

lab

April 3, 2026

1 Lab 12 — MCP Concepts & Protocol

Explore the MCP protocol: connect to servers, call tools, and survey the ecosystem.

Exercises: 1. Connect to an MCP server and list its tools 2. Call a tool via the MCP protocol 3. Survey 5 popular MCP servers

```
[ ]: # Setup
import json
import asyncio
from dotenv import load_dotenv

load_dotenv()

# We'll simulate MCP interactions with mock classes for this lab.
# In production, you'd use: from mcp import ClientSession, StdioServerParameters
print("Setup complete.")
```

```
[ ]: # Mock MCP Server (simulates protocol responses)

class MockMCPTool:
    def __init__(self, name, description, input_schema):
        self.name = name
        self.description = description
        self.inputSchema = input_schema

class MockMCPResult:
    def __init__(self, tools):
        self.tools = tools

class MockToolContent:
    def __init__(self, text):
        self.type = "text"
        self.text = text

class MockToolResult:
    def __init__(self, content):
        self.content = content
```

```

class MockMCPSession:
    """Simulates an MCP ClientSession for learning purposes."""
    def __init__(self):
        self._tools = [
            MockMCPTool("read_file", "Read the contents of a file",
                {"type": "object", "properties": {"path": {"type": "string"}}, "required": ["path"]}),
            MockMCPTool("list_directory", "List files in a directory",
                {"type": "object", "properties": {"path": {"type": "string"}}, "required": ["path"]}),
            MockMCPTool("search_files", "Search for files matching a pattern",
                {"type": "object", "properties": {"pattern": {"type": "string"}, "path": {"type": "string"}}, "required": ["pattern"]}),
        ]
        self._files = {
            "/tmp/test.txt": "Hello from MCP!\nThis is a test file.",
            "/tmp/config.json": '{"key": "value", "debug": true}',
        }

    async def initialize(self):
        return {"protocolVersion": "2025-03-26", "capabilities": {"tools": {}}}

    async def list_tools(self):
        return MockMCPResult(self._tools)

    async def call_tool(self, name, arguments):
        if name == "read_file":
            path = arguments.get("path", "")
            content = self._files.get(path, f"Error: file not found: {path}")
            return MockToolResult([MockToolContent(content)])
        elif name == "list_directory":
            return MockToolResult([MockToolContent("test.txt\nconfig.json")])
        elif name == "search_files":
            return MockToolResult([MockToolContent(f"Found: /tmp/test.txt")])
        return MockToolResult([MockToolContent(f"Unknown tool: {name}")])

print("Mock MCP session ready.")

```

1.1 Exercise 1 — MCP Client: Connect & List Tools

Connect to the mock MCP server and discover its capabilities.

```

[ ]: # TODO: Create a MockMCPSession, call initialize(), then list_tools()
      # Print each tool's name, description, and input schema

```

```
raise NotImplementedError("Exercise 1: connect and list tools")
```

1.2 Exercise 2 — Call a Tool via MCP

Execute tool calls through the MCP protocol and handle results.

```
[ ]: # TODO: Call read_file with path "/tmp/test.txt" and print the result
      # TODO: Call list_directory and print the result
      # TODO: Try calling read_file with a non-existent path and handle the error

      raise NotImplementedError("Exercise 2: call tools via MCP")
```

1.3 Exercise 3 — MCP Server Survey

Research 5 popular MCP servers and document their capabilities.

```
[ ]: # TODO: Create a comparison table of 5 MCP servers
      # For each, document: name, source, tools, resources, transport, use case
      # Present as a formatted table (you can use a dict or pandas DataFrame)

      raise NotImplementedError("Exercise 3: survey MCP servers")
```