elastic

Search. Observe. Protect.

Engagement Report

kramp.com

TCCF B.V.

2024 March 12th

Deniz Coskun

elastic.co

Table of Contents

Executive Summary

Overview

Kramp.com is an agricultural e-commerce platform that caters primarily to businesses, operating across 36 countries in Europe. The company boasts over 1 billion in revenue generated through e-commerce, with a remarkable 80% of transactions initiated via their search functionality. They are reputed for their efficient next-day delivery service, which caters to customers who frequently fill their carts throughout the day, often performing price comparisons with competitors before placing orders at day's end.

A notable aspect of Kramp's customer behavior is the frequent use of item number searches, accounting for about 70% of queries, suggesting a customer base that is knowledgeable about specific product IDs and SKUs. Their platform utilizes an autocomplete layer to provide suggestions and quick product references, enhancing the user search experience.

While Kramp has developed landing pages to direct customer traffic, these are currently underutilized. For search result optimization, NDCG (Normalized Discounted Cumulative Gain) metrics are used, and 'golden sets' have been established based on historical clicks. However, these sets have not been updated recently.

The company has observed that one-third of search queries include terms that haven't been seen in the last two years, indicating a dynamic inventory or changing customer needs. Item number searches typically perform better in terms of relevancy than term searches. A notable challenge is the high rate of zero-result searches, which occur in every fifth search, signaling an immediate area for improvement. Spell suggestion functionality, which only triggers on 11% of searches, currently offers suboptimal recommendations.

A potential solution to the zero-result problem could involve implementing a successor-predecessor logic that may help redirect searches to relevant alternatives when exact matches are not found. Additionally, the company maintains its facets in a separate PostgreSQL database, apart from its main e-commerce system.

Business Value

tbd

Outcomes

tbd

Next Steps

tbd

Relevancy Issues

Below, we are going to categorize relevancy issues into three categories. Please note that these are the main categories. In each category, there may be additional classification (e.g. recall issue > zero result).

Precision

In search, a precision issue arises when the search system retrieves a high number of irrelevant results amongst the relevant ones for a given query. A low precision score means that users must sift through many irrelevant results to find the useful ones, leading to a frustrating search experience.

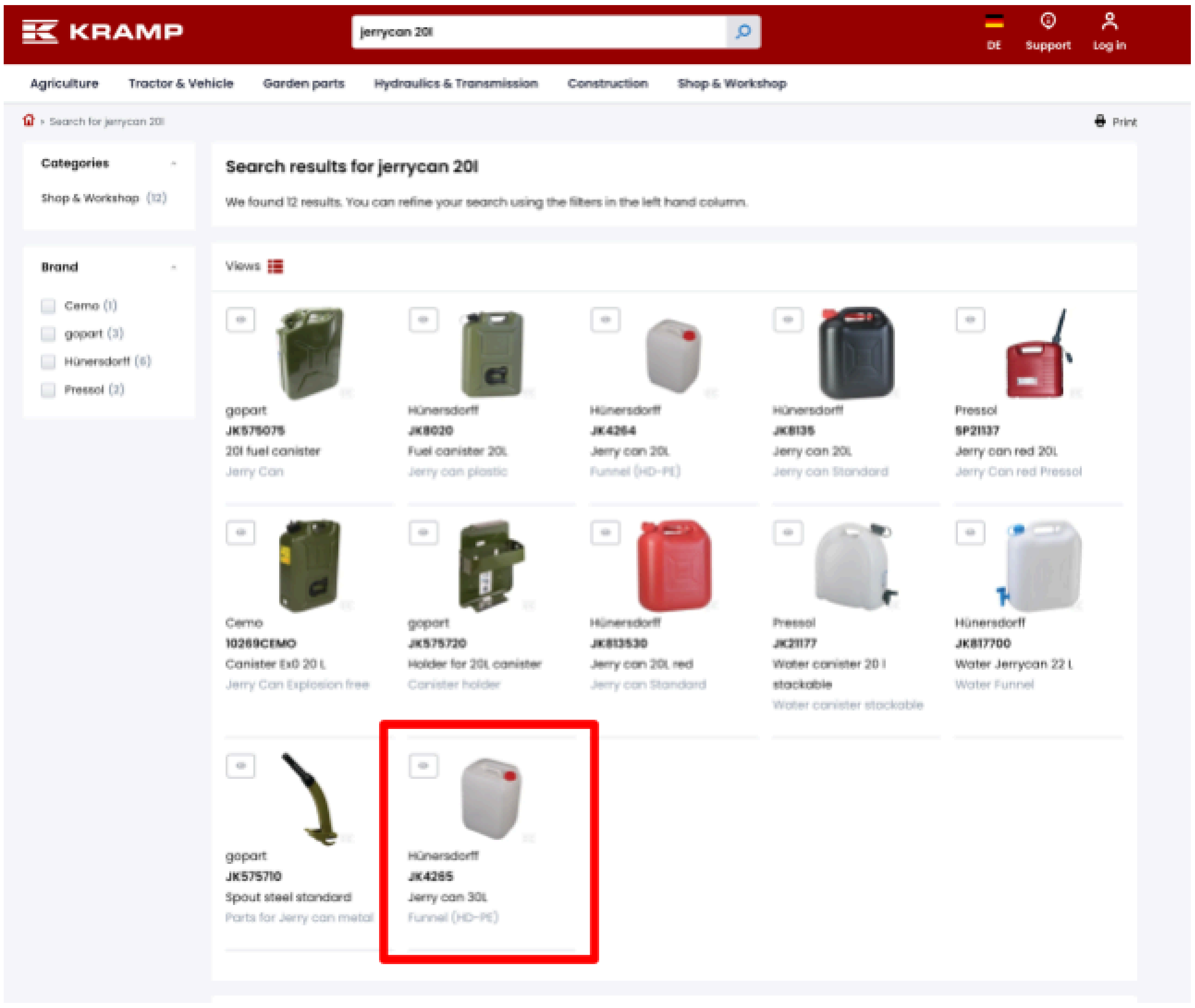
Examples

Query	Expectation	Shop	Explanation
Simmerring	Gives 4441 results, no filters because of OE-items	DE	
bearing 40×50	We don't sell this size, but show 130 other results which customer then has to look through to find nothing	DE	"type": "cross_fields" used in the multi-match query. This treats search fields with the same analyzer as one big search field, in which you have all the matches.
3/2 ventiel	First result shown is 6/2 ventiel	NL	Data quality issue. You have the keyword in the data.
jerrycan 20l	Shows result for jerrycan 30l	NL	You have the keyword in the search fields. Data quality issue.
3923793	Gives suggestion for '23793', which is not related product	NL	Spell suggestions on variations of an item number don't make sense
huishoudladder	We do sell products called 'huishoudtrap'	NL	

Possible Reasons

Poor precision can be a result of many things (usually in a combined way). Simply put, for some searches you deliver many irrelevant results because of your current query and index configuration. Matching documents is a technical operation. If you are matching irrelevant documents, you either mistakenly have the data (tokens) in these documents or your search is too relaxed.

