XMAS DEV 2025

🎅📨 “Dalla letterina a Babbo Natale all’ordine perfetto: magia (MCP) sotto l’albero”

Gaetano Paternò

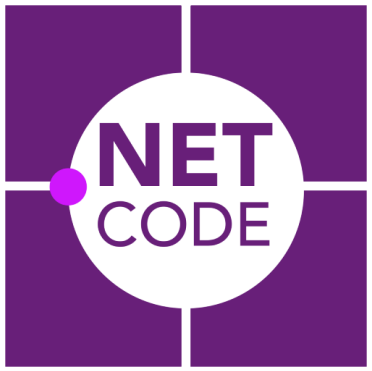
Microsoft Aenduo BC Soft consulenza sistemi informativi JETBRAINS Packt›

Platinum Sponsor

Gold Sponsor

Technical Sponsor

Community



DotNetAbruzzo

ANTHROP\C Model Context Protocol

Model Context Protocol, o MCP, è uno standard aperto che sta diventando rapidamente il “linguaggio universale” con cui gli LLM parlano con strumenti esterni.

MCP permette a un modello di:

- scoprire le funzionalità disponibili,
- capire quali parametri servono,
- invocare funzioni tipizzate,
- ricevere risposte standardizzate.

USB-C PER LE AI APPLICATION

⌛ Prima ogni sistema inventava il proprio metodo per far usare strumenti all'LLM.

👉 Ora abbiamo uno standard unico per:

- definire tool
- descriverli
- scoprire cosa è disponibile
- chiamarli in modo coerente

Questa standardizzazione è potentissima perché:

- elimina protocolli custom
- rende strumenti sviluppati da team diversi compatibili tra loro
- permette all'LLM di ragionare sapendo esattamente cosa può fare
- accelera l'integrazione tra AI e backend

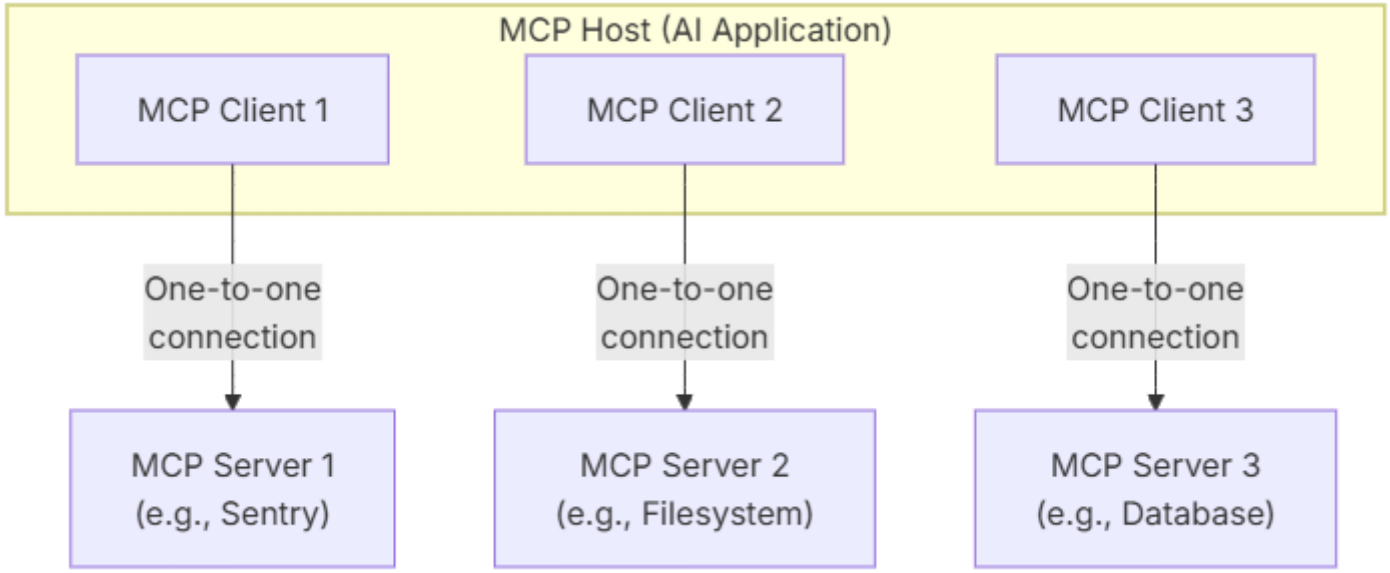
…il tutto senza costruire infrastrutture proprietarie.

I PRINCIPALI PARTECIPANTI ALL'ARCHITETTURA MCP

MCP Host: l'applicazione AI che coordina e gestisce uno o più client MCP.

