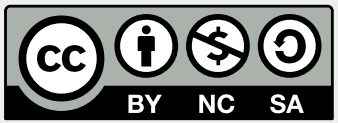
Model Context Protocol (MCP)

Arquitectura, capas y protocolo de contexto para IA

Autor: Alexander Saravia

Protocolos Agénticos

21 de noviembre de 2025



Agenda

1. Introducción
   - Qué es, para qué sirve, qué no es
2. Conceptos y participantes
   - Conceptos de MCP
   - Participantes
3. Arquitectura por capas
   - Capas
   - Capa de datos
   - Capa de transporte
4. Protocolo de la capa de datos
   - Data Layer Protocol
   - Lifecycle management
   - Primitivas
   - Notificaciones
5. Ejemplo básico cliente–servidor
   - Escenario
6. Conclusiones y referencias

Introducción

Motivación

- Los modelos de lenguaje necesitan conectarse con:
  - Fuentes de datos (archivos, bases de datos, APIs).
  - Herramientas de acción (automatización, sistemas externos).
  - Flujos de trabajo y *prompts* reutilizables.
- Sin un protocolo estándar, cada integración es ad–hoc:
  - Alto acoplamiento entre modelo, herramienta y proveedor.
  - Difícil de mantener, versionar y auditar.
- **MCP** propone un “USB-C para la IA”:
  - Una forma uniforme de conectar aplicaciones de IA con servidores de contexto.

Introducción

Qué es, para qué sirve, qué no es

Definición general

El **Model Context Protocol (MCP)** es un estándar abierto de capa de aplicación para conectar aplicaciones de IA (por ejemplo, asistentes basados en LLM) con sistemas externos mediante un protocolo cliente–servidor basado en JSON-RPC 2.0.

- Define:
  - Un **conjunto de participantes**: host, clientes MCP y servidores MCP.
  - Una **capa de datos** (protocolo) con primitivas y ciclo de vida.
  - Una **capa de transporte** (STDIO, HTTP streamable, etc.).
- Se centra en el **intercambio de contexto**: recursos, herramientas, prompts y notificaciones entre cliente y servidor.

Para qué sirve MCP

- **Estandarizar** cómo una aplicación de IA:
  - Descubre herramientas remotas.
  - Llama a esas herramientas.
  - Lee recursos y usa *prompts* compartidos.
- **Desacoplar** el modelo de IA de:
  - El proveedor de datos.
  - El lenguaje de implementación del servidor.
  - El tipo de transporte (local o remoto).
- Permitir:
  - Conectar un mismo asistente a múltiples servidores MCP.
  - Reutilizar servidores MCP entre distintos asistentes y modelos.
  - Construir arquitecturas de agentes multi–tool sobre una base común.

Qué *no* es MCP

- **No es** un modelo de IA ni una API de *inference*.
- **No es** un orquestador de agentes completo:
  - No define estrategias de planificación, *routing* o memoria de alto nivel.
- **No decide** cómo usar el contexto:
  - No prescribe cómo se construyen los prompts finales del LLM.
  - No define políticas de negocio, *ranking* ni autorización de alto nivel.
- **Alcance explícito**:
  - Se limita al **protocolo de intercambio de contexto**.
  - Delega decisiones de UX, orquestación y uso del LLM a la aplicación host.

Conceptos y participantes

Conceptos y participantes

Conceptos de MCP

Conceptos de MCP

La documentación de MCP organiza los conceptos principales en:

1. **Participantes** (*participants*).
2. **Capas** (*layers*):
   - Capa de datos.
   - Capa de transporte.
3. **Protocolo de la capa de datos**:
   - Gestión del ciclo de vida.
   - Primitivas.
   - Notificaciones.
4. **Ejemplo de intercambio** cliente–servidor.

Conceptos y participantes

Participantes

Arquitectura cliente–servidor

MCP sigue un modelo cliente–servidor con roles bien definidos:

- **MCP Host**:
  - Aplicación de IA (p.ej. editor, chat, IDE) que coordina uno o varios clientes MCP.
- **MCP Client**:
  - Componente que mantiene la conexión 1:1 con un servidor MCP concreto.
  - Gestiona el protocolo de capa de datos (JSON-RPC) con ese servidor.
- **MCP Server**:
  - Programa que expone contexto: herramientas, recursos, prompts, etc.
  - Puede ejecutarse de forma local (STDIO) o remota (HTTP).

