Search GitLab, Redmine, and repositories with a single query

**Deploy an enterprise search server with Fess**

FOSDEM 2023

Takashi Kumagai, TOSHIBA Corporate Software Engineering Center
February 4, 2023
Today’s talk

- Our engineers use multiple content management tools

- **Searching** became a problem
  - Can’t search laterally across multiple tools
  - Can’t search texts inside binary files

- How we use Fess to solve the problem of search

- How we overcame (some of the) shortcomings of Fess

- Contributions (patches)

- Conclusion
Contents

01 Introduction
02 Background
03 Fess and enterprise search
04 Customization (patching Fess)
05 Automating configurations
06 Conclusion
 Contents

01 Introduction
02 Background
03 Fess and enterprise search
04 Customization (patching Fess)
05 Automating configurations
06 Conclusion
Who am I

- An engineer at Corporate Software Engineering Center in Toshiba
- 2006 – 2015
  - Develop applications for Toshiba’s products (HDTV, etc.)
- 2015 – 2022
  - Maintain and improve the company’s cloud infrastructure and automation processes.
    - Build in-house tools
    - Write automation scripts
    - Automate E2E tests
    - Configure and deploy search servers to departments in other Toshiba companies

Development team engineers

Vũ Thị Thanh Thanh

Đỗ Thành Trung

Hoàng Trung Hiếu

Lê Tiến
Contents

01 Introduction
02 Background
03 Fess and enterprise search
04 Customization (patching Fess)
05 Automating configurations
06 Conclusion
Providing software development tools

Our team

Deploy tools

- GitLab
- REDMINE
- git

Use

- Engineers (users)
  - Product development dept. at company A
- Product development dept. at company B
- R&D department at company B

…

about 200 deployments
2 major issues resulting from using multiple tools

Can’t search across tools

Resources, e.g. source code, documents, and wiki, are stored in multiple content management tools (fragmentation).

Can’t search texts inside binary files
Goal

One search box to rule them all
Approach

- Making a search engine by ourselves would be too expensive.
- Better to find an **OSS search server** and configure/customize it for our needs.

![Diagram]

- Requirements:
  - Shall solve the aforementioned 2 problems.
  - Shall be complete with the essential features we need, e.g.
    - Search box UI for users
    - Features for admins (easily set up and run web crawlers, etc.).
  - *(A few more requirements; more on them later)*

- An OSS search server named **Fess** stood out as the closest to what we needed.
Fess

*software for searching for information within an enterprise*

- **An enterprise search** server
- Developed by a company named CodeLibs
- Uses Elasticsearch as the search engine
  - Supports indexing of certain binary files, e.g., Microsoft Office, PDF, and zip.
- Comes with several types of crawlers, which can crawl
  - documents on a web server
  - file system
  - data store (such as a CSV or database).
Fess web crawler

Contents on the web

GitLab

Web pages

Uploaded files

Redmine

Web pages

Uploaded files

Web crawler

Crawls and indexes contents
Web crawler configuration basics (1 of 3)

- Crawls and indexes web page contents (texts, uploaded files, etc.) by following links recursively.
- Fess provides an admin console GUI to create a crawler.
# Web crawler configuration basics (2 of 3)

## Web crawler parameters

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>URLs</td>
<td>Starting point of the web crawling/indexing</td>
<td><a href="https://mygitlab.io">https://mygitlab.io</a></td>
</tr>
</tbody>
</table>
| Include URLs (regex)   | Web crawler will crawl the page only if the URL matches any of the listed regex. | ^https://mygitlab\io/myproject/-/issues\d+$  
|                        |                                                                             | ^https://mygitlab\io/myproject/-/wikis?!.*/edit$.*$                                      |
| Exclude URLs (regex)   | Prevent crawlers from indexing irrelevant pages.                            | ^https://mygitlab\io/help($|/?.*$)                                                     
|                        |                                                                             | ^https://mygitlab\io/(!.*/-.*/-)/.+/-/new/.*$                                             |
| Depth                  | Maximum recursion depth                                                    | 1000                                                                                     |
| Max Access Count       | The maximum number of pages to crawl.                                       | 10000000                                                                                 |
| Permissions            | Determines who can find the indexed contents                                | {group}mygroup  
|                        |                                                                             | {user}user1  
|                        |                                                                             | {user}user2  
|                        |                                                                             | {user}user3                                                                 |
Permission settings and access control

