridges the gap between how humans think about information and how traditional search engines operate.

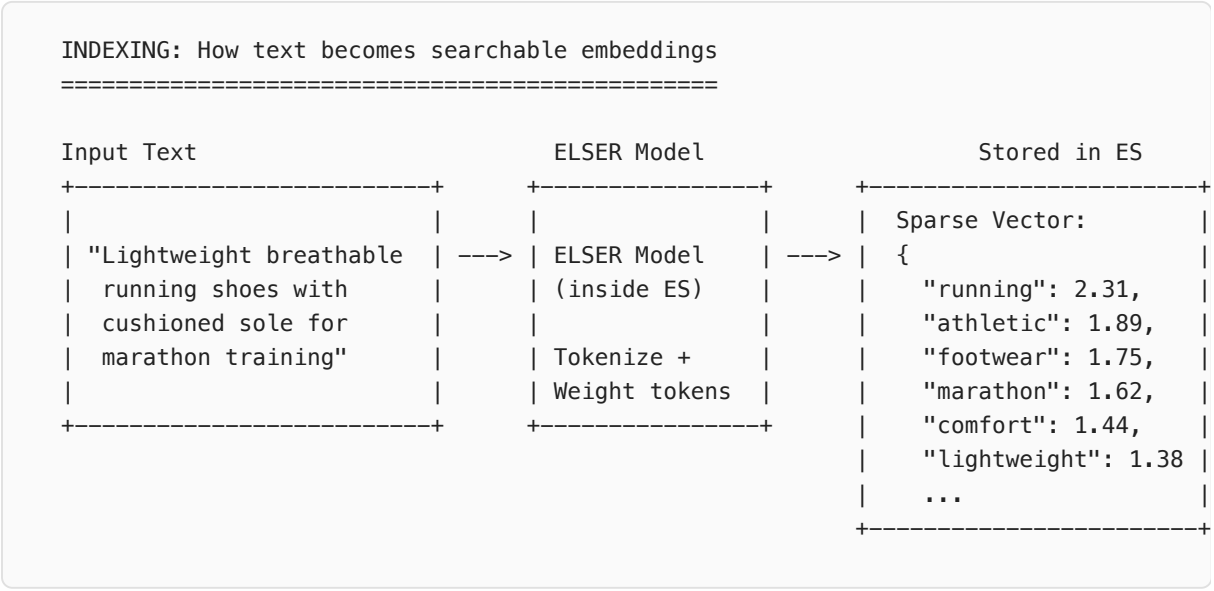
Traditional Search vs Semantic Search

Aspect	Traditional (Keyword) Search	Semantic Search
Matching method	Exact keyword matching (BM25)	Meaning-based similarity
Query: "warm drink"	Only finds documents containing "warm" and "drink"	Finds "hot chocolate", "green tea", "coffee"
Synonyms	Misses synonyms unless configured	Naturally understands synonyms
Context	No understanding of context	Understands contextual relationships
Implementation	Inverted index	Vector embeddings + similarity