Diagrama de participantes (esbozo)

- Host crea uno o varios clientes MCP.
- Cada cliente se conecta a un servidor MCP.
- El LLM “ve” un conjunto unificado de herramientas y recursos.

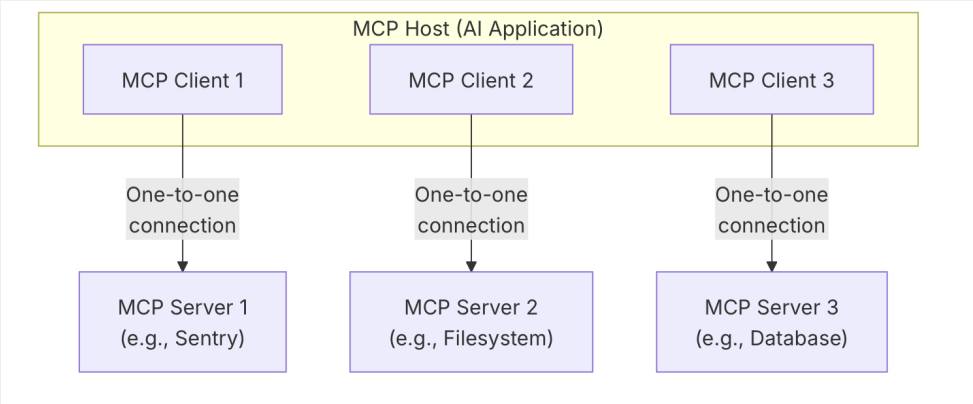


Figura: Relación Host–Clientes–Servidores MCP (esquema).

Arquitectura por capas

Arquitectura por capas

Capas

Dos capas principales

MCP se organiza en dos capas conceptuales:

- **Capa de datos** (*data layer*):
  - Define el protocolo basado en JSON-RPC 2.0.
  - Incluye ciclo de vida, primitivas y notificaciones.
- **Capa de transporte** (*transport layer*):
  - Define cómo viajan los mensajes: STDIO, HTTP, etc.
  - Gestiona autenticación, framing y canales.

Comparativa: Capa de datos vs Capa de transporte

Cuadro: Comparativa entre la capa de datos y la capa de transporte en MCP

	Capa de datos	**Capa de transporte**
Rol	Define el protocolo JSON-RPC, los tipos de mensajes, ciclo de vida y primitivas (tools, resources, prompts, sampling, etc.).	Define cómo viajan los mensajes: STDIO local o HTTP streamable, incluyendo framing, autenticación y canales.
Abstracción	Independiente del proveedor de nube o del runtime donde vive el servidor.	Puede cambiar entre procesos locales, contenedores o servicios remotos sin modificar el protocolo de datos.
Ejemplos	`initialize`, `tools/list`, `tools/call`, notificaciones de progreso.	STDIO sobre `stdin/stdout`, HTTP POST + SSE, futuros transportes (por ejemplo WebSocket) compatibles.

Nota: Adaptado de la documentación oficial de MCP [1].

[1] `https://modelcontextprotocol.io/docs/learn/architecture`

Arquitectura por capas

Capa de datos

Capa de datos: responsabilidades

- Implementa un protocolo basado en **JSON-RPC 2.0**:
  - Mensajes de petición–respuesta.
  - Notificaciones unidireccionales sin respuesta.
- **Gestión del ciclo de vida**:
  - Inicialización y negociación de capacidades.
  - Terminación de la conexión.
- **Primitivas del servidor**:
  - *Tools*, *Resources*, *Prompts*.
- **Primitivas del cliente**:
  - *Sampling*, *Elicitation*, *Logging* y tareas (*tasks*, experimental).
- **Utilidades transversales**:
  - Notificaciones, seguimiento de progreso, ejecución durable.

