MCP SERVER | NO CODE | CLOUD HOSTED

OpenCost (K8s Cost) MCP

Reconcile complex cloud billing data instantly.

OpenCost (K8s Cost) connects your AI client directly to your Kubernetes billing data. Query cost allocations for specific workloads, check costs tied to physical assets like Nodes or Disks, and reconcile complex cloud provider bills (AWS CUR, Azure Export, GCP). It gives you natural language visibility into exactly where your cluster spending goes.

D Quality Score 51.59/100

kubernetes | cost-optimization | cloud-billing | infrastructure-monitoring | resource-allocation | finops

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/opencost-k8s-cost

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.

OpenCost (K8s Cost) MCP

6 tools available
Cloud-hosted on Vinkius

OpenCost lets you talk to your infrastructure budget. Instead of diving through confusing dashboards and merging multiple billing spreadsheets, you just ask your AI agent questions about your Kubernetes costs in plain English. You can find out which specific namespace or controller is chewing up the most resources, or see how much compute power on a physical Node cost this month. It pulls together data from core cloud provider bills—like AWS CUR or Azure exports—and cross-references it with internal workload allocations. This means you don't just get a total number; you get visibility into exactly why that number is what it is, helping your team maintain budget discipline while developing and running services. When you connect this MCP via the Vinkius catalog, your agent gains access to all these cost metrics in one place.

Core Capabilities

01 — Determine workload spending

Query costs based on Kubernetes workloads across clusters, nodes, and namespaces.

02 — Inspect physical resource costs

Retrieve cost data associated with underlying infrastructure assets like disks or load balancers.

03 — Reconcile cloud billing reports

Match internal Kubernetes spending against official bills from major cloud providers (AWS, Azure, GCP).

04 — Track third-party service expenses

Get time-series data and summaries for external tools like Datadog or MongoDB Atlas.

05 — Filter costs by metadata

Aggregate spending by labels, annotations, or defined service levels to pinpoint budget usage.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/opencost-k8s-cost** — connect your AI agent in three steps.

01 Subscribe to this MCP and enter your OpenCost API Base URL.

02 Your AI client connects the cost data source to your agent's context.

03 You ask a natural language question, and the system returns precise spending figures linked to specific resources.

The bottom line is you get an immediate, accurate report on infrastructure costs without ever leaving your chat window or IDE.

Built For

DevOps and FinOps Engineers. You're the one who gets tired of spending weekends clicking through disparate cloud dashboards just to answer 'Where did this feature get so expensive?'

DevOps Engineer

Uses the MCP to quickly identify namespaces or resources that have become unexpectedly costly, preventing budget overruns before they happen.

FinOps Analyst

Connects cloud billing reports with internal K8s usage data to prove cost attribution and reconcile discrepancies between systems.

Engineering Manager

Gets a summary of project-based spending trends, providing quick financial oversight during development cycles without needing deep technical knowledge.

What Changes When You Connect

01 Identify spending culprits: Instead of manually checking ten namespaces, you can ask the agent to check allocation for all clusters and pinpoint exactly which controllers or pods are driving up costs using `get_allocation`.

02 Connect the dots: You don't have to merge separate AWS CUR files with your internal reports. The MCP lets you use `get_cloud_cost` to pull provider data and compare it directly against K8s usage metrics in one query.

03 Track hardware costs: Need to know what underlying infrastructure is costing? Use `get_assets` to retrieve cost data for physical components like Nodes or Disks, giving a full picture of your spend.

04 Monitor external tools: Beyond the cloud provider, you can track third-party services. Use `get_custom_cost_timeseries` and `get_custom_cost_total` to summarize costs from Datadog or MongoDB Atlas over time.

05 Pinpoint leakage: The MCP allows aggregation by labels and annotations. This lets your agent filter out the noise and tell you precisely where a specific project's budget is going.

Real-World Applications

Why did our staging environment suddenly spike in cost?

The Engineering Manager asks the agent to run `get_allocation` for the last quarter. The agent responds, detailing that a new logging controller deployed to the 'staging' namespace accounted for 40% of the unexpected increase, allowing them to immediately scale it back.

We need to compare our internal metrics against the official AWS bill.

The FinOps Analyst uses `get_cloud_cost` to retrieve the latest AWS CUR data. The agent then compares this raw provider data with the resource totals pulled via `get_allocation`, instantly flagging a 15% mismatch in EBS volume costs that needs review.

How much did our development team spend on monitoring tools last month?

The DevOps Engineer prompts the agent to get a total summary using `get_custom_cost_total` for Datadog and MongoDB Atlas. The result provides a clear, single figure showing the full cost of observability across all projects.

I need to know the compute cost for our primary nodes right now.

The user asks about backing assets today. The agent uses `get_assets` and reports that the 12 m5.large cluster nodes are incurring $45.20 in compute costs, including associated EBS volumes.

Patterns to Avoid

Treating billing data like general knowledge

✗ AVOID

Asking the AI agent generally, 'What is our cloud spending?' This results in vague answers because the system doesn't know which bill or resource you mean.

✓ INSTEAD

Be specific and direct. Instead of a broad question, ask: 'Using `get_cloud_cost`, what was the total cost for EBS volumes last month?'. This forces the agent to use the correct tool.

