JakartaEE meets AI

Emmanuel Hugonnet | ehugonne@redhat.com

WildFly

Large Language Models (LLMs)

- Neural Networks
  - Transformer based
  - Recognize, Predict, and Generate text
  - Trained on a VERY large corpus of text
  - Deduce the statistical relationships between tokens
  - Can be fine-tuned

A LLM predicts the next token based on its training data and statistical deduction

Java meets AI

WildFly

Java

python

LangChain

JakartaEE meets AI

WildFly

smallrye-llm

CDI

JAKARTA EE

Microprofile config

Microprofile Fault Tolerance

Microprofile OpenTelemetry

WildFly meets AI

Support for streaming/chat models :

- github-chat-model / github-streaming-chat-model
- groq-chat-model / groq-streaming-chat-mode
- mistral-ai-chat-model / mistral-ai-streaming-chat-mode
- ollama-chat-model / ollama-streaming-chat-mode
- openai-chat-model / openai-streaming-chat-mode

Support for embedding models:

- 8 in-memory-embedding-models
- ollama-embedding-model

Support for embedding stores:

- in-memory-embedding-store
- neo4j-embedding-store
- weaviate-embedding-store

Support for content retriever for RAG:

- default-embedding-content-retriever
- neo4j-content-retriever
- web-search-engines

Support for tool provider (MCP):

- mcp-sse
- mcp-stdio

Components of LangChain4j

Chains

Tools

AI Services

Basics

Image Models

Language Models

Prompt Templates

Output Parsers

Memory

RAG

Document Loaders

Document Splitters

Embedding Models

Embedding Stores

WildFly AI Feature Pack

- RAG components:
  - embedding models
  - embedding stores
  - content retrievers
- Model Context Protocol Client:
  - SSE
  - Stdio
- Model Context Protocol server provides :
  - Tools: to be called form a LLM
  - Prompts: or prompts template to be shared across applications
  - Resources: Data as text or binary

AI Service


```
@RegisterAIService(chatLanguageModelName = "ollama", tools =
{org.wildfly.ai.websocket.Weather.class,org.wildfly.ai.websocket.Calculator.class}, scope =
SessionScoped.class)
public interface SimpleAIService {
    @SystemMessage("""
                   You are an AI named Bob answering general question.
                   Your response must be polite, use the same language as the question, and be
relevant to the question.""")
    String chat(@UserMessage String question);
}
```

AI Service with MCP client


```
@RegisterAIService(chatMemoryName = "chat-ai-service-memory", contentRetrieverName = "embedding-store-retriever",
streamingChatLanguageModelName = "streaming-mistral", toolProviderName = "mcp")

public interface ChatAiService {

@SystemMessage(""" You are a customer support agent of a car rental company named 'Miles of Smiles'. Before providing
information about a specific booking or canceling a booking, you MUST always check: booking number, customer first
name and last. You should not answer to any request not related to car booking or Miles of Smiles company general
information. When a customer wants to cancel a booking, you must check his name and the Miles of Smiles cancellation
policy first. Any cancelation request must comply with cancellation policy both for the delay and the duration. Today
is {{current_date}}. """)

TokenStream streamingChat(String question);

}
```

MCP Server

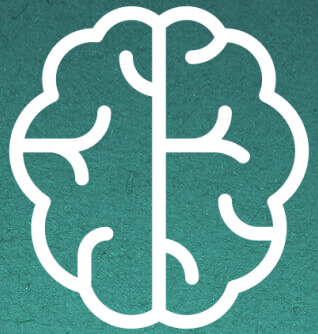

```java
@Prompt(name = "Prometheus-metrics-chart", description = "Prometheus metrics chart")
        PromptMessage prometheusMetricsChart() {
    return PromptMessage.withUserRole(new TextContent("using available tools, get Prometheus metrics from wildfly server. "
                + "You will repeat the invocation 3 times, being sure to wait 2 seconds between each invocation. "
                + "After all the 3 invocation has been completed you will organize the data in a table. "
                + "Then you will use this table to create a bar chart to visually compare the data. "
                + "Be sure to use at least 5 different data column and be sure to represent all data as bar in the chart"));
}

@Tool(description = "Get weather alerts for a US state.", name = "alerts")
public String getAlerts(@ToolArg(description = "Two-letter US state code (e.g. CA, NY)") String state) {
    return formatAlerts(weatherClient.getAlerts(state));
}

@Resource(uri = "file:///${jboss.server.log.dir}/server.log", mimeType = "text/plain", name = "server.log")
         TextResourceContents serverLog() throws IOException {
    Path path = new File(System.getProperty("jboss.server.log.dir"), "server.log").toPath();
    return TextResourceContents.create("file:///${jboss.server.log.dir}", Files.readString(path));
}
```

