LYFTDATA PRODUCT DOCUMENTATION

Get Started and Install

Evaluation path, core concepts, downloads, installation, configuration, workers, and upgrade basics.

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

Contents

Overview

Plan your first LyftData setup from trial to production

Source: get-started/overview.md

LyftData is a unified data operations platform that lets teams connect sources, transform records, and deliver results without stitching together bespoke scripts. Describe the pipeline once and the platform owns scheduling, retries, and observability.

Quick start checklist

- [ ] Confirm prerequisites – Review system requirements to make sure the target machine has enough CPU, RAM, and disk headroom.
- [ ] Download the release bundle – Grab the latest binaries from Downloads; Community Edition runs without a license, but you can review licensing upgrades if you plan a production rollout. If you just need a trial, jump straight to the Evaluation Quickstart.
- [ ] Bring up the control plane – Follow the platform-specific guide (e.g., Linux server install) to start the server; the built-in worker comes online automatically in Community Edition. Evaluators can complete the single-host setup in ~15 minutes using the quickstart.
- [ ] Ship your first job – Use Getting Started for a guided tour of the UI, then complete Running a Job to validate end-to-end execution.
- [ ] Harden for production – Consult the Install & configure section for TLS, backups, and multi-worker runbooks before touching real datasets.

Each step above links to a short walkthrough so you can stay in-context without hunting across the docs.

Why teams choose LyftData

- **Connect data fast**: pull from databases, SaaS APIs, files, or object stores using built-in connectors.
- **Shape pipelines safely**: mix built-in actions (filters, enrichers, preprocessors, etc.) with Run & Trace so you can verify every change.
- **Operate with confidence**: centralized logging, metrics, and alerting keep operations predictable even as workloads grow.

Architecture basics (for trials)

In an evaluation setup, you can run everything on a single machine:

~~~text Browser / CLI ──► Server (UI + API + scheduling) │ └─► Built-in worker ──► Sources / Destinations ~~~

- The **server** is the control plane: UI + API, job definitions, scheduling, and deployment coordination.
- The **worker** runs the pipelines: it connects to sources, transforms events, and writes to destinations.
~~~

- In **Community Edition**, the built-in worker runs alongside the server by default, so you don't need extra hosts to validate end-to-end execution.

Example pipeline

~~~yaml name: analytics-sync input: http-poll: url: https://api.example.com/events json: true response: status-field: http_status response-field: body trigger: interval: duration: 1h actions:

- filter:

how: expression: http_status == 200

- add:

output-fields: exported_at: "${time|now_time_iso}" output: s3: bucket-name: lyftdata-exports object-name: name: "analytics/${time|now_time_fmt %Y/%m/%d}/events" guid-prefix: "-" guid-suffix: ".jsonl" ~~~ This job polls an API hourly, filters successful responses, stamps each record, and persists to S3. Chain additional jobs through Worker channels when you need fan-out, enrichment, or multi-step processing.

Your first five-minute pipeline

- **Prerequisites** – Confirm the server is running and the built-in worker shows **Online** using the Getting Started checklist. You do not need a license while you remain on Community Edition.
- **Walkthrough** – Follow the First pipeline guide for the full download, install, and deploy flow.
- **What you'll build** – Create a simple pipeline in the visual editor, inspect it with **Run & Trace**, then deploy it to your worker.

Want the UI-only tour? Jump to Running a Job.

Preparing for production

- **Observability baseline** – Turn on metrics shipping and review the Troubleshooting guide so you know what "healthy" looks like before launch.
- **Scaling playbooks** – Plan how many workers you need once you upgrade beyond Community Edition, and decide whether they live near sources or destinations. The advanced scheduling guide covers common patterns.
- **Security review** – Walk through TLS, RBAC, and secret storage checklists in the Install section prior to onboarding real customers.

Where to go next

- **Install & configure** – Deep dives on system requirements, TLS, and deployment patterns live in the Install overview.
- **Build resilient jobs** – The Build overview and advanced scheduling pages walk through orchestration patterns such as fan-out and retries.
- **Connect integrations** – Start with the Integrations catalog for per-source quick starts.
- **Operate at scale** – Start with the Operate & Scale overview, then use Monitoring, Backup & Recovery, and Security Hardening.
~~~

- **Work as a team** – Share the Team benefits guide with data engineers, operators, and reviewers to align on workflows.

With these foundations in place, you can iterate quickly on pipelines while the platform handles orchestration, scaling, and observability.

Installation Overview

Choose an installation path for evaluation or production

Source: install/overview.mdx

This section covers the installation of LyftData. Start by deciding which journey you are on, then follow the matching checklist below.

1. For a single-host trial, follow the Evaluation Quickstart 2. Prefer containers? Use Docker and Docker Compose instead of host installs 2. Confirm System Requirements and Prerequisites 3. Download the binary via Downloads or Prepare the binary 4. Install the server on your target OS and verify it is reachable 5. (Optional) add external workers, then review Configuration basics and Networking & TLS

Choose your path

- **Evaluation (single host, ~15 minutes):** Ideal for proof-of-concept or demos. Use the Evaluation Quickstart to bring up a server with the built-in worker, log in, and run a sample job.
- **Production rollout:** Plan for dedicated server and worker hosts, service accounts, and network approvals. Start with Prerequisites and follow the platform guides for every host. Budget ~45–60 minutes per host for initial provisioning (service account, binary placement, service manager wiring, validation).

Day-zero admin setup

The first server start no longer prints a generated admin password. Choose the path that matches how you are running LyftData:

Modality	Recommended first-run path	Why
Local terminal on the same host	Omit `--admin-init-password`, keep the server on `127.0.0.1`, and use the one-time setup URL shown in the ready splash or written to `<staging-dir>/bootstrap/initial-admin.url`.	The token stays local to the host and the browser flow is the simplest.
`systemd`, `launchd`, or Windows service	Prefer `LYFTDATA_ADMIN_INIT_PASSWORD` for repeatable provisioning, or run `lyftdata server create-admin --staging-dir ...` locally on the host.	Service logs do not include the setup token, so headless installs are easier to automate with an explicit password or CLI creation.
Containers / Compose	Prefer `LYFTDATA_ADMIN_INIT_PASSWORD` or pre-create the admin with `lyftdata server create-admin` against the mounted staging volume.	Container stacks usually listen on non-loopback addresses; remote bootstrap is blocked by default until an admin exists.

