MCP SERVER | NO CODE | CLOUD HOSTED

ContextQA MCP for AI Agents

Automate Software Quality Assurance and API Testing

ContextQA lets you take full control of context-aware AI testing right from your chat interface. Manage entire test suites, trigger live runs across multiple environments, and inspect complex API payloads using natural conversation. It's designed for QA engineers and DevOps teams who need deep visibility into automated software quality assurance.

A+ Quality Score 100/100

automated-testing | ai-healing | test-automation | qa-testing | software-quality

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

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.

ContextQA MCP

10 tools available
Cloud-hosted on Vinkius

ContextQA connects the complexity of modern application testing to simple conversation with any AI agent. Instead of logging into a separate dashboard or writing boilerplate scripts, you manage your entire test lifecycle right where you work. You can ask your AI client to list all available projects and then immediately dispatch live tests against them. Need to check if an API endpoint meets OpenAPI standards? Just ask. The platform also monitors active runs, letting you inspect the specific results of AI-healing attempts—showing exactly where a test failed or what structural change caused it. By connecting this MCP via Vinkius, your agent gains direct access to thousands of other development tools, making comprehensive software quality assurance accessible through plain language commands.

Core Capabilities

01 — Manage Test Environments and Projects

List defined test environments and group automated validations into projects.

02 — Map GUI Test Suites

Extract the structure of user interface (GUI) test suites across different project boundaries.

03 — Run Live Automated Tests

Dispatch live testing commands to queue entire test suites against ContextQA clusters directly from your chat.

04 — Audit API and Swagger Payloads

Enumerate automated HTTP assertions and verify structural data payloads against OpenAPI configurations.

05 — Monitor AI-Healing Failures

Inspect detailed test runs to view specific AI-healing states, including failure boundaries and screen captures.

06 — Verify System Boundaries

List physical runtime URLs and group active contexts to verify testing scope across different application layers.

One Click on Vinkius — From Prompt to Execution

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

01 Subscribe to the ContextQA MCP on Vinkius.

02 Enter your unique ContextQA API Key into your AI client's settings.

03 Ask your agent to perform a task, like listing test suites or triggering a run.

The bottom line is you get full command-line control over sophisticated automated testing workflows using only natural conversation prompts.

Built For

QA Engineers and DevOps teams need this MCP when they're tired of jumping between multiple dashboards, manually checking logs, and writing complex scripts just to verify a simple feature. It empowers developers who want to validate API payloads directly from their IDE or Product Owners who need real-time visibility into release readiness without needing technical deep dives.

QA Engineer

A QA engineer uses this MCP to monitor active test runs and analyze AI-healing execution results, pinpointing exactly why a specific automated step failed.

DevOps Team Lead

A DevOps team lead audits overall test suite coverage across multiple projects and monitors pipeline integration statuses in real time via conversation.

Software Developer

A developer uses this MCP to verify API test payloads against schemas or inspect failed test case boundaries immediately within their coding environment.

What Changes When You Connect

01 Stop switching between dashboards. You can monitor active test runs and inspect specific AI-healing states, like failing steps or screen captures, all from your chat.

02 Validate complex APIs instantly. Use this MCP to enumerate automated HTTP assertions and verify structural payloads against OpenAPI configurations without writing a single line of code.

03 Control the entire lifecycle. Easily list bounded test environments using `list_projects` and dispatch live testing commands with `trigger_run`, all through natural language conversation.

04 Deep visibility into failures. Use `get_execution` to query specific AI-healing states, helping you find the precise root cause of a failure that was hard to track manually.

05 Comprehensive coverage mapping. List physical runtime URLs using `list_environments` to verify testing boundaries across multiple application layers before deployment.

Real-World Applications

Investigating a Broken Checkout Flow

A QA Engineer notices the checkout flow is failing intermittently. Instead of logging into three different tools, they ask their agent to run `list_suites` for the 'Checkout-Flow', then use `get_execution` on the failed run ID to pinpoint if the issue was a structural DOM change or an authentication failure.

Validating New Microservice APIs

A developer needs to confirm that a new payment API endpoint adheres exactly to the OpenAPI spec. They use `list_api_tests` and ask their agent to verify the payload structure against the expected schema, getting instant confirmation on success or failure.

Auditing Release Readiness

A Product Owner is concerned about a release candidate. Using `list_projects`, they can see all bounded test environments and then use the MCP to trigger comprehensive runs across multiple critical areas, monitoring the overall health before sign-off.

Debugging Environment Discrepancies

A DevOps team member suspects a bug only appears in staging. They use `list_environments` and group active contexts to verify that all layers—frontend, backend, and database connections—are pointing to the correct target URLs for accurate testing.

Patterns to Avoid

Writing full test scripts in chat

✗ AVOID

The user tries to manually write out a series of API calls or complex steps like, 'First hit X endpoint, then parse JSON Y, then use that value Z...' which is tedious and error-prone.

✓ INSTEAD

Instead, let your agent manage the process. Use `list_api_tests` to check configurations, and use `trigger_run` to execute entire pre-defined job pipelines in one command.

Ignoring environment scope

✗ AVOID

Running a test suite against the wrong target (like running production tests on staging data), leading to misleading results.

✓ INSTEAD

Always start by calling `list_environments` and confirm all required physical runtime URLs are mapped correctly before you ever trigger a run.