Sample configuration


```
<subsystem xmlns="urn:jboss:domain:ai:1.0">
  <chat-language-models>
    <mistral-ai-chat-model name="streaming-mistral" api-key="${env.MISTRAL_API_KEY:YOUR_KEY_VALUE}"
base-url="https://api.mistral.ai/v1" log-requests="true" log-responses="true" model-name="mistral-small-latest"
streaming="true"/>
  </chat-language-models>
  <embedding-models>
    <in-memory-embedding-model name="all-minilm-l6-v2" module="dev.langchain4j.embeddings.all-minilm-l6-v2" embedding-
class="dev.langchain4j.model.embedding.onnx.allminilml6v2.AllMiniLmL6V2EmbeddingModel"/>
  </embedding-models>
  <embedding-stores>
    <in-memory-embedding-store name="in-memory" path="embeddings.json" relative-to="jboss.server.config.dir"/>
  </embedding-stores>
  <content-retrievers>
    <embedding-store-content-retriever name="embedding-store-retriever" embedding-model="all-minilm-l6-v2" embedding-
store="in-memory" max-results="2" min-score="0.7"/>
  </content-retrievers>
  <mcp>
    <mcp-tool-provider name="mcp" mcp-clients="mcp-sse"/>
    <mcp-client-sse name="mcp-sse" connect-timeout="6000000" log-requests="true" log-responses=true" sse-path="/sse" socket-
binding="mcp-sse"/>
  </mcp>
</subsystem>
```



Demo

Emmanuel Hugonnet | ehugonne@redhat.com

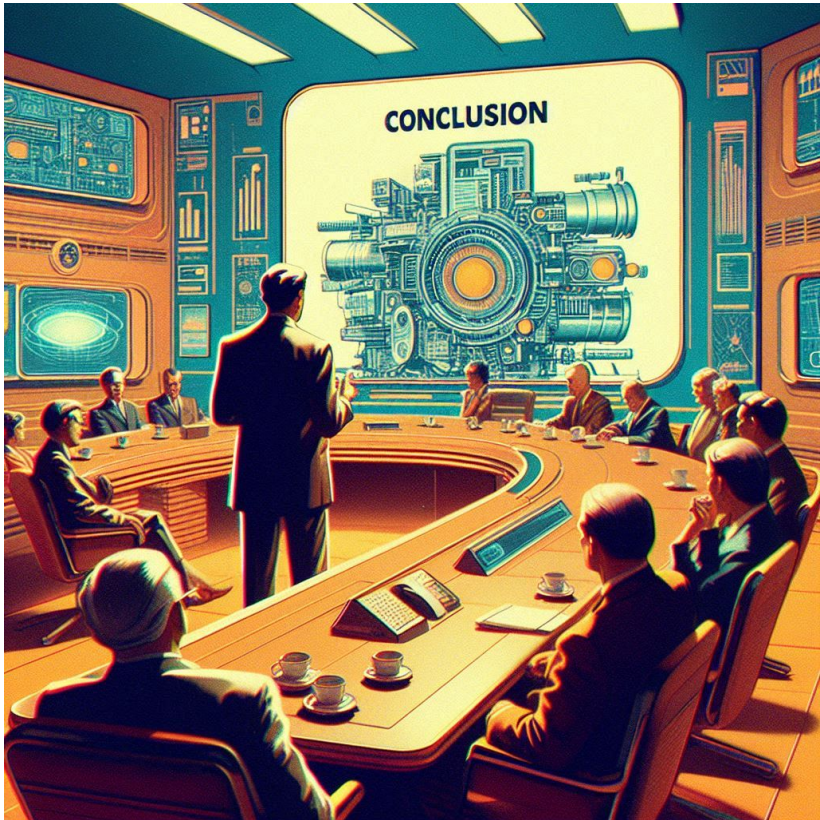
WildFly

What's next

- WASM MCP Server
- More providers
- Fault Tolerance integration
- Graph RAG
- MCP specification changes

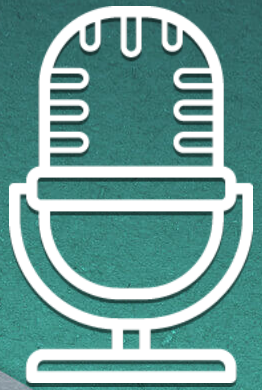
Conclusion

- Use WildFly Management to create required resources
- Layers to trim down the configuration to our needs
- CDI injection of those resources
- AI Service support
- MCP Server and MCP client support



Resources

- LangChain4J:
  - https://docs.langchain4j.dev/
- Model Context Protocol:
  - https://spec.modelcontextprotocol.io/
- smallrye-llm: a cross vendor LangChain4J -CDI integration library
  - https://github.com/smallrye/smallrye-llm
- WildFly AI Feature Pack:
  - https://github.com/wildfly-extras/wildfly-ai-feature-pack
- WildFly MCP: to interact with WildFly servers through MCP
  - https://github.com/wildfly-extras/wildfly-mcp
- Admin your WildFly with LLM and MCP
  - https://www.youtube.com/watch?v=wg1hAdOoe2w
- Making WildFly Glow with Intelligence:
  - https://www.wildfly.org/news/2025/02/10/Glowing-with-AI/
- Playing with Generative AI with WildFly:
  - https://www.wildfly.org/news/2024/11/04/WildFly-playing-with-generative-ai/



Questions

Emmanuel Hugonnet | ehugonne@redhat.com

WildFly