If you need to refresh a local setup link, use `lyftdata server bootstrap --staging-dir ... --bind-address ... --print-url`. See Configuration basics for the file, command, and URL-based options in one place.

Minimal deployments

lyftdata_architecture: ../../../assets/lyftdata-highlevel-simple.svg

A minimal LyftData installation consists of a single server process. The server provides a built-in worker, which is used for testing jobs, but it can also run as many jobs as the server's host resources can accommodate.

Multi-worker deployments

lyftdata_architecture: ../../../assets/lyftdata-highlevel-architecture-no-annotations.svg

LyftData scales by deploying additional workers, and connecting them to a server. For multi-worker deployments, all jobs are managed through the server.

Workers typically run on separate hosts, but they can also run on the same host as the server, provided default worker ports are appropriately configured to avoid conflicts.

After bringing up the server and initial workers, follow the post-install checklist to enable metrics, verify health, and build your first job.

> note

Server and worker processes can run from the same binary.

> Note

Where to go next

- Review the installation prerequisites to confirm each host is ready.
- Follow the platform guide that matches your target OS: Linux, macOS, or Windows.
- Read Workers if you plan to add external workers or scale out.
- Review Networking & TLS before exposing the UI or connecting remote workers.
- After the service is running, complete the post-install checklist and move into the Day 1 production guide.

Core Concepts

LyftData Core Concepts

Source: get-started/core-concepts.md

Understanding how LyftData models control, orchestration, and execution helps you design pipelines that scale cleanly from a local experiment to production workloads. The three pillars are the Server (control plane), Jobs (pipeline definitions), and Workers (runtime execution).

Server: control plane

The server is LyftData's source of truth. It stores configuration, maintains system health, and orchestrates work across workers. Because the Server also fronts the web UI and API, it is the hub for both day-one configuration and day-two operations.

Responsibilities

- Persist and version Job definitions and associated assets
- Schedule work by matching Jobs to eligible Workers and enforcing concurrency policies
- Aggregate metrics, logs, and alerts so operators have a single pane of glass

Jobs: pipeline units

A Job describes how one stream of data moves from an input through optional processing steps to a single output. Jobs can be authored as YAML, managed through the visual editor, or generated through automation.

Lifecycle 1. Define the input (files, APIs, queues, databases, and more) and the trigger that determines when it should run 2. Add Actions—filters, parsers, enrichers, or Lua scripts—to shape the payload 3. Select a single output destination (warehouse, lake, search index, message queue, etc.) 4. Stage and deploy; the Server handles scheduling, retries, and observability

Composing flows

A single Job always links one input to one output, but you can compose multi-step or branching flows by wiring Jobs together with Worker channels. Channels are in-memory pathways that let one Job's output feed another Job's input without external brokers.

~~~text Job: ingest-events -> Channel: ingest-feed Job: enrich-events <- Channel: ingest-feed -> Channel: analytics-fanout Job: load-warehouse <- Channel: analytics-fanout Job: push-alerts <- Channel: analytics-fanout ~~~

This pattern keeps each pipeline step isolated while enabling fan-out and enrichment. For fan-in, point multiple Jobs to the same channel and downstream processor. See Build overview and advanced scheduling for orchestration patterns, back-pressure controls, and retry strategies.

Workers: execution layer
~~~

Workers are stateless runtimes that pull assignments from the Server, materialize Job pipelines, and emit telemetry. You can run Workers next to data sources for low-latency ingestion or in a central plane for simplified networking.

How Workers operate 1. Register with the Server and advertise capabilities (connectors, capacity, and optional labels) 2. Receive Job payloads and instantiate the runtime (inputs, actions, outputs, triggers) 3. Execute the pipeline, streaming metrics, logs, and traces back to the Server 4. Tear down gracefully, reporting status so the Server can reschedule or retry

Worker isolation limits blast radius: a failure in one Worker typically impacts only the Jobs it owns. Additional Workers can be added at any time for horizontal scale or to segment workloads (e.g., by region or data sensitivity).

Scaling and operational guidance

- **Capacity planning** – Establish a baseline by tracking throughput, concurrency, and memory per Job. Add Workers when queue depth or execution latency grows faster than expected.
- **Observability** – Forward logs and metrics from Workers to your monitoring stack. Familiarize yourself with the Troubleshooting guide and set alerts on retry spikes or sustained channel backlogs.
- **Resilience** – Use staging and canary deployments before promoting configuration changes. Pair Worker channels with retries to confine failures to specific pipeline stages.
- **Security** – Apply TLS, RBAC, and secret management practices outlined in the Install runbooks before attaching production datasets.

Configuration and management

Most teams iterate through the Server UI during development, then promote Jobs via source control and CI. Environment variables and configuration files govern cluster-wide behavior (licensing, telemetry sinks, connector credentials). For deeper reference, explore the Configuration docs and topic-specific guides across Install, Build, and Operate sections.

By internalizing these concepts—Server authority, Job composition, Worker execution, and channel-driven orchestration—you can assemble reliable pipelines without re-implementing scheduling, scaling, or observability from scratch.

Where to go next

- Start building with the First pipeline guide to apply these concepts immediately.
- Dive into pipeline design in the Build overview.
- Orchestrate multi-job systems with Deployments (workflows + Deployment Manager).
- Compare licensing paths in Community Edition vs licensed deployments.

First pipeline guide

Evaluate LyftData, install the server, and run your first job in under an hour

Source: get-started/day-zero.mdx

Use this guided path when you want to evaluate LyftData or onboard a new teammate. It bundles the essentials—download, install, and deploy—so you can experience the full product without exploring the entire reference section. In Community Edition the server ships ready to use: no license is required and the built-in worker comes online automatically. If you only have 15 minutes and a single host, start with the Evaluation Quickstart and return here when you are ready for the full install.

> note

This is the canonical quick start. The **Overview**, **Getting Started**, and **Running a Job** pages summarize this flow and point back here when you only need headlines.

> Note

1. Check your environment (5 minutes)

> tip

Confirm you have a host with the required CPU/RAM and an OS from the supported list.

> Note

- Review System requirements to confirm hardware, OS version, network access, and ports.
- If you expect to upgrade later, decide whether future workers live on separate hosts so you can plan firewall and directory layouts now.

