# Moving the Needle on eCommerce Search and Listing

## Migrating from Database Search to Elasticsearch with a Plugin-Based, Queue-Driven Architecture

**Technical White Paper**

**Developed, implemented, and deployed by WizardLabz LLC**
Website: https://wizardlabz.com
Contact: info@wizardlabz.com

**Document version:** 1.0
**Author:** WizardLabz LLC

# Table of Contents

## Table of Contents

## 1. Executive Summary

This white paper documents how **WizardLabz LLC** redesigned and delivered a production-ready eCommerce search system by migrating from a traditional database-driven search approach to an Elasticsearch-powered search engine, deployed as a plugin within an existing extensible eCommerce platform.

The implementation focused on improving three areas simultaneously:

- **Performance**: Reduce search and listing response times from seconds to milliseconds
- **Relevance**: Improve ranking quality using Elasticsearch scoring and typo assistance
- **Operational safety**: Maintain full fallback to the existing database search system

The solution indexes approximately 50,000+ products, supports variant-aware search, and uses a queue-driven indexing pipeline so product updates trigger reindexing asynchronously without impacting storefront traffic.

A key validation milestone was the system's performance during Black Friday week, where search and filtering traffic increased significantly (5–7× typical load) and the solution delivered stable performance with zero search outages and no database saturation incidents.

This system was designed, built, deployed, and operationalized by WizardLabz LLC, and distributed to developers via a privately hosted NuGet package.

---

## 2. Background and Problem Statement

The platform originally served:

- Search pages
- Category listing pages
- Tag listing pages
- Brand listing pages

using the primary transactional database as the query engine.

As catalog size and traffic increased, the database-driven approach produced:

- Slow response times, especially with filters and joins
- Timeouts during load spikes
- Increasing database resource contention
- Limited relevance tuning beyond basic ordering rules
- Poor user experience when queries were mistyped or incomplete

In high-volume eCommerce, search is not a feature; it is a conversion pathway. When search is slow or inaccurate, users do not "wait," they leave.

WizardLabz was engaged to address search performance and quality while maintaining platform stability and preserving existing frontend templates and routing behavior.

## 3. Objectives and Non-Negotiable Constraints

WizardLabz defined a set of goals that shaped the architecture.

### Primary Objectives

1. Make search and listing pages fast at scale
2. Improve relevance quality using a proper search engine
3. Add variant-aware search to support real catalog discovery
4. Reduce query load on the transactional database
5. Provide an operational model suitable for peak seasonal traffic

### Constraints

1. No frontend rewrite
2. Same backend response structures (existing templates must continue to work)
3. Safe fallback to database search if Elasticsearch is unavailable or empty
4. Must fit into the platform's existing plugin architecture
5. Keep dependencies minimal and distribution developer-friendly

## 4. Solution Overview

WizardLabz implemented Elasticsearch as a parallel search system integrated through a plugin. The plugin intercepts listing routes and decides at runtime whether to serve content from Elasticsearch or let the core engine serve content from the original database search.

The system includes:

- A structured Elasticsearch index containing products, variants, images, and taxonomy
- A queue-driven reindexing pipeline triggered by product updates
- Variant-aware search and result shaping for storefront cards
- Relevance scoring and query assistance ("Did you mean")
- Operational tooling for admins to monitor indexing, errors, and status
- Private NuGet package distribution for controlled rollout

## 5. Architecture

At a high level, the solution operates as follows:

1. A product is created/updated in the platform
2. The platform emits an indexing request into a queue
3. Index workers consume messages and rebuild Elasticsearch documents
4. Storefront routes are served through the plugin:
    - If Elasticsearch has content: serve Elasticsearch results
    - If Elasticsearch is unavailable/empty: fall back to DB search

### Architecture Flow (Text Diagram)

Product Update

- Reindex Message Queue
- Index Worker(s)
- Elasticsearch Index (Products + Variants + Taxonomy)
- Plugin Route Interceptor
- Search / Category / Tag / Brand Listing Pages
- Fallback to DB Search if needed

---

## 6. Queue-Driven Indexing

WizardLabz introduced a queue mechanism to decouple indexing work from user-facing requests.

