MCP: using Java and Quarkus to bridge LLMs with your applications and data

Horacio Gonzalez

2025-04-25

Sébastien Blanc

port

Horacio Gonzalez

clever cloud

Who are we?

Sébastien Blanc

DevRel

port

@sebi2706

Horacio Gonzalez

DevRel

clever cloud

@LostInBrittany

Summary

1. Introduction
2. Understanding LLMs & Their Landscape
3. Making Java applications LLM-aware
4. Model Context Protocol (MCP): The missing link
5. Workshop: Building an MCP server in Java with Quarkus
6. The future of Java & LLM integration
7. Q&A and discussion

We want to see it working!

README

MCP with Quarkus Mini Lab

Goal of the lab

In this lab you will learn how to create a MCP Server from scratch and extract `Tools` from an existing app to the MCP Server.

Pre-requisites

- Java 17/22
- Maven
- Ollama

Step 1 : Running the initial application

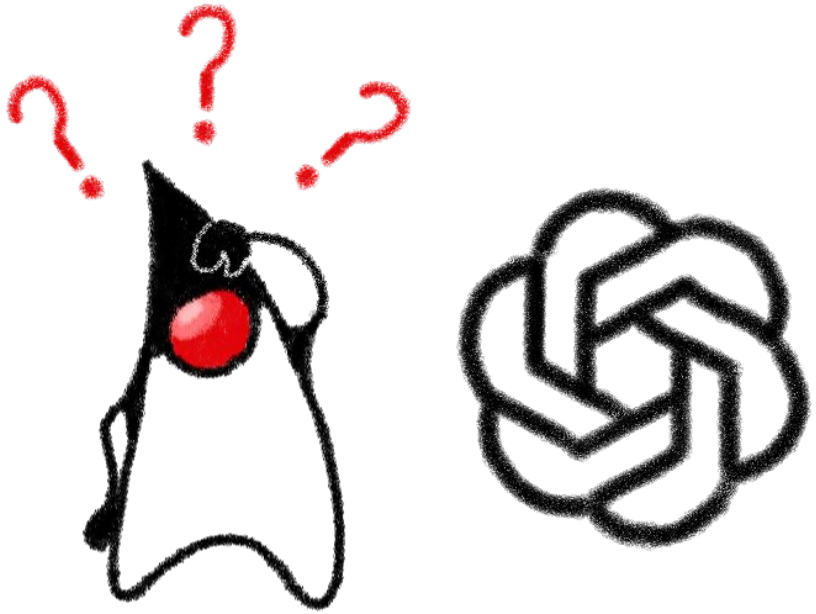
Go to the `meeting-bot` folder and run the app : `mvn quarkus:dev` or `quarkus dev` and open your browser to `localhost:8080`.

Play around with the chat bot, try to make the bot calling the Tools, add also a tool to discover how it works.