Let’s take “jerrycan 20l” as an example. If I search for this term, I get one 30l version:



Search results for jerrycan 20l

We found 12 results. You can refine your search using the filters in the left hand column.

Categories
- Shop & Workshop (12)

Brand
- Cemo (1)
- gopart (3)
- Hünersdorff (6)
- Pressol (2)

gopart
JK575075
20l fuel canister
Jerry Can

Hünersdorff
JK8020
Fuel canister 20L
Jerry can plastic

Hünersdorff
JK4264
Jerry can 20L
Funnel (HD-PE)

Hünersdorff
JK8135
Jerry can 20L
Jerry can Standard

Pressol
SP21137
Jerry can red 20L
Jerry Can red Pressol

Cemo
10269CEMO
Canister ExO 20 L
Jerry Can Explosion free

gopart
JK575720
Holder for 20L canister
Canister holder

Hünersdorff
JK813530
Jerry can 20L red
Jerry can Standard

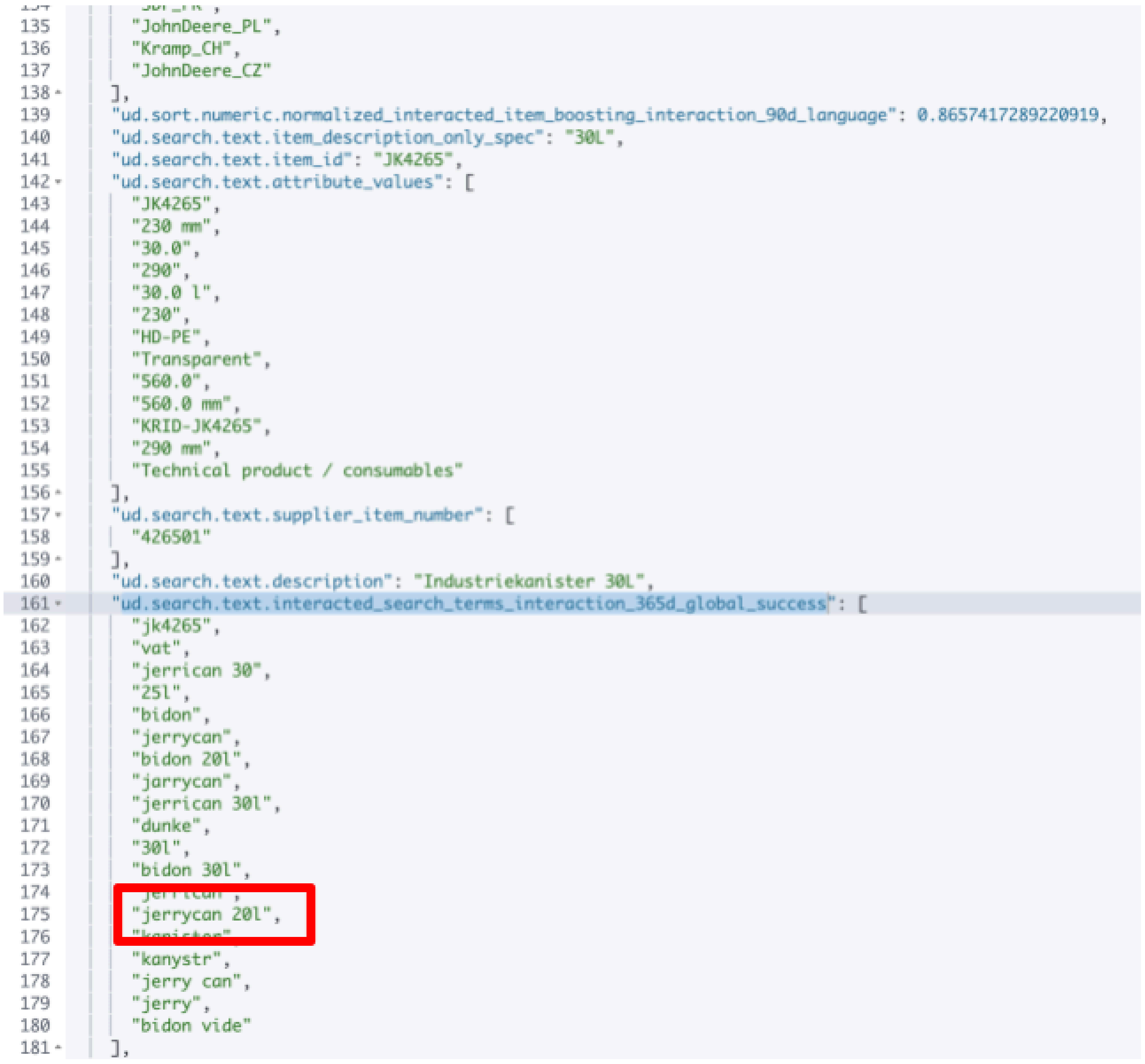
Pressol
JK21177
Water canister 20 l stackable
Water canister stackable

Hünersdorff
JK817700
Water Jerrycan 22 L
Water Funnel

gopart
JK575710
Spout steel standard
Parts for Jerry can metal

Hünersdorff
JK4265
Jerry can 30L
Funnel (HD-PE)

After a quick look at the explain API, I can already see the matching field. Looking at the field information, it is clear that you have this search term added to this article:



```
    "JDF_FR",
    "JohnDeere_PL",
    "Kramp_CH",
    "JohnDeere_CZ"
  ],
  "ud.sort.numeric.normalized_interacted_item_boosting_interaction_90d_language": 0.8657417289220919,
  "ud.search.text.item_description_only_spec": "30L",
  "ud.search.text.item_id": "JK4265",
  "ud.search.text.attribute_values": [
    "JK4265",
    "230 mm",
    "30.0",
    "290",
    "30.0 l",
    "230",
    "HD-PE",
    "Transparent",
    "560.0",
    "560.0 mm",
    "KRID-JK4265",
    "290 mm",
    "Technical product / consumables"
  ],
  "ud.search.text.supplier_item_number": [
    "426501"
  ],
  "ud.search.text.description": "Industriekanister 30L",
  "ud.search.text.interacted_search_terms_interaction_365d_global_success": [
    "jk4265",
    "vat",
    "jerrican 30",
    "25l",
    "bidon",
    "jerrycan",
    "bidon 20l",
    "jarrycan",
    "jerrican 30l",
    "dunke",
    "30l",
    "bidon 30l",
    "jerrican",
    "jerrycan 20l",
    "kanister",
    "kanystr",
    "jerry can",
    "jerry",
    "bidon vide"
  ],
```

It is the same with the “3/2 ventiel” search, where you have the “6/2 ventiel” product in the top results. Looking at the explanation, it is clear that you have the keyword “3/2 ventiel” keyword in a “6/2 ventiel” product.

It looks like the majority of your issues are caused by the poor data quality or suboptimal keyword enrichment process.

