

Connect AI agents

Every Resourcecloud instance serves its own MCP endpoint at `/mcp`. One line and one URL - no source code, no Node, no binary.

Overview

The endpoint speaks the Model Context Protocol over streamable HTTP (stateless, no event stream). An agent signs in with your user name and an API token and acts as you: it sees exactly what you see and may do exactly what you may do. A tool call runs the same code as the REST API, so there is no second set of rules.

Not hosted: dispatching a task to a local coding-agent harness needs your local repository and stays in the Node server of `mcp-server/`.

1. Create a token

- 1 Open your **Profile** and go to **API Keys**.
- 2 Click **Add new API token**, give it a description (for example *My AI agent*).
- 3 Choose the access. **Read only** is the default and is right for most agents: they can look up tasks, time, leave, capacity, incidents and files, and change nothing. Choose **Read and write** only when the agent should book time, create tasks or comments, and so on.
- 4 Copy the token. It is shown once.

An agent reads text that other people wrote (task descriptions, comments, incident reports). Such text can contain instructions aimed at the agent. A read-only token makes sure that no such instruction can change anything.

2a. Command-line client

```
claude mcp add --transport http resourcecloud https://YOUR-INSTANCE/mcp \
  --header "Authorization: Basic $(printf 'USERNAME:TOKEN' | base64)"
```

Check with `claude mcp list`, or ask: *"Which of my tasks are due this week?"*

2b. Desktop apps and other clients

Clients that only speak stdio can use the `mcp-remote` bridge (needs Node on that computer):

```
{
  "mcpServers": {
    "resourcecloud": {
      "command": "npx",
      "args": ["-y", "mcp-remote", "https://YOUR-INSTANCE/mcp",
        "--header", "Authorization: Basic BASE64_OF_USERNAME:TOKEN"]
    }
  }
}
```

Any client with a field for HTTP headers works the same way: URL `https://YOUR-INSTANCE/mcp`, header `Authorization: Basic base64(username:token)`.

How tasks are written

A task's **Description** is only a short subheader of its name. The full brief - objective, hints, plan - is the task's **first comment**. Later comments may add details. The endpoint tells the agent so when it connects: to implement a task it reads all comments, and when it creates a task it keeps the Description short and writes the brief as the first comment. Comments are HTML, not Markdown (for example `<h4>`, `<ul>`, `<p>`).

Tools

Area	Tools
Tasks, projects, stories, comments	<code>list_tasks</code> , <code>get_task</code> , <code>create_task</code> , <code>update_task</code> , <code>delete_task</code> , and the same for projects, stories, comments, services, solutions
Teams, users, customers	<code>list_teams</code> , <code>get_team</code> , <code>list_team_members</code> , <code>list_users</code> , <code>get_user</code> , <code>list_customers</code> , ...
Time	<code>list_time_entries</code> , <code>book_time</code> , <code>update_time_entry</code> , <code>delete_time_entry</code> , <code>get_timer</code> , <code>start_timer</code> , <code>stop_timer</code>
Leave	<code>list_leave</code> , <code>request_leave</code> , <code>submit_leave</code> , <code>cancel_leave</code> , <code>withdraw_leave</code> , <code>list_absences</code>
Capacity	<code>get_capacity</code> , <code>list_reservations</code> , <code>create_reservation</code> , ...
Incidents	<code>list_incidents</code> , <code>get_incident</code> , <code>get_incident_sla</code> , <code>raise_incident</code> , <code>list_sla_violations</code> , <code>list_problems</code>
Meetings, files	<code>list_meetings</code> , <code>get_meeting_minutes</code> , <code>list_files</code> , <code>get_file</code> , <code>list_file_versions</code>
Planning	<code>list_story_tasks</code> , <code>list_story_blockers</code> , <code>set_story_blockers</code> , <code>add_task_to_story</code> , <code>remove_task_from_story</code> (a milestone is a story with team 0 and a date), <code>list_task_dependencies</code> , <code>add_task_dependency</code> , <code>remove_task_dependency</code> , <code>list_subtasks</code> , <code>create_subtask</code>
Custom fields, project templates	<code>list_custom_fields</code> , <code>create_custom_field</code> , ... (see the custom fields guide); <code>list_project_templates</code> , <code>get_project_template</code> , <code>save_project_as_template</code> , <code>update_project_template</code> , <code>delete_project_template</code> , and <code>create_project</code> with <code>template_id</code> (see the project templates guide)
Webhooks (read only)	<code>list_webhook_events</code> , <code>list_webhooks</code> , <code>list_webhook_deliveries</code> (see the webhooks guide); creating a webhook and its secret stays in the web UI and the REST API

Agile	<code>list_sprints</code> , <code>plan_sprint</code> , <code>list_sprint_tasks</code> , <code>list_epics</code> , <code>list_releases</code> , ...
-------	---

A read-only token is offered the read tools only. The tools correspond one to one to the routes in `api/openapi.yaml`.

Security

- **Same rules as the pages and the REST API.** Another team's data, other people's hours and leave, and other people's private tasks are refused - also for an RC Admin where the pages refuse them.
- **Token scope.** A read-only token never writes, on `/mcp` and on the REST API alike. Tokens created before this feature became read-write and keep working.
- **Origin check.** Requests from a browser page of another site are refused.
- **Rate limit.** 120 calls per minute and user; then HTTP 429.
- **Log.** Every call is recorded with user, tool, time and result status (table `mcp_log`, kept 30 days). Arguments and results are not stored.
- Use HTTPS. Basic authentication sends the token with every request.

Limits

- Sign-in is by token, not OAuth, so claude.ai connectors (which need OAuth) cannot connect yet.
- No file download, no autoplanner or importance values yet (they need an API route first).
- Answers over 200 KB are cut off with a hint; use `limit / offset` or a filter.