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissions</td>
<td>Determines who can find the indexed contents</td>
<td>{group}mygroup, {user}user1, {user}user2, {user}user3</td>
</tr>
</tbody>
</table>

- Allows admin to implement per-user access control
- Permission settings are **per-web crawler**
  - e.g. 100 projects on GitLab → create 100 web crawlers, each of which has its own permission settings
- **username** in {user}username can be either
  - Users created on Fess
  - Users authenticated via LDAP directory service
Contents

01 Introduction
02 Some background
03 Fess and enterprise search
04 Customization (patching Fess)
05 Automating configurations
06 Conclusion
# Customization (patching Fess)

- Patches delineated in this segment:
  - Authentication for web crawlers
  - Crawling and indexing repository contents

<table>
<thead>
<tr>
<th>Patch</th>
<th>Link* / patched file(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Disable default PostConstruct</td>
</tr>
<tr>
<td>2</td>
<td>Make webdriver quit after it finishes crawling</td>
</tr>
<tr>
<td>3</td>
<td>(webdriver) add option for specifying chrome arguments</td>
</tr>
<tr>
<td>4</td>
<td>(webdriver) add option for chrome.prefs</td>
</tr>
<tr>
<td>5</td>
<td>(webdriver) add authentication</td>
</tr>
<tr>
<td>6</td>
<td>(webdriver) support crawl files</td>
</tr>
<tr>
<td>7</td>
<td>(webdriver) add basic auth</td>
</tr>
<tr>
<td>8</td>
<td>Add saml auth</td>
</tr>
<tr>
<td>9</td>
<td>map filesystem paths to URLs</td>
</tr>
<tr>
<td>10</td>
<td>(webdriver) customize last modified</td>
</tr>
<tr>
<td>11</td>
<td>(webdriver) prioritize fess cookies</td>
</tr>
<tr>
<td>12</td>
<td>(webdriver) add waiting configurations</td>
</tr>
</tbody>
</table>

*paths of Link URLs following https://github.com/

- Patches merged upstream
- Patch submitted but not merged
- Patches kept proprietary
Authentication for web crawlers

- Web crawlers need authentication.
- Fess’s **web authentication** supports certain types of login pages but not all.

- We wrote a patch to authenticate the web crawler through our SAML-enabled GitLab sign-in page
  - **Patched webdriver client and http form**
Patching webdriver client and HTTP form

New optional parameters on Fess admin console

Fess console: (web authentication)

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>user</td>
</tr>
<tr>
<td>Password</td>
<td>*****</td>
</tr>
<tr>
<td>Parameters</td>
<td>encoding=UTF-8</td>
</tr>
<tr>
<td></td>
<td>token_method=GET</td>
</tr>
<tr>
<td></td>
<td>token_url=<a href="https://tccloud2.toshiba.co.jp/ditc/gitlab/users/sign_in">https://tccloud2.toshiba.co.jp/ditc/gitlab/users/sign_in</a></td>
</tr>
<tr>
<td></td>
<td>token_pattern=name=&quot;authenticity_token&quot; +value=&quot;([^]]+)&quot;</td>
</tr>
<tr>
<td></td>
<td>token_name=authenticity_token</td>
</tr>
<tr>
<td></td>
<td>login_method=POST</td>
</tr>
<tr>
<td></td>
<td>login_parameters=username=${username}&amp;password=${password}</td>
</tr>
<tr>
<td></td>
<td>saml_url=<a href="https://mygitlab.io/users/auth/saml">https://mygitlab.io/users/auth/saml</a></td>
</tr>
<tr>
<td></td>
<td>saml_login_url_pattern=action=&quot;([^]+)&quot;</td>
</tr>
<tr>
<td></td>
<td>saml_response_pattern=name=&quot;SAMLResponse&quot; +value=&quot;([^]+)&quot;</td>
</tr>
<tr>
<td></td>
<td>saml_response_name=SAMLResponse</td>
</tr>
<tr>
<td></td>
<td>saml_callback_url=<a href="https://mygitlab.io/users/auth/saml/callback">https://mygitlab.io/users/auth/saml/callback</a></td>
</tr>
</tbody>
</table>

Optional saml parameters are defined in the Parameters field.

The patched web driver client attempts SAML login if optional parameters are defined.