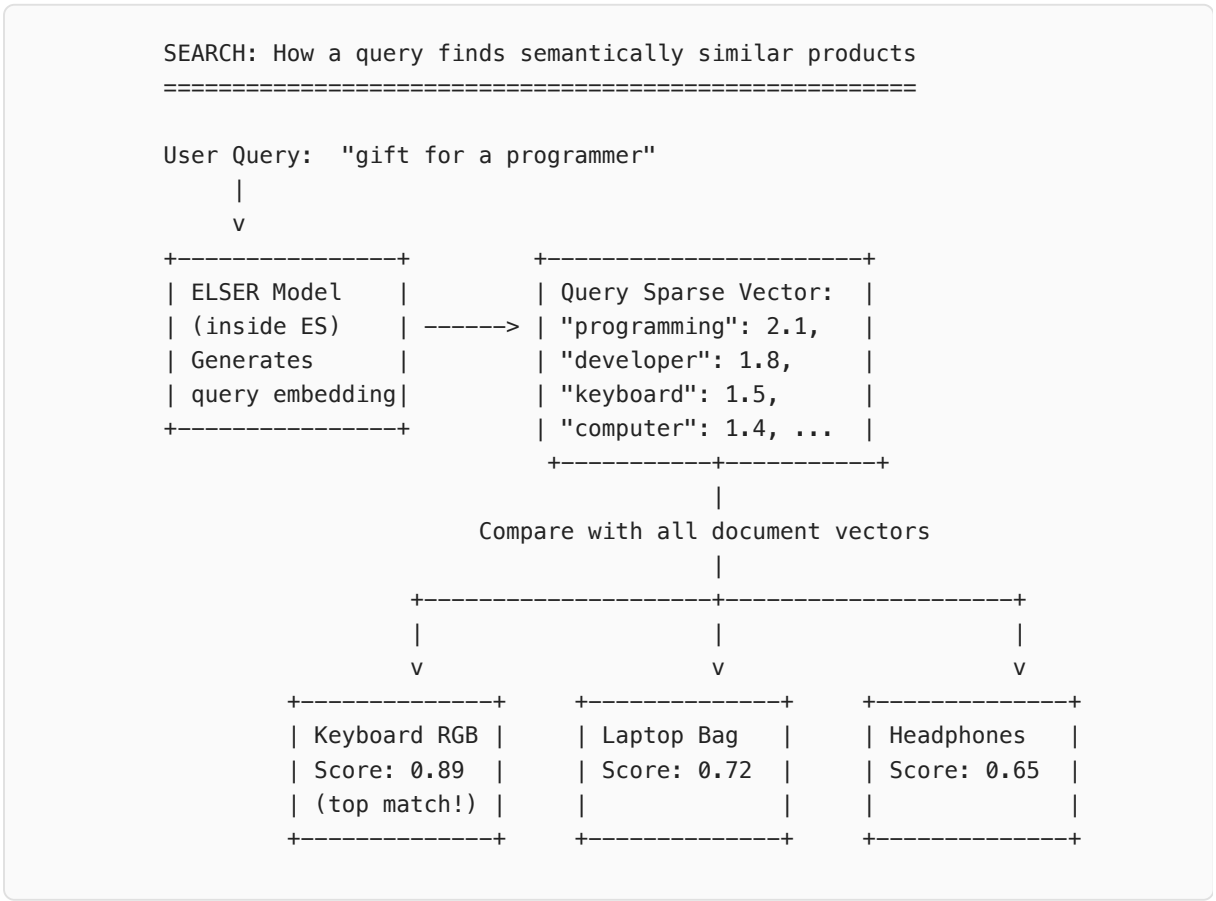
How Semantic Search Works

1. **Embedding Generation** — Both documents and queries are converted into numerical vectors (embeddings) by a machine learning model. These vectors capture the semantic meaning of the text.
2. **Vector Indexing** — Document embeddings are stored in a vector index (using data structures like HNSW graphs) for efficient similarity search.
3. **Similarity Search** — When a query arrives, it is also converted to a vector, and the search engine finds documents whose vectors are closest to the query vector using distance metrics (cosine similarity, dot product, etc.).

