

Atelier: coder avec une IA générative

Utilisation de Large Language Models pour développer un logiciel
en respectant les standards de l'industrie

Partie 5

Nicolas Debeissat
nicolas.debeissat@mail-formateur.net

Plan du module

Installation de l'extension continue.dev

Dans VSCode, avec ollama en local

Utiliser la configuration de base

Création d'une stack React + FastAPI + Postgresql

Configuration avancée et codebase

Comment adapter ses réponses

RAG avec Langchain

API de génération de requêtes ElasticSearch

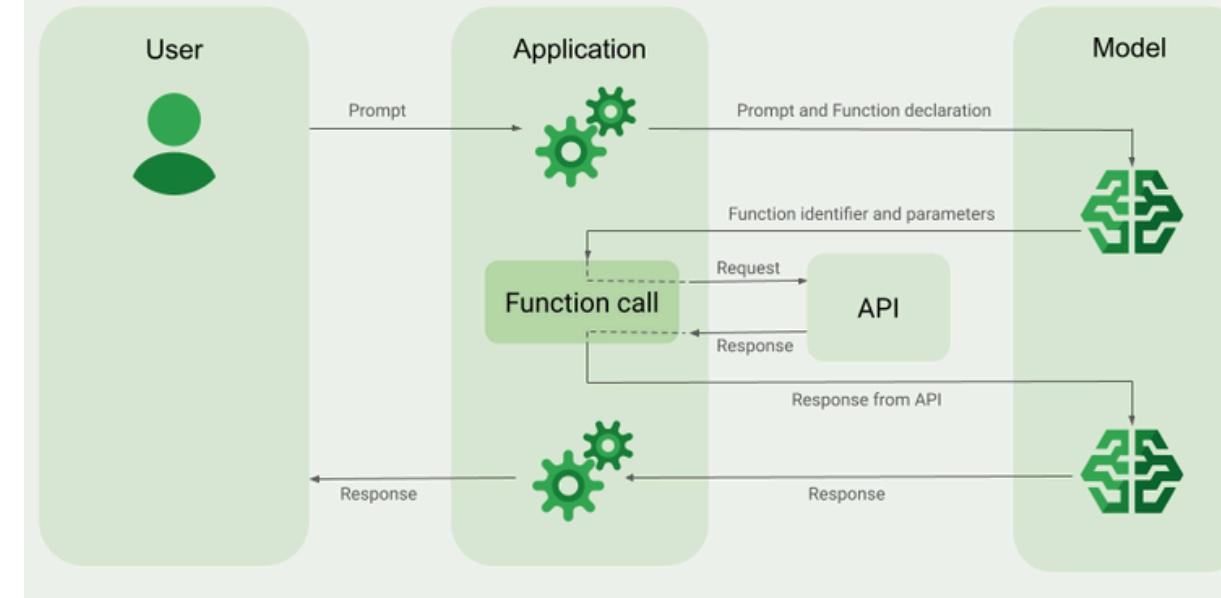
Model Context Protocol (mode Agent)

[Création d'outils et connexion à son agent](#)

