MCP SERVER | NO CODE | CLOUD HOSTED

Upstash Redis MCP

Manage Cache & State Without Leaving Your Chat Window

Upstash connects your serverless Redis database directly into any AI agent, letting you manage complex data structures and caching layers using plain conversation. You can run read/write operations—like fetching user sessions, incrementing counters, or managing message queues—without ever touching a command line or dedicated terminal client.

A+ Quality Score 98.33/100

redis | serverless | key-value-store | caching | data-structures | rest-api

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

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.

Upstash MCP

23 tools available
Cloud-hosted on Vinkius

Connecting your Upstash Redis data store to an AI agent means treating it like an extension of your own memory. Instead of having to jump between chat interfaces and a separate database console, you simply ask your agent to perform actions on the cached data.

Your agent handles all the necessary complexity. You can tell it to check if a key exists, retrieve all fields from a user's session hash, or push messages onto a list for background processing. If you need to run multiple commands at once, like checking three different keys and then deleting one, your agent sequences them for you. It’s about making the database an active participant in your workflow. This MCP is available across Vinkius, giving you access to this Redis power from any compatible client.

This functionality takes data management out of the terminal and right into the flow of natural conversation.

Core Capabilities

01 — Manage Key-Value Pairs

Set a value for a specific key or retrieve that value using simple read/write commands.

02 — Handle Structured Data Sets

Store and access collections of related data, like user attributes or feature flags, within complex hash structures.

03 — Process Message Queues

Use lists to add items to the end of a queue (right) or process them from the beginning (left), mimicking stack and queue behavior.

04 — Track Unique Memberships

Maintain unique collections of identifiers, such as user IDs who have signed up for an event, ensuring no duplicates are stored.

05 — Monitor Database Health and State

Check if a key exists, see how long it has until expiration (TTL), or determine its data type.

06 — Execute Batch Operations

Run multiple distinct commands—like getting three different values and then incrementing a counter—in a single request.

One Click on Vinkius — From Prompt to Execution

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

01 You connect your Upstash credentials (URL and Token) to the MCP within Vinkius.

02 Your AI agent interprets your natural language request, determining which Redis data operation is needed (e.g., 'get' or 'hset').

03 The MCP executes the command against your live database and returns the resulting data directly to your chat interface.

The bottom line is that you treat your serverless Redis cache like an active, conversational source of truth for your entire application stack.

Built For

This MCP is essential for developers and operations teams who need to interact with cached data structures without context switching. If you're tired of opening a separate terminal just to check if a feature flag was set or how many times a counter ticked, this tool saves you time.

Backend Developer

Managing session storage, rate limiting counters, and temporary state data structures directly from the chat interface during development.

DevOps Engineer

Auditing database health by checking key patterns or setting time-to-live (TTL) expirations on critical configuration keys across environments.

Full-Stack Developer

Manipulating complex application state, such as user roles and permissions stored in hash records, to test application logic before deployment.

What Changes When You Connect

01 You avoid complex data flow by using `sadd` or `smembers`. Instead of writing code to maintain unique user lists, you simply ask your agent to manage the set membership.

02 Rate limiting becomes trivial. You can use `incr` and `decr` in conversational prompts to track usage counts instantly, making it easier to enforce business logic without adding boilerplate code.

03 Auditing data structure health is fast. Use `list_keys` or `key_type` to see what keys exist and confirm they are stored as the expected hash format when debugging a session problem.

04 Batch operations are simplified with the `pipeline` tool. Instead of sending five separate requests, you ask your agent to run them all at once for maximum efficiency.

05 Data lifecycle management is automatic. Setting an expiration using `expire` ensures that temporary keys—like shopping cart states—are deleted without manual cleanup tasks.

Real-World Applications

Debugging a broken user session

A full-stack developer notices user data is corrupted. They ask their agent to use `hgetall` on the relevant session key and then run `key_type` to confirm the structure, instantly pinpointing if it's stored incorrectly.

Implementing a simple message queue

An operations engineer needs background processing. They use `rpush` to add tasks to a queue list and then have their agent process them using `lrange` or `pop`, completely bypassing the need for dedicated worker services.

Tracking feature flag rollout

A backend developer wants to test new features on specific users. They use `set` to write a unique key like 'feature:v2:user123' and then set a short TTL using `expire`, guaranteeing the flag expires after testing.

Building an event notification system

A team needs multiple services to react when user data changes. They use `publish` on a central channel, allowing any listening service (like logging or email) to automatically respond without direct coupling.

Patterns to Avoid

Manual CLI Context Switching

✗ AVOID

Having to open up the Upstash web dashboard, manually type `HGETALL user:123`, wait for the result, then switch tabs to run `EXPIRE user:123 3600`.

✓ INSTEAD

Instead, ask your agent directly: 'Get all data from user:123 and set its expiry to one hour.' The agent executes both `hgetall` and `expire` sequentially for you.

Over-relying on single operations

✗ AVOID

Telling the agent simply to 'change the user status'. This might only execute a basic `set`, forgetting that the key needs an expiration or multiple fields need updating.

✓ INSTEAD

Be specific: 'Update the user's profile hash by setting their email and then set the entire record to expire in 24 hours.' Use `hset` combined with `expire`.

Forgetting transactional needs

✗ AVOID

Trying to increment a counter, check its type, and then delete it using three separate commands. If one fails, the whole process breaks.

✓ INSTEAD

Use the `pipeline` tool. Ask your agent: 'Run this sequence: get the current count, increment it by 1, and confirm the new value.' This guarantees all steps run together.