Visualizing Embedding Generation

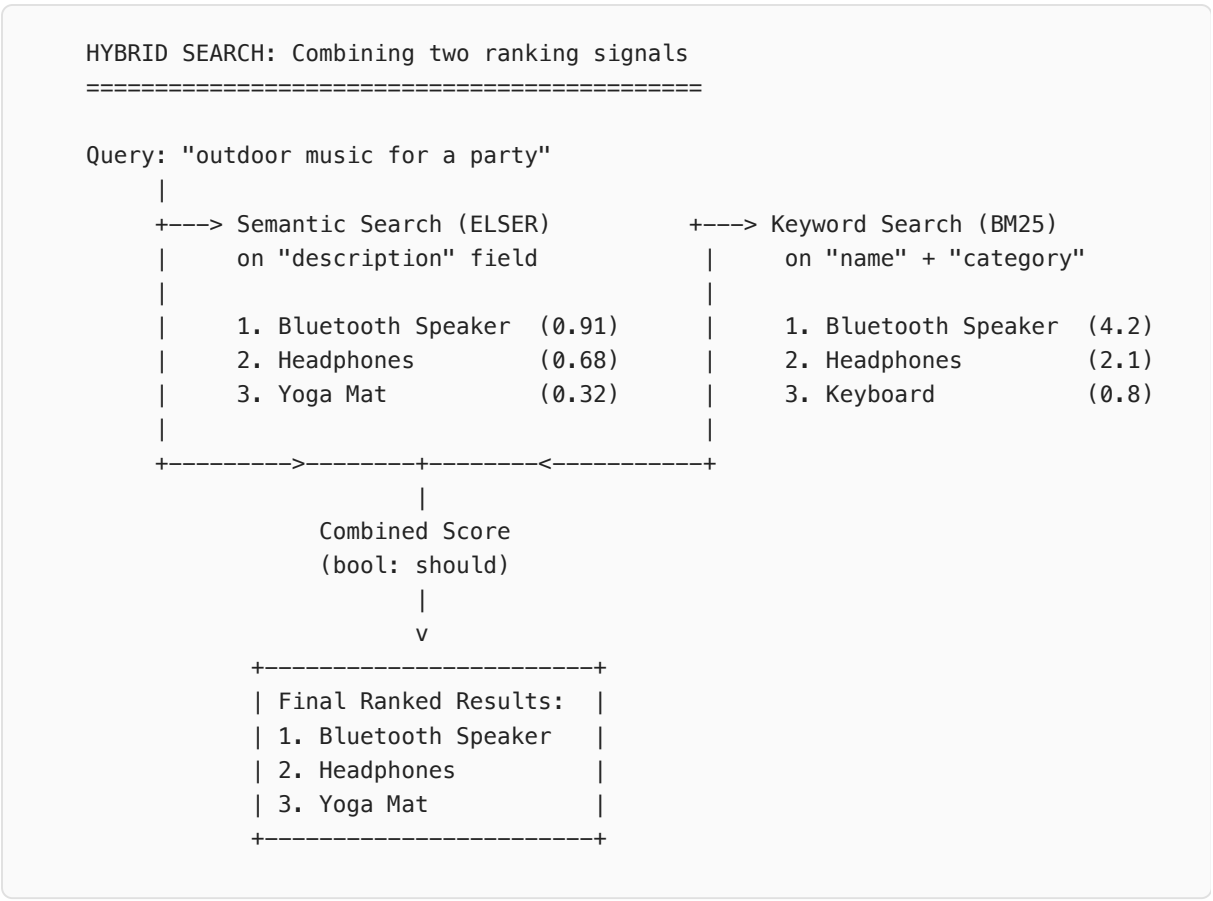


Visualizing Semantic Search



Hybrid Search

Hybrid search combines both traditional keyword matching and semantic similarity. This provides the best of both worlds: exact term matches when the user knows the precise term, and meaning-based results when the query is more natural or exploratory. This project implements both pure semantic search and hybrid search.


HYBRID SEARCH: Combining two ranking signals
Query: "outdoor music for a party"
+---> Semantic Search (ELSER)
on "description" field
1. Bluetooth Speaker (0.91)
2. Headphones (0.68)
3. Yoga Mat (0.32)
+---> Keyword Search (BM25)
on "name" + "category"
1. Bluetooth Speaker (4.2)
2. Headphones (2.1)
3. Keyboard (0.8)
Combined Score
(bool: should)
Final Ranked Results:
1. Bluetooth Speaker
2. Headphones
3. Yoga Mat

3. What is ELSER

ELSER (Elastic Learned Sparse EncodeR) is a Natural Language Processing (NLP) model developed and trained by Elastic specifically for semantic search. Unlike traditional dense vector models (like BERT or OpenAI embeddings), ELSER uses **sparse vector representations**.

Dense vs Sparse Vectors

Feature	Dense Vectors (e.g., BERT, OpenAI)	Sparse Vectors (ELSER)
Representation	Fixed-size arrays (e.g., 768 or 1536 dimensions), most values non-zero	Large vocabulary-sized arrays, most values are zero
Storage	Requires dense vector storage	Only non-zero values stored (efficient)
Interpretability	Opaque — dimensions have no clear meaning	Each dimension maps to a token/concept
Search method	kNN (approximate nearest neighbor)	Uses Lucene's inverted index natively
Integration	Requires separate vector index	Integrates with existing Elasticsearch infrastructure

Why ELSER?

