MCP en producción

Lo que su arquitectura de IA necesita y probablemente no tiene todavía

IA NFO Systems

2026

Contenido

MCP en producción **3**
Lo que su arquitectura de IA necesita y probablemente no tiene todavía . . . 3
Resumen ejecutivo . . . 3
El problema que su equipo ya conoce . . . 4
1. Por qué las integraciones artesanales no escalan . . . 4
2. Qué es MCP — la especificación . . . 4
3. Lo que MCP resuelve — y lo que no resuelve . . . 5
4. Implementación sobre infraestructura existente — tres patrones . . . 6
5. Seguridad y control de acceso . . . 8
6. Los desafíos reales de adopción técnica . . . 10
7. Por dónde empezar — el primer servidor MCP que vale la pena construir . 11
8. Siguientes pasos — para quien quiere ir más rápido de lo que su equipo puede solo . . . 12
Sobre IA NFO Systems . . . 12

MCP en producción

Lo que su arquitectura de IA necesita y probablemente no tiene todavía

IA NFO Systems · ia.nfo.systems

> Este documento está escrito para el director o jefe de tecnología que ya tiene agentes de IA en algún estado — piloto, producción parcial, o evaluación activa — y que enfrenta el mismo problema de fondo: las integraciones entre sus agentes y sus sistemas internos son artesanales, frágiles y no escalan. No es un documento introductorio sobre inteligencia artificial. Es un análisis técnico de criterio sobre MCP: qué resuelve, qué no resuelve, y cómo evaluar si vale implementarlo en su contexto específico.

Resumen ejecutivo

Si su equipo lleva más de tres meses intentando que un agente de IA use datos internos de forma confiable en producción, el problema ya tiene nombre: arquitectura de integración. Cada agente conectado a mano a cada sistema produce una matriz de dependencias que crece cuadráticamente, se rompe con cada actualización y no tiene un punto centralizado de diagnóstico cuando falla. MCP resuelve eso con un cambio arquitectónico: en lugar de N×M integraciones punto a punto, cada sistema publica sus capacidades una vez y cualquier agente compatible las consume.

Este documento no asume que MCP es la respuesta correcta para su contexto. Asume que usted necesita el criterio técnico para decidirlo.

En este documento encontrará:

- El diagnóstico preciso de por qué las integraciones artesanales no escalan — con la fórmula de complejidad que lo demuestra
- La especificación de MCP en términos concretos: qué define, qué transporta, qué deja fuera
- Lo que MCP no resuelve — la sección que la mayoría de los documentos sobre este protocolo omiten
- Tres patrones de implementación sobre infraestructura existente, incluyendo el caso de sistemas legados sin API
- Las implicaciones reales de seguridad: autenticación, autorización granular y control de qué ve el modelo
- Un test de usabilidad para saber si su servidor MCP está listo para producción antes de declararlo así
- El criterio para elegir el primer servidor que produce evidencia en menos de cuatro semanas

---

El problema que su equipo ya conoce

Si lleva más de tres meses intentando que un agente de IA use datos internos de forma confiable en producción, su equipo ya describió alguna versión de este escenario:

El agente funciona en el demo. En el demo, el contexto está preparado, los datos son los correctos, y la respuesta es convincente. En producción, el agente no tiene acceso a los sistemas reales, o tiene acceso parcial construido a mano para ese caso específico, o tiene acceso que se rompe cada vez que el sistema fuente cambia algo.

El resultado es predecible: el agente vive en un piloto permanente. O vive en producción con un alcance tan acotado que no justifica el esfuerzo de mantenerlo.

---

1. Por qué las integraciones artesanales no escalan

La forma en que la mayoría de los equipos conectan agentes de IA a sistemas internos produce una estructura de complejidad que crece cuadráticamente.