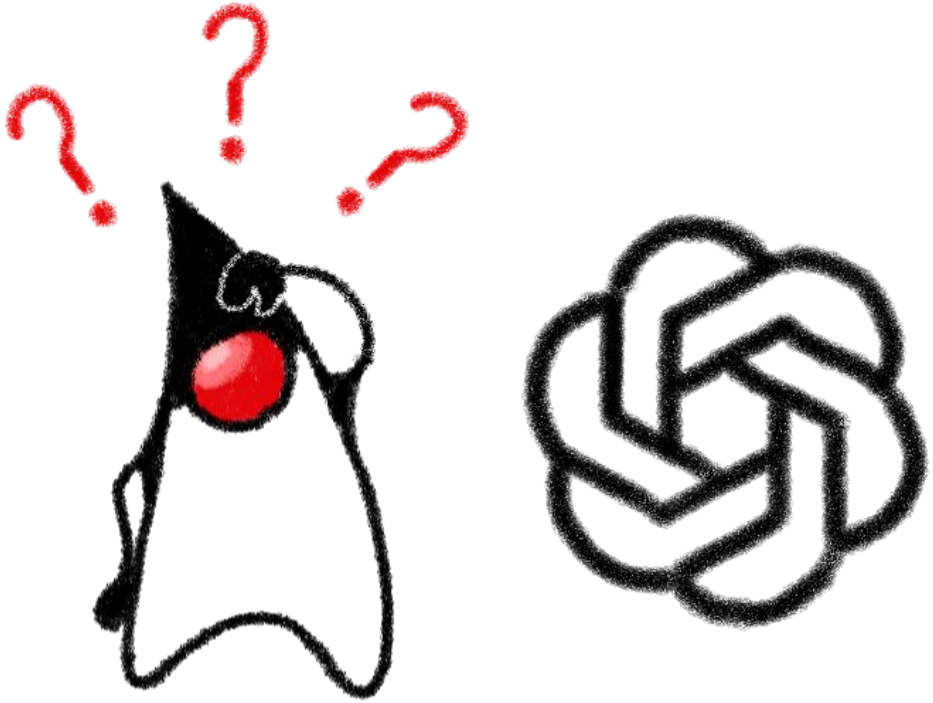
https://github.com/sebi-and-horacio/mcp-quarkus-lab

Introduction

LLMs are changing software development—
how can Java developers take full advantage?



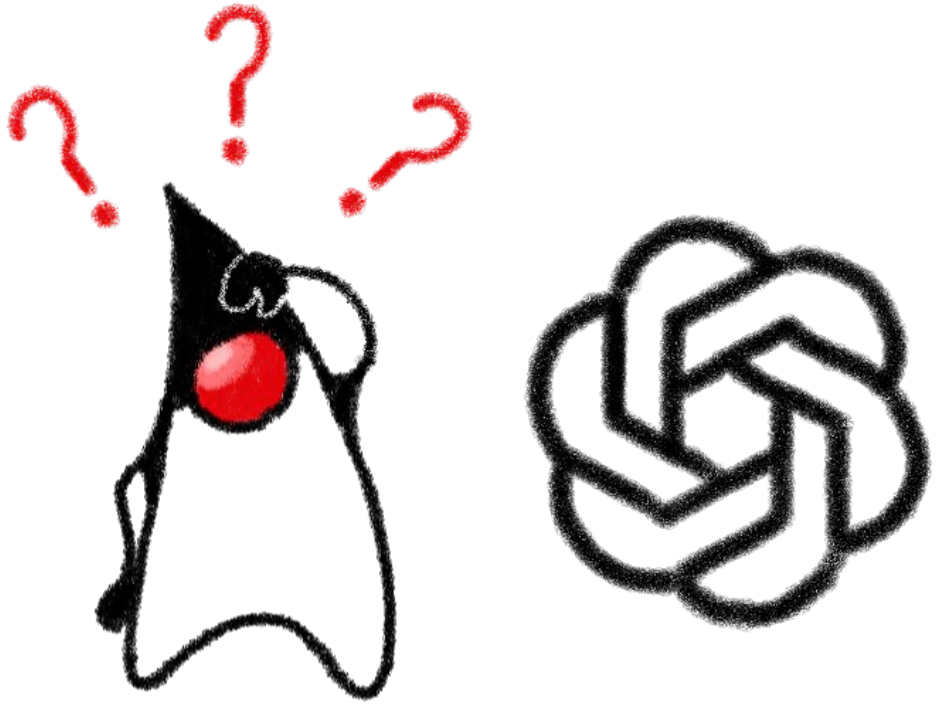
Why are we talking about this?



LLMs are changing development,
but most Java apps don't fully leverage them

How do you use LLMs for your dev job?

1. Who here has already used LLM?
2. Who here has already used LLM professionally?
3. Who here has already used LLM with code?
4. Who here has already used LLMs in a Java app?



