MCP SERVER | NO CODE | CLOUD HOSTED

# Recharge MCP

## Manage every step of your recurring revenue lifecycle.

Recharge manages the entire lifecycle of subscription billing directly through your AI agent. You can create customers, update addresses, manage plans, process charges, and adjust subscriptions—all without leaving your chat window. It lets you handle recurring revenue tasks that used to require jumping between a CRM, a payment processor, and an internal dashboard.

F Quality Score 15.83/100

recurring-billing | subscription-management | customer-data | shipping-logistics | ecommerce-automation

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/recharge-alternative

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.

# Recharge MCP

70 tools available
Cloud-hosted on Vinkius

Think about the mess of managing recurring payments. You're constantly juggling customer records, tracking delivery schedules, making sure addresses are current, or adjusting charges for specific issues. This MCP lets your AI agent handle those operational headaches through simple conversation.

Need to update a billing address or cancel a subscription because a client moved? Just ask. It pulls the necessary data and executes the change. You can list all customers, check credit balances, and even manually process queued charges if needed. For operations teams that deal with high volumes of recurring revenue, this is huge. Because it's hosted on Vinkius, you connect once from your preferred AI client, and suddenly, complex billing tasks become natural conversations.

This tool gives your agent the power to create new users, generate checkouts, manage payment methods, or even delay a prepaid order—all while maintaining accurate records. It’s about giving your team immediate operational control over every aspect of their subscription base.

## Core Capabilities

**01 — Manage Customer Records**

Create new customer profiles and retrieve detailed information, including payment methods and credit balances.

**02 — Control Subscriptions**

Cancel, reactivate, update, or delete entire subscriptions to manage the client lifecycle.

**03 — Handle Payments and Charges**

Process charges manually, refund payments, apply discounts, and skip future billing dates for specific customers.

**04 — Update Logistics Data**

Create or update shipping addresses, merge duplicate records, and manage delivery schedules.

**05 — Build Core Products**

Programmatically create new products, plans, and subscriptions needed for your e-commerce catalog.

## One Click on Vinkius — From Prompt to Execution

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

01 Subscribe to this MCP and provide your Recharge API Access Token.

02 Your AI agent connects the credentials and verifies access to all billing tools.

03 You simply ask your agent to perform a task, like 'Update customer 1234's address' or 'Cancel subscription ID 900.'

04 The agent executes the command using the necessary tool and reports back the success or failure status.

The bottom line is that your AI client translates complex, multi-step billing tasks into single conversational commands.

## Built For

Billing Managers who hate running manual reports. Support Agents overwhelmed by repetitive account updates. E-commerce Ops leads constantly tracking inventory and payments across different systems.

**Support Agent**

Fixing customer accounts in real time; for example, immediately changing a shipping address or skipping the next month's charge during a live chat.

**Billing Manager**

Running audits and reports on subscription health, checking credit balances, or listing all customers that need plan updates.

**E-commerce Operations Lead**

Handling product catalog management—like creating a new product or setting up a bulk discount for multiple plans before launch.

## What Changes When You Connect

01 Stop manually tracking delivery dates. You can use `get_customer_delivery_schedule` to instantly see when a customer expects their next shipment, helping support agents resolve issues before they escalate.

02 Handle billing adjustments on the fly. If a client needs a pause or a discount, you don't need to open another tab; your agent can execute `skip_charge` or `apply_charge_discount` immediately via conversation.

03 Keep customer data clean and accurate. Use `merge_addresses` when dealing with old records that have duplicate addresses, ensuring future billing cycles use the correct location.

04 Accelerate onboarding for new users. Your agent can now fully handle the setup process by running `create_customer`, then immediately following up with `create_payment_method` and finally `create_subscription` in a single flow.

05 Gain full control over your catalog. Need to launch a sale? Instead of manually updating every product, you can run `bulk_manage_product_plans` to adjust pricing across dozens of items simultaneously.

