UNIVERSIDAD DEL VALLE DE GUATEMALA

Data Science

Protocolo de mcp

Joaquin Campos

Guatemala, septiembre 2025

Introducción

Este informe presenta la construcción de un chatbot capaz de operar con el Model Context Protocol (MCP) a través de cuatro servidores: uno local para gestión de música (playlists y canciones), dos servidores MCP oficiales (Filesystem y GitHub) y un servidor remoto desplegado en Cloudflare Workers. La solución permite operar desde una interfaz de consola (CLI) y una pequeña GUI, además de inspeccionar el comportamiento con la herramienta MCP Inspector.

El objetivo fue demostrar la integración real de servicios heterogéneos bajo un mismo host, analizar la comunicación en red (capturando el tráfico con Wireshark) y documentar el proceso de despliegue y las decisiones técnicas necesarias para conseguir un flujo estable de JSON-RPC entre cliente y servidor.

Arquitectura general

El host (aplicación principal) descubre y consume herramientas MCP de forma dinámica. Según variables de entorno, puede conectarse por stdio a un servidor local o, alternativamente, mediante SSE (Server-Sent Events) a un servidor remoto publicado en Cloudflare Workers. El transporte remoto expone dos rutas: /sse (stream) y /sse/message (envío de solicitudes). Se añadieron logs JSONL en el host para correlacionar cada solicitud y respuesta con su identificador (id), lo que facilitó el análisis junto con Wireshark y el MCP Inspector.

Comunicación

Capa de protocolo: usamos JSON-RPC 2.0.

- Solicitud: objeto con jsonrpc: "2.0", method, params e id (entero).
- Respuesta: mismo id y uno de: result (éxito) o error (fallo con code + message).
- Notificaciones: no tienen id y no esperan respuesta.

Llamadas clave en MCP:

- tools/list → devuelve el catálogo de herramientas.
- tools/call → ejecuta la tool con sus argumentos.

Demultiplexado por id: el host mantiene un mapa id → cola. Al enviar una solicitud, registra la cola antes de emitirla; cuando llega la respuesta con ese id, la entrega a la cola correcta. Así soporta concurrencia y reintentos.

Local (MCP de música) por STDIO
El host lanza el servidor como proceso hijo y le escribe una línea JSON por request (newline-delimited JSON) en stdin. Hilos separados leen stdout/stderr. Si el proceso muere, se puede relanzar y reintentar (misma id).

Remoto (Worker en Cloudflare) por SSE + HTTP

- El cliente abre GET /sse (stream Server-Sent Events) y recibe un evento de sincronización connect con connectionId (esto no es JSON-RPC; sirve para establecer sesión).
- Cada request JSON-RPC se envía con POST /sse/message como:
- { "connectionId": "...", "message": { "jsonrpc": "2.0", "id": 6, "method": "...", "params": {...}}}
- Las respuestas viajan de vuelta por el stream SSE y el cliente las enruta por id.
- Si el SSE cae (timeout), el cliente reconecta a /sse y continúa; como todo va con id, no hay confusión.

Implementaciones por MCP

MCP de listas de reproducción (local)

MCP de Listas de Reproducción (local): provee herramientas add_song, list_playlists, get_playlist, export_playlist y clear_library. Permite añadir canciones, proponer candidatos y asignar automáticamente a playlists según metadatos. Se implementó el flujo de confirmación (“needs_confirmation”) y se normalizó el esquema de herramientas para el LLM (de inputSchema a input_schema), además de inyectar el parámetro workspace cuando aplica.

MCP Filesystem (oficial)

MCP Filesystem (oficial): habilita operaciones habituales sobre archivos y carpetas, útil para revisar exportaciones, escribir notas de trabajo y validar resultados del host.

MCP GitHub (oficial)

MCP GitHub (oficial): ofrece operaciones con repositorios (consultar, crear ramas o PR, leer issues y más). Se usó para versionar el proyecto e integrar cambios; se documentaron temas de permisos del token al intentar acciones de escritura.