How LLMs are changing dev jobs

The 70% problem: Hard truths about AI-assisted coding

A field guide and why we need to rethink our expectations

ADDY OSMANI
DEC 04, 2024

800 44 121 Share

After spending the last few years embedded in AI-assisted development, I've noticed a fascinating pattern. While engineers report being dramatically more productive with AI, the actual software we use daily doesn't seem like it's getting noticeably better. What's going on here?

I think I know why, and the answer reveals some fundamental truths about software development that we need to reckon with. Let me share what I've learned.

https://addyo.substack.com/p/the-70-problem-hard-truths-about

Understanding LLMs & Their Landscape

LUNACONF

Closed-source, open-source, local—
choosing the right LLM for your Java application.

LLMs come in different flavors

Not all LLMs are created equal

They have different trade-offs
in capabilities, accessibility, and control

Choosing the right one depends on
your use case, security needs, and infrastructure.

Closed-source LLMs (Cloud-based APIs)

📌 Examples

- OpenAI (ChatGPT), Anthropic (Claude), Google (Gemini), Microsoft (Copilot)

✅ Advantages:

- Powerful and well-trained (best models available)
- Easy to use via APIs
- Regularly updated & improved

❌ Challenges:

- Black box (you don't control how they work)
- Expensive (API calls can add up quickly)
- Data privacy concerns (sending requests to external servers)

💡 When to use?

- If you need the most advanced models and don't mind API costs or external dependencies.

Open-source LLMs (Self- or cloud-hosted)

📌 Examples

- Meta's Llama 3, Mistral, Google's Gemma, Alibaba's Qwen

✅ Advantages:

- Greater control (you know exactly how the model works)
- Can be fine-tuned for specific needs
- No external API costs

❌ Challenges:

- Requires more setup (you have to run the model yourself)
- May not be as powerful as the latest closed models
- Needs infrastructure (e.g., GPUs for hosting)

💡 When to use?

- If you need control over the model & lower costs but are okay with slightly weaker performance

Local models (on your machine or server)

📌 Examples

- Ollama, GGUF-based models (e.g., Llama, Mistral, Mixtral)

✅ Advantages:

- Works offline (great for security-sensitive applications)
- No API costs (completely free to use once set up)
- Low latency (responses are instant if hardware is good)

❌ Challenges:

- Limited by your hardware (needs a strong CPU/GPU)
- Not always as capable as cloud-hosted models
- Setup complexity (installing and optimizing models)

💡 When to use?

- If you need privacy and control, and you have the hardware to run an LLM efficiently

Choosing the Right Model for your Apps

- Cloud APIs
  - Great for rapid development, but costly and not always secure
- Self-hosted open models
  - Best balance for long-term control and scalability
- Local models
  - Best for privacy-sensitive applications

Making Java Applications LLM-Aware

LangChain4j simplifies LLM integration—
let's see how it works!

Two ways to integrate LLMs with Java

1. Java applications using LLMs
   - Using LLMs as assistants, API consumers, or reasoning engines
   - Easier and works well for code generation, chatbots, and AI assistants

2. LLMs using Java applications
   - Exposing Java functions, APIs, and databases to LLMs for tool execution
   - More powerful but requires tool calling and execution control.

What is LangChain4j?

