MCP done right ◆

Connecting Claude Code to databases, issue trackers, APIs and external services with the Model Context Protocol: transports, scopes, OAuth, timeouts and security.

Updated: July 2026 · Source: official Anthropic documentation (code.claude.com/docs) · Vol. 7 of the Claude Code cheatsheet series.

1. What it is and what it's for

stop copy-pasting data into the chat

MCP (Model Context Protocol) is the open-source standard for AI↔tool integrations. An MCP server gives Claude the **tools and data** of an external system: the sign you need one is catching yourself pasting content from another window into the chat — issue tracker, monitoring dashboard, database. Once connected, Claude reads and acts directly. Real-world examples: "implement the feature from JIRA issue ENG-4521 and open a PR on GitHub", "check feature usage in Sentry", "find the emails of 10 users who used it in PostgreSQL", "update the email template from the new Figma designs".

Rule number one: trust first, then connect

An MCP server is code (or a service) with access to your session. Servers that fetch external content expose you to **prompt injection** risk. Start with the reviewed connectors in the **Anthropic Directory** (claude.ai/directory) and inspect community ones before adding them. In an enterprise, admins can enforce server allowlists/denylists via managed configuration.

2. Installing servers

three transports, one crucial double dash

```
# remote HTTP (recommended for cloud services)
claude mcp add --transport http notion \
  https://mcp.notion.com/mcp

# with a Bearer token
claude mcp add --transport http secure-api \
  https://api.example.com/mcp \
  --header "Authorization: Bearer TOKEN"

# local stdio (processes on your machine)
claude mcp add --env AIRTABLE_API_KEY=KEY \
  --transport stdio airtable \
  -- npx -y airtable-mcp-server

# SSE: deprecated, use HTTP where possible
# WebSocket (servers that push events):
# via add-json with "type":"ws"
```

The -- is not decorative

For stdio servers, the double dash separates Claude's options (`--transport`, `--env`, `--scope`) from the server command: everything after it passes through untouched. Without it, Claude Code would try to interpret the server's flags (like `--port`) as its own.

Day-to-day management

- `claude mcp list / get / remove` from the shell; `/mcp` in-session shows status, tool counts and manages **OAuth** for remote servers.
- `claude mcp login <name>` runs OAuth from the CLI; `--no-browser` over SSH prints the URL to open elsewhere.
- Automatic reconnection for HTTP/SSE with exponential backoff (5 attempts); stdio servers are local processes and are not reconnected.
- Servers can hot-update their own tools (`list_changed` notifications).
- **Channels**: a server can also *push* messages into the session (CI results, alerts, chat) so Claude reacts while you're away — opt-in with `--channels`.

◆ Scopes: where the configuration lives

Scope	Loads in	Shared with the team	Stored in
`local`	Current project only	No (default)	`~/.claude.json`, under the project path
`project`	Current project only	**Yes, via git**	`.mcp.json` in the repo root
`user`	All your projects	No	`~/.claude.json`

Chosen with `--scope`. Servers from the `.mcp.json` of a freshly cloned repo stay **pending approval** until you trust the workspace: a repository cannot self-approve its own servers. Use `local` for credentials that must not end up in version control, `project` for the team standard, `user` for your cross-project tools.

3. Operations: timeouts, context, permissions

the details that make the difference

Knob	Effect
`MCP_TIMEOUT=10000 claude`	Server startup timeout (ms).
`"timeout": 600000 (in .mcp.json)`	Wall-clock limit **per individual tool call** for that server (min 1000 ms); overrides MCP_TOOL_TIMEOUT.
`MAX_MCP_OUTPUT_TOKENS=50000`	Raises the threshold above which an MCP tool's output triggers a warning (default 10,000 tokens).
`CLAUDE_CODE_MCP_TOOL_IDLE_TIMEOUT`	Calls to remote servers that stay silent for 5 minutes get aborted; this changes the window (0 = disable).
`Tool search (on by default)`	MCP tool definitions are **deferred**: only the names enter the context until a tool is actually needed. This is what makes servers with dozens of tools sustainable. The set of names must still stay stable to avoid invalidating the cache.

MCP in permissions and subagents

- Canonical tool name: `mcp__<server>__<tool>` (for plugin servers: `mcp__plugin_<plugin>_<server>__<tool>`).
- Rules: `mcp__github` in deny removes every tool of that server; `mcp__*` all MCP tools. Allow globs only after the literal server prefix: `mcp__github__get_*`.
- In subagents: the `mcpServers` field gives a worker a specific server (by already-configured name or inline definition); `disallowedTools: mcp__github` takes it away.
- Connect MCP servers **at session start**: mid-session connects/disconnects invalidate the prompt cache (vol. 3).

Building your own

The official plugin scaffolds most of it:

```
/plugin marketplace add \
  anthropics/claude-plugins-official
/plugin install \
  mcp-server-dev@claude-plugins-official
/mcp-server-dev:build-mcp-server
```

Claude asks about your use case and generates a remote HTTP or local stdio server. For stdio servers, Claude Code sets `CLAUDE_PROJECT_DIR` in the process environment: resolve project paths without depending on the working directory. Finally, plugins can **bundle** MCP servers: they connect on their own when the plugin is enabled — the cleanest way to distribute them to a team.

▶ MCP + skills: the winning pair

MCP gives Claude the **tools** (the database connection, the Slack API); a skill teaches it to **use them well** (your DB schema and the team's query patterns, the message formatting rules). Tools without usage knowledge = guesswork; knowledge without tools = theory. Together, ideally bundled in a plugin, they become the integration that installs in one command.

Sources: code.claude.com/docs — mcp, mcp-quickstart, managed-mcp, channels · Connector directory: claude.ai/directory · Standard: modelcontextprotocol.io