Let's take another example. This time, we look at "bearing 40×50". According to the customer, there shouldn't be any results for this query, but they are getting products with different sizes. Again, a quick look at the explanation, I can see that the parts that are matching:

- ud.search.text.sales_category_name.standard:40
- ud.search.text.attribute_values.standard:bearing
- ud.search.text.sales_category_name.standard:50
- ud.search.text.sales_category_name.standard:40
- ud.search.text.sales_category_name.standard:50

As you can see, this time you don't have the whole search term in one field, but it is distributed over multiple fields. The reason why you are still matching is that you are using **"type": "cross_fields"** in your multi match query. This treats fields with the same analyzer as though they were one big field and looks for each word in any field. This is why you are getting (irrelevant) results.

Additionally, when I analyze the search query with this analyzer using following request:

```
Unset
GET kramp-item-de-2023-07-06/_analyze
{
 "explain": true,
 "field": "ud.search.text.sales_category_name.standard",
 "text": "bearing 40x50"
}
```

I get these tokens → [bearing, 40, 50, 40, 50]. "40" and "50" are duplicated. Are there any reasons behind this decision or is this a not calculated side effect of your analyzer? This may cause some recall issues. You may want to remove duplicates.

Possible Solutions

(to-do) define solutions in detail

- Improve data quality
- Optimize query by adding minimum should match parameter
- Use filter rules
- Enrich documents with exclude keywords
- Rewrite query
- Optimize suggester call
- Optimize synonyms
- Optimize dictionary for compound terms

Recall

A recall issue in search is the type of a problem where a search system fails to retrieve all relevant documents or results for a given query. Recall is a metric used in information retrieval to measure the completeness of the results returned by a search engine. It is defined as the proportion of all relevant documents that are successfully retrieved in a search.

Examples

Here are some examples from your Excel sheet:

Query	Expectation	Shop	Explanation
simmering	Should show results/suggestion 'simmerring'	DE	You have matching documents for the simmering, this is why there are results and your search does not fall to the autocomplete query.
axialkugellager	Gives 5 results, 'axial kugellager' gives 185	DE	According to the customer, the decompounded version delivers the better results. In order to improve the recall of the 'axialkugellager', you can either use decompounding, use synonyms or rewrite the user's query.
axialkugellager	Should show results for	DE	Both queries share the same 'tokens'. You have two options here. You can use

	'axialrillenkugellager'		decompounding or synonyms
klemmschelle	5 results, synonym should be 'rohrschelle' with 1500 results	DE	This is also not a partial match. 'klemmschelle' and 'rohrschelle' are very close to each other but they are not synoynms. You can still use synonyms with the trade-off of reducing the precision. Other alternatives would be mimicking weighted synonyms by using/implementing a rule engine.
gummi metal puffer	Gives 5 results, 'gummi metall puffer' gives 836		In this example, you again have data quality problems. You have the misspelled term in your index. You also get a correct phrase suggestion, but since you have results, you are not falling back to the suggestions.
khun	1 result, should show results/suggestion 'kuhn' (6039)		Data quality issue. Technically, this search is correct. You have the misspelled word in your index and therefore not falling back to the suggestion.
aftekenblo	Should show results/suggestion 'aftekenblok'	NL	
oliekeering	Should show results/suggestion 'oliekeerring'	NL	
milieusticker	Should show results/suggestion 'milieu sticker'	NL	Decomounding
buizenklem	Should show results/suggestion 'buisklem'	NL	singular/plural
ijzerzaagje	Should show results/suggestion 'ijzerzaag'	NL	Diminutives
huishoudladder	We do sell products called 'huishoudtrap'	NL	
paint remover	We call these 'verf verwijderaar' in Dutch	NL	If you have some high volume foreign language searches, it is possible that some users adopted these terms. Depending on the overall volume, it is

			probably not worth implementing a language identifier. Solving such issues with synonyms is in most cases the fastest way.
eskimo	We do sell products called 'snowshovels'	NL	We don't sell the brand, but sell products that are alternative
iso 1813	Should give us v-belts, but we don't have this data	NL	

Possible Reasons

In order to solve recall issues, it is important to understand the possible causes. A recall issue can occur due to several reasons, such as:

1. **Poor data quality** is one of the main reasons for recall issues. Note that searching is the science of matching. User's query needs to match the tokens in your index. If you have faulty data (missing keywords, misspellings, etc.) in your index, this will prevent possible matches.
2. Suboptimal **text analysis** can also be a reason which may have a negative impact on the recall rate. Both the text in your documents and search queries should be analyzed in a way to have best conditions for matching. This is not an easy task. Text analysis changes from language to language, but it also varies for each use case. Starting with the language analyzer from Elasticsearch is suggested. Over time, adding your use case specific logic (stemming, synonyms, decompounding, etc.) to create your custom analyzers is the recommended approach.
3. **Missing translation of user's intent**. Users can search for a product by not using the official/technical term. It is important to still understand what users want and translate their language into your language so that you can match relevant products. This is also not an easy task and requires an analysis of the usage.
4. **Restricted search scope** can also be a reason for recall issues. If you have redirects enabled for certain search terms and/or falsely filtering search results, this can decrease your recall rate massively. This also happens when you are not searching all relevant fields.

Possible Solutions

(to-do) define solutions in detail

- Improve data quality
- Enrich data (e.g. with keywords)
- Add synonyms
- Use/extend decompounding
- Rewrite queries

Ranking

A ranking issue in search refers to the problem with how search results are sorted and presented to users. When a user enters a query into a search engine, the engine's algorithms determine the most relevant and useful results to return. Ideally, these results should be ranked in order of relevance, with the best matches at the top of the list.

In its core, the ranking issue is a mixture of recall and precision issues at the top results level.

Possible Solutions

(to-do) define solutions in detail

- Boosting (positive/negative)
    - (to-do) example
- Adjusting BM25 (similarity modules)
    - (to-do) example

Query Rules Engine

Assume, you have set up all your analytics pipeline. You receive usage signals from your search applications (e.g. search, click, add-to-cart, dwell time, etc.) and you analyze these signals to better understand the intent of your users. Now you need to translate this understanding into the search engine language. Below, I list some options, which you can use for this purpose.

The primary advantage of a query rules engine is its ability to provide a layer of control and customization over the search process that can address specific business needs, improve search relevancy, and enhance the overall user experience. It's especially valuable for e-commerce search engines, content management systems, and other applications where search relevance directly impacts user satisfaction and business outcomes.

Elasticsearch Query rules APIs

Elasticsearch announced query rules in Elasticsearch 8.10. Query rules allow you to configure per-query rules that are applied at query time to queries that match the specific rule.

Currently, there is one type of rule available in this API. You can pin results for certain queries. You can expect more rule types in future, but unfortunately there is no timeline about this.