2. Download the binary (5 minutes)

- Visit the Downloads page and pull the archive that matches your platform.
- Optional: verify the checksum before extracting.
- Keep the binary accessible; it powers both the server and workers.

3. Start the server (10 minutes)

Follow the quick start so the control plane is online:

1. Walk through Getting Started to launch the server and complete initial setup with the one-time setup link. 2. Confirm the built-in worker shows **Online** in the dashboard. (Additional workers require a license upgrade and are optional for later.) 3. Run `curl http://localhost:3000/api/workers | jq '.[].status'` to verify the API reports `"online"` for the built-in worker.

> tip

The verification command uses `jq`. If it is not installed, install it (for example `brew install jq` or `apt install jq`) or drop the `| jq '.[].status'` portion and inspect the raw JSON from `curl`.

> Note

4. Run your first job (15 minutes)

Use the visual editor to create, test, and stage a job:

- Follow Running a Job to build the default echo → time → print pipeline.
- Experiment with **Run & Trace** to see how events move through each action.
- Stage the job and deploy it to the built-in worker.

When you are done, check the live run history—this proves the pipeline deployed successfully.

5. Explore what's next (10 minutes)

Once the basics work, branch out based on your goal:

- **Install & configure**: If you need a production-ready service, jump into the Install overview and the OS-specific guides.
- **Build pipelines**: Learn how to connect real systems by reading the Build overview and trying the S3 integration example.
- **Operate & scale**: Prepare for day two with Monitoring and the Scaling playbook.

Checklist recap

- [ ] Environment verified against requirements
- [ ] Binary downloaded and extracted
- [ ] Server running with admin access
- [ ] Built-in worker reporting online
- [ ] First job staged, deployed, and producing output

Keep this list handy when onboarding new teammates or spinning up a fresh environment—it covers the minimum viable path to demonstrate LyftData end to end.

Team Benefits

How LyftData helps pipeline engineers, operators, and security teams collaborate

Source: get-started/team-benefits.mdx

LyftData gives data engineers, operators, and security reviewers a shared control plane for building and governing pipelines. This page outlines the core workflows each role gains out of the box so you can align internal processes with the platform from day one.

Quick highlights

Role	Benefits	Features to lean on
Pipeline architects & developers	Design, test, and iterate quickly without YAML gymnastics	Visual editor, Run & Trace, built-in actions, job staging
Platform admins / SREs	Centrally manage workers, upgrades, and observability	Dashboard, worker inventory, deployment history, API/CLI
Security & compliance	Audit changes, enforce controls, and review sensitive connectors	Staged jobs, context separation, license & worker limits, logs

For pipeline architects and developers

- **Visual-first authoring** – Build jobs in the UI and keep the configuration synced automatically; switch to raw mode only when needed.
- **Versioned staging** – Every stage freezes a deployment-ready snapshot, making review and rollback straightforward.
- **Safe experimentation** – Use **Run & Trace** to inspect every action in-line before you deploy, and transient runs stop automatically.
- **Reusable building blocks** – Built-in actions (filters, enrichers, preprocessors, validators, etc.) cover the majority of transformation patterns without scripting.
- **Context-driven configuration** – Inject secrets or environment-specific overrides without hard-coding them into job definitions.

Suggested workflow

- [ ] Sketch the pipeline in the visual editor; save a meaningful job name as soon as you create it.
- [ ] Use sample data with **Run & Trace** until every action produces the expected output.
- [ ] Stage the job and request review before deploying to production workers.
- [ ] Document owner, SLA, and inputs/outputs in the job notes or repository README.

For platform admins and SREs

- **Central control plane** – Monitor worker health, queue depth, and recent jobs from one dashboard, and drill down as needed.
- **Predictable deployments** – Staging and deployment history record who shipped what, when, and to which worker.
- **API & CLI coverage** – Automate routine checks (, ,) or integrate with existing tooling.

- **Scalable topology** – Add or remove workers as throughput needs change; isolate workloads by labels or regions when you upgrade past Community Edition.
- **Operational guardrails** – Built-in worker limits, retention settings, and capture pipelines minimize blast radius during incidents.

Operating checklist

- [ ] Review dashboard metrics daily (worker status, queue depth, alert feed).
- [ ] Capture deployment snapshots before major changes ().
- [ ] Track worker additions/removals in your runbook; keep labels consistent.
- [ ] Automate health checks via API or cron so outages are detected quickly.

For security and compliance teams

- **Immutable staging records** – Every staged job snapshot can be referenced during audits; diffs show exactly what changed.
- **Scoped access** – Keep production credentials in context values or secret stores; Community Edition limits outbound calls so the footprint is small.
- **Worker governance** – Only licensed deployments can add external workers, giving you a natural control point for vetting infrastructure.
- **Comprehensive logging** – Server logs and job history record who deployed what; pair with SIEM ingestion for long-term retention.
- **API visibility** – License endpoints () expose current mode so policy engines can assert the right controls.

Review tips

- [ ] Require staging approval before any job reaches production workers.
- [ ] Audit context keys regularly to ensure secrets remain scoped to the right environments.
- [ ] Monitor license state and worker inventory; unexpected changes usually indicate configuration drift.
- [ ] Subscribe to deployment or job failure alerts so you can triage sensitive connectors quickly.

Where to go next

- Walk through the First pipeline guide to see these collaboration points in action.
- Read the Build overview for deeper guidance on actions, context management, and deployment automation.
- Explore Community Edition vs licensed deployments to plan when to introduce additional workers or throughput.

Configuration Basics

Where settings live and how to change them safely

Source: install/configuration-basics.mdx

LyftData can be configured in three ways. Most deployments use all three: CLI flags for ad-hoc runs, environment variables for services, and UI settings for day-two operations.

Configuration surfaces

- **CLI flags**: Use `lyftdata run server --help` and `lyftdata run worker --help` (or `lyftdata-worker --help` on worker-only hosts) to discover flags and defaults. Flags are ideal for local evaluation because they are explicit and easy to copy/paste.
- **Environment variables**: Service managers typically set `LYFTDATA_*` variables so the service starts consistently on boot. Use the patterns shown in the OS install guides (Linux `/etc/default/*`, macOS launchd plist `EnvironmentVariables`, Windows machine-scoped env vars).
- **UI settings**: Some runtime settings (for example per-worker metrics) are managed from the web UI after the worker is registered. Start with Next steps for the common post-install toggles.

