MCP & AGENT INTEGRATION

OpenSAI Agent Integration Guide

Connect MCP-compatible AI agents with the OpenSAI Usage Skill to turn natural-language requests into explainable distributed executions.

Overview | Quickstart | User Manual | FAQ

Once connected, MCP-capable clients—such as Claude Desktop, Claude Code, Cursor, or OpenClaw—can inspect online edge devices, discover registered applications, submit deadline- or quality-driven jobs, track execution progress, and explain why OpenSAI chose specific devices, CPU/GPU resources, and tuning knobs. For example, you can ask: *"Run `cluster_analytics` with 30,000 points per shard and a 3-second deadline, then report the result, elapsed time, quality score, and chosen knob settings."*

OpenSAI's agent stack separates execution mechanics from operational guidance:

- **MCP Server (`opensai_mcp`)**: Exposes 17 structured `opensai_*` tools for cluster discovery, job submission, app management, rate limiting, and auditing;
- **Usage Skill (SKILL.md)**: Teaches the agent how to prioritize registered applications, map user intent into QoS and knob arguments, poll idempotently without duplicate submissions, and interpret measured metrics accurately.



OpenSAI Agent MCP and Usage Skill Architecture

> ✦ **Tip**
>
> **Recommended path**: Start with **local stdio** on one machine (Sections 1–4) for a zero-token end-to-end setup. Use Sections 5–6 when sharing a remote cluster over HTTP or installing a lightweight client on your laptop.

1. Prepare the MCP Server Environment

First launch OpenSAI using the Quickstart Guide and verify that `opensai devices list` shows an online device. Then install the agent dependencies and run the built-in connectivity check from the project root:

Linux / macOS:

```bash
bash scripts/setup.sh --agent
source .venv/bin/activate
python -m opensai_mcp --check
```

Windows PowerShell:

```powershell
powershell -ExecutionPolicy Bypass -File scripts\setup.ps1 -Agent
.venv\Scripts\Activate.ps1
python -m opensai_mcp --check
```

(In an existing `.venv`, you can equivalently run `python -m pip install -r Offloading-System/requirements-agent.txt`.) Once `--check` passes, `opensai_mcp` can reach the hub and read its live application and algorithm catalogs.

2. Connect Over Local stdio

Register a local stdio MCP server named `opensai` in your client with these fields:

Setting	Value
`command`	**Absolute path** to the Python interpreter inside `.venv`
`args`	`["-m", "opensai_mcp"]`
`cwd`	Absolute path to your extracted OpenSAI project root
`env`	`OPENSAI_HOST` (default `127.0.0.1`) and `OPENSAI_PORT` (default `8888`)

Clients using the standard `mcpServers` JSON format can adapt this configuration (replace `/path/to/OpenSAI_Release` with your local path):

```json
{
  "mcpServers": {
    "opensai": {
      "command": "/path/to/OpenSAI_Release/.venv/bin/python",
      "args": ["-m", "opensai_mcp"],
      "cwd": "/path/to/OpenSAI_Release",
      "env": {
        "OPENSAI_HOST": "127.0.0.1",
        "OPENSAI_PORT": "8888"
      }
    }
  }
}
```

On Windows, point `command` to `.venv\Scripts\python.exe` and escape backslashes as `\\` inside JSON strings (for example, `"C:\\OpenSAI_Release\\.venv\\Scripts\\python.exe"`). Reload the MCP connection or open a fresh session—tools such as `opensai_list_devices` and `opensai_list_apps` should appear.

3. Load the Usage Skill

Inside the extracted `OpenSAI_Release.zip` directory, equivalent Usage Skills are provided in both languages:

- English: `skills/en/opensai/SKILL.md`
- Chinese: `skills/zh-CN/opensai/SKILL.md`

Copy the `opensai/` folder for your preferred language into your client's skills directory, or import it through the client UI (both define the skill name `opensai`, so install one version). For clients without skill-directory support, paste the body of `SKILL.md` into workspace-level custom instructions.

> ◆ **Important**
>
> **Order matters**: Loading `SKILL.md` provides operational guidance only—it does not establish a server connection. Always configure the MCP server (Section 2) first so the `opensai_*` tools are available.

4. Verify with a Guided First Conversation

Run these three prompts in sequence to confirm end-to-end operation:

1. **Read-only discovery**: *"Inspect the online OpenSAI devices and registered applications, and tell me what I can run right now."*
2. **Baseline job**: *"Submit a pi calculation with 200,000 samples, wait for completion, and report the output and total duration."*
3. **Goal-driven adaptation** (after registering `cluster_analytics` and installing `numpy` on workers): *"Run `cluster_analytics` with 30,000 points per shard and a 3-second target. Report the output, elapsed time, quality score, and selected knob settings."*

If a longer run is still in progress after one wait window, say: *"Continue checking the job ID we already received without submitting a new run."*

> ✓ **Verification Check**
>
> The agent lists real online devices, returns actual job outputs, and the exact same `job_id` appears under **Runs** in the Web console (`http://127.0.0.1:5000`).

