MCP SERVER | NO CODE | CLOUD HOSTED

Pusher Channels MCP

Control real-time messages, from debugging to production.

Pusher Channels MCP lets your AI client manage real-time messaging infrastructure directly through natural language. Trigger events, monitor which channels are active, list connected users, and even force logouts without touching a debug console or API dashboard. It's operational control for pub/sub systems.

F Quality Score 3.6/100

real-time | websockets | pub-sub | event-broadcasting | webhooks

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

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.

Pusher Channels MCP

6 tools available
Cloud-hosted on Vinkius

This MCP connects your entire Pusher Channels setup to any compatible AI client, giving your agent direct access to real-time messaging controls. Instead of navigating separate dashboards to check system health, you talk to your AI and it executes the necessary operations immediately. You can trigger events—whether that's sending a single notification or broadcasting data across dozens of channels—by simply stating what needs to happen. It also lets you monitor who's connected right now; you get lists of active channels and see exactly which users are subscribed to presence streams. If security demands it, the agent can terminate specific user connections instantly. Connecting through Vinkius means your AI client accesses this control layer alongside thousands of other tools, making it a single point for managing complex web infrastructure.

Core Capabilities

01 — Broadcast real-time events

You can trigger specific messages or send multiple event batches to one or more channels instantly.

02 — Audit channel status and users

The MCP lists all active channels, helps you filter them by prefix, and retrieves the current list of subscribed user IDs for any given presence channel.

03 — Control individual connections

It handles security and session management by terminating all WebSocket connections for a specific user ID when needed.

One Click on Vinkius — From Prompt to Execution

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

01 First, subscribe to this MCP and provide your Pusher App ID, Key, Secret, and Cluster credentials.

02 Next, tell your AI client what action you need—for example, 'List all channels starting with payment-'.

03 The agent executes the command against your live infrastructure and reports the status or data back to you.

The bottom line is that your AI acts like a real-time operations console for your messaging system.

Built For

This MCP is built for backend developers and ops engineers who are tired of context switching between code editors, debugging consoles, and separate monitoring dashboards. If you manage live pub/sub or real-time features, this saves serious time.

Backend Developer

You use the MCP to trigger test events and verify webhook payloads directly from your natural language prompt, speeding up local debugging.

DevOps Engineer

You monitor overall channel activity, check user counts across multiple channels, and manage connections without ever opening the Pusher dashboard.

Technical Support Specialist

When a user reports a connection issue, you can quickly identify if they are connected to a presence channel or terminate problematic sessions immediately for investigation.

What Changes When You Connect

01 You stop jumping into the Pusher Debug Console. Instead, your AI client becomes a full infrastructure operator, allowing you to trigger test events and verify payloads instantly.

02 Manage user presence easily. You can use `list_channel_users` to get immediate lists of who's connected to a channel without manually checking status dashboards.

03 Maintain security by controlling sessions. The `terminate_user_connections` tool lets you force logouts or end problematic connections simply by naming the user ID.

04 Handling large updates is fast. Use `trigger_batch_events` when you need to send data across multiple channels at once, instead of running several single triggers.

05 Deep debugging visibility comes from tools like `get_channel`, letting you query a channel's detailed state—perfect for tracing message flow issues.

Real-World Applications

Debugging an order webhook failure

A developer notices that the 'orders' channel isn't receiving test notifications. They prompt their agent, asking it to `trigger_event` with a sample payload. The AI confirms if the event was broadcasted correctly and checks the channel state using `get_channel`, confirming the issue is upstream.

Auditing user access during an incident

The ops engineer suspects unauthorized access. They ask their agent to use `list_channels` to see all active channels, then select a suspicious channel and run `list_channel_users`. This immediately reveals who is currently subscribed.

Forcing a user logout due to security risk

A support team member needs to instantly disconnect an account. They instruct their agent to use `terminate_user_connections` with the user ID. The connection drops immediately, securing the session without needing console access.

Updating multiple system components simultaneously

A new feature launches across five different microservices that rely on real-time updates. Instead of running five separate commands, the engineer uses `trigger_batch_events` to update all channels in one go.

Patterns to Avoid

Manually checking dashboards

✗ AVOID

A developer has to open the Pusher dashboard, navigate to the specific channel, and manually click 'Test Event' repeatedly just to debug a single payload structure.