Si tiene N sistemas y M agentes, cada agente necesita su propia integración con cada sistema relevante. En el mejor caso, reutiliza código entre agentes. En la práctica, cada combinación tiene particularidades: el agente de soporte necesita el CRM en modo lectura con filtro por cliente; el agente de ventas necesita el CRM en modo lectura con filtro por oportunidad abierta y además acceso al catálogo de precios; el agente de dirección necesita ambos más las bitácoras de operación. Tres agentes, tres integraciones distintas al mismo CRM.

Multiplique eso por el número de sistemas que tiene.

Cada integración artesanal tiene su propio esquema de autenticación, su propio manejo de errores, su propia lógica de serialización del contexto, y su propia deuda de mantenimiento. Cuando el CRM actualiza su API — lo que ocurre con certeza, no como posibilidad — algo se rompe. El equipo descubre cuándo el agente produce una respuesta incorrecta o cuando un usuario reporta el problema.

El costo no es solo tiempo de desarrollo inicial. Es la carga operativa acumulada de mantener un grafo de integraciones que crece con cada agente nuevo y con cada actualización de cada sistema fuente.

MCP resuelve este problema con un cambio arquitectónico: en lugar de N×M integraciones punto a punto, cada sistema publica sus capacidades una vez mediante un servidor MCP, y cualquier agente compatible las consume sin integración adicional.

---

2. Qué es MCP — la especificación

MCP (Model Context Protocol) es un protocolo abierto, publicado por Anthropic en noviembre de 2024, que estandariza la comunicación entre clientes de IA (los agentes o los hosts que los ejecutan) y servidores que exponen capacidades — datos, herramientas, o prompts reutilizables.

La especificación define tres tipos de primitivas que un servidor MCP puede exponer:

Resources — datos que el cliente puede leer. Un recurso tiene un URI, un tipo MIME, y contenido. Puede ser estático (un documento de política) o dinámico (el estado actual de una orden de producción). El cliente solicita el recurso; el servidor lo entrega. El modelo recibe el contenido como contexto.

Tools — acciones que el cliente puede ejecutar. Una herramienta tiene un nombre, un esquema de parámetros en JSON Schema, y produce un resultado. El modelo decide cuándo invocar una herramienta y con qué argumentos; el servidor ejecuta la acción y devuelve el resultado. La herramienta puede ser de solo lectura (consultar el estado de un cliente) o con efectos (crear un ticket, actualizar un registro).

Prompts — plantillas de prompt que el servidor expone para que el cliente las use. Menos frecuentes en implementaciones iniciales, pero útiles para estandarizar instrucciones complejas que dependen de contexto del sistema.

El transporte es JSON-RPC 2.0. El protocolo soporta dos modos de transporte: stdio (para procesos locales) y HTTP con Server-Sent Events (para servidores remotos). La autenticación no está prescrita por el protocolo base — se implementa a nivel de transporte, típicamente con OAuth 2.0 o tokens de API sobre HTTPS.

La especificación completa está en modelcontextprotocol.io. Lo que importa para decisiones de arquitectura es entender qué está dentro del protocolo y qué está fuera.

---

3. Lo que MCP resuelve — y lo que no resuelve

Esta es la sección que la mayoría de los documentos sobre MCP omiten. Vale leerla antes de comprometerse con una implementación.

Lo que MCP resuelve bien

Estandarización de la interfaz de integración. Si construye su servidor MCP correctamente, cualquier cliente compatible puede consumir sus capacidades. Hoy eso incluye Claude, el ecosistema de herramientas de desarrollo de Anthropic, y un número creciente de plataformas de terceros. La superficie de integración pasa de N×M a N+M.

Separación de responsabilidades. El servidor MCP sabe cómo acceder a los datos y qué operaciones son válidas. El agente sabe cuándo y por qué necesita esos datos. Esa separación hace que ambos lados sean más fáciles de mantener y de razonar independientemente.

Portabilidad del contexto. Un servidor MCP construido hoy funciona con cualquier cliente compatible en el futuro. Si cambia el modelo de lenguaje que usa, o el framework de agentes, o el proveedor de IA, sus integraciones MCP no necesitan rehacerse.

