MCP SERVER | NO CODE | CLOUD HOSTED

Uber Eats MCP

Automate every step of your restaurant logistics workflow.

Uber Eats MCP handles all restaurant and delivery operations through your AI client. Use it to monitor incoming orders, update menus instantly, track couriers in real-time, and manage store status across multiple locations. It turns complex manual processes into simple natural language commands.

A+ Quality Score 98.33/100

order-management | menu-management | delivery-tracking | merchant-services | real-time-monitoring

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

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.

Uber Eats MCP

14 tools available
Cloud-hosted on Vinkius

You can connect your AI agent directly to the full Uber Eats marketplace API. This means you don't have to jump between apps or manually update statuses when things go wrong. You can ask it to monitor all pending orders, check if ingredients are running low by reviewing menu items, and then process them instantly.

Need to handle a large rush? Your agent checks incoming orders for capacity, accepts the ones you can manage, marks others as unavailable, and even notifies the customer with an accurate delay estimate. You can also view detailed order histories to address complaints or track down specific delivery issues. When managing multiple locations, this MCP gives you a central place to check store configurations and monitor everything from one chat window. It’s housed within Vinkius, giving your agent access to thousands of services so you only connect once.

Core Capabilities

01 — Monitor real-time order flow

View all current orders—pending, accepted, or rejected—to keep a live count of kitchen workload.

02 — Manage menu availability and pricing

Quickly toggle items as available or out of stock and update prices without logging into the merchant portal.

03 — Track courier movement and ETAs

Get live GPS coordinates for delivery couriers and accurate estimated times of arrival.

04 — Handle order lifecycle events

Move an order from pending to accepted, preparing, ready for pickup, or finally delivered using simple commands.

05 — Review detailed transaction history and issues

Pull up full customer details, special instructions, or review documented complaints and refund requests for any specific order.

06 — View multi-store operational data

Check the configuration, status, and unique identifiers for all your registered restaurant locations.

One Click on Vinkius — From Prompt to Execution

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

01 First, connect your Uber Eats merchant account to the MCP using an OAuth token.

02 Next, tell your AI agent what you need—for example, 'Check all pending orders and accept any that are simple.'

03 Finally, the agent executes the necessary API calls to update statuses, retrieve data, or modify menus on your behalf.

The bottom line is: it uses natural language commands to automate core restaurant logistics tasks across multiple systems.

Built For

This is for operations managers and multi-location owners who are tired of logging into separate portals just to manage a daily influx of orders. If your job involves rapidly changing menu prices or manually updating statuses during peak dinner rush, this MCP saves you time.

Restaurant Operations Manager

Uses the tool to monitor order queues and quickly adjust restaurant capacity by accepting or rejecting orders in real-time.

Multi-Location Franchise Owner

Checks store configuration details across all franchise units, ensuring every location has correct delivery settings before opening.

Ghost Kitchen Operator

Manages the menu and inventory by updating item availability immediately when specific ingredients run out.

What Changes When You Connect

01 Speed up order intake by letting the agent monitor incoming orders and automatically accepting them if kitchen capacity allows. You don't have to manually review each new ping when dinner service starts.

02 Maintain inventory accuracy by using `update_menu_item_availability` to toggle items as out-of-stock immediately, preventing customers from ordering unavailable goods.

03 Keep customers informed during delays. Use the agent to mark that food preparation has started (`mark_order_prep_started`) so delivery estimates stay accurate and people don't call in asking where your order is.

04 Streamline issue resolution by pulling up full details for any order using `get_order` . You can review special instructions or customer complaints without leaving the chat window.

05 Manage multiple physical locations from one spot. By calling `get_stores` , you get a master list of all store IDs needed to run menu updates or check operational status across your entire chain.

Real-World Applications

Handling the dinner rush surge

The manager sees 50 pending orders come in during peak hour. Instead of clicking through 50 screens, they ask their agent to review all and accept only those matching capacity limits. The agent uses `get_orders` followed by multiple calls to `accept_order`, keeping the kitchen flowing without human bottleneck.

Dealing with supply shortages

The chef runs out of a key topping mid-shift. Instead of calling someone to update the menu, they ask their agent to check the full catalog using `get_menus` and then immediately use `update_menu_item_availability` on that specific item.

Investigating an incorrect refund

A customer calls about a disputed charge. The owner asks their agent to pull up the order history using `get_order_issues`. The agent finds the timestamp of the complaint and the resolution status, giving them concrete data for the conversation.

Pre-opening checklist

A franchise manager needs to verify that all 12 store locations are active. They ask their agent to list all operational sites using `get_stores` and check each one's specific setup details with `get_store` before the day begins.

Patterns to Avoid

Trying to track a delivery without an ID

✗ AVOID

The user asks, 'Where is my food?' and expects a location update. The agent fails because it needs specific identifiers.

✓ INSTEAD

Always start by asking the agent to retrieve the order details using `get_order` first. Once you have the unique order ID, you can then use `get_delivery_status` to get real-time tracking.

Attempting mass status changes without filtering

✗ AVOID

The user asks, 'Accept all orders,' but doesn't specify which statuses are acceptable (e.g., only pending ones). The agent might try to process rejected or completed orders.