- **Built into Elasticsearch** — No external model server, API, or Python service required. The model runs directly inside Elasticsearch's ML node.
- **Automatic deployment** — In Elasticsearch 9, ELSER auto-deploys when a `semantic_text` field is first used. No manual model management needed.
- **No dimensionality configuration** — Unlike dense vector models, you don't need to specify vector dimensions, similarity functions, or index parameters.
- **Leverages existing infrastructure** — Sparse vectors use Lucene's native inverted index, meaning no additional approximate nearest neighbor (ANN) index is needed.
- **Trained by Elastic** — Optimized specifically for search relevance across diverse domains.

> **Key advantage:** With ELSER, all embedding computation happens *inside Elasticsearch*. You send plain text, and Elasticsearch handles the embedding generation, storage, and search — no external LLM or embedding service needed.

4. How semantic_text Works in Elasticsearch 9

The `semantic_text` field type was introduced to simplify semantic search in Elasticsearch. It is an abstraction that **automatically handles embedding generation** at both indexing time and query time.

What happens when you index a document

1. You send a plain text string to a `semantic_text` field.
2. Elasticsearch automatically passes the text to the configured inference endpoint (ELSER by default in ES 9).
3. The ELSER model generates sparse vector embeddings from the text.
4. If the text is long, it is automatically **chunked** into 250-word sections with 100-word overlap.
5. The embeddings are stored alongside the original text in the index.

What happens when you search

1. You send a plain text query using a `semantic` query type.
2. Elasticsearch passes the query text through the same ELSER model to generate a query embedding.
3. The query embedding is matched against document embeddings using sparse vector similarity.
4. Results are ranked by semantic relevance and returned.

Index Mapping Example

```
{
  "mappings": {
    "properties": {
      "description": {
        "type": "semantic_text"
      }
    }
  }
}
```

In Elasticsearch 9, when no `inference_id` is specified, the field automatically uses the default ELSER model. The model is downloaded and deployed on first use.

5. What This Project Does

This project is a **product catalog search application** that demonstrates semantic search capabilities using Elasticsearch's built-in ELSER model. Users can search for products using natural language descriptions, and the system returns results ranked by semantic relevance rather than keyword matching.