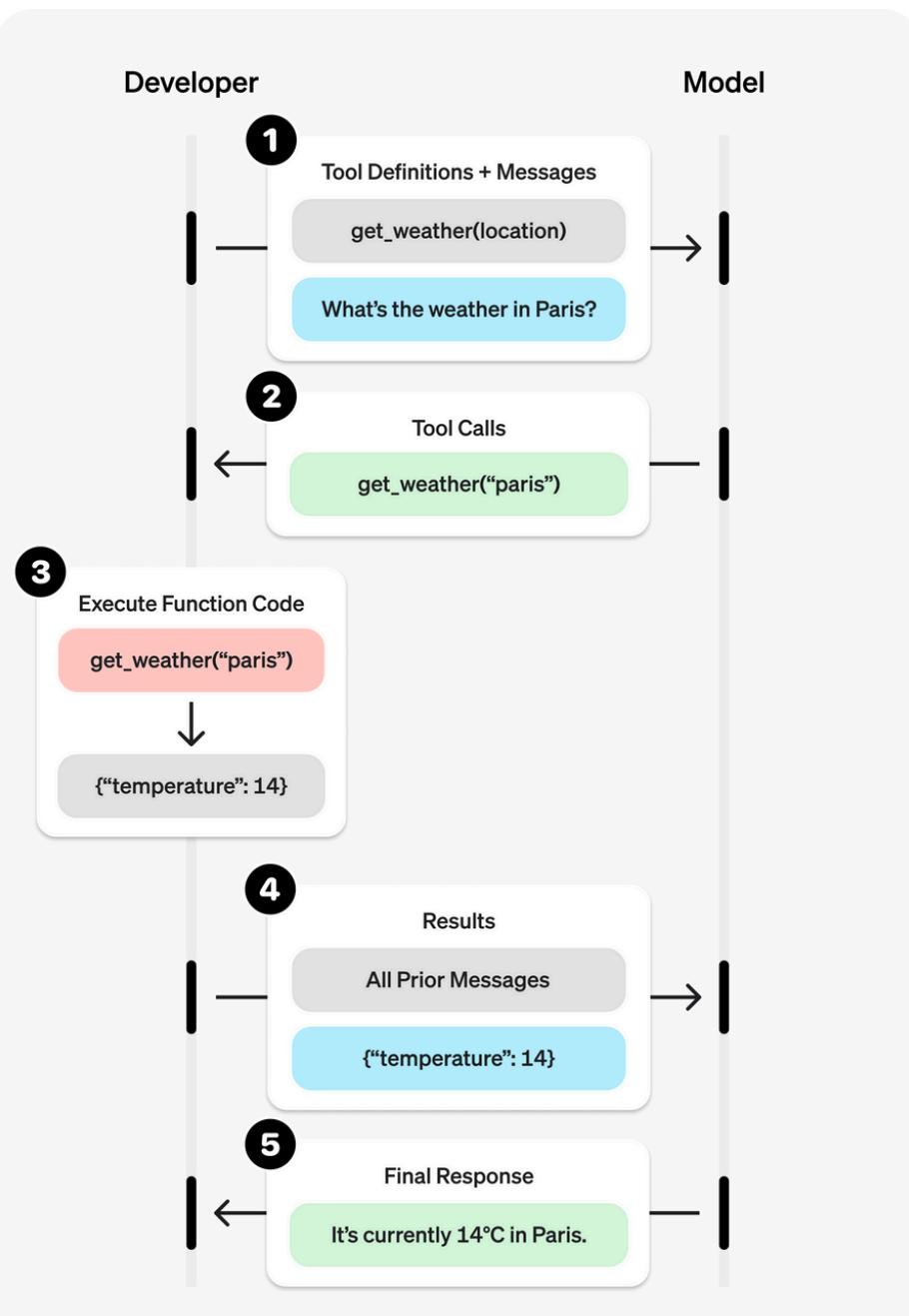
Evaluation

Quizz et rendu du projet effectué en cours

Function call



- L'utilisateur envoie le prompt suivant :
 - “combien d'utilisateurs français ai-je dans ma base de données ?”
- L'application ajoute au prompt la liste des fonctions dont il dispose
- Réponse du LLM :
 - Il faut utiliser l'outil “compte_utilisateurs_par_pays” avec le paramètre pays = “France”
- L'application exécute la fonction choisie avec les paramètres
- L'application envoie la réponse au LLM avec l'historique de conversation pour que le LLM traduise la réponse en langage naturel : “Vous avez 45 utilisateurs français.”



