MCP SERVER | NO CODE | CLOUD HOSTED

Dagger (Programmable CI) MCP for AI Agents

Orchestrate complex software delivery pipelines and manage container builds

Dagger (Programmable CI) lets your AI agent manage complex software delivery pipelines directly. Use this MCP to orchestrate entire build processes, pull images, execute raw GraphQL queries for testing logic, and securely handle secrets—all through natural language commands.

A+ Quality Score 100/100

ci-cd | container-orchestration | pipeline-automation | graphql | infrastructure-as-code | build-automation

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/dagger-programmable-ci

SCAN TO VIEW

The connectivity layer between AI and the world's software.

Vinkius sits between AI and every application. All communication passes through Vinkius Cloud via the Model Context Protocol (MCP) — with governance, observability, and security at every layer.

Your AI Connections Run Through Vinkius Cloud

The world's largest managed MCP catalog

Vinkius is the connectivity layer where AI connects to the software your business already runs. We handle the hosting, the security, the credentials, the uptime — you get agents that actually do things.

We operate the world's largest managed MCP catalog. Major SaaS platforms, CRMs, databases, and cloud providers — running, monitored, production-ready. This MCP server is hosted and maintained by the Vinkius Cloud for AI Agents.

> *The agent doesn't manage credentials, doesn't manage uptime, doesn't manage security. Vinkius does.*
>
> — Architecture principle

Four Pillars of the Vinkius Runtime

01 — Security by design

Credentials stay encrypted at rest via AES-256. The AI agent never touches raw keys — they're injected into a sandboxed V8 isolate at runtime. Actions are logged, and connections have an emergency kill switch.

02 — Built on MCP Fusion

This MCP server was built with **MCP Fusion**, the open-source framework (Apache 2.0) that powers the entire Vinkius catalog. Schema-as-firewall strips undeclared fields, compiled PII redaction runs at zero overhead, and cryptographic lockfiles produce git-diffable audit trails.

03 — Deterministic observability

Eight immutable metrics per endpoint: request volume, p95 latency, error rate, active connections, cost attribution. A live payload feed logs every tool call with mutation detection.

04 — Autonomous operations

Servers are deployed, monitored, and patched autonomously. New capabilities and security patches ship weekly. Zero-downtime deployments ensure continuous availability across all managed MCP servers.

AES-256	Ed25519	24h TTL	V8 Isolate
Encryption at rest	PKI vault signatures	Ephemeral session keys	Sandboxed execution

One Token. Instant Access.

Every MCP server on Vinkius is accessed through a **Connection Token**. Tokens are generated in the cloud dashboard and produce a unique MCP endpoint URL. Paste this URL into any MCP-compatible client — no SDK required.

A single token can serve **multiple AI clients simultaneously**, or you can issue separate tokens per client for granular access control. Each token tracks its own request count, last activity timestamp, and can be individually enabled or revoked.

MCP ENDPOINT

```
https://edge.vinkius.com/{token}/mcp
```

Claude · Cursor · VS Code · Windsurf · Grok · Gemini

Security Is the Architecture

Security in Vinkius is not a feature — it's the foundation of the runtime. The gateway enforces multiple independent protection layers between AI agents and third-party APIs.

01 — Ed25519 PKI Vault

Every workspace has an Ed25519 Master Key. Session keys are generated ephemerally (24h TTL) and signed by the Master Key. Credentials never leave the vault boundary.

02 — V8 Isolate Sandboxing

Tool code runs inside isolated-vm V8 isolates with 64 MB memory caps and per-request timeouts. No filesystem access, no network access except through the SSRF-guarded fetch bridge.

03 — SSRF Guard

All outbound HTTP requests are DNS-resolved and validated before execution. Private IP ranges (10.x, 172.16-31.x, 192.168.x, AWS metadata 169.254.x) are blocked at the network layer.

04 — DLP & PII Redaction

A ResponseGuard pipeline intercepts every tool response. Configurable redaction patterns strip sensitive fields (emails, SSNs, card numbers) before data reaches the AI agent.

05 — Cryptographic Audit Trail

Every request is signed into a SHA-256 hash chain with Ed25519 signatures. Events form a tamper-proof, SIEM-exportable forensic record.

06 — Honeytoken Trap System

Phantom credentials are injected into isolated environments. If a honeytoken is used outside Vinkius infrastructure, the server is quarantined instantly.

