MOSAIC Developer Guide

Living Document created by Sebastian Gürtl and Alexander Nussbaumer

Last update: 23 May 2024

Contents

General note: We recommend to use Linux, macOS or Windows Subsystem for Linux (WSL) for the installation and development of MOSAIC.

1) Installation

The first step towards the installation of MOSAIC is to clone (or download) the repository by running the following command:

```
https://opencode.it4i.eu/openwebsearcheu-public/mosaic.git
```

After cloning this repository, change the directory to the cloned repository:

```
cd mosaic
```

Now you can use one of the possibilities described below to build and start MOSAIC.

Installation Options

Option 1: Use Scripts

Prerequisites:

- Install Java (JDK 17+)

We provide several scripts with which you can build and launch the application. To first build and then start the application, execute the following commands:

```
# change the directory
cd scripts
# import indexes, clean the project and create a packaged JAR
./build.sh
# run the executable
./start.sh "[OPTION]" [API_PORT]
```

If you want to build and run the application on Windows, you can use the batch files instead of the bash scripts.

Option 2: Build and Run Application Manually

Prerequisites:

- Install Java (JDK 17+)
- Install Maven (3.8.2+)

To build a JAR and run the Java executable you just need to execute the following commands:

```
# change the directory
cd search-service
# clean the project and compile the source code
mvn clean compile
# build the executable
mvn package
# run the executable
java  [-Dquarkus.http.port=<API_PORT>]  -jar  core/target/service.jar
[OPTION]
```

A concrete example for running the executable without a custom port number and without any options is:
```
java -jar core/target/service.jar
```

A concrete example for running the executable with a custom port number (e.g., 8009) is:
```
java -Dquarkus.http.port=8009 -jar core/target/service.jar
```

Setting the port number using `-Dquarkus.http.port=<YOUR_API_PORT>` is optional. By default, the search service runs on port `8008`. Note that this approach assumes that your Lucene index(es) is already located in the directory "lucene" and its associated metadata is located in the respective directory in "resources". If you need to import your CIFF index to a Lucene index first, you can use the importer script.

Option 3: Dev Mode

Prerequisites:
- Install Java (JDK 17+)
- Install Maven (3.8.2+)

This option is particularly interesting for developers. You can run the search service in dev mode that enables live coding using:
```
# change the directory
cd search-service
# run MOSAIC in dev mode
mvn quarkus:dev [-Dquarkus.http.port=<API_PORT>] \
      [-Dquarkus:args="OPTION"]
```

Option 4: Use Docker

Prerequisites:
- Install Docker

Another option to run the application is to build a Docker image and then launch the application in a Docker container by executing the following commands:
```
# create an image
docker build -t mosaic .
# start a container
docker run --rm -p 8008:8008 mosaic -p <YOUR_API_PORT>
```

Note that it is optional to pass `<YOUR_API_PORT>` as this already has a default value in the Dockerfile. Be aware that `<YOUR_API_PORT>` indicates the port which is used inside the container. If you want to change the port on your host machine, you need to modify the binding in the command above. For more information, see Docker reference.

A concrete example for starting the container without parameters is:
```
docker run --rm -p 8008:8008 mosaic
```

A concrete example for starting the container with parameters is:

```
docker run --rm -p 8008:8008 mosaic -p 8008
```

As soon as you build the Docker image, the CIFF file for each corresponding directory in “resources” is automatically imported to Lucene indexes in the image so you do not have to import the CIFF files to Lucene indexes manually beforehand. Note, that you have to update the image whenever there are changes in the source code.

Application Startup Process

During the startup process of the MOSAIC search service, several actions are performed:

- The path containing the index(es) is set
- The path containing the metadata is set
- The id-column, which is used to find the corresponding metadata in the Parquet file(s) for a Lucene document, is set
- The path of the config file is set
- The components and modules are loaded according to the config file
- The path of the database file is set
- The database is created (if the database does not exist yet) and, if the option `-n <num>` is passed when starting the service, the full plain text stored in the column `plain_text` is limited to `<num>` characters for each row.
- The OpenSearch document is updated according to the config file

CLI Options

To enable flexible and simple utilisation and development of MOSAIC, a handful of CLI options when starting the MOSAIC search service are supported:

`-l, --lucene-dir-path <dir>`	`path of directory containing the Lucene index(es) (default = lucene directory as in the repository)`
`-p, --parquet-dir-path <dir>`	`path of directory containing the Parquet file(s) (default = resources directory of this repository)`
`-i, --id-column <col>`	`column that contains the document identifiers (default = record_id)`
`-n, --num-characters <num>`	`number of characters selected from the plain text column to be stored in the associated DB table column`
`-d, --db-file-path <dir>`	`path of directory containing the database file (file is created when starting MOSAIC for the first time) (default = /tmp/mosaic_db)`

Windows

If you use Windows, a problem could occur if you use the default database file path. To solve this problem, you can use another file path to create the file directly in the repository. Thus, for example, you could use the option `-d mosaic_db`.

