Docs

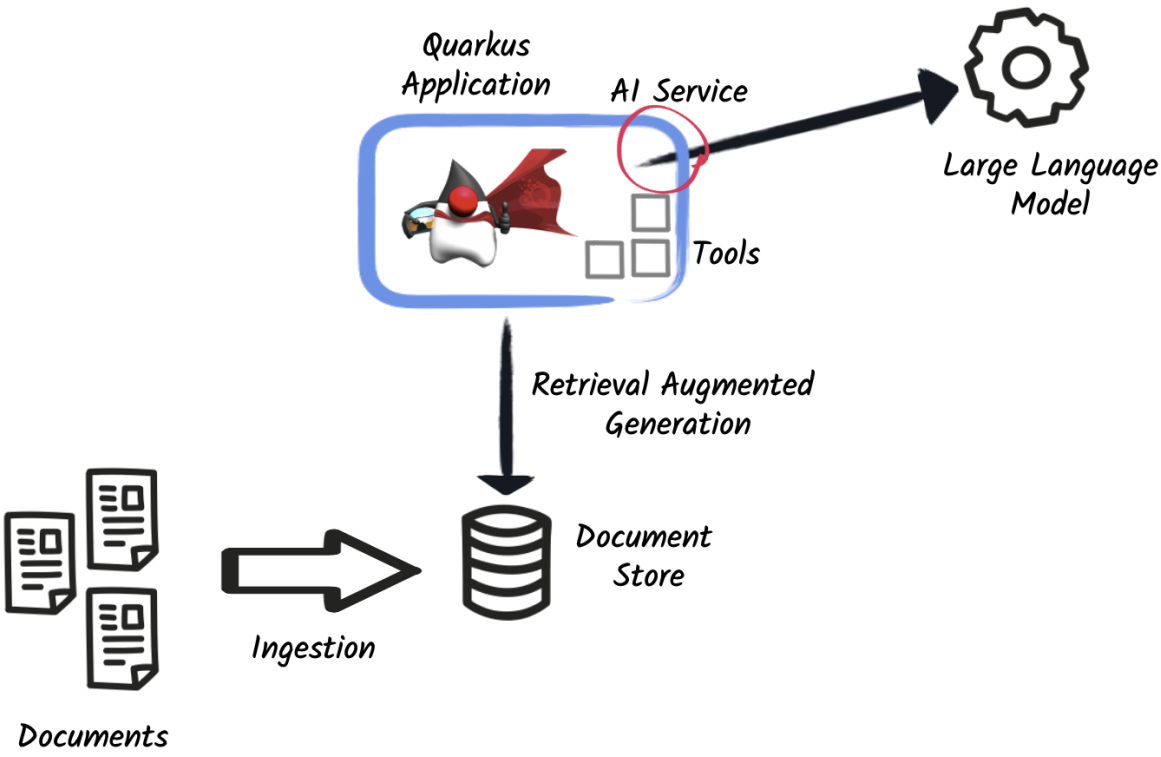
Quarkus LangChain4j

Large Language Models (LLMs) are AI-based systems designed to understand, generate, and manipulate human language, showcasing advanced natural language processing capabilities. The dynamic LLM landscape is reshaping our interactions with applications and the very construction of these applications. The Quarkus LangChain4j extension seamlessly integrates LLMs into Quarkus applications, enabling the harnessing of LLM capabilities for the development of more intelligent applications.

For instance, an application utilizing this extension can:

- Automatically triage or classify documents
- Extract structured and unstructured information from various data sources
- Construct chatbots for system interaction
- Generate personalized text such as emails or reports

This extension is built upon the LangChain4j library. It offers a declarative approach to interact with diverse LLMs like OpenAI, Hugging Face, or Ollama. It facilitates LLM-invoked functions within Quarkus applications and allows document loading within the LLM "context".



Quick Overview

To incorporate Quarkus LangChain4j into your Quarkus project, add the following Maven dependency:

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

or, to use hugging face:

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

Then, include your OpenAI API key in your `application.properties` file (or any other mandatory configuration):

```
quarkus.langchain4j.openai.api-key=sk-...
```

TIP

Alternatively, utilize the `QUARKUS_LANGCHAIN4J_OPENAI_API_KEY` environment variable.

TIP

If you want to see the LLM requests or responses in the console log, also set `quarkus.langchain4j.log-requests=true` and/or `quarkus.langchain4j.log-responses=true`.

Once you've added the dependency and configuration, the next step involves creating an *AI service*, serving as the integration point. This service is the interface your application code will utilize to interact with the LLM. A basic example is demonstrated below:

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

import dev.langchain4j.service.SystemMessage;
import dev.langchain4j.service.UserMessage;
import io.quarkiverse.langchain4j.RegisterAiService;

@RegisterAiService( ①
```

```
        tools = EmailService.class ②
)
public interface MyAiService {

    @SystemMessage("You are a professional poet") ③
    @UserMessage("""
                Write a poem about {topic}. The poem should be {lines} lines long.
Then send this poem by email. ④
            """)
    String writeAPoem(String topic, int lines); ⑤
}
```

① The `@RegisterAiService` annotation registers the *AI service*.

② The `tools` attribute defines the *tools* the LLM can employ. During interaction, the LLM can invoke these tools and reflect on their output.

③ The `@SystemMessage` annotation registers a *system message*, setting the initial context or "scope".

④ The `@UserMessage` annotation serves as the *prompt*.

⑤ The method invokes the LLM, initiating an exchange between the LLM and the application, beginning with the system message and then the user message. Your application triggers this method and receives the response.

Advantages over vanilla LangChain4j

The extension offers the following advantages over using the vanilla LangChain4j library in Quarkus:

- Seamless integration with the Quarkus programming model
  - CDI beans for the LangChain4j models
  - Standard configuration properties for configuring said models
- Declarative AI Services
- Built-in observability
  - Metrics
  - Tracing
  - Auditing
- Build time wiring
  - Reduced footprint of the library
  - Feedback about misuse at build time
- Leverage runtime Quarkus components

    - REST calls and JSON handling are performed using the libraries used throughout Quarkus
        - Results in reduces library footprint
        - Enables GraalVM native image compilation
- Dev Services
    - Quarkus starts embedding stores and chat memory stores for dev and test mode automatically
- Dev UI features
    - View table with information about AI services and tools
    - Add embeddings into the embedding store
    - Search for relevant embeddings in the embedding store
- Automatic chat memory management
    - For REST and WebSocket contexts, Quarkus can automatically handle the lifecycle of chat memory

Retrieval Augmented Generation

For the easiest possible way to get started with Retrieval Augmented Generation, refer to Easy RAG.