Superficie de auditoría. Toda interacción entre un agente y sus sistemas pasa por el servidor MCP. Eso crea un punto único donde registrar qué consultó el agente, cuándo,

con qué parámetros, y qué devolvió el sistema. En entornos con requerimientos de cumplimiento, eso es significativo.

Lo que MCP no resuelve

No resuelve la calidad del contexto. Si sus datos están mal estructurados, duplicados, o desactualizados, el servidor MCP los entregará mal estructurados, duplicados, o desactualizados. MCP es un canal, no un transformador de calidad de datos.

No resuelve la lógica de qué contexto necesita el agente. Decidir cuándo invocar qué herramienta, con qué parámetros, y cómo usar el resultado sigue siendo responsabilidad del diseño del agente. MCP hace disponibles las capacidades; no determina cómo usarlas.

No prescribe autenticación ni autorización granular. El protocolo base no tiene un modelo de permisos incorporado. Usted implementa el control de acceso en el servidor. Eso da flexibilidad, pero también significa que la seguridad es su responsabilidad, no una garantía del protocolo.

No resuelve latencia de sistemas legados. Si su ERP tarda cuatro segundos en responder una consulta, el agente esperará cuatro segundos. MCP no agrega caché ni optimización de consultas por sí solo — eso es diseño del servidor.

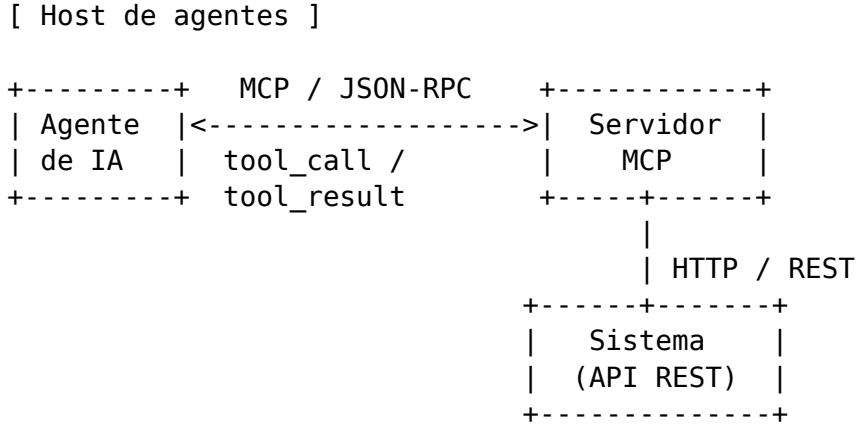
La especificación todavía está madurando. MCP 1.0 fue publicado en 2024. Hay áreas — particularmente en el modelo de autorización y en capacidades de streaming — que seguirán evolucionando. No es una especificación inestable, pero tampoco tiene el historial de diez años de REST. Vale diseñar el servidor con eso en mente.

---

4. Implementación sobre infraestructura existente — tres patrones

Patrón 1 — Sistema con API REST documentada

Cuándo aplica: El sistema fuente tiene una API REST con documentación razonable y autenticación estándar (API key, OAuth, JWT). Es el punto de partida más limpio.



Arquitectura: El servidor MCP actúa como una capa de traducción delgada entre el protocolo MCP y la API existente. Recibe una llamada de herramienta del agente,