Day-zero admin options

LyftData no longer prints a generated admin password on first start. Instead, choose the bootstrap path that matches your deployment style:

You are doing this	Recommended path	Notes
Local interactive run on a laptop, workstation, or VM	Use the setup URL flow	Omit `--admin-init-password`, keep the server on `127.0.0.1`, then open the setup URL shown in the ready splash or stored in `<staging-dir>/bootstrap/initial-admin.url`.
Service install on a host you can log into	Use `LYFTDATA_ADMIN_INIT_PASSWORD` or `lyftdata server create-admin`	This is the most predictable path for `systemd`, `launchd`, and Windows services because logs do not include the token.
Container / Compose deployment	Use `LYFTDATA_ADMIN_INIT_PASSWORD` or pre-create the admin with `lyftdata server create-admin` against the mounted staging volume	Published container ports are typically non-loopback. Remote bootstrap is blocked by default until an admin exists.

Useful local commands:

- `lyftdata server bootstrap --staging-dir ... --bind-address ... --print-url` refreshes the setup token and rewrites the local `initial-admin.url` file.
- `lyftdata server create-admin --staging-dir ...` creates the first admin directly without using the browser setup link.

Where LyftData stores state

Plan your data directories early, especially for production:

- **Server staging directory**: `--staging-dir` / `LYFTDATA_STAGING_DIR` is the server's primary data directory. It stores job definitions and operational data needed to run the control plane.
- **Worker jobs directory**: `LYFTDATA_JOBS_DIR` is the worker's local directory for cached job definitions and worker state.
- **Service user home directory**: Some platforms may also write small runtime files under the service user's home directory. Ensure the service user home is writable and managed like application state.

For host preparation patterns (service accounts, directory ownership, permissions), see Prerequisites.

Recommended defaults (evaluation)

For a trial eval on a single host:

- Keep the server bound to localhost (`127.0.0.1:3000`).
- Set `LYFTDATA_LICENSE_EULA_ACCEPT=yes` for the server process so automation does not stall on the one-time prompt.
- Use the setup URL flow unless you specifically need a shared/headless password-based bootstrap.

Where to go next

- Complete configuration reference: Configuration
- Connectivity and certificates: Networking & TLS
- Common failures: Troubleshooting

Prepare The Binary

Download, verify, and place the LyftData executable

Source: install/download-and-install.mdx

Every deployment starts with a LyftData executable:

- `lyftdata` (server/CLI)
- `lyftdata-worker` (worker-only; optional for external worker hosts)

> note

Complete the installation prerequisites for each host before downloading the binary.

> Note

For evaluation and single-host installs, `lyftdata` is sufficient (server + built-in worker). For dedicated external worker hosts, you can deploy `lyftdata-worker` instead.

> note

The examples below use the server/CLI binary (`lyftdata`) and the current stable release path (`release/latest`). For worker-only installs, replace `lyftdata-…` with `lyftdata-worker-…` in the download URLs and use `lyftdata-worker --version` when verifying. If you need a beta build or another artifact, start from the Downloads page.

> Note

Linux (systemd hosts)

1. **Download the archive**

```
   mkdir -p /opt/lyftdata
cd /opt/lyftdata curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-
unknown-linux-gnu.tar.xz
```

2. **(Optional) Verify the checksum**

```
   curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-unknown-linux-
gnu.tar.xz.sha256
sha256sum -c lyftdata-x86_64-unknown-linux-gnu.tar.xz.sha256
```

3. **Extract the archive and make it executable**

```
   tar -xJf lyftdata-x86_64-unknown-linux-gnu.tar.xz
cp lyftdata-x86_64-unknown-linux-gnu/lyftdata ./lyftdata chmod +x ./lyftdata
```

If you prefer a global location, create a symlink:

```
sudo ln -s /opt/lyftdata/lyftdata /usr/sbin/lyftdata
```

4. **Confirm the version**

```
./lyftdata --version
```

macOS (launchd hosts)

1. **Download the archive**

```
mkdir -p /usr/local/lib/lyftdata
cd /usr/local/lib/lyftdata curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-aarch64-apple-darwin.tar.xz
```

macOS downloads are currently published for Apple Silicon (arm64). See System Requirements for platform requirements.

2. **(Optional) Verify the checksum**

```
curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-aarch64-apple-darwin.tar.xz.sha256
shasum -a 256 -c lyftdata-aarch64-apple-darwin.tar.xz.sha256
```

3. **Extract and install the binary**

```
tar -xJf lyftdata-aarch64-apple-darwin.tar.xz
chmod +x lyftdata-aarch64-apple-darwin/lyftdata sudo mv lyftdata-aarch64-apple-darwin/lyftdata /usr/local/bin/lyftdata # Remove the Gatekeeper quarantine flag so the unsigned binary can run sudo xattr -d com.apple.quarantine /usr/local/bin/lyftdata || true
```

> note

If you extracted the archive with Finder, the quarantine flag can be applied to the whole directory. Run `sudo xattr -dr com.apple.quarantine /usr/local/lib/lyftdata` before moving the binary, or execute the commands above from Terminal to clear the attribute. Use `man xattr` on macOS to learn more about how the command manages extended attributes.

> Note

4. **Confirm the version**

```
lyftdata --version
```

Windows (PowerShell)

1. **Download the archive**

```
   $downloadDir = 'C:\Program Files\LyftData'
New-Item -ItemType Directory -Path $downloadDir -Force | Out-Null Invoke-WebRequest -Uri
https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-pc-windows-msvc.zip -OutFile
"$downloadDir\lyftdata.zip" Unblock-File -Path "$downloadDir\lyftdata.zip" -ErrorAction
SilentlyContinue
```

2. **(Optional) Verify the checksum**

```
   Invoke-WebRequest -Uri https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-pc-
windows-msvc.zip.sha256 -OutFile "$downloadDir\lyftdata.zip.sha256"
$expected = (Get-Content "$downloadDir\lyftdata.zip.sha256").Split()[0] $actual = (Get-
FileHash "$downloadDir\lyftdata.zip" -Algorithm SHA256).Hash if ($expected -ne $actual) {
throw "Checksum mismatch: expected $expected but saw $actual" }
```

3. **Extract the archive**

```
   Expand-Archive -Path "$downloadDir\lyftdata.zip" -DestinationPath $downloadDir -Force
Unblock-File -Path "$downloadDir\lyftdata.exe" -ErrorAction SilentlyContinue
```

4. **Confirm the version**

```
   & "$downloadDir\lyftdata.exe" --version
```

Add `$downloadDir` to PATH or copy `lyftdata.exe` into a directory that is already on PATH so services can find it.