Querqy Plugin

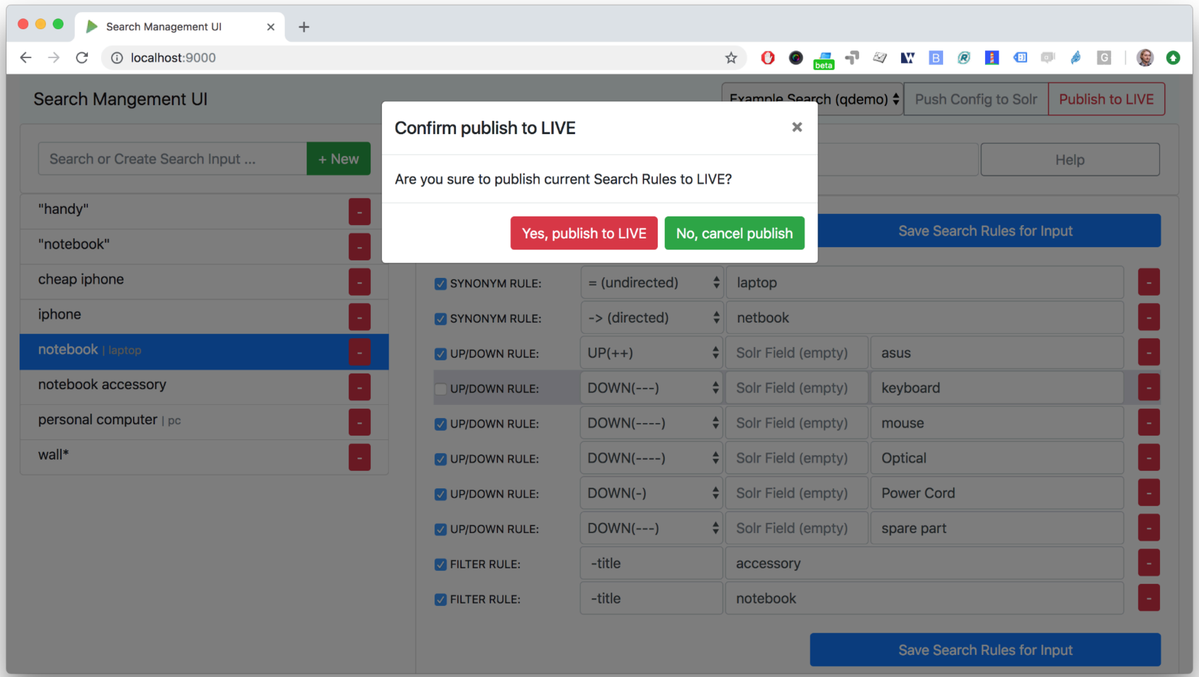
Querqy is an open-source search query preprocessing library. It provides tools to enhance and control search relevancy. There is a querqy plugin for Elasticsearch.

In our session, I have demonstrated this plugin and its capabilities. We also discussed its advantages and disadvantages. This plugin is not from Elastic. Therefore, it is not part of the support agreement and Elastic does not provide support for 3rd party plugins. In addition to that, there is also a version limitation. The latest Querqy version is compatible with the Elasticsearch 8.9.2. You can’t expect a Querqy release for each minor release. This means, you may wait until a compatible version is released. Alternatively, you can create your own version since this is an open source project.

Let’s look at the advantages of this tool:

1. **Query Rewriting:** Querqy can rewrite search queries before they are executed by the search engine. This allows for implementing custom logic such as synonym expansion, spell correction, and more sophisticated manipulation to better match user intent.
2. **Rules Engine:** Querqy allows you to define rules to match particular query patterns and apply transformations or add additional query clauses. This can be used to boost or filter products, control query-dependent ranking, or handle business-specific edge cases.
3. **Downranking and Upranking:** Based on the rules you define, Querqy can adjust the relevance scoring of search results. This means you can promote (uprank) or demote (downrank) certain items in the search results.
4. **Synonyms and Stopwords:** It supports handling synonyms more intelligently than simply replacing terms in the user's query. You can control synonym expansion differently based on whether the synonym is part of a phrase or a standalone term. Querqy also helps in handling stop words (common words that are usually ignored) effectively.

You can manage Querqy via API endpoint, which is also very useful for automation and version control. There is also another open source project called SMUI (Search Management UI). This is a user interface to manage rules manually if you want. Below you can see a screenshot (taken from their documentation).



Confirm publish to LIVE

Are you sure to publish current Search Rules to LIVE?

Yes, publish to LIVE | No, cancel publish

"handy"
"notebook"
cheap iphone
iphone
notebook | laptop
notebook accessory
personal computer | pc
wall*

✓ SYNONYM RULE:	= (undirected)	laptop	
✓ SYNONYM RULE:	-> (directed)	netbook	
✓ UP/DOWN RULE:	UP(++)	Solr Field (empty)	asus
UP/DOWN RULE:	DOWN(---)	Solr Field (empty)	keyboard
✓ UP/DOWN RULE:	DOWN(----)	Solr Field (empty)	mouse
✓ UP/DOWN RULE:	DOWN(----)	Solr Field (empty)	Optical
✓ UP/DOWN RULE:	DOWN(-)	Solr Field (empty)	Power Cord
✓ UP/DOWN RULE:	DOWN(---)	Solr Field (empty)	spare part
✓ FILTER RULE:	-title	accessory	
✓ FILTER RULE:	-title	notebook	

Manual Implementation

You may decide to implement your own back office for search. Elasticsearch provides all necessary API. It is in your hands to dynamically change the query based on the rulesets you define for each search term.

You have all the freedom to implement the types of rules you would need for your use case. You can get some starting inspiration from Querqy if you want. Add more rule types if you need or don't implement the rules you don't need.