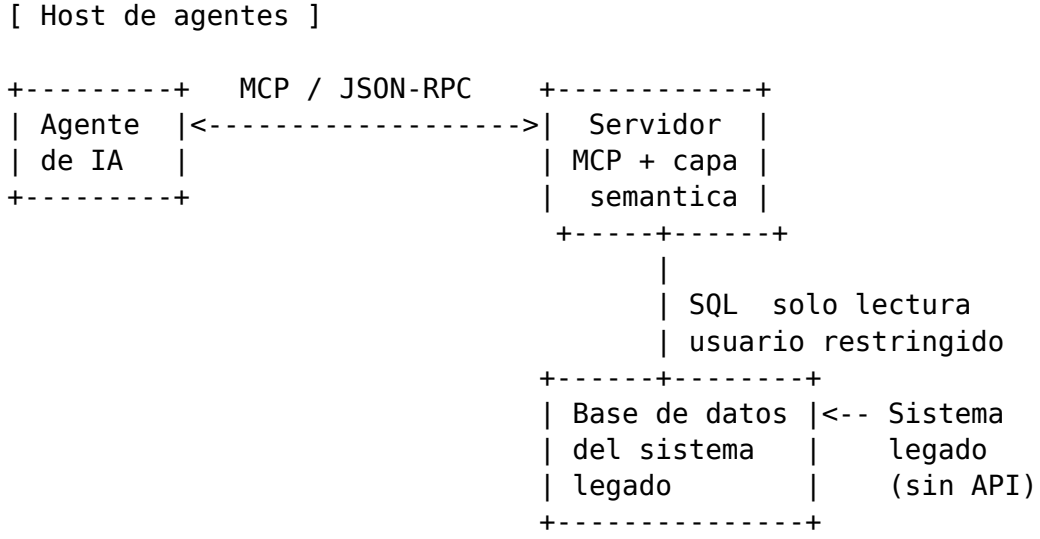
la traduce a la llamada HTTP correspondiente, recibe la respuesta, la serializa en el formato que el agente puede usar, y la devuelve.

Lo que implica: Un servidor MCP en este patrón es relativamente simple — en la mayoría de los casos, menos de 500 líneas de código en el lenguaje de su elección (hay SDKs oficiales para Python, TypeScript, y Java). La complejidad está en decidir qué subset de la API exponer y cómo modelar las herramientas para que el agente las use correctamente.

Riesgo principal: La granularidad de las herramientas. Si expone demasiado como una sola herramienta (ej: "consulta el CRM"), el agente tendrá dificultad para usarla bien. Si expone demasiado fino (ej: una herramienta por cada endpoint), el servidor se vuelve difícil de mantener. El equilibrio correcto depende de cómo el agente razona sobre las capacidades disponibles.

Patrón 2 — Sistema legado sin API

Cuándo aplica: El sistema fuente no tiene API. Tiene una base de datos relacional accesible, o un sistema de archivos estructurado, o un servicio con protocolo propietario. Es el caso más frecuente en organizaciones con más de diez años de historia tecnológica.



Arquitectura: El servidor MCP accede directamente a la fuente de datos — típicamente la base de datos — con una capa de read-only controlada. No se modifica el sistema legado. Se crea un usuario de base de datos con permisos de solo lectura sobre las tablas relevantes, y el servidor MCP expone vistas curadas de esos datos como resources o como resultados de herramientas.

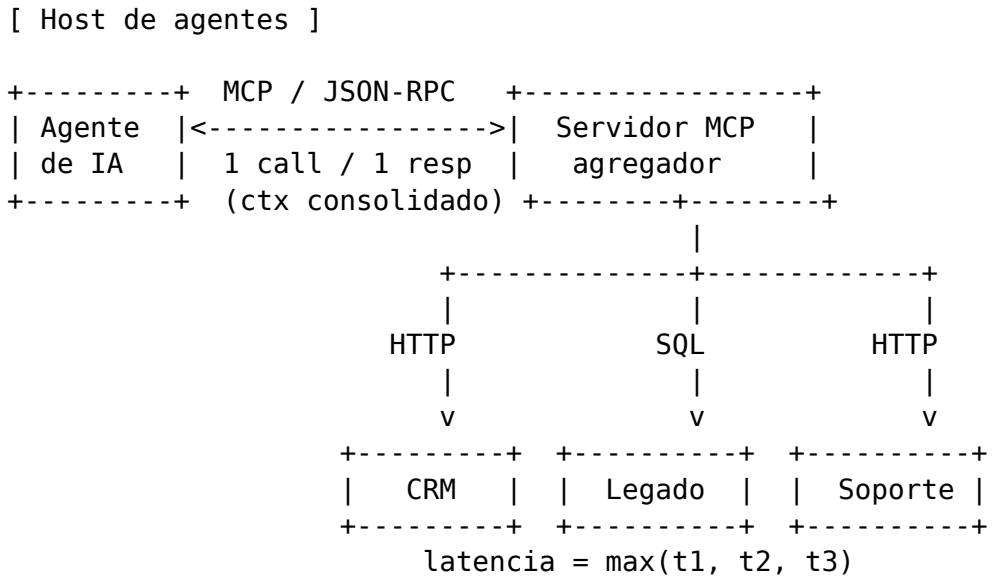
Lo que implica: Mayor responsabilidad en el servidor. El servidor no puede confiar en que la API del sistema legado valide sus parámetros — tiene que hacerlo él mismo. Las consultas SQL que el servidor ejecuta necesitan estar parametrizadas correctamente para evitar inyección. El esquema de la base de datos puede tener décadas de decisiones históricas que hay que interpretar antes de exponerlas.

