



GitLab

Elasticsearch Deep Dive

June 13, 2019



1. What is Elasticsearch? Why do we want it?
2. Differences between DB search and ES search
3. Why isn't it used on GitLab.com
4. Initial setup
5. Elasticsearch schema and analyzers
6. Interaction with Rails models
7. How search works
8. gitlab-elasticsearch-indexer

What is Elasticsearch? Why do we want it?



- Search and analytics engine built on Apache Lucene
 - Open-source
 - RESTful
 - Distributed
- most popular search engine
 - log analytics
 - full-text search

What is Elasticsearch? Why do we want it?



- Accepts JSON documents using the API or ingestion tools such as Logstash.
- Automatically stores the original document
- Adds a searchable reference to the document in the cluster's index.
- Permits you to search and retrieve the document using the Elasticsearch API
 - Can also use Kibana to visualize your data and build interactive dashboards.

What is Elasticsearch? Why do we want it?



- High performance
 - The distributed nature of Elasticsearch enables it to process large volumes of data in parallel, quickly finding the best matches for your queries.
- Near real-time operations
 - Elasticsearch operations such as reading or writing data usually take less than a second to complete. This lets you use Elasticsearch for near real-time use cases such as application monitoring and anomaly detection.

Differences between DB search and ES search



- Main difference: Allows for global blob (Code) and commit search
 - DB only allows project-level searches
- Note: Filtered search does not currently use Elasticsearch

Search

test Group: Any Project: Any

Advanced search functionality is enabled.

Projects 21 Issues 1 Merge requests 0 Milestones 0 **Code 1000+** Commits 1000+ Wiki 1 Users 0

Showing 81 - 100 of 2598 blobs for "test"

Gnuwget / Wget2: unit-tests/test-dl-dummy.c

```
34
35 WGET_EXPORT void d_test_write_param(char *buf, size_t len);
36 WGET_EXPORT void concat(d_test_fn_, PARAM)(char *buf, size_t len);
37
38 void d_test_write_param(char *buf, size_t len)
```

Commit451 / Lab Coat: app/src/test/java/com/commit451/gitlab/ApiTests.kt

```
25 //for logging
26
27 gitLab = TestUtil.login()
28
29 val projectResponse = gitLab
```

Flightjs / Flight: test/test-main.js

```
8
9 // ask Require.js to load these files (all our tests)
10 deps: tests,
11
12 // start test run, once Require.js is done
```

Why isn't it used on GitLab.com?



- Enabling it for all projects would result in a 66% storage increase
 - Analysis available here:
https://gitlab.com/gitlab-com/gl-infra/infrastructure/issues/1597#note_38908523
- Administration support is lacking
 - Work is ongoing: <https://gitlab.com/groups/gitlab-org/-/epics/428>
- No way to do zero-downtime deploys - requires a rails restart at a minimum (for now)
 - Problem is equivalent to DB migrations, but no good tooling around it
- Good news: an MVC is going live very soon as we've completed enough work (<https://gitlab.com/groups/gitlab-org/-/epics/853>) to allow us to enable it for a subset of groups/projects



- Installing Elasticsearch
 - Requirements available [in our documentation](#)
- Initial indexing of content
 - Done via rake tasks
 - [Soon to be added to the admin console](#)
 - ``gitlab:elastic:index``
 - Runs all indexing operations in the foreground, except repo indexing
 - Suitable for all but extremely large instances, which must run each indexing operation separately in order to avoid overloading sidekiq
- Enabling indexing and search via Elasticsearch



Elasticsearch

Elasticsearch integration. Elasticsearch AWS IAM.

- Elasticsearch indexing
- Use the [new repository indexer \(beta\)](#)
- Search with Elasticsearch enabled

URL

The url to use for connecting to Elasticsearch. Use a comma-separated list to support clustering (e.g., "http://localhost:9200, http://localhost:9201").

Number of Elasticsearch shards

How many shards to split the Elasticsearch index over. Changes won't take place until the index is [recreated](#).

Number of Elasticsearch replicas

How many replicas each Elasticsearch shard has. Changes won't take place until the index is [recreated](#).



Elasticsearch indexing restrictions

Limit namespaces and projects that can be indexed

Namespaces to index

Jlevy × Gitlab Org ×

Projects to index

Gnuwget / elastic_test48 ×



- All objects have the same document-type and live in the same index
 - Permits us to have parent-child relationships
 - We depend on these relationships for permission checks
 - Requires us to implement our own separate type checks
 - All types share all fields, which means we have lots of sparse fields
 - ES 6.0 has great storage improvements for sparse fields which means we don't get a big storage penalty
- We should probably move to one index per type, but:
 - We lose the ability to filter by project attributes OR
 - We are forced to denormalize project data into every class type, ballooning storage usage



