IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 05

MCP y conectividad

Aplicación

iacedemy.com — 2026

MÓDULO 05

MCP y conectividad

Nivel: Aplicación

Autor: Ricardo Gutierrez

Publicación: Mayo 2026

Plataforma: iacedemy.com

Que es MCP

Hasta ahora tu IA vive en una burbuja. Le copias texto, responde texto, tu aplicas a mano. Eso se acaba hoy. MCP (Model Context Protocol) es el protocolo que le da ojos, oidos y manos a Claude. Acceso directo a GitHub, bases de datos, navegador, APIs. Sin copiar. Sin pegar. Sin intermediarios.

Es probablemente el cambio mas grande en como interactuamos con la IA desde que aparecieron los chatbots. La analogia mas clara: piensa en MCP como USB pero para IA. USB es un conector universal que cualquier dispositivo puede implementar. Conectas un raton, un teclado, un disco duro, y funciona sin instalar drivers. MCP es lo mismo: cualquier herramienta puede implementar el protocolo y la IA la usa automaticamente.

Antes y despues de MCP

Antes: quieres que Claude revise un Pull Request. Abres GitHub, copias el diff, lo pegas en el chat, pides la review, copias la respuesta, vuelves a GitHub, pegas los comentarios. Cinco pasos manuales, tres minutos de tu tiempo.

Despues con MCP: le dices "revisa el PR 42" y el accede directamente a GitHub, lee el diff, analiza el codigo, y te da la review. Un comando. Diez segundos.

Arquitectura del protocolo

```
Tu peticion → Claude Code → Protocolo MCP → GitHub API
                                          → PostgreSQL
                                          → Browser (Puppeteer)
                                          → Filesystem
                                          → Exa (busqueda web)
                                          → Slack
```

Cada herramienta se expone como un "servidor MCP" que ofrece tools. Claude las descubre automaticamente y las usa cuando las necesita. Tu no tienes que decirle "usa el MCP de GitHub". Le dices "revisa mi ultimo PR" y el sabe que necesita la tool de GitHub para hacerlo. Es descubrimiento automatico de capacidades.

Tecnicamente, un servidor MCP es un proceso que se ejecuta en tu maquina (o en un servidor remoto) y habla el protocolo MCP con Claude Code. Cuando inicias Claude

Code con MCPs configurados, el se conecta a cada servidor, descubre que tools ofrece, y las anade a su repertorio de herramientas disponibles.

Que NO es MCP

- **No es una API.** Es un protocolo que estandariza como las IAs hablan con APIs.
- **No es magia.** Cada MCP solo hace lo que sus tools permiten. Si el MCP de GitHub no tiene una tool para borrar repos, la IA no puede borrar repos.
- **No es exclusivo de Claude.** Es un estandar abierto. Cursor, Windsurf, Cline y otros editores ya lo soportan.

Configuracion en settings.json

Toda la configuracion vive en un archivo: `settings.json`. Puede estar a tres niveles, y esto es importante entenderlo bien.

```
~/.claude/settings.json          # Global (tu usuario, todas las sesiones)
.claude/settings.json            # Proyecto (compartido con equipo via git)
.claude/settings.local.json      # Local (solo tu maquina, gitignored)
```

El global aplica a todas tus sesiones. El de proyecto se comparte con el equipo (se commitea). El local es solo para ti y se ignora en git. La logica es: configuracion compartible en el de proyecto, secretos y personalizaciones en el local.

Regla de oro que no puedes romper: los tokens van en el archivo local o en variables de entorno. NUNCA en el archivo de proyecto que se commitea. Un token en un commit es un token expuesto. Da igual que el repo sea privado hoy, puede ser publico manana.

Estructura basica de configuracion

```
{
 "mcpServers": {
   "github": {
     "command": "npx",
     "args": ["-y", "@modelcontextprotocol/server-github"],
```

```
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "${GITHUB_TOKEN}"
      }
    }
  }
}
```

La sintaxis `${GITHUB_TOKEN}` referencia una variable de entorno de tu sistema. El token real esta en tu shell (en `.zshrc` o `.bashrc`), no en el archivo de configuracion. Asi puedes commitear settings.json sin riesgo.

GitHub MCP en accion

Paso 1: Crear un Personal Access Token

Usa tokens fine-grained, no los clasicos. Los fine-grained te permiten dar permisos especificos a repos especificos. Permisos minimos necesarios: Contents (read), Pull requests (read/write), Issues (read/write). NADA mas. Principio de minimo privilegio. No des permisos que no necesitas.

