

Lab 020: Construa um Servidor MCP em Python

Nível: L200 Caminho: ? MCP Tempo: ~45 min ? Custo: Gratuito ? Executa localmente, sem necessidade de conta na nuvem

O Que Você Vai Aprender

Como construir um servidor MCP do zero usando FastMCP (Python) Como definir Tools com esquemas e descrições adequados Como executar o servidor e conectá-lo ao MCP Inspector e ao GitHub Copilot (VS Code) Como adicionar transporte HTTP/SSE para compatibilidade com agentes na nuvem

Introdução

Um servidor MCP é um programa que expõe ferramentas (funções) para agentes de IA por meio do protocolo MCP. Quando um agente precisa realizar uma ação ? consultar um banco de dados, chamar uma API, ler um arquivo ? ele chama as ferramentas do seu servidor MCP.

Neste laboratório, construímos um servidor MCP de busca de produtos com duas ferramentas:

1. `list_categories` ? retorna categorias de produtos 2. `search_products` ? busca produtos por palavra-chave

Configuração de Pré-requisitos

[diagram or code sample]

Certifique-se de ter Python 3.10+: [diagram or code sample]

tip "Início Rápido com GitHub Codespaces" Open in GitHub Codespaces

Todas as dependências já estão pré-instaladas no devcontainer.

? Arquivos de Apoio

note "Baixe estes arquivos antes de iniciar o laboratório" Salve todos os arquivos em uma pasta lab-020/ no seu diretório de trabalho.

Arquivo | Descrição | Download | `outdoorgear_mcp_server_starter.py` | Script inicial com TODOs | ? Download |

Exercício do Laboratório

Passo 1: Crie o projeto

[diagram or code sample]

Crie `server.py`:

[diagram or code sample]

Passo 2: Defina suas Tools

Adicione as ferramentas após a lista `PRODUCTS`:

[diagram or code sample]

Passo 3: Execute o servidor (modo stdio)

Adicione o ponto de entrada no final de `server.py`:

[diagram or code sample]

Execute: [diagram or code sample]

O servidor agora está ouvindo em stdio. Este é o modo padrão para ferramentas locais.

Passo 4: Teste com o MCP Inspector

Abra um novo terminal e execute: [diagram or code sample]

O Inspector será aberto no seu navegador. Experimente:

1. Clique em "Tools" para ver suas três ferramentas 2. Clique em `list_categories` ? "Run tool" ? veja as categorias 3. Clique em `search_products` ? preencha keyword: "tent" ? "Run tool"

success "Você deverá ver a Camping Tent nos resultados"

Passo 5: Execute como servidor HTTP/SSE (para agentes remotos)

Para agentes hospedados na nuvem como Microsoft Foundry, precisamos de transporte HTTP/SSE.

Adicione a opção de inicialização:

[diagram or code sample]

Execute no modo HTTP: [diagram or code sample]

Você verá: [diagram or code sample]

Teste com curl: [diagram or code sample]

Passo 6: Conecte ao GitHub Copilot no VS Code

1. No VS Code, crie .vscode/mcp.json no seu workspace:

[diagram or code sample]

2. Abra o GitHub Copilot Chat no VS Code 3. Digite: @copilot What product categories are available?

O GitHub Copilot chamará sua ferramenta list_categories e incluirá o resultado na resposta! tip "Suporte a MCP no VS Code" Certifique-se de ter a extensão GitHub Copilot versão 1.99+ instalada. Pode ser necessário habilitar o MCP nas configurações do VS Code:

"chat.mcp.enabled": true

Adicionando um Resource (Bônus)

O MCP também suporta Resources ? dados que o agente pode ler. Adicione um recurso que expõe o catálogo completo de produtos:

[diagram or code sample]

? Arquivo Inicial

Este laboratório inclui um arquivo inicial com marcadores TODO para guiá-lo na construção do servidor:

[diagram or code sample]

[diagram or code sample]

O arquivo inicial contém o catálogo de produtos OutdoorGear (P001?P007) já preenchido. Você implementa: list_categories, search_products, get_product_details e uma ferramenta desafio compare_products.

? Desafio: Adicione uma Tool compare_products

Depois que suas 3 ferramentas básicas estiverem funcionando, adicione uma quarta:

[diagram or code sample]

Teste no MCP Inspector perguntando: *"Compare a TrailBlazer Tent 2P e a TrailBlazer Solo.

Qual é mais leve?"*

O agente deverá chamar compare_products(["P001", "P003"]) e retornar uma comparação estruturada.

Resumo

Você construiu um servidor MCP totalmente funcional que:

? Define 3 ferramentas com descrições adequadas (o LLM usa essas descrições para decidir quando chamar) ? Executa em modo stdio para ferramentas locais ? Executa em modo HTTP/SSE para agentes remotos ? Funciona com o MCP Inspector para testes ? Integra com o GitHub Copilot no VS Code

Próximos Passos

Versão em C#: ? Lab 021 ? Servidor MCP em C# Conectar ao Microsoft Foundry Agent Service: ?

Lab 030 ? Foundry Agent Service + MCP Adicionar consultas reais a banco de dados: ? Lab 031

? pgvector Busca Semântica

Modernização da Fase 3

warning "Verified 2026-10-27" Adote a especificação MCP 2026-07-28. Prefira servidor stateless, discovery, SDK Tier 1 atual e estado multi round-trip explícito. Adicione

Registry e autorização somente quando o modelo de confiança exigir.