Emergency Kill Switch

EU AI Act Art. 14(1) Compliant

The kill switch is an **emergency halt** mechanism — not a simple toggle. When triggered, it executes three actions atomically:

01 — Server deactivated

The MCP server is immediately taken offline across the entire cluster.

02 — All tokens revoked

Every connection token is invalidated. Total lockout — reconnection blocked until new tokens are issued.

03 — WebSocket connections killed

Active connections terminated via Redis pubsub broadcast. Propagates to every runtime node in the cluster.

Full Visibility. Zero Guesswork.

The Vinkius cloud dashboard includes a full MCP Governance suite — real-time analytics and security controls for production AI operations.

Control Plane

KPI dashboard with request volume, latency, success rate, token consumption, and AI-generated operational briefings.

Agent Activity

Which AI clients are connecting, how often, and what they're doing — real-time session tracking.

FinOps

Cost tracking per tool, payload compression savings, budget optimization signals, and consumption trends.

Tool Health

Slowest and most error-prone tools, with actionable root-cause insights and performance baselines.

Firewall & DLP

PII redaction activity, sensitive data protection counters, and security event timeline.

Incident Log

Error trends, failure rates, status-code breakdowns, and forensic audit trail access.

> Get started at **cloud.vinkius.com** — connect your AI agent in under 60 seconds.

Dagger (Programmable CI) MCP

10 tools available
Cloud-hosted on Vinkius

This MCP connects your AI client directly to the Dagger Engine, giving you programmable control over your entire CI/CD flow. Instead of jumping between dashboards or writing complex YAML files, your agent handles the orchestration. It can initialize scratch containers, pull necessary images, and query Git repositories for source code. Need to run a specific test? Your agent executes raw GraphQL queries to compose that logic dynamically. The MCP also manages resource lifecycles—it pulls secrets securely and caches volumes so you don't repeat work. When you use Vinkius, your AI client gets access to this entire suite of tools, letting you debug pipelines or run full builds right from the chat interface. It's all about treating your infrastructure like another API endpoint.

Core Capabilities

01 — Execute complex build graphs

Run raw GraphQL queries against the Dagger engine to define and execute directed acyclic graph operations.

02 — Manage container resources

Initialize scratch containers, pull images, and manage OCI-compatible states for your builds.

03 — Query source code repositories

Connect to Git repositories to fetch the latest source code directly into your pipeline environment.

04 — Securely handle configuration data

Create and access secrets using various sources, including environment variables or local file paths.

05 — Inspect build state

Query the current module status or check the engine version to ensure your pipeline environment is consistent.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/dagger-programmable-ci** — connect your AI agent in three steps.

01 Ensure a Dagger Engine is active in your local development environment.

02 Provide the session port and token generated by the Dagger CLI within your AI client's connection parameters.

03 Use natural language commands to instruct your agent to build, test, or deploy resources.

The bottom line is that you interact with infrastructure tasks conversationally; the MCP translates those instructions into a structured series of engine operations.

Built For

This MCP targets experienced technical roles—DevOps Engineers, SREs, and Software Developers. If your job involves managing complex deployments or debugging build failures across multiple services, this is for you. It's built for people who are tired of context-switching between terminals, dashboards, and code editors.

DevOps Engineer

Automates the debugging and execution of complex deployment pipelines without ever leaving their chat or coding interface.

Site Reliability Engineer (SRE)

Inspects engine states, manages persistent cache volumes, and orchestrates infrastructure tasks using programmable CI logic.

Software Developer

Runs full builds, tests, and container operations directly from their code editor to validate local changes before committing.

What Changes When You Connect

01 Stop context switching. Your agent runs full build, test, and deployment cycles without you ever leaving the chat interface.

02 Control infrastructure logic directly. Use `execute_graphql_query` to define deep, dynamic operational graphs that standard CI tools can't handle.

03 Manage dependencies cleanly. The MCP handles secrets using `query_secret`, ensuring sensitive data is accessed securely during every build step.

04 Validate environment consistency. Check the module state or run `query_version` to guarantee your pipeline runs against expected engine parameters.

05 Source code access is instant. Use `query_git` and `query_directory` together to pull specific source versions into a fresh, isolated container.

Real-World Applications

Diagnosing a failing microservice build

A developer notices a service failed deployment. They ask their agent to check the host environment (`query_host`), initialize a scratch container (`query_container`), and then execute a specific GraphQL query to pinpoint which dependency failed, getting a clean report instantly.