Ignoring AI-healing details

✗ AVOID

Simply seeing 'Test Failed' without knowing *why* it failed. The failure reason is often complex (e.g., structural DOM changes).

✓ INSTEAD

Use `get_execution` to dive deep into the run results. This shows you the AI-healing state and exactly which step boundary caused the test to break.

The Right Fit

You should use this MCP if your job requires managing complex, multi-layered automated testing—especially when you need visibility beyond simple pass/fail reports. If verifying API payloads against schemas or running comprehensive test suites across different environments is a regular part of your workflow, this connector is essential. However, don't use it just because you want to run ad-hoc scripts. For quick, single-script testing where the logic doesn't involve complex state management (like multi-step API calls), a general purpose scripting tool might be faster. This MCP excels at orchestration and deep audit capabilities.

ContextQA MCP for AI Agents: Simplifying Automated Test Suite Management

Today, running comprehensive test suites is a massive chore. You have to hop into your testing platform, select the correct project and environment, manually list which GUI tests apply, and then click 'Run.' If something fails, you often get vague error codes telling you *where* it failed, but not *why*, forcing hours of debugging.

With this MCP, you talk to your agent. You ask it to run the 'User-Onboarding' suite in the staging environment. It handles the entire sequence: checking boundaries, dispatching the job, and confirming completion. The punchline is that you get actionable context—you know exactly what broke and why.

ContextQA MCP for AI Agents: Validating API Payloads with ContextQA

Manually verifying APIs means copying the OpenAPI spec, setting up a client (like Postman), and running dozens of assertions just to confirm structure. This process is slow, repetitive, and easy to get wrong if you miss an edge case in payload formatting.

Now, you tell your agent: 'Verify this API against its schema.' The MCP automatically performs the necessary HTTP assertions and validates structural payloads using tools like `list_api_tests`. You don't write a single test request; you just ask it to prove correctness.

ContextQA: 10 Tools for Automated Test Suite Management

Use these tools to manage projects, list available environments, validate APIs, and trigger comprehensive test runs via your agent.

#	TOOL	DESCRIPTION
01	`list_projects`	Lists bounded ContextQA environments that hold groups of automated validations.
02	`get_project`	Retrieves specific Project mapping UUIDs for analyzing execution spaces.
03	`list_suites`	Performs structural extraction matching asynchronous GUI test Suites payloads.
04	`list_cases`	Discovers explicit routing limits that structure ContextQA case definitions.
05	`get_case`	Validates Data Science object extraction and tracks specific test steps boundaries.
06	`list_executions`	Inspects deep internal interactions, tracking global run data chunks.
07	`get_execution`	Executes static queries targeting specific AI-healing test run states.
08	`list_environments`	Lists configured environments, mapping target layers and testing limits.
09	`list_api_tests`	Extracts native REST and OpenAPI testing configuration details.
10	`trigger_run`	Dispatches a live command to execute specific jobs against defined test pipelines.

See It in Action

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

U I need to see all bounded environments for our mobile app project.

ContextQA Project Environments

- `mobile-app-staging` (Active, Last Run: 2 hours ago)
- `mobile-app-prod` (Archived, Last Run: Yesterday)
- `mobile-api-sandbox` (Testing, Active)

You can request a run on any of these environments using the `trigger_run` tool.

U Why did the last execution for project 'vinkius-app-prod' fail?

Analysis Report: vinkius-app-prod (Execution ID: exec_123)

- **Failure Step:** 5. Click Login Button.
- **Root Cause:** Element not found. ContextQA AI-healing attempted location but failed due to a DOM structural change in the main login container.
- **Recommendation:** Review front-end code for selectors on the login button element.

U Can you list all API test configurations I have?

OpenAPI Assertions Found (via `list_api_tests`)

ENDPOINT	METHOD	SCHEMA COVERAGE	LAST VERIFIED
`/user/profile`	GET	Full Payload Check	2026-06-10
`/payment`	POST	Partial (Schema Only)	N/A
`/inventory`	PUT	Full Payload Check	2026-05-01

Frequently Asked Questions

01 **How does ContextQA help me debug a failing test run?**

ContextQA provides deep visibility into failures by showing you the AI-healing state, which tracks exactly why an element wasn't found or what structural change caused the failure. It tells you more than just 'failed'; it tells you *why* it failed.

02 **Do I need to write code to test my API endpoints?**

No. You don't write code; you use ContextQA to enumerate automated HTTP assertions and verify payloads against OpenAPI configurations using natural conversation. It handles the technical complexity for you.

03 **What is 'AI-healing' in the context of ContextQA?**

AI-healing refers to the platform's ability to detect when a test breaks due to small changes (like a button moving) and attempt to automatically adjust the test logic. You can inspect these attempts using specific execution tools.

04 **Can ContextQA manage multiple testing environments?**

Yes, it lists multiple bounded test environments using `list_environments`. This lets you ensure that whether you are testing on staging or pre-prod, the context and boundaries are set correctly every time.

05 **Is ContextQA only for GUI tests?**

Not at all. While it manages complex GUI suites, it also specializes in backend quality assurance by verifying structural payloads against OpenAPI configurations and running API assertions.

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 `"contextqa": { "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

ContextQA 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 ContextQA. 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	ContextQA MCP
Server ID	`019d757b-77c2-7115-b6ac-ddacec759e4a`
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/contextqa.