MCP SERVER | NO CODE | CLOUD HOSTED

TestMu AI MCP

Audit Builds and Debug Sessions via Conversation

TestMu AI manages cross-browser testing data by letting your agent talk directly to your test automation builds. You can list build runs, check specific browser sessions, and pull deep debugging artifacts like video recordings, screenshots, console errors, or precise command logs—all through natural conversation. It gives you full control over complicated CI/CD quality assurance without ever leaving your chat window.

A+ Quality Score 98.33/100

cross-browser-testing | automation-testing | selenium | playwright | visual-testing | ci-cd

MAINTAINED BY VINKIUS CLOUD FOR AI AGENTS · JUNE 2026
vinkius.com/mcp/testmu-ai-formerly-lambdatest

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.

TestMu AI (formerly LambdaTest) MCP

10 tools available
Cloud-hosted on Vinkius

Need to debug why a UI element failed on an old version of Safari? You don't have to jump between dashboards and log files anymore. This MCP connects your agent directly to TestMu AI, giving you immediate access to your entire cross-browser testing suite history. Instead of searching through complex UIs, you ask the questions—like 'Show me all build failures in staging' or 'Give me the full logs for session ABC.' Your agent handles the heavy lifting: it fetches everything from high-level aggregated telemetry on Selenium and Playwright builds to deep context within individual test sessions. You can pull absolute URLs for bug screenshots, retrieve full video recordings of browser executions, and extract command-level logs with exact timestamps. This capability is hosted on Vinkius, making sure any MCP-compatible client you use—whether it's Claude or Cursor—can access this crucial testing data. It's all about bringing your entire quality assurance pipeline into a natural conversation.

Core Capabilities

01 — Audit Build Status

Lists every test automation build and provides high-level telemetry for Selenium, Playwright, or Cypress suites.

02 — Inspect Test Sessions

Deep dives into a single browser run to extract console errors, network data, and metadata needed for debugging specific UI failures.

03 — Fetch Visual Artifacts

Retrieves absolute URLs for bug screenshots or full video recordings of any test execution.

04 — Get Command Logs

Extracts precise, command-level logs from the testing framework, including W3C WebDriver protocols and timestamps.

05 — Verify Platform Support

Queries the grid to list exactly which combinations of macOS, Windows, iOS, and Android versions are available for testing.

One Click on Vinkius — From Prompt to Execution

Available at **vinkius.com/mcp/testmu-ai-formerly-lambdatest** — connect your AI agent in three steps.

01 Subscribe to this MCP and enter your TestMu AI Username and Access Key.

02 Connect the service to your preferred agent (Claude, Cursor, etc.).

03 Tell your agent what you need—for example, 'Show me all build failures from last night'—and get the data returned instantly.

The bottom line is: it turns complex, multi-tab debugging into a simple conversation with your AI client.

Built For

This MCP is essential for QA Automation Engineers and Software Developers who spend too much time clicking through dashboards just to debug a single flaky UI issue. If you're tired of manually gathering logs, screenshots, and build metrics from multiple tabs at 2 AM, this tool gives your agent the keys.

QA Automation Engineer

Monitors build health across different CI/CD pipelines and inspects diagnostic logs through natural conversation without leaving their IDE or dashboard.

Software Developer

Retrieves session screenshots, video recordings, and network data to debug UI failures directly from their development workspace for rapid iteration cycles.

DevOps Team Lead

Audits CI/CD test results across various environments and monitors secure tunnel connections efficiently to verify deployment readiness.

What Changes When You Connect

01 Stop switching tabs. Instead of jumping to a dashboard to see build status, ask your agent to list all test automation builds using the `list_builds` tool. You get the high-level overview immediately.

02 No more guessing why something broke. If you find an issue, use `get_session` to pull deep context on that specific failure, grabbing console errors and network details without leaving your chat window.

03 Instant visual proof: Need to show a bug? Use `get_screenshots` or `get_video` to fetch the absolute URL for the exact screenshot or video recording of the failure. It's instant evidence.

04 Mastering regressions is easier with `get_build_sessions`. If a build fails, you can ask your agent to pull all related test sessions so you know exactly which part of the test suite needs fixing.

05 Verify compatibility before writing code. Use `list_platforms` to confirm that Windows 11/Chrome or macOS Sonoma/Safari is actually available in the grid. You'll never be surprised by missing capabilities again.

Real-World Applications

Diagnosing a Flaky Login Bug

A developer notices a login issue only on older Android versions. They ask their agent to check the platform support using `list_platforms`. Then, they use `get_session` on a failing session ID to get console errors and network configs, quickly pinpointing if it's an authentication timeout or a UI element failure.

Auditing Pre-Release Builds

A DevOps team member needs to verify the health of the latest release candidate. They ask their agent to `list_builds` and filter for 'Staging.' The agent returns aggregated telemetry, confirming that all critical Playwright suites passed before approval.

Tracking a Specific Regression

A QA engineer finds an unexpected failure in the staging environment. They use `get_build_sessions` tied to the build ID, narrowing down the issue to one specific session. Then they grab `get_video` and `get_screenshots` for that session to show their manager.

Checking Environment Scope

A team is planning a new feature requiring iOS 17 support. They ask the agent to check supported capabilities via `list_platforms`, confirming that the necessary OS/browser combination exists before committing development resources.

Patterns to Avoid

Treating it like a simple status check

✗ AVOID

Asking, 'Did the build pass?' and getting only a vague True/False answer that doesn't explain *why* or *what* failed.

✓ INSTEAD

Don't just ask for success. Ask your agent to `get_build` telemetry for the specific ID, focusing on the failure count and then asking it to find all associated sessions using `get_build_sessions`.