✓ INSTEAD

Use `get_orders` first, specifying the status filter you need, like PENDING. This narrows down the list so your agent can accurately use `accept_order` on only the relevant items.

Updating a menu item without knowing its location ID

✗ AVOID

The user tries to change an item's price but forgets which store it belongs to. The API call fails because it lacks scope.

✓ INSTEAD

You must first list all your sites using `get_stores`. This provides the necessary external IDs, which you then reference when making menu changes or checking specific store information with `get_store`.

The Right Fit

Use this MCP if your core business pain point involves real-time logistics management and high-volume transaction flow. Specifically, if you need to react instantly to incoming orders, change inventory status mid-shift, or track couriers live, this is essential. It's built for operational execution.

Don't use this MCP if your goal is deep financial analysis (e.g., calculating quarterly profit margins across 10 years). For that kind of historical data aggregation and complex modeling, a dedicated business intelligence tool or a custom database connection will be better suited. This MCP excels at the 'now'—the immediate workflow tasks like checking order status with `get_orders` or managing menu availability via `update_menu_item_availability`. It handles movement, not macroeconomics.

The Daily Manual Grind of Running a Restaurant

Today, when a rush hits, you're juggling multiple screens. You check the order portal for new pings, then switch to your inventory app to see if ingredients are available. Next, you jump into the scheduling tool to manually estimate delivery times and finally, you copy the customer's name from one system into another just to process a complaint.

With this MCP, those manual switches disappear. You tell your agent what needs doing—like 'We have five pending orders, but we are low on prep staff.' The agent handles the whole sequence: checking capacity, accepting only the viable ones, and marking everything else as unavailable automatically.

Uber Eats MCP: Full Order & Menu Control

You no longer have to manually update statuses or check item IDs. The agent can retrieve all necessary menu data using `get_menus` and then instantly use that information to toggle availability via `update_menu_item_availability`, solving the shortage problem in seconds.

What's different now is control. You move from reacting to alerts to proactively managing logistics, knowing your AI client has full operational oversight of every order stage.

Uber Eats: 14 Tools for Operations Management

These tools let your AI agent perform specific actions across the entire Uber Eats API, from accepting an order to checking a store's operational status.

#	TOOL	DESCRIPTION
01	accept_order	Accepts a pending order, confirming preparation is starting and triggering courier assignment.
02	cancel_order	Cancels an already accepted order for unavoidable reasons, requiring a specific reason code.
03	complete_order	Closes the order lifecycle after delivery is confirmed and final payment processing is triggered.
04	get_delivery_status	Retrieves real-time tracking information for a specific order, including courier location and estimated arrival time.
05	get_menus	Fetches the complete list of available items, prices, and IDs for a restaurant's menu structure.
06	get_order_issues	Pulls up records of customer complaints or reported problems linked to an order ID, along with resolution status.
07	get_order	Retrieves complete details for a single specific order, including special instructions and item breakdowns.
08	get_orders	Lists all orders across your restaurants, allowing filtering by status like PENDING or READY.
09	get_store	Gets detailed information and operational settings for one specific restaurant location.
10	get_stores	Lists all unique store IDs, names, and addresses associated with your entire merchant account.
11	mark_order_prep_started	Updates the order status to 'Preparing' and notifies the customer that food is being made.
12	mark_order_ready	Changes the order status, notifying couriers that the food is packaged and ready for immediate pickup.
13	reject_order	Rejects a pending order using specific codes (like 'too_busy') when you cannot fulfill it.

#	TOOL	DESCRIPTION
14	update_menu_item_availability	Toggles an item's availability status, marking it either in-stock or out-of-stock instantly.

See It in Action

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

U Show me all pending orders and accept them automatically

I'll check your pending orders and accept them right away.

U Update the price of 'Margherita Pizza' to R$45.90 and mark it as unavailable

I'll update the menu item price and toggle its availability.

U Track the delivery status of order #12345 and tell me where the courier is

I'll get real-time delivery tracking with courier location and ETA.

Frequently Asked Questions

01 **How do I know if an incoming Uber Eats order was accepted by the restaurant?**

You monitor the status using `get_orders`. The status changes from PENDING to ACCEPTED when your system successfully accepts the request.

02 **What is the difference between rejecting and canceling an order with Uber Eats MCP?**

Rejecting happens before acceptance (e.g., using `reject_order` if you are too busy). Canceling happens after acceptance, which usually carries a higher risk of penalties.

03 **Can I check the menu availability for all my locations with Uber Eats MCP?**

No, you must first list all your store IDs using `get_stores`, and then run `get_menus` separately for each location ID to get a full catalog.

04 **If I mark an order as ready, what happens next?**

Calling `mark_order_ready` triggers the system notification that tells couriers your food is packaged and waiting for pickup at your specified location.

05 **How do I check if a customer filed a complaint on an order?**

Use the `get_order_issues` tool. It pulls up records of complaints, timestamps, and whether a refund has already been issued for that specific transaction.

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 `"uber-eats": { "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

Uber Eats 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 Uber Eats. 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	Uber Eats MCP
Server ID	`019d7617-9a5b-73db-b3c3-a7ea56cf0da4`
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/uber-eats.