MCP Server-Client

with FastMCP

Complete Technical Guide

FastMCP • LangChain • OpenAI • SSE Transport

Table of Contents

1. What is FastMCP?

FastMCP is a Python library that makes it easy to create **MCP (Model Context Protocol) servers**.

Simple Explanation

Think of FastMCP as a **tool box builder**:
• You create tools (functions that do specific tasks)
• FastMCP wraps these tools into a server
• Other applications can connect to this server and use your tools

Key Features

Feature	Description
Simple Tool Creation	Use @mcp.tool() decorator to create tools
Multiple Transports	Supports SSE, stdio, HTTP
Auto Documentation	Automatically documents your tools
Type Safety	Uses Python type hints
Easy Integration	Works with LangChain, OpenAI, etc.

FastMCP vs Regular Functions

Regular Function (runs in same process):

```
def get_time():
    return datetime.now()

result = get_time()  # Direct call
```

FastMCP Tool (runs on server, accessible remotely):

```
@mcp.tool()
def get_time():
    return datetime.now()

mcp.run()  # Now accessible over network
```

2. Project Overview

This project demonstrates a **client-server architecture** using MCP with two tools:
• **get_current_date_time()** - Returns current date and time
• **web_search(query)** - Searches DuckDuckGo for information

Architecture Diagram

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  MCP Server (Port 8000)     ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■  ■
■  Tools:                     ■
■  • get_current_date_time()  ■
■  • web_search(query)        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
              ↕
    SSE Connection (Real-time)
              ↕
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■  MCP Client                 ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■  ■
■  • Connects to server       ■
■  • Uses OpenAI GPT-4o       ■
■  • Calls tools via AI       ■
■  • Returns answers          ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Project Structure

```
mcp-project/
■■■ server/
■   ■■■ mcp_server.py         # MCP Server with tools
■   ■■■ requirements.txt      # Server dependencies
■
■■■ client/
    ■■■ mcp_client.py         # MCP Client
    ■■■ .env                  # OpenAI API key
    ■■■ requirements.txt      # Client dependencies
```

3. How It Works - Simple Explanation

The Restaurant Analogy

Imagine you're at a restaurant:

1. **Server (Kitchen)**: Has tools to make food (date/time, web search)
2. **Client (Waiter)**: Takes your order and asks kitchen for help
3. **AI (Chef)**: Decides which tools to use based on your request

Step-by-Step Process

Step 1: Server Startup

You run: **python server/mcp_server.py**

Server does:
1. Creates FastMCP instance named 'TeluskoTools'
2. Registers 2 tools (date_time, web_search)
3. Starts SSE server on http://127.0.0.1:8000
4. Waits for client connections

Step 2: Client Connects

You run: **python client/mcp_client.py**

Client does:
1. Connects to server at http://127.0.0.1:8000/sse
2. Asks server: 'What tools do you have?'
3. Server responds: 'I have get_current_date_time and web_search'
4. Client stores these tools

Step 3: User Query Processing

Query: 'Current time in the country where messi visited in Dec 2025'

Step	Action	Component
1	AI thinks: "I need to search for Messi visit info"	AI (GPT-4o)
2	Calls web_search("messi visited december 2025")	Client → Server
3	Server executes DuckDuckGo search	Server
4	Returns: "Messi visited USA for Inter Miami"	Server → Client
5	AI thinks: "Now I need current time"	AI (GPT-4o)
6	Calls get_current_date_time()	Client → Server
7	Server gets system time	Server
8	Returns: "2025-12-29 10:30:45"	Server → Client
9	AI combines all information	AI (GPT-4o)
10	Returns: "Current time in USA is 10:30:45"	Client → User

4. Architecture

Transport: SSE (Server-Sent Events)

What is SSE?

• One-way communication from server to client
• Real-time updates
• Lightweight and simple
• Perfect for tool responses

Communication Flow

```
Client                          Server
  ■                               ■
  ■■■■■ Connect to /sse ■■■■■■■■→■
  ■                               ■
  ■←■■■ Connection Open ■■■■■■■■■■
  ■                               ■
  ■■■■■ Request Tool List ■■■■■■→■
  ■←■■■ Send Tool List ■■■■■■■■■■■
  ■                               ■
  ■■■■■ Call web_search ■■■■■■■■→■
  ■←■■■ Search Results ■■■■■■■■■■■
  ■                               ■
  ■■■■■ Call get_time ■■■■■■■■■■→■
  ■←■■■ Current Time ■■■■■■■■■■■■■
```

5. Installation & Setup

Prerequisites

• Python 3.8 or higher
• Internet connection (for web search)
• OpenAI API key

Step 1: Install Server Dependencies

```
cd server
pip install -r requirements.txt
```

What gets installed:
• fastmcp - MCP server framework
• langchain-community - LangChain tools
• duckduckgo-search - Web search functionality

Step 2: Install Client Dependencies

```
cd client
pip install -r requirements.txt
```

What gets installed:
• langchain-openai - OpenAI integration
• langchain-mcp-adapters - MCP client adapter
• python-dotenv - Environment variable management

Step 3: Configure Environment

```
cd client
cp .env.example .env
```

Edit .env and add your OpenAI API key:

```
OPENAI_API_KEY=sk-your-actual-key-here
```

6. Running the Application

■■ IMPORTANT: Run in This Order!