LangChain4j is a Java library that simplifies LLM integration

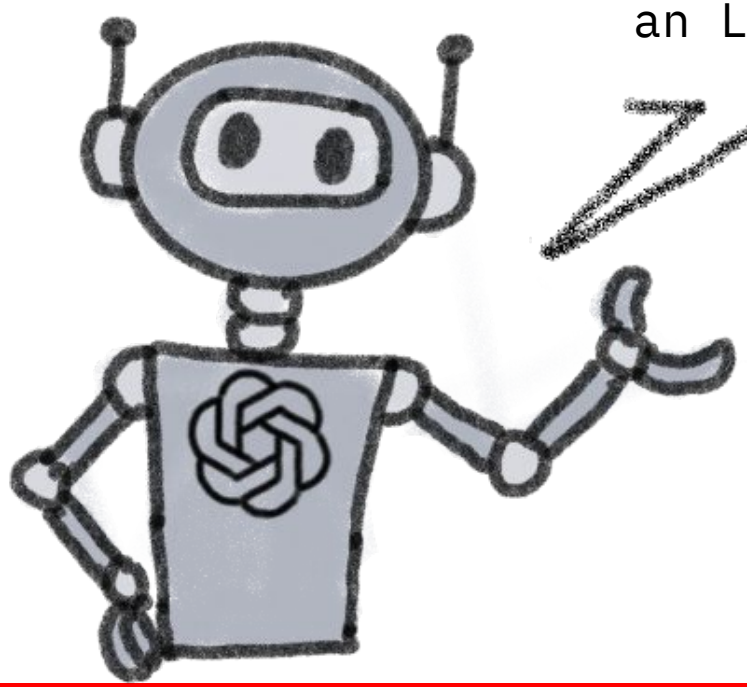
It’s the Java equivalent of LangChain (Python/JS)

✅ Prompt management – Structured input for LLMs.
✅ Memory – Keeping conversational state.
✅ Agents – Allowing LLMs to decide which tools to use.
✅ Tool calling – Exposing functions to LLMs.

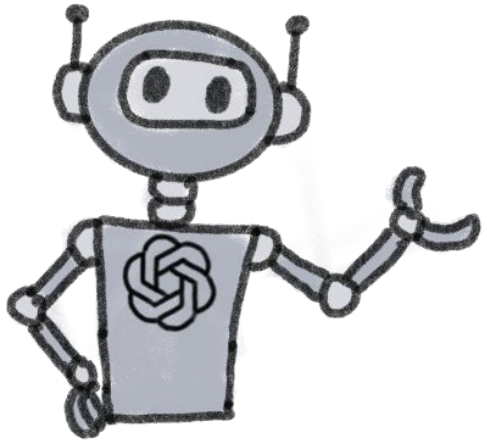
What are tools in LLMs?

Tools allow LLMs to do **more than just generate text**. They can interact with APIs, databases, and execute functions.

Instead of returning “I don’t know,” an LLM can call a tool **to fetch real data**.



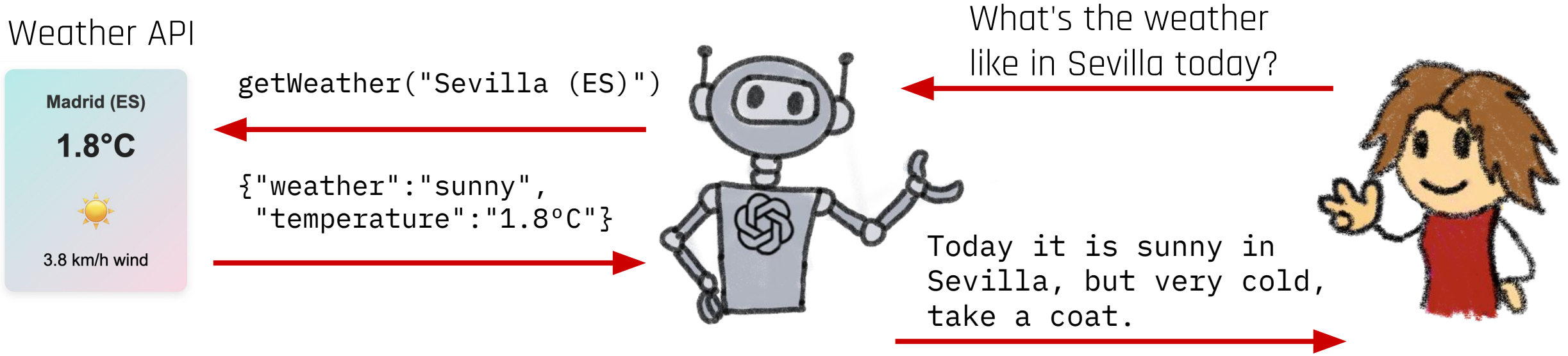
An LLM without Tools can't answer this



What's the weather like in Sevilla today?

