Documentation

Table of contents

About

Metadata-Hub

This application is developed in the course of the AMOS project.

Motivation

Big-Data environments often comprise large amounts of data that have been integrated from various sources. These can only be turned into valuable information through the use of metadata, which is significantly more leightweight, allowing for faster accessing and handling when compared to the actual data. Therefore, the aim of the Metadata-Hub is to provide a platform-independent retrieval, storage, and query mechanism for metadata on large file systems. This will allow end-user applications to obtain resilient and meaningful data as basis for complex data analyses in order to mine valubale information for the application-specific context.

Goals

Our mission is to create a first **prototype** of the Metadata-Hub, an independent piece of software that intelligently crawls large file systems in order to gain and store metadata about the files it finds. An intelligent algorithm continuously traverses the file system and collects interesting metadata. This metadata is stored in a designated metadata store, thereby generating an easy-to-access and persistent index of the whole file system. Finally, existing end-user applications will be able to query and consume the collected metadata.

Wiki

This wiki is the official documentation of the Metadata-Hub project. All included images are preview images that are linked to the larger original image. Simply click on the image to view it in higher resolution.

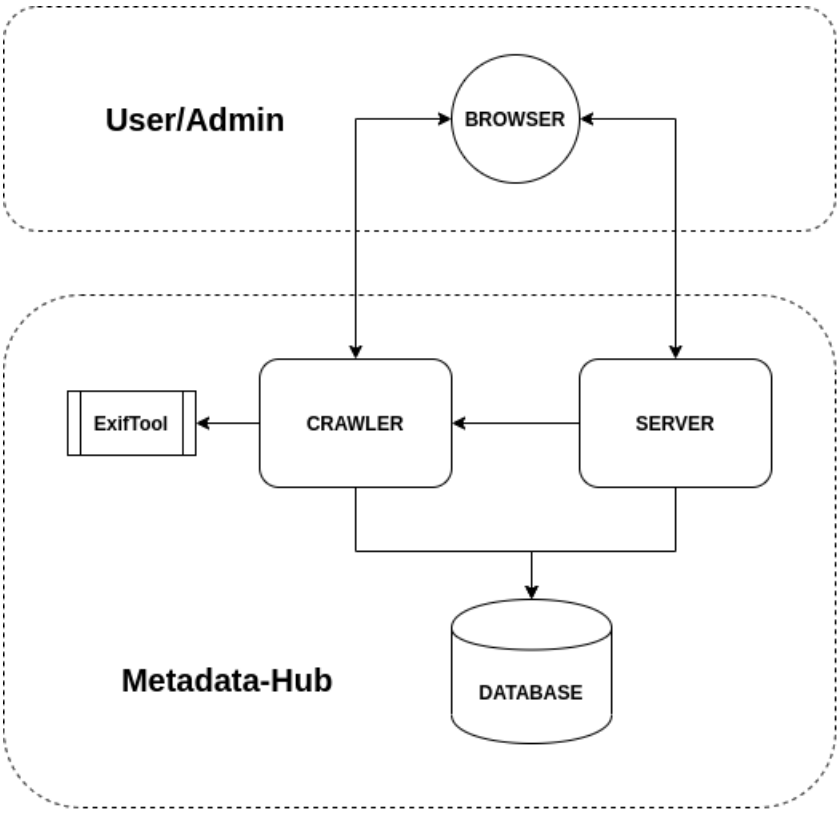
Architecture

This chapter provides a short overview about the conceptual architecture of the Metadata-Hub application. It aims to give a high-level understanding of the structure of the application rather than a detailled description of its components. For a more detailed documentiation, please refer to the Documentation chapter.

The Metadata-Hub application consists of three major components that are listed below.

- **Crawler**
  This component implements the tree walk algorithm which crawls the target directories/filesystems. The tree walk can be manually configured, e.g. input directories or rough CPU restrictions. It extracts metadata of the traversed files with the ExifTool and stores this data in the database. Furthermore, it stores state about its execution(s) such that the tree walk can be initialized with already traced data or being paused/continued.
- **Server**
  This component provides an interface for queries about the metadata. It uses GraphQL as the query language. Furthermore, it provides a web interface for the user to create queries without too detailed technical knowledge.
- **Database**
  This component is responsible for persisting the metadata. The used database is PostgreSQL. Furthermore, it is used to store state about executions of the tree walk for the controlling mechanism.

The interaction of these components is depicted in the following graphic and explained below.



Both the server and the crawler communicate with the database. Both the server and the crawler are accessible via the user's web browser. The database itself isn't required to be accessed directly by the user.

Documentation

This chapter provides a more detailed overview about the single components. It aims to give a deeper understanding of how they work and what functionality they provide. Though, this is no source code documentation. Therefore, please refer to the GitHub repository.

Crawler

Tree Walk Interface

This section will shortly explain how to access the tree walk interface , as well as the different configuration options provided.

The interface can be accessed in a web browser, by using the address and port specified while starting the docker container. The following gif shows an example of the interface:

- Interface access example

The different options have the following meaning:

- **Input directories**
  This field expects a path to a directory and a corresponding boolean separated by a comma. This value determines, if the tree walk is supposed to be executed recursively (If the value is set to true, all subdirectories will be scanned). You can also enter multiple pairs, by separating them with a semicolon.

  An example input could look like this:

  `/home/metadata-hub, True; /home/user, false`

- **Output directory**
  This field is deprecated and is used for debugging purposes. Enter a directory path in the project folder.

- **Trace file**
  This field is used to input a directory path to a trace file. This file will be used to store data on the execution. Visited and completed nodes in the tree walks traversal path will be saved, so future tree walk runs can exclude them. This can potentially safe time, if you do not want to analyze directories twice.

