Building a Local MCP Server

Quick Summary

- We will create a local MCP server named `local-demo`.
- We will enforce **Python 3.12+** using both `requires-python` and `uv python pin`.
- We will test with **MCP Inspector** first, then connect to **Claude Desktop**.

1. Introduction

This document provides a **complete and practical guide** to building a local **Model Context Protocol (MCP)** server using **Python** and the **FastMCP** framework.

It is written for:

- Windows systems
- Machines with **multiple Python versions installed**
- Local testing using **MCP Inspector** and integration with **Claude Desktop**

Outcome

By the end, you will have a working MCP server and a verified test workflow (Inspector $\rightarrow$ Claude).

2. What the MCP Server Provides

2.1. Tools (Callable Actions)

- `add(a, b)` | Adds two integers and returns the result
- `echo(text)` | Returns the provided text unchanged

2.2. Resource (Read-only Data by URI)

- `greeting://{name}` | Returns a greeting message dynamically

Tools vs Resources

Tools are actions Claude can call (most reliable in Claude Desktop).
Resources are data fetched by URI (some clients may not auto-fetch them).

3. System Requirements

- Windows 10 or Windows 11
- Python **3.12.x** installed (required)
- Internet access
- Claude Desktop (optional but recommended)

4. Install uv (One-Time Setup)

uv manages Python versions, virtual environments, and dependencies.

4.1. Installation

```
irm https://astral.sh/uv/install.ps1 | iex
```

Verify:

```
uv --version
```

5. Project Initialization

Create and enter a project directory:

```
mkdir "D:\Test Tools\testing"
cd "D:\Test Tools\testing"
```

Initialize:

```
uv init
```

This creates `pyproject.toml`.

6. Enforce Python 3.12+ (CRITICAL)

Why this matters

If you have multiple Python versions installed, uv (or Claude) may accidentally select an older version. This can cause **environment recreation**, **dependency mismatch**, or **server disconnect** issues.

6.1. Set `requires-python` in `pyproject.toml`

Open the file:

```
notepad pyproject.toml
```

Under `[project]`, ensure:

```
[project]
requires-python = ">=3.12"
```

6.2. Pin the Python version

List Python versions:

```
uv python list
```

Pin Python 3.12:

```
uv python pin 3.12
```

Verify:

```
type .python-version
```

Expected:

```
3.12
```

7. Create the Virtual Environment

Remove any previous venv:

```
Remove-Item -Recurse -Force .venv
```

Create a new venv:

```
uv venv
```

Verify Python version used by the project:

```
uv run python -V
```

Expected

You should see `Python 3.12.x`. If you see 3.9/3.10, re-check `requires-python` and `.python-version`.

8. Install MCP SDK and CLI

Install FastMCP + CLI utilities:

```
uv add "mcp[cli]"
```

Verify:

```
uv run python -c "from mcp.server.fastmcp import FastMCP; print('FastMCP
   OK')"
```

Check MCP CLI:

```
uv run mcp --help
```

9. Implement the MCP Server

Create `server.py`:

```
notepad server.py
```

9.1. Server Code

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("local-demo")

@mcp.tool()
def echo(text: str) -> str:
   """Echo␣text␣back."""
   return text

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

@mcp.resource("greeting://{name}")
def greeting(name: str) -> str:
   """Return␣a␣greeting␣message."""
   return f"Hello,␣{name}!"

if __name__ == "__main__":
   # STDIO transport (required for Inspector + Claude Desktop)
   mcp.run()
```

STDIO Note

Running `server.py` directly may show a traceback if no client is attached. That is normal for STDIO servers. Always test using Inspector or Claude Desktop.

10. Test with MCP Inspector

Always test here frst.

Start Inspector:

```
uv run mcp dev server.py
```

10.1. Required Tests

Tool: add

```
{ "a": 10, "b": 25 }
```

Tool: echo

```
{ "text": "hello" }
```

Resource (manual fetch)

```
greeting://Mehedi
```

Expected:

```
Hello, Mehedi!
```

Resource Listing Tip

Templated resources (like `greeting://{name}`) may not appear in a list automatically. In Inspector, test them by manually reading a concrete URI such as `greeting://Mehedi`.

11. Connect to Claude Desktop

11.1. Correct Configuration

Important

Do not point Claude to `uv.exe` (especially if it comes from Python 3.9). Use the project venv Python directly.

Update `claude_desktop_config.json`:

```
{
  "mcpServers": {
    "local-demo": {
      "command": "D:\\Test Tools\\testing\\.venv\\Scripts\\python.exe",
      "args": ["D:\\Test Tools\\testing\\server.py"]
    }
  }
}
```

11.2. Restart Claude Desktop Fully

- Close Claude Desktop
- Open Task Manager
- End all Claude processes
- Reopen Claude Desktop

12. Test in Claude Desktop

Use explicit prompts:

```
Use the add tool with a=12 and b=30
```

```
Call the echo tool with text "MCP works"
```

Resources in Claude

Claude Desktop may discover resources but not reliably fetch them by prompt. If you need the same functionality, expose it as a tool (e.g., `get_greeting(name)`).

13. Common Issues and Solutions

Server Disconnected

Cause: Wrong interpreter used (often Python 3.9).
Fix: Use `.venv\Scripts\python.exe` in Claude confg.

Traceback on Startup

Cause: STDIO server has no client attached.
Fix: Use Inspector or Claude Desktop (normal behavior).

Resources Missing in Claude

Cause: Client behavior; resources are not always auto-used.
Fix: Provide equivalent functionality as a tool for reliability.

14. Final Verifcation Checklist

- `requires-python = ">=3.12"` is set in `pyproject.toml`
- `.python-version` is pinned to `3.12`
- `uv run python -V` shows Python 3.12.x
- Inspector tests pass: `uv run mcp dev server.py`
- Claude Desktop uses venv Python and can call tools

Success

Your MCP server is correctly implemented and verifed.

15. New Tool Addition

After successfully building and integrating the basic MCP server, the following enhancements can be implemented to transform it into a more powerful and production-ready system.

15.1. DuckDuckGo Search Tool

Install:

```
uv add duckduckgo-search
```