NoSQL Connector Framework Implementation

This section describes in details each of the components involved in the Connector Framework.

What's next?

- NoSQL Connector Framework Security Implementation
- MongoDB Collections Description
- Write Your Own Connector from Scratch

Component's Bundle

The main component's bundle jar component the framework uses is **aspire-connector-framework,** this bundle contains all the Stages, components and also provides interfaces for the specific connector implementations.

- CrawlController
- ScanQueueLoader
- ProcessQueueLoader
- ProcessDeletes
- CrawlEnd
- ScanReleaseController
- Scan
- FlagContainer
- IncludeExclude
- CheckSnapshot
- GenerateHierarchy
- EnqueueScan
- AddUpdateSnapshot
- EnqueueProcess
- ProcessReleaseController
- ProcessCrawlRoot
- PopulateOrDelete
- FetchUrl
- EnqueueScan
- MarkProcessComplete
- MarkScanComplete

Connector AppBundle

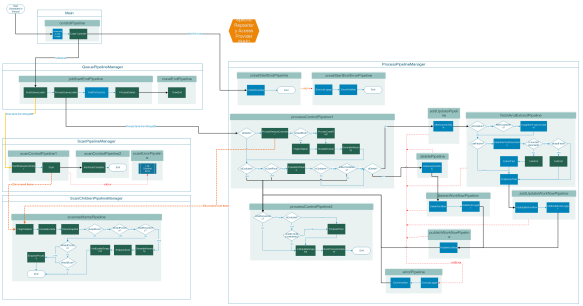
In Aspire every component needs to be referenced from an AppBundle or application.xml file, which describes the job execution flow. For the Connector Framework we have one common AppBundle called **app-rap-connector.**

This AppBundle is automatically loaded when Aspire detects it needs to load a connector. It contains all the PipelineManagers, Pipelines and references to the Connector Framework Components and Stages from the **aspire-connector-framework** bundle.

PipelineManagers and Pipelines

Main (PipelineManager)

- controlPipeline



QueuePipelineManager

- jobStartEndPipeline
- crawlEndPipeline

ScanPipelineManager

- scanControlPipeline1
- scanControlPipeline2
- scanErrorPipeline

ScanChildrenPipelineManager

- scannedItemsPipeline

ProcessPipelineManager

- crawlStartEndPipeline
- crawlStartEndErrorPipeline
- processControlPipeline1
- processControlPipeline2

- addUpdatePipeline
- fetchAndExtractPipeline
- addUpdateWorkflowPipeline
- publishWorkflowPipeline
- deletePipeline
- deleteWorkflowPipeline
- errorPipeline

Components

• CrawlController

The CrawlController is the main entry point for incoming crawl start signals, also it controls the ConnectionPool and manages the NoSQLConnections to Mongo used by the rest of the components. It also handles the distributed crawl start and synchronizes the crawl status with Mongo so all Aspire servers have the same.

• ScanQueueLoader

It is a component configured to claim items from the ***scanQueue*** collection in Mongo, marks each item as in-progress **"P"** in Mongo (so no other server claims the same item for scanning) and enqueues them jobs into the **ScanPipelineManager**. All the items claimed by this component are containers that should be scanned.

• ProcessQueueLoader

It is a component configured to claim items from the ***processQueue*** collection in Mongo, marks each item as in-progress **"P"** in Mongo (so no other server claims the same item for processing) and enqueues them as jobs into the **ProcessPipelineManager**. All the items claimed by this components are items that should be considered for workflow processing.

• ProcessDeletes

This stage gets executed at the end of the crawl, it queries MongoDB for the uncrawled items according to the ***snapshot*** collection, gets a list of items to delete, and enqueues them to the ***processQueue*** collection with the action as **"delete".** Then those items will be picked up by the **ProcessQueue Loader** component in order to process them.

• CrawlEnd

Marks the crawl as complete **"C"** in the ***status*** collection.

• ScanReleaseController

Filters the items before they get scanned, so they get returned to the ***scanQueue*** collection if the crawl is paused, so it can be scanned later.

• Scan

Calls the specific connector implementation (*RepositoryAccessProvider* or RAP) to scan an item and get back all its children, then they get enqueued to the **ScanChildrenPipelineManager** as Aspire jobs.

• FlagContainer

Calls the specific connector implementation (RAP) to mark an item as container.

• IncludeExclude

Calls the specific connector implementation (RAP) of the include/exclude filters

• CheckSnapshot

Compares the current item against the ***snapshot*** collection to determine if this should be added or updated in incremental crawls.

• GenerateHierarchy

Gets the hierarchy of the current item from the ***hierarchy*** collection and generates the structure for it in the metadata. It also stores information about each container that passes through this so later on, its children can get its hierarchy.

• EnqueueScan

Enqueues the current item into the ***scanQueue*** collection with status as available **"A"**.

• AddUpdateSnapshot

Add the current item to the ***snapshot*** collection or update it in case it already exists.

- ## EnqueueProcess

Enqueues the current item into the ***processQueue*** collection with status as **"A"**.

- ## ProcessReleaseController

Filters the items before they get processed, so they get returned to the ***processQueue*** collection if the crawl is paused, so it can be processed later.

- ## ProcessCrawlRoot

Calls the specific connector implementation (RAP) with the **"crawlRoot"** item and enqueues any extra root items added by the connector to the **ScanChildrenPipelineManager** as an Aspire job.

- ## PopulateOrDelete

Calls the specific connector implementation (RAP) to populate the current item's metadata, if the item is marked as a "delete" then, it calls the corresponding registered methods in the RAP.

- ## FetchUrl

If the specific connector implementation does not define a custom Fetcher, it uses the **FetchUrl** stage, other wise it calls the connector specific fetcher to fetch the content from the repository.

- ## EnqueueScan

Enqueues the current item into the ***scanQueue*** collection with status as **"A"**.

- ## MarkProcessComplete

Marks the current item as completed **"C"** in the ***processQueue*** collection.

- ## MarkScanComplete

Marks the current item as completed **"C"** in the ***scanQueue*** collection.