Additionally, it can happen that database file causes an exception after accessing it once because it cannot be opened anymore due to other processes using it. If this is the case, replace the creation of the database connection in the constructor of `DbConnection` with:

```
Properties readOnlyProperty = new Properties();
readOnlyProperty.setProperty("duckdb.read_only", "true")
conn = DriverManager.getConnection("jdbc:duckdb:" +
       CoreUtils.getDatabaseFilePath(), readOnlyProperty);
```

2) API

The MOSAIC search service provides an API to receive queries and return responses. Currently, four endpoints are accessible at <host>:<port> using a browser or any API testing platform (e.g., Postman).

Endpoints

/search

This endpoint searches in the index(es) using the provided REST query parameters and returns the results in JSON format.

Depending on the modules that are enabled, different query parameters can be used. For further information, please check the specification for each module. In general, there is no required query parameter, although in most cases `q` will be used for query terms.

An example GET request for this endpoint using the default port 8008 is:
`http://localhost:8008/search?q=europe`

Via the API, MOSAIC will return a response containing a list of search results where each result is composed of the fields of the enabled modules. If no index name is passed as parameter, MOSAIC searches in all available indexes and returns a list of results for each index.

An example response for the request above having only the `core` module enabled could yield the following:

```
{
  "results": [
    {
      "simplewiki": [
        {
          "id": "cfe49c84-2244-430e-83e3-fe3f30aae21e",
          "url":
"https://simple.wikipedia.org/wiki/Anthem_of_Europe",
          "title": "Wikipedia: Anthem of Europe",
          "textSnippet":
"https://simple.wikipedia.org/wiki/Anthem_of_Europe",
          "language": "spa",
          "warcDate": "2024-01-15T22:19:46Z",
          "wordCount": 11
        },
        ...
      ],
    },
    {
       "unis-graz": [
        { ... },
      ],
      ...
   },
   ...
  ]
}
```

/searchxml

Similar to the endpoint `/search`, this endpoint is used to search in the index(es) based on passed REST query parameters in the GET request. However, the results are returned in the format of the OpenSearch Protocol (XML).

The structure of the XML response is:

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
  <title>MOSAIC Search: {searchTerms}</title>
  <description>Search results for "{searchTerms}" at MOSAIC
               Search Service</description>
  <author>
    <name>OpenWebSearch.eu</name>
  </author>
  <opensearch:totalResults>1121</opensearch:totalResults>
  <opensearch:startIndex>1</opensearch:startIndex>
  <opensearch:itemsPerPage>20</opensearch:itemsPerPage>
  <opensearch:Query role="request" searchTerms="{searchTerms}"
                    startPage="1"/>
  <link rel="alternate"
       href="{baseUrl}/search?q={searchTerms}&pw=1&limit=20"
       type="application/json"/>
  <link rel="self"
        href="{baseUrl}/searchxml?q={searchTerms}&pw=1&limit=20"
        type="application/atom+xml"/>
  <link rel="next"
         href="{baseUrl}/searchxml?q={searchTerms}&pw=2&limit=20"
         type="application/atom+xml"/>
  <link rel="last"
        href="{baseUrl}/searchxml?q={searchTerms}&pw=56&limit=20"
        type="application/atom+xml"/>
  <link rel="search"
        type="application/opensearchdescription+xml"
        href="{baseUrl}/opensearch.xml"/>
  <item>
    ...
  </item>
  <item>
    ...
  </item>
  ...
</feed>
```

The node `item` represents one search result and contains the field specified in the enabled modules.

/index-info

The endpoint `/index-info` returns detailed information of the index(es) which are used in the current MOSAIC search service execution. The endpoint expects no REST query parameters and the response is in JSON format. For each present index, the information includes the name of the index, the number of indexed documents and a list of languages which appear in this index:

```
{
  "results": [
    {
      "simplewiki": {
        "documentCount": 285392,
        "languages": [
          "deu",
          "eng",
          "est",
          "fra",
          "ltz",
          "pol",
          "unknown",
          "zho"
        ]
      }
    },
    ...
  ]
}
```

For instance, this endpoint can be used to display the available indexes in the front-end.

/full-text

As described in CLI Options, the option `-n <num>` allows to import only a part of the full plain text of each document into the database. This is particularly beneficial for larger indexes to reduce the creation time and file size of the database. In order to get the full plain text of a web document, MOSAIC provides the endpoint `/full-text`. The endpoint expects the parameter `id`. Additionally, the parameter `column` can be used to specify the metadata column the passed web document ID should be matched (default: `record_id`). If you have multiple indexes and you already know the index name that contains the document you are processing, you can also pass the optional parameter `index`.

Depending on the host and port where the application is running, the format of the GET request for the endpoint `/full-text` with a response in JSON format is:

```
http://localhost:8008/full-
text?id=<id>&column=<column>&index=<indexName>
```

A concrete example for a simple GET request to retrieve the full plain text of a document with the ID `0f02f96c-a2da-49c2-9e6b-95e17d95cbf1` is:

```
http://localhost:8008/full-text?id=0f02f96c-a2da-49c2-9e6b-
95e17d95cbf1
```

The response format is in JSON and has the following structure:

```
{
  "id": <id>,
  "fullText": <fullText>
}
```

3) Index

Indexes are used to search for relevant information based on a given query. MOSAIC utilizes the index files created by the OpenWebSearch.eu pipeline.

Index Files

One index consists of one CIFF file and one or multiple associated Parquet files.

Parquet

Apache Parquet is a column-oriented file format to efficiently store and retrieve data. The OpenWebSearch.eu pipeline creates on or multiple Parquet files based on the crawled data. MOSAIC uses Parquet files to enrich a Lucene search result with metadata. Please refer to the schema of fixed columns to see the available columns in a Parquet file. Of course, you can modify existing columns or add new columns. For instance, you could generate a summary based on the given full plain text and add a new column in the Parquet file.

CIFF

A CIFF file is an inverted index exchange format. In the OpenWebSearch.eu pipeline, a CIFF file is created based on one or multiple Parquet files. MOSAIC uses the lucene-ciff application to import the inverted index in CIFF format to a Lucene index. MOSAIC uses the imported Lucene index to search for documents based on a given query.

Create an Index Locally

OpenWebSearch.eu provides the infrastructure for creating an index, based on a list of URLs, locally. To create an index locally, Docker and a scraping tool like `wget` is required. For more instruction details and explanations, please refer to the tutorial.

Step 1: Scraping a List of Webpages

```
# Create a directory that will be mounted in the Docker
  containers
mkdir -p data
wget --input-file urls.txt \
    --delete-after \
    --no-directories \
    --warc-file data/crawl
```

Step 2: Preprocessing

```
docker run \
    --rm \
    -v "$PWD/data":/data:Z \
opencode.it4i.eu:5050/openwebsearcheu-public/preprocessing-
pipeline \
    /data/crawl.warc.gz \
    /data/metadata.parquet.gz
```

Step 3: Indexing

```
docker run \
    --rm \
    -v "$PWD/data":/data:Z \
opencode.it4i.eu:5050/openwebsearcheu-public/spark-indexer \
    --description "CIFF description" \
    --input-format parquet \
    --output-format ciff \
    --id-col record_id \
    --content-col plain_text \
    /data/metadata.parquet.gz \
    /data/index/
```

Step 4: Consuming the Index

After step 3, you should have the following files:

- `data/crawl.warc.gz`
- `data/metadata.parquet.gz`
- `data/index/index.ciff.gz`

If you have MOSAIC installed already, you can simply move or copy the CIFF and Parquet file to a new directory “<index-name>” in the folder “resources”. When building MOSAIC, a new Lucene index with the name <index-name> is created an the inverted index is imported. After that, you can start MOSAIC and the new index can be used for searching web documents.

If you just want to serve the index without having MOSAIC installed, you can first import the CIFF file into a Lucene index:

```
mkdir -p data/serve/lucene
docker run \
    --rm \
    -v "$PWD/data":/data:Z \
    opencode.it4i.eu:5050/openwebsearcheu-public/mosaic/lucene-
ciff \
    /data/index/index.ciff.gz \
    /data/serve/lucene/demo-index
mkdir -p data/serve/metadata/demo-index
cp data/metadata.parquet.gz \
   data/serve/metadata/demo-index/metadata.parquet.gz
```

After that, you can use the Docker image of the MOSAIC search service to run the application and serve the index:

```
docker run \
    --rm \
    -v "$PWD/data":/data:Z \
    -p 8008:8008 \
    opencode.it4i.eu:5050/openwebsearcheu-public/mosaic/search-
service \
    --lucene-dir-path /data/serve/lucene/ \
    --parquet-dir-path /data/serve/metadata/
```

The application should be running on localhost:8008 now and you should be able to perform a search query (e.g., http://localhost:8008/search?q=europe).

4) Modules

MOSAIC uses multiple modules to handle REST query parameters and metadata. They enable to additionally filter search results and can append additional fields in the response.

Existing Modules

Core: This module is required. The core module is responsible for the basic search and the response. Please refer to the core module description for the list of REST query parameters and the response fields for both the JSON and XML format.

Shared: This module is required. It enables the interaction between the core module, which contains the web framework, and the additional metadata modules.

Geo: This module is optional. Please refer to the geo module description for the list of REST query parameters and the response fields for both the JSON and XML format.

Keywords: This module is optional. Please refer to the keywords module description for the list of REST query parameters and the response fields for both the JSON and XML format.

Creating a new Module

MOSAIC allows developers to simply add new modules by themselves. Since the framework is based on Maven modules, these are the steps to incorporate a new metadata module.

Step 1: Create new Maven Module

Create a new Maven module with <MODULE_NAME> as name (replace <MODULE_NAME> with the actual name of the module) and search-service as parent module. By default, the newly created Maven module should have the same folder structure as the existing modules. You can either use an IDE to create a new Maven module or use the following command in the directory “search-service”:

```
mvn archetype:generate -DgroupId=eu.ows.mosaic
                       -DartifactId=<MODULE_NAME>
                       -DinteractiveMode=false
```

Step 2: Add Dependency in new Module

Add a dependency for the `shared` module in the file `pom.xml` of the newly created module:

```
<dependencies>
  <dependency>
    <groupId>eu.ows.mosaic</groupId>
    <artifactId>shared</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

Step 3: Add Dependency in Core Module

```
<dependency>
```

```
  <groupId>eu.ows.mosaic</groupId>
  <artifactId>MODULE_NAME</artifactId>
  <version>1.0-SNAPSHOT</version>
  <optional>true</optional>
</dependency>
```

Step 4: Register Module in Parent Module

If not done automatically, register your new module as such in the file `pom.xml` in the parent `search-service` by appending it to the existing modules:
`<module>MODULE_NAME</module>`

Step 5: Create Java File

Create a new Java file in
`search-service/<MODULE_NAME>/src/main/java/eu/ows/mosaic/`
that contains a class which extends `MetadataModule`. For example, name this Java file and class `<MODULE_NAME>Metadata`.

Step 6: Override Methods

Override methods in the newly created class as you like. Particularly, override
- `getMetadataColumns()` and
- `getFilterColumns()`

which are responsible for retrieving additional metadata columns and defining metadata filter columns respectively. For more information about the methods, take a look at the abstract class MetadataModule.

Example: Using the column “url_suffix”

In this example, we will add a new metadata column `url_suffix` that will be part of the response and can also be used for filtering. First, we override the method `getMetadataColumns()` (please note that we should also override the remaining methods if the column is not a primitive) so that the column `url_suffix` will be serialized in the response:

```
@Override
public Set<String> getFilterColumns() {
    return Set.of("keyword");
}
```

In order to use this column for filtering, we override the method `getFilterColumns()` (please note that you should also override the remaining methods if the column is not a primitive) so that the column `url_suffix` is considered when parsing the query parameters and adding them to the SQL filter:

```
@Override
public Set<String> getMetadataColumns() {
    return Set.of("text");
}
```

Step 7: Enable new Module in Config

Add an entry in `search-service/core/src/main/resources/config.json` in the `plugins` object to enable the module for MOSAIC.

If you build and start the application, MOSAIC should now load and use the newly created module.

5) Front-end

The repository contains a simple front-end, providing basic functionality to use the modules. Feel free to modify the existing front-end or create a new front-end utilizing the REST API of the MOSAIC search service.