MCP Client: un componente che mantiene una connessione a un server MCP e ottiene il contesto da un server MCP affinché l'host MCP possa utilizzarlo.

MCP Server: un programma che fornisce contesto ai client MCP.



MCP È COSTITUITO DA DUE STRATI

Data layer

Il data layer implementa un protocollo di scambio basato su JSON-RPC 2.0 che definisce la struttura e la semantica del messaggio.

- **Lifycycle management**: gestisce l'inizializzazione della connessione, la negoziazione delle capacità e la terminazione della connessione tra client e server.
- **Server features**: consente ai server di fornire funzionalità di base tra cui strumenti per azioni di intelligenza artificiale, risorse per dati di contesto e richieste per modelli di interazione da e verso il client.
- **Client features**: consente ai server di chiedere al client di campionare dall'LLM host, ottenere input dall'utente e registrare i messaggi sul client.
- **Utility features**: supporta funzionalità aggiuntive come notifiche per aggiornamenti in tempo reale e monitoraggio dei progressi per operazioni di lunga durata.

Transport layer

Il transport layer gestisce i canali di comunicazione e l'autenticazione tra client e server.

MCP supporta due meccanismi di trasporto:

- **Stdio transport**: utilizza flussi di input/output standard per la comunicazione diretta dei processi tra processi locali sulla stessa macchina, garantendo prestazioni ottimali senza sovraccarico di rete.
- **Streamable HTTP trasport**: utilizza HTTP POST per i messaggi client-server con eventi inviati dal server opzionali per le funzionalità di streaming. Questo trasporto consente la comunicazione con server remoti e supporta metodi di autenticazione HTTP standard, tra cui token portanti, chiavi API e intestazioni personalizzate. MCP consiglia di utilizzare OAuth per ottenere token di autenticazione.

“I SERVER MCP SONO PROGRAMMI CHE ESPONGONO FUNZIONALITÀ SPECIFICHE ALLE APPLICAZIONI AI TRAMITE INTERFACCE DI PROTOCOLLO STANDARDIZZATE.”

- **Server Console**

Ideale quando:

- l’applicazione gira lato utente
- i tool interagiscono con il file system locale
- si vuole distribuire un agente locale (desktop agent, automazioni, CLI intelligenti)

Il transport usato è solitamente **stdio** e l’applicazione comunica con l’LLM via pipe.

- **Server Web (.NET Minimal API o ASP.NET Core)**

Si usa quando:

- vogliamo esporre tool a più client contemporaneamente
- desideriamo integrare MCP in un sistema enterprise
- abbiamo un microservizio AI-dedicated
- vogliamo scalare in cloud

Il transport è **HTTP/HTTPS**.

Il vantaggio enorme è che il codice dei tool non cambia.
Cambia solo il modo in cui il server comunica con i client MCP.

DEMO: Costruzione del Server MCP in .NET

builder.Services.AddMcpServer()

Registra tutto il codice necessario per:

- discovery
- tool loading
- validazione dei parametri
- gestione del protocollo

WithHttpTransport()
Aggiunge il transport HTTP/HTTPS. Si usa nei server web.

WithConsoleTransport()
Per applicazioni console. Il transport avviene via stdio.

WithToolsFromAssembly()
Esegue il discovery nell'assembly specificato.

app.AddMcp("/mcp")
Il path è opzionale, ma esplicitarlo:

- evita cambiamenti futuri
- rende il servizio più prevedibile
- facilita i client

Claude

```
{
  "mcpServers": {
    "demo": {
      "command": "dotnet",
      "args": ["run --project",
        "C:\\Users\\tanop\\source\\repos\\MCPTest\\Server\\Server.Console.csproj",
        "--no-build"]
    }
  }
}
```

```
{
  "mcpServers": {
    "demo": {
      "command": "npx",
      "args": ["mcp-remote",
        "http://localhost:5000"]
    }
  }
}
```

ANALISI DETTAGLIATA DEI TOOL MCP

🔷 [McpServerToolType] – Sistema di Discovery

Quando una classe è annotata con questo attributo, stiamo dicendo al runtime:

“In questa classe ci sono uno o più tool MCP. Scoprili automaticamente e rendili disponibili al client.”

È l’entry point del discovery.

Perché è utile?

- Riduce la configurazione manuale
- Permette una struttura modulare
- Facilita la manutenzione (aggiungi un metodo → diventa un tool)

🔷 [McpServerTool] – Dichiarazione del Tool

Ogni tool esposto al modello è un metodo marcato con questo attributo.