Building an isolated test environment

An SRE needs to test code against a private resource. They ask the agent to pull required assets using `query_http`, retrieve credentials via `query_secret`, and then use these inputs in a raw GraphQL query for validation.

Updating dependencies across multiple services

A DevOps engineer needs to ensure all services are built from the latest Git commit. They ask the agent to run `query_git` first, then use that source code to define and execute a multi-stage build graph.

Creating reproducible test runs

A developer wants to ensure the local environment matches production. They request the agent confirm the engine version (`query_version`) and query the current module state, ensuring all necessary caches are available via `query_cache_volume`.

Patterns to Avoid

Manual build step failures

✗ AVOID

Trying to manually pull dependencies and execute a series of shell commands in the chat. This approach is brittle, lacks state tracking, and fails if one command exits non-zero.

✓ INSTEAD

Don't rely on sequential text commands. Use `execute_graphql_query` to define the entire build process as a single directed acyclic graph (DAG) operation. The engine handles the dependencies, making the pipeline reliable.

Hardcoding sensitive variables

✗ AVOID

Pasting API keys or database credentials directly into the prompt for testing. This is a massive security risk and leaves no audit trail.

✓ INSTEAD

Always ask your agent to handle secrets using `query_secret`. The MCP securely fetches credentials from defined sources (like environment variables) without exposing them in plain text anywhere.

Ignoring resource scope

✗ AVOID

Attempting to run a build that requires external files but forgetting to provide the file path or URL. The process stops immediately.

✓ INSTEAD

First, use `query_http` if the asset is online, or `query_directory` if it's local. Then include the resulting ID in your GraphQL workflow definition for guaranteed access.

The Right Fit

Use this MCP when your deployment logic needs to be treated as a programmable graph of operations. You need to define dependencies (e.g., 'Stage B cannot start until Stage A completes and produces X artifact'). If you are only running simple, sequential shell scripts or fetching basic data, then the core GraphQL functionality might be overkill; standard scripting tools work fine. However, if your process involves managing containers, querying Git states, or combining multiple resource types (secrets, caches, source code) into one atomic unit, this MCP is essential. Don't use it just because you can. Use it when reliability and dependency management are mission-critical.

Dagger Programmable CI: Automating Complex Build Logic in DevOps

Today, building a software artifact is a mess of context switches. You write a script, run it locally, find a dependency missing, switch to the UI to upload credentials, then go back to your code to fix the pathing. It's slow, error-prone, and involves copy-pasting IDs between four different tools.

With this MCP, you tell your agent what needs to happen—'Build Service X using Git commit Y and run it in a container with secret Z.' The system handles all the steps: pulling code via `query_git`, establishing the environment (`query_container`), fetching secrets (`query_secret`), and orchestrating the entire flow through a single, reliable GraphQL definition. You get full automation without leaving your chat.

Dagger Programmable CI: Improving Container Orchestration in SRE

SREs often spend time verifying that the build environment is consistent—checking if the right base image was pulled, if the cache volume exists, or what version of the engine they are running against. This requires manually checking logs and querying multiple internal services.

Now, you can check resource integrity in a single query. By using `query_version` and `query_cache_volume`, your agent verifies that all prerequisites for deployment are met before a single line of code runs. It’s proactive validation, not reactive debugging.

Dagger (Programmable CI): 10 Tools for Pipeline Orchestration

These tools give your AI agent direct access to the Dagger Engine's core functionalities, allowing it to manage containers, query Git repos, and define complex build graphs.

#	TOOL	DESCRIPTION
01	`execute_graphql_query`	Run a raw GraphQL query against the Dagger engine to define complex operational graphs.
02	`query_cache_volume`	Creates and manages cache volumes for persistent build data.
03	`query_container`	Initializes a scratch container environment and returns its unique state ID.
04	`query_current_module`	Retrieves detailed information about the current module's operational state.
05	`query_directory`	Creates an empty directory within the build context and returns its ID for later use.
06	`query_git`	Connects to a Git repository to query its current state or fetch source code.
07	`query_host`	Retrieves details about the underlying host computing environment.
08	`query_http`	Downloads a file from any specified URL for use in the pipeline.
09	`query_secret`	Creates or references secrets, supporting environment variables, local files, and command outputs.
10	`query_version`	Checks and reports the specific version number of the Dagger Engine currently running.

See It in Action

Real prompts you can use once this MCP is connected to your AI agent through Vinkius Cloud.

U What's the latest state of the core library repo?

