MCP SERVER | NO CODE | CLOUD HOSTED

Liveblocks MCP

Control real-time state and collaboration data.

Liveblocks MCP connects your AI agent to real-time collaboration infrastructure. Manage collaborative spaces, authorize user permissions, track shared document state (Yjs), and resolve comment threads directly from your client. Use this to build complex multiplayer features without leaving your IDE.

F Quality Score 3.6/100

real-time-collaboration | room-management | user-presence | shared-state | multiplayer-infrastructure | api-auth

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

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.

Liveblocks MCP

18 tools available
Cloud-hosted on Vinkius

This connector lets you control the hardest part of modern web apps: real-time state. Instead of building out a clunky backend just to handle user presence or shared document editing, you can orchestrate those experiences using natural language prompts through your AI client. Your agent handles room lifecycle—creating spaces, managing who gets in, and deleting them when they're done. You can read the current status of collaborative storage or patch it directly via commands. It also keeps track of conversations within a space, letting you query comment threads to see what was decided days ago. When working with complex real-time systems, having all this functionality centralized is huge. With Vinkius hosting this MCP, you connect once and get access to powerful tools for handling everything from user authentication to broadcasting custom events across any compatible client.

Core Capabilities

01 — Manage collaborative spaces

Create, read, update, or delete entire collaboration rooms.

02 — Control user access and identity

Obtain tokens to authorize users and identify them within a room for proper permissions.

03 — Work with shared document state

Inspect, patch, or update the raw JSON data of collaborative documents (Yjs).

04 — Track discussions and threads

Create new conversation threads and query existing comment history within a room.

05 — Monitor presence and events

Get lists of active users in a room or push custom data broadcasts to all connected clients.

One Click on Vinkius — From Prompt to Execution

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

01 Subscribe to this MCP using your Vinkius credentials.

02 Input your Liveblocks Secret Key into the connection settings.

03 Tell your AI client what you need to do, like 'List all active users in the main editor room,' and it executes the command.

The bottom line is that your agent treats complex real-time infrastructure like a simple set of commands you can type into conversation.

Built For

This MCP is for full-stack developers building anything multi-user. If your job requires managing shared state, user permissions, or live collaboration features, this saves days of backend boilerplate.

Full-stack Developer

Needs to quickly debug a room's storage structure or programmatically manage access tokens without writing dedicated microservices.

DevOps Engineer

Must automate the provisioning of collaborative environments and enforce complex user access rules across multiple project rooms.

Product Manager

Needs a simple way to monitor which collaboration sessions are active or review historical comment threads before committing to a feature build.

What Changes When You Connect

01 Debug complex storage issues instantly. Instead of manually connecting to a database, use `get_storage` or `patch_storage` to inspect and modify the room's shared state directly through your agent.

02 Manage user entry points without writing auth middleware. Use `authorize_user` and `identify_user` to ensure only authorized clients can enter specific collaboration rooms.

03 Keep track of project decisions efficiently. You can use `create_thread`, `list_threads`, or `resolve_thread` to manage the history of discussions, keeping context organized within a room.

04 Monitor who is working right now. The `list_active_users` tool gives you an immediate list of all users connected to a given room, useful for status checks or presence systems.

05 Control the entire lifecycle of collaborative spaces. Use `create_room`, `update_room`, and `delete_room` to provision and decommission collaboration environments programmatically.

Real-World Applications

The onboarding team needs to check permissions for a new project room.

A dev asks their agent: 'List all rooms associated with the marketing department, then authorize user X and get the full metadata.' The agent uses `list_rooms` first, identifies the correct space, runs `authorize_user`, and confirms access rights using `get_room`.

A shared document needs a state correction.

The PM notices some data is corrupted. They ask their agent to 'Inspect and fix the main editor's storage.' The agent runs `get_storage`, identifies the faulty tree, and then uses `patch_storage` to apply the necessary JSON fix.

Tracking a complex feature discussion.

A developer needs to know what was decided last week. They ask their agent: 'Check out the thread for the payment gateway design.' The agent uses `get_thread` and then runs `resolve_thread` after confirming the final decision.

Multiplayer game session setup.

A developer needs to set up a new private gaming room. They prompt: 'Create a new room called 'Alpha Test' and broadcast an event saying it's open.' The agent uses `create_room` followed by `broadcast_event`.

Patterns to Avoid

Using simple CRUD for state changes

✗ AVOID

Trying to manage shared data using only basic 'update' functions, resulting in lost context and corrupted document history.

✓ INSTEAD

Don't just update the whole room. Use `patch_storage` to apply precise JSON Patch operations, or use `get_ydoc` followed by `update_ydoc` for binary synchronization.

Manual user permission checks

✗ AVOID

Writing custom code every time a new user needs access to a room, leading to brittle and redundant authorization logic.

✓ INSTEAD

Let the agent handle this. Use `identify_user` first, then use `authorize_user` to issue the necessary token for immediate access.

Forgetting historical context

✗ AVOID

Starting a new discussion without linking it to past decisions, forcing teammates to search through old emails or documents.