Function call

Définition des outils

```
tools = [
    {
        "type": "function",
        "name": "get_horoscope",
        "description": "Get today's horoscope for an astrological sign.",
        "parameters": {
            "type": "object",
            "properties": {
                "sign": {
                    "type": "string",
                    "description": "An astrological sign like Taurus or Aquarius",
                },
            },
            "required": ["sign"],
        },
    },
]
def get_horoscope(sign):
    return f"{sign}: Next Tuesday you will befriend a baby otter."
```

Appel du LLM

```
input_list = [
    {"role": "user", "content": "What is my horoscope? I am an Aquarius."}
]
response = client.responses.create(
    model="gpt-5",
    tools=tools,
    input=input_list,
)
input_list += response.output
for item in response.output:
    if item.type == "function_call":
        if item.name == "get_horoscope":
            horoscope =
get_horoscope(json.loads(item.arguments))
            input_list.append({
                "type": "function_call_output",
                "call_id": item.call_id,
                "output": json.dumps({
                    "horoscope": horoscope
                })
            })
```

Traduction en langage naturel

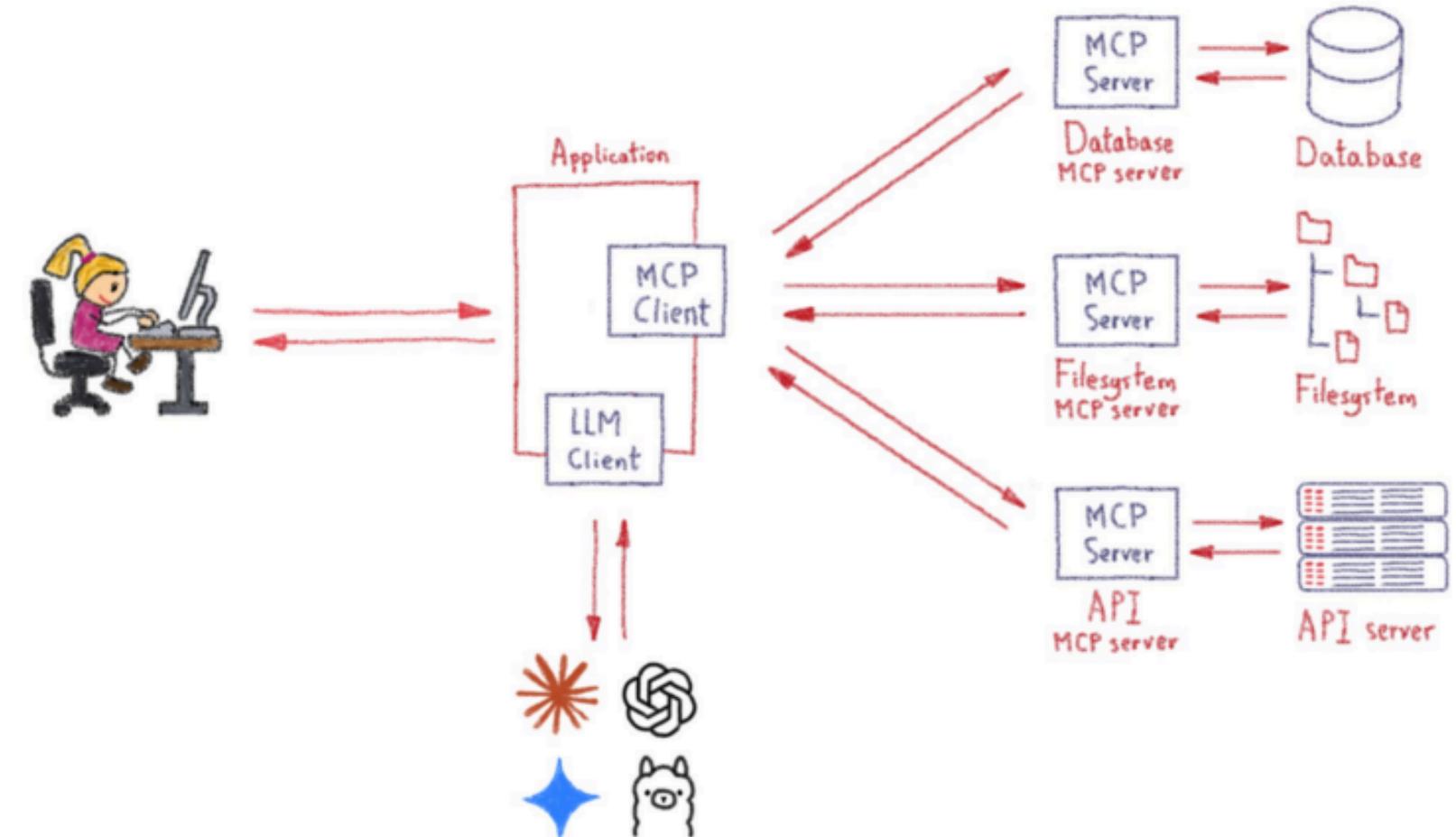
```
response = client.responses.create(
    model="gpt-5",
    instructions="Respond only with a horoscope generated by a tool.",
    tools=tools,
    input=input_list,
)
```

