Easy RAG: the easiest way to get started with Retrieval Augmented Generation

The Quarkus LangChain4j project provides a separate extension named `quarkus-langchain4j-easy-rag` which provides an extremely easy way to get a RAG pipeline up and running. After adding this extension to your application, all that is needed to ingest documents into an embedding store is to add a dependency that provides an embedding model, and provide a single configuration property, `quarkus.langchain4j.easy-rag.path`, which denotes a path in the local filesystem where your documents are stored. On startup, Quarkus will automatically scan all files in the directory and ingest them into an in-memory embedding store.

Apache Tika, a library for parsing various file formats, is used under the hood, so your documents can be in any of its supported formats (plain text, PDF, DOCX, HTML, etc), including images with text, which will be parsed using OCR (OCR requires to have the Tesseract library installed in your system - see https://cwiki.apache.org/confluence/display/TIKA/TikaOCR).

You also don't even need to add a persistent embedding store - the `quarkus-langchain4j-easy-rag` extension will automatically register a simple in-memory store for you if no other store is detected. You also don't have to provide an implementation of `RetrievalAugmentor`, a basic default instance will be generated for you and wired to the in-memory store.

WARNING

An in-memory embedding store is OK for very simple use cases and getting started quickly, but is extremely limited in terms of scalability, and the data in it is not stored persistently. When using Easy RAG, it is still possible to use a persistent embedding store, such as Redis, by adding the relevant extension (like `quarkus-langchain4j-redis`) to your application.

You have to add an extension that provides an embedding model. For that, you can choose from the plethora of extensions like `quarkus-langchain4j-openai`, `quarkus-langchain4j-ollama`, or import an in-process embedding model - these have the advantage of not having to send data over the wire.

NOTE

If you add two or more artifacts that provide embedding models, Quarkus will ask you to choose one of them using the `quarkus.langchain4j.embedding-model.provider` property.

Getting started with a ready-to-use example

To see Easy RAG in action, use the project `samples/chatbot-easy-rag` in the quarkus-langchain4j repository. Simply clone the repository, navigate to `samples/chatbot-easy-rag`, declare the `QUARKUS_LANGCHAIN4J_OPENAI_API_KEY` environment variable containing your OpenAI API key and then start the project using `mvn quarkus:dev`. When the application starts, navigate to `localhost:8080` and ask any questions that the bot can answer (look into the files in `samples/chatbot-easy-rag/src/main/resources/documents` to see what documents the bot has ingested). For example, ask:

```
What are the benefits of a standard savings account?
```

NOTE

This application contains an HTML+JavaScript based UI. But with Quarkus dev mode, it is possible to try out RAG even without having to write a frontend. If the application is in dev mode, simply open the Dev UI (http://localhost:8080/q/dev-ui) and click the 'Chat' button inside the LangChain4j card. A built-in UI for chatting will appear, and it will make use of the RAG pipeline created by the Easy RAG extension. More information about Dev UI features is on the Dev UI page.

To control the parameters of the document splitter that ingests documents at startup, use the following properties:

- `quarkus.langchain4j.easy-rag.max-segment-size`: The maximum length of a single document, in tokens. Default is 300.
- `quarkus.langchain4j.easy-rag.max-overlap-size`: The overlap between adjacent documents. Default is 30.

To control the number of retrieved documents, use `quarkus.langchain4j.easy-rag.max-results`. The default is 5.

To control the path matcher denoting which files to ingest, use `quarkus.langchain4j.easy-rag.path-matcher`. The default is `glob:**`, meaning all files recursively.

For finer-grained control of the Apache Tika parsers (for example, to turn off OCR capabilities), you can use a regular XML config file recognized by Tika (see Tika documentation), and specify `-Dtika.config` to point at the file.