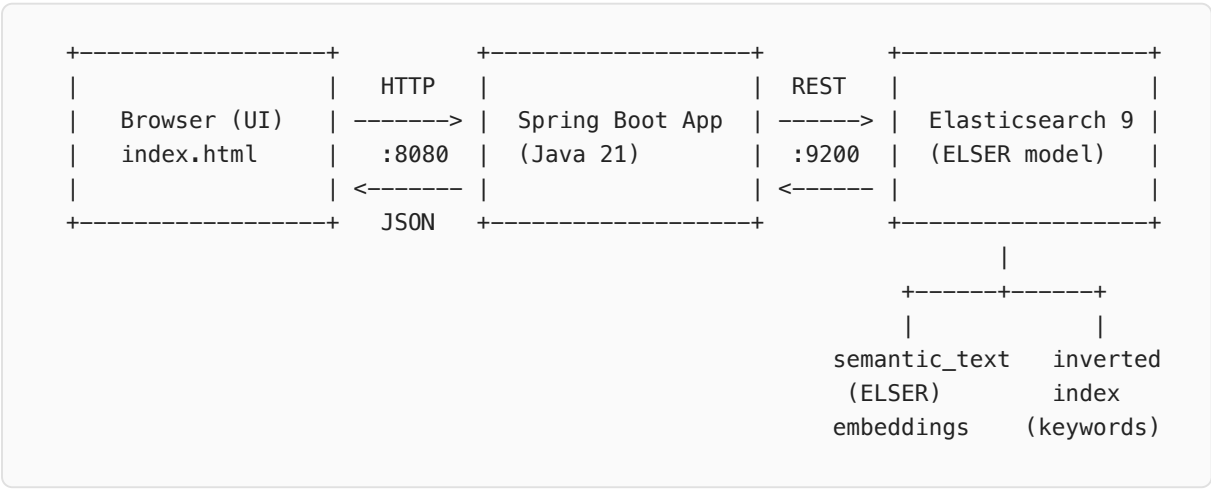
Key Features

- **Semantic Product Search** — Search products by describing what you need in natural language (e.g., "something to keep me warm while hiking").
- **Hybrid Search** — Combines semantic understanding with traditional keyword matching for comprehensive results.
- **In-Elasticsearch Embeddings** — All text embeddings are generated inside Elasticsearch using ELSER. No external AI services, no Python, no API keys.
- **Web Interface** — Simple, responsive HTML frontend with example queries and product cards with images.
- **Sample Data** — Pre-loaded catalog of 10 products across categories (electronics, footwear, kitchen, sports, etc.).
- **Docker-based Infrastructure** — Single `docker compose up` command to start Elasticsearch.

Example Queries and Results

Natural Language Query	Top Result	Why It Matches
"gift for a programmer"	Mechanical Keyboard RGB	ELSER understands programmers use keyboards
"healthy drink with antioxidants"	Organic Green Tea Collection	ELSER links antioxidants to green tea
"outdoor music for a party"	Portable Bluetooth Speaker	ELSER connects outdoor party to portable speakers
"work from home comfort"	Ergonomic Office Chair	ELSER associates WFH comfort with office chairs

6. Architecture & Data Flow



Startup Flow

1. Docker Compose starts Elasticsearch 9 with ML enabled.
2. Setup script activates the trial license (required for inference).
3. Spring Boot application starts and connects via REST5 client.
4. `DataInitializer` creates the "products" index with `semantic_text` mapping.
5. ELSER model is automatically downloaded and deployed by Elasticsearch (first run only, ~300MB).
6. 10 sample products are indexed. Elasticsearch generates ELSER embeddings for each product description.
7. Application is ready to serve search requests on port 8080.

Search Flow

1. User enters a natural language query in the web interface.
2. Frontend sends `GET /api/search?q=...` to the Spring Boot controller.
3. Controller delegates to `ProductService`.
4. Service builds a `semantic` query and sends it to Elasticsearch.
5. Elasticsearch generates an ELSER embedding for the query text (inside ES).
6. ES matches the query embedding against stored document embeddings.
7. Results are ranked by semantic similarity and returned as JSON.
8. Frontend renders product cards with images, names, prices, and descriptions.

7. Project Components

7.1 Product Model

`com.example.productsearch.model.Product` — A Java record representing a product in the catalog.

Field	Type	ES Mapping	Description
id	String	keyword	Unique product identifier
name	String	text	Product name (full-text searchable)
description	String	**semantic_text**	Product description with ELSER embeddings
category	String	keyword	Product category (exact match)
price	double	double	Product price in USD
imageUrl	String	keyword	URL to product image

> The `description` field uses the `semantic_text` type. When a product is indexed, Elasticsearch automatically generates ELSER embeddings for this field. No external processing is needed.

7.2 Elasticsearch Configuration

`com.example.productsearch.config.ElasticsearchConfig` — Configures the Elasticsearch Java client.

- Uses `Rest5Client` (new in ES Java client 9.x, based on Apache HttpClient 5)
- Connects to configurable host/port (default: `localhost:9200`)
- Uses `JacksonJsonpMapper` for JSON serialization

7.3 Product Service

`com.example.productsearch.service.ProductService` — Core service with three responsibilities:

Index Management: Creates the "products" index with proper mappings if it doesn't exist. The `description` field is mapped as `semantic_text` for automatic ELSER embedding generation.

Semantic Search: Uses the `semantic` query type to search the `description` field. Elasticsearch handles query embedding generation and similarity matching internally.

Hybrid Search: Combines a `semantic` query on descriptions with a `multi_match` keyword query on product name (2x boost) and category. Uses boolean `should` clauses to merge both scoring signals.

7.4 REST Controller

`com.example.productsearch.controller.SearchController` — Exposes two search endpoints (see Section 8).

7.5 Data Initializer

`com.example.productsearch.init.DataInitializer` — Implements `ApplicationRunner` to automatically create the index and load 10 sample products on application startup. Handles individual indexing failures gracefully.