Perché i tool vanno raggruppati e non moltiplicati?

Gli LLM ragionano in termini probabilistici, più tool esponiamo, più aumentiamo:

- la confusione
- la sovrapposizione semantica
- il rischio che l’LLM scelga il tool sbagliato

Un buon server MCP ha:

- tool generali e utili
- ben descritti
- non ridondanti

ANALISI DETTAGLIATA DEI TOOL MCP

🔷 Discovery: Automatico vs Manuale

Hai due approcci:

✔️ Automatico

Con '*WithToolsFromAssembly*' vengono caricati tutti i tool trovati.

Pro: rapida evoluzione, ottimo per dev e prototipi
Contro: rischio di esporre tool non desiderati

✔️ Manuale

Con '*WithTools<T>*' registri tu solo ciò che serve.

Pro: maggiore controllo
Contro: più manutenzione

🔷 L'importanza delle Description

La Description è uno dei punti più importanti dell'intero talk.

Gli LLM fanno *tool selection* basandosi su ciò che leggono nella descrizione.

Una buona Description risponde a:

1. Cosa fa il tool
2. Quando va usato
3. Con quali parametri
4. Quali sono i limiti
5. Che tipo di risultato produce

Questo aiuta enormemente il modello ad agire correttamente.

NO Swagger ? -: MCP Inspector

npx @modelcontextprotocol/inspector

🔍 **Test e Debug: niente Swagger**

Gli endpoint REST hanno Swagger.

MCP **non** ha endpoint per ogni tool:
ha **un solo endpoint**, e al suo interno passiamo:

- ID del tool
- parametri tipizzati
- schema MCP

Per ispezionare il server serve **MCP Inspector**.

Con MCP Inspector possiamo:

- vedere i tool caricati
- visualizzare la documentazione generata
- eseguire test in tempo reale
- vedere i payload di protocollo
- simulare il comportamento del client LLM

È la controparte “Swagger” del mondo MCP.

🔐 **CORS per Client Web**

Se il server MCP è esposto via HTTP e acceduto da un client in browser:
CORS deve essere abilitato, altrimenti il browser blocca le richieste.

DEMO: Costruzione del Client MCP in .NET

Il client è sorprendentemente semplice:

1. Creazione del transport
var transport = new HttpClientTransport("<server-url>");

2. Creazione del client MCP
var client = await McpClient.CreateAsync(transport);

Il client esegue automaticamente:
- handshake
- caricamento schema tools
- validazione

3. Listing tool
var tools = await client.ListToolsAsync();

4. Invocazione tool
(Tipicamente generando un dictionary con i parametri o usando DTO tipizzati)

Claude

OpenAI

Ollama

XMASDEV
<OH>OH</OH>

SDK DISPONIBILI

- TypeScript
- Python
- Go
- Kotlin
- Swift
- Java
- C#
- Ruby
- Rust
- PHP

https://modelcontextprotocol.io/docs/getting-started/intro

CONCLUSIONE E VISIONE

MCP è uno dei mattoni fondamentali per costruire applicazioni AI veramente integrate.

Grazie alla sua natura:

- aperta
- standardizzata
- tipizzata
- interoperabile

…possiamo oggi creare con .NET applicazioni intelligenti molto più solide.

PRESENTE E FUTURO?

Abbiamo e avremo:

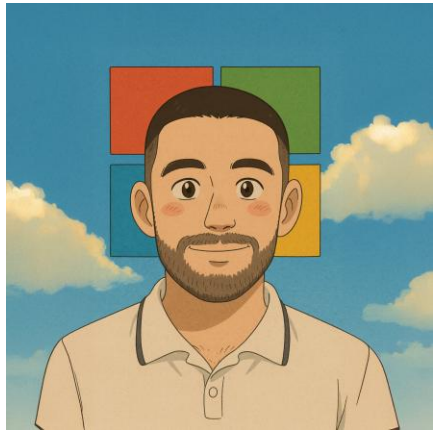
- marketplace sempre più ricchi di tool MCP interoperabili
- applicazioni che composte da strumenti di team diversi
- agent autonomi in grado di negoziare funzionalità
- ecosistemi AI modulari

XMASDEV
<OH>OH</OH>

PLEASE VOTE FOR THIS SESSION

Vote online at:

https://vote.dotnetdev.it/vote/xqt335vh/1040445



facebook.com
linkedin.com
github.com
} /tanopaterno

www.gaetanopaterno.it
info@gaetanopaterno.it

EtnaDev
SOFTWARE DEVELOPERS COMMUNITY

Microsoft CERTIFIED ASSOCIATE

Microsoft CERTIFIED EXPERT