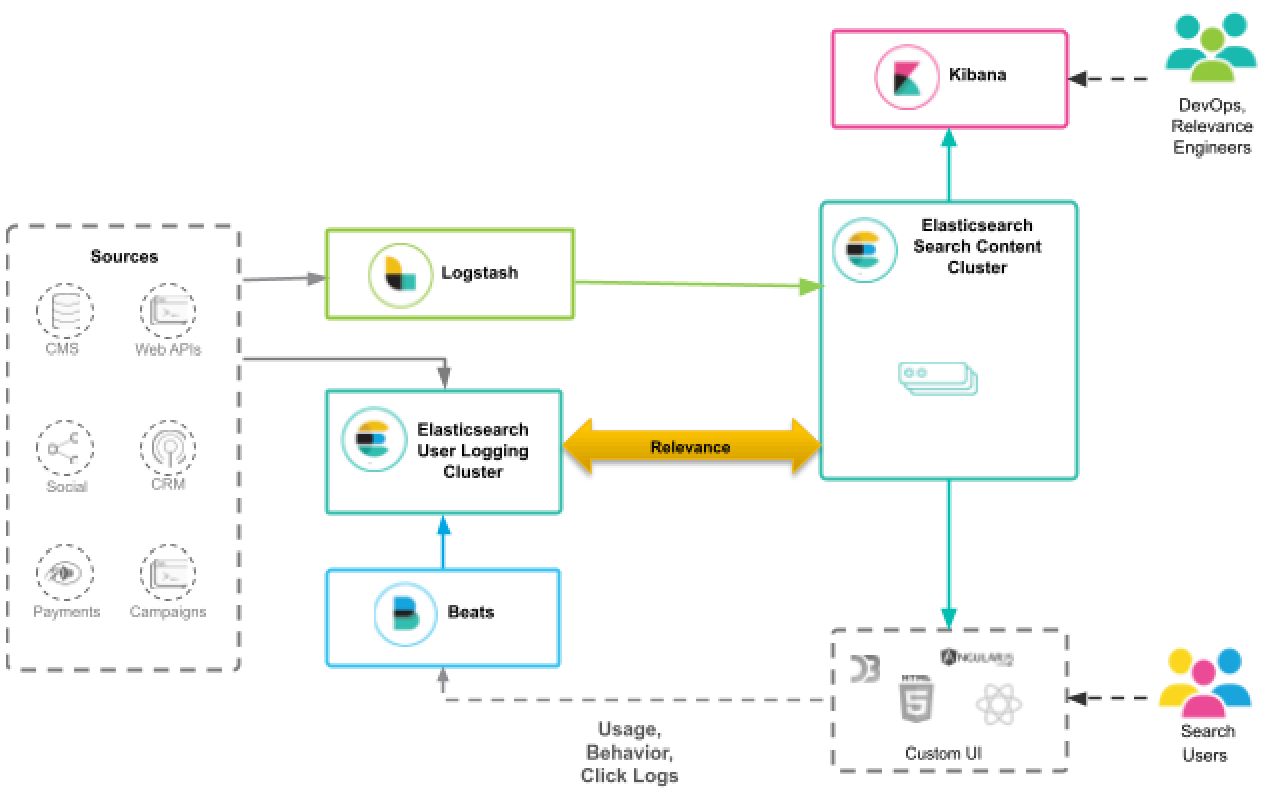
If you want to implement your own tool, I don't recommend using plugin technology. Instead, implement all the logic in a "search-service" which queries Elasticsearch APIs.

The main disadvantage of this approach is of course the initial investment. It will take some time to implement your own query rules engine.

Architecture Reference

Below, you can see a possible architecture for a search use. This architecture diagram contains some components which are optional. At the core of your architecture, you have an Elasticsearch cluster for search. You index your data into this cluster. It serves your search results.

Kibana is mostly used by the internal users (e.g. DevOps, Search Engineers, ML Engineers, Reporting) to manage Elasticsearch clusters.


Kibana
DevOps, Relevance Engineers
Sources
CMS
Web APIs
Social
CRM
Payments
Campaigns
Logstash
Elasticsearch Search Content Cluster
Elasticsearch User Logging Cluster
Relevance
Beats
Usage, Behavior, Click Logs
Custom UI
Search Users

Logstash is not a mandatory component. If you need to modify your documents during indexing, you can also use ingest pipelines instead of Logstash. For that purpose, you'll need some Ingest nodes in your cluster. You can use Logstash (and/or Ingest Pipelines) to manipulate/enrich your data.

Beats (or Elastic Agent) can be used to collect data from your search applications. If you have other tools for collecting usage data (e.g. analytics tracking, heatmaps, etc.), you can store the data in an Elasticsearch cluster or in another database of your choice.

The part with the relevance would contain your analysis pipeline. You would regularly analyze the usage data and create search rules, which optimizes your search results and overall search user experience.

The pipeline looks different for each use case and company. It depends on the data you collect and also on the KPIs you define. The goal is to maximize NDCG score before deployment of the rules and increase the KPIs after deploying the rules.

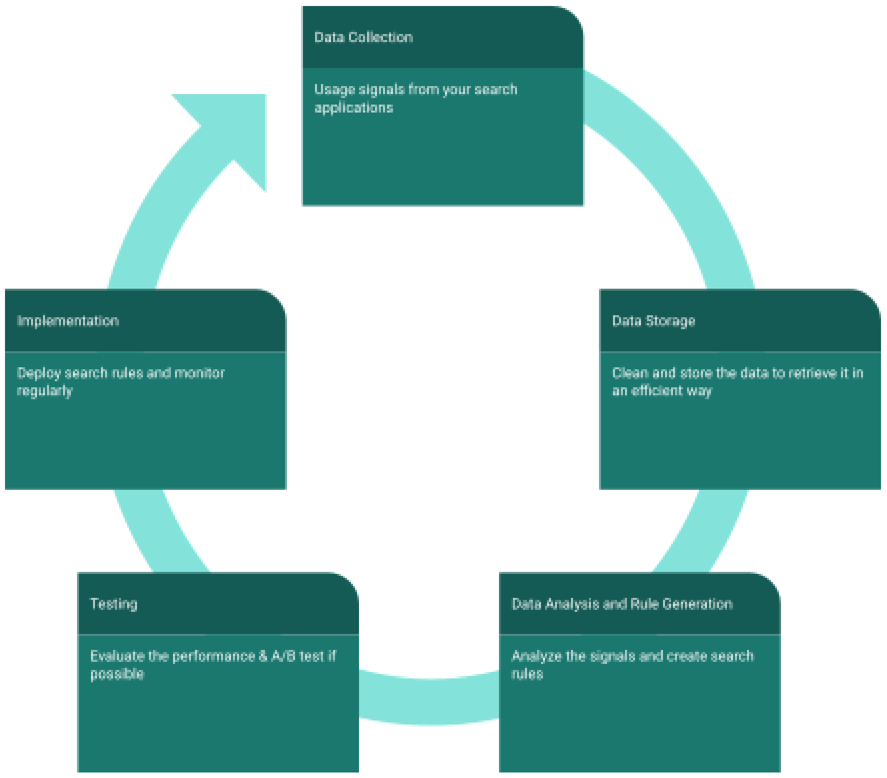
In order to store your search rules, you can use a separate Elasticsearch index or another database of your choice. In the query time, you would get the rules for each query from your rules index (or rules database) and dynamically adjust the query with the rules. Some customers choose to implement their own service for this, others use plugins such as Querqy. I explained the advantages and disadvantages of both approaches in the section "Query Rules Engine".

In the section below, I explained how I created a search optimization workflow in the past. This is also very aligned with the many other e-commerce use cases. This includes some of my previous customers and also use cases from different international conferences. This can be a starting point for your own workflow.

It is important to know that without setting up a working feedback mechanism, you'll always have issues with search optimization. You need to regularly measure and optimize by taking user signals into account.

Search Optimization Workflow

By following a search optimization workflow, a search application can become more responsive to users' needs, providing a better search experience resulting in higher engagement and customer satisfaction. There is no optimization without insights.


Data Collection
Usage signals from your search applications
Data Storage
Clean and store the data to retrieve it in an efficient way
Data Analysis and Rule Generation
Analyze the signals and create search rules
Testing
Evaluate the performance & A/B test if possible
Implementation
Deploy search rules and monitor regularly

1) Data Collection

You can collect different types of data from your search applications for the usage. We call the usage data as signals, because they signal us an intent. Below, there are some examples of what kind of data you can collect:

- **Search Logs:** Capture all search queries entered by users, including metadata such as timestamps, user identifiers, and any search filters used.
- **Click-through Data:** Record which results users click on after performing a search.