✓ INSTEAD

Use `create_thread` and `get_thread` together. This keeps the conversation history linked directly within the collaboration room itself.

The Right Fit

Use this MCP if your application's core functionality revolves around multiple users interacting with a single, evolving piece of data—think shared whiteboards, live document co-editing, or complex gaming lobbies. You need mechanisms for real-time presence (`list_active_users`), state synchronization (`patch_storage`), and structured communication history (`get_thread`). Don't use this if you just need simple database record keeping; general CRUD tools work fine then. If your application is single-user, or only handles asynchronous messaging without shared document states, this connector is overkill. This MCP solves the problem of 'state persistence in a group context,' making it essential for any truly collaborative tool.

Managing real-time collaboration state used to feel like an entire backend job.

Today, building something that handles multiple users collaborating on the same document—like a shared whiteboard or a group planning session—means dealing with layers of complexity. You have to manage user tokens, track who is online, and implement custom logic just to sync every keystroke across different clients, all while worrying about data corruption.

With this MCP, you stop writing that boilerplate code. Instead, your agent handles the entire infrastructure layer. You simply ask it to 'get the current state' or 'update the shared document.' The complexity of real-time synchronization disappears; you just get clean, actionable data.

Control the collaborative room lifecycle with Liveblocks MCP.

Manually managing rooms means writing separate functions for listing spaces, creating them when needed, and cleaning up old or abandoned environments. It's tedious, error-prone code that slows down development velocity.

Now you can use `list_rooms` to see all available collaboration areas and `create_room` instantly programmatically. You treat the entire room lifecycle—from inception to deletion—as a single conversational task.

Liveblocks: 18 Tools for Collaborative Systems

These tools give your agent granular control over every aspect of a collaborative space, from user authentication to shared document patching.

#	TOOL	DESCRIPTION
01	`authorize_user`	Generates an access token, allowing a client to join a specific room.
02	`broadcast_event`	Sends a JSON event to all users currently connected in the specified room.
03	`create_room`	Sets up an entirely new collaborative space with a unique ID.
04	`create_thread`	Starts a new discussion thread and writes the first comment automatically.
05	`delete_room`	Permanently removes an existing collaboration room.
06	`get_room`	Retrieves all metadata and details for a specified room.
07	`get_storage`	Reads the entire current state tree of the room's shared data storage.
08	`get_thread`	Pulls up all comments and details for a specific discussion thread.
09	`get_ydoc`	Outputs the collaborative document's current state as a JSON object representation from Yjs.
10	`identify_user`	Obtains an ID token for a specific client, managing user permissions on the backend.
11	`initialize_storage`	Sets up or resets the room's collaborative storage state.
12	`list_active_users`	Retrieves a list of all user IDs currently connected and active in a specific room.
13	`list_rooms`	Lists all available rooms, with options to filter by metadata or access level.
14	`list_threads`	Fetches a list of every discussion thread that exists within a room.
15	`patch_storage`	Applies precise JSON Patch operations to modify the room's storage state.
16	`resolve_thread`	Marks a discussion thread as resolved, closing it out for future reference.
17	`update_room`	Modifies the properties of an existing room, such as its name or description.

#	TOOL	DESCRIPTION
18	update_ydoc	Sends a binary update to synchronize the shared Yjs document state across clients.

See It in Action

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

U List the first 10 rooms in my Liveblocks project.

I've retrieved your rooms. You have active rooms like 'main-editor', 'collaboration-hub', and 'test-room-1'. Would you like to see the metadata for any of these?

U Create a new room with ID 'sprint-planning' and set default access to room:write.

Room 'sprint-planning' has been successfully created with 'room:write' as the default access level. You can now start authorizing users for this space.

U Who is currently active in the room 'main-editor'?

Checking active users... I found 3 users currently connected to 'main-editor': 'alice_dev', 'bob_designer', and 'charlie_pm'.

Frequently Asked Questions

01 **How do I manage user access using Liveblocks MCP?**

You use `identify_user` to get an ID token for the client, and then you run `authorize_user` with that token. This sequence ensures the user has explicit permission to enter the room.

02 **Can I read the full shared document history with Liveblocks MCP?**

Yes. You can use `get_ydoc` to get a JSON representation of the current state, or you can use `get_storage` for more detailed tree information.

03 **What is the difference between listing rooms and getting room details with Liveblocks MCP?**

Listing rooms (`list_rooms`) gives you a directory of available spaces. Getting room details (`get_room`) fetches all the specific metadata for one, already identified, space.

04 **How do I prevent users from entering a room without permission?**

You must first use `identify_user` to get their ID token. Then, you invoke `authorize_user` using that token and specify the desired permissions for access.

05 **Can Liveblocks MCP handle simple messages in a room?**

While it doesn't send plain text messages directly, it handles structured discussions. You can use `create_thread` to start a discussion or `broadcast_event` for custom system alerts.

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 `"liveblocks": { "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

Liveblocks 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 Liveblocks. 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	Liveblocks MCP
Server ID	`019e38b9-385c-737c-ae0f-89e2c1671ed1`
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/liveblocks.