**1. What is MCP?**

**MCP = Model Context Protocol**

* It's like a **USB-C port for AI** — a **standard way to connect tools/data to AI models** (especially LLMs).
* It helps you expose:
  + **Resources** = Data (like a GET API)
  + **Tools** = Actions (like a POST API)
  + **Prompts** = Reusable messages for AI

🧠 **Think of MCP as a special API system for AI to read data and do actions.**

**⚡ 2. Why FastMCP?**

* MCP is powerful but complex.
* **FastMCP makes it easier** to build MCP servers.
* You can just **decorate functions** in Python and it handles the rest.

**Benefits:**

* 🚀 Fast to build
* 🍀 Simple and clean
* 🐍 Python-friendly
* 🔍 Complete tools for production

**🛠️ 3. Installation**

Use uv to install (recommended):

bash

uv pip install fastmcp

More options are available in the Installation Guide.

**🧩 4. Core Concepts**

These are key ideas in FastMCP:

**a. FastMCP Server**

* It's the main object where everything is registered (like tools/resources/prompts).

python

from fastmcp import FastMCP

mcp = FastMCP(name="MyAssistantServer")

**b. Tools**

* Allow AI to **perform actions** (e.g., multiply numbers).

python

@mcp.tool

def multiply(a: float, b: float) -> float:

return a \* b

**c. Resources & Templates**

* Provide **read-only data** (like config info or user profiles).

python

@mcp.resource("config://version")

def get\_version():

return "2.0.1"

@mcp.resource("users://{user\_id}/profile")

def get\_profile(user\_id: int):

return {"name": f"User {user\_id}", "status": "active"}

**d. Prompts**

* Let you define **reusable message templates** for the AI.

python

@mcp.prompt

def summarize\_request(text: str) -> str:

return f"Please summarize the following text:\n\n{text}"

**e. Context**

* Gives you access to **MCP session features** like:
  + Logging
  + Calling other tools
  + Reading resources
  + Progress reporting

python

@mcp.tool

async def process\_data(uri: str, ctx: Context):

await ctx.info(f"Processing {uri}...")

data = await ctx.read\_resource(uri)

summary = await ctx.sample(f"Summarize: {data.content[:500]}")

return summary.text

**🔗 5. MCP Clients**

* **Connect** to any MCP server and **use its tools/resources**.

python

from fastmcp import Client

async def main():

async with Client("my\_server.py") as client:

result = await client.call\_tool("add", {"a": 5, "b": 3})

print(result.text)

* Supports:
  + stdio (local Python scripts)
  + http or sse (for web)

You can even connect to **multiple servers at once**.

**🧠 6. Advanced Features**

**a. Proxy Servers**

* Add logic on top of another MCP server using .as\_proxy().

**b. Composing Servers**

* Combine multiple MCP servers into one using mcp.mount().

**c. OpenAPI / FastAPI Conversion**

* Convert an existing OpenAPI/FastAPI app into an MCP server.

**🔐 7. Authentication & Security**

FastMCP supports secure access:

* Secure your server endpoints
* Clients can handle login/keys securely
* Supports enterprise-grade auth

**▶️ 8. Running Your Server**

Create a file server.py:

python

from fastmcp import FastMCP

mcp = FastMCP("Demo")

@mcp.tool

def hello(name: str) -> str:

return f"Hello, {name}!"

if \_\_name\_\_ == "\_\_main\_\_":

mcp.run()

You can run the server using:

bash

python server.py

Supports:

* stdio (default)
* http (web)
* sse (streaming)

**🤝 9. Contributing**

**Prerequisites:**

* Python 3.10+
* uv for virtual environments

**Setup:**

bash

git clone https://github.com/jlowin/fastmcp.git

cd fastmcp

uv sync

Activate the virtual environment and you're ready.

**Tests:**

bash

pytest # or for coverage:

uv run pytest --cov=src --cov=examples --cov-report=html

**Code checks:**

bash

uv run pre-commit install

uv run pre-commit run --all-files

**Pull Requests:**

1. Fork repo
2. Create a branch
3. Make changes (with tests/docs)
4. Push and create PR

**✅ Summary**

| **Feature** | **Purpose** |
| --- | --- |
| **FastMCP** | Tool to easily build MCP-compatible AI APIs |
| **Tools** | Let AI do actions |
| **Resources** | Let AI read data |
| **Prompts** | Help AI generate smart messages |
| **Context** | Advanced features: logging, sampling, etc. |
| **Client** | Connect to and use any MCP server |
| **Run** | Use mcp.run() to launch your AI server |
| **Secure** | Built-in authentication support |
| **Easy Dev** | Decorators + async = minimal effort |

**What Are Transports in FastMCP?**

**Transports** are the way your FastMCP server communicates with clients (like LLMs or other tools). Think of them as **communication channels** between your server and the outside world.

FastMCP supports multiple transport methods:

**🖥️ 1. STDIO (Standard Input/Output)**

* **Default transport**
* Best for local command-line tools or when using with LLMs that support STDIO (like OpenAI function calling via tools).
* Communicates through the **terminal's input/output** streams.

**✅ When to Use:**

* Local development
* Simple CLI-based tools
* LLMs that connect via stdio (common in LangChain and LangGraph)

**🧪 Example:**

python

mcp.run() # This uses stdio by default

Or explicitly:

python

mcp.run(transport="stdio")

**🌐 2. HTTP Transport**

* Runs a web server (FastAPI-based) to handle requests over HTTP.
* Useful for **web-based AI clients**, **frontend apps**, or public APIs.

**✅ When to Use:**

* Deploying on web
* Connecting over browser or network
* Integration with REST APIs or external services

**🧪 Example:**

python

mcp.run(transport="http", host="127.0.0.1", port=8000, path="/mcp")

**🔁 3. SSE (Server-Sent Events)**

* A streaming protocol that **pushes events from server to client**.
* Often used when AI models or tools need **real-time streaming updates**.

**✅ When to Use:**

* Clients expect continuous updates
* Integration with SSE-compatible systems

**🧪 Example:**

python

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

**📊 Summary Table**

| **Transport** | **Use Case** | **Advantages** | **Code Example** |
| --- | --- | --- | --- |
| stdio | Local CLI, LLM toolchains | Fast, easy for local dev | mcp.run() |
| http | Web APIs, frontend apps | Network accessible | mcp.run(transport="http", ...) |
| sse | Streaming events to AI | Real-time, streaming | mcp.run(transport="sse", ...) |