5. Connect Over Remote Streamable HTTP

When your AI client runs on a separate machine from the MCP host, generate a strong random token on the MCP server machine and start HTTP mode:

```bash
export OPENSAI_HOST=127.0.0.1
export OPENSAI_MCP_TOKEN="$(python -c 'import secrets; print(secrets.token_urlsafe(32))')"
echo "Bearer token: $OPENSAI_MCP_TOKEN"
python -m opensai_mcp --http --listen 0.0.0.0 --listen-port 8765
```

Configure your remote MCP client with:

Client Setting	Example
Transport	`Streamable HTTP`
Server URL	`http://192.168.1.10:8765/mcp` (use your MCP host's reachable address)
Authorization Header	`Bearer <OPENSAI_MCP_TOKEN>`

> ▲ **Warning**
>
> **Security & file-path semantics**: Non-loopback HTTP listeners require a Bearer token by default (missing or mismatched tokens return `HTTP 401`); use a VPN / Tailscale or HTTPS reverse proxy outside trusted LANs. In remote mode, `path` arguments refer to the **MCP server host's filesystem**—when registering an application from your laptop, either copy the folder to the MCP host or pass `manifest` and `files` source text directly via the registration tool.

6. Lightweight Client-Only Setup (Connect to an Existing Cluster)

If your team already operates an OpenSAI hub and worker cluster, you can set up a lightweight local environment on your laptop by cloning github.com/INSAI-institute/OpenSAI (`git clone https://github.com/INSAI-institute/OpenSAI.git`) or extracting `OpenSAI_Release.zip`—installing only the SDK, CLI, and MCP server without Redis or worker runtime packages:

Linux / macOS:

```bash
python -m venv .venv
source .venv/bin/activate
python -m pip install ".[mcp]"
python -m opensai_mcp --hub-host 192.168.1.10 --check
```

Windows PowerShell:

```powershell
python -m venv .venv
.venv\Scripts\Activate.ps1
python -m pip install ".[mcp]"
python -m opensai_mcp --hub-host 192.168.1.10 --check
```

Replace `192.168.1.10` with your hub's IP address, then configure local stdio (Section 2) and load `skills/en/opensai/SKILL.md` (Section 3).

7. MCP Tool Reference & Parameter Conventions

Tool Name	Purpose
`opensai_list_devices`, `opensai_cluster_status`	Inspect online workers, compute slots, and overall cluster health
`opensai_list_algorithms`, `opensai_catalog`	Discover available step algorithms, parameters, and knob schemas
`opensai_list_apps`	List registered applications with inputs, knobs, and default QoS
`opensai_run_app`, `opensai_submit_task`	Execute a registered application with goals, or run a single algorithm
`opensai_validate_dag`, `opensai_submit_dag`	Validate and submit an ad-hoc multi-step DAG
`opensai_get_result`, `opensai_wait_result`	Fetch a non-blocking status snapshot or wait for terminal completion

Tool Name	Purpose
`opensai_explain_job`	Retrieve placement, CPU/GPU core allocation, and knob selection rationale
`opensai_validate_app`, `opensai_register_app`, `opensai_unregister_app`	Validate, register, or remove authorized application packages
`opensai_list_data`, `opensai_upload_data`	Inspect device-local datasets or upload small files to chosen workers

Note that MCP tools accept QoS fields (`deadline_s`, `mode`, `min_quality`) at the top level, whereas the Python SDK groups them inside `qos={...}`. Example `opensai_run_app` payload:

```json
{
  "app": "cluster_analytics",
  "inputs": {"points": 30000, "seed": 1},
  "deadline_s": 3,
  "wait": true,
  "wait_timeout": 60
}
```

8. Team Access Control, Rate Limits & Auditing

Administrators sharing an MCP endpoint across a team can scope tool visibility, write permissions, and submission rates via environment variables:

Environment Variable	Default	Purpose
`OPENSAI_MCP_TOOLS`	All enabled	Comma-separated allowlist of exposed tool names
`OPENSAI_MCP_ALLOW_REGISTER`	`1`	Set `0` to disable application registration and removal
`OPENSAI_MCP_ALLOW_UPLOAD`	`1`	Set `0` to disable small-file dataset uploads
`OPENSAI_MCP_MAX_NODES`	`64`	Maximum nodes permitted in a single submitted DAG
`OPENSAI_MCP_MAX_WAIT_S`	`120`	Upper bound (seconds) per `opensai_wait_result` call
`OPENSAI_MCP_SUBMIT_RATE`	`2`	Per-second submission cap per submission tool and actor
`OPENSAI_MCP_ACTOR`	`openclaw`	Caller identifier recorded in audit logs and rate limiters
`OPENSAI_MCP_AUDIT_LOG`	Unset	Optional file path for JSONL tool-invocation audit logs

Because registering an application distributes Python step code to workers, only register packages supplied or explicitly approved by the user, and always run `opensai_validate_app` first.

9. Troubleshooting Connection Issues

Symptom	Resolution
No `opensai_*` tools appear in the client	Verify the absolute `.venv` Python path, `cwd`, and `--agent` installation; run `python -m opensai_mcp --check` in a terminal
Tools appear, but calls return connection errors	Confirm the OpenSAI hub is running and `OPENSAI_HOST` / `OPENSAI_PORT` point to a reachable hub (`8888` by default)
Remote HTTP returns `HTTP 401`	Ensure the client's `Authorization: Bearer <TOKEN>` matches `OPENSAI_MCP_TOKEN` on the MCP server
Application or upload path cannot be found	In remote HTTP mode, `path` resolves on the MCP host; stage files there or pass `manifest` + `files` source text
`opensai_wait_result` returns `done=false`	The job is still running in the background; poll again using the same `job_id`
Tool returns `RATE_LIMITED`	Wait for the returned `retry_after_s` interval before retrying

See the User Manual for full cluster and SDK documentation (中文智能体接入指南).