MCP SERVER | NO CODE | CLOUD HOSTED

Gerrit MCP

Manage code reviews from conversation.

Gerrit MCP lets you manage every step of your code review process directly through conversation. Query changes by status or owner, track the full history of patch sets, audit approvals, and map out project branches without leaving your agent chat. It gives your AI client complete oversight of your entire Git workflow.

A+ Quality Score 100/100

code-review | git-management | patch-sets | version-control | software-development | pull-requests

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/gerrit

SCAN TO VIEW

The infrastructure that powers AI agents in the real world.

Vinkius connects AI to the world's software through secure, enterprise-grade infrastructure — enabling real-world execution at scale, built on the Model Context Protocol (MCP).

Your AI Connections Run Through Vinkius Cloud

The world's largest managed MCP catalog

Vinkius is the cloud infrastructure where AI agents connect 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.

Gerrit MCP

10 tools available
Cloud-hosted on Vinkius

Need to keep tabs on what's being coded? This MCP connects your Gerrit instance directly to any AI agent, giving you total control over code reviews and repository management through natural conversation. Instead of opening dozens of browser tabs just to check status updates or commit SHAs, you talk to your agent. You can ask for all open changes in a specific project, filtering by owner or status using syntax like 'status:open'. Need to verify who needs to approve something? Ask the agent to list reviewers and check if required labels, like 'Code-Review' or 'Verified', are present. It also helps you map out your repository structure; you can query all projects, list every branch in a project, and even audit user groups to see who has access control. If you use Vinkius, you get this entire suite of developer tools connected once and accessible from any compatible client.

Core Capabilities

01 — Track open changes

Find specific code reviews by status, owner, or project name using advanced search syntax.

02 — Audit version history

Retrieve the complete list of patch sets for a change to track every commit SHA and parent revision in its lifecycle.

03 — Monitor approvals

List who has reviewed a change and verify if necessary approval labels have been applied by team members.

04 — View repository structure

Get metadata for all projects, list branches within a project, or check the latest commit SHAs to understand repo boundaries.

05 — Manage access and users

List associated user emails, audit system groups, or fetch detailed information about authenticated accounts.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/gerrit** — connect your AI agent in three steps.

01 Subscribe to this MCP and provide your Gerrit URL, username, and HTTP password.

02 Connect the credentialed MCP to your preferred AI client (Claude, Cursor, etc.).

03 Use natural conversation to query changes, list branches, or check project details using specific syntax.

The bottom line is you manage complex code review tasks by talking to your agent instead of navigating multiple developer tools.

Built For

This MCP is for the maintainer who gets burned out clicking through dashboards just to verify a merge status. It's also for the engineering manager who needs a quick, comprehensive view of team approval progress without logging into Gerrit.

Software Engineer

Uses this MCP to check open changes and list patch sets directly within their chat or IDE interface when they don't want context switching.

Engineering Manager

Audits code review progress by asking the agent to enumerate reviewers and verify necessary approval labels across multiple pending merges.

DevOps Engineer

Tests project boundaries by listing all projects, retrieving detailed metadata, or verifying group access permissions for CI/CD setup.

What Changes When You Connect

01 You stop switching tabs. Instead of manually navigating to a change ID, you ask the agent to list all patch sets for that change, getting the full history and commit SHAs immediately.

02 Review status checks are instant. You can query changes using `query_changes` with syntax like 'status:open' so your agent only pulls exactly what needs reviewing, filtering out noise.

03 Approval tracking gets clearer. By calling `list_reviewers`, you instantly see who has provided feedback and verify if critical approval labels were added, without manually checking comments.

04 Project structure is visible at a glance. Use `list_projects` to get an overview of your entire repository landscape, or use `list_branches` to map out which branches exist in a given project.

05 Identity management simplifies. You don't need to remember user IDs; simply asking the agent for user information using `get_account` keeps context clean.

Real-World Applications

Checking merge readiness

An engineer needs to know if a change is ready to merge. They ask their agent, 'Who are the reviewers for change #123?' The agent uses `list_reviewers` and reports back on required approval labels, telling them exactly what's missing before they even open Gerrit.

Debugging version history

A maintainer suspects a specific commit SHA was overwritten. They ask the agent to list all patch sets for that change. The agent uses `list_patchsets` and gives them the full, chronological record of revisions, letting them pinpoint the exact moment something broke.

Auditing team access

A DevOps engineer needs to verify if a new contractor has access rights. They ask the agent to list all groups and then query group details using `list_groups`, verifying who controls what projects across the organization.

Finding stale changes

An engineering manager wants to clean up old, unreviewed code. They prompt the agent with 'List my open changes in the core API project.' The agent uses `query_changes` and provides a list of subjects, numbers, and owners that need attention.

Patterns to Avoid

Treating it like a simple search engine

✗ AVOID

Asking the agent to just 'tell me about project X'. The response is usually vague or incomplete because you didn't specify what kind of data (metadata, branches, status) you actually needed.

✓ INSTEAD

Be specific. To check metadata, ask the agent to use `get_project`. If you need branch names, explicitly ask it to run `list_branches` for that project.

