SAMIR GENAI ACADEMY

PROFESSIONAL STUDY GUIDE SERIES

Phase 03 · Agentic AI Engineering

15

Model Context Protocol (MCP)

Resources, Tools, Prompts & Sampling — the new agent standard

Module 15 of 25 · **1 hour**

Instructor: **Samir Kumar** · Edition 2026

STUDY GUIDE

01 Overview

> Resources, Tools, Prompts & Sampling — the new agent standard. This guide gives you the core theory, the tools that matter, a hands-on build, working code, and the pitfalls to avoid — everything you need to learn the topic to a professional standard and apply it in real projects.

1 hour	4 + 4	3	15/25
LESSON LENGTH	CONCEPTS + BUILD STEPS	FRAMEWORKS & TOOLS	PHASE 03

02 Learning Objectives

- Explain MCP and why a standard for agent-tool connections matters.
- Distinguish Resources, Tools, Prompts, and Sampling.
- Build a custom MCP server exposing data and actions.
- Connect an MCP server to an agent/host.

03 Core Concepts

Why MCP

Model Context Protocol is an open standard (the 'USB-C for AI') that lets any compliant host connect to any compliant server. Instead of bespoke integrations per tool per app, you build an MCP server once and every MCP-aware agent can use it.

Primitives

Servers expose four primitives: Resources (read-only data the model can load), Tools (actions the model can invoke), Prompts (reusable templated workflows), and Sampling (server asks the host's model to complete a sub-task). These cover read, act, and reuse.

Architecture

A host (IDE, chat app, agent) runs MCP clients that connect to servers over stdio or HTTP. Capability negotiation on connect tells each side what's available, keeping the protocol extensible and safe.

Building servers

SDKs (Python/TypeScript) make a server a few decorators: declare tools and resources, run the transport. The same server then works across Claude Desktop, IDEs, and custom agents.

