MCP SERVER | NO CODE | CLOUD HOSTED

ThirdWeb MCP

Manage identity and execute complex blockchain actions.

ThirdWeb MCP lets your AI agent manage user identities and interact with smart contracts across multiple blockchains. Handle everything from OAuth sign-ins and passkey verification to reading contract data or executing complex token swaps, all through natural language prompts.

A+ Quality Score 98.33/100

web3 | smart-contracts | wallet-management | authentication | ethereum

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

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.

ThirdWeb MCP

29 tools available
Cloud-hosted on Vinkius

You can build Web3 features directly into your applications without writing a single blockchain interaction line. This MCP connects your agent to identity services and smart contracts across chains like Solana and Ethereum. Need to know if a user exists? You can use the `get_user_details` tool to search for profiles by email or address. When a new user signs up, you can start that process using `initiate_auth`, which supports social logins, email, and passkeys. Once authenticated, your agent can manage the wallet lifecycle—for example, creating a fresh wallet with `pregenerate_wallet`. Beyond identity, this MCP lets your agent interact with contracts. You can execute complex read-only calls to check data using `read_contract`, or you can trigger state changes, like sending tokens or executing swaps across chains. By connecting via Vinkius, you give your agent access to a full range of blockchain and identity operations, letting you focus on the application logic instead of the underlying protocols.

Core Capabilities

01 — Manage User Authentication

Your agent can start user sign-ins using email, social media OAuth flows, or physical passkeys.

02 — Query On-Chain Data

Fetch historical contract events and read specific data points from any supported blockchain through multicalls.

03 — Execute Transactions

Initiate state changes like deploying contracts, sending tokens, or executing token swaps across mainnets.

04 — Handle Wallet Lifecycles

Your agent can create new wallets before a user signs up and retrieve details for authenticated users.

05 — Track Transaction Status

Check the status of any broadcasted transaction to ensure it completed successfully on-chain.

One Click on Vinkius — From Prompt to Execution

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

01 First, subscribe to this MCP and provide your specific API keys and secrets.

02 Next, link your agent to the Vinkius catalog. This gives your AI client access to all available Web3 tools.

03 Finally, ask your agent to perform a task—like 'check the balance of wallet X' or 'authenticate user Y'. The MCP translates that request into blockchain calls.

The bottom line is: you use natural language to trigger complex, multi-step actions across multiple blockchains without writing low-level code.

Built For

This MCP is essential for Web3 developers building decentralized applications (DApps), product managers needing live on-chain data feeds, and backend engineers who manage complex user identity flows across multiple chains.

Web3 Developer

Writing the core logic for a DApp, they use this MCP to simulate contract interactions or handle advanced authentication flows (like `social_auth`) directly from their development environment.

Product Manager

Needs to prove platform activity. They ask the agent to run queries like `query_events` to analyze historical user behavior and contract usage.

DevOps Engineer

Responsible for maintaining cross-chain functionality, they use tools like `list_solana_wallets` and `broadcast_solana_transaction` to automate asset movement checks.

What Changes When You Connect

01 Identity management becomes conversational. You use `initiate_auth` to start a sign-up, and your agent handles the multi-step process (email code, social OAuth) without you needing separate authentication endpoints.

02 Data querying is instant. Instead of building complex RPC calls for every piece of info, simply ask your agent to read contract data using `read_contract` or query history with `query_events`. The results come back immediately.

03 Wallet setup and management are streamlined. You can pre-register users with `pregenerate_wallet`, so the wallet exists before they ever hit 'sign up.'

04 Cross-chain operations simplify. Need to move assets? Your agent handles calculating quotes via `get_solana_swap_quote` and executing trades using `execute_solana_swap` across different chains.

05 Your code stays clean. By routing complex logic—like a full token transfer, requiring signing with `sign_solana_transaction` and then calling `send_solana_tokens`—through the MCP, your application remains simple.

Real-World Applications

Onboarding a new user who only has an email.

A developer asks the agent to onboard a user. The agent first uses `initiate_auth` via email, prompts for the code, and then completes the login using `complete_auth`. This handles the entire secure identity flow in three steps.

