Coding Agents & IDEs

Point your editor or CLI at Radium. Claude Code, Cursor, VS Code, and the rest, each in four steps or fewer.

- Use Radium with Cursor
- Run Claude Code on Radium

Use Radium with Cursor

Connect Radium's Tycho model to Cursor using Cursor's built-in OpenAI-compatible API support. Settings, model ID, verification, and troubleshooting.

Use Radium with Cursor

Cursor has built-in support for OpenAI-compatible APIs, and Radium serves one, so connecting Tycho is four settings and no plugin.

This page covers Tycho specifically. Any Radium model reachable through the OpenAI-compatible endpoint can be added the same way by repeating step three with a different model identifier.

> **Scope** This connects Radium to Cursor Chat and Cursor Agent. Cursor Tab, the inline autocomplete, keeps using Cursor's own model regardless of what you select here. See About Tab autocomplete.

Before you start

- Cursor installed
- A Radium API key

Open model settings

In Cursor, go to:

```
Cursor Settings → Models
```

Connect the API

Radium's API is OpenAI-compatible, so the connection goes under the OpenAI section of the Models panel, not Anthropic. Under API Keys, set:

Setting	Value
OpenAI API Key	Your Radium API key
Override OpenAI Base URL	`https://api.radium.cloud/v1`

Enable both toggles, **OpenAI API Key** and **Override OpenAI Base URL**. Neither takes effect while off.

> **The one to get right** This goes in the OpenAI fields even though you are not using OpenAI. Cursor does not have a generic custom-provider slot, and the OpenAI-compatible one is the correct place for any OpenAI-shaped endpoint, Radium included.

Add Tycho

Still in the Models panel, select **Add Custom Model** and enter the identifier exactly:

```
tycho-1.0
```

Enable the model once it is added. It will not appear in the picker otherwise.

Verify the connection

Open a new Cursor Chat or Agent session and select `tycho-1.0` from the model picker. Send:

```
Reply with exactly: RADIUM_OK
```

Expected response:

```
RADIUM_OK
```

If you see that, the connection is live and Tycho is answering Chat and Agent requests.

About Tab autocomplete

Cursor Tab, the inline completion as you type, continues to run on Cursor's own model no matter what is selected in the Models panel. Radium serves `tycho-1.0` for Chat and Agent sessions only. This is a Cursor limitation, not a Radium one, and there is currently no setting that changes it.

When it does not work

- Confirm the base URL includes `/v1`. Without it, requests go to the wrong path.
- Confirm the model ID is exactly `tycho-1.0`. Cursor does not correct typos in a custom model entry.
- Confirm the **OpenAI API Key** toggle is enabled, not just filled in.
- Start a new Cursor session after selecting Tycho. An existing session can keep using whatever model it started with.
- Fully quit and restart Cursor if the new settings do not take effect. Cursor does not always pick up a provider change without a restart.

Next

- API quickstart, for calling Radium directly
- Use Radium with Claude Code
- Tool calling

Run Claude Code on Radium

Point the Claude Code CLI at Radium's Anthropic-compatible Messages API. Setup script, manual configuration, model selection, and troubleshooting.

Run Claude Code on Radium

Claude Code sends Anthropic-compatible API traffic to whatever endpoint you point it at, and Radium serves an Anthropic-compatible Messages API, so moving Claude Code across is a change of base URL, model name, and key.

Your projects stay where they are. So do your prompts, your permissions, your local tools, and any MCP servers you have configured. Claude Code keeps running the session and Radium answers the model calls.

> **Supported models** `hal-1.0`, `clarke-1.0`, and `tycho-1.0`. The examples on this page use `hal-1.0`, and any of the three can be substituted wherever it appears.

What changes

What you change

- The API base URL
- The model name
- Your credentials
- One settings file

What stays put

- Your prompts and project context
- Local tools and permissions
- MCP servers and their configuration
- Everything about how you work

Settings used throughout this guide

Setting	Value

API base URL	`https://api.radium.cloud`
Model	`hal-1.0`, `clarke-1.0`, or `tycho-1.0`
Authentication	Radium API key
Settings file	`~/.claude/settings.json`

> **Read this one twice** Claude Code appends `/v1/messages` to the base URL on its own, so leave it off. Setting `ANTHROPIC_BASE_URL` to the full Messages endpoint is the most common way this goes wrong, and it returns a 404 every time.

Before you start

- A Radium API key
- Claude Code, installed and current
- Node.js 18 or later, if you plan to run the setup script or the verifier
- macOS, Linux, or a supported Windows environment such as WSL or Git Bash

Check what you have:

```
node --version
claude --version
```

If Claude Code is not installed:

```
npm install -g @anthropic-ai/claude-code
```

Setup with the script

Run this from the folder containing the integration package:

```
./01-setup-radium-api.sh \
  --api-base api.radium.cloud \
  --model hal-1.0
```

The script asks for your API key without printing it, and then it does five things:

1. Normalises the base URL, so a trailing slash or a missing scheme does not break anything.
2. Backs up your existing `~/.claude/settings.json` with a timestamp.
3. Preserves every setting already in the file that has nothing to do with Radium.