✓ INSTEAD

Just tell your agent to `trigger_event` directly. Your AI client handles the console interaction for you, letting you focus on the payload data instead of the mechanics.

Over-relying on APIs/Code

✗ AVOID

A support team member must write and execute a full API call script every time they need to check if a user is still connected or list active channels.

✓ INSTEAD

Use the natural language capabilities. Ask your agent to `list_channel_users` for immediate status checks, making it as simple as conversation.

Forgetting connection control

✗ AVOID

A user leaves a session open and needs to be immediately logged out, but the team only knows how to disable the service entirely.

✓ INSTEAD

The `terminate_user_connections` tool lets you target specific users by ID. You can kill one person's connection without disrupting anyone else on the channel.

The Right Fit

Use this MCP if your application relies heavily on real-time, publish/subscribe communication where event triggering and user presence are critical parts of the workflow. Specifically, if you need to monitor who is subscribed (`list_channel_users`) or manage sessions by force-disconnecting a specific user (`terminate_user_connections`), this is essential. Don't use it if your primary goal is simple data storage or retrieval; for that, you'd look at database connectors. If all you need to do is send a one-off message without knowing the channel status beforehand, consider an integration focused on general messaging services instead. This MCP gives deep operational control over the Pub/Sub layer.

Debugging real-time connections used to feel like a maze of dashboards and consoles.

Today, checking if your event payload hit the right place means jumping between multiple tabs. You open the Pusher debug console for one check, then switch to the channel list just to see who's listening. It’s a manual, frustrating process that takes minutes just to verify basic connectivity.

With this MCP, you ask your agent what needs doing. If an event failed, it runs `get_channel` and reports the specific state failure instantly. The entire operational check—from listing channels with `list_channels` to triggering a test message—gets done in one chat window.

Pusher Channels MCP: Full Control Over Your Live Pub/Sub System

The need to manually check connection status, list all active subscribers, or run test events is gone. You don't write the boilerplate code for these checks; your agent does it.

You get reliable, immediate system visibility and control without needing deep knowledge of the Pusher API reference page. It just works.

Pusher Channels: 6 Tools for Real-Time Control

These tools give your agent granular control over every aspect of your real-time communication infrastructure, from basic status checks to complex event broadcasting.

#	TOOL	DESCRIPTION
01	`get_channel`	Fetches specific metadata and status information for one particular communication channel.
02	`list_channel_users`	Retrieves the list of user IDs currently subscribed to a defined presence channel.
03	`list_channels`	Fetches an overview and status data for multiple existing channels.
04	`terminate_user_connections`	Stops all active WebSocket connections associated with a single user ID, forcing a logout or reset.
05	`trigger_batch_events`	Sends multiple events to various channels simultaneously in one efficient operation.
06	`trigger_event`	Sends a single, specific event with a defined data payload to one or more channels.

See It in Action

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

U Trigger a 'new-order' event on the 'orders' channel with data '{"id": 123, "total": 50.00}'.

Event 'new-order' successfully triggered on channel 'orders'. The payload has been broadcasted to all active subscribers.

U List all active channels that start with 'presence-'.

I found 3 active presence channels: 'presence-chat-room-1', 'presence-lobby', and 'presence-editor-v2'. Would you like to see the users in any of these?

U Terminate all connections for user ID 'user_999'.

All active WebSocket connections for 'user_999' have been terminated. The user will need to re-authenticate to reconnect.

Frequently Asked Questions

01 **How do I check if a specific user is connected using Pusher Channels MCP?**

Use `list_channel_users`. This tool fetches a clear list of all user IDs that are currently subscribed to the presence channel, giving you an instant count.

02 **What's the difference between `trigger_event` and `trigger_batch_events`?**

`trigger_event` is for sending one specific notification or data packet. Use `trigger_batch_events` when you need to send multiple, related events across several channels at once.

03 **Can I force a user off the channel using Pusher Channels MCP?**

Yes. The `terminate_user_connections` tool lets your agent kill all active WebSocket connections for a given user ID, forcing them to reauthenticate.

04 **What if I want to see what channels are available in my app?**

Use `list_channels`. This function fetches an overview of all existing communication channels and their current status data for auditing purposes.

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 `"pusher-channels": { "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

Pusher Channels 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 Pusher Channels. 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	Pusher Channels MCP
Server ID	`019e38dd-6db3-731b-b4b3-97c94a3981a3`
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/pusher-channels.