

Lab 020 : Construire un serveur MCP en Python

Niveau : L200 Parcours : ? MCP Durée : ~45 min ? Coût : Gratuit ? S'exécute localement, aucun compte cloud nécessaire

Ce que vous apprendrez

Comment construire un serveur MCP à partir de zéro en utilisant FastMCP (Python) Comment définir des outils (Tools) avec des schémas et descriptions appropriés Comment exécuter le serveur et le connecter au MCP Inspector et à GitHub Copilot (VS Code) Comment ajouter le transport HTTP/SSE pour la compatibilité avec les agents cloud

Introduction

Un serveur MCP est un programme qui expose des outils (fonctions) aux agents IA via le protocole MCP. Quand un agent a besoin d'effectuer une action ? interroger une base de données, appeler une API, lire un fichier ? il appelle les outils de votre serveur MCP.

Dans ce lab, nous construisons un serveur MCP de recherche de produits avec deux outils :

1. `list_categories` ? retourne les catégories de produits 2. `search_products` ? recherche des produits par mot-clé

Prérequis

[diagram or code sample]

Assurez-vous d'avoir Python 3.10+ : [diagram or code sample]

tip "Démarrage rapide avec GitHub Codespaces" Open in GitHub Codespaces

Toutes les dépendances sont pré-installées dans le devcontainer.

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-020/ dans votre répertoire de travail.

Fichier | Description | Téléchargement | `outdoorgear_mcp_server_starter.py` | Script de démarrage avec des TODOs | ? Télécharger |

Exercice du lab

Étape 1 : Créer le projet

[diagram or code sample]

Créez `server.py` :

[diagram or code sample]

Étape 2 : Définir vos outils

Ajoutez les outils après la liste `PRODUCTS` :

[diagram or code sample]

Étape 3 : Exécuter le serveur (mode stdio)

Ajoutez le point d'entrée à la fin de `server.py` :

[diagram or code sample]

Exécutez-le : [diagram or code sample]

Le serveur écoute maintenant sur stdio. C'est le mode par défaut pour les outils locaux.

Étape 4 : Tester avec le MCP Inspector

Ouvrez un nouveau terminal et exécutez : [diagram or code sample]

L'Inspector s'ouvrira dans votre navigateur. Essayez :

1. Cliquez sur "Tools" pour voir vos trois outils 2. Cliquez sur `list_categories` ? "Run tool" ? voyez les catégories 3. Cliquez sur `search_products` ? remplissez keyword: "tent" ? "Run tool"

success "Vous devriez voir la Camping Tent dans les résultats"

Étape 5 : Exécuter en tant que serveur HTTP/SSE (pour les agents distants)

Pour les agents hébergés dans le cloud comme Microsoft Foundry, nous avons besoin du transport HTTP/SSE. Ajoutez l'option de démarrage :

[diagram or code sample]

Exécutez en mode HTTP : [diagram or code sample]

Vous verrez : [diagram or code sample]

Testez avec curl : [diagram or code sample]

Étape 6 : Connecter à GitHub Copilot dans VS Code

1. Dans VS Code, créez .vscode/mcp.json dans votre espace de travail :

[diagram or code sample]

2. Ouvrez GitHub Copilot Chat dans VS Code 3. Tapez : @copilot What product categories are available?

GitHub Copilot appellera votre outil list_categories et inclura le résultat dans sa réponse

!

tip "Support MCP dans VS Code" Assurez-vous d'avoir l'extension GitHub Copilot version

1.99+ installée. Vous devrez peut-être activer MCP dans les paramètres de VS Code :

"chat.mcp.enabled": true

Ajouter une ressource (Bonus)

MCP prend également en charge les Ressources ? des données que l'agent peut lire. Ajoutez une ressource qui expose le catalogue complet des produits :

[diagram or code sample]

? Fichier de démarrage

Ce lab inclut un fichier de démarrage avec des marqueurs TODO pour vous guider dans la construction du serveur :

[diagram or code sample]

[diagram or code sample]

Le fichier de démarrage contient le catalogue de produits OutdoorGear (P001?P007) déjà rempli. Vous implémentez : list_categories, search_products, get_product_details, et un outil de défi compare_products.

? Défi : Ajouter un outil compare_products

Une fois les 3 outils de base fonctionnels, ajoutez un quatrième :

[diagram or code sample]

Testez-le dans le MCP Inspector en demandant : *"Comparez le TrailBlazer Tent 2P et le TrailBlazer Solo. Lequel est le plus léger ?"*

L'agent devrait appeler compare_products(["P001", "P003"]) et retourner une comparaison structurée.

Résumé

Vous avez construit un serveur MCP entièrement fonctionnel qui :

? Définit 3 outils avec des descriptions appropriées (le LLM les utilise pour décider quand les appeler) ? S'exécute en mode stdio pour les outils locaux ? S'exécute en mode HTTP/SSE pour les agents distants ? Fonctionne avec le MCP Inspector pour les tests ? S'intègre avec GitHub Copilot dans VS Code

Prochaines étapes

Version C# : ? Lab 021 ? Serveur MCP en C# Connecter à Microsoft Foundry Agent Service : ?

Lab 030 ? Foundry Agent Service + MCP Ajouter des requêtes sur une vraie base de données :

? Lab 031 ? Recherche sémantique pgvector

Modernisation de Phase 3

warning "Verified 2026-10-27" Ciblez MCP 2026-07-28. Préférez serveur stateless, discovery, SDK Tier 1 actuel et état multi round-trip explicite. Ajoutez Registry et autorisation selon le modèle de confiance.

[diagram or code sample]

Model Context Protocol specification MCP authorization MCP Registry

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré

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()

```