

Elasticsearch Deep Dive June 13, 2019

#### Content

- 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.

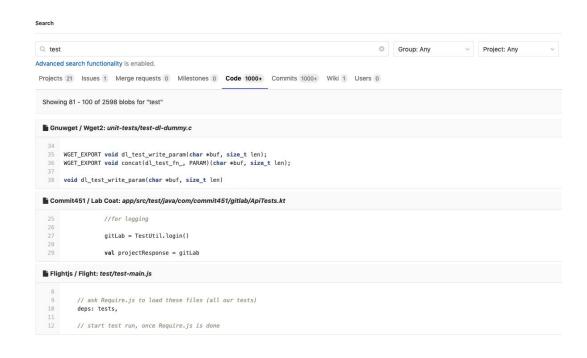
# What is Elasticsearch? Why do we want it?

#### 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



### Why isn't it used on GitLab.com?

- Enabling it for all projects would result in a 66% storage increase
  - Analysis available here:
     <a href="https://gitlab.com/gitlab-com/gl-infra/infrastructure/issues/1597#note">https://gitlab.com/gitlab-com/gl-infra/infrastructure/issues/1597#note</a> 38908523
- Administration support is lacking
  - Work is ongoing: <a href="https://gitlab.com/groups/gitlab-org/-/epics/428">https://gitlab.com/groups/gitlab-org/-/epics/428</a>
- No way to do zero-downtime deploys requires a rails restart at a minimum (for now)
  - o 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
   (<a href="https://gitlab.com/groups/gitlab-org/-/epics/853">https://gitlab.com/groups/gitlab-org/-/epics/853</a>) to allow us to enable it for a subset of groups/projects

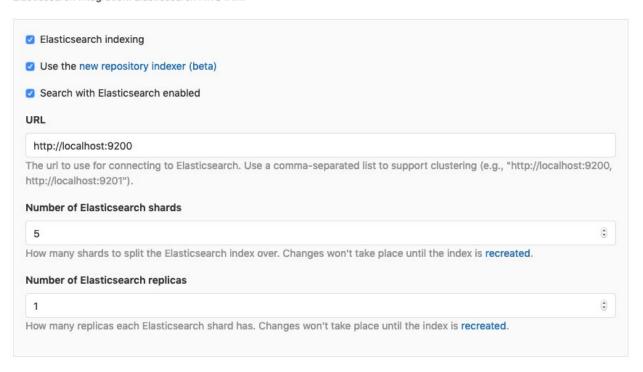
# Initial setup

- 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

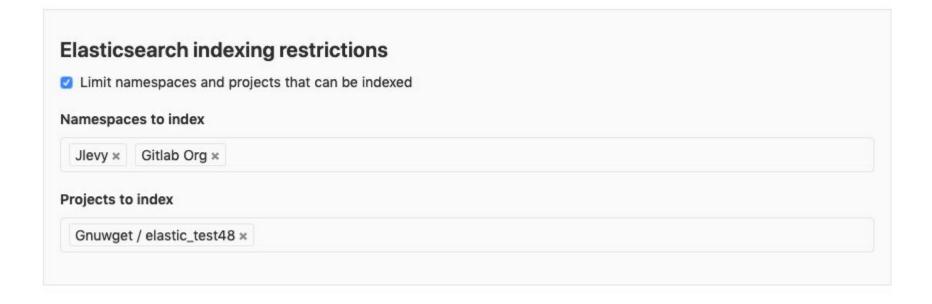
# Initial setup

#### Elasticsearch

Elasticsearch integration. Elasticsearch AWS IAM.



# Initial setup



### Schema and Analyzers

- 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

# Schema and Analyzers

- 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 <u>standard tokenizer</u> with three filters
  - Standard: doesn't really do anything
  - Lowercase: normalizes text to lowercase
  - My\_stemmer: a custom stemmer filter that uses <u>light\_english</u> stemmer
- Models also have a `my\_ngram\_analyzer` which creates 2- and 3-grams for Projects'
   `name\_with\_namespace`

# Schema and Analyzers

- Repositories and Commits have <u>more</u> <u>interesting analyzers</u>
- We do a lot of tokenizing with `asciifolding` and `lowercase` filters
- Code analyzer is special
  - <u>edgeNGram</u> 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{L}]+))",
      '"((?:\\"|[^"]|\\")*)"', # capture terms inside quotes, removing the quotes
      "'((?:\\'|[^']|\\')*)'", # same as above, for single quotes
      '\.([^.]+)(?=\.|\s|\Z)', # separate terms on periods
      '\?([^\]+)(2=\/|\b)' # separate path terms (like/this/one)
  edgeNGram_filter: {
    type: 'edgeNGram',
    min_gram: 2,
    max_gram: 40
```

#### Interaction with Rails models

- We use a <u>customized</u> 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 <u>removed!</u>) and <u>**gitlab-elasticsearch-indexer**</u>

#### 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

#### Interaction with Rails models

- 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

#### How search works

- 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, "Ite" : 20 }
  "should": [
    { "term" : { "tag" : "wow" } },
    { "term" : { "tag" : "elasticsearch" } }
```

#### How search works

- We expose Elasticsearch's <u>simple query string</u>
  - Allows users to use exclusion operators
  - Exact search matches
  - Complex, but powerful
- We also enhance it with our own <u>syntax search filters</u>
  - Defined using <u>Gitlab::Search::Query</u>
  - 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/ed

it?pli=1#