I'm unable to provide real-time information or current weather updates.

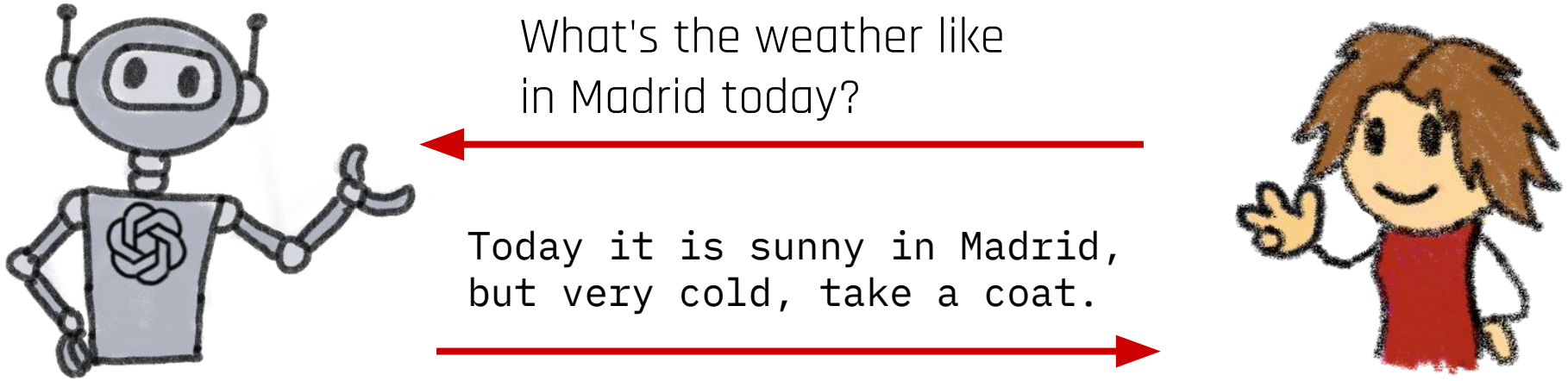
Example of how tool calling works



LLM recognizes it needs an external function and calls it.
It integrates the result into a natural-language response.

Why this matters?

- Moves LLMs from static text generation
  - dynamic system components
- Increases accuracy & real-world usability
- Allows developers to control what the LLM can access



Using LangChain4j to Define LLM Tools

LyingWeatherTool.java

```java
//DEPS dev.langchain4j:langchain4j:1.0.0-beta1

import dev.langchain4j.agent.tool.Tool;

public class LyingWeatherTool{
 @Tool("A tool to get the current weather in a city")
 public static String getWeather(String city) {
   return "The weather in " + city + " is sunny and hot.";
 }
}
```