Arquitectura por capas

Capa de transporte

Capa de transporte: responsabilidades

- Gestiona los **canales de comunicación** entre cliente y servidor:
  - Establecimiento de conexión.
  - Framing de mensajes.
  - Autenticación y seguridad.
- Provee diferentes mecanismos:

  STDIO: comunicación local por `stdin/stdout|`.

  HTTP `streamable`: `HTTP POST + Server-Sent Events (SSE) para streaming.`
- Abstrae el transporte de la capa de datos:
  - La carga útil JSON-RPC es la misma, cambian sólo los “tubos”.

Protocolo de la capa de datos

Protocolo de la capa de datos

Data Layer Protocol

Idea central

La capa de datos define el **esquema y la semántica** de los mensajes JSON-RPC entre clientes y servidores MCP.

- Mensajes de **petición**:
  - Tienen `id`, `method` y `params`.
  - Esperan una respuesta con el mismo `id`.
- Mensajes de **respuesta**:
  - Devuelven `result` o `error`.
- **Notificaciones**:
  - No llevan `id`, no se espera respuesta.
  - Usadas para cambios de estado (*listChanged*, progreso, etc.).

Protocolo de la capa de datos

Lifecycle management

Lifecycle management

- MCP es un protocolo **stateful**.
- El ciclo de vida típico incluye:
  1. **Inicialización** (`initialize`): negociación de versión y capacidades.
  2. **Uso normal**: intercambio de peticiones de primitivas (tools, resources, prompts…).
  3. **Terminación**: cierre ordenado de la sesión.
- Objetivos principales:
  - Asegurar compatibilidad de versión del protocolo.
  - Descubrir qué primitivas y características soporta cada lado.
  - Intercambiar información de identidad para depuración y auditoría.

Ejemplo: mensaje `initialize`

Listing 1: Ejemplo simplificado de petición initialize (JSON-RPC)

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "initialize",
  "params": {
    "protocolVersion": "2025-06-18",
    "capabilities": {
      "elicitation": {}
    },
    "clientInfo": {
      "name": "example-client",
      "version": "1.0.0"
    }
  }
}
```

- `protocolVersion`: versión de MCP que habla el cliente.
- `capabilities`: primitivas y features que el cliente soporta.
- `clientInfo`: metadatos de identificación.

Notificación de cliente listo

Después de la negociación, el cliente puede enviar una notificación:

Listing 2: Notificación de que la inicialización terminó