Repository Status Report

- **Source:** `https://github.com/dagger/core-lib`
- **Status:** Ready to build.
- **Directory ID:** `dir-repo789` (Use this ID for subsequent operations.)
- **Latest Commit:** v1.5.2 (SHA: a3b4c5d)

Ready to start building from this state.

U Can you set up a temporary build environment and pull the necessary secret?

Got it. I've spun up a scratch container for you (`cont-abc123xyz`).

Next, I'm fetching the required API key:

- **Secret Name:** `API_KEY`
- **Source Type:** Environment Variable (Required)
- **Value Status:** Found and secured. You can now use this ID to chain further operations.

U Run the full test graph for my service, using the latest code

GraphQL Execution Plan

- ✅ **Step 1:** Initialize Container (`cont-abc123xyz`).
- ✅ **Step 2:** Query Codebase (using `dir-repo789`).
- ✅ **Step 3:** Execute Test Logic:

* `query(service: { test })` → Success.
* `query(test: { validateCache })` → Passed.

Overall build status: SUCCESS. The resulting artifact is ready.

Frequently Asked Questions

01 **How does Dagger (Programmable CI) MCP help me run complex deployments?**

This MCP lets you define your entire deployment process as a single, programmable workflow. Instead of writing many small steps, you use raw GraphQL queries to tell the agent exactly how all parts—containers, secrets, and source code—must interact.

02 **Do I need to be a DevOps expert to use Dagger (Programmable CI) MCP?**

No. While it handles complex infrastructure logic, you interact with it using natural language commands via your agent. The MCP translates your conversational requests into the precise technical steps needed for a successful build.

03 **Can Dagger (Programmable CI) MCP handle external files or URLs?**

Yes. It has tools to pull remote assets from URLs (`query_http`) and also manage local directory structures, allowing you to bring any needed file into the build context for testing.

04 **Is Dagger (Programmable CI) MCP better than traditional Jenkins setups?**

It's a modern alternative. While older systems rely on rigid pipelines and configuration files, this MCP allows you to dynamically query and manage resources in real-time through your agent, offering much greater flexibility.

05 **What if my build fails halfway through with Dagger (Programmable CI) MCP?**

The system tracks the full state. You can ask your agent to check the current module status or query the host environment to pinpoint exactly where and why the failure occurred, saving you hours of debugging.

Go Live in 60 Seconds

Get your connection token from **cloud.vinkius.com**, then paste the endpoint URL into any MCP-compatible client.

YOUR MCP ENDPOINT

```
https://edge.vinkius.com/[TOKEN]/mcp
```

CLIENT	WHERE TO CONFIGURE
Claude AI	Profile → Customize → Connectors → "+" → Add custom connector → Paste endpoint
Cursor	Settings → Features → MCP Servers → "+ Add New MCP Server" → Type: SSE → Paste endpoint
VS Code	Ctrl/Cmd+Shift+P → "MCP: Add Server" → add `"dagger-programmable-ci": { "url": "…" }`
Windsurf	MCP Settings → `mcp_settings.json` → Add endpoint URL
ChatGPT	Settings → Tools & plugins → Add MCP server → Paste endpoint
Gemini	Extensions → Add MCP Server → Paste endpoint URL

ASK AN AI ABOUT THIS

Let your preferred AI explain this MCP server

- Ask ChatGPT
- Ask Claude
- Ask Perplexity
- Ask Gemini
- Ask Grok

READY TO CONNECT

Dagger (Programmable CI) is live on Vinkius Cloud.

Get your connection token, paste it into your AI agent, and start building. No SDK. No deployment. Just results.

Start at cloud.vinkius.com →

vinkius.com · support@vinkius.com

INDEPENDENT PLATFORM DISCLAIMER

Vinkius is an independent platform and is not affiliated with, endorsed by, sponsored by, verified by, or otherwise authorized by Dagger (Programmable CI). All third-party trademarks, logos, and brand names are the property of their respective owners. Their use in this document is strictly for informational purposes to identify service compatibility and interoperability.

DOCUMENT INFORMATION

Generated	June 2026
MCP Server	Dagger (Programmable CI) MCP
Server ID	`019e3884-f577-717d-87af-e93ec01431b7`
Platform	Vinkius Cloud for AI Agents
Endpoint	`https://edge.vinkius.com/{token}/mcp`

LICENSE & USAGE

This document is generated automatically by the Vinkius PDF Engine. Content reflects the MCP server configuration at the time of generation and may change as updates are deployed. For the most current information, visit vinkius.com/mcp/dagger-programmable-ci.