### Why a Queue

Indexing is not part of the customer's request-response path. If indexing were synchronous:

- Product updates would slow down
- Rebuild operations would affect storefront performance
- Peak traffic events would amplify instability

With a queue:

- Indexing happens asynchronously
- Workers can scale independently
- Failures are isolatable and recoverable

### Indexing Behavior

- Product updates publish a message indicating which product should be reindexed

- Workers consume and rebuild the product document including nested variants
- Throughput stabilizes at approximately 2,000 products per minute
- Full rebuild of 50,000 products completes in about 25–30 minutes depending on environment

---

## 7. Index Design and Document Modeling

WizardLabz designed a dedicated Elasticsearch document structure optimized for:

- Full-text relevance
- Fast filters and faceting
- Variant-level querying
- Stable routing and link building

### 7.1 Product Model (NEST Mapping)

```
using Nest;
using System;
using System.Collections.Generic;

namespace eCommerceEngine.ElasticSearchPlugin.Models
{
    [ElasticsearchType]
    public class Product
    {
        [Keyword(Name = "id")]
        public Guid id { get; set; }

        [Keyword(Name = "url")]
        public string url { get; set; }

        [Keyword(Name = "sku")]
        public List<string> Sku { get; set; }

        [Text(Name ="name")]
        public string Name { get; set; }

        [Keyword(Name = "name_keyword")]
        public string NameKeyword { get; set; }

        [Text(Name="description",Analyzer = "custom_html_analyzer")]
        public string Description { get; set; }

        [Number(Name = "star_rating")]
        public decimal StarRating { get; set; }

        [Number(Name = "review_count")]
        public int ReviewCount { get; set; }

        [Keyword(Name="category")]
        public List<string> Category { get; set; }
        [Keyword(Name = "category_url")]
        public List<string> CategoryURL { get; internal set; }

        [Keyword(Name="tag")]
        public List<string> TagText { get; set; }

        [Keyword(Name = "tag_url")]
        public List<string> TagSlug { get; internal set; }

        [Keyword(Name = "brand")]
        public string Brand { get; set; }

        [Keyword(Name = "brand_url")]
        public string BrandUrl { get; set; }

        [Nested]
        [PropertyName("variants")]
        public List<Variant> Variants { get; set; } = new List<Variant>();

        [Nested]
        [PropertyName("image_urls")]
        public List<IndexedImage> ImageUrls { get; set; }

        [Date(Name = "last_modified")]
        public DateTime LastModified { get; set; } = DateTime.UtcNow;
    }
}
```

## 7.2 Variant Model (Nested)

```
using Nest;
using System;

namespace eCommerceEngine.ElasticSearchPlugin.Models
{
    [ElasticsearchType]
    public class Variant
    {
        [Keyword(Name = "id")]
        public Guid Id { get; set; }

        [Keyword(Name = "variant_sku")]
        public string Sku { get; set; }

        [Keyword(Name = "barcode")]
        public string Barcode { get; set; }

        [Number(Name = "price")]
        public decimal Price { get; set; }

        [Text(Index = false, Store = true, Name = "img_file_name")]
        public string ImageFileName { get; set; }

        [Number(Name = "variant_display_order")]
        public int DisplayOrder { get; set; }
    }
}
```

### 7.3 Specifications

```
using Nest;

namespace eCommerceEngine.ElasticSearchPlugin.Models
{
    [ElasticsearchType]
    public class Specifcation
    {
        [Keyword(Name = "name")]
        public string Name { get; set; }

        [Keyword(Name = "value")]
        public string Value { get; set; }
    }
}
```

### Design Notes

- `Name` is a full-text field; `NameKeyword` is used for exact matches/boosting
- `Description` uses a `custom_html_analyzer` to remove HTML markup noise
- Categories, tags, and brand fields are keywords for fast aggregation and routing
- Variants are nested to support variant-aware search and listing cards
- Variant image file names are stored but not indexed (presentation-friendly, search-safe)

---

## 8. Variant-Aware Search and Listing Cards

The new system supports searching for variants (SKU/barcode/price) and producing listing results that reflect the relevant variants.

This enables the platform to display:

- Product cards with the correct variant context
- Variant display order
- Cleaner shopping experience for catalogs where variant attributes matter

This was not feasible in the original DB approach without expensive joins and complicated query logic.

---

## 9. Relevance Strategy and "Did You Mean"

### 9.1 Relevance Based on Elasticsearch Scoring

Search ranking is now based on Elasticsearch relevance scoring, enabling:

- Better matching for user intent
- Field weighting (e.g., name > description)
- Exact match boosting through keyword fields
- Reduced reliance on hard-coded ordering logic

### 9.2 “Did You Mean” for Mistyped Queries

WizardLabz introduced query assistance features that detect likely typos and suggest corrected queries.

This improves:

- Search success rate
- Conversion for long-tail or hard-to-spell items
- Overall user experience

## 10. Performance and Scale Results

The migration provided a major improvement in response time and stability.

### Key Numbers

- Index size: ~**50,000 products**
- Indexing speed: ~**2,000 products per minute**
- Typical search time: **seconds → milliseconds**

### Summary Table

| Metric | Before (DB Search) | After (Elasticsearch) |
|---|---|---|
| Average search time | ~5.2 seconds | ~180 ms |
| Filter-heavy listing time | ~3.8 seconds | ~95 ms |
| Catalog size indexed | N/A | ~50,000 products |
| DB query load from search | High | Reduced dramatically |
| Peak readiness | Risky | Stable and scalable |

## 11. Black Friday Week Stress Test

A key validation milestone was Black Friday week.