Problèmes du function call

- Chaque application maintient dans son code la liste des outils ?
- Chaque application est responsable de l'appel des outils ?

=> nécessite la création d'un standard pour pouvoir créer et utiliser des serveurs spécifiques à chaque usage, responsable de :

- lister les outils disponibles et renvoyer la liste au client MCP
- appeler les outils demandés par le client et retourner le résultat



Model Context Protocol

1. Le client MCP agrège les listes d'outils données par les serveurs MCP
2. A partir de la question de l'utilisateur, le client demande au LLM quel outil utiliser et avec quels paramètres
3. Le client demande au serveur MCP d'appeler l'outil et récupère la réponse
4. Le client envoie l'historique de la conversation, avec l'appel d'outil, et son résultat, au LLM :
 - Soit le LLM décide que la question a été entièrement traitée, et traduit la réponse en langage naturel
 - Soit le LLM décide qu'il faut appeler un autre outil, et retourne le nom du nouvel outil à utiliser, avec les paramètres à utiliser
 - => retour à l'étape 3

TD : création d'un serveur MCP

- uv init mcp-server
- code mcp-server/
- cd mcp-server
- uv venv
- source .venv/bin/activate
- uv add mcp requests mcp[cli]
- uv run mcp dev main.py

TD : création d'un serveur MCP

```
from mcp.server.fastmcp import FastMCP
import requests

mcp = FastMCP("Demo")

@mcp.tool()
def get_mapping_elasticsearch(index_name: str) -> dict:
    """ Get the mapping of an Elasticsearch index. """
    url = f"http://localhost:9200/{index_name}"
    headers = {"Content-Type": "application/json"}
    try:
        response = requests.get(url, headers=headers)
        response.raise_for_status()
        return response.json()[index_name]['mappings']
    except requests.exceptions.RequestException as e:
        return f"An error occurred: {e}"
```

```
@mcp.tool()
def search_elasticsearch(index_name: str, query: dict) -> dict:
    """ Searches Elasticsearch with the provided query. """
    url = f"http://localhost:9200/{index_name}/_search"
    headers = {"Content-Type": "application/json"}
    try:
        body = {"query": query} if "query" not in query else query
        response = requests.post(url, headers=headers, json=body)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.RequestException as e:
        return f"An error occurred: {e}"
```

```
if __name__ == "__main__":
    mcp.run()
```

TD : création d'un serveur MCP

- uv run mcp dev main.py
- Ouvrir l' inspector
- Démarrer un elasticsearch

```
docker run -d --name elasticsearch -p 9200:9200  
-p 9300:9300 -e "discovery.type=single-node" -e  
"xpack.security.enabled=false" elasticsearch:9.1.4
```

- Stopper le docker :

```
docker stop elasticsearch
```

```
docker rm elasticsearch
```

