MCP SERVER | NO CODE | CLOUD HOSTED

SemVer Version Manager MCP

Keep your dependencies mathematically sound.

SemVer Version Manager gives your AI client absolute authority over software versions. Stop relying on large language models to guess dependency compatibility or sort release tags. This MCP deterministically evaluates semantic version ranges and orders messy arrays of releases, making your CI/CD pipeline mathematically perfect every time.

A+ Quality Score 100/100

semantic-versioning | dependency-management | software-lifecycle | version-control | parsing

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/semver-version-manager

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.

SemVer Version Manager MCP

2 tools available
Cloud-hosted on Vinkius

When working with package files like `package.json`, LLMs frequently mess up the math. They might hallucinate that a version satisfies a peer dependency range (like `>=1.4.0 <2.0.0`), or they'll fail to sort release tags correctly, especially when pre-releases are involved. This MCP bypasses those guesswork moments by connecting directly to the official NPM SemVer engine. It lets your AI client run real version checks—the kind that keep production builds from failing due to faulty logic. Whether you're validating a specific version against a complex range or need to reliably sort dozens of release candidates, this tool handles all the heavy lifting with true determinism. Because Vinkius hosts this MCP, you get access to mathematically sound dependency resolution right where you need it.

Core Capabilities

01 — Check version compatibility

It checks if a given software version satisfies a specific semantic version range (e.g., `^1.4.2`).

02 — Sort release versions

It takes an unordered list of different software tags and sorts them correctly according to semantic rules.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/semver-version-manager** — connect your AI agent in three steps.

01 You tell your agent exactly what you need: either a specific version check against a range, or an unsorted array of release versions.

02 The MCP sends the data to the underlying NPM SemVer engine for calculation and verification.

03 Your client gets back a definitive answer—a clear pass/fail on compatibility, or a perfectly ordered list.

The bottom line is you get reliable version math that doesn't rely on guesswork.

Built For

Software engineers and DevOps specialists who spend time debugging build failures caused by dependency hell. If your pipeline breaks because the AI client misinterpreted a caret constraint, this is for you.

DevOps Engineer

They use it to validate package dependencies before merging code into main branches, preventing unexpected runtime errors due to version mismatch.

Backend Developer

They rely on it when integrating third-party libraries or calculating which versions of a component are compatible with the current build scope.

Release Manager

They use it to organize and verify the correct chronological order of multiple release candidates (including pre-release tags) for documentation purposes.

What Changes When You Connect

01 Eliminate dependency guessing: Instead of hoping the AI client gets version math right, use `check_semver_compatibility` to confirm if a specific package version fits its required range. This stops build failures cold.

02 Perfect release ordering: If you have a mix of versions (like `v2.0.0`, `v2.0.0-rc.1`, and `v1.5.0`), the `sort_semver_list` tool delivers them in the correct, deterministic sequence every time.

03 Absolute determinism: This MCP uses the official NPM SemVer engine, meaning your version logic is based on established standards, not general LLM knowledge. You get reliable results for critical CI/CD steps.

04 Reduced debugging time: By offloading complex version checks, you cut down hours spent manually verifying which libraries are compatible with each other across different environments.

05 Handles pre-releases properly: It correctly handles complicated tags like alpha and beta versions, ensuring that your release notes or build order is perfectly accurate.

Real-World Applications

Validating a new package dependency

A developer needs to confirm if their current app version (`2.4.1`) satisfies the peer dependency range of an old library (`^2.0.0`). They use `check_semver_compatibility` and instantly get confirmation, knowing they won't deploy a broken build.

Preparing release notes for documentation

The Release Manager pulls in 50 version tags from Jira—a mix of major, minor, patch, and beta releases. They use `sort_semver_list` to generate the exact chronological list needed for the official changelog.

Troubleshooting CI/CD build failures

A developer's agent keeps failing because it thinks a version is compatible when it isn't. They use `check_semver_compatibility` to run a deterministic test, proving the dependency failure and fixing the underlying configuration.

Comparing multiple component versions

An agent needs to compare three different library tags (`v1.0.0`, `v2.0.0-beta`, `v1.9.5`) to decide which one to use in a new module. They pass them through `sort_semver_list` for accurate comparison.

Patterns to Avoid

Asking the agent about version ranges

✗ AVOID

Prompting your AI client: 'Is 2.4.1 compatible with ^2.0.0?' and getting a vague, non-deterministic answer that might be right or wrong.

✓ INSTEAD

Use `check_semver_compatibility` to ask the tool directly. This forces the use of the official SemVer engine, giving you a hard pass/fail result instead of educated guesswork.