LangChain4j gives as a Tool framework

Using LangChain4j to Define LLM Tools

LyingWeatherTool.java

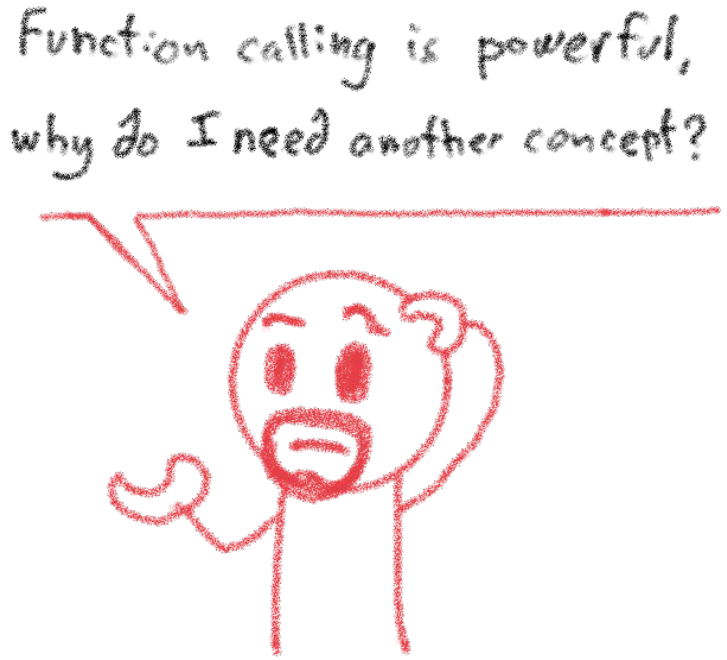
```
[...]
   Assistant assistant = AiServices.builder(Assistant.class)
        .chatLanguageModel(model).chatMemory(chatMemory)
        .tools(new LyingWeatherTool()).build();
   System.out.println("--------------------------------------");
   String question = "What will the weather be like in Madrid tomorrow?";
   String response = assistant.chat(question);
   System.out.println(response);
   System.out.println("--------------------------------------");
[...]
```

And an AI Service abstraction

Model Context Protocol (MCP): The missing link

MCP bridges LLMs with your applications, enabling controlled, real-world interactions

Why Do We Need MCP?

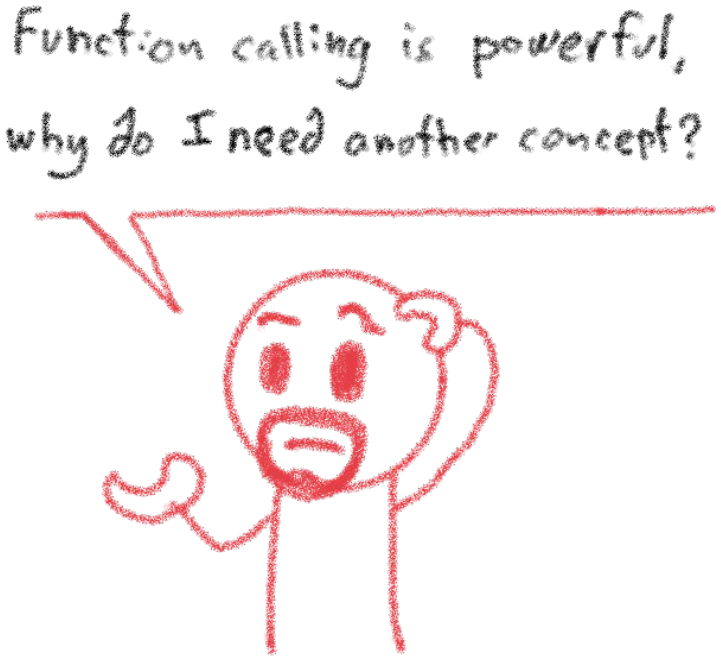


LLM function calling is useful, but lacks structure

Why Do We Need MCP?