  An example input could look like this:

  `/home/metadata-hub/crawler/trace.log`

- **ExifTool**
  This field is used to choose between the linux and windows executables of the exiftool. Pick the operating system you are currently working on.

- **Clear trace data before start**
  This field is used to determine, whether the trace.log should be used or not. If yes is picked, previously visited node of the directory tree will be skipped. If no is picked, the tree walk will traverse the entire directory tree.

- **Power level**
  This field is used to determine how many system resources should be reserved for the tree walk execution. 1 provides the system with the minimum amount, 4 with the maximum,

- **Work package size**
  This field represents the value of the desired work packages the crawler will scan. The algorithm attempts to evenly split the files of all directories into this size.

- **Update current execution**
  This field determines, if the submission that is about to be sent to the crawler will replace the old one. If no is selected the crawler will not accept a new submission and prompt the user to wait.

API

This section will shortly explain the API of the crawler. All these endpoints support `GET` and `POST` requests except */info* that only supports `GET` requests.

- **/config**
  Create a configuration for the tree walk and start it. This endpoint provides the interface described above. After submitting the configuration, a success or error page is shown.

- **/start?config={CONFIG}&update=[True]**
  Start the tree walk. This is the proper way to start the tree walk in a automated way.

- `CONFIG` (*required*) This is the configuration of the execution. It can either be a filepath pointing to a valid configuration file or a valid JSON configuration.
- `update` (*optional*) By providing *update=True*, a possible running execution will be stopped and the new one will be started. Without providing *update* or with *update=False*, the request will be ignored if a execution is running/paused.

- **/pause**
  Pause a currently running execution of the tree walk. The request will be ignored when the tree walk is currently not running. The execution can be continued or stopped later on
- **/continue**
  Continue a paused execution of the tree walk. The request will be ignored if the tree walk is not paused.
- **/stop**
  Stop a running or paused execution of the tree walk. The request will be ignored if the tree walk is neither paused nor running.
- **/info**
  Retrieve information about the current status of the tree walk. Useful to check the status and progress of a possible running execution.
- **/shutdown**
  Shutdown the crawler completely. This will force a possible running execution of the tree walk to end and exit the crawler process.

Database

- Include a more detailed information about the server component
- Do not reference code, but focus on the major functionality
- For example a further explanation of the API and how to use it

Server

- Include a more detailed information about the server component
- Do not reference code, but focus on the major functionality
- For example a further explanation of the API and how to use it

Installation

This chapter will show you how to install the software requirements and the Metadata-Hub application. It is important to mention that this is **no** application that should be used in a production environment because of predefined user/password settings that cannot be changed.

Requirements

The application is published as a Docker image. Thus, it requires your system to have the Docker Engine installed. Therefore, please refer to the official installation instructions of Docker at https://docs.docker.com/get-docker. Please make sure to install at least version *19.03* and check the installation before continuing.

Installation

The image is published using the DockerHub registry at amosproject2/metadatahub. There are two versions of the application you can use:

- `latest` The latest stable version, usually updated once a week
- `dev` The currently developed version, might be unstable

The `latest` version is the recommended one to use, thus

```
$ docker pull amosproject2/metadatahub
```

will pull the image tagged with `latest` by default.

Configuration

The image is build with a default configuration that specifies some mandatory settings that cannot be changed, such as the default database user and port settings inside the container (see more at Usage).

The storage of the database will kept inside the container by default. Indeed, it can be useful to store this data on the host system to access it later on. Therefore, simply create a Docker volume that is used to store the content of the database.

```
$ docker volume create --name metadatahub-database -d local
```

It is also possible to use multiple volumes to have separate 'databases' for the various container.

Usage

After setting everything up, you are ready to run the image. Therefore, use the following template.

```
docker run \
    -p {HOST_SERVER_PORT}:8080 \
    -p {HOST_CRAWLER_PORT}:9000 \
    -v {DATA}:/filesystem \
    -v metadatahub-database:/var/lib/postgresql/12/main \
    amosproject2/metadatahub
```

The following values have to be specified by the user:

- `HOST_SERVER_PORT`
  The port that publishes the *server* with the graphical user interface for data querying on the *host* machine.
- `HOST_CRAWLER_PORT`
  The port that publishes the *crawler* for starting/stopping/etc. the crawling mechanism on the *host* machine.
- `DATA`
  The directory/filesystem you want to crawl.

The ports *8080* and *9000* must not change thus they are required for internal communication. For example, you can run the image with the following command.

```
docker run \
    -p 9999:8080 \
    -p 9998:9000 \
    -v /home/john/data:/filesystem
    -v metadatahub-database:/var/lib/postgresql/12/main \
    amosproject2/metadatahub
```

You should be able to access both *localhost:9999* and *localhost:9998* for the corresponding services.

If you encounter any errors, please refer to the FAQ section.

Usage

Each entry is linked to a GIF that shows the specified functionality.

Crawler

- How to start the crawler with the web interface
- How to start the crawler with the REST interface
- How to get status information
- How to stop the crawler
- How to pause the crawler
- How to continue a paused execution of the crawler

FAQ

I have problems with docker permissions using Linux.

It is most likely your user does not belong to the `docker` group. Please have a look at these instructions.

I have problems with a running container and want to inspect them.

You can start a shell session in your running container:

```
docker exec -it {container-id} /bin/bash
```

This will start a *bash* session inside the container with the ID `container-id`. Furthermore, you can inspect the log files at `/metadatahub/server.log` and `/metadatahub/crawler.log`.