The Right Fit

You need this MCP if your primary pain point is interacting with Redis data structures (hashes, sets, lists) without opening a terminal. If you mostly just need simple key-value lookups and don't care about complex structures or workflows, a basic key-value tool might suffice. However, because Upstash handles specialized types like Lists and Sets, this MCP shines when your application state is complex—think session management, message queues, or feature flag systems. Don't use this if you need to manage data in a relational database (SQL); that requires an entirely different connection. But if your stack is built on microservices using Redis for caching or queuing, this is the right tool.

The Cache Layer Is Always Out of Reach

When you need to check a user's session data or update a feature flag, you currently have to context switch. You leave your development environment, open the Redis CLI in a separate terminal window, authenticate, manually type out commands like `HGETALL key:user:` and then copy-paste results into your ticket tracker.

With this MCP, that manual step disappears. You just ask your agent, 'What are all the fields for user 456?' Your agent runs the necessary command internally and delivers the formatted data right back to you in conversation.

Upstash Redis MCP Gives You Full Control Over Data Structures

Previously, managing structured data meant separate concerns. Updating a list required `LPUSH` or `RPUSH`. Tracking unique users meant running `SADD`. If you needed to combine these steps—say, adding three IDs and then reading the whole set—it was a multi-step chore across several tabs.

Now, your agent coordinates it all. You tell it: 'Add these five user IDs to the active group list.' It runs the `sadd` operation and confirms success instantly. Your AI acts as an in-memory data engineer.

Upstash Redis: 23 Tools for Data Management

Use these tools in your AI client to perform every common Redis operation—from simple key lookups to complex list management and batch executions.

#	TOOL	DESCRIPTION
01	`decr`	Decrements a numeric value associated with a key, useful for tracking limited resources or decreasing counts.
02	`del`	Removes an entire key from the database; be careful because this action cannot be undone.
03	`exists`	Quickly checks whether or not a specific data key is present in your Upstash Redis instance.
04	`expire`	Sets an automatic deletion timer (TTL) on a key, ensuring temporary data cleans itself up later.
05	`get`	Retrieves the simple string value associated with a given key name.
06	`hget`	Pulls out one specific field's value from a structured hash record.
07	`hgetall`	Retrieves every single field and its value contained within a full hash structure.
08	`hset`	Adds or updates specific fields and values inside an existing structured hash key.
09	`incr`	Increases a numeric counter value by one, useful for simple counting mechanisms.
10	`list_keys`	Lists keys in the database based on patterns or prefixes to help you audit what data exists.
11	`llen`	Returns a count of how many elements are currently stored within a specific Redis list.
12	`lpush`	Adds new values to the beginning (left side) of a message queue list.
13	`lrange`	Retrieves elements from a specific range within an ordered Redis list.
14	`pipeline`	Executes many different database commands simultaneously in one call, improving efficiency for batch tasks.
15	`publish`	Sends a message out to all listening subscribers on a specific channel, triggering event logic.
16	`rpush`	Adds new values to the end (right side) of a message queue list.
17	`sadd`	Adds one or more unique items to a set, automatically ignoring any duplicates.

#	TOOL	DESCRIPTION
18	`set`	Writes a new value to a key and optionally sets an automatic expiration time.
19	`sismember`	Checks quickly if a unique member ID is already part of a specific set.
20	`smembers`	Retrieves every single unique member ID that belongs to a given set.
21	`srem`	Removes one or more specified members from a unique collection set.
22	`ttl`	Checks how many seconds are left until a key automatically deletes itself.
23	`key_type`	Identifies and reports the underlying data type of a given key (e.g., hash, list, string).

See It in Action

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

U Check if my Upstash Redis database is responding.

Connection verified! The database responded with PONG. Your Upstash Redis instance is healthy and accepting commands.

U Set a feature flag called 'dark_mode' to true for user 'user123' with a 1 hour expiry.

Done! I've set the key `dark_mode:user123` to `true` with a 3600-second (1 hour) expiry. The flag will automatically expire after that time.

U Show me all keys matching the pattern 'session:*' and check their data types.

I found 5 keys matching 'session:*'. Checking their types: `session:abc` is a hash, `session:def` is a hash, `session:ghi` is a string, `session:jkl` is a set, and `session:mno` is a list. Most sessions are stored as hashes (field-value pairs for user data).

Frequently Asked Questions

01 **How do I check if a key exists using Upstash Redis MCP?**

You use the 'exists' tool to quickly confirm key presence without retrieving any value. This is great for pre-checking data before attempting a read or write operation.

02 **Can I run multiple commands at once with Upstash Redis MCP?**

Yes, you use the 'pipeline' tool to execute several commands in one request. This saves time and is essential for efficient batch updates across your data structure.

03 **What is the difference between `incr` and `set` with Upstash Redis MCP?**

`incr` automatically increases a numeric counter by one, ensuring atomic counting. The 'set' tool simply writes a new value or overwrites an existing string value.

04 **How do I delete data safely using Upstash Redis MCP?**

Use the 'del' tool to remove keys entirely. However, remember that this action is irreversible, so always confirm which key you are deleting first.

05 **Does Upstash Redis MCP support message queuing?**

Yes, it supports queue patterns using lists. You can add items with 'rpush' and then process them by retrieving ranges or popping the elements out of the list.

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 `"upstash": { "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

Upstash 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 Upstash. 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	Upstash MCP
Server ID	`019d8496-4907-7232-bb59-ae5b7e9eaf04`
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/upstash.