- Lire les logs du docker :

```
docker logs -f elasticsearch
```

The screenshot shows the MCP Inspector v0.15.0 interface. At the top, there is a message: "Open inspector with token pre-filled: http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=2ab5236fb0e20728a73f03136199fce389ba3a1e625969b128d2d3bafdc25e64". The main window has tabs for Resources, Prompts, Tools, Ping, Sampling, Roots, and Auth. The Tools tab is selected. It contains a section for "get_mapping_elasticsearch" which says "Get the mapping of an Elasticsearch index." and "index_name". Below this is a "Run Tool" button. On the left, there are sections for Transport Type (STDIO), Command (uv), Arguments (run --with mcp mcp run main.py), Environment Variables, Configuration, and buttons for Restart and Disconnect. A status bar at the bottom indicates "Connected". In the bottom right, there is a "History" section with items 2. tools/list and 1. initialize, and a "Server Notifications" section stating "No notifications yet".

TD : configuration dans Continue

```
name: Mcp windows
```

```
version: 0.0.1
```

```
schema: v1
```

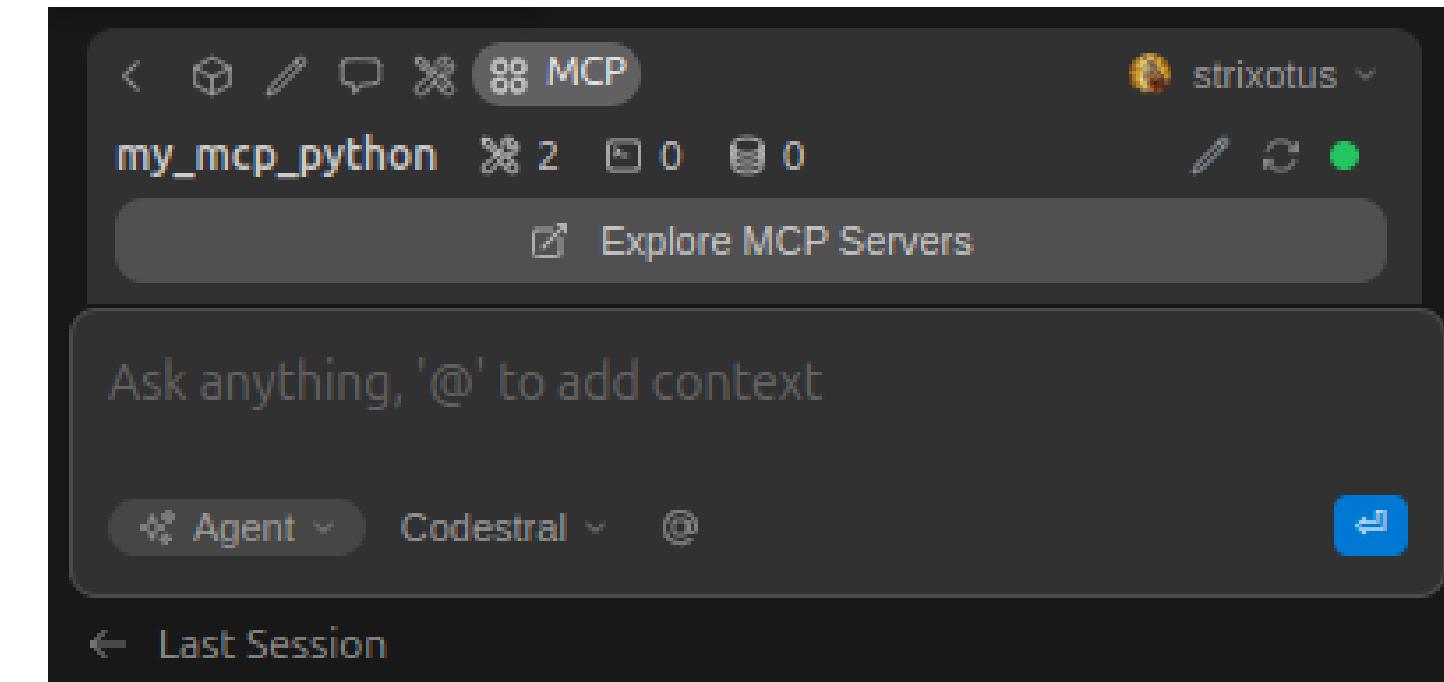
```
mcpServers:
```

```
- name: Mcp windows
```

```
  command: C:\Users\nicol\Documents\epsi\cours\atelier_coder_ia\mcp-
server\.venv\Scripts\python.exe
```

```
  args:
```

```
- C:\Users\nicol\Documents\epsi\cours\atelier_coder_ia\mcp-server\main.py
```