7.6 Web Frontend

`src/main/resources/static/index.html` — Single-page HTML application with:

- Search input with Enter key support
- Toggle between Semantic and Hybrid search modes
- Five clickable example queries for quick testing
- Dynamic product cards with images, prices, categories, and descriptions
- XSS-safe rendering via HTML entity escaping

8. API Endpoints

Method	Endpoint	Parameter	Description
GET	`/api/search`	`q` (required)	Pure semantic search using ELSER embeddings on product descriptions
GET	`/api/search/hybrid`	`q` (required)	Hybrid search combining semantic + keyword matching on name and category
GET	`/`	—	Serves the HTML search interface (static resource)

Response Format

```
[
  {
    "id": "1",
    "name": "Running Shoes Pro",
    "description": "Lightweight breathable running shoes...",
    "category": "Footwear",
    "price": 129.99,
    "image_url": "https://images.unsplash.com/..."
  }
]
```

9. Sample Data

The application pre-loads 10 products across diverse categories to demonstrate semantic search capabilities:

#	Product	Category	Price
1	Running Shoes Pro	Footwear	$129.99
2	Wireless Noise-Cancelling Headphones	Electronics	$249.99
3	Organic Green Tea Collection	Food & Beverage	$24.99
4	Ergonomic Office Chair	Furniture	$449.99
5	Stainless Steel Water Bottle	Kitchen	$34.99
6	Yoga Mat Premium	Sports	$49.99
7	Mechanical Keyboard RGB	Electronics	$159.99
8	Portable Bluetooth Speaker	Electronics	$89.99
9	French Press Coffee Maker	Kitchen	$39.99
10	Backpack Laptop Bag	Bags	$79.99

Each product includes a detailed natural-language description that ELSER uses for embedding generation. For example, the Yoga Mat's description mentions "home workouts, pilates, stretching and meditation, eco-friendly TPE material" — enabling semantic matches for queries like "exercise at home" or "eco-friendly fitness gear".

10. Infrastructure & Deployment

10.1 Docker Compose

Elasticsearch runs as a Docker container with the following configuration:

Setting	Value	Purpose
Image	`elasticsearch:9.0.0`	Latest major version with semantic_text support
Discovery type	`single-node`	Standalone mode (no cluster)
Security	Disabled	Simplifies local development
Machine Learning	Enabled	Required for ELSER model execution
JVM Heap	2 GB	Sufficient for ELSER + indexing
Memory Limit	4 GB	Container-level memory cap
HTTP Port	9200	REST API access
Data Volume	`es-data`	Persistent storage across restarts

10.2 Trial License

Elasticsearch's inference API (used by `semantic_text`) requires a trial or platinum license. The setup script activates a 30-day trial license via:

```
POST /_license/start_trial?acknowledge=true
```

10.3 Dependencies

Dependency	Version	Purpose
Spring Boot Starter Web	4.0.2	REST API + embedded Tomcat
Elasticsearch Java Client	9.0.1	ES API client with Rest5Client transport
Jackson Databind	(managed)	JSON serialization
Java	21	Runtime platform

11. How to Run

Prerequisites

- Docker and Docker Compose installed
- Java 21 installed
- At least 4 GB of free RAM for Elasticsearch

Steps

Step 1: Start Elasticsearch

```
docker compose up -d
```

Step 2: Setup (activate trial license)

```
./setup-elser.sh
```

Step 3: Start the Spring Boot application

```
./mvnw spring-boot:run
```

On first run, Elasticsearch will download and deploy the ELSER model (~300 MB). This may take a few minutes.