Riesgo principal: El esquema de datos legado raramente corresponde al modelo

mental del agente. Una tabla con columnas de dos letras y valores codificados en números enteros requiere una capa de traducción semántica que no es trivial. Planifique tiempo para esa interpretación antes de asumir que el servidor será simple.

Patrón 3 — Múltiples sistemas heterogéneos

Cuándo aplica: El agente necesita contexto de varios sistemas al mismo tiempo, y consolidar ese contexto en el servidor es más eficiente que dejar que el agente haga múltiples llamadas independientes.



Arquitectura: Un servidor MCP agregador que expone herramientas compuestas. Una herramienta como `obtener_contexto_cliente` puede internamente consultar el CRM, el sistema de facturación, y las bitácoras de soporte, y devolver un objeto consolidado. El agente hace una llamada; el servidor hace tres.

Lo que implica: El servidor agrega complejidad operacional — ahora tiene múltiples dependencias, y si cualquiera de los sistemas fuente está caído, el servidor tiene que decidir qué hacer: fallar completamente, devolver contexto parcial con indicación de qué falta, o servir desde caché. Esa lógica de resiliencia hay que diseñarla explícitamente.

Riesgo principal: El servidor agrega latencia. Si los tres sistemas que consulta responden en paralelo en 800ms cada uno, el resultado llega en 800ms. Si responden en serie, en 2.4 segundos. El diseño del servidor tiene que forzar paralelismo donde sea posible.

---

5. Seguridad y control de acceso

Las preguntas de seguridad que importan en MCP no son diferentes de las que importan en cualquier sistema que expone datos internos. Lo que cambia es quién hace las preguntas y desde dónde.

Autenticación del cliente

El servidor MCP necesita saber que el cliente que le habla está autorizado. Para servidores remotos (HTTP + SSE), la recomendación actual de la especificación es OAuth 2.0. Para procesos locales (stdio), la autenticación ocurre a nivel del proceso — el servidor confía en el proceso que lo invoca.

En entornos empresariales, el cliente MCP típicamente es el host de agentes (el framework que ejecuta el agente), no el modelo directamente. La autenticación se configura entre el host y el servidor.

Autorización granular por herramienta

El servidor controla qué herramientas expone y con qué restricciones. No hay un modelo de permisos prescrito por el protocolo — usted lo implementa. Las opciones comunes:

- **Por herramienta:** algunos clientes pueden invocar solo un subset de herramientas
- **Por parámetro:** la herramienta acepta cualquier cliente, pero los valores de los parámetros se validan contra el perfil del cliente (ej: un agente solo puede consultar datos de los clientes asignados a su región)
- **Por volumen:** límites de tasa de invocación para prevenir uso abusivo

Qué ve el modelo

Este es el punto que más preocupa a los equipos de seguridad, y con razón. El contenido que devuelve el servidor MCP — el resultado de una herramienta, el texto de un resource — entra al contexto del modelo. Si ese contenido incluye información sensible, el modelo la tiene disponible para generar su respuesta.

