LYFTDATA PRODUCT DOCUMENTATION

AI and MCP

MCP server setup, security model, connection guidance, and AI-assisted workflow examples.

Version 2.1.0. Generated from the docs.lyftdata.com source corpus on 2026-05-25.

Contents

AI

Use assistants with LyftData through a client-hosted MCP server that stays aligned with your normal login, permissions, and live runtime state.

Source: ai/index.mdx

LyftData can work with desktop assistants and managed AI clients through the Model Context Protocol (MCP). This is part of the normal LyftData client experience, not a separate server-side AI product: you run an MCP server locally with `lyftdata mcp-server`, your assistant connects to that local process, and the client talks to the LyftData server you already use.

What you can do with AI in LyftData

- Monitor a live environment and summarize warnings, recent notifications, or worker health.
- Troubleshoot a slowdown or failed rollout by inspecting traces, logs, deployment state, and recent events.
- Author or review job definitions with the assistant grounded in the current LyftData surface.
- Keep separate assistant sessions for different environments or duties, such as production monitoring versus non-production authoring.

Trust, permissions, and writes

LyftData does not create a parallel security system for AI access. MCP sessions reuse the same server URL, login state, and permissions you already use elsewhere in LyftData.

- The default posture is read-first.
- Mutating tools stay unavailable unless you start the MCP server with `--allow-write`.
- Even in write-enabled mode, the session still operates inside the normal LyftData permission model.

Live state and operator visibility

Assistants can consume recent notifications and (where supported) websocket-backed updates for live signals. Operator visibility into assistant tool activity is also available, but the current behavior matters: activity forwarding is opt-in, metadata-only, and best-effort. It appears only when the MCP server is started with `--activity-forward`, and it is designed as a review aid rather than a permanent full transcript.

> note

MCP availability depends on the server feature set. In Community Edition, MCP is disabled unless a license enables the `mcp-server` feature.

> Note

> tip

Start with MCP Overview for the architecture and tool surface, then move to Security & Access or Configure & Connect for practical setup guidance.

> Note

How teams usually use it

Most teams end up with more than one assistant session:

- A read-only session that watches a production environment and answers operational questions.
- A separate session for authoring and testing against a safer environment.
- A tightly controlled write-enabled session used only when an operator wants help making a real change.

This separation is simple because the MCP server is part of the client: you can point different sessions at different LyftData servers, credentials, or profiles without inventing a second control plane.

Next pages

- MCP Overview
- Security & Access
- Configure & Connect
- Examples & Workflows

MCP Overview

Understand how the LyftData MCP server works, where it runs, and how sessions authenticate and stay read-first by default.

Source: ai/mcp/index.mdx

The Model Context Protocol (MCP) is the protocol that lets assistants call structured tools instead of guessing. In LyftData, MCP support is provided by a client-hosted MCP server started with `lyftdata mcp-server`.

The important architectural detail is where it runs: the MCP server is part of the LyftData client experience, not an always-on server-side component. Your assistant connects to the local MCP process, and that process connects to the LyftData server using the same API and auth path used by the CLI.

Setup and authentication

Most MCP sessions authenticate by reusing a successful CLI login. The MCP server looks for cached login state for the same target URL and profile you start it with.

The MCP server does not run an interactive login flow. If cached login state is missing or expired, re-run `lyftdata login …` with the same `--url` and `--profile` you plan to use for MCP (see CLI).

Recommended first-time setup:

```
export LYFTDATA_URL=https://server-host:3000/
lyftdata login admin lyftdata mcp-server --stdio
```

If you use profiles, the profile must match at login time and at MCP startup time:

```
lyftdata --profile prod --url https://prod.example login admin
lyftdata --profile prod --url https://prod.example mcp-server --stdio
```

Common “looks logged out” causes:

- The URL is different (`LYFTDATA_URL` / `--url` mismatch).
- The profile is different (`LYFTDATA_PROFILE` / `--profile` mismatch).
- The token in the cached login expired (re-run `lyftdata login …`).

For managed or headless runs, you can provide credentials explicitly:

- `--jwt "$JWT"` uses a bearer token directly.
- `--api-key "$API_KEY"` mints short-lived JWTs on demand (use `--ttl` to set desired token lifetime).