4. Writes the base URL, the model, and the key.
5. Restricts the file permissions to your user.

The backup lands beside the settings file, named like this:

```
~/.claude/settings.json.bak.20260807153000
```

Nothing you had configured is lost, and you can put it back by hand in one move.

Setup by hand

If you would rather not run a script against your machine, add these entries to `~/.claude/settings.json` yourself:

```
{
  "env": {
    "ANTHROPIC_BASE_URL": "https://api.radium.cloud",
    "ANTHROPIC_MODEL": "hal-1.0",
    "ANTHROPIC_API_KEY": "YOUR_RADIUM_API_KEY"
  }
}
```

Keep any other settings that are already in the file, and keep the JSON valid. Then restrict access to it:

```
chmod 600 ~/.claude/settings.json
```

Shell environment variables work too, and they override the settings file, which is worth remembering when something behaves strangely later:

```
export ANTHROPIC_BASE_URL="https://api.radium.cloud"
export ANTHROPIC_MODEL="hal-1.0"
export ANTHROPIC_API_KEY="$RADIUM_API_KEY"
```

Verify the connection

Test Radium directly first, then test Claude Code. Doing it in that order tells you which of the two is at fault when something fails.

Step one, the endpoint on its own:

```
./02-test-radium-api.mjs --model hal-1.0
```

Step two, through Claude Code:

```
claude -p "Say only: radium model connected."
```

You should get back:

```
radium model connected.
```

Then open a project and work normally:

```
cd YOUR_PROJECT
claude
```

If step one passes and step two fails, the problem is in Claude Code's configuration and not in the endpoint.

Choosing a model

Primary model selection

Model	Built for
`hal-1.0`	Higher-capability coding and agentic work
`clarke-1.0`	General-purpose and structured work
`tycho-1.0`	Faster, lighter-weight tasks

To change it, run the setup script again with a different model:

```
./01-setup-radium-api.sh \
  --api-base api.radium.cloud \
  --model clarke-1.0
```

Another backup is written before anything is overwritten.

A faster model underneath

Claude Code reaches for a smaller model to handle helper operations during a session, and those operations are the kind of work Tycho is built for. Running Hal as the primary with Tycho underneath puts each task on the model sized for it, and the session gets quicker in the places

where capability was never the constraint.

```
{
  "env": {
    "ANTHROPIC_BASE_URL": "https://api.radium.cloud",
    "ANTHROPIC_MODEL": "hal-1.0",
    "ANTHROPIC_SMALL_FAST_MODEL": "tycho-1.0",
    "ANTHROPIC_API_KEY": "YOUR_RADIUM_API_KEY"
  }
}
```

Some Claude Code versions and some Radium deployments need this set explicitly, and others do not. Confirm the mapping with Radium before you rely on it in a managed deployment.

Bearer-token deployments

Radium deployments authenticate one of two ways. The default is an API key, and some deployments use a bearer token instead. Use whichever mode Radium supplied, and start with the API key when nobody has told you otherwise.

With the script:

```
./01-setup-radium-api.sh \
  --api-base api.radium.cloud \
  --model hal-1.0 \
  --auth bearer
```

By hand, swap the key variable for the token variable:

```
{
  "env": {
    "ANTHROPIC_BASE_URL": "https://api.radium.cloud",
    "ANTHROPIC_MODEL": "hal-1.0",
    "ANTHROPIC_AUTH_TOKEN": "YOUR_RADIUM_API_KEY"
  }
}
```

Set one or the other. Setting both is a 401 waiting to happen.

When it does not work

404 on every request The base URL has a path on it. Use `https://api.radium.cloud` and let Claude Code add `/v1/messages` itself.

401 or 403 Check that the key belongs to the environment you are pointing at, check whether the deployment expects an API key or a bearer token, and check that only one of `ANTHROPIC_API_KEY` and `ANTHROPIC_AUTH_TOKEN` is set. Restart Claude Code after any change.

Claude Code still appears to reach Anthropic Look at `~/.claude/settings.json` and confirm the base URL, then look for shell environment variables overriding it, because they win. Close any open sessions and start a new one. If you are still unsure, run the direct verifier, which separates an endpoint problem from a configuration problem.

Model not found Use the exact identifier issued for your account. This integration supports `hal-1.0`, `clarke-1.0`, and `tycho-1.0`.

Tools or streaming behave differently Claude Code uses streaming, tool calls, and more than one model route in a single session. API compatibility covers the shape of the request and it does not guarantee that every model responds the same way, so run the workflows your team actually uses before a broad rollout.

Security and operations

- Never commit `~/.claude/settings.json` or any populated credential file.
- The setup script stores your key locally in that settings file, so organisations that require short-lived credentials should use a managed key helper or a gateway instead.
- Prompts, tool results, and attached project context are data sent to the configured endpoint, and they should be treated that way.
- Use separate keys for development, staging, and production.
- Test quality, tool behaviour, latency, and usage on repositories that look like your real ones.

Point it at a real project, run a day of work through it, and you will know.

Next

- API quickstart, for calling Radium directly
- Tool calling, for the Messages API tool contract
- MCP and Radium, for how MCP servers fit alongside Claude Code
- Errors and troubleshooting