MCP SERVER | NO CODE | CLOUD HOSTED

Topological Sort Engine MCP for AI Agents

Resolving Complex Build Dependencies and Task Scheduling

The Topological Sort Engine handles complex dependency resolution by calculating valid execution orders for Directed Acyclic Graphs (DAGs). It uses industry-standard algorithms—Kahn's and Depth-First Search—to establish precise task sequences, ensuring that no prerequisites are skipped. You can also detect circular dependencies where tasks depend on each other endlessly.

A+ Quality Score 100/100

dag | topological-sort | kahn-algorithm | dfs | cycle-detection | dependency-resolution

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JULY 2026
vinkius.com/mcp/topological-sort-engine

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.

Topological Sort Engine MCP

3 tools available
Cloud-hosted on Vinkius

Managing complex systems often means figuring out the right order to run tasks or build modules. If Task B needs Task A to finish first, you need a guaranteed sequence. This MCP solves dependency mapping for any system built as a DAG. It calculates multiple possible linear execution orders using proven algorithms like Kahn's and Depth-First Search. Need to know if your task list is impossible because two tasks depend on each other in a loop? The engine finds those exact circular dependencies, showing you exactly which nodes are stuck. If connecting this capability feels overwhelming, remember Vinkius hosts thousands of MCPs; you just connect your AI client once and get access to the whole catalog.

This is essential for CI/CD pipelines, complex build systems, or any workflow where prerequisites must be met before proceeding.

Core Capabilities

01 — Generate a linear task sequence using Kahn's Algorithm

The tool calculates the correct execution order by iteratively reducing node in-degrees.

02 — Determine an alternative task sequence via DFS

It generates a valid execution flow by performing a depth-first traversal of the graph nodes.

03 — Isolate all circular dependencies in the DAG

The system pinpoints and lists every node involved in dependency loops, preventing failed builds.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/topological-sort-engine** — connect your AI agent in three steps.

01 Define your graph structure by providing a list of nodes (tasks) and edges (dependencies).

02 Select the desired sorting method: Kahn's Algorithm for an in-degree based sort, or DFS for a depth-first sequence.

03 The MCP returns a guaranteed linear execution order or lists all nodes found within any circular dependency loops.

The bottom line is you feed it your interconnected tasks, and it outputs the precise, valid running order—or tells you exactly what's broken.

Built For

This MCP is for DevOps engineers, build system architects, and workflow automation specialists. If your job involves ensuring that complex systems or microservices deploy in the correct sequence, this tool saves hours of debugging failed pipelines.

DevOps Engineer

Manages CI/CD pipelines, using the MCP to validate deployment steps and guarantee services update in the proper order.

Build System Architect

Designs complex software build systems, using it to map module dependencies and resolve prerequisite chains for massive codebases.

Workflow Automation Specialist

Creates automated business processes that require multiple steps to run sequentially (e.g., data processing pipelines), ensuring no step runs prematurely.

What Changes When You Connect

01 Guarantees correct execution order: Instead of guessing the sequence, use `calculate_kahn_sort` to get a mathematically proven linear flow for your tasks.

02 Catches impossible builds early: Use `identify_cycles` to pinpoint exactly which nodes are stuck in circular dependencies before deploying anything.

03 Offers algorithmic flexibility: You can choose between Kahn's Algorithm or DFS traversal using dedicated tools, depending on the required output structure.

04 Reduces manual validation time: Your agent handles the complex graph traversal logic that used to take hours of manual dependency mapping.

05 Increases pipeline reliability: By resolving prerequisites automatically, you drastically cut down on 'dependency failure' errors in production.

Real-World Applications

A multi-service microservices deployment failed

The DevOps team needs to know the correct update order for 15 services. They ask their agent to use `calculate_kahn_sort` on the service dependency map, immediately receiving a validated sequence that ensures core services update first.

A data pipeline cannot run due to circular dependencies

The data architect runs the graph through the engine. The agent uses `identify_cycles`, revealing that 'Ingestion' depends on 'Validation,' and 'Validation' also depends on 'Ingestion.' The cycle is found, stopping wasted debugging time.

Building a complex software module with prerequisite steps

The developer needs to ensure three modules compile in the right order. They use `calculate_dfs_sort`, which provides an alternative, valid sequence that respects all internal library dependencies.

Patterns to Avoid

Assuming a simple alphabetical sort fixes dependencies

✗ AVOID

Trying to run tasks A, B, C just because they are listed alphabetically, even though the dependency map shows that C must precede A.

✓ INSTEAD

Don't rely on arbitrary order. Use `calculate_kahn_sort` or `calculate_dfs_sort` with your graph data. The MCP respects the actual prerequisite relationships defined in your system.