- **Engagement Metrics:** Monitor how users interact with the search results (time spent on the page, bounce rate, etc.).
- **Session Tracking:** Follow user sessions to understand sequences of queries and actions that reflect the search experience.

2) Data Storage

Store the gathered data in a robust and scalable data storage system that supports fast querying and analysis, such as a time series database for logs or a NoSQL database for structured data. In my past use cases, I used both Elasticsearch and Google BigQuery. It depends on the type of the data you store. You may have other preferences.

Clean the data by removing noise such as bots and accidental queries.

3) Data Analysis and Rule Generation

- **Search Success Metrics:** Define what constitutes a successful search (e.g., if a user clicks a result and doesn't return to the search quickly). You already have your KPIs, so this should be pretty straightforward.
- **Query Analysis:** Identify frequently searched terms, low-success rate queries, and high-drop-off points.
- **User Journey Analysis:** Map typical user paths to understand how they arrive at their desired results or where they encounter difficulties.
- **Segmentation:** Analyze behavior and search patterns by user segment (new vs. returning user, location, device, etc.).
- **Machine Learning:** Apply machine learning algorithms to predict search relevance and to group queries semantically.

Based on the insights gathered, create query rules such as:

- Synonyms and Equivalents: For commonly misspelled words or different terminologies.
- Result Boosting: Prioritize certain types of content that are more frequently engaged with.
- Filters and Facets: Adjust these elements to better meet user expectations.

- Zero Results Rules: Create fallbacks for queries that return no results, suggesting similar items or broader categories.

4) Testing

Before deploying the query rules directly to the production environment, I recommend testing it. You can use the rank evaluation API from Elasticsearch and calculate NDCG for the search results using the query rules. This will give you feedback about each rule, so that you can decide whether or not to use the rule.

Alternatively, if you have the capabilities, using an A/B testing framework to measure the impact of changes is also a good idea.

5) Implementation & Continuous Optimization

Once you implement/deploy the rules to the production environment, you should regularly monitor your performance. Continuously collect and analyze new usage data to refine or update query rules as needed.

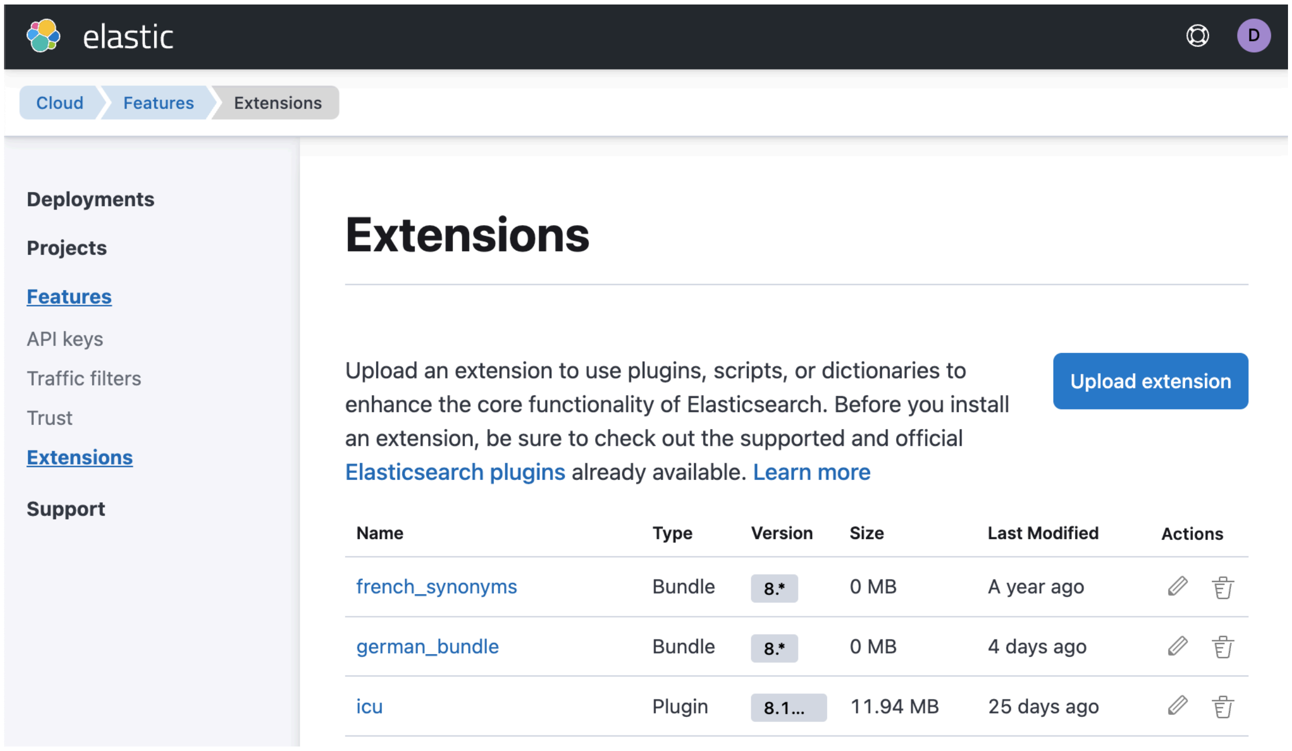
Querqy Demo

In one of our sessions, we demonstrated Querqy to show how query rule engines will help you to translate your relevancy and business rules into search engine queries.

Installation

Querqy provides an Elasticsearch plugin. The current version available is for 8.9.2. This means, you would need an Elasticsearch cluster with the same version. Download Querqy Plugin for Elasticsearch.

Go to Elastic Cloud > Features > Extension and click on "Upload extension".