## Real-World Applications

**A customer calls about an incorrect charge.**

The support agent doesn't know if the charge was valid. They ask their agent to `get_charge` details and then use `get_customer_credit_summary`. If the charge was wrong, they can immediately run `refund_charge` and explain the exact numbers to the client.

**The company is expanding to a new region.**

The operations lead needs to update 50 old customer records with new tax addresses. They use `list_customers` to pull the IDs, then call `update_customer` and `change_subscription_address` iteratively until all records are current.

**A key product plan is expiring or needs a price change.**

Instead of updating documents and hoping people read them, the billing manager uses their agent to run `update_plan` for the affected products. This ensures all future subscription setups use the correct, current pricing.

**A large batch of prepaid orders needs to be processed.**

The fulfillment team doesn't want 50 separate API calls. They use `create_async_batch` and then later trigger `process_async_batch`, letting the system handle all the heavy lifting in the background.

## Patterns to Avoid

### Trying to fix billing errors manually.

✗ AVOID

The agent sees a customer's address is wrong and tries to copy-paste it into an internal spreadsheet, hoping to remember which fields need updating for the subscription record later.

✓ INSTEAD

Use `update_address` first to ensure the data exists in your system. Then call `update_customer` or `change_subscription_address` to link that correct address to the billing profile.

### Ignoring necessary cleanup steps.

✗ AVOID

The team deletes a customer record without first checking if there are active subscriptions linked to them, causing the deletion to fail mid-process and leaving data in an inconsistent state.

✓ INSTEAD

Always verify dependencies. Use `get_customer` or `list_subscriptions` before attempting massive changes like calling `delete_customer`. It ensures you know what you're breaking.

### Assuming a discount is automatically applied.

✗ AVOID

The sales rep verbally promises a 20% off deal and tells the agent to 'apply it,' but nothing happens because they didn't specify *when* or *where* the discount should hit.

✓ INSTEAD

Be specific. If you want to apply a one-time coupon, call `apply_charge_discount` when processing the charge. If it’s permanent, use `update_plan`.

## The Right Fit

Use this MCP if your core job revolves around managing customer billing cycles, recurring revenue, and physical logistics data. You need to perform CRUD operations on customers, subscriptions, addresses, and payments; think of it as having a full-stack Billing Admin GUI inside your chat window.

Don't use this if you only need simple read-only reporting (e.g., 'Just give me the list of all customer names'). For that, simply listing records is enough. Don't use it if your process involves complex inventory forecasting or deep financial accounting analysis; those require specialized BI tools. If you just need to know *if* a charge happened, `list_charges` works. But if you need to *change* the state of billing—like updating the next charge date with `set_subscription_next_charge_date` or refunding money via `refund_charge`—you need this full capability set.

## The Billing Data Nightmare

Right now, managing a single client's account means jumping between three different tabs. You check the CRM for their name and email; you switch to the billing platform to see if they have credit left or what their next charge date is; then you open the shipping portal just to confirm the address hasn't changed since last week.

It's a manual relay race of copy-pasting IDs, navigating permission walls, and praying you don't miss an update. The process is slow, prone to human error, and takes up valuable time that could be spent talking to customers.

## Recharge MCP: Full Billing Control

With this MCP, those manual steps vanish. When a customer calls, you ask your agent to `get_customer` data and simultaneously run `get_customer_delivery_schedule`. You don't switch tabs; the AI gives you all three pieces of information back in one response.

You control the entire lifecycle—from creating the initial checkout via `create_checkout` to updating the payment method with `update_payment_method`, and finally processing the charge. It's a single, fluid conversation that executes complex backend logic.