Paso 2: Exportar y verificar

```
 # Exportar el token
export GITHUB_TOKEN="ghp_xxxxxxxxxxxxx"

# Verificar en Claude Code
claude
> /mcp
```

Si ves la lista de tools de GitHub, todo esta funcionando.

Demo 1: Listar repos

"Lista mis ultimos 5 repos con mas estrellas." Claude ejecuta list_repos, devuelve tabla con nombre, descripcion, estrellas. Sin MCP, tendrias que ir a github.com, ordenar por estrellas, copiar los datos. Con MCP, una frase.

Demo 2: Code review de un PR

"Lee el diff del PR numero 42 en mi-repo y hazme un code review con severidad." Claude accede al PR, lee los archivos modificados, genera un informe con findings, severidad (critical/warning/info) y sugerencias concretas.

Esto sin MCP requiere: abrir browser, ir al PR, copiar el diff (que puede ser de 500 lineas), pegarlo en Claude, pedir review, copiar la respuesta, volver al PR para comentar. Seis pasos manuales convertidos en un comando de una linea.

Demo 3: Busqueda transversal

"Busca en mi organizacion todos los archivos que usen console.log en produccion." Claude usa search_code, recorre multiples repos, devuelve la lista con archivo y linea. Imagina tener 20 repos y querer saber donde hay console.logs que no deberian estar en produccion. A mano, imposible de hacer rapido. Con MCP, 10 segundos.

PostgreSQL, Browser y Exa

PostgreSQL MCP

ATENCION: la regla mas importante de este bloque. Siempre solo lectura. Siempre base de datos de desarrollo. NUNCA produccion. Si conectas Claude a tu base de datos de produccion con permisos de escritura, es cuestion de tiempo hasta que algo salga mal.

```
 {
  "mcpServers": {
    "postgres": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-postgres"],
      "env": {
        "DATABASE_URL": "postgresql://readonly_user:pass@localhost:5432/
dev_db"
      }
    }
  }
}
```

Nota el usuario: `readonly_user`. Crea un usuario de PostgreSQL que solo tiene permisos de SELECT. Nada de INSERT, UPDATE, DELETE. Aunque Claude decida que necesita modificar datos, el usuario de la base de datos no se lo permite. Doble barrera de seguridad.

Ejemplo: "Cuantos usuarios se registraron este mes y cual es la tendencia semanal?" Claude escribe la query SQL, la ejecuta contra tu base de datos de desarrollo, obtiene los datos reales, y te da un analisis con tendencia. Todo en un comando.

Browser MCP (Puppeteer)

```
{
  "mcpServers": {
    "puppeteer": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-puppeteer"]
    }
  }
}
```

Un navegador real que Claude controla. Puede navegar, hacer clic, rellenar formularios, tomar capturas. Util para testing visual, scraping de datos publicos, o verificar que tu deploy se ve bien.

Ejemplo: "Ve a mi landing page, toma un screenshot y dime 3 mejoras de UX." Claude abre el navegador headless, navega, captura screenshot, analiza y sugiere mejoras concretas.

Exa MCP (busqueda web inteligente)

Exa es busqueda semantica. No es Google. Busca por significado, no por keywords. Si le pides "articulos sobre como empresas usan agentes IA en produccion", encuentra articulos relevantes aunque no contengan esas palabras exactas.

Ejemplo: "Busca los 5 articulos mas relevantes sobre MCP publicados este mes." Claude busca con Exa, devuelve articulos reales con titulo, URL, fecha y resumen.

12 MCPs recomendados

Categoria	MCP	Para que
Codigo	GitHub	Repos, PRs, issues, code search
Codigo	GitLab	Lo mismo para repos GitLab
Datos	PostgreSQL	Queries a base de datos
Datos	Supabase	DB + Auth + Storage
Web	Puppeteer	Navegar, screenshot, scraping
Web	Exa	Busqueda semantica de contenido
Archivos	Filesystem	Leer/escribir fuera del proyecto
Comunicacion	Slack	Leer/enviar mensajes, canales
Comunicacion	Gmail	Leer/enviar emails
Productividad	Google Calendar	Eventos, disponibilidad
Productividad	Notion	Paginas, bases de datos
DevOps	Docker	Contenedores, logs, compose

La regla: no instales los 12 a la vez. Empieza con GitHub + Filesystem. Son los que mas valor dan desde el dia uno. Anade los demas cuando los necesites realmente. Cada MCP que agregas es un servidor mas que se ejecuta, consume recursos, y anade complejidad.