La mitigación es diseño del servidor, no del protocolo. El servidor decide qué campos incluir en la respuesta. Si la consulta de un cliente devuelve datos bancarios, el servidor puede devolver solo los campos relevantes para el caso de uso del agente — no el objeto completo.

Una regla operativa útil: diseñe cada herramienta como si su respuesta pudiera aparecer en un log público. Si eso le incomoda, la herramienta está exponiendo más de lo que debería.

Auditoría

El servidor MCP es el punto natural de auditoría. Cada invocación de herramienta puede registrarse: timestamp, cliente, herramienta, parámetros, duración, resultado (o hash del resultado si el contenido es sensible). En entornos regulados, ese log es la evidencia de qué consultó el agente y cuándo.

---

6. Los desafíos reales de adopción técnica

“MCP es una iniciativa de Anthropic — si ellos pivotean, perdemos la inversión”

MCP fue publicado como especificación abierta con licencia MIT. La gobernanza está diseñada para ser independiente del proveedor. Lo más relevante: la adopción ya supera a Anthropic. OpenAI anunció soporte para MCP en sus APIs. Microsoft Copilot tiene integración MCP. AWS, Google Cloud, y Cloudflare tienen implementaciones propias. El ecosistema de herramientas de desarrollo — Cursor, Zed, Continue — tiene soporte MCP nativo.

Un protocolo con adopción multi-vendor no es una apuesta a un proveedor. Es una apuesta a un estándar.

“Agrega una capa más que puede fallar”

Correcto. Un servidor MCP es un proceso adicional con sus propias dependencias, su propio ciclo de vida, y su propia superficie de fallo. Eso es real.

La comparación relevante no es “MCP vs. nada”. Es “MCP vs. integraciones artesanales que ya tiene”. Una integración artesanal por agente también puede fallar — y cuando falla, el fallo es opaco, sin un punto de auditoría centralizado, y la corrección requiere entender el código específico de esa integración. Un servidor MCP centraliza los fallos en un punto instrumentable.

“Nuestros sistemas no tienen API”

El Patrón 2 de la sección anterior aplica directamente. La mayoría de los sistemas legados tienen una base de datos relacional accesible aunque no expongan API. Lo que implica ese camino en la práctica: crear un usuario de base de datos con permisos de solo lectura sobre las tablas relevantes, construir el servidor MCP con consultas parametrizadas, y dedicar tiempo real a la traducción semántica — los esquemas de datos con décadas de historia raramente hablan el mismo idioma que un agente moderno.

Ese trabajo de traducción semántica es donde se subestima el esfuerzo. No es complejidad técnica: es comprensión del dominio. Una columna llamada `EST_CLI` con valores `1`, `2`, `3` necesita un traductor que sepa que significa “activo”, “inactivo” y “en riesgo” antes de que el agente pueda hacer algo útil con ella. Planifique ese tiempo explícitamente.

El resultado, bien ejecutado, es un servidor MCP más seguro que cualquier integración artesanal que acceda a los mismos datos sin un punto de control centralizado — porque el servidor define exactamente qué se expone, con qué granularidad, y con auditoría completa de cada consulta.

“¿Cómo prevenimos que el modelo vea datos que no debe?”

El servidor controla exactamente qué devuelve. El diseño correcto expone herramientas de propósito específico, no acceso genérico a sistemas. “Obtener historial de com-

pras del cliente X” es una herramienta con respuesta controlada. “Consultar la base de datos con SQL arbitrario” no es una herramienta — es una vulnerabilidad.

“¿Qué pasa cuando la especificación cambia?”

MCP tiene versionado semántico y los cambios breaking se anuncian con anticipación. El diseño defensivo correcto es versionar su servidor explícitamente y actualizar cuando el ecosistema de clientes que usa lo requiera — no en cada release. La compatibilidad hacia atrás ha sido prioridad declarada del equipo de mantenimiento desde la publicación inicial, y la especificación de madurez actual ya fue cubierta en la sección anterior.