Transport options

- **Stdio** (`--stdio`) is the default and recommended transport for desktop assistants and local launchers.
- **HTTP** (`--http 127.0.0.1:8765`) is an advanced option for environments where the assistant cannot spawn a local stdio process. Treat it as a privileged network surface and keep it bound to loopback or a tightly controlled proxy.

Safety model

LyftData's MCP surface is designed to be read-first.

- Starting the server without `--allow-write` keeps mutation tools unavailable.
- Starting with `--allow-write` only unlocks write-capable tools for that session.
- Write-capable sessions still depend on the permissions granted to the underlying login or token.

That split is deliberate. It lets teams adopt AI-assisted inspection and troubleshooting before they decide whether any session should be allowed to change live state.

What you can do with MCP

The MCP surface is designed for practical operator and authoring workflows:

- Inspect jobs, workers, deployments, versions, and runtime signals.
- Pull recent notifications and logs for troubleshooting.
- Author, lint, and review job changes before you decide to stage or apply them.
- Perform controlled operational changes when you intentionally enable writes.

Trigger workflows with Triggers (dynamic tools)

LyftData also supports a Trigger registry: curated, named "run this" actions with schema-validated parameters. When a trigger is published for MCP, it appears as an additional tool in the MCP tool list. Tool names are derived from the trigger slug, for example `trigger_invoke__smoke_trigger`.

Calling a trigger tool dispatches a message to its target job. These tools are dynamic and per-identity: they show up in the tool list only when MCP publication is enabled and the authenticated identity is allowed to invoke them. Because this is a real runtime action, trigger tools require a write-enabled MCP session (`--allow-write`).

To track completion, call `trigger_invocation_get` with the `invocation_id` returned by the trigger tool.

> note

Trigger tools accept the trigger's parameter schema plus two reserved control keys: `_idempotency_key` and `_wait_timeout_seconds`.

> Note

Refreshing dynamic tools

Most MCP clients cache the tool list per session. If you publish/revise Triggers (or change MCP publication/policy) and do not see updated `trigger_invoke__*` tools, refresh the tool list by reconnecting the MCP client. If your MCP client does not expose a refresh action, restart the `lyftdata mcp-server` process.

Reserved key mapping

Reserved MCP arguments map directly to the invoke API fields:

MCP tool arg	Invoke request field	Purpose
`_idempotency_key`	`idempotency_key`	Dedupe retries into one invocation record.
`_wait_timeout_seconds`	`wait_timeout_seconds`	Bound how long the invoke call waits before returning the latest invocation state.

For the dispatch payload shape, response-slot result envelopes, and proxy capture behavior, see Triggers.

> tip

Use Configure & Connect for launch commands and troubleshooting, and Examples & Workflows for tool-driven sequences.

> Note

Continue with Security & Access for the permission model and observability options.

Configure and Connect

Launch LyftData MCP with the correct transport, auth source, and write mode, then verify the session and troubleshoot common failures.

Source: ai/mcp/configure-and-connect.mdx

This page is the operator runbook for starting a LyftData MCP session.

Preflight checklist

1. Verify the target URL and auth path with the normal CLI first (and run `lyftdata login …` if you intend to use cached login; MCP will not prompt interactively). 2. Confirm the target server license enables `mcp-server`. 3. Decide whether the session will use cached login, `--jwt`, or `--api-key`. 4. Decide whether the session must stay read-only or use `--allow-write`. 5. Use `--stdio` unless you have a specific operational reason to expose HTTP transport.

Common launch patterns

Read-only stdio with cached login

`--stdio` is the default if no other transport is selected, but keeping it explicit makes launcher configuration easier to audit.

```
lyftdata --url https://your-lyftdata.example \
mcp-server --stdio
```

Read-only stdio with a named profile

```
lyftdata --profile prod --url https://prod.example \
mcp-server --stdio
```

Headless stdio with an explicit JWT

```
lyftdata --url https://your-lyftdata.example \
mcp-server --stdio --jwt "$JWT"
```

Headless stdio with API-key-based JWT minting

```
lyftdata --url https://your-lyftdata.example \
mcp-server --stdio --api-key "$API_KEY" --ttl 3600
```

Use this mode when the session will run long enough that in-process JWT refresh is useful.