Manually sorting version arrays

✗ AVOID

Trying to sort a list containing `v1.0.0`, `v2-beta`, and `v1.9.5` by simply alphabetizing the strings, which results in incorrect order.

✓ INSTEAD

Pass that array directly to `sort_semver_list`. It understands semver rules, ensuring you get the correct numerical sequence regardless of pre-release tags.

Assuming LLM knowledge is enough for code

✗ AVOID

Writing a prompt like: 'Update all dependencies to be compatible with 3.x.' and relying on the general model context without specific checks.

✓ INSTEAD

Use `check_semver_compatibility` repeatedly within your agent workflow to validate every single dependency update against its required range before committing code.

The Right Fit

You must use this MCP if version math is mission-critical. If your task involves comparing, sorting, or validating software versions based on semantic rules (like NPM dependencies), you need this tool. Don't use it if you just need to compare two simple numbers; general logic works fine there. However, do not rely on it for determining which *feature* is available in a version—it only handles the mathematical constraints of the version number itself. For example, while `check_semver_compatibility` can tell you that `2.4.1` falls within `^2.0.0`, it won't tell you if the feature set for 2.4.1 is complete.

Dependency versioning is a nightmare to manage.

Today, managing dependencies means constantly copy-pasting versions between build logs and ticket tracking systems. You run through dozens of tags—release candidates, alpha builds, stable patches—just trying to figure out the right order or if an older version violates a minimum requirement. It's tedious, error-prone work that requires deep knowledge of semver rules.

With this MCP, your agent handles all that complexity instantly. You feed it the messy list of versions, and instead of getting vague text output, you get a mathematically guaranteed, perfectly sorted list or a definitive compatibility check result. It takes guesswork out of software deployment.

Version Managers: Deterministic Math for Your Agent

The biggest manual step that vanishes is the comparison itself. You never have to manually verify if `v1.5.0` correctly precedes `v2.0.0-beta.1`, or whether your current build version satisfies a complex range like `>=3.0.0 <4.0.0`. These are specific, repetitive checks that used to require writing custom scripts.

Now, you just ask the MCP. It runs the check using the official NPM engine and gives you an unambiguous answer every time. This isn't just better; it changes how reliable your entire software process is.

SemVer Version Manager: 2 Tools

These tools let your AI client check if a version fits a dependency range or sort messy lists of release tags correctly.

#	TOOL	DESCRIPTION
01	`check_semver_compatibility`	Checks if a given software version fits within a defined semantic version range, like `^1.4.2`.
02	`sort_semver_list`	Takes an array of mixed release tags and correctly sorts them according to official semantic versioning rules.

See It in Action

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

U Verify deterministically if version `2.4.1` satisfies the peer dependency range `^2.0.0`.

✅ **SemVer Result:** Satisfied. `2.4.1` is safely within the `^2.0.0` caret constraints.

U Sort this array of 12 release tags (including `-rc.1` and `-beta`) in descending order.

✅ **Sorted Order:**

1. `v2.0.0`
2. `v2.0.0-rc.1`
3. `v2.0.0-beta.2`

...

U What is the result of applying a 'minor' bump to version `1.4.3-alpha`?

✅ **New Version:** `1.5.0`. The alpha pre-release tag is dropped during a standard minor bump.

Frequently Asked Questions

01 **Does SemVer Version Manager handle pre-release tags?**

Yes, absolutely. It uses the official NPM engine, so it correctly handles complex pre-release tags like `-rc.1` and `alpha`, ensuring proper sorting and compatibility checks.

02 **How do I use SemVer Version Manager to check dependency ranges?**

You call the `check_semver_compatibility` tool. You provide two inputs: the version you want to test, and the required range (like `^2.0.0`). It returns a clear 'Satisfied' or 'Not satisfied' status.

03 **Can SemVer Version Manager sort versions from different years?**

Yes. The `sort_semver_list` tool is designed to handle full semantic versioning logic, ensuring that the order remains correct even when mixing major, minor, and patch releases across large time gaps.

04 **Is SemVer Version Manager only for JavaScript projects?**

No. While it uses NPM's engine as its backend source of truth, the MCP provides deterministic semantic versioning logic that applies to any software project using standard semver guidelines.

05 **What is the difference between this and writing a regex?**

A regex can validate a format (e.g., 'does it look like X.Y.Z'), but `check_semver_compatibility` performs actual mathematical logic on version constraints, which is far more reliable for dependency management.

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 `"semver-version-manager": { "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

SemVer Version Manager 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 SemVer Version Manager. 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	SemVer Version Manager MCP
Server ID	`019e38e9-f20e-70aa-abcd-be58c398735d`
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/semver-version-manager.