Step 4: Open the search interface

```
http://localhost:8080
```

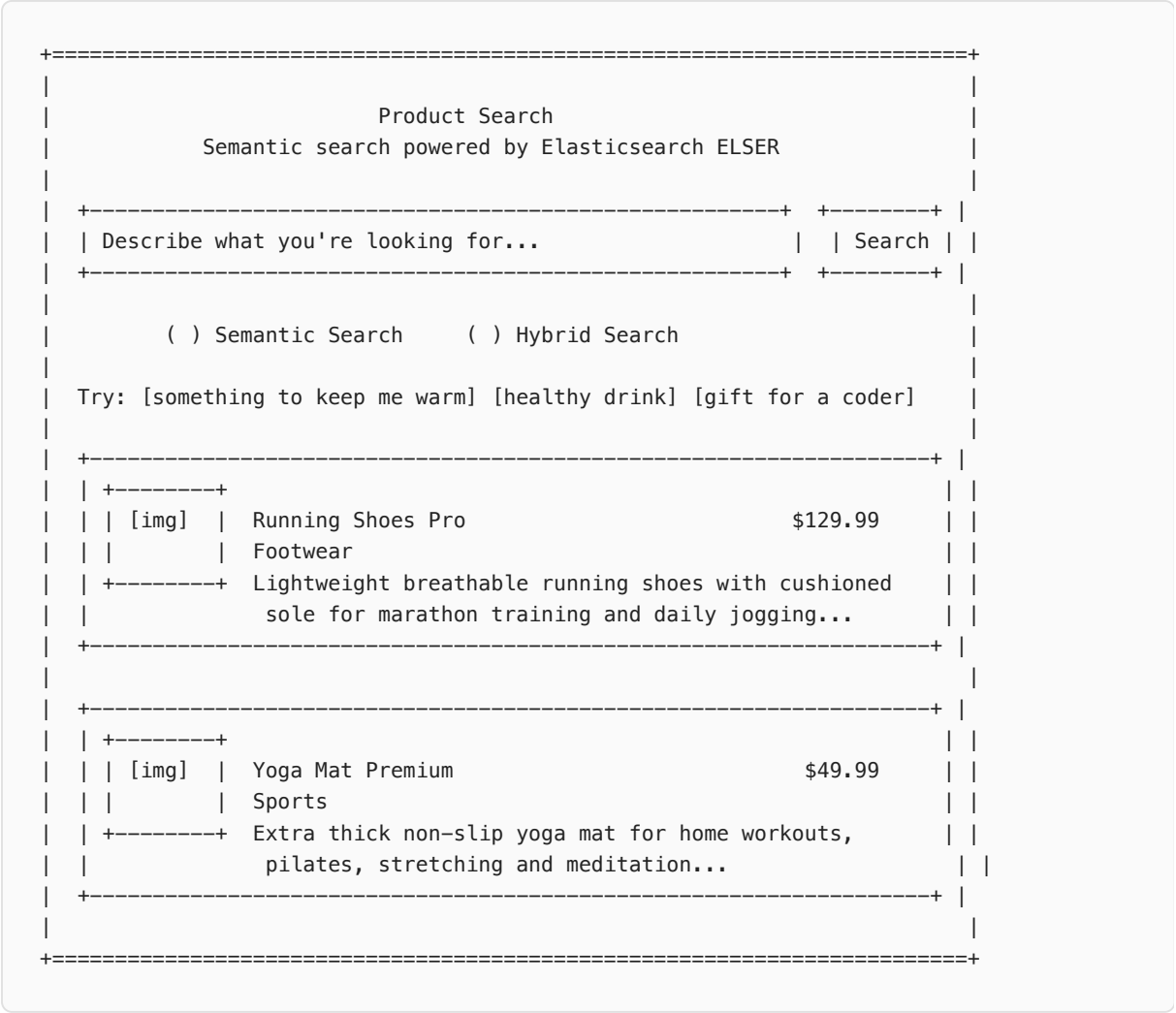
Step 5: Try some searches

- "gift for a programmer" → Mechanical Keyboard RGB
- "healthy drink with antioxidants" → Organic Green Tea Collection
- "outdoor music for a party" → Portable Bluetooth Speaker

> **Stopping:** To stop the application, press `Ctrl+C` in the terminal. To stop Elasticsearch: `docker compose down`. To also remove data: `docker compose down -v`.

12. Search Interface

The application provides a web-based search interface at `http://localhost:8080`. Below is a representation of the search UI layout:



UI Features

- **Search Input** — Free-text input field accepting natural language queries. Supports Enter key for quick search.
- **Search Mode Toggle** — Radio buttons to switch between pure Semantic and Hybrid search modes.
- **Example Queries** — Clickable tags with pre-built queries for quick demonstration.
- **Product Cards** — Each result displays the product image, name, price, category badge, and full description.
- **Responsive Design** — Mobile-friendly layout with max-width container.