MCP SERVER | NO CODE | CLOUD HOSTED

OpenFGA (Fine-Grained Auth) MCP

Manage access rules by simply asking your agent.

OpenFGA (Fine-Grained Auth) connects your AI agent to an open-source system for Relationship-Based Access Control (ReBAC). Manage complex permissions, define access models, and check user rights against specific resources using natural conversation. It lets you program security policies without writing boilerplate API code.

A+ Quality Score 98.33/100

authorization | rebac | access-control | permissions | security-policy | identity-management

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/openfga-fine-grained-auth

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.

OpenFGA (Fine-Grained Auth) MCP

16 tools available
Cloud-hosted on Vinkius

You can use this MCP to manage your application's most sensitive logic: who sees what. Instead of manually constructing authorization queries or clicking through multiple administrative dashboards, you talk to your AI agent and it handles the complexity for you. You define entire data stores, model relationships between users and objects, and track all permissions using plain conversation.

For example, instead of writing a complex SQL join to check if 'User A' can view 'Document B' because they are part of 'Group C', you simply ask your agent to run the authorization check. The MCP handles reading those relationship tuples and instantly evaluating the result. By connecting this OpenFGA instance through Vinkius, you give your AI client direct, conversational control over your security layer. You can audit models, list all users with specific access rights, or even monitor the health of your entire authorization setup—all without leaving your chat interface.

Core Capabilities

01 — Check user permissions

Instantly verifies if a specified user has a defined relationship to a particular object.

02 — Manage data environments

Create, list, and delete isolated stores to keep authorization data separate for different applications or testing phases.

03 — Model system rules

Define and retrieve the complex types and relations that govern how your entire system's permissions work.

04 — Read access history

Queries stored relationship tuples to see exactly which users have what rights to which objects.

05 — Audit object visibility

Lists all the resources (objects) a particular user is allowed to interact with.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/openfga-fine-grained-auth** — connect your AI agent in three steps.

01 Subscribe to this MCP and provide your OpenFGA API URL and any necessary authentication tokens.

02 Your AI client establishes a connection, allowing it to speak the language of relationship-based access control (ReBAC).

03 You ask your agent questions like, 'Does user X have view rights on object Y?' and get an immediate pass/fail answer.

The bottom line is that you talk naturally about security policies, and the MCP translates those requests into secure, functional data checks.

Built For

This connector is built for people whose job involves ensuring nothing gets exposed or misused. If your daily routine requires manually checking permissions across multiple systems or building complex authorization logic in code, you need this.

Security Engineer

Auditing relationship tuples and verifying that the entire authorization model works correctly without needing to run tedious manual API calls.

Backend Developer

Testing new permission rules or iterating on complex access models directly from their IDE, speeding up development cycles immensely.

DevOps/SRE

Monitoring the health of authorization stores and managing environments across different clusters to ensure continuous uptime and policy integrity.

What Changes When You Connect

01 You eliminate manual API calls. Instead of writing code to check if a user has permission, you tell your agent to run `check_relation`, and it handles the complex logic immediately.

02 It keeps your data clean. You can use `list_stores` to keep sensitive environments (like 'Production' vs. 'Staging') completely isolated within separate stores.

03 You gain full visibility into permissions. Running `read_tuples` lets you query stored relationship data, so you always know exactly who has access to what.

04 Development gets faster. Developers use the MCP to `write_authorization_model` and instantly test changes without deploying new code just for a policy tweak.

05 You get immediate health checks. The `health_check` tool lets SREs quickly verify system availability, minimizing downtime due to authorization issues.

Real-World Applications

Auditing access after a security incident

An engineer needs to know if three different users can view a specific sensitive document. Instead of running three separate manual checks, they prompt their agent: 'List all users who have the viewer relation to document:123.' The agent executes `list_users` and provides an instant list.

Implementing a new feature with complex rights

A backend developer needs to add a new type of permission. They use the MCP to run `get_authorization_model` to see existing patterns, then use `write_authorization_model` to define and test the new rules before committing any code.

Preparing for multi-environment deployment

A DevOps team needs to ensure that their staging environment is completely separate from production. They use the agent to `create_store` specifically named 'Staging' and another called 'Production', ensuring no data overlap.

Reviewing historical permission changes

A security manager wants to see if an admin accidentally granted access last week. They ask the agent to `read_changes` on a specific object, immediately retrieving a timeline of every relationship tuple modification.

Patterns to Avoid

Writing authorization logic in application code

✗ AVOID

If you embed permission checks (like checking if user X is in group Y) into your main Python files, those rules are hard to update and require a full redeployment just for one policy change.

✓ INSTEAD

Use this MCP to define the rule set using `write_authorization_model`. This keeps the logic separate from the application code. When a policy changes, you only update the model via your agent, not your codebase.

Checking permissions in isolation

✗ AVOID

Calling simple API endpoints one by one to check access for 10 different users and 20 different resources is tedious, error-prone, and slows down the workflow.

✓ INSTEAD

Use `batch_check_relations`. This tool lets you pass multiple user/object pairs in a single request, getting all the answers back efficiently.

Assuming data integrity

✗ AVOID

Relying on manual database queries to verify relationship tuples without proper tracking leads to outdated or inaccurate security records.

✓ INSTEAD

Use `read_tuples` and `read_changes`. These tools give you a verified, auditable view of the current state and all historical changes to your access records.

The Right Fit

Use this MCP if your core problem is defining and enforcing 'who can do what' based on their relationship to an object. If your security model is complex, relying on groups and types (ReBAC), this tool is essential. Think of it as the central brain for all permissions.