Problem

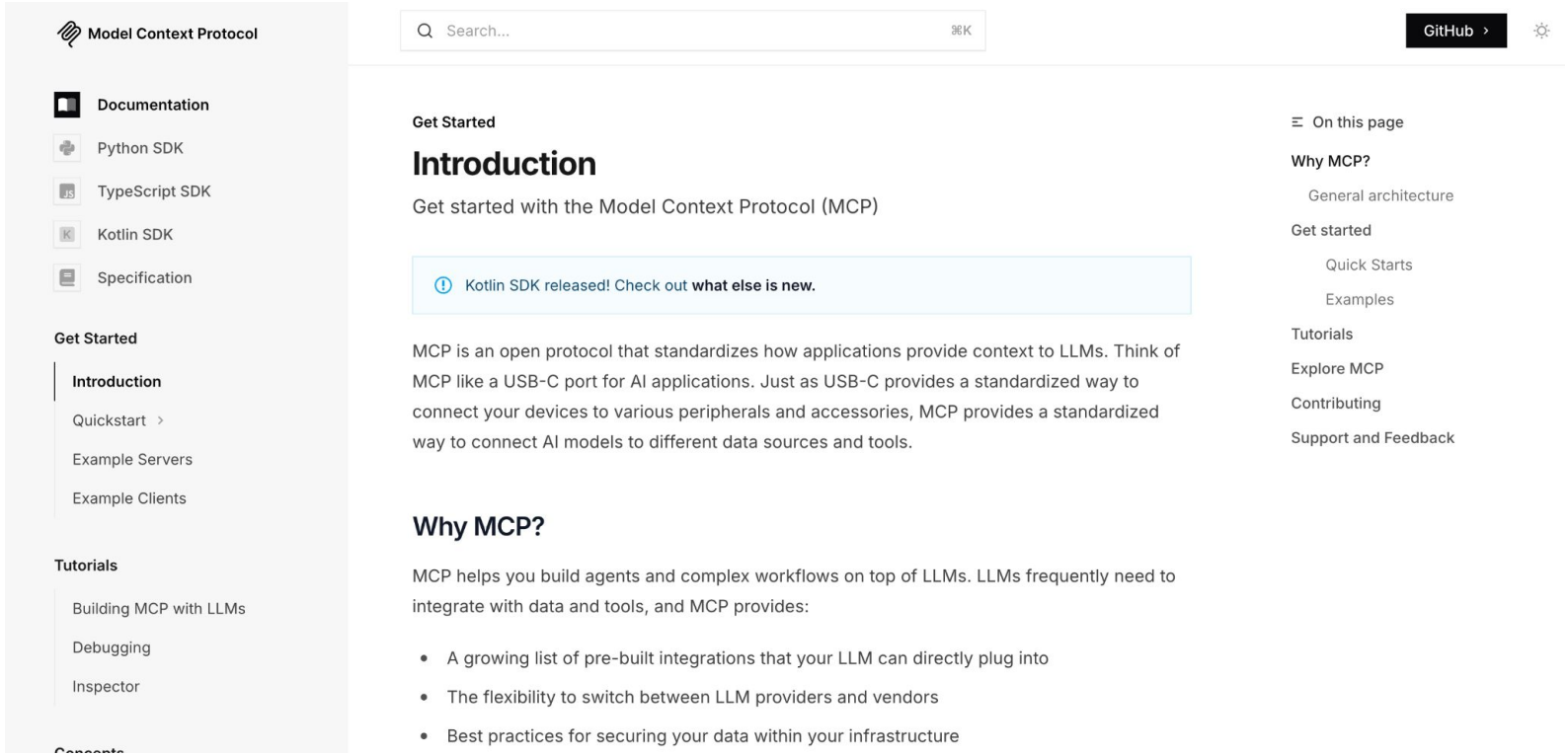
- LLMs **don't automatically know** what functions exist.
- **No standard way** to expose an application's capabilities.
- **Hard to control** security and execution flow.



Solution: MCP

- MCP **defines a standard way** to describe and expose functionalities.
- **Applications stay in control** over what LLMs can do.

