Building Real-World Agentic APIs with MCP: A Practical Guide

1. Why MCP? (What Can It Do That LLMs Alone Can't?)

Imagine you want to:

- **Search the web for real-time information**
- **Write directly to a document file on your computer**
- **Ask for the next train departure using the NS (Dutch Railways) API**

A normal LLM (like ChatGPT or Claude) can't do these things on its own—it can only answer from its training data. But with MCP, you can expose real tools (Python functions, APIs, etc.) that the LLM can call to get live answers, interact with files, or use any API you want.

2. The Agentic Flow: How Tool Calls Work

An MCP tool call is, at its core, just a function call. In LangGraph, you can test this with just a ToolNode and a tool function—no need to build a full graph for simple cases.

Minimal Example:

```
# 1. Define your tool
async def get_next_train(station: str) -> str:
    return "Next train to Utrecht departs at 14:32 from platform 5"

# 2. Register the tool with a ToolNode
from langgraph.prebuilt import ToolNode
from langchain_core.messages import HumanMessage

node = ToolNode(tools=[get_next_train])

# 3. Run the node and stream events
state = {"messages": [HumanMessage(content="What's the next train from Amsterdam Centraal?")]}
async for event in node.astream_events(state):
    print(event)  # Shows tool call, tool result, and final agent message events
```

What you see in the event stream:

- **on_tool_start:** Shows the tool name and the arguments passed.

```
    station: Amsterdam Centraal
```

- **on_tool_end:** Shows the direct output from your tool.

```
Next train to Utrecht departs at 14:32 from platform 5
```

- **on_chain_end:** Shows the final message the agent sends to the user (may be rephrased or formatted).

```
The next train to Utrecht departs at 14:32 from platform 5
```

This is the most direct way to see the chronological handling of a tool call in LangGraph—just a node, a tool, and the output events.

3. Setting Up MCP Tools and Environments

You can get MCP servers from websites like Smithery or other open-source projects. Here's how to get started with different types of tools and environments:

Example: Installing the NS MCP Server (with npx)

Many MCP servers can be installed and run with a single command using `npx` (which handles environment setup and dependencies for you):

```
npx @smithery-ai/ns-mcp-server --stdio
```

This command downloads and runs the NS MCP server, creating a virtual environment in the background so you don't have to manage packages manually.

Example: Setting Up a Python MCP Server (FastMCP)

```
python -m venv .venv
source .venv/bin/activate  # On Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

```
from fastmcp import FastMCP

mcp = FastMCP("Demo 🚀")

@mcp.tool
def add(a: int, b: int) -> int:
    """Add two numbers"""
```

```
    return a + b

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

Start it with:

```
python my_mcp_server.py
```

Specifying MCP Servers and Environment Variables in Config Files

You can also manually edit a config file (like `claude_desktop_config.json`), or Cursor's config file, to specify the MCP server location and any needed environment variables:

```
{
  "mcpServers": {
    "ns-local": {
      "command": "npx",
      "args": [
        "@smithery-ai/ns-mcp-server",
        "--stdio"
      ],
      "env": {
        "NS_API_KEY": "your-ns-api-key-here"
      }
    }
  }
}
```

4. How LLMs Connect to MCP Servers

- **Local MCP Server:** The AI app (like Cursor or Claude Desktop) is configured to connect to a local process (your script), often via a config file or UI setting. Example: you run your server, and the app connects to `localhost` or launches your script directly.
- **Remote MCP Server:** The AI app is given the URL of your server (e.g., `https://mcp.mycompany.com/mcp`). The app connects over the internet, often with OAuth authentication.
- **Bridged/Hybrid:** A local "bridge" process forwards requests to a remote server, useful for compatibility.

In all cases: The AI app needs to know where to find your MCP server—either by launching it, connecting to a local address, or using a remote URL. For example, with FastMCP running locally, any

AI app that supports MCP can connect as long as it's configured to look for a local MCP server (such as in `claude_desktop_config.json` or Cursor's config file).

Minimal LangGraph client example:

Suppose you want to use tools from someone else's MCP server (not just your own). You can use `bind_tools` to connect to a remote MCP server and then use a `ToolNode` to invoke those tools.

```
from langgraph.prebuilt import ToolNode
from langchain_openai import AzureChatOpenAI

# 1. Bind tools from a remote MCP server (e.g., Smithery or a friend's server)
llm = AzureChatOpenAI(...)
remote_tools = llm.bind_tools(mcp_url="https://mcp.someone-else.com/mcp")

# 2. Register the remote tools with a ToolNode
node = ToolNode(tools=remote_tools)

# 3. Invoke a tool from the remote MCP server
result = await node.ainvoke({"messages": ["What's the next train from Amsterdam Centraal?"]})
print(result)
```

By following this guide, you'll be ready to build, run, and share powerful agentic APIs with MCP