

MCP in pratica ⇌

Tre integrazioni reali passo-passo: Sentry per il debugging in produzione, GitHub per issue e PR, il database in sola lettura. Più risorse @, prompt /mcp__ e il server dedicato a un subagente.

Aggiornato: luglio 2026 · Fonte: documentazione ufficiale Anthropic (code.claude.com/docs) · Vol. 12 (extra) della serie — teoria e sicurezza nel vol. 7.

1. Esempio 1: Sentry — errori di produzione in chat

HTTP remoto + OAuth

```
# 1. aggiungi il server remoto
claude mcp add --transport http sentry \
  https://mcp.sentry.dev/mcp

# 2. in sessione: autenticali nel browser
/mcp

# 3. e poi chiedi, in italiano
"Quali sono gli errori più frequenti
nelle ultime 24 ore?"
"Mostrami lo stack trace dell'errore abc123"
"Quale deployment ha introdotto
questi errori nuovi?"
```

Il flusso OAuth, una volta sola

- Il server risponde 401/403 → `/mcp` lo marca "da autenticare": scegli il server, si apre il browser, login e via. I token sono salvati e **rinnovati da soli**; "Clear authentication" nel menu revoca.
- Da terminale: `claude mcp login sentry`; via SSH aggiungi `--no-browser` e apri l'URL stampato sulla tua macchina.
- Se un refresh fallisce, Claude Code ti punta subito a `/mcp` → "Re-authenticate" (v2.1.195+): niente tool call che muoiono in silenzio.

2. Esempio 2: GitHub — issue e PR senza cambiare finestra

```
# token fine-grained da
# github.com/settings/personal-access-tokens
# con accesso ai soli repo che servono
claude mcp add --transport http github \
  https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer IL_TUO_PAT"

# poi:
"Fai la review della PR #456 e suggerisci
miglioramenti"
"Apri una issue per il bug che abbiamo
appena trovato"
"Mostrami le PR aperte assegnate a me"
```

Le regole del token

- Auth **statica nel header** (niente OAuth): il PAT fine-grained limita il raggio d'azione ai repo scelti. Se il server rifiuta il header configurato, la connessione risulta fallita — Claude Code *non* ripiega su OAuth: controlla il token o toglilo il header.
- Occhio allo scope: il comando sopra salva in `local` (default, solo tu, solo quel progetto). Il token **non** va in `--scope project / .mcp.json`, che finisce in git.
- Permessi granulari sui tool: `mcp_github_get_*` in allow, oppure `mcp_github` in deny per spegnere tutto il server (vol. 7).

3. Esempio 3: il database interrogabile

stdio locale + DSN read-only

```
# server stdio: gira sulla tua macchina;
# il -- separa i flag di Claude dal comando
claude mcp add --transport stdio db -- \
  npx -y @bytebase/dbhub \
  --dsn "postgres://readonly:pass@prod.db.com:5432/
analytics"

# poi interroga in linguaggio naturale:
"Qual è il fatturato totale di questo mese?"
"Mostrami lo schema della tabella orders"
"Trova i clienti senza acquisti da 90 giorni"
```

La sicurezza sta nel DSN

La misura di sicurezza vera è l'utenza `readonly` nel **database**: qualunque cosa Claude chieda, il DB rifiuta le scritture. Difesa in profondità: la stessa filosofia dell'hook che filtra le SELECT nel vol. 10 — lì a livello subagente, qui a livello credenziali. Per le credenziali usa `--env` o tienile fuori da `.mcp.json` condiviso.

Ricorda (vol. 7)

- I server di un `.mcp.json` appena clonato restano in attesa di approvazione; reset con `claude mcp reset-project-choices`.
- Stato e conteggio tool: `/mcp`; da shell `claude mcp list / get / remove`.

4. Oltre i tool: risorse @ e prompt /

il server come fonte e come comando

Risorse: @server:protocollo://path

- Digita @ e le risorse dei server connessi appaiono accanto ai file, con fuzzy search.
- "Analizza @github:issue://123 e proponi un fix"
- Più risorse nello stesso prompt: "Confronta @postgres:schema://users con @docs:file://database/user-model"
- La risorsa viene fetchata e allegata da sola: niente copia-incolla.

Prompt del server: /mcp_server_prompt

- I prompt esposti da un server diventano comandi: digita / e compaiono come /mcp__nomeserver__nomeprompt.
- Senza argomenti: /mcp_github_list_prs
- Con argomenti, separati da spazio: /mcp_github_pr_review 456 · /mcp_jira_create_issue "Bug login" high
- Scoperti dinamicamente dai server connessi; il risultato viene iniettato in conversazione.

5. Il server di un solo subagente

ponte col vol. 10

```
# .claude/agents/browser-tester.md
# il server esiste solo mentre
# il subagente lavora
---
name: browser-tester
description: Testa le feature in un
  browser reale con Playwright.
mcpServers:
  - playwright:
      type: stdio
      command: npx
      args: ["-y", "@playwright/mcp@latest"]
---
```

Quando conviene

- Server usato solo da un flusso specifico? Definiscilo **inline nel subagente**, non in .mcp.json: il main non paga le definizioni dei tool in contesto e il server si accende/spegne col worker.
- Il tool search (attivo di default) rende comunque leggeri i server con tanti tool: in contesto entrano solo i nomi finché un tool non serve. Per i pochi tool che servono sempre: "alwaysLoad": true sul server in .mcp.json.

► Checklist prima di connettere

1. Il server è nella Anthropic Directory (claude.ai/directory) o l'hai ispezionato? I server che fetchano contenuti esterni espongono a prompt injection. **2.** Credenziali nel posto giusto: scope local o --env, mai token in .mcp.json committato. **3.** Minimo privilegio: PAT fine-grained, utenze DB read-only, regole mcp_server_tool nei permessi. **4.** Connetti a inizio sessione: aggiungere server a metà invalida la cache dei prompt (vol. 3).

Fonti: code.claude.com/docs — mcp (practical examples, authenticate, resources, prompts, tool search) · Teoria, transport e scope: vol. 7 · Subagenti: vol. 10.