- Analyzers are where the search magic happens
 - Prepare the data for better searching
 - Each analyzer increases storage needs
 - They're composed of tokenizers and filters
- For models by default we use the standard tokenizer with three filters
 - Standard: doesn't really do anything
 - Lowercase: normalizes text to lowercase
 - My_stemmer: a custom stemmer filter that uses light_english stemmer
- Models also have a ``my_ngram_analyzer`` which creates 2- and 3-grams for Projects'
``name_with_namespace``



- Repositories and Commits have more interesting analyzers
- We do a lot of tokenizing with `asciifolding` and `lowercase` filters
- Code analyzer is special
 - `edgeNGram` filter that creates grams between 2 and 40 characters wide
 - Custom `code` filter with lots of regex patterns
 - Extracts digits, class names, terms inside quotes, separates terms on periods, and separates path terms
 - Custom `sha_analyzer` which tokenizes using ngrams between 5 and 40 characters

```
filter: {
  code: {
    type: "pattern_capture",
    preserve_original: true,
    patterns: [
      "(\\p{Ll}+|\\p{Lu}\\p{Ll}+|\\p{Lu}+)",
      "(\\d+)",
      "(?=[\\p{Lu}]|[\\p{Ll}]+)",
      "'((?:\\\\"|"[^"]|\\")*)"', # capture terms inside quotes, removing the quotes
      "'((?:\\\\"|'[^']*|\\')*)'", # same as above, for single quotes
      '\\.([^.]+)(?=[.\\s\\Z])', # separate terms on periods
      '\\/(?[^\v/]+)(?=[\v|b])' # separate path terms (like/this/one)
    ]
  },
  edgeNGram_filter: {
    type: 'edgeNGram',
    min_gram: 2,
    max_gram: 40
  }
}
```



- We use a customized elasticsearch-rails gem to link up our models with ES
- **ApplicationSearch** module is the entry-point that defines callbacks and shared methods
 - Each class defines their own `*Search` module (for example, `ProjectsSearch`)
 - These classes define base elasticsearch query structure and special indexing concerns
- **ApplicationSearch** defines basic security concerns like filtering by projects the current user has access to
- **Elasticsearch::Git::Repository** defines Blob, WikiBlob, and Commit interactions
 - Need a separate module because repos are not in the database
 - We only index the default branch, otherwise costs would skyrocket
 - We have two indexers: a rails script (due to be removed!) and **gitlab-elasticsearch-indexer**



- <https://gitlab.com/gitlab-org/gitlab-elasticsearch-indexer>
- Written in Go
- Replacement for bin/elastic_repo_indexer, slated for 12.1
- Greatly improved speed (3-10x!) and resource usage
 - Better memory handling, but still memory hungry
 - Much better I/O (our bottleneck when reading repository data) and encoding detection
 - Allows us to hide from the sidekiq OOM killer
- Used only for blobs (which includes wiki blobs) and commits
- Talks to Gitaly, gets a diff between last_commit as found in IndexStatus and the current SHA
 - Add new blobs, reindexes changed blobs, and deletes removed blobs to the ES index
 - Indexes commits as well!
 - Assumes that commits are only ever added (oops):
<https://gitlab.com/gitlab-org/gitlab-ee/issues/10937>



- **ApplicationSearch** defines callbacks for incremental indexing when models get updated
 - Insert, Update, and Destroy all trigger ES updates via **ElasticIndexerWorker**
- Repositories get updated via **GitPush** worker hooks
 - **ElasticCommitIndexerWorker** calls **Gitlab::Elastic::Indexer**
 - **Gitlab::Elastic::Indexer** decides whether to call rails script or **gitlab-elasticsearch-indexer**
 - Can trigger partial updates (FROM and TO SHAs)
 - The last commit that was indexed is kept in the DB in the **IndexStatus** model



- An elasticsearch query is a JSON structure that can contain multiple filters
- We implement permissions as bool filters on the original Elasticsearch query
 - We can filter for projects a user has access
 - Filter for projects with features enabled (ex. public issue tracker)
- Highlighting is given to us by Elasticsearch
 - “Highlight” field in query with fields to highlight
 - Response contains a “highlight” element for each search hit with highlighted fragments

```
{
  "query": {
    "bool": {
      "must": {
        "term": { "user": "kimchy" }
      },
      "filter": {
        "term": { "tag": "tech" }
      },
      "must_not": {
        "range": {
          "age": { "gte": 10, "lte": 20 }
        }
      },
      "should": [
        { "term": { "tag": "wow" } },
        { "term": { "tag": "elasticsearch" } }
      ]
    }
  }
}
```



- We expose Elasticsearch's simple_query_string
 - Allows users to use exclusion operators
 - Exact search matches
 - Complex, but powerful

- We also enhance it with our own syntax search filters
 - Defined using **Gitlab::Search::Query**
 - Relevant usages in `lib/gitlab/file_finder.rb` and `ee/lib/elasticsearch/git/repository.rb`
 - Allow users to filter by path, filename, or extension

Questions?

Check the Google Doc at

<https://docs.google.com/document/d/1cwo5n3XYaTDAJ48sMZJ8bHQVJ0RD5dlsdf28L96OZQw/edit?pli=1#>