> note

If you extracted the archive with File Explorer, Windows may mark the files with the "Downloaded from the Internet" flag. Running `Unblock-File` clears that mark so SmartScreen does not prompt when the service starts—see the Microsoft PowerShell documentation for more details. Alternatively, right-click the ZIP (or `lyftdata.exe`), choose **Properties**, and select **Unblock** before running the installer commands.

> Note

Once the executable is available on the host, continue with the platform-specific server and worker setup guides.

Downloads

Download LyftData binaries and verify checksums before install

Source: install/downloads.mdx

Use this page to download the latest LyftData build for your platform. Match the archive to your operating system and architecture before starting the First pipeline guide. If you just want to try LyftData on a single machine, grab the server/CLI binary and head directly to the Evaluation Quickstart. Binary downloads are available for Linux, Windows, and macOS (Darwin).

> Try now

Prefer the guided flow? Try now

> Note

Verify each download against the matching SHA256 checksum before you extract it.

> note

Downloads vary between 20 and 120 MiB in size. When extracted, each download contains the following content:

- `lyftdata` or `lyftdata.exe` (server/CLI), **or** `lyftdata-worker` / `lyftdata-worker.exe` (worker-only)
- README.md
- LICENSE

> Note

Which download do I need?

- **Server + UI + CLI:** download `lyftdata` (required).
- **External worker host:** download `lyftdata-worker` (recommended) or use `lyftdata` if you prefer a single binary everywhere.

Refer to the Installation overview for platform-specific installation and configuration steps.

Downloads portal

Use the downloads portal to pick the latest release or beta build for your platform (including SHA256 checksums). If there is no “Release (latest)” yet, use “Beta (latest)”.

Open downloads

Product documentation PDFs

Downloadable product documentation is available from the Downloadable Docs page, including the versioned 2.1.0 PDF set.

Evaluation Quickstart

Launch LyftData on a single host in under 15 minutes

Source: install/evaluation-quickstart.mdx

Goal

Spin up the LyftData server with its built-in worker on a single host so you can explore the UI, run a sample job, and evaluate the product without permanent infrastructure changes.

- **Audience:** product managers, solution engineers, and platform leads running a proof-of-concept.
- **Time to complete:** ~15 minutes end to end.
- **What you will have at the end:** server listening on `127.0.0.1:3000` (HTTPS by default), built-in worker ready for jobs, and admin access to the UI.

> tip

Prefer to evaluate in containers? Use the Docker and Docker Compose guide.

> Note

Before you start

- One 64-bit workstation (Linux, macOS, or Windows 10/11) with admin privileges.
- Port `3000` available locally.
- Ability to unzip or extract archives and run a terminal or PowerShell session as administrator.

Optional: choose the admin password you plan to create during the setup flow.

Step 1 – Stage the binary

Follow the matching command block for your platform. Each one downloads the current stable binary, extracts it into `~/lyftdata-eval` (or the Windows equivalent), and confirms the version. If you need a beta or another artifact, choose it from the Downloads page.

Linux

```
mkdir -p ~/lyftdata-eval && cd ~/lyftdata-eval
curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-unknown-linux-
gnu.tar.xz curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-unknown-
linux-gnu.tar.xz.sha256 sha256sum -c lyftdata-x86_64-unknown-linux-gnu.tar.xz.sha256 tar -xJf
lyftdata-x86_64-unknown-linux-gnu.tar.xz cp lyftdata-x86_64-unknown-linux-gnu/lyftdata
./lyftdata chmod +x ./lyftdata ./lyftdata --version
```

macOS

```
mkdir -p ~/lyftdata-eval && cd ~/lyftdata-eval
curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-aarch64-apple-darwin.tar.xz
curl -LO https://downloads.lyftdata.com/release/latest/lyftdata-aarch64-apple-
darwin.tar.xz.sha256 shasum -a 256 -c lyftdata-aarch64-apple-darwin.tar.xz.sha256 tar -xJf
```

```
lyftdata-aarch64-apple-darwin.tar.xz cp lyftdata-aarch64-apple-darwin/lyftdata ./lyftdata
chmod +x ./lyftdata
```

```
xattr -d com.apple.quarantine ./lyftdata || true ./lyftdata --version
```

Windows

```
$evalDir = "$env:UserProfile\lyftdata-eval"
New-Item -ItemType Directory -Path $evalDir -Force | Out-Null Set-Location $evalDir Invoke-
WebRequest -Uri https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-pc-windows-
msvc.zip -OutFile lyftdata.zip Invoke-WebRequest -Uri
https://downloads.lyftdata.com/release/latest/lyftdata-x86_64-pc-windows-msvc.zip.sha256 -
OutFile lyftdata.zip.sha256 $expected = (Get-Content lyftdata.zip.sha256).Split()[0] $actual
= (Get-FileHash lyftdata.zip -Algorithm SHA256).Hash if ($expected -ne $actual) { throw
"Checksum mismatch: expected $expected but saw $actual" } Unblock-File -Path lyftdata.zip -
ErrorAction SilentlyContinue Expand-Archive -Path lyftdata.zip -DestinationPath $evalDir -
Force Unblock-File -Path .\lyftdata.exe -ErrorAction SilentlyContinue .\lyftdata.exe --
version
```

Step 2 – Launch the server with the built-in worker

Run the server in a new terminal session. Set the EULA flag for the life of that session and create a temporary staging directory.

> note

LyftData serves **HTTPS** by default using a self-signed certificate. When you open the UI, your browser will show a certificate warning—proceed for evaluation.

If you prefer plain HTTP for a local-only eval, add `--disable-tls` and use `http://localhost:3000` instead.