Write-enabled stdio with activity forwarding

```
lyftdata --profile ops --url https://prod.example \
mcp-server --stdio --allow-write \ --activity-forward \ --activity-session-label ops-change-
window
```

HTTP transport on loopback

```
lyftdata --profile staging --url https://staging.example \
mcp-server --http 127.0.0.1:8765
```

Use HTTP only when the MCP client cannot spawn a local stdio process. The build must include `mcp-http`, and the listener should stay on loopback or behind a tightly controlled proxy.

Verify the session

- Ask the MCP client to list tools and confirm the expected read-only or write-capable surface.
- Call `mcp_server_metrics` and confirm `transport`, `allow_write`, and the reported timeout and concurrency settings.
- If you need push notifications, call `mcp_notifications_transport_get` and confirm that the returned websocket and recent-notification endpoints are reachable from the client.

Troubleshooting

`mcp-server` exits immediately

Common causes:

- The target server does not enable the `mcp-server` license feature.
- The client build does not include MCP support.
- `--http` was requested, but the build does not include HTTP transport support.

The session hits the wrong environment

Check `--url` and `--profile` first. Cached logins are tied to the selected environment, so a wrong URL or profile often looks like an auth failure.

Authentication fails

Common causes:

- No cached login for the selected environment.
- Expired or invalid JWT.
- Invalid API key for JWT minting.
- TLS trust mismatch against a private certificate.

For self-signed or private PKI environments, use the same TLS settings you would use with the normal CLI. Prefer fixing trust configuration over weakening TLS checks.

Write tools are missing or rejected

The session was started without `--allow-write`, or the underlying identity lacks permission for the requested mutation.

Activity feed is empty

Check whether the MCP process was started with `--activity-forward`. Activity forwarding is opt-in and best-effort.

Push notifications are unavailable

Call `mcp_notifications_transport_get` first. If the client cannot open the returned websocket endpoint, fall back to `job_notifications_recent`. Remember that `/api/mcp/notifications/*` is admin-scoped.

Examples and Workflows

Concrete MCP tool sequences for inspection, troubleshooting, authoring, and controlled writes against LyftData.

Source: ai/mcp/examples-and-workflows.mdx

This page focuses on tool sequences rather than prompt phrasing. The MCP surface is most useful when the client chains a small number of LyftData tools against the correct environment.

Read-only inventory patterns

Goal	Tool sequence	Result
List configured jobs	`jobs_list` -> `job_show_rendered`	Inventory the job set and inspect rendered job definitions
Find where a job runs	`workers_with_job` -> `worker_connectivity_status_get`	Identify attached workers and current connectivity
Inspect recent runtime evidence	`workers_logs_recent` -> `job_notifications_recent`	Pull logs and notification summaries without mutating anything
Check MCP process health	`mcp_server_metrics`	Confirm transport, write mode, timeout counts, and concurrency counters

Rollout or slowdown investigation

Use a read-only session for the entire sequence:

1. Call `deployments_list` or `deployment_get` to identify the deployment under discussion. 2. Call `deployment_status_get` for current desired vs observed state. 3. Call `deployment_diff_since_last_apply` to see what changed recently. 4. Call `deployment_signals_list` and `deployment_lifecycle_warnings_get` for current warnings and signal state. 5. Call `workers_logs_recent` and `job_notifications_recent` for recent evidence. 6. If the question is "did the slowdown start with the latest release?", call `job_slowdown_release_correlation`.

This sequence avoids mutation while still grounding the answer in deployment state, worker evidence, and notification history.

Job authoring sequence

Run this against a non-production environment or an isolated authoring profile:

1. Use `spec_search` and `spec_item_get` to inspect the current DSL surface. 2. Use `job_template_list` and `job_template_get` to find a starting template. 3. Use `job_auto_detect`, `job_auto_detect_sample`, or `job_auto_detect_file` when the first draft should be inferred from sample data. 4. Use `job_yaml_preview` to render the draft output. 5. Use `job_lint` to validate the job before any staging or deployment step. 6. Use `job_show_rendered` to inspect the final rendered configuration the server will evaluate.

Controlled write sequence

Split analysis and mutation into separate processes:

1. In a read-only session, inspect the target with `job_show_rendered`, `deployment_status_get`, and any relevant notification or log tools. 2. In a separate `--allow-write` session, perform the approved mutation with the narrowest tool possible, such as `job_stage`, `job_deploy`, `deployment_action_enqueue`, or `deployment_apply_with_refresh`. 3. Re-run `deployment_status_get`, `deployment_signals_list`, and `job_notifications_recent` immediately after the change. 4. If activity forwarding is enabled, use the UI activity feed to review the recent tool metadata for that write-enabled session.

Trigger registry (dynamic tools)

If your environment has published Triggers with MCP enabled, your MCP client will discover additional tools named `trigger_invoke__*`. These calls dispatch real runtime messages, so they require a write-enabled MCP session.

1. Start MCP with `--allow-write` and refresh the tool list in your assistant. 2. Invoke a trigger tool, for example `trigger_invoke__smoke_trigger`, with its schema-defined parameters. 3. Use `trigger_invocation_get` with the returned `invocation_id` to poll until you see a terminal state.

Session split pattern

Session	Typical flags	Typical tools
Production read-only	`--profile prod --stdio`	`deployments_list`, `deployment_status_get`, `workers_logs_recent`, `job_notifications_recent`
Staging authoring	`--profile staging --stdio`	`spec_search`, `job_template_list`, `job_yaml_preview`, `job_lint`
Controlled change window	`--profile ops --stdio --allow-write --activity-forward --activity-session-label ops-change-window`	`job_stage`, `job_deploy`, `deployment_action_enqueue`, `deployment_apply_with_refresh`

This split keeps production inspection, authoring, and live mutation in separate trust domains even when all three sessions run on the same machine.

Security and Access

Authentication, authorization, write gating, activity forwarding, and telemetry for LyftData MCP sessions.

Source: ai/mcp/security-and-access.mdx

LyftData MCP does not introduce a second auth or RBAC system. The session inherits identity and permissions from the LyftData client process that hosts it.

Authentication sources

Auth source	Flags	Best fit	Notes
Cached login	none	Desktop or operator-driven sessions	Reuses the token from `lyftdata login` for the selected `--url` and `--profile`
Explicit JWT	`--jwt "$JWT"`	Managed runs where another system owns token issuance	The MCP process uses the bearer token directly
API-key-minted JWT	`--api-key "$API_KEY"` and optional `--ttl`	Long-running headless sessions	The process mints short-lived JWTs from the API key and refreshes them before expiry

Authorization boundary

- Server-side permissions remain authoritative. If the underlying identity cannot read or mutate a resource through LyftData, the MCP session cannot do it either.
- Admin-scoped tools and endpoints still require admin credentials. This includes MCP notification transport discovery for `/api/mcp/notifications/*`.
- Changing `--url` or `--profile` changes the trust boundary because cached logins are keyed to the selected environment and profile.

Write gating

- `--allow-write` is a local session switch that exposes mutating tools for that process.
- The flag does not elevate the underlying server identity.
- Without the flag, write-capable tools fail fast instead of silently mutating state.

> note

Keep read-only and write-capable sessions as separate processes. That gives you a clean process boundary, cleaner audit context, and less risk of using the wrong session for a live change.

> Note

Operator visibility

Activity forwarding

- `--activity-forward` publishes metadata-only tool activity to the LyftData server for operator review.
- `--activity-session-label` adds a human-readable label such as `dev-laptop`, `ci`, or `prod-watch`.
- Forwarding is best-effort and non-blocking. Tool execution continues even if activity publishing fails.
- Full request and response bodies are not forwarded.

Notifications and live state

- `job_notifications_recent` is the polling-friendly path for assistants that cannot hold a websocket side channel.
- `mcp_notifications_transport_get` returns the MCP notification endpoints, including `/api/mcp/notifications/recent` and `/api/mcp/notifications/subscribe`.
- The `/api/mcp/notifications/*` endpoints are admin-scoped.

Process telemetry

- `--otel-tracing` enables OpenTelemetry export for the MCP process.
- `--otel-service-name` overrides the default service name, `lyftdata-mcp`.
- `mcp_server_metrics` is the quickest in-band check for process health, write mode, error counts, and timeout counts.

Practical posture

- Use cached login for normal operator desktop sessions.
- Use `--jwt` or `--api-key` when token ownership and rotation need to be explicit.
- Keep routine monitoring and diagnostics read-only.
- Enable `--allow-write` only for tightly scoped operational sessions.
- Enable activity forwarding deliberately and document the session-label convention in your runbook.