Ignoring workload boundaries

✗ AVOID

Just seeing a massive lump sum of cost and not knowing which team or service caused it.

✓ INSTEAD

Use `get_allocation` combined with label filtering. Ask: 'Show me the allocation for all services labeled 'project-alpha' last week.' This immediately isolates project spending.

Forgetting third-party costs

✗ AVOID

Only looking at AWS/Azure bills and completely missing the cost of observability tools like Datadog.

✓ INSTEAD

Always check external services. Use `get_custom_cost_total` to ensure your summary includes all necessary monitoring and SaaS expenses.

The Right Fit

Use this MCP if your primary job involves reconciling multiple, complex cost inputs: Kubernetes usage data, raw cloud provider bills (AWS CUR, etc.), AND third-party service invoices. You need a single source of truth to answer 'Why is the bill what it is?'

Don't use it if you only need basic billing reports or simple resource counts. If all you need is a list of nodes and their status without cost attached, a general cloud provider API might suffice. Similarly, if your costs are already perfectly siloed by one single system, this MCP offers overkill. But if your money lives in the gap between your K8s manifests and your actual invoices, this is exactly what you need.

The Struggle of Merging Cloud Bills

Today, figuring out cloud spend means a nightmare. You're staring at three separate dashboards: the AWS cost report, the Azure export file, and your internal K8s usage dashboard. Then you have to copy-paste data from one into a spreadsheet, trying to find matching resource IDs and dates so you can even start reconciling which bill paid for what actual workload.

With this MCP, that entire process disappears. You talk to your agent like talking to a teammate, asking it to compare the cost allocation of your 'production' namespace against the official cloud provider billing report. It handles the cross-referencing instantly and gives you the answer—no spreadsheets required.

Get Full Visibility with OpenCost (K8s Cost) MCP

You no longer have to manually query costs for specific assets or run separate reports for every external tool. The agent can consolidate everything: it checks the backing cost data using `get_assets`, pulls billing details via `get_cloud_cost`, and sums up third-party overhead with a single prompt.

What's different is that you get deep, actionable insights without ever leaving your conversational workflow. It turns complex financial auditing into simple conversation.

OpenCost (K8s Cost) with 6 Tools

These tools allow your agent to query specific cost data points, retrieve asset details, pull cloud bills, and summarize third-party expenses based on the OpenCost API.

#	TOOL	DESCRIPTION
01	`get_allocation`	Queries the cost and resource usage allocated to specific Kubernetes workloads.
02	`get_assets`	Retrieves detailed backing cost information for individual infrastructure assets.
03	`get_cloud_cost`	Pulls raw cloud spending data directly from official provider billing reports (AWS, Azure, GCP).
04	`get_custom_cost_timeseries`	Provides time-series samples of costs for third-party services like Datadog or MongoDB Atlas.
05	`get_custom_cost_total`	Generates a total summary calculation for third-party service expenses over a defined period.
06	`set_log_level`	Changes the internal logging verbosity of the OpenCost system at runtime.

See It in Action

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

U Show me the cost allocation for all namespaces over the last 7 days.

I've retrieved the allocation data for the last 7 days. The 'production' namespace accounts for 65% of the spend ($420.50), followed by 'staging' at 20% ($130.00). Would you like to see the breakdown by controller kind?

U What are the backing asset costs for our nodes today?

Querying asset data... For today, your cluster nodes have incurred a cost of $45.20. This includes compute costs for 12 m5.large instances and associated EBS volumes. No load balancer costs were detected in this window.

U Get the total summary for third-party service costs for the current month.

The total custom costs for this month sum up to $1,250.00. The largest contributors are Datadog ($800.00) and MongoDB Atlas ($450.00). Would you like to see the daily timeseries for these services?

Frequently Asked Questions

01 **How does OpenCost (K8s Cost) MCP handle multiple cloud providers?**

The MCP uses the `get_cloud_cost` tool to connect directly to various provider billing reports, including AWS CUR, Azure Export, and GCP Billing data. This means you can reconcile costs across different clouds from one place.

02 **Can OpenCost (K8s Cost) MCP tell me which service is the most expensive?**

Yes. You can use `get_allocation` to query costs and resources allocated to specific namespaces, pods, or controllers to pinpoint your biggest spending areas.

03 **What if I need cost data for services like Datadog?**

The MCP supports this via third-party tools. You use `get_custom_cost_timeseries` and `get_custom_cost_total` to track external service expenses over time or get a summary total.

04 **Is OpenCost (K8s Cost) MCP only for AWS?**

No. It is designed to be cloud-agnostic, pulling data from multiple sources including Azure and GCP in addition to AWS billing reports.

05 **How do I check the cost of a physical node using OpenCost (K8s Cost) MCP?**

Use the `get_assets` tool. This retrieves backing cost data broken down by individual assets, giving you visibility into nodes, disks, and load balancers.

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 `"opencost-k8s-cost": { "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

OpenCost (K8s Cost) 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 OpenCost (K8s Cost). 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	OpenCost (K8s Cost) MCP
Server ID	`019e38cd-b00c-706a-aefe-9f756dec0209`
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/opencost-k8s-cost.

© 2026 Vinkius · vinkius.com · support@vinkius.com