However, don't use this MCP if you simply need to read or write basic unstructured data (like a user profile name or a transaction amount). For those tasks, standard database connectors are better suited. You also shouldn't rely on it for general logging; use dedicated monitoring tools for that. If your only goal is listing all users, the `list_users` tool helps, but if you need to *act* on that user list (e.g., send them an email), you need a separate messaging or workflow MCP instead.

The Manual Headache of Managing Access Rights

Right now, checking permissions means jumping between dashboards and API playgrounds. You write the query in one place, copy the results to another, manually verify if the data structure is correct, and then paste it into a third tool just to confirm the access status. It’s slow, and you're always worried about missing a single comma or forgetting which environment you're running against.

With this MCP, that manual process evaporates. You simply tell your agent what needs checking—for instance, 'Does the finance team have write access to Q3 reports?' Your agent uses its connection to OpenFGA and runs all the necessary checks (like `check_relation` or `list_objects`) in one go. The result is a clean, immediate confirmation right inside your chat window.

OpenFGA (Fine-Grained Auth) MCP: Defining Access Rules

The biggest manual step that goes away is the need to constantly write and test raw authorization queries. You don't have to remember the exact

This MCP gives you conversational control over security logic. It means you can update, audit, and validate complex permission rules instantly,

syntax for expanding a relation or how to structure a batch check; you just ask your agent to perform the action.

making security policy management faster and far less risky.

OpenFGA (Fine-Grained Auth) MCP: 16 Tools

These tools let you interact with every aspect of OpenFGA authorization, from checking a single user's access to managing entire data stores.

#	TOOL	DESCRIPTION
01	`check_relation`	Determines if an individual user has the right to access a specific object.
02	`create_store`	Sets up a brand new, isolated OpenFGA data store for your application.
03	`delete_store`	Permanently removes an existing OpenFGA data store.
04	`expand_relation`	Visualizes a complex permission by expanding it into a readable tree structure.
05	`get_authorization_model`	Retrieves the definition of a specific authorization model type used in your system.
06	`get_store`	Fetches detailed information about an existing OpenFGA store.
07	`health_check`	Quickly verifies the operational status of your entire OpenFGA instance.
08	`list_authorization_models`	Presents a list of all authorization models currently defined in your system.
09	`list_objects`	Retrieves a comprehensive list of every object that a user has access to.
10	`list_stores`	Lists all the different OpenFGA stores you have running.
11	`list_users`	Finds and lists every user who has been granted a relationship to an object.
12	`read_changes`	Reads records of changes made to the system's permission tuples over time.
13	`read_tuples`	Queries and retrieves stored relationship data directly from the database.
14	`write_authorization_model`	Writes or updates an authorization model definition in your system.
15	`write_tuples`	Adds new relationship tuples or removes existing ones to manage permissions.

#	TOOL	DESCRIPTION
16	batch_check_relations	Performs multiple user permission checks in a single request.

See It in Action

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

U List all my OpenFGA stores.

I've retrieved your OpenFGA stores. You have 3 stores: 'Staging' (ID: 01H1...), 'Production' (ID: 01H2...), and 'Lab' (ID: 01H3...).

U Check if user 'anne' has the 'viewer' relation to 'document:doc1' in store 01H1...

I checked the relation in store 01H1... and the result is 'allowed: true'. User 'anne' does have 'viewer' access to 'document:doc1'.

U Create a new OpenFGA store named 'Security-Audit-Logs'.

The store 'Security-Audit-Logs' has been successfully created with ID: 01J9...

Frequently Asked Questions

01 **How does the OpenFGA (Fine-Grained Auth) MCP work with different environments?**

You use `list_stores` to see all your available stores. You then use `create_store` or reference an existing store ID when checking a relation, ensuring you're always testing against the correct data environment.

02 **What if I need to check permissions for many users at once using OpenFGA (Fine-Grained Auth)?**

Use the `batch_check_relations` tool. This lets you group multiple user and object pairs into one request, making your auditing process much faster than checking them individually.

03 **Can I model a completely new type of permission in OpenFGA (Fine-Grained Auth)?**

Yes. You use the MCP to `write_authorization_model` and define your custom types and relations, which makes the rule available for future checks.

04 **Where do I find out what kind of access a specific user has?**

You can run `list_objects`. This tool gathers all the objects that a user is entitled to access based on their current relationship tuples in the store.

05 **Is OpenFGA (Fine-Grained Auth) MCP suitable for auditing past changes?**

Yes. The `read_changes` tool allows you to query historical records, so you can track when and how a user's permissions were changed over time.

06 **How can I check if a specific user has access to a resource?**

You can use the `check_relation` tool. Provide the store ID and the relationship details (user, relation, and object) to get an immediate boolean response on whether the access is permitted.

07 **Can I see the history of changes made to relationship tuples?**

Yes, the `read_changes` tool allows you to retrieve the changelog of relationship tuples for a specific store, optionally filtered by object type.

08 **How do I define a new authorization model?**

Use the `write_authorization_model` tool. You will need to provide the store ID, the schema version, and a JSON array of type definitions that describe your relations.

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 `"openfga-fine-grained-auth": { "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

OpenFGA (Fine-Grained Auth) 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 OpenFGA (Fine-Grained Auth). 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	OpenFGA (Fine-Grained Auth) MCP
Server ID	`019e38ce-7d20-71c6-b082-ce7b67e6b6f4`
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/openfga-fine-grained-auth.