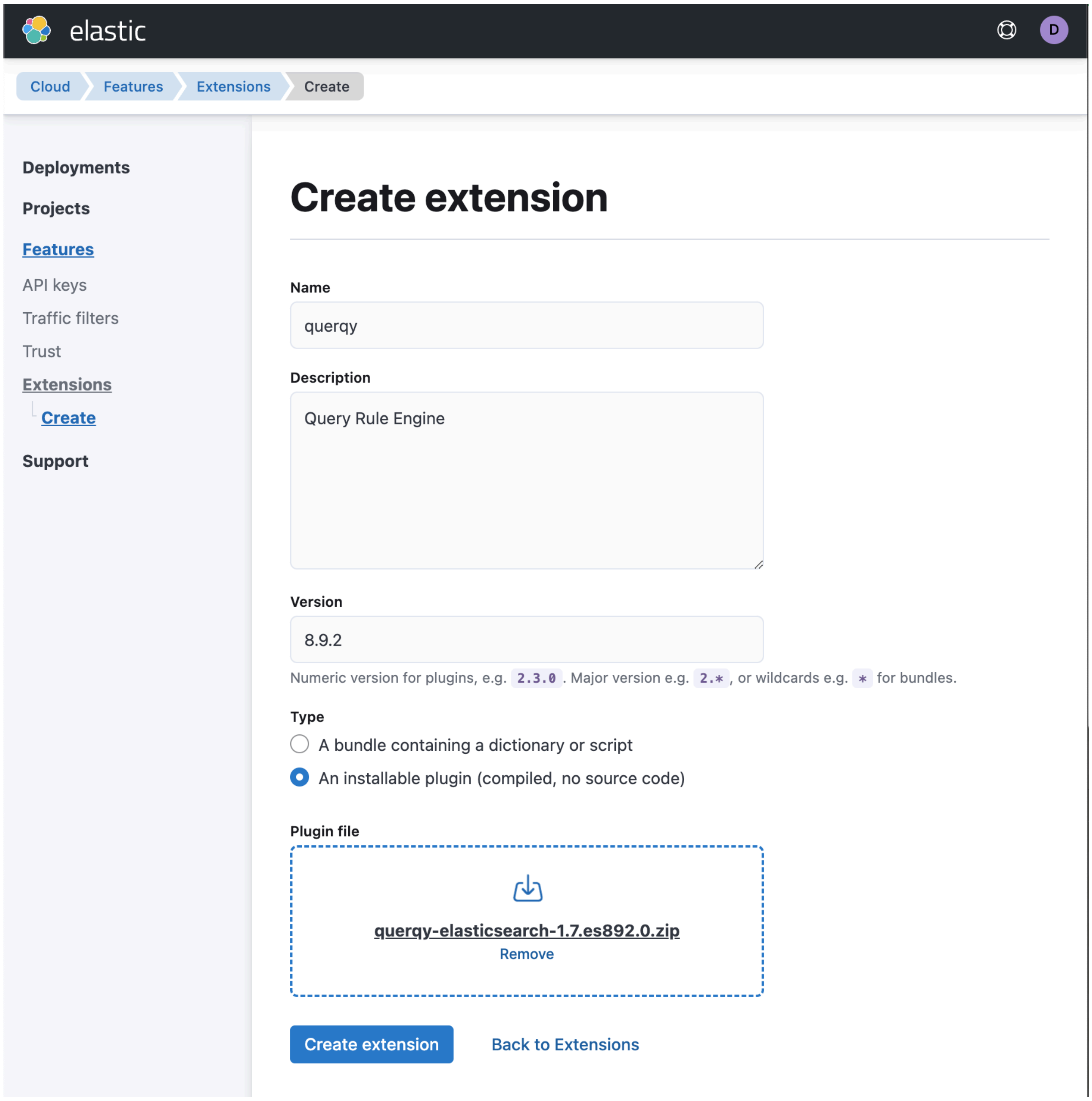


Extensions

Upload an extension to use plugins, scripts, or dictionaries to enhance the core functionality of Elasticsearch. Before you install an extension, be sure to check out the supported and official Elasticsearch plugins already available. Learn more

Name	Type	Version	Size	Last Modified	Actions
french_synonyms	Bundle	8.*	0 MB	A year ago	
german_bundle	Bundle	8.*	0 MB	4 days ago	
icu	Plugin	8.1...	11.94 MB	25 days ago	

Upload the zip file, put the relevant version number, name your extension and click on the "Create extension" button.



Create extension

Name: querqy

Description: Query Rule Engine

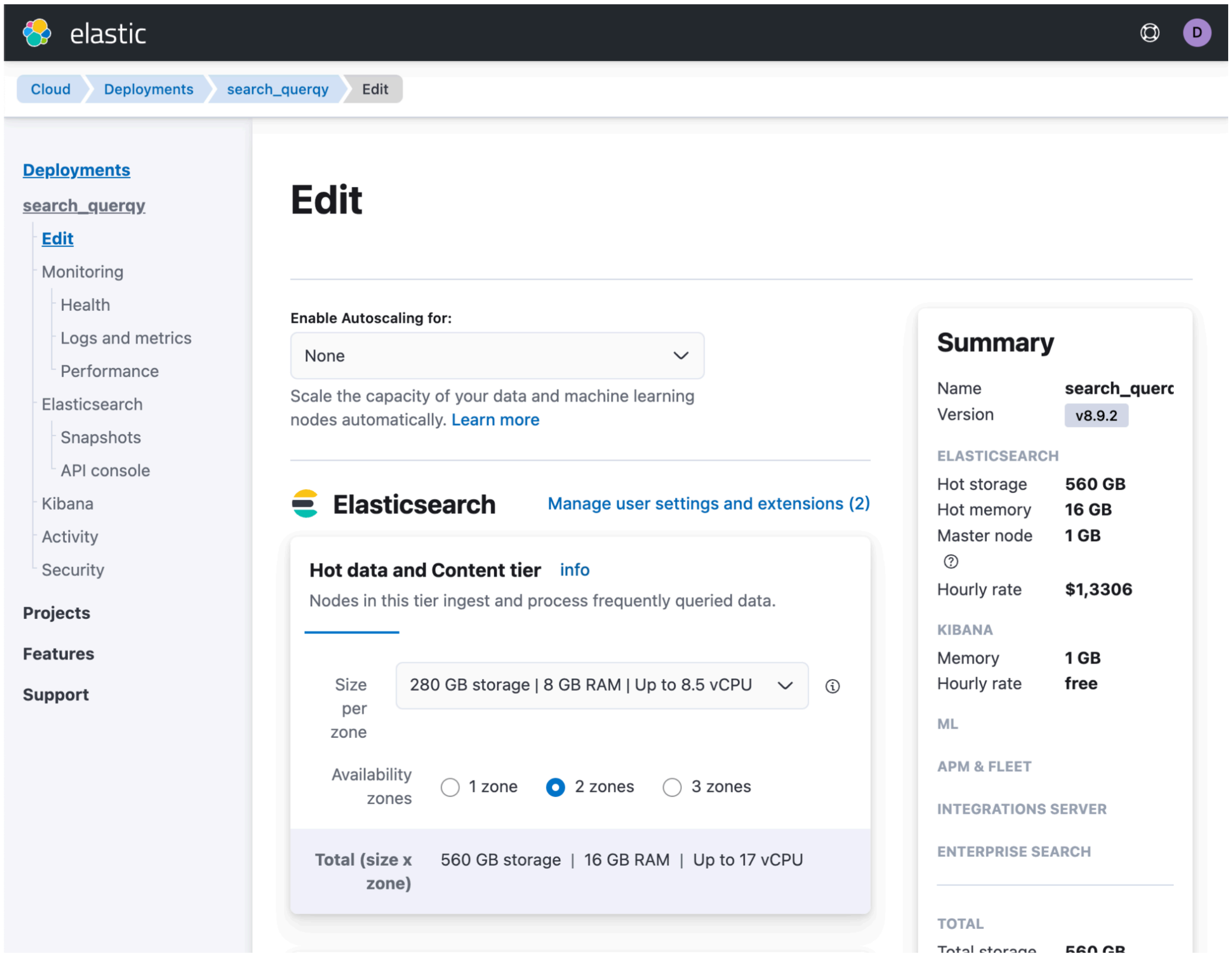
Version: 8.9.2

Numeric version for plugins, e.g. `2.3.0`. Major version e.g. `2.*`, or wildcards e.g. `*` for bundles.