[diagram or code sample]

Model Context Protocol specification MCP authorization MCP Registry

Kit de apoio

O build do site gera downloads estáveis em PDF e ZIP. O ZIP contém o guia do lab e todos os arquivos de apoio declarados para este lab.

Baixar PDF gerado Baixar ZIP gerado

Supporting file: `outdoorgear_mcp_server_starter.py`

""" `outdoorgear_mcp_server_starter.py` ? OutdoorGear MCP Server Starter

Lab 020: Build an MCP Server in Python

This starter file contains TODO markers where you need to write the code. Follow the instructions in each TODO to complete the lab.

Prerequisites: pip install fastmcp

Run with: `python outdoorgear_mcp_server_starter.py` """

`from fastmcp import FastMCP`

===== Server initialization

=====

TODO 1: Initialize the FastMCP server with: `name = "outdoorgear-catalog"` `description = "OutdoorGear Inc. product catalog MCP server"` `mcp = FastMCP(...)` `mcp = None` # Replace this line with the FastMCP initialization

===== Data: OutdoorGear product

`catalog` =====

```
PRODUCTS = [ { "id": "P001", "name": "TrailBlazer Tent 2P", "category": "Tents",
"description": "Lightweight 2-person backpacking tent, 3-season, aluminum poles",
"price_usd": 249.99, "weight_grams": 1800, "in_stock": True, "season": "3-season", }, {
"id": "P002", "name": "Summit Dome 4P", "category": "Tents", "description": "4-season
expedition tent for 4 people, steel poles, snow rated", "price_usd": 549.99,
"weight_grams": 3200, "in_stock": True, "season": "4-season", }, { "id": "P003", "name":
"TrailBlazer Solo", "category": "Tents", "description": "Ultra-light solo tent, 3-season,
850g, fast-and-light hiking", "price_usd": 299.99, "weight_grams": 850, "in_stock": True,
"season": "3-season", }, { "id": "P004", "name": "ArcticDown -20°C Sleeping Bag",
"category": "Sleeping Bags", "description": "Winter sleeping bag, 800-fill down, rated to
-20°C, mummy shape", "price_usd": 389.99, "weight_grams": 1400, "in_stock": True, "season":
"winter", }, { "id": "P005", "name": "SummerLight +5°C Sleeping Bag", "category": "Sleeping
Bags", "description": "Lightweight summer sleeping bag, synthetic insulation, +5°C",
"price_usd": 149.99, "weight_grams": 700, "in_stock": True, "season": "summer", }, { "id":
"P006", "name": "Osprey Atmos 65L Backpack", "category": "Backpacks", "description":
"65-liter backpacking pack, anti-gravity suspension, hipbelt pockets", "price_usd": 289.99,
"weight_grams": 1980, "in_stock": True, "season": "all", }, { "id": "P007", "name":
"DayHiker 22L Daypack", "category": "Backpacks", "description": "22-liter lightweight
daypack, laptop sleeve, hydration compatible", "price_usd": 89.99, "weight_grams": 580,
"in_stock": True, "season": "all", }, ]
```

===== TOOL 1: `list_categories`

=====

TODO 2: Define a tool called `list_categories` using the `@mcp.tool()` decorator. Requirements:

- Takes no parameters - Returns a sorted list of unique category names (strings) from

`PRODUCTS` - Include a docstring: "List all available product categories." Hint: use `set()` +

`sorted()` to get unique sorted categories `@mcp.tool()` `def list_categories()` -> `list[str]: ...`

===== TOOL 2: `search_products`

=====

TODO 3: Define a tool called search_products using the @mcp.tool() decorator. Parameters: - keyword: str ? search term matched against product name and description - category: str | None = None ? optional filter (use list_categories for valid values) - max_price: float | None = None ? optional maximum price in USD - in_stock_only: bool = True ? if True, only return in-stock items - max_results: int = 5 ? maximum number of results Returns: list[dict] ? matching products (each dict has id, name, category, price_usd, in_stock) Docstring should explain each parameter (the LLM reads this to decide when/how to use the tool!) @mcp.tool() def search_products(keyword: str, ...) -> list[dict]: ...

===== TOOL 3: get_product_details
=====

TODO 4: Define a tool called get_product_details using the @mcp.tool() decorator. Parameters: - product_id: str ? the product ID (e.g. "P001") Returns: - The full product dict if found (all fields including description, weight_grams, season) - {"error": "Product not found", "product_id": product_id} if not found @mcp.tool() def get_product_details(product_id: str) -> dict: ...

===== TOOL 4: compare_products

(CHALLENGE ? more advanced) =====

TODO 5 (Challenge): Define a tool called compare_products. Parameters: - product_ids: list[str] ? list of product IDs to compare (2-4 items) Returns: dict with structure: { "products": [...], # list of found products "not_found": [...], # product_ids that weren't found "lightest": "P003", # id of lightest product (by weight_grams) "cheapest": "P007", # id of cheapest product (by price_usd) } @mcp.tool() def compare_products(product_ids: list[str]) -> dict: ...

===== Entry point
=====

if __name__ == "__main__": # TODO 6: Start the server by calling mcp.run() # This runs the server in stdio mode (default for local development) pass # Replace with: mcp.run()