Docs

Redis Document Store for Retrieval Augmented Generation (RAG)

When implementing Retrieval Augmented Generation (RAG), a capable document store is necessary. This guide will explain how to leverage a Redis database as the document store.

Leveraging the Redis Document Store

To utilize the Redis document store, you'll need to include the following dependency:

```
<dependency>
    <groupId>io.quarkiverse.langchain4j</groupId>
    <artifactId>quarkus-langchain4j-redis</artifactId>
    <version>0.14.1</version>
</dependency>
```

This extension relies on the Quarkus Redis client. Ensure the default Redis datasource is configured appropriately. For detailed guidance, refer to the Quarkus Redis Quickstart and the Quarkus Redis Reference.

IMPORTANT

The Redis document store's functionality is built on the Redis JSON and Redis Search modules. Ensure these modules are installed, or consider using the Redis Stack. When the `quarkus-langchain4j-redis` extension is present, the default image used for Redis is `redis-stack:latest` but this can be changed by setting `quarkus.redis.devservices.image-name=someotherimage` in your `application.properties` file.

IMPORTANT

The Redis document store requires the dimension of the vector to be set. Add the `quarkus.langchain4j.redis.dimension` property to your `application.properties` file and set it to the dimension of the vector. The dimension depends on the embedding model you use. For example, `AllMiniLmL6V2QuantizedEmbeddingModel` produces vectors of dimension 384. OpenAI's `text-embedding-ada-002` produces vectors of dimension 1536.

Upon installing the extension, you can utilize the Redis document store using the following code:

```
package io.quarkiverse.langchain4j.samples;
```

```
import static dev.langchain4j.data.document.splitter.DocumentSplitters.recursive;

import java.util.List;

import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;

import dev.langchain4j.data.document.Document;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import io.quarkiverse.langchain4j.redis.RedisEmbeddingStore;

@ApplicationScoped
public class IngestorExampleWithRedis {

    /**
     * The embedding store (the database).
     * The bean is provided by the quarkus-langchain4j-redis extension.
     */
    @Inject
    RedisEmbeddingStore store;

    /**
     * The embedding model (how is computed the vector of a document).
     * The bean is provided by the LLM (like openai) extension.
     */
    @Inject
    EmbeddingModel embeddingModel;

    public void ingest(List<Document> documents) {
        EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
                .embeddingStore(store)
                .embeddingModel(embeddingModel)
                .documentSplitter(recursive(500, 0))
                .build();
        // Warning - this can take a long time...
        ingestor.ingest(documents);
    }
}
```

Configuration Settings

By default, the extension utilizes the default Redis datasource for storing and indexing the documents. Customize the behavior of the extension by exploring various configuration options:

🔒 Configuration property fixed at build time - All other configuration properties are overridable at runtime

Configuration property	Type	Default
🔒 `quarkus.langchain4j.redis.client-name` The name of the Redis client to use. These clients are configured by means of the `redis-client` extension. If unspecified, it will use the default Redis client. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_CLIENT_NAME`	string	
`quarkus.langchain4j.redis.dimension` The dimension of the embedding vectors. This has to be the same as the dimension of vectors produced by the embedding model that you use. For example, AllMiniLmL6V2QuantizedEmbeddingModel produces vectors of dimension 384. OpenAI's text-embedding-ada-002 produces vectors of dimension 1536. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_DIMENSION`	long	required ❗
`quarkus.langchain4j.redis.index-name` Name of the index that will be used in Redis when searching for related embeddings. If this index doesn't exist, it will be created. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_INDEX_NAME`	string	`embedding-index`
`quarkus.langchain4j.redis.metadata-fields` Names of extra fields that will be stored in Redis along with the embedding vectors. This corresponds to keys in the `dev.langchain4j.data.document.Metadata` map. Storing embeddings with metadata fields unlisted here is possible, but these fields will then not be present in the returned EmbeddingMatch objects. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_METADATA_FIELDS`	list of string	
`quarkus.langchain4j.redis.distance-metric` Metric used to compute the distance between two vectors. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_DISTANCE_METRIC`	`l2`, `ip`, `cosine`	`cosine`
`quarkus.langchain4j.redis.vector-field-name` Name of the key that will be used to store the embedding vector. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_VECTOR_FIELD_NAME`	string	`vector`

`quarkus.langchain4j.redis.scalar-field-name` Name of the key that will be used to store the embedded text. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_SCALAR_FIELD_NAME`	string	`scalar`
`quarkus.langchain4j.redis.prefix` Prefix to be applied to all keys by the embedding store. Embeddings are stored in Redis under a key that is the concatenation of this prefix and the embedding ID. If the configured prefix does not ends with `:`, it will be added automatically to follow the Redis convention. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_PREFIX`	string	`embedding:`
`quarkus.langchain4j.redis.vector-algorithm` Algorithm used to index the embedding vectors. Environment variable: `QUARKUS_LANGCHAIN4J_REDIS_VECTOR_ALGORITHM`	`flat`, `hnsw`	`hnsw`

Under the Hood

Each ingested document is saved as a JSON document in Redis, containing the *embedding* stored as a vector. The document store also generates an index for each ingested document. To retrieve relevant documents, the extension employs the Redis *search* command.