### Traffic Profile

- 5–7× typical load

- Heavy filtering and listing navigation
- Continuous campaigns across categories/brands/tags

### Results

- Zero search outages
- No database saturation driven by search or listing
- Stable response times
- No emergency scaling required
- No increase in search-related support incidents

Black Friday week validated that the system was not only fast in controlled tests, but also **reliable under real peak pressure**.

---

## 12. Operational Safety and Database Fallback

WizardLabz deliberately avoided turning Elasticsearch into a single point of failure.

If the index is:

- missing
- empty
- not built yet
- unreachable
- failing queries

…then the plugin responds as:

"I don't have content to serve."

This leaves the main engine to serve the exact same listing/search pages using the original DB-based logic.

### Why This Matters

- Safe rollout
- Safe rebuilds
- Safe new stores with empty catalogs
- No downtime coupling to search infrastructure

---

## 13. Plugin-Based Routing and Extensibility

The eCommerce engine supports plugins that can catch slugs and serve content for routes like:

- `/search`
- `/category/{catname}`
- `/tag/{tagname}`
- `/brand/{brandname}`

WizardLabz implemented the Elasticsearch solution as a plugin that:

- intercepts these routes
- serves Elasticsearch content when available
- allows DB fallback when not available
- applies content results in the same format the storefront already expects

---

## 14. Backward Compatibility: Same Backend Signature

A major success factor was preserving the original response signatures.

WizardLabz ensured:

- The plugin returns the **same object structure** used by the legacy DB search
- The storefront templates continued rendering without changes
- The front end did not need to be rewritten to adopt Elasticsearch

This reduced both risk and adoption time significantly.

---

## 15. Admin Tooling and Monitoring

WizardLabz built admin controls and APIs that allow admin users to:

- Monitor the queue and backlog
- Trigger full or partial rebuilds
- Watch indexing progress
- View indexing and serving errors
- Retrieve index status and health

This makes search operational, not mysterious.

---

## 16. Deployment and Distribution (Private NuGet)

The Elasticsearch plugin is shipped as a **privately hosted NuGet package**.

Benefits:

- Controlled distribution and versioning
- Only approved developers/environments can consume the package
- Easy install (reference the package, configure settings)
- Consistent deployments across environments

### Dependencies

The only required additional dependency on top of the core engine is:

- The official Elasticsearch .NET client library (NEST)

---

## 17. Technology Stack

- **Backend:** .NET (plugin-based integration into eCommerce engine)
- **Search Engine:** Elasticsearch
- **Messaging:** Queue-driven reindexing pipeline
- **Distribution:** Private NuGet feed + internal package hosting
- **Fallback:** Relational DB search (existing engine capability)
- **Index Client:** NEST (.NET Elasticsearch client) – not the latest – still provides typed APIs for a more robust solution

---

## 18. Security Considerations

The implementation assumes standard production security practices:

- Elasticsearch endpoints restricted by network/security rules
- Credentials stored via environment configuration/secure configuration mechanisms
- Admin indexing actions restricted to admin users

---

## 19. Observability and Diagnostics

Operational visibility focuses on:

- Queue backlog and throughput
- Index rebuild status
- Indexing failures

- Serving failures (query-time errors)
- Index availability

This visibility was essential for confident operation during peak periods (including Black Friday week).

---

## 20. Roadmap: Vector Search and Similarity Retrieval

WizardLabz's planned next phase is vector-based similarity retrieval:

- Generate embeddings for products (name, description, specifications)
- Store vectors in Elasticsearch
- Use k-NN / vector similarity queries to retrieve items by similarity
- Enable "More like this" recommendations and semantic discovery

This can be implemented as a hybrid strategy:

- Keyword relevance + vector similarity scoring

---

## 21. Outcomes and Lessons Learned

### Outcomes

- Major response time improvement
- Reduced database load and improved platform stability
- Better relevance quality and discoverability
- Operational safety via DB fallback
- Zero frontend rewrite due to signature compatibility
- Production validation under Black Friday week peak load

### Lessons Learned

- Treat search as a platform capability, not a query
- Decouple indexing from request flow via a queue
- Preserve contracts to reduce migration risk
- Provide operational tooling from day one
- Validate under real seasonal traffic, not just benchmarks

---

## 22. About WizardLabz LLC

**WizardLabz LLC** builds and modernizes digital systems that businesses rely on.

We specialize in:

- High-scale eCommerce systems
- Search and listing modernization
- Plugin-based extensibility
- Performance and operational reliability
- Incremental modernization without breaking the business

This project was **designed, built, deployed, and validated by WizardLabz LLC**, including architecture, implementation, rollout strategy, and operational tooling.

Website: https://wizardlabz.com
Contact: info@wizardlabz.com