MCP in practice ⇄

Three real integrations step by step: Sentry for production debugging, GitHub for issues and PRs, the read-only database. Plus @ resources, /mcp__ prompts and the server dedicated to a subagent.

Updated: July 2026 · Source: official Anthropic documentation (code.claude.com/docs) · Vol. 12 (extra) of the series — theory and security in vol. 7.

1. Example 1: Sentry — production errors in chat

remote HTTP + OAuth

```
# 1. add the remote server
claude mcp add --transport http sentry \
  https://mcp.sentry.dev/mcp

# 2. in session: authenticate in the browser
/mcp

# 3. then ask, in plain English
"What are the most common errors
 in the last 24 hours?"
"Show me the stack trace for error abc123"
"Which deployment introduced
 these new errors?"
```

The OAuth flow, once only

- The server responds 401/403 → `/mcp` marks it "needs authentication": pick the server, the browser opens, log in and go. Tokens are stored and **refreshed on their own**; "Clear authentication" in the menu revokes.
- From the terminal: `claude mcp login sentry`; over SSH add `--no-browser` and open the printed URL on your machine.
- If a refresh fails, Claude Code points you straight to `/mcp` → "Re-authenticate" (v2.1.195+): no tool calls dying silently.

2. Example 2: GitHub — issues and PRs without switching windows

```
# fine-grained token from
# github.com/settings/personal-access-tokens
# with access to only the repos you need
claude mcp add --transport http github \
  https://api.githubcopilot.com/mcp/ \
  --header "Authorization: Bearer YOUR_PAT"

# then:
"Review PR #456 and suggest
 improvements"
"Open an issue for the bug we
 just found"
"Show me the open PRs assigned to me"
```

The token rules

- **Static auth in the header** (no OAuth): the fine-grained PAT limits the blast radius to the chosen repos. If the server rejects the configured header, the connection shows as failed — Claude Code does *not* fall back to OAuth: check the token or remove the header.
- Watch the scope: the command above saves to `local` (default, only you, only that project). The token does **not** belong in `--scope project` / `.mcp.json`, which ends up in git.
- Granular tool permissions: `mcp__github__get_*` in allow, or `mcp__github` in deny to switch off the whole server (vol. 7).

3. Example 3: the queryable database

local stdio + read-only DSN

```
# stdio server: runs on your machine;
# the -- separates Claude's flags from the command
claude mcp add --transport stdio db -- \
  npx -y @bytebase/dbhub \
  --dsn "postgresql://readonly:pass@prod.db.com:5432/
analytics"

# then query in natural language:
"What is the total revenue this month?"
"Show me the schema of the orders table"
"Find customers with no purchases in 90 days"
```

The security lives in the DSN

The real security measure is the `readonly` user **in the database**: whatever Claude asks, the DB rejects writes. Defense in depth: the same philosophy as the hook that filters SELECTs in vol. 10 — there at the subagent level, here at the credentials level. For credentials use `--env` or keep them out of a shared `.mcp.json`.

Remember (vol. 7)

- Servers in a freshly cloned `.mcp.json` stay pending approval; reset with `claude mcp reset-project-choices`.
- Status and tool count: `/mcp`; from the shell `claude mcp list / get / remove`.

4. Beyond tools: @ resources and / prompts

the server as source and as command

Resources: @server:protocol://path

- Type `@` and the connected servers' resources appear alongside files, with fuzzy search.
- "*Analyze `@github:issue://123` and propose a fix*"
- Multiple resources in the same prompt: "*Compare `@postgres:schema://users` with `@docs:file://database/user-model`*"
- The resource is fetched and attached on its own: no copy-paste.

Server prompts: /mcp__server__prompt

- Prompts exposed by a server become commands: type `/` and they appear as `/mcp__servername__promptname`.
- Without arguments: `/mcp__github__list_prs`
- With arguments, separated by spaces: `/mcp__github__pr_review 456` · `/mcp__jira__create_issue "Bug login" high`
- Discovered dynamically from connected servers; the result is injected into the conversation.

5. The server of a single subagent

bridge to vol. 10

```
# .claude/agents/browser-tester.md
# the server exists only while
# the subagent is working
---
name: browser-tester
description: Tests features in a real
  browser with Playwright.
mcpServers:
  - playwright:
      type: stdio
      command: npx
      args: ["-y", "@playwright/mcp@latest"]
---
```

When it pays off

- Server used only by one specific flow? Define it **inline in the subagent**, not in `.mcp.json`: the main agent doesn't pay for the tool definitions in context and the server starts/stops with the worker.
- Tool search (on by default) keeps tool-heavy servers light anyway: only the names enter context until a tool is needed. For the few tools you *always* need: `"alwaysLoad": true` on the server in `.mcp.json`.

▶ Checklist before connecting

1. Is the server in the Anthropic Directory (claude.ai/directory), or have you inspected it? Servers that fetch external content expose you to prompt injection. **2.** Credentials in the right place: `local` scope or `--env`, never tokens in a committed `.mcp.json`. **3.** Least privilege: fine-grained PATs, read-only DB users, `mcp__server__tool` rules in permissions. **4.** Connect at session start: adding servers mid-session invalidates the prompt cache (vol. 3).

Sources: code.claude.com/docs — mcp (practical examples, authenticate, resources, prompts, tool search) · Theory, transports and scopes: vol. 7 · Subagents: vol. 10.