TD : appel du serveur

- Créer un index my-index

```
curl -X PUT -H "Content-Type: application/json" -d '{"mappings": {"properties": {"@timestamp": {"type": "date"}, "message": {"type": "text"} }}}' "http://localhost:9200/my-index"
```

- Voir l'index http://localhost:9200/_cat/indices et <http://localhost:9200/my-index>
- Crée un document

```
curl -X PUT -H "Content-Type: application/json" -d '{"@timestamp": "2099-11-15T13:12:00", "message": "Hello world !"}' "http://localhost:9200/my-index/_create/1"
```

- Voir le document http://localhost:9200/my-index/_doc/1
- Demander à continue en mode Agent :

“get the mapping of elasticsearch index my-index and with that mapping generate search query on message = “Hello” and search elasticsearch with the query”

The screenshot shows a user interface for a search or monitoring tool. At the top, there is a header with a dropdown menu labeled "CONTINUE: CONTINUE" and some icons. Below the header, there is a text input field containing the message "Hello". To the right of the input field, there is a button labeled "SEARCH". Underneath the input field, there is a section titled "Result" which displays the following information:

- > 1 rule
- ✓ Continue used the Mcp windows get_mapping_elasticsearch tool
- ✓ Continue used the Mcp windows search_elasticsearch tool

The main content area shows a success message: "The mapping of the Elasticsearch index "my-index" has been successfully retrieved. The search query for the message "Hello" has also been executed, and the results are as follows:" followed by a list of results:

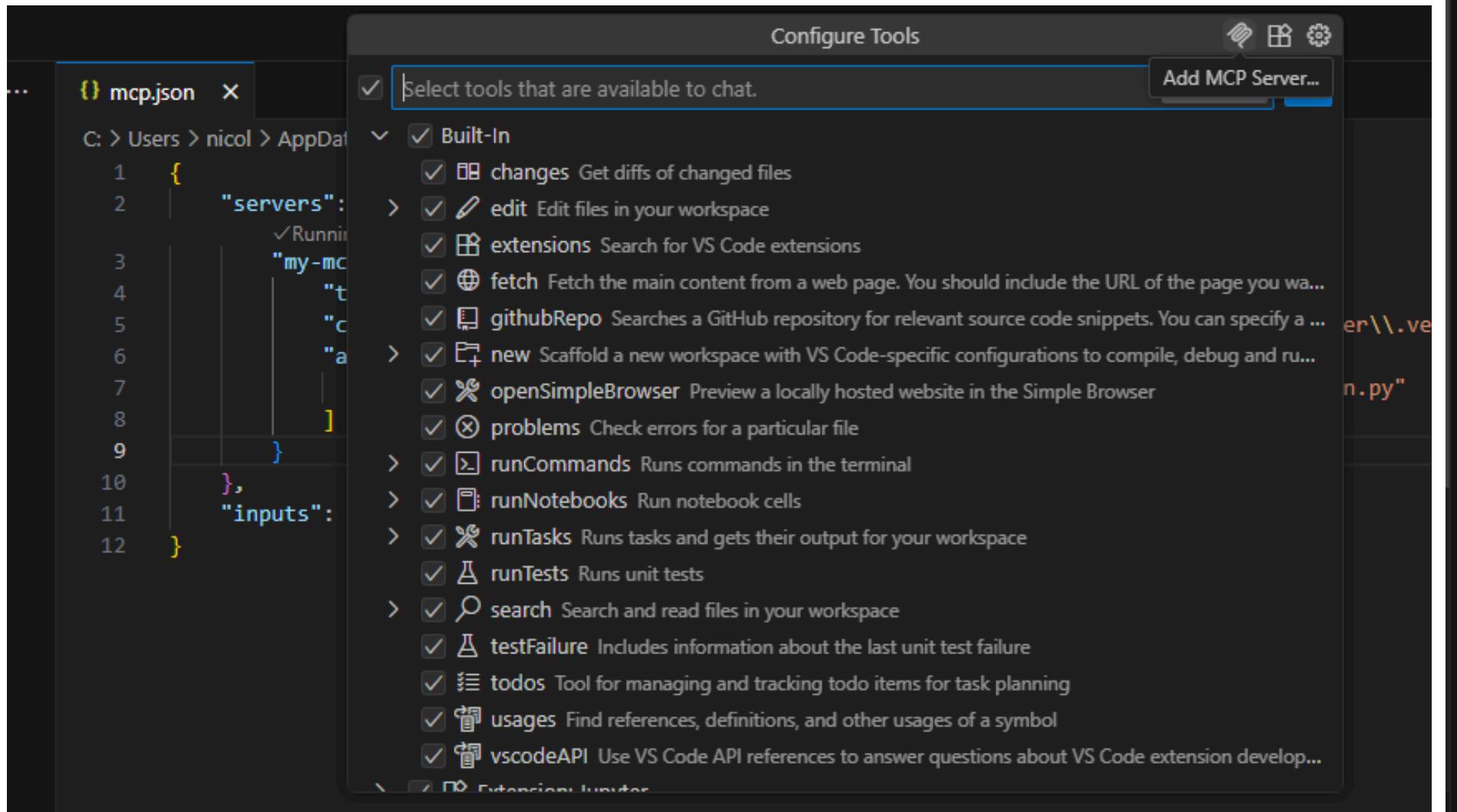
- Total hits: 1
- Max score: 0.2876821
- Index: my-index
- ID: 1
- Score: 0.2876821
- Timestamp: 2099-11-15T13:12:00
- Message: Hello world !

At the bottom right of the interface, there are several small icons for navigation and settings.

TD : appel du serveur

- Dans Copilot :

```
{
  "servers": {
    "my-mcp-server-5bf5d8ea": {
      "type": "stdio",
      "command": "C:\\Users\\nicol\\Documents\\epsi\\cours\\atelier_coder_ia\\mcp-server\\.venv\\Scripts\\python.exe",
      "args": [
        "C:\\Users\\nicol\\Documents\\epsi\\cours\\atelier_coder_ia\\mcp-server\\main.py"
      ]
    },
    "inputs": []
  }
}
```



CHAT

```
Ran get_mapping_elasticsearch - my-mcp-server-5bf5d8ea (MCP Server)
```

The mapping for the Elasticsearch index `my-index` is:

- `@timestamp`: date
- `message`: text

To search for documents where `message = "Hello"`, you need a query like:

```
{
  "query": {
    "match": {
      "message": "Hello"
    }
  }
}
```

Would you like me to run this search query on your Elasticsearch index?

yes

```
Ran search_elasticsearch - my-mcp-server-5bf5d8ea (MCP Server)
```

The search found 1 document in the `my-index` index where the message matches "Hello":

- `@timestamp`: 2099-11-15T13:12:00
- `message`: Hello world !

Let me know if you need to refine the query or process the results further.

⟳ ⌛ ⌂ ⌃

Add Context... mcp.json +

Add context (#), extensions (@), commands (/)

Agent Agent GPT-4.1 GPT-4.1