BSH AIM RAG - Scheduler

The **BSH AIM RAG Scheduler**, developed by the **AIX team** (formerly CDL), is a scalable and efficient solution for managing the synchronization of files stored in Amazon S3 with **Qdrant**, a high-performance vector database. It supports both **initial indexing** and **incremental updates** by detecting changes in stored files. That means only newly **added**, **modified**, or **deleted** documents get processed — saving time and resources.

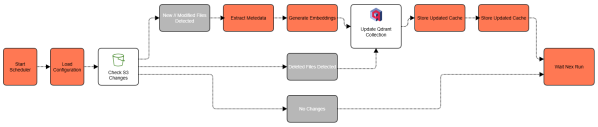
Key Features

Change Detection	Metadata Extraction	Error Handling	Configurability	User-Friendly Interface
Uses an S3 housed cache to track file hashes, only processing new or modified files	Automatically extracts metadata from file paths	Includes comprehensive logging and error handling	All important parameters can be set via command-line arguments	A cool UI allowing embedding and indexing of data housed in S3 in clicks.

Modular Architecture

The **BSH AIM RAG Scheduler** follows a **modular architecture**, where each module is responsible for a specific function:

Module	Responsibility	How It Works	Data Flow	Benefits

Config	Loads configurations from environment variables	• Loading environment variables
		• Managing AWS, S3, and Qdrant configurations
		• Handling collection-specific settings through CollectionConfig dataclass
			• main.py reads config initializes sync manager
			• sync_manager coordinates operations:
			  ◦ S3_manager reads files
			  ◦ metadata_extractor processes paths
			  ◦ embeddings creates vectors
			  ◦ vector_store stores in Qdrant
			  ◦ cache_manager tracks changes
			• Process repeats for incremental updates
			
			Flow Chart
			 
				This modular architecture allows for:
				• Easy addition of new embedding providers
				• Simple collection configuration changes
				• Efficient incremental updates
				• Reliable deletion handling
				• Scalable document processing

S3Manager	Manages all interactions with Amazon S3	• Listing files in collection-specific folders • Reading JSON documents from S3 • Streaming documents for processing • Using folder prefixes to scope operations to specific collections		
MetadataExtractor	Extracts key metadata from file paths	• Parsing S3 keys to extract country information • Mapping folder structures to metadata attributes • Providing consistent metadata format for document indexing		
Embedding Provider	Interfaces with multiple embedding models	• Providing unified interface for different embedding providers • Supporting HuggingFace, Azure OpenAI, and DashScope models • Converting text to vector embeddings • Handling model-specific configurations		
VectorStore	Manages interactions with Qdrant, including indexing	• Creating and managing collections • Converting documents to vector points • Handling upserts and deletions • Managing document IDs and metadata • Providing collection information		
CacheManager	Maintains local file cache for efficient change detection	• Storing document hashes in S3 • Detecting new, modified, and deleted files • Managing cache files per collection • Providing efficient change detection		

SyncManager	Orchestrates the entire sync process	• Coordinating between S3 and Qdrant • Managing initial and incremental indexing • Handling file deletions • Processing documents in batches • Maintaining sync state		
Main	Entry point and CLI interface	• Parsing command-line arguments • Loading collection configurations • Initializing components • Managing sync operations • Providing flexible execution options (initial/incremental /specific collection)		

Collection Configuration

Each collection (e.g., "**hr-data-china**") is configured in **`collections_config.json`**, it:

- Creates or connects to the Qdrant collection with the same name
- Maps to the matching S3 folder
- Performs initial indexing if needed
- Identifies new, modified, and deleted files
- Updates the Qdrant collection accordingly

```
            {
    "collections": [
        {
            "name": "hr-data-china",
            "embedding_model": "text-embedding-v3",
            "embedding_type": "alibaba"
        },
         {
            "name": "hr-data-generic",
            "embedding_model": "text-embedding-ada-002",
            "embedding_type": "openai"
        },
    ]
}
```

Supported Embedding Models

The scheduler supports multiple embedding models from different providers:

Provider		Embedding Models	Dimension	Supported Languages	Supported	Pricing (1M Tokens)	For More
blocked URL	Hugging Face	all-MiniLM-L6-v2	384	• en	✅	free	• SentenceTransformers Documentation • Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
		BAAI/bge-m3	1024	• en • zh	✅	free	• BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation
blocked URL	Azure Open AI	text-embedding-ada-002	1536	• en • zh • ..	✅	$0.10	• Vector embedding | OpenAI
		text-embedding-3-large			❌	$0.13	
		text-embedding-3-small			❌	$0.02	
blocked URL	Alibaba	text-embedding-v3	1024	• en • zh	✅	Free for a limited quota. If the throttling limit is exceeded, your API request fails due to throttling. You must wait for a period of time until the throttling conditions are met before you can call the API again.	

text-embedding-v2			✗
text-embedding-v1			✗

Each provider implements an **`EmbeddingProvider`** interface to ensure consistency across different models. Additional models can be added by implementing the following class:

```
import numpy as np
from abc import ABC, abstractmethod

class EmbeddingProvider(ABC):
    @abstractmethod
    def encode(self, text: str) -> np.ndarray:
        pass

    @abstractmethod
    def get_dimension(self) -> int:
        pass
```

Getting Started

Sync Commands

This scheduler or indexer tool synchronizes data between Amazon S3 and Qdrant vector database. Below is a comprehensive list of all available commands and their functions.

Command	Format	Description	Example
`--config`	`--config PATH`	Specifies the path to the collection configuration JSON file that defines S3 buckets and Qdrant collections to synchronize.	`python main.py --config configs/collections.json`
`--initial-only`	`--initial-only`	Performs only the initial indexing of data from S3 to Qdrant and then exits without monitoring for changes. Useful for first-time setup.	`python main.py --config configs/collections.json --initial-only`
`--incremental-only`	`--incremental-only`	Skips the initial full indexing and only processes files that have changed since the last synchronization. Useful for update operations.	`python main.py --config configs/collections.json --incremental-only`
`--collection`	`--collection NAME`	Removes records from Qdrant when corresponding files are deleted from S3. Can be combined with other commands.	`python main.py --config configs/collections.json --collection documents`

`--remove-deleted`	`--remove-deleted`	Removes records from Qdrant when corresponding files are deleted from S3. Can be combined with other commands.	`python main.py --config configs/collections.json --incremental-only --remove-deleted`

Command Combination

You can combine multiple commands for more specific synchronization operations:

Combination	Description	Example
`--config --collection --initial-only`	Perform initial indexing for a specific collection only	`python main.py --config configs/collections.json --collection aix_cdl_data --initial-only`
`--config --incremental-only --remove-deleted`	Process only changes and remove deleted files across all collections	`python main.py --config configs/collections.json --incremental-only --remove-deleted`
`--config --collection --incremental-only --remove-deleted`	Process only changes and remove deleted files for a specific collection	`python main.py --config configs/collections.json --collection customer_data --incremental-only --remove-deleted`
`--config configs/collections.json`	• Perform full synchronization for all collections • This initiates the sync process, handles incremental updates, and processes deletions	`python main.py --config configs/collections.json`