Terminal 1: Start the Server

```
cd server
python mcp_server.py
```

Expected Output:

```
INFO:     Started server process
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000
```

■ Server is ready when you see 'Uvicorn running'

Terminal 2: Run the Client

```
cd client
python mcp_client.py
```

Expected Output:

```
[Tool calls being made...]
The current time in the United States is 10:30:45 AM.
```

7. Server Code Explanation

Complete Server Code:

```
from fastmcp import FastMCP
from datetime import datetime
from langchain_community.tools import DuckDuckGoSearchRun

mcp = FastMCP("TeluskoTools")
search_tool = DuckDuckGoSearchRun()

@mcp.tool()
def get_current_date_time() -> str:
    """Get the current date and time."""
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")

@mcp.tool()
def web_search(query: str) -> str:
    """Search the web using DuckDuckGo."""
    return search_tool.run(query)

mcp.run(transport="sse", host="127.0.0.1", port=8000)
```

Line-by-Line Breakdown:

Code	Explanation
from fastmcp import FastMCP	Import FastMCP framework for creating MCP servers
from datetime import datetime	Import datetime to get current time
from langchain_community.tools...	Import DuckDuckGo search tool
mcp = FastMCP("TeluskoTools")	Create MCP server instance named "TeluskoTools"
search_tool = DuckDuckGoSearchRun()	Initialize DuckDuckGo search tool
@mcp.tool()	Decorator that registers function as MCP tool
def get_current_date_time():	Tool function - gets current date/time
return datetime.now()...	Returns formatted current time
def web_search(query: str):	Tool function - searches web with query
return search_tool.run(query)	Executes search and returns results
mcp.run(transport="sse"...)	Starts server on port 8000 with SSE transport

8. Client Code Explanation

Complete Client Code:

```
import asyncio
from langchain_openai import ChatOpenAI
from langchain_mcp_adapters.client import MultiServerMCPClient
from langchain_core.messages import HumanMessage, ToolMessage, SystemMessage

async def chat(query: str) -> str:
    client = MultiServerMCPClient({
       "telusko": {
        "transport": "sse",
        "url": "http://127.0.0.1:8000/sse"
       }
    })

    tools = await client.get_tools()
    llm = ChatOpenAI(model="gpt-4o")
    llm_with_tools = llm.bind_tools(tools)

    messages = [
        SystemMessage(content="Use tools when needed."),
        HumanMessage(content=query),
    ]

    while True:
        ai_msg = llm_with_tools.invoke(messages)
        messages.append(ai_msg)

        if not ai_msg.tool_calls:
            return ai_msg.content

        for tc in ai_msg.tool_calls:
            tool_fn = next(t for t in tools if t.name == tc["name"])
            tool_result = await tool_fn.ainvoke(tc["args"])
            messages.append(ToolMessage(content=str(tool_result),
                                       tool_call_id=tc["id"]))

response = asyncio.run(chat("Current time where messi visited Dec 2025"))
print(response)
```

Key Client Components:

Component	Purpose
MultiServerMCPClient	Connects to MCP server and gets tools
ChatOpenAI	Creates GPT-4o instance for AI processing
llm.bind_tools(tools)	Gives AI access to MCP tools
SystemMessage	Instructions for AI behavior
HumanMessage	User query to be processed
while True loop	Continues until AI has final answer
ai_msg.tool_calls	List of tools AI wants to execute
tool_fn.ainvoke()	Executes tool on server asynchronously
ToolMessage	Adds tool results to conversation
asyncio.run()	Runs async chat function

9. Complete Flow Diagram

Detailed Flow with Example Query

Query: 'Current time in the country where messi visited in Dec 2025'

```
Step 1: Client Initialization
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ client = MultiServerMCPClient(...)    ■
■ → Connects to http://127.0.0.1:8000    ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ tools = await client.get_tools()      ■
■ → Server sends: [get_current_date_time■
■                   web_search]          ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Step 2: First AI Call
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ llm_with_tools.invoke(messages)       ■
■ AI decides: "Need to search for       ■
■              Messi visit info"         ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
                ↓
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ AI Response includes tool_calls:       ■
■ [{                                     ■
■   name: "web_search",                  ■
■   args: {query: "messi dec 2025"}     ■
■ }]                                     ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Step 3: Execute First Tool
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Client: tool_fn.ainvoke(args)         ■
■ ↓                                      ■
■ Server: web_search("messi...")        ■
■ ↓                                      ■
■ Returns: "Messi visited USA"          ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Step 4: Second AI Call
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ llm_with_tools.invoke(messages)       ■
■ AI decides: "Now need current time"   ■
■ tool_calls: [get_current_date_time]   ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Step 5: Execute Second Tool
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Server: get_current_date_time()       ■
■ Returns: "2025-12-29 10:30:45"        ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

Step 6: Final AI Response
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ AI has all info:                       ■
■ • Country: USA                        ■
■ • Time: 10:30:45                      ■
■ Returns: "Current time in USA is      ■
■          10:30:45 AM"                 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Key Concepts Summary

Concept	What	Why	How
FastMCP	Framework for MCP servers	Makes tool creation simple	Use @mcp.tool() decorator
SSE	Server-Sent Events	Real-time, lightweight	Client connects, server sends
Async/Await	Non-blocking execution	Don't freeze while waiting	Use async def and await
Tool Calling	AI decides which tools	AI handles complex tasks	AI analyzes, calls, processes
Message History	Conversation context	AI needs full context	Append each message/response