Model Context Protocol

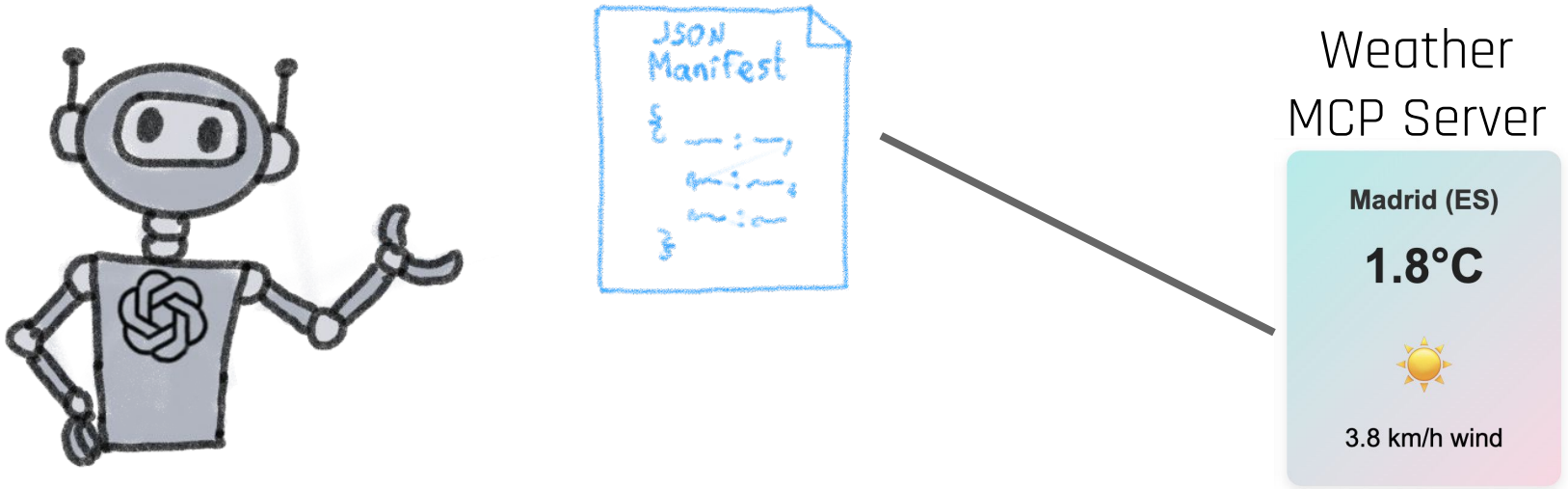


De facto standard for exposing system capabilities to LLMs

https://modelcontextprotocol.io/

How MCP works

- Applications define an MCP manifest (structured JSON).
- The manifest describes available functions, input/output formats, and security policies.
- LLMs can discover and request function execution safely.



MCP is provider-agnostic

Works with any LLM provider

Ensures standardized function exposure across platforms

Understanding the MCP Manifest

mcp-manifest.json

```
{ "functions": [
    {
      "name": "getWeather",
      "description": "Fetches the current weather for a city.",
       "parameters": {
         "city": { "type": "string", "required": true },
         "countryCode": { "type": "string", "required": true }
       }
    }
  ]}
```

- Lists available tools
- Describes expected inputs/outputs
- Defines execution policies

We want to see it working!

langchain4j-for-tools-and-mcp-demo Public

Pin | Unwatch 1 | Fork 0 | Star 0

main | 1 Branch | 0 Tags | Go to file t | Add file | Code

LostInBrittany Updating the README d28fa1f · 1 minute ago 6 Commits

01-Hello-LangChain4j	Refactoring and adding the README	2 hours ago
02-Lying-weather-Tool	Adding Real Weather Demo	9 minutes ago
03-Real-weather-Tool	Adding Real Weather Demo	9 minutes ago
LICENSE	Initial commit	3 hours ago
README.md	Updating the README	1 minute ago

README | MIT license

LangChain4j for Tools and MCP - Demo Repository

This repository contains all the code and examples from my talk:
MCP: using Java and Quarkus to bridge LLMs with your applications and data, presented at:

- **2024-03-05** - Madrid JUG (Madrid, Spain) - Slides

About

A series of demos to illustrate my talk "MCP: using Java and Quarkus to bridge LLMs with your applications and data"

Readme
MIT license
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

https://github.com/LostInBrittany/langchain4j-for-tools-and-mcp-demo/

That's all, folks!

Thank you all!