



DCSAI Technologies

TECHNICAL WHITEPAPER · v1.0 · MAY 2026

DCS Agent Studio

An IDE for building production AI agents

Code editor, visual builder, LLM collaborator. Three modes, one signed artifact.

Abstract

DCS Agent Studio is an IDE for authoring agents that run on the DCS Platform. It provides three editing modes: a code editor (for engineers who want to write `agent.yaml` directly), a visual builder (for non-engineers who want a drag-and-drop flow), and an LLM collaborator (for anyone who wants to describe what the agent should do in natural language and have the IDE write the `agent.yaml` for them). All three modes produce the same artifact — a signed, content-addressed agent that runs on the Platform.

Studio exists because authoring agents is a different skill from authoring traditional code. Prompts are not code; tests for non-deterministic systems are not unit tests; deploying an agent into production is closer to deploying a person than deploying a microservice. Studio is the workbench that makes these new skills as productive as possible.

Who this is for

Engineers building production agents. Product managers and analysts who want to author agents without engineering. Teams adopting AI for the first time who need a guided workbench. Educators teaching agent design. Anyone who has tried to write an agent in plain text and given up.

Contents

1	Introduction	3
2	Three editing modes	5
3	The Code Editor	8
4	The Visual Builder	11
5	The LLM Collaborator	14
6	Test Runner	17
7	Live Preview	19
8	Version Control + Git	21
9	Templates + Starters	24
10	Performance	26
11	Comparison vs. Cursor + Anthropic CUE + Make	28
12	Implementation Guide	30
13	References	32

1. Introduction

Code editors evolved over 50 years to support a specific workflow: write deterministic instructions, run them, observe the result, iterate. Agent authoring breaks several of those assumptions: the instructions are partly natural language, the results are partly random, and "running" an agent costs real money. A code editor optimised for deterministic code is the wrong tool.

Agent Studio takes the things a code editor gets right (syntax highlighting, autocomplete, multi-file project view) and adds the things agent authoring needs that the editor doesn't: a prompt editor with rendered preview, a test runner that knows how to evaluate non-deterministic output, a live sandbox that runs the agent against test inputs without spending real money, and a collaborator panel where the IDE itself helps you write the agent.

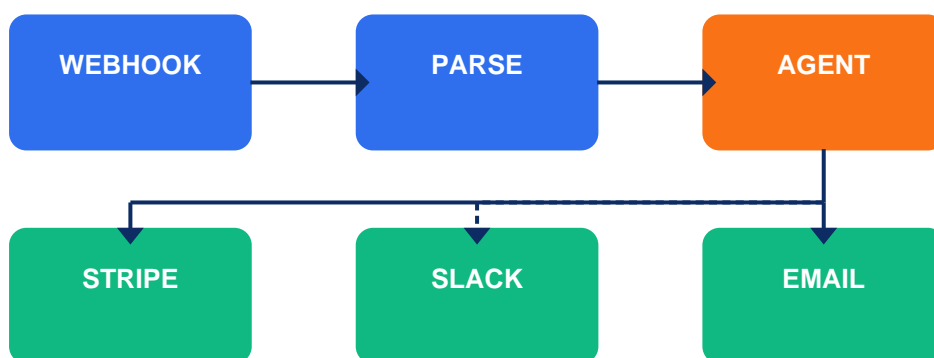
2. Three editing modes

Agent Studio IDE — file tree + prompt editor + test runner



Figure 2.1 — Code editor mode showing file tree, agent.yaml, tests, terminal.

Visual builder — drag, drop, connect



Visual builder compiles to the same agent.yaml as the code editor.

Figure 2.2 — Visual builder mode for drag-and-drop authoring.

3. The Code Editor

Built on the Monaco editor (same engine as VS Code). Adds DCS-specific extensions:

- Autocomplete for agent.yaml fields, tool capabilities, model names
- Inline validation of guardrails (e.g., max_refund must be a number)
- Hover-to-preview prompts (rendered markdown shown in popup)
- Live tool documentation (hover over a tool name to see the MCP server's capability docs)
- Test runner integrated in the side panel

4. The Visual Builder

Drag-and-drop canvas for non-engineers. Each node represents a step in the agent's flow: trigger, prompt, tool call, condition, branch. Edges represent data flow. The visual graph compiles to agent.yaml on save; you can switch to the code editor at any time to inspect or hand-edit.