Ignoring authentication requirements

✗ AVOID

Assuming the agent can list all users or groups without proper credentials. This results in a generic 'permission denied' error and leaves you stuck.

✓ INSTEAD

Ensure your initial setup includes providing the necessary Gerrit URL, username, and HTTP password so the MCP can run `get_account` and access restricted data.

Mixing up projects and changes

✗ AVOID

Asking to 'list all commits' without specifying a project or change ID. The agent gets confused because commit history is tied to specific objects.

✓ INSTEAD

First, use `list_projects` to find the right repo name. Then, specify that project when querying: 'List open changes in [Project Name]' using `query_changes`.

The Right Fit

Use this MCP if your daily workflow involves frequent status checks on code reviews and patch sets. If you need to know who reviewed a change, what the current branch state is, or why a merge is blocked due to missing approvals, this tool is essential. However, don't use it if all you need is general Git command execution (like pure `git commit` messages). For simple commits or local branching operations that haven't been pushed and reviewed, stick to your local CLI. This MCP excels at the *coordination* layer—the reading, auditing, and status checking of code that has already entered the official review pipeline.

The manual process of reviewing a single change is a nightmare.

Today, if you need to check one person's changes against three different branches and verify two required approvals, you open Gerrit. You find the change number; you switch tabs to see the list of reviewers; you run another query to get the patch set history, and then you have to manually cross-reference all that data—all before you can even start writing code.

With this MCP connected via Vinkius, you just ask your agent. 'What is the status of change #501?' The agent runs `get_change`, aggregates the reviewer list using `list_reviewers`, and summarizes the approval labels in one clean response. You get the answer immediately, without touching a browser tab.

Gerrit MCP Gives You Full Code Review Oversight

Gone are the days of having to manually cross-reference commit SHAs from different patch sets or running through multiple filter queries just to find an open change by owner. The agent handles all that data aggregation for you.

You don't just get a list; you get context. You ask, and it reports back on the project structure, the users involved, and the current state of approvals. It’s not just better—it fundamentally changes how fast your team can actually merge code.

Gerrit: 10 Tools for Code Review Operations

These tools allow you to query specific data points within Gerrit, whether it's listing projects, retrieving user details, or tracking the commit history of a change.

#	TOOL	DESCRIPTION
01	`query_changes`	Find specific code reviews (changes) on Gerrit using advanced filters like status or owner.
02	`get_change`	Retrieves the full, detailed information for a single specified change ID.
03	`list_projects`	Lists every project (repository) available on your Gerrit instance.
04	`get_project`	Gets full, detailed metadata about a specific Gerrit project.
05	`list_branches`	Lists all defined branches within any given Gerrit project.
06	`get_account`	Fetches the profile information for the currently authenticated Gerrit user account.
07	`list_emails`	Lists all email addresses associated with your verified Gerrit account.
08	`list_reviewers`	Retrieves a list of users who have reviewed a specific change.
09	`list_patchsets`	Lists all historical revisions (patch sets) for a change, including commit SHAs and parent commits.
10	`list_groups`	Returns the names, IDs, and ownership information for all defined groups in Gerrit.

See It in Action

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

U List my open changes in project 'core-engine'

Retrieving your changes... I found 2 open changes in 'core-engine': #501 'Fix memory leak' and #502 'Refactor logging'. Would you like to see the current patch set for either of these?

U Who are the reviewers for change #501?

Retrieving reviewers for #501... I found 3 reviewers: Alice (Code-Review: +2), Bob (Code-Review: +1), and Charlie (Verified: +1). The change is currently awaiting more approvals to be merged.

U List all branches in project 'frontend-v2'

Retrieving branches for 'frontend-v2'... I found 3 branches: 'master' (SHA: abc123), 'release-v1.0' (SHA: def456), and 'feature-auth'. Would you like the latest commit message for any of these?

Frequently Asked Questions

01 **How do I check my open Gerrit changes using the query_changes tool?**

You simply ask your agent to run `query_changes` and provide a filter, such as 'status:open' or 'owner:self'. The MCP handles the advanced syntax so you don't have to remember it.

02 **Can I use Gerrit MCP to see all branches in a project?**

Yes. You ask the agent to list all projects first, then specify which one you want details on and run `list_branches` for that project's current state.

03 **What is the difference between get_change and list_patchsets?**

The `get_change` tool gives you the high-level metadata (who, what, when). The `list_patchsets` tool drills down further to give you a full historical record of every commit SHA that contributed to that change.

04 **Is Gerrit MCP useful for auditing group permissions?**

Absolutely. You can use the agent to run `list_groups` and then query specific groups to verify who has control over which parts of your organization's project access.

05 **Does this MCP help with pull request workflows?**

Yes, it manages the core components of PR workflows. You can check approvals (`list_reviewers`) and track the full commit history required to complete a successful merge.

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 `"gerrit": { "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

Gerrit 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 Gerrit. 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	Gerrit MCP
Server ID	`019d75a3-8ee0-70ec-a0e0-538d27e5fef4`
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/gerrit.