MCP remoto en Cloudflare Workers

MCP Remoto (Cloudflare Workers): servidor sin autenticación por SSE con herramientas time (hora actual), sha256 (hash de texto), echo (repetir texto) y utilidades de cálculo básico. La sesión se maneja mediante un Durable Object que emite un evento 'connect' y gestiona los mensajes JSON-RPC por el canal SSE. Endpoint de ejemplo: https://remote-mcp-jjcam.jjcampos2003.workers.dev/sse

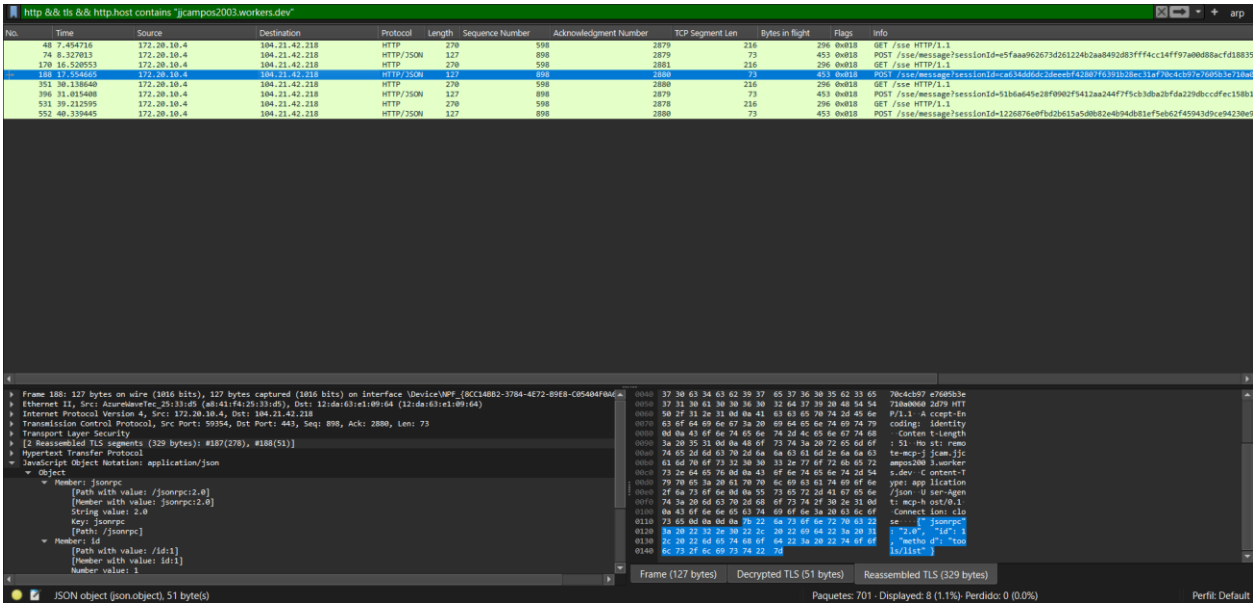
Flujo de comunicación y capturas (Wireshark)

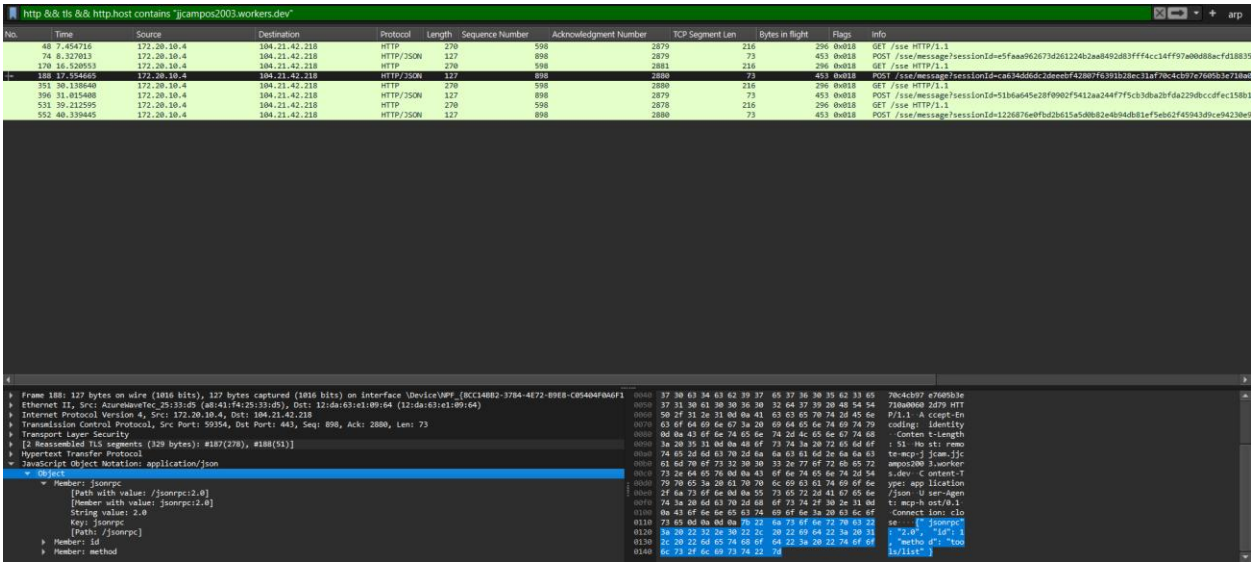
Para observar el intercambio, se activó el registro de claves TLS (SSLKEYLOGFILE) y se configuró Wireshark con ese fichero en “TLS – (Pre)-Master-Secret log filename”. Como el dominio de Cloudflare usa HTTP/3 sobre QUIC, el filtro práctico fue ‘quic || http3’. En la práctica se ven paquetes Handshake/Initial, seguidos de frames de datos cifrados (Protected Payload).