5. The LLM Collaborator

A right-side panel where you describe what you want in natural language and the IDE writes the agent for you. The collaborator can:

- Generate a new agent from scratch: "Write me a refund-handler for Stripe with a \$500 cap"
- Modify an existing agent: "Add a Slack notification when the refund is over \$200"
- Suggest improvements: "Review this agent and tell me what could go wrong"
- Generate tests: "Write 5 test cases for this agent including edge cases"

6. Test Runner

Tests for non-deterministic systems work differently from unit tests. The DCS test runner supports three assertion styles:

- **Exact match** — for tool calls (the agent should call `Stripe.refund_create` with \$42.50)
- **Schema match** — for outputs that should have a certain structure (JSON shape, required fields)
- **LLM judge** — for outputs that should "look right" (a second model evaluates the first model's output against a rubric)

7. Live Preview

A sandboxed runtime that executes the agent against test inputs without billing real money. Tool calls are mocked using the tools.json declarations. Model inference uses a free tier (limited to 1000 tokens/run, 100 runs/day per author). Useful for rapid iteration before shipping to production.

8. Version Control + Git

Studio works on top of Git. Each agent is a directory in a repo; commits trigger a *dcs build* in CI; merging to main deploys to staging. The full DCS Platform Build Pipeline (Chapter 4 of the Platform whitepaper) runs on every push.

9. Templates + Starters

Studio ships with 24 starter templates for common patterns:

- refund-handler · ticket-router · support-triage · meeting-scheduler · invoice-extractor
- lead-qualifier · content-summarizer · code-reviewer · pr-describer · changelog-writer
- research-assistant · doc-translator · email-drafter · email-classifier · contract-redliner
- data-extractor · csv-cleaner · sql-writer · dashboard-narrator · chart-explainer
- monitoring-alert-triage · log-investigator · incident-summarizer · runbook-executor

10. Performance

Operation	p50	p99	Notes
Editor key-to-paint latency	8 ms	24 ms	Monaco; local
Autocomplete suggestion	40 ms	120 ms	Static + LLM-augmented
Hover prompt preview	14 ms	38 ms	Cached after first hover
Test runner (5 tests)	4 s	12 s	Includes LLM judge calls
Live preview (1 sandbox run)	1.4 s	4 s	Mocked tools; throttled model
Visual → code compile	180 ms	420 ms	Graph to agent.yaml
Build + deploy to staging	8 s	24 s	Full DCS build pipeline

11. Comparison vs. Alternatives

	Cursor	Anthropic CUE	Make / n8n	DCS Agent Studio
Agent-specific lints	✗	~	✗	✓
Visual builder	✗	✗	✓	✓
LLM collaborator	✓	~	✗	✓
Non-deterministic test runner	✗	✗	✗	✓
Signed-artifact output	✗	✗	✗	✓
Live sandboxed preview	~	~	✓	✓
Git-native workflow	✓	✗	✗	✓

12. Implementation Guide

12.1 Start from scratch

```
$ dcs studio new my-first-agent  
Opening agent-studio at studio.dcsai.ai/my-first-agent  
Starter templates available in the right panel.  
Or ask the collaborator: "Write me a refund agent for Stripe."
```

13. References

- [1] Microsoft. **Monaco Editor**. (Embedded code editor)
- [2] Anthropic. **Computer Use Examples (CUE)**. 2024.
- [3] GitHub. **Copilot Workspace**. (LLM collaborator pattern)
- [4] Cursor. **The AI-first code editor**. 2024.
- [5] Make (Integromat). **Visual workflow automation**. (Visual builder pattern)

Published under CC BY 4.0. Agent Studio is in private beta; request access at studio@dcsai.ai.