Type
- ○ A bundle containing a dictionary or script
- [✓ An installable plugin (compiled, no source code)]

Plugin file: querqy-elasticsearch-1.7.es892.0.zip

Now, go to the configuration of the deployment, to which you want to add this extension, in Elastic Cloud and edit it. Next to Elasticsearch, you will see a link “Manage user settings and extensions”. Click on it. A window on the right side will appear.



Edit

Enable Autoscaling for: None

Scale the capacity of your data and machine learning nodes automatically. Learn more

Elasticsearch — Manage user settings and extensions (2)

Hot data and Content tier info

Nodes in this tier ingest and process frequently queried data.

Size per zone: 280 GB storage | 8 GB RAM | Up to 8.5 vCPU

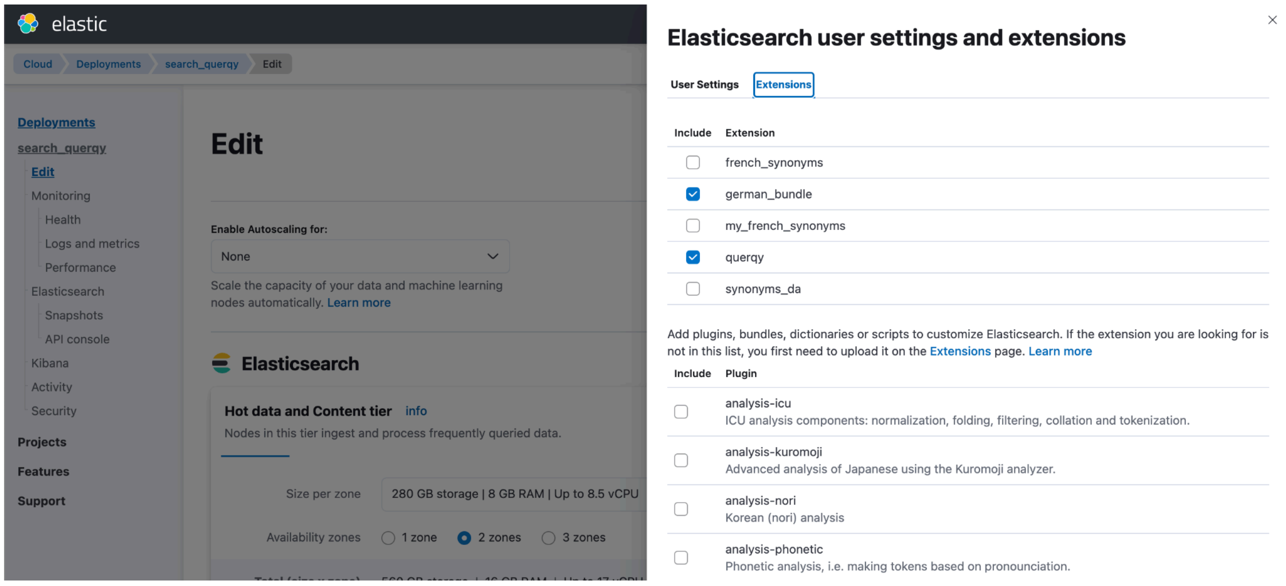
Availability zones: 1 zone [✓ 2 zones] 3 zones

Total (size x zone): 560 GB storage | 16 GB RAM | Up to 17 vCPU

Summary

Name	search_querc
Version	v8.9.2
ELASTICSEARCH	
Hot storage	560 GB
Hot memory	16 GB
Master node	1 GB
Hourly rate	$1,3306
KIBANA	
Memory	1 GB
Hourly rate	free
ML	
APM & FLEET	
INTEGRATIONS SERVER	
ENTERPRISE SEARCH	
TOTAL	
Total storage	560 GB

Click on the tab “Extensions” and activate the Query plugin and save your settings.



Edit

Enable Autoscaling for: None

Scale the capacity of your data and machine learning nodes automatically. Learn more

Elasticsearch

Hot data and Content tier info

Nodes in this tier ingest and process frequently queried data.

Size per zone: 280 GB storage | 8 GB RAM | Up to 8.5 vCPU

Availability zones: ○ 1 zone [● 2 zones] ○ 3 zones

Elasticsearch user settings and extensions

User Settings | Extensions

Include	Extension
☐	french_synonyms
✓	german_bundle
☐	my_french_synonyms
✓	querqy
☐	synonyms_da

Add plugins, bundles, dictionaries or scripts to customize Elasticsearch. If the extension you are looking for is not in this list, you first need to upload it on the Extensions page. Learn more

Include	Plugin
☐	analysis-icu ICU analysis components: normalization, folding, filtering, collation and tokenization.
☐	analysis-kuromoji Advanced analysis of Japanese using the Kuromoji analyzer.
☐	analysis-nori Korean (nori) analysis
☐	analysis-phonetic Phonetic analysis, i.e. making tokens based on pronounciation.

After the settings are saved, you can use the plugin.

Usage

We are going to create a ruleset with only one rule for simplicity reasons. Of course you can add more rules.

```
Unset
curl --request PUT \
 --url <ELASTICSEARCH_ENDPOINT>/_querqy/rewriter/common_rules \
 --header 'Authorization: ApiKey <API_KEY>' \
 --data '{
 "class": "querqy.elasticsearch.rewriter.SimpleCommonRulesRewriterFactory",
 "config": {
  "rules": "metal =>\nSYNONYM: metall"
 }
}'
```

We take one of the examples from you, which is apparently a common misspelling (metal → metall). Adding a synonym rule to this misspelled word will replace it with the correct one (also in multi-term searches).

When I run your original query for the search “gummi metal puffer”, I get zero results. Once I change your original query to use Querqy rules, I get 307 results. This is the exact same number when I search for “gummi metall puffer”.

The place I change in your original query is the must query. Instead of the multi-match in your original query, I use the following snippet:

```
Unset
{
 "querqy": {
  "matching_query": {
   "query": "gummi metal puffer"
  },
  "query_fields": [<SAME SEARCH FIELDS>],
  "rewriters": ["common_rules"],
  "minimum_should_match": "100%"
 }
}
```

We can create more rules for your other examples. Let’s take the example “klemmschelle” vs. “rohrschelle”. Very similar products, but not really synonyms. Assume, as a result of your data analysis, you see that your users who are searching for “klemmschelle”, are also looking for “rohrschelle” products. Your analysis shows you would have better NDCG if you first show “klemmschelle” products, followed by “rohrschelle” products. If you simply put a synonym, you may have “rohrschelle” products in better positions, so you want to somehow define a ranking. This is possible with **weighted synonyms**. Here is an example rule:

```
Unset
{
 "class": "querqy.elasticsearch.rewriter.SimpleCommonRulesRewriterFactory",
 "config": {
  "rules": "klemmschelle =>\nSYNONYM(0.8): rohrschelle"
 }
}
```

The 0.8 next to the rule type synonym shows the weight of the synonym, which is 80%. Please note that you may need to test different weights to find optimal NDCG.

Handling Zero Results

tbd