Checking if a token transfer was successful.

A product manager needs to validate an asset movement. They ask the agent to get the transaction status using `get_transaction_status` and then confirm the balance change with `get_solana_balance`. This confirms both execution and outcome.

Building a smart contract audit tool.

An engineer wants to see exactly what happened last week. They ask the agent to query historical data using `query_events` or get detailed transaction logs with `query_transactions`, providing them with full context for debugging.

Preparing a user profile before signup.

A backend service needs to guarantee that every new user has an address. Instead of waiting for the sign-up, it proactively calls `pregenerate_wallet` and then uses `get_user_details` to confirm the structure is ready.

Patterns to Avoid

Mixing up read and write operations.

✗ AVOID

The developer tries to check a contract balance and also update it using two different, unrelated functions in one go. This often leads to race conditions or failed state changes because the logic is too broad.

✓ INSTEAD

For viewing data, always use `read_contract`. If you must change the data, group those specific modifications into a single atomic write call using `write_contract` for reliability.

Assuming user identity based on IP address.

✗ AVOID

The app tries to link an account without first verifying who they are. This is insecure and fails when the user switches devices or uses a VPN, leading to data mismatch errors.

✓ INSTEAD

Always start by running `initiate_auth` using secure methods like social providers (`social_auth`) or passkeys before attempting any profile linkage with `link_profile`.

Manually broadcasting raw, unverified transactions.

✗ AVOID

The engineer attempts to send funds by manually encoding and sending a transaction without first checking the required payment details or calculating quotes. The transaction often fails due to insufficient gas or bad formatting.

✓ INSTEAD

First, check the costs using `get_payment_requirements`, get a quote with `get_solana_swap_quote`, then sign it with `sign_solana_transaction` before finally executing with `send_raw_transactions`.

The Right Fit

Use this MCP if your primary pain point is managing the complexity of identity and asset movement across multiple, disparate blockchain environments. If you need to handle user sign-up flows—whether that's standard OAuth or secure passkey verification—this is where you start. You should use it whenever your application

needs to read data from a contract (`read_contract`) *or* modify the state of an asset (`write_contract`). However, don't use this if all you need is simple HTTP API access; those kinds of endpoints are better handled by standard REST connectors. Also, do not rely on it for internal database CRUD operations; this MCP only handles external blockchain and identity actions. If your goal is just listing user names without checking their wallets or contracts, a simpler directory service tool will suffice.

Managing Web3 User Identity Is a Nightmare

Right now, onboarding a new user means building multiple authentication paths: one for Google OAuth, another for email/password with SMS verification, and yet another dedicated flow just for passkeys. You have to manage the state transitions—from 'pending' to 'verified' to 'linked'—across disparate services, leading to bloated code and poor UX.

With this MCP, you simply ask your agent to authenticate a user. It handles the entire chain of events: it triggers `initiate_auth` for email, waits for verification, and then allows you to link multiple accounts using `link_profile` . You get a fully managed identity lifecycle exposed as one simple command.

Execute Contracts with Ease Using ThirdWeb MCP

The old way required writing specific RPC calls for every single contract interaction: one chunk of code to check the total supply, another set to read a specific user's balance, and yet another complex function just to execute the transfer. This made debugging a multi-chain asset movement painful.

Now you can let your agent run `read_contract` or trigger a transaction with `write_contract` . You talk to it like talking to a database—you describe the action, and the MCP handles all the complex serialization, gas estimation, and chain logic behind the scenes. It just works.

ThirdWeb: 29 Tools for Blockchain Ops

These tools let you interact with every aspect of Web3 development—from managing user logins to executing complex token swaps on Solana and Ethereum.