# Recharge Alternative: 50+ Subscription Tools

These tools allow you to read, write, update, and delete every core piece of data related to billing, customers, products, and addresses.

| # | TOOL | DESCRIPTION |
|---|---|---|
| 01 | activate_subscription | Restarts a subscription that was previously cancelled. |
| 02 | add_async_batch_tasks | Adds multiple tasks to a scheduled batch of operations. |
| 03 | apply_charge_discount | Reduces the cost of a charge that is waiting to be processed. |
| 04 | bulk_manage_product_plans | Makes changes to product plans across many items at once. |
| 05 | cancel_subscription | Stops a subscription from renewing or billing in the future. |
| 06 | change_subscription_address | Updates the physical delivery address associated with an active subscription. |
| 07 | clone_order | Creates a copy of an existing prepaid order for record-keeping or manual fulfillment. |
| 08 | create_address | Generates and saves a brand new physical address record. |
| 09 | create_async_batch | Starts a large, scheduled batch of background billing tasks. |
| 10 | create_checkout | Generates the initial necessary checkout structure for a new purchase. |
| 11 | create_customer | Registers a brand-new user account in the system. |
| 12 | create_metafield | Adds custom, non-billing data points to an existing record. |
| 13 | create_payment_method | Saves a customer's credit card or payment details for future use. |
| 14 | create_plan | Defines the terms, price, and duration of a repeatable service plan. |
| 15 | create_product | Adds an item that can be sold through your e-commerce store. |
| 16 | create_subscription | Sets up a new recurring billing arrangement for a customer. |

| # | TOOL | DESCRIPTION |
|---|---|---|
| 17 | create_webhook | Sets up an automated alert that sends data to another service when an event happens. |
| 18 | delay_order | Postpones a prepaid order by one billing cycle interval. |
| 19 | delete_address | Removes an address record, provided no active subscriptions use it. |
| 20 | delete_customer | Permanently deletes a customer and all associated records. |
| 21 | delete_metafield | Removes custom data points from a record. |
| 22 | delete_order | Eliminates a scheduled order before it can be fulfilled or charged. |
| 23 | delete_payment_method | Removes saved payment details from a customer's profile. |
| 24 | delete_plan | Deletes the definition of a billing plan. |
| 25 | delete_product | Removes an item listing from your store's catalog. |
| 26 | delete_subscription | Permanently removes a specific active subscription record. |
| 27 | delete_webhook | Deletes an automated alert setup. |
| 28 | get_address | Retrieves the full details of a specific address. |
| 29 | get_async_batch | Retrieves the status and contents of a scheduled billing batch. |
| 30 | get_charge | Fetches all details about a specific charge attempt or transaction. |
| 31 | get_checkout_shipping_rates | Checks and retrieves the available shipping costs for an order checkout. |
| 32 | get_checkout | Retrieves all details about a specific checkout session. |
| 33 | get_customer_credit_summary | Calculates and returns the current monetary balance remaining on a customer account. |
| 34 | get_customer_delivery_schedule | Predicts all upcoming delivery dates for a given customer address. |
| 35 | get_customer | Retrieves the complete profile and history of a specific user. |
| 36 | get_metafield | Fetches custom data points attached to any record. |
| 37 | get_order | Retrieves the full details of a historical or scheduled order. |

| # | TOOL | DESCRIPTION |
|---|---|---|
| 38 | get_payment_method | Fetches saved payment method information for a customer. |
| 39 | get_plan | Retrieves the details of a specific billing plan definition. |
| 40 | get_product | Fetches all available data about a single product listing. |
| 41 | get_subscription | Retrieves the full status and history of an active subscription. |
| 42 | get_webhook | Fetches details about a specific automated webhook alert. |
| 43 | list_addresses | Provides a list of all saved addresses in the system. |
| 44 | list_charges | Lists multiple transaction records and charge attempts over time. |
| 45 | list_customers | Generates a list of all customer accounts in the database. |
| 46 | list_metafields | Lists all custom data fields available for use across records. |
| 47 | list_orders | Generates a comprehensive list of historical and pending orders. |
| 48 | list_plans | Lists all defined billing plans available for sale or use. |
| 49 | list_subscriptions | Generates a list of all active and past subscriptions. |
| 50 | merge_addresses | Combines multiple source addresses into one clean target record. |
| 51 | process_async_batch | Forces the execution of a previously scheduled billing batch of tasks. |
| 52 | process_charge | Manually runs a charge that was queued but hasn't processed yet. |
| 53 | process_checkout | Completes the payment and order process for a given checkout session. |
| 54 | refund_charge | Reverses a previously successful charge transaction, returning funds to the customer. |
| 55 | remove_charge_discount | Takes an existing discount off a queued charge, reversing that promotion. |
| 56 | set_subscription_next_charge_date | Adjusts the exact date when a subscription will attempt its next billing cycle. |