04 Visual Diagram

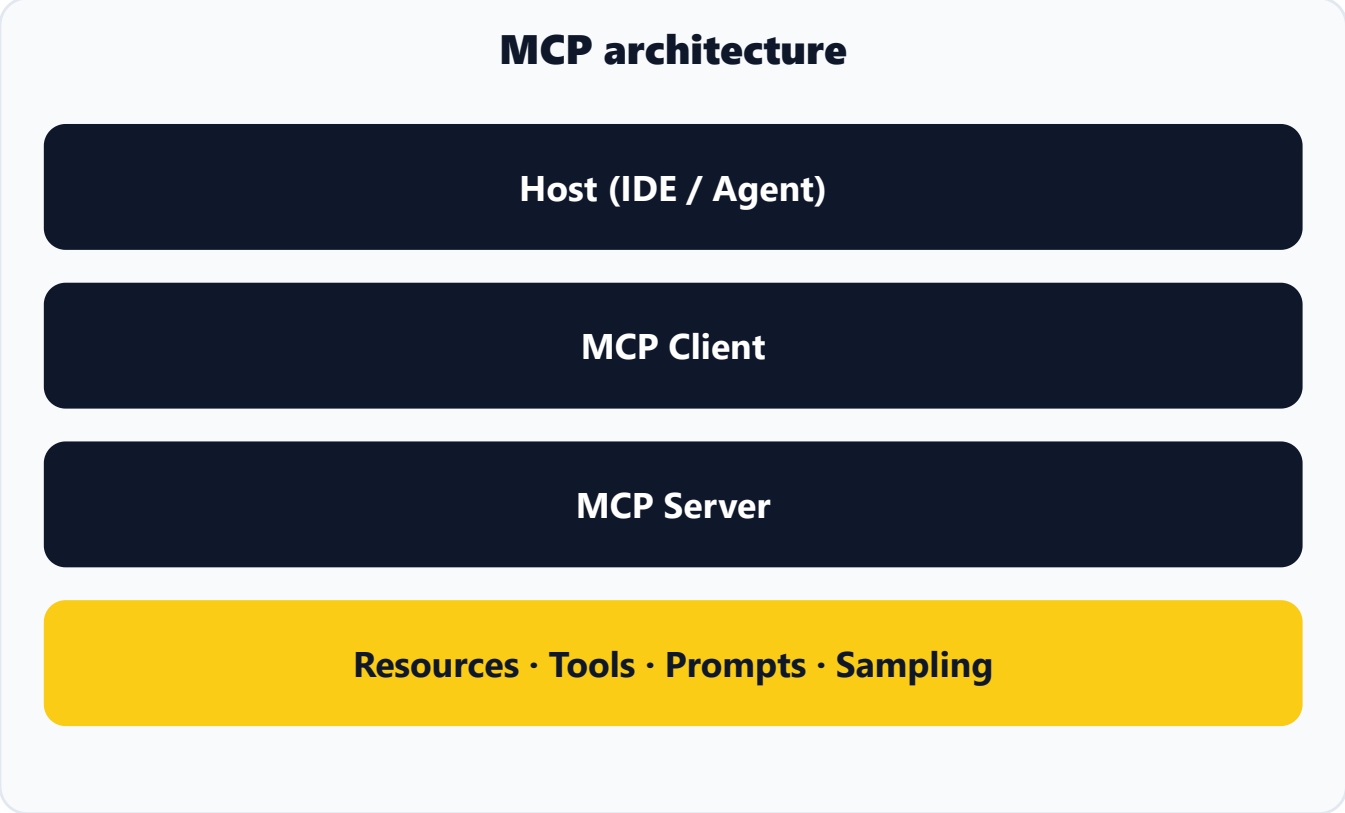

Figure 15 — A host's client connects to any MCP server. Build the server once; every MCP-aware app can use it.

05 Key Frameworks & Tools

MCP Python SDK (FastMCP) | MCP TypeScript SDK | MCP Inspector

06 Hands-On Build

🛠 Custom MCP Server for API/Database access

1. Scaffold an MCP server with the Python SDK.
2. Expose a Resource (read records) and a Tool (perform an action).
3. Test it with MCP Inspector.
4. Connect it to an MCP-aware host and call it from an agent.

07 Code Walkthrough

A minimal MCP server with FastMCP — PYTHON

```python
from mcp.server.fastmcp import FastMCP

mcp = FastMCP("orders")

@mcp.tool()
def create_order(sku: str, qty: int) -> str:
    """Create an order and return its id."""
    return f"order-{sku}-{qty}"

@mcp.resource("orders://recent")
def recent_orders() -> str:
    """Return recent orders as text."""
    return "order-ABC-2, order-XYZ-5"

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

08 🧪 Lab Assistant

Lab 15 · Build a custom MCP server

Prerequisites:pip install mcp · MCP Inspector

1. **Declare one @mcp.tool() and one @mcp.resource()**
   > Expected: The server exposes an action and read-only data
2. **Run it and open MCP Inspector**
   > Expected: Tool and resource are listed and callable
3. **Connect it to an MCP host and invoke from an agent**
   > Expected: The agent calls your tool

Troubleshooting

- Inspector can't connect → check the transport (stdio/HTTP).
- Tool not found → confirm the decorator and a docstring.

09 Pitfalls & Key Takeaways

⚠ COMMON PITFALLS

- Exposing destructive tools without auth or confirmation.
- Confusing Resources (read) with Tools (act).
- Skipping the Inspector and debugging blind.

✓ KEY TAKEAWAYS

- MCP standardises agent-to-tool connections — build once, reuse everywhere.
- Resources read, Tools act, Prompts template, Sampling delegates.

- SDKs make servers a few decorators; Inspector verifies them.

📺 **Companion video:** Watch the module 15 lesson in your Student Dashboard, then use this guide to revise, copy the code, and complete the build. Download both for offline study.

10 Further Resources

modelcontextprotocol.io spec • MCP Python/TS SDKs • MCP Inspector