Workflows conectados y seguridad

Workflow combinado: Health Check

Combinamos GitHub MCP + agente reviewer + hooks en un solo workflow.

```
 # .claude/commands/health-check.md
Analiza la salud de este repositorio:

1. Lee los PRs abiertos via GitHub MCP
```

```
2. Lee los issues abiertos
3. Analiza la estructura del proyecto
4. Verifica si hay dependencias con CVEs conocidos
5. Genera un informe con score de 1 a 10

Formato del informe:
- Score general
- PRs pendientes (edad, autor)
- Issues criticos
- Deuda tecnica detectada
- Recomendaciones top 3
```

Resultado: Score 7/10. Areas de mejora: 2 PRs con mas de 1 semana sin review, coverage de tests al 62%, 1 dependencia con CVE conocido. Todo real. Todo automatico. Todo en un comando que cualquier miembro del equipo puede ejecutar.

Seguridad MCP: 6 reglas no negociables

1. **Tokens en variables de entorno.** NUNCA en archivos commiteados. NUNCA. Ni siquiera en repos privados.
2. **PostgreSQL SIEMPRE con usuario de solo lectura.** Crea un usuario `readonly_mcp` con solo SELECT. Sin excepciones.
3. **GitHub token con permisos minimos.** Fine-grained, solo los repos que necesitas, solo las acciones que necesitas.
4. **Circuit breakers en workflows con loops.** Limita a 10 iteraciones maximo. Un loop sin limite quema tokens y puede hacer dano.
5. **Hooks de validacion.** Un hook PreToolUse que pida confirmacion antes de acciones destructivas.
6. **Rotacion de tokens cada 90 dias.** Pon un recordatorio en tu calendario. No negociable.

Hook de auditoria para queries SQL

```
{
  "hooks": {
    "PreToolUse": [{
      "matcher": "mcp__postgres__query",
      "command": "echo 'SQL query interceptada para auditoria' >> ~/.claude/
mcp-audit.log"
```

```
      }]
    }
  }
```

Este hook registra cada query SQL que Claude ejecuta. No la bloquea, pero la registra. Asi tienes un audit trail de todo lo que la IA hizo en tu base de datos.

Ejercicio practico

Ejercicio M05: Conecta tu primer MCP

1. Crea un Personal Access Token en GitHub (fine-grained, permisos minimos).
2. Exportalo como variable de entorno en tu `.zshrc` o `.bashrc`.
3. Configura el MCP de GitHub en `.claude/settings.json`.
4. Verifica con `/mcp` que las tools estan disponibles.
5. Ejecuta 3 operaciones: listar repos, leer un PR, buscar codigo.
6. Anade el hook de auditoria para loggear operaciones MCP.
7. Crea el comando `/health-check` y ejecutalo en tu proyecto.

Bonus: Configura el MCP de PostgreSQL con un usuario read-only contra tu base de datos de desarrollo y ejecuta 3 queries de analisis.

Conclusiones clave

Key takeaways del M05

1. MCP es USB para IA: un protocolo universal que conecta Claude con herramientas externas.
2. La configuracion vive en settings.json a 3 niveles: global, proyecto y local.
3. Tokens SIEMPRE en variables de entorno, NUNCA en archivos commiteados.
4. PostgreSQL SIEMPRE con usuario read-only. NUNCA produccion.
5. Empieza con GitHub + Filesystem. Anade MCPs solo cuando los necesites.

> 6. 6 reglas de seguridad no negociables: tokens seguros, permisos minimos, circuit breakers, hooks, rotacion.

Tu IA ya no esta ciega

Con MCP, Claude tiene acceso directo a tu codigo, tus datos y tus herramientas. En el M06 usamos todo esto para producir contenido profesional: imagenes, video, audio y texto a escala.

Ir al Modulo 06

IACADEMY

iacedemy.com

De fundamentos a arquitectura de IA.
12 módulos prácticos. 24 recursos descargables.
Quizzes con certificado. Vídeos profesionales.

Empieza gratis en iacedemy.com/free