| # | TOOL | DESCRIPTION |
|---|---|---|
| 57 | skip_address_charges | Prevents future charges from being applied to a specific address for certain subscriptions. |
| 58 | skip_charge | Temporarily blocks a single, upcoming charge attempt for any reason. |
| 59 | test_webhook | Runs a test payload through an existing webhook alert to ensure it works. |
| 60 | unskip_charge | Re-enables a charge that was previously skipped or blocked. |
| 61 | update_address | Modifies the details of an existing address record. |
| 62 | update_checkout | Makes changes to a checkout session before it is finalized. |
| 63 | update_customer | Modifies any core piece of data on a customer's profile, like their name or email. |
| 64 | update_metafield | Changes the value of custom data points attached to a record. |
| 65 | update_order | Modifies details on an existing order, like shipping instructions or notes. |
| 66 | update_payment_method | Updates saved payment information for a customer. |
| 67 | update_plan | Changes the pricing or terms of an existing billing plan. |
| 68 | update_product | Modifies details, descriptions, or inventory levels for a product listing. |
| 69 | update_subscription | Changes the terms, price, or status of an active subscription plan. |
| 70 | update_webhook | Modifies the destination URL or trigger criteria for an automated alert. |

# See It in Action

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

**U** List the most recent 10 customers from Recharge.

I've retrieved the latest customers. The list includes 'Jane Doe' (ID: 88231), 'John Smith' (ID: 88232), and 8 others. Would you like to see the details for a specific customer?

**U** Show me the delivery schedule for customer 123456.

Fetching the schedule for customer 123456... They have 3 upcoming deliveries: June 15th (Monthly Coffee), July 15th (Monthly Coffee), and August 15th (Monthly Coffee).

**U** Get the details for subscription ID 998877.

Inspecting subscription 998877... It is an active 'Premium Skincare Bundle' priced at $45.00, renewing every 30 days. The next charge date is set for June 20th.

# Frequently Asked Questions

01 **How do I check a customer's current credit balance using Recharge MCP?**

You use `get_customer_credit_summary`. This tool calculates and returns the exact monetary balance available on that customer's account, so you know if they can afford a new service.

02 **What is the best way to update an address for multiple users?**

You should first use `list_customers` to get all IDs, then run `update_customer` or `change_subscription_address` in a loop. This ensures every relevant record gets updated consistently.

03 **Can I pause a subscription using Recharge MCP?**

You can manage this by calling `skip_charge`. If you want to stop it permanently, use the `cancel_subscription` tool instead. The choice depends on whether you plan to reactivate later.

04 **How do I manually process a payment that failed?**

Use `process_charge`. This function allows your agent to force-run a charge that was queued up but couldn't process automatically. Always verify the original transaction details first using `get_charge`.

05 **I need to delete an old customer record; is it safe?**

You can use `delete_customer`, but be careful because this permanently removes all child resources associated with that account. Always confirm the ID before running any deletion command.

# 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 `"recharge-alternative": { "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

# Recharge 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 Recharge. 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 | Recharge MCP |
| Server ID | `019e38e1-4452-7300-bd0d-cc75e5bba044` |
| 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/recharge-alternative.