Only relying on screenshots

✗ AVOID

Getting a screenshot that shows an error but provides no underlying technical context (e.g., Was it network related or code related?).

✓ INSTEAD

Always pair the visual proof with logs. First, use `get_screenshots`, and then immediately follow up by asking for `get_session` to pull console errors and network data.

Forgetting platform scope

✗ AVOID

Writing code that passes on a local machine but fails in the CI/CD pipeline because it was testing against an outdated browser version.

✓ INSTEAD

Before starting, use `list_platforms` to confirm which OS and browser combinations are actively available. This prevents you from building tests for unsupported environments.

The Right Fit

Use this MCP when your debugging process involves gathering complex, multi-layered technical evidence—things like command logs, video recordings, or network artifacts. If the problem is 'the build failed,' this tool allows you to drill down using `get_build` and then use `get_session_logs` for the root cause. Don't use it if all you need is a simple list of active tunnels; that job is better suited for the dedicated `list_tunnels` function alone. Crucially, this MCP excels at taking data from disparate sources (screenshots, console errors, video) and presenting them in one conversation thread, which is something basic CI/CD dashboard integrations can't do.

The Mess of Manual Debugging Artifact Collection

Right now, when a test fails, you open the dashboard. You copy the session ID. Then you switch to the logs tab and download the command log file. If it's a UI bug, you click 'Screenshot,' then you have to manually check network calls in another browser tab. It’s five different tabs and three hours of context switching.

With this MCP, your agent handles all that clicking for you. You simply ask your AI client to find the failure evidence. It pulls the screenshot, the full video link, the console error, and the command logs—all summarized in one natural response.

Getting All Test Artifacts with TestMu AI

The manual steps that vanish are the copy-pasting of session IDs across tools, downloading zip files just to check a single error line, and manually cross-referencing platform capabilities against your requirements document.

Now, you speak natural language. You get structured data back from `list_builds` or the precise technical details needed for debugging using `get_session`. The whole process becomes conversational.

TestMu AI (formerly LambdaTest) with 10 Tools

These tools let your agent access the full spectrum of automated testing data, from high-level build summaries to individual command logs.

#	TOOL	DESCRIPTION
01	`list_builds`	Lists all completed automated test builds, grouping thousands of Selenium, Cypress, Playwright, or Appium tests into parent CI/CD build containers.
02	`get_build`	Retrieves detailed telemetry for a specific test automation build run.
03	`list_sessions`	Lists recent automated test sessions, providing the operating system, browser version, and final completion status.
04	`get_session`	Gets deep context for a specific test session by identifying console errors, network configurations, and relevant URL artifacts.
05	`get_session_logs`	Extracts precise command logs from the Selenium or Appium framework for a selected test run.
06	`get_screenshots`	Fetches visual bug screenshots that were captured during a specific test session's failure hook.
07	`get_video`	Retrieves the full video recording URL for any completed test execution.
08	`list_tunnels`	Lists all active secure Tunnels, useful for monitoring localhost testing environments.
09	`list_platforms`	Returns a complete list of every OS and browser combination that is currently supported in the testing grid.
10	`get_build_sessions`	Extracts all explicit test sessions associated with a specific build run, allowing isolation of regression analysis.

See It in Action

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

U List the last 5 automation builds in my TestMu AI account

I've retrieved the latest 5 builds. Highlights include 'Prod-Regression-987' (Passed), 'Staging-UI-Check-986' (Failed - 12 sessions), and 'Critical-Hotfix-985' (Passed). Would you like to see the test sessions for the failed staging build?

U Show me the logs and screenshots for session ID 'abc-123-xyz'

Retrieving data for session 'abc-123-xyz'... I've fetched 3 failure screenshots and the full Selenium command logs. The console shows a 'NoSuchElementException' at the login flow. Would you like the video recording link as well?

U Which OS and browser versions are currently supported in the grid?

I've queried the platform capabilities. Currently supported: Windows 11 (Chrome 120+, Firefox 118+), macOS Sonoma (Safari 17, Chrome 120+), and latest Android/iOS versions. I can provide the full JSON matrix if you need specific sub-versions.

Frequently Asked Questions

01 **How do I use TestMu AI to check if a specific browser version is supported?**

Use the `list_platforms` tool. This function returns an exact matrix of all OS and browser versions currently available in the testing grid, so you know what environments your code will actually run on.

02 **Can TestMu AI help me find command logs for a failed test?**

Yes. Use `get_session_logs` to extract precise Selenium or Appium command logs. This gives developers the low-level protocol data needed to understand exactly where the testing framework broke.

03 **What is the difference between listing builds and getting build telemetry with TestMu AI?**

The `list_builds` tool shows you a list of all completed runs. The `get_build` tool takes one specific ID from that list and returns highly detailed metrics for *that* single run.

04 **Does TestMu AI only provide screenshots, or can it give me videos?**

It provides both. You can use `get_screenshots` for immediate visual bugs, or you can use the `get_video` tool to retrieve a full video recording of the entire test execution.

05 **If I have multiple builds, how do I check all sessions?**

First, run `list_builds` to identify the build you care about. Then, use `get_build_sessions` with that specific build ID to pull every single test session associated with it for review.

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 `"testmu-ai-formerly-lambdatest": { "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

TestMu AI (formerly LambdaTest) 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 TestMu AI (formerly LambdaTest). 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	TestMu AI (formerly LambdaTest) MCP
Server ID	`019d75c3-e3ee-7276-aa86-11ef6528d979`
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/testmu-ai-formerly-lambdatest.