```
{
  "jsonrpc": "2.0",
  "method": "notifications/initialized"
}
```

- No hay `id`: es una notificación, no se espera respuesta.
- Señala al servidor que el cliente está preparado para usar las primitivas.

Protocolo de la capa de datos

Primitivas

Primitivas principales

Las primitivas definen lo que el servidor puede ofrecer al cliente:

Tools: funciones ejecutables que el LLM puede invocar.
- Acciones: llamadas a APIs, consultas a BD, operaciones de archivos, etc.

Resources: fuentes de datos de sólo lectura.
- Ej.: archivos, vistas de BD, contenido de documentación.

Prompts: plantillas reutilizables de interacción.
- Pueden incluir contexto, ejemplos, argumentos dinámicos.

Cada tipo de primitiva dispone de métodos para:
- `*/list` (descubrir qué hay disponible).
- `*/get` (recuperar detalles de un elemento).
- `tools/call` (ejecutar una herramienta).

Primitivas del cliente MCP

El cliente también expone capacidades que los servidores pueden usar:

Sampling:
- `sampling/complete` permite al servidor pedir una completación de LLM al host sin acoplarse a un proveedor concreto.

Elicitation:
- `elicitation/request` para solicitar más información o confirmación al usuario final.

Logging:
- Enviar logs estructurados al host para monitoreo y depuración.

Tasks (experimental):
- Envoltorios durables para operaciones de larga duración con seguimiento de estado.

Protocolo de la capa de datos

Notificaciones

Notificaciones en MCP

- Las notificaciones son mensajes JSON-RPC sin `id`.
- Sirven para **actualizaciones en tiempo real**:
  - Cambios en el conjunto de herramientas (`tools/list_changed`).
  - Cambios en recursos (`resources/updated`).
  - Progreso en operaciones largas.
- Características:
  - No bloquean: no se espera respuesta.
  - Permiten que el host mantenga un *snapshot* coherente del estado de cada servidor MCP.
  - Reducen la necesidad de *polling*.

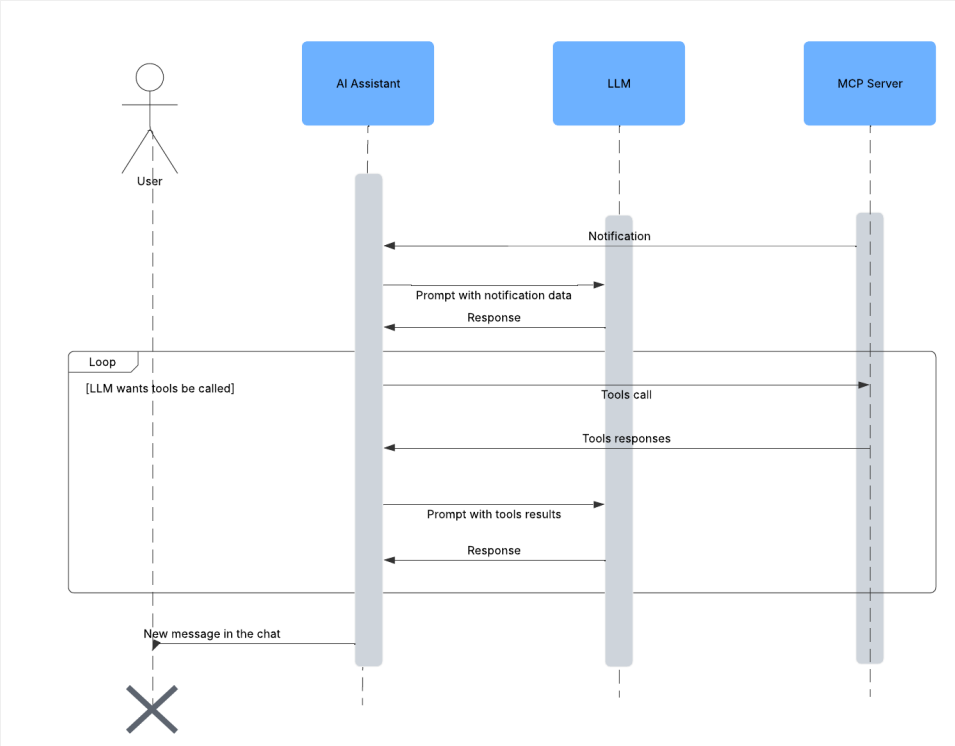


Figura: Esquema de flujo de notificaciones.

Ejemplo básico cliente–servidor

Ejemplo básico cliente–servidor

Escenario

Objetivo

Mostrar un ejemplo mínimo de interacción MCP entre un cliente y un servidor.

- Servidor MCP: expone una herramienta `get_weather` y un recurso con el esquema de datos.
- Cliente MCP: se conecta por STDIO, negocia capacidades y lista herramientas.
- Host: un asistente de IA que decide cuándo llamar a `get_weather`.

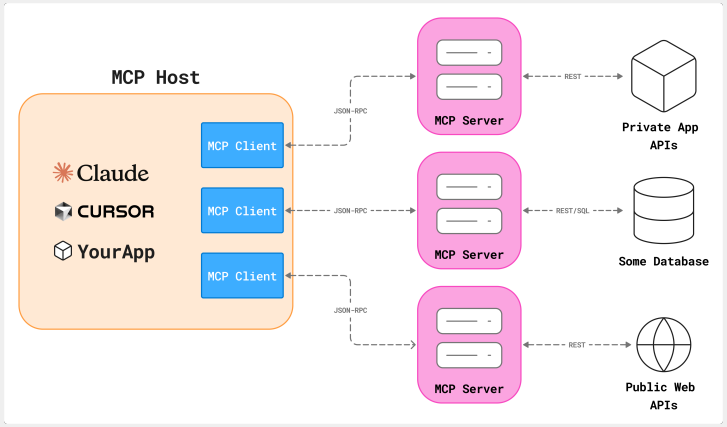


Figura: Visión general del ejemplo cliente–servidor.

Paso 1: el cliente envía `initialize`.

Listing 3: Petición initialize simplificada

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "method": "initialize",
  "params": {
    "protocolVersion": "2025-06-18",
    "capabilities": {
      "tools": {},
      "resources": {},
      "prompts": {},
      "elicitation": {}
    },
    "clientInfo": {
      "name": "weather-client",
      "version": "0.1.0"
    }
  }
}
```