Ignoring circular references entirely

✗ AVOID

Running a build script and letting it fail hours later because two core libraries depend on each other, causing an infinite loop.

✓ INSTEAD

Always run `identify_cycles` first. This tool gives you immediate feedback, pointing out the exact nodes involved in the dependency loop so you can fix the underlying code.

Using generic scheduling tools for complex graphs

✗ AVOID

Relying on simple queue management that only handles linear steps and cannot account for branching or merging dependencies.

✓ INSTEAD

This MCP is built specifically for DAGs. Use its algorithms to model the graph's true structure, making sure your scheduler understands every prerequisite.

The Right Fit

Use this if you have a clearly defined set of tasks (nodes) and know which tasks absolutely require others to complete first (edges). If your goal is simply to run tasks randomly or in alphabetical order, don't use it. Instead, if your problem involves resource allocation across time (like scheduling meetings), look for calendar-based MCPs. However, if your core problem is the *sequence* of execution based on prerequisites—be it code compiling, data transforming, or services updating—this engine provides the definitive answer. If you are only checking a few nodes manually and don't have a full graph definition, this tool might be overkill; stick to simple scripting instead.

Topological Sort Engine for Dependency Resolution in Build Systems

Right now, when your software build fails, you often face an infuriating manual process. You copy error logs into spreadsheets, tracing which module failed because its prerequisite wasn't ready. You spend hours arguing over dependency arrows and manually re-ordering the compile steps just to find a linear path forward.

With this MCP, you feed the full graph structure to your agent. It handles the messy logic instantly. Whether you use `calculate_kahn_sort` or another method, you get back one clean, validated execution manifest that guarantees every module will build in the correct sequence.

Topological Sort Engine for Task Scheduling and Workflow Orchestration

In workflow automation, manual steps are tedious. You have to check if data ingestion finished before transformation can start, and then verify that validation only runs after both the first two stages complete. This means constant state checking across multiple tools.

Now, your agent treats the entire process like a DAG. It uses specialized sorting functions to map out every possible path forward, giving you one single source of truth for execution order instead of dozens of conditional checks.

Topological Sort Engine: 3 Tools for DAG Dependency Resolution

These tools allow your agent to calculate valid linear execution orders, determine alternative paths, or find circular dependency loops in any graph structure.

#	TOOL	DESCRIPTION
01	`calculate_dfs_sort`	Generates a valid execution order by traversing the graph using Depth-First Search.
02	`identify_cycles`	Lists all nodes that are part of circular dependencies, which blocks sorting.
03	`calculate_kahn_sort`	Generates a linear execution order by following the steps of Kahn's Algorithm.

See It in Action

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

U We have 5 services: Auth, UserProfile, Billing, Reporting, and Gateway. Show the deployment order.

Deployment Order (Kahn's Algorithm):

1. `Auth` (Needs nothing)
2. `UserProfile` (Depends on: `Auth`)
3. `Billing` (Depends on: `Auth`, `UserProfile`)
4. `Gateway` (Depends on: `Billing`)
5. `Reporting` (Depends on: `Gateway`)

Note: This order guarantees all prerequisites are met before starting the task.

U Check if this list of tasks has any dependency loops.

Cycle Detection Results:

🚨 **CYCLE FOUND!** 🚨

Nodes involved in a loop:

- `Task A` → `Task B`
- `Task B` → `Task C`
- `Task C` → `Task A`

The graph cannot be sorted. You must break the dependency between these three nodes.

U What's a valid sequence for Modules X, Y, Z with dependencies X→Y and Z→X?

Valid Execution Orders:

- **DFS Sort Example:** [Z, X, Y]
- **Kahn Sort Example:** [Z, X, Y]

Both sequences are valid. The process must start at the root nodes (like `Z`) and proceed outward.

Frequently Asked Questions

01 **What is the difference between Kahn’s algorithm and DFS-based sorting?**

Kahn's algorithm uses an in-degree reduction approach (BFS), while the DFS-based method explores paths deeply before backtracking to reconstruct the sequence.

02 **How can I find nodes that are causing a circular dependency?**

You can use the `identify_cycles` tool, which specifically traverses the graph to isolate and return only the nodes involved in loops.

03 **What happens if my input graph is not a DAG?**

If a cycle is detected, the sorting tools will return a failure state with an error message describing the nature of the loop.

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 `"topological-sort-engine": { "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

Topological Sort Engine 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 Topological Sort Engine. 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	July 2026
MCP Server	Topological Sort Engine MCP
Server ID	`019f2d2f-3785-723c-8cb2-cc8bc95036d7`
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/topological-sort-engine.

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