Crawler supports SAML login

The patched form parser checks for optional saml parameters and stores them if they are present.
Crawling and indexing repository contents

- Many of our repositories are several GBs in size.
- Indexing the repo contents using web crawlers turned out to be too slow.

Approach 1

Web crawler → Load balancer → apache → GitLab → repo

Web crawlers can index files in repo but it takes much longer
- HTTP request to GitLab → GitLab fetches repo file contents → GitLab renders the file contents on the page*

Approach 2

File crawler

Clone files (main branch HEAD)

Files on local filesystem

Open the file locally and read its content → much faster
Problem with approach 2

File path pointed to the local file on the Fess instance

file:///home/user/repo/myproject/README.md

Patched web driver client re-maps the filesystem path to URL

https://mygitlab.io/myproject/-/blob/main/README.md
Patching file system client (re-mapping file system paths to URLs)

Fess admin console
(file crawling configuration)

<table>
<thead>
<tr>
<th>URLs</th>
<th><a href="https://mygitlab.io">https://mygitlab.io</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Included URLs</td>
<td><a href="https://mygitlab%5Cio/.+">https://mygitlab\io/.+</a></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Config Parameters</td>
<td>client.prefixUrl=/home/fess/repo/mygroup/</td>
</tr>
<tr>
<td></td>
<td>client.mapUrl=&quot;<a href="https://mygitlab.io/mygroup/">https://mygitlab.io/mygroup/</a>&quot; + project +</td>
</tr>
<tr>
<td></td>
<td>&quot;-/blob/main/&quot; + relativeFilePath</td>
</tr>
</tbody>
</table>

The patched config parser checks for optional `client.prefixUrl` and `client.mapUrl` parameters and stores them if they are present:

```
If( client.prefixUrl && client.mapUrl ) {
  url = remap(filepath)
}
```
Contents

01  Introduction
02  Some background
03  Fess and enterprise search
04  Customization (patching Fess)
05  Automating configurations
06  Conclusion
Automating configurations

- Manual configuration using GUI became impractical as the number of configurations increased.
- For each Fess instance,
  - More than 10 configurations
  - Several hundred web crawlers to create
- Without automation, the administrator would have to
  - manually edit lots of config files.
  - do repetitive yet complicated GUI operations to create web crawlers, web auth objects, schedulers, etc. on admin console.
**Why automate**

Search results should show only the resources (pages, files, etc.) in projects the user has access to.

Need to create as many web crawlers as the number of the GitLab projects. → ...
Automation example using Fess APIs

Creating web crawlers for each GitLab project:

```python
# Get all the projects on GitLab (GitLab API)
projects = gitlab.get_all_projects()

for project in projects:
    # Get all members of project
    users = gitlab.get_project_members(project)

    # Create web crawler for project (Fess API)
    webcrawler = fess.create_webcrawler(project.url, users, ...)

    # Create web authentication for the web crawler
    webauth = fess.create_webauth(project.url + '/login', webcrawler, username, password)

    # Create job scheduler for the web crawler
    ...
```

Automation script (Python)
Contents

01 Introduction
02 Some background
03 Fess and enterprise search
04 Customization (patching Fess)
05 Automating configurations
06 Conclusion
Conclusion

- Did Fess solve our problems?
  - Can’t search across tools
    - → Solved. Our users can now search content management tools laterally.
  - Can’t search texts inside binary files
    - → Solved. Our users can find PDF and Office documents by texts contained in them.

- Web crawler performance
  - What our preliminary deployments revealed

Fess
- 4-core CPU
- 16 GB RAM

GitLab
- 500 projects
- 5000 issues

Takes about a couple of days to index everything
Committed to People, Committed to the Future.
Appendix: Crawling dynamic web pages

1. Fess’s webdriver-based crawler client can index most of the dynamically loaded contents (Fess 13)
   • You need to change the crawler client from the default to a webdriver-based one, which requires editing config files (XML) and downloading and copying jar files to a specific directory (but no patching is required).
   • Web driver client tends to be less tested/mature; we fixed memory leaks and the PRs were accepted and merged upstream.

2. Our dev team wrote a patch to index contents on GitLab 15
   • The stock webdriver client didn’t support issue pages on GitLab 15
   • Patched web driver client to wait for the specified elements to appear on the page.

- Starting Fess 14, webdriver client is no longer supported.
- A new crawler client powered by playwright is on its way for Fess 14, but no roadmap has been given by CodeLibs