---

7. Por dónde empezar — el primer servidor MCP que vale la pena construir

El primer servidor MCP no debería ser el más ambicioso. Debería ser el que produce evidencia más rápido.

Los criterios para elegir el punto de entrada:

Alta frecuencia de consulta. Si el agente va a consultar ese sistema muchas veces por día, el valor de la integración se acumula rápido y los problemas de rendimiento aparecen pronto — cuando todavía es barato arreglarlos.

Datos bien estructurados. El primer servidor no es el momento para resolver ambigüedad semántica en datos legados. Elija un sistema cuya estructura de datos sea clara, con nombres de campos comprensibles y tipos consistentes.

Bajo riesgo de seguridad. Empiece con datos que no sean sensibles. Un catálogo de productos, una base de conocimiento interna, un registro de estados de operaciones. Demuestre que la arquitectura funciona antes de conectar datos financieros o de clientes.

Resultado medible. ¿Qué cambia si este agente tiene acceso a este sistema? Si no puede responder eso con un número o con un comportamiento observable, el caso de uso no está suficientemente definido.

El test de usabilidad que la mayoría omite

Antes de declarar el servidor listo para producción, haga esta prueba: tome un modelo que no haya visto el sistema antes. Dele acceso a las herramientas sin ninguna explicación en el system prompt sobre qué hacen ni cuándo usarlas. Plantee escenarios reales. Observe si el modelo las invoca correctamente, con los parámetros correctos, en el momento correcto.

Si el modelo no las usa bien sin instrucciones explícitas, el problema raramente está en el modelo. Está en cómo están nombradas y descritas las herramientas. Un servidor MCP exitoso no se mide por si el agente puede invocar sus herramientas — se mide por si las invoca correctamente sin ayuda.

Ese test es la diferencia entre un servidor que funciona en demos y uno que funciona en producción.

---

8. Siguientes pasos — para quien quiere ir más rápido de lo que su equipo puede solo

Si su equipo ya tiene el problema claro pero la implementación está bloqueada — por tiempo, por complejidad de los sistemas fuente, o por la curva de aprendizaje de la especificación — hay una conversación técnica que tiene sentido tener.

No es una presentación de producto. Es una revisión de arquitectura de treinta minutos donde trabajamos con los datos de su infraestructura y salimos con tres cosas concretas:

1. Cuál de sus sistemas actuales tiene el mayor valor de contexto para un agente — con el argumento técnico que lo justifica
2. Qué patrón de implementación aplica y qué complejidad real implica en su stack específico
3. El scope del primer servidor: qué expone, qué no expone, y por qué puede estar en producción en cuatro semanas o menos

El entregable de la sesión es ese scope — documentado, con criterios de éxito definidos, listo para que su equipo lo evalúe o lo ejecute.

Para agendar:

- → contacto@nfo.systems
- → ia.nfo.systems

Sobre IA NFO Systems

Llevamos trece años integrando sistemas en producción — ERP, CRM, bases de datos legadas, APIs propietarias, protocolos de industria. Cuando llegó MCP, reconocimos el patrón: es el mismo problema de integración que hemos resuelto de forma artesanal durante una década, ahora con un estándar que lo hace sostenible.

Implementamos servidores MCP sobre infraestructura existente. Conocemos el Patrón 2 — sistemas legados sin API — no como caso teórico sino como trabajo frecuente. Sabemos cuánto tarda la traducción semántica de un esquema de datos con veinte años de decisiones históricas y cuándo conviene no hacerla.

Si el problema que describe no es algo que podemos resolver bien, lo decimos antes de empezar.

> **Para el director general o dueño de su organización:** existe una versión ejecutiva de este documento que explica MCP sin tecnicismos, con tres escenarios de impacto

> en resultados de negocio y las cinco objeciones más frecuentes resueltas. Disponible en ia.nfo.systems o por solicitud a contacto@nfo.systems.

---

ia.nfo.systems · México
contacto@nfo.systems

Servicios provistos por NFO Systems SAPI de CV.