La correlación se hace con los logs del host: cada solicitud JSON-RPC enviada por POST a /sse/message queda como tools/list o tools/call con su id; la respuesta llega por el stream SSE con un objeto {"jsonrpc":"2.0","id":N,"result":...}. Además, el evento de sincronización ‘connect’ entrega el connectionId. Así, se clasificó: a) Sincronización: eventos ‘state/update’ y ‘connect’ del servidor; b) Solicitudes: POST /sse/message con métodos JSON-RPC; c) Respuestas: mensajes SSE con ‘result’ o ‘error’ que citan el mismo id.

Resultados

Entre algunos de los resultados los podemos enseñar a traevs de la lectura de los json rcps en el wireshark





Dificultades y cómo se resolvieron

- Ejecución de Wrangler en PowerShell: política de ejecución bloqueaba ‘wrangler.ps1’. Se usó ‘wrangler.cmd’ o se ajustó la política temporalmente.
- Autenticación y subdominio workers.dev: fue necesario iniciar sesión y crear el subdominio antes del primer deploy.
- Conexión SSE sin ‘connect’: el host necesitaba esperar el evento de sesión. Se ajustó el cliente para registrar el id antes de enviar solicitudes.
- Ruta de POST: se corrigió el envío a ‘/sse/message’ respetando la URL base del stream.

- Esquemas de tools: algunos servidores usan ‘inputSchema’. Se normalizó a ‘input_schema’ y se manejó el parámetro ‘workspace’.
- Comando ‘echo’ ambiguo: se añadió en el CLI un atajo determinista para ‘echo …’ y ‘repite: …’, reduciendo respuestas genéricas del LLM.
- Captura QUIC/HTTP3: al principio no había archivo de claves TLS; al configurarse, se pudieron decodificar los flujos.

Lecciones aprendidas

Separar el ‘host’ de los servidores MCP permite mezclar capacidades locales y remotas con muy poco acoplamiento. La observabilidad (logs y trazas) es clave para diagnosticar problemas de transporte. Cloudflare Workers resulta una plataforma ágil para publicar un MCP remoto de bajo costo y fácil despliegue.

Conclusiones

Se logró integrar cuatro MCP (local, filesystem, GitHub y remoto en Cloudflare) en un mismo chatbot, operable por CLI y verificado con MCP Inspector. El canal SSE y la normalización de esquemas garantizaron compatibilidad con JSON-RPC, y el uso de Wireshark permitió comprender la comunicación HTTP/3/QUIC de extremo a extremo. La base queda lista para añadir nuevas herramientas o evolucionar el frontend sin cambiar la arquitectura.

Apéndice: comandos y atajos útiles

- wrangler.cmd login / wrangler.cmd whoami / wrangler.cmd deploy
- npx @modelcontextprotocol/inspector
- python -m host.main tools | chat | repl
- Variables de entorno: MCP_SERVER_URL, MCP_TRANSPORT=sse, ANTHROPIC_API_KEY, etc.
- Filtros útiles en Wireshark: quic || http3; http2 and tcp.port == 443 and http2.headers.authority contains "workers.dev" http && tls && http.host contains "jjcampos2003.workers.dev"

VER DIFERENTES README PARA MAS INFORMACION