Connectors Features

Incremental Crawling

By default, the Connector Framework allows connectors to handle incremental crawling using the snapshot NoSQL database. This snapshot contains an entry for each item discovered by the last crawl with an id, a subset of the metadata, a signature and a crawl id. On the following incremental the action is determined using the following criteria:

- **Add:** there is no entry on the snapshot with the given item id.
- **Update:** there's an entry on the snapshot with the given id, but different signature.
- **Delete:** the crawl id on each item that is visited during an incremental crawl is updated to reflect the new crawl id. In the end of each incremental crawl, all items that didn't get the new crawl id are sent to the delete process.

Hierarchy

The Connector Framework is able to create the hierarchical structure of a seed based on how items are being discovered. This feature depends on the specific type of connector in use. To know if a connector supports hierarchy generation, check its documentation.

Fetch Content

Connectors can fetch the content of a document and set the content stream on the job so that it can be processed on a later stage. Each connector will allow content fetching on specific types of items, check the documentation to see which ones are allowed.

Text Extraction

If text extraction is enabled, the content stream opened during the fetch stage is sent to Apache Tika to extract the text content of the document. Take into account that the text extraction will consume the content stream, thus making it unavailable for other components to work with it.

Non-Text Documents

The connectors that allow content fetching can be configured so that certain document types are not processed on the text extraction stage, leaving the data stream open so it can be processed on a Workflow stage. Non-Text documents can be identified using a comma separated list of extensions or a file containing a list of regex patterns to match the documents (one regex pattern per line).

Document Level Security

To support document level security, the Connector Framework provides support for access control lists (ACLs). At minimum an ACL entry will contain the following information:

- **fullname:** could be a combination of domain and entity name, or any value that identifies an entity in the given repository.
- **access:** either allow or deny.
- **entity:** could be user or group.

Each connector can add other attributes to the ACL entries depending on the repository needs.

Identity Crawling

For connectors that support document level security, identity crawling is available. This type of crawl will fetch entities (users, groups and memberships), pass those entities to a Workflow to allow for custom stages to be added and finally stores them in the identity cache NoSQL database.

Identity crawling doesn't support incremental crawls, which means that every crawl will create a new entry in the identity cache database. Old entries are deleted after a given number of crawls.

Is important to note that the memberships extracted by the Identity crawls may not be flattened, which means that it may only show direct relationships between users and groups. To expand those relationships and to use multiple seeds for group expansion, use the Group Expansion Connector.

Group Expansion

As mentioned on the previous section, group expansion is done using the Group Expansion Connector. This connector will retrieve all entities stored in the identity cache for all seeds and will create an expanded version of each of those entities. It is important to notice that those expanded entities are not stored in the identity cache and that the user can add a publisher on the Workflow to store the expanded identities. For more information on how this works, please check the Group Expansion Connector documentation.

Real Life Example

As we said earlier there is no Group Expansion endpoint as we had in Aspire 2, 3 and 4. The way we intended this to work in Aspire 5 is by having an index with the expanded groups which can be queried directly by the Elasticsearch UI to obtain the direct and indirect groups of any given user.

How this works in Aspire 5:

- The user/group data is retrieved from the original sources by executing ***Identity Crawls***
- All the data regarding direct memberships is stored in the **Identity Cache**
- Groups are not guaranteed to be already expanded, and certainly will not be merged with identities from other sources.
- Some identity crawlers such as ***Azure AD*** do store the expanded list of groups per user.
- Each entry in the cache has a timestamp identifying the crawl on which it was found so that rollbacks can be done more easily if needed
- Once all identities have been extracted, you might then merge and expand them using the **Group Expansion connector**
- This connector connects to the **Identity Cache**, get the identities from the configured sources (using their latest version), and expands their group list recursively.
- Identities from multiple sources are joined together by id
- Each identity is then sent to the Workflow to be processed as a regular item
- Here you can index the users in a separate index using the format that suits you the most.

Here is a more concrete example using AzureAD + SharePoint Online:

You have a SP site called **SiteA**, which has a site local group (exists only in SharePoint Online within siteA) called: "**SiteA Owners**", as a member it has an Azure AD group called "**Engineering**"

In AzureAD you might have this:

- Engineering
    - Members:
        - user@company.comblocked URL

The right order to execute this in Aspire 5 would be:

1. Execute the Azure AD identity crawler

This will generate these entries in the identity cache:

- Engineering
- user@company.comblocked URL
    - Memberof:
        - Engineering

2. Execute the SP Online SiteA indentity crawl

This will generate this entry in the identity cache:

- SiteA Owners
- Engineering
    - Memberof:
        - "SiteA Owners"

3. Now you execute the Group Expander including the data from both seeds (Azure AD + SP Online)

This will generate three jobs to be indexed in the index of your preference:

- Engineering
    - Memberof:
        - SiteA owners

- SiteA Owners
- user@company
    - Memberof:
        - Engineering
        - SiteA Owners

Failed Documents Processing

The Connector Framework is able to retry any documents that fail during a crawl. These failures include any exceptions raised during a Workflow stage or at index time. If enabled, the connector will retry all errored documents or any documents where the exception matches one the configured regex patterns. For failed documents processing, the user can configure the following:

- Set a maximum number of in-crawl retries, which will be retries for each failed document on a single crawl.
- Set a maximum number of crawl retries, which means how many consecutive crawls should the failed documents be retried.
- Enable the deletion of failed documents from the snapshot after all in-crawl retries have failed. If a connector uses snapshots for incremental crawling, enabling this will override the max crawl retries option, since the document will be discovered on every incremental crawl as long as it still exists in the repository and it keeps failing.
- Provide a list of regex patterns that will be matched against the exception thrown by the failed document so that only certain exceptions are retried.

This feature consists of two phases: pre-reprocess and post-reprocess.

Pre-Reprocess Phase

This phase is only executed at the beginning of each incremental crawl. The failed documents will be retried once in this phase, the in-crawl retries will be reset to 0 and the crawl retries will be increased by 1.

Post-Reprocess Phase

This phase is executed after each crawl (full and incremental). The failed documents will be retried as many times as configured with the maximum in-crawl retries.