#	TOOL	DESCRIPTION
01	broadcast_solana_transaction	Sends a transaction signal to the Solana blockchain.
02	complete_auth	Validates a code or challenge and finishes a user login process.
03	create_solana_wallet	Generates a brand new digital wallet address on Solana.
04	deploy_contract	Puts a smart contract onto the blockchain using bytecode and ABI definitions.
05	execute_solana_swap	Trades one digital token for another on the Solana main network.
06	fetch_with_payment	Runs a request against an external API that requires payment credentials.
07	get_payment_requirements	Determines what is needed to pay for access to a specific resource.
08	get_solana_balance	Checks the current balance of any specified Solana wallet.
09	get_solana_swap_quote	Calculates how much one token will convert to another on Solana before a swap happens.
10	get_transaction_status	Retrieves the current status (success, failed, pending) of a broadcasted transaction.
11	get_user_details	Searches and retrieves existing user profiles based on criteria like address or email.
12	get_wallet_me	Pulls the specific details of the user who is currently logged in through the agent.
13	initiate_auth	Starts a new authentication process using email, phone number, or passkey methods.
14	link_profile	Connects multiple separate social media accounts to one master wallet address.
15	list_solana_wallets	Returns a list of known Solana wallet addresses for management purposes.

#	TOOL	DESCRIPTION
16	list_transactions	Fetches a history listing of recent transactions associated with an address.
17	pregenerate_wallet	Creates and saves a new wallet for a user before they ever log into the application.
18	query_events	Retrieves all historical events that occurred from a specific contract.
19	query_transactions	Fetches a detailed transaction log for a given smart contract address.
20	read_contract	Runs one or more read-only calls on a contract, allowing you to view data without changing anything.
21	send_raw_transactions	Broadcasts highly encoded transactions that bypass standard function calls.
22	send_solana_tokens	Sends a specified amount of Solana tokens to another wallet address.
23	settle_payment	Completes the payment process by writing the required data directly onto the blockchain.
24	sign_solana_message	Creates a verifiable signature for a message on the Solana network.
25	sign_solana_transaction	Signs a transaction payload to authorize its execution on Solana.
26	social_auth	Handles standard OAuth procedures for logging in via major social media providers.
27	unlink_profile	Separates a linked social identity from an existing wallet address.
28	verify_payment	Confirms that a signed payment payload is valid and correctly formatted.
29	write_contract	Executes one or more calls that modify the state of a contract, making changes permanent on-chain.

See It in Action

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

U Read the total supply of the contract 0x123... on chain 1.

I've executed the `read_contract` call. The total supply for the contract on Ethereum (Chain ID: 1) is 1,000,000 tokens.

U Start an email authentication for dev@example.com.

I've triggered `initiate_auth` via email for dev@example.com. The user should receive a verification code shortly. Please provide the code to `complete_auth` when ready.

U Search for user details associated with address 0xAbc123...

Using `get_user_details`, I found that this address is linked to User ID 'user_987' with the verified email 'admin@web3app.com'.

Frequently Asked Questions

01 **How do I start user authentication with ThirdWeb MCP?**

You initiate the process using `initiate_auth`. This tool supports various methods like email, phone numbers, and social logins (via `social_auth`). You must follow up by completing the login flow using `complete_auth`.

02 **What is the difference between reading and writing contracts with ThirdWeb MCP?**

Use `read_contract` when you only need to view data, like checking a total supply. If you need to change anything—send tokens or update a record—you must use `write_contract`.

03 How do I transfer Solana tokens using ThirdWeb MCP?

First, check the current balance with `get_solana_balance`. Then, you can calculate the exchange rate with `get_solana_swap_quote`, and finally execute the transfer by calling `send_solana_tokens`.

04 Can ThirdWeb MCP manage user profiles across different social networks?

Yes. You use `link_profile` to tie multiple social identities (via `social_auth`) to a single wallet address, ensuring your user profile is comprehensive even if they use different logins.

05 What should I do if a transaction fails?

Always check the status after broadcasting with `get_transaction_status`. This tool tells you exactly why the transaction failed or if it's still pending confirmation on the chain.

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 `"thirdweb": { "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

ThirdWeb 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 ThirdWeb. 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	ThirdWeb MCP
Server ID	`019e38fb-2dc5-702e-96b5-a3a044fb3e99`
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/thirdweb.