El servidor responde con las capacidades que soporta y su `serverInfo`.

Paso 2: el cliente descubre qué herramientas expone el servidor.

Listing 4: Solicitud tools/list

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "method": "tools/list"
}
```

Respuesta esquemática:

Listing 5: Solicitud tools/list

```
{
  "jsonrpc": "2.0",
  "id": 2,
  "result": {
    "tools": [
      {
        "name": "get_weather",
        "title": "Get current weather",
        "description": "Returns weather for a given city.",
        "inputSchema": {
          "type": "object",
          "properties": {
            "city": { "type": "string" }
          },
          "required": ["city"]
        }
      }
    ]
  }
}
```

3. Ejecución de herramienta

Paso 3: el host decide invocar la herramienta.

Listing 6: Solicitud tools/call

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "method": "tools/call",
  "params": {
    "name": "get_weather",
    "arguments": {
      "city": "Quito"
    }
  }
}
```

Respuesta típica:

Listing 7: Respuesta tools/call (resumida)

```
{
  "jsonrpc": "2.0",
  "id": 3,
  "result": {
    "content": [
      {
        "type": "text",
        "text": "Weather in Quito: 18C, cloudy."
      }
    ]
  }
}
```

Ejemplo ilustrativo usando un servidor MCP simplificado:

Listing 8: Esqueleto de servidor MCP (conceptual)

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

mcp = FastMCP("weather_server")

@mcp.tool()
def get_weather(city: str) -> str:
   # Lgica real: llamar API de clima, etc.
   return f"Weather in {city}: 18C, cloudy."

if __name__ == "__main__":
   # Ejecuta servidor por STDIO
   mcp.run_stdio()
```

Nota: el SDK real puede variar; este ejemplo es conceptual.

Resumen del flujo cliente–servidor

1. Host crea un cliente MCP y establece transporte (STDIO/HTTP).
2. Cliente y servidor ejecutan **initialize** y negocian capacidades.
3. Cliente usa las primitivas:
   - `tools/list`, `resources/list`, `prompts/list`.
   - `tools/call` para ejecutar acciones.
4. Servidor envía **notificaciones** cuando cambia su estado (por ejemplo, nuevas herramientas).
5. Host orquesta cómo el LLM usa ese espacio de herramientas y recursos.

Conclusiones y referencias

Conclusiones

- MCP proporciona una **capa estándar** para conectar IA con sistemas externos.
- La separación en **capa de datos** y **capa de transporte** permite flexibilidad de implementación.
- Las **primitivas** (tools, resources, prompts, sampling, elicitation, logging) permiten construir agentes y asistentes ricos sin acoplarse a un proveedor concreto.
- Las **notificaciones** y la gestión de ciclo de vida soportan escenarios dinámicos y multi–servidor.
- MCP no reemplaza al LLM ni al orquestador: es la “interfaz estándar” sobre la cual se construyen arquitecturas más complejas.

Referencias

- Model Context Protocol – Sitio oficial:
  https://modelcontextprotocol.io/
- *Architecture overview* – Documentación MCP:
  https://modelcontextprotocol.io/docs/learn/architecture
- Especificación JSON-RPC 2.0:
  https://www.jsonrpc.org/specification
- Ejemplos y SDKs MCP (repositorio oficial):
  https://github.com/modelcontextprotocol
- Recursos introductorios y guías de terceros (blogs / artículos técnicos):
  - https://betterstack.com/community/guides/ai/mcp-explained/
  - https://opencv.org/blog/model-context-protocol/