> Note

Linux

```
cd ~/lyftdata-eval
export LYFTDATA_LICENSE_EULA_ACCEPT=yes ./lyftdata run server \ --staging-dir
"$HOME/lyftdata-eval/server" \ --bind-address 127.0.0.1:3000
```

macOS

```
cd ~/lyftdata-eval
export LYFTDATA_LICENSE_EULA_ACCEPT=yes ./lyftdata run server \ --staging-dir
"$HOME/lyftdata-eval/server" \ --bind-address 127.0.0.1:3000
```

Windows

```
Set-Location $env:UserProfile\lyftdata-eval
$env:LYFTDATA_LICENSE_EULA_ACCEPT = "yes" .\lyftdata.exe run server  --staging-dir
"$env:UserProfile\lyftdata-eval\server"  --bind-address 127.0.0.1:3000
```

The server process keeps the built-in worker attached automatically, so no external workers are required for this pilot. On first start LyftData enters **Initial Setup Required** and writes a one-time setup link to `<staging-dir>/bootstrap/initial-admin.url`. In an interactive terminal the ready splash also shows the full setup URL. Leave the terminal window open; it streams logs and will surface any configuration errors immediately.

> tip

If you want a shared or headless pilot instead of a local-only setup flow, rerun the command with `--admin-init-password <password>`.

> Note

Step 3 – Sign in and validate

1. Open the one-time setup URL from the terminal output, or inspect `<staging-dir>/bootstrap/initial-admin.url` and copy the `URL=` value. 2. Create the `admin` password in the setup form. 3. Sign in at `https://localhost:3000` as `admin` with the password you just created. 4. Go to **Workers** and confirm the **Built-In Worker** shows `Online`. 5. Navigate to **Jobs** → **New job**, create the “Hello, world” template (or another simple pipeline), and run it. The run should complete on the built-in worker.

Success checklist

- Initial setup completed from the one-time setup link.
- Web UI reachable at `https://localhost:3000`.
- Built-in worker online in the dashboard.
- Sample job run succeeds.

If any step fails, consult the Troubleshooting guide and review the logs in the terminal window where the server is running.

Clean up or continue

- Stop the evaluation by pressing `Ctrl+C` in the server terminal.
- Ready to go deeper? Promote this host to a service by following the platform guide for your OS and enabling metrics via Next steps after install.

- Planning a production rollout? Share the Prerequisites checklist with your infrastructure partners and schedule host provisioning.

Track adoption

For evaluation pilots, capture the following signals as you proceed toward production:

- Number of evaluators who completed the quickstart.
- Time from first login to first successful job run.
- Whether metrics collection and dashboards were enabled during the pilot.

Document these metrics in your rollout plan so you can justify scaling beyond the pilot phase.

Networking & TLS

Make the server reachable and secure connections between the UI, CLI, and workers

Source: install/networking-and-tls.mdx

LyftData serves **HTTPS (TLS)** by default. If you do not provide TLS material, the server generates and persists a **self-signed** certificate (suitable for evaluation, not production).

For local evaluation you can either:

- Use `https://…` and accept the browser warning (and use `-k` with `curl`), **or**
- Start the server with `--disable-tls` to serve plain HTTP (recommended only when terminating TLS behind a reverse proxy, or for local-only evaluation).

> Recommended production Linux pattern

For internet-facing Linux hosts, keep LyftData bound to `127.0.0.1:3000`, run it with `--disable-tls`, and terminate TLS in Caddy or Nginx.

That keeps the raw server port off the network and centralizes certificates, hostname routing, and access policy in the proxy layer.

> Note

Basic connectivity model

- **Browser / CLI → Server**: Users access the UI and API on the server address (default `127.0.0.1:3000`, HTTPS by default).
- **Worker → Server**: Workers connect back to the server using `LYFTDATA_URL` (including the scheme, `http://` or `https://`) and authenticate with `LYFTDATA_WORKER_API_KEY`.

If the server binds only to `127.0.0.1`, remote browsers and remote workers cannot reach it.

Bind address and ports

- Change the server listen address with `--bind-address` (or `LYFTDATA_BIND_ADDRESS` when running as a service).
- Ensure firewalls allow browser/CLI access and worker access to the chosen port (default `3000`).
- Set `LYFTDATA_URL` on workers to the exact URL they should use to reach the server (include the scheme, such as `http://` or `https://`).

TLS options

LyftData supports two common deployment patterns:

- **Default (self-signed TLS)**: If you do not configure any TLS flags, LyftData serves HTTPS using a self-signed certificate. This is ideal for evaluation. External workers or CLI clients may need `--tls-insecure` (or `LYFTDATA_TLS_INSECURE=true`) until you replace it with a trusted certificate.

- **Built-in TLS**: Configure the server with a certificate and key using the `lyftdata run server` TLS flags (see `lyftdata run server --help`). This keeps the deployment simple when you do not already run an ingress proxy.
- **Reverse proxy termination**: Terminate TLS in a proxy (for example Caddy or Nginx) and forward requests to the server over HTTP (run the server with `--disable-tls`). This is the recommended production pattern on Linux. When using a proxy, ensure it forwards the correct client IP headers so logs and audits reflect real clients.

For broader hardening guidance, see Security Hardening.

Verify reachability

After changing networking or TLS settings, confirm:

- The UI loads in a browser on the expected URL.
- Liveness responds:
- HTTPS (default, self-signed): `curl -k https://<server-host>:3000/api/liveness`
- HTTP (only when `--disable-tls` is set): `curl http://<server-host>:3000/api/liveness`
- Workers show `Online` in the UI after you deploy them.

Common pitfalls

- `LYFTDATA_URL` uses `http://` but the server is serving `https://` (or vice versa).
- The server is still bound to `127.0.0.1`, so remote clients cannot connect.
- Certificate hostname mismatches (CN/SAN does not match the URL you use).

If you get stuck, start with Troubleshooting.

Installation Prerequisites

Prepare hosts before installing LyftData services

Source: install/prerequisites.mdx

Before installing the LyftData server or workers, make sure each host meets the baseline requirements below. These checks keep the platform secure, ensure upgrades go smoothly, and reduce duplicated setup steps across operating systems.

Readiness checklist

Task	Primary owner	Typical effort
Confirm host OS, CPU architecture, and service manager support	Infrastructure / IT	5 minutes
Create non-interactive service account and dedicated data directory	Infrastructure / IT	10–15 minutes per host
Decide whether clients reach LyftData directly on `3000` or through a reverse proxy on `80/443`, then validate DNS and firewall reachability	Networking / Security	10 minutes
Collect staging path, admin bootstrap password, and license acceptance decision	Platform team	5 minutes
Download and stage the LyftData binary (see Prepare the binary)	Platform / DevOps	10 minutes

Collect these details before touching the platform-specific guides so the install can proceed without handoffs or rework.

Supported platforms

- **Linux**: 64-bit Linux (GNU) on x86-64 or aarch64 (server and worker).
- **macOS**: Apple Silicon (arm64) on macOS 11.0+.
- **Windows**: Windows 10/11 or Windows Server 2016+ (x86-64).

Check Downloads for the currently published builds. All platforms use the same LyftData binary format (server/CLI or worker-only)—download the matching archive once per host and confirm `lyftdata --version` prints the expected build.

Accounts and directories

Create a dedicated service account on every host before you register it as a server or worker. The account should:

- Run without interactive logins (`/usr/bin/false` shell on Unix, `PasswordNeverExpires` on Windows).
- Own a writable data directory (for example `/var/lib/lyftdata-server` , `/var/lib/lyftdata-worker` , or `C:\\ProgramData\\LyftData\\server`).
- Restrict permissions to the service account (`chmod 700` on Unix, explicit ACLs on Windows).

Co-locating multiple services on the same host is supported, but create separate accounts and directories per role to maintain isolation.

Linux service accounts

```
sudo adduser --system --home /var/lib/lyftdata --disabled-login --group <user>
sudo mkdir -p /var/lib/lyftdata sudo chown <user>:<user> /var/lib/lyftdata sudo chmod 700
/var/lib/lyftdata
```

Replace `<user>` with the service name (for example `lyftdata` for the simple Linux server guide, `lyftdata-server` for a reverse-proxied production host, or `lyftdata-worker` for external workers) and update the home directory to match your staging path (e.g., `/var/lib/lyftdata-server` or `/var/lib/lyftdata/lyftdata`).

macOS service accounts

```
sudo sysadminctl -addUser <user> -shell /usr/bin/false -home /var/lib/<dir> -password -
sudo mkdir -p /var/lib/<dir> sudo chown <user>:staff /var/lib/<dir> sudo chmod 700
/var/lib/<dir>
```

Substitute `<user>` and `<dir>` for the role you are provisioning (for example `lyftdata` / `lyftdata-server` for the server).

Windows service accounts

```
New-LocalUser -Name "<user>" -Description "LyftData service" -PasswordNeverExpires -
UserMayNotChangePassword
Add-LocalGroupMember -Group "Users" -Member "<user>" New-Item -ItemType Directory -Path
"C:\ProgramData\LyftData\<dir>" -Force $acl = Get-Acl "C:\ProgramData\LyftData\<dir>" $rule =
New-Object System.Security.AccessControl.FileSystemAccessRule("
<user>","FullControl","ContainerInherit,ObjectInherit","None","Allow")
$acl.SetAccessRule($rule) Set-Acl "C:\ProgramData\LyftData\<dir>" $acl
```

Replace `<user>` / `<dir>` with values such as `lyftdata` / `server` or `lyftdata-worker` / `worker`, and keep the data directory permissions scoped to the service principal.

When running as services, it is recommended to set `LYFTDATA_STAGING_DIR` (server) and `LYFTDATA_JOBS_DIR` (worker) to these dedicated directories so upgrades, backups, and resets are predictable.

Network and firewall

Component	Default listener	Purpose
Server UI & API	`127.0.0.1:3000`	Web UI, CLI, worker coordination

Adjust firewall rules so the server port is reachable from browsers, the CLI, and any remote workers. For production, document who owns the rule change and track the approval in your rollout plan.

If you deploy behind a reverse proxy, keep `127.0.0.1:3000` loopback-only and expose only the proxy on `80/443`.

Licensing and secrets

- Accept the EULA once per host by setting `LYFTDATA_LICENSE_EULA_ACCEPT=yes` for both server and worker services.
- Community Edition supports the built-in worker only; adding external workers requires a license. See Licensing for options.
- Never hard-code API keys in scripts. Store worker API keys in scoped environment files or secret stores and limit access to administrators.

Environment variables (service installs)

LyftData supports configuration via CLI flags or environment variables. Service managers typically use environment variables so services start consistently at boot.

Common server variables:

- `LYFTDATA_STAGING_DIR` (recommended): server data directory. If unset, LyftData uses an OS default.
- `LYFTDATA_LICENSE_EULA_ACCEPT=yes` (recommended): avoids the one-time prompt during automation.
- `LYFTDATA_ADMIN_INIT_PASSWORD` (optional): sets the initial `admin` password. If unset, LyftData enters initial setup and writes a one-time setup link to `<staging-dir>/bootstrap/initial-admin.url`.
- `LYFTDATA_BIND_ADDRESS` (optional): defaults to `127.0.0.1:3000`.

Common external worker variables:

- `LYFTDATA_URL` (required): base server URL including scheme (`https://…` by default; use `http://…` only when the server runs with `--disable-tls`).
- `LYFTDATA_WORKER_API_KEY` (required for pre-issued enrollment): authenticates the worker against the server.
- `LYFTDATA_AUTO_ENROLLMENT_KEY` (optional alternative): use auto-enrollment instead of a pre-issued API key.
- `LYFTDATA_JOBS_DIR` (recommended): worker data directory. If unset, the worker uses an OS default.
- `LYFTDATA_WORKER_ID` (optional): if unset, the worker can persist identity locally after enrollment.

If you are using a self-signed server certificate during evaluation, you may need `--tls-insecure` (or `LYFTDATA_TLS_INSECURE=true`) on CLI/worker clients until you install a trusted certificate.

Configure optional settings—`LYFTDATA_BIND_ADDRESS`, `LYFTDATA_LOG_RETENTION_DAYS`, and `LYFTDATA_DB_DISK_USE_MAX_PERCENT`—only when you need to expose additional interfaces or adjust retention defaults.

Windows service management

Windows deployments can install services natively via `lyftdata service ...` (recommended) or by using the MSI installer. Use NSSM only for legacy installs when running an older binary that does not include the native service commands.

Next steps

1. Validate the prerequisites on every host. 2. Follow Prepare the binary to fetch and verify the executable. 3. Continue with the platform-specific server or worker guide to wire up the service manager.

System Requirements

Baseline host requirements for LyftData servers and workers

`Source: install/system-requirements.md`

Use this page to confirm your hosts meet the baseline requirements before starting the First pipeline guide or any production install. LyftData ships as prebuilt, self-contained 64-bit binaries for each supported platform.

> tip

Run the checks below on every server and worker you plan to provision so you can spot blockers before installation day.

> Note

Quick checklist

- [ ] Confirm the operating system appears in the supported list below
- [ ] Verify hardware meets or exceeds the recommended CPU, RAM, and disk targets
- [ ] Ensure networking/firewall rules allow workers to reach the server on the configured ports
- [ ] Decide where staging directories live and ensure the service account can write to them

Supported operating systems

LyftData is available for 64-bit Linux, Windows, and macOS. Server and worker processes are interoperable between different operating systems.

Recommended hardware

OS	Architecture	RAM	CPU Cores	Disk space
Linux (gnu)	x86-64	4 GB	4 cores	10 GB
Linux (gnu)	aarch64	4 GB	4 cores	10 GB
Darwin / Mac	aarch64	8 GB	4 cores	10 GB
Windows	x86-64	8 GB	4 cores	10 GB

Supported operating system versions

OS	Architecture	Minimum Versions
Linux (gnu)	x86-64	kernel 3.2+, glibc 2.17+
Linux (gnu)	aarch64	kernel 4.1+, glibc 2.17+
Darwin / Mac	aarch64	macOS 11.0+, Big Sur+
Windows	x86-64	Windows 10+, Server 2016+

Sizing and scaling

The following baseline guidance can help you plan capacity. Actual sizing depends on connector mix, event sizes, and action complexity.

- Small (dev/test, single worker):
- Server: 2 vCPU, 2–4 GB RAM, 10 GB disk
- Worker: 2 vCPU, 2–4 GB RAM, 10 GB disk
- Throughput: up to tens of events/sec with simple actions
- Medium (team, multiple jobs, 1–3 workers):
- Server: 4 vCPU, 8 GB RAM, 50–100 GB disk
- Workers (x1–3): 4 vCPU, 8 GB RAM, 20–50 GB disk each
- Throughput: hundreds of events/sec depending on I/O
- Large (production, 3+ workers):
- Server: 8 vCPU, 16–32 GB RAM, 200+ GB disk
- Workers (horizontally scale): 4–8 vCPU, 8–32 GB RAM, 50–200 GB disk each
- Throughput: scale linearly by adding workers for parallelizable jobs

Notes and tips:

- Disk usage: the Server stores logs/metrics in the staging directory. Cleanup runs when the disk usage threshold is reached (see Configuration). Plan headroom accordingly.
- Network: place workers close to data sources and sinks to minimize latency and egress.
- Scaling strategy: prefer horizontal scaling by adding workers and splitting pipelines across jobs using worker channels when fan-out or fan-in is needed.
- Spiky loads: use scheduled triggers or rate limits on inputs to smooth ingestion.
- Observability: monitor worker CPU, memory, and backpressure; increase worker count before saturation.

Platform notes: Windows and Mac

LyftData provides native builds for Windows and macOS. The quick-start commands are the same; differences are mostly around service management and paths.

- Windows:
- Run in a terminal for development: `lyftdata run server`
- For running as a service, use the MSI installer or `lyftdata service install` (native Windows Service manager support). Ensure the service account has write access to the staging directory.
- Paths: prefer directories without spaces for data directories. Environment variables can be set via System Properties or a service wrapper.
- macOS:
- Run in Terminal for development: `./lyftdata run server`
- For background operation, use launchd (create a plist) or a process supervisor like `brew services`. Ensure the data directory is writable by the user running the service.

On both platforms:

- Default Server bind address is 127.0.0.1:3000 (see Configuration). Use `--bind-address 0.0.0.0:3000` to accept remote connections.

- You can accept the EULA non-interactively for automation using the environment variable noted in the Configuration page.
- Workers authenticate with an API key and connect to the Server URL similar to Linux deployments.

Uninstall & Reset

Start over safely during an evaluation, or remove services during decommissioning

Source: install/uninstall-and-reset.mdx

This page covers two different scenarios:

- **Evaluation reset**: wipe a local trial environment and start fresh.
- **Production uninstall**: remove services during decommissioning (with backups and warnings).

Evaluation reset (start over)

If you used the Evaluation Quickstart, all state is local to the directories you chose (`~/lyftdata-eval` on Unix, `%UserProfile%\\lyftdata-eval` on Windows). To reset:

1. Stop the server process (Ctrl+C in the terminal running `lyftdata run server`). 2. Delete the evaluation directory you created (this removes jobs, run history, logs, and settings for that eval). 3. Re-run the quickstart from scratch.

If you want to keep your current jobs/history but reset only credentials, do not delete the staging directory; instead change the admin password and re-issue worker keys.

Production uninstall (decommissioning)

Uninstalling a production deployment is destructive. Before removing anything:

- Take and validate a backup: Backup & Recovery
- Record your current configuration and the paths you used (`LYFTDATA_STAGING_DIR` , `LYFTDATA_JOBS_DIR` , service unit/plist/NSSM details).

At a high level, decommissioning is:

1. Stop the server and worker services. 2. Remove the service definitions (systemd unit / launchd plist / NSSM service). 3. Remove the data directories only after you are confident you no longer need the deployment.

If you are troubleshooting a broken installation rather than decommissioning, you usually want a reset (wipe the data directory and re-install) rather than a full uninstall. Start with Troubleshooting.

Upgrades & Rollbacks

Upgrade safely and recover quickly if something goes wrong

Source: install/upgrades-and-rollbacks.mdx

Upgrading LyftData is usually “replace the binary and restart services”, but you should treat it like an application upgrade: take a backup, verify liveness, and have a rollback plan.

Before you upgrade

- Read the release notes: Releases overview
- Take a backup of the server data directory and any external state you rely on: Backup & Recovery
- Pick a maintenance window if you are running production workloads.

Default upgrade procedure (production)

1. **Back up** the server’s staging directory (`LYFTDATA_STAGING_DIR`) and record the current `lyftdata --version` . 2. **Stop** the server service (and any workers you manage on the same host). 3. **Replace** the `lyftdata` binary with the new version (keep file permissions and service paths the same). 4. **Start** the server service, then start worker services. 5. **Verify**:

- Liveness responds on the expected URL:
- HTTPS (default, self-signed): `curl -k https://<server-host>:3000/api/liveness`
- HTTP (only when the server runs with `--disable-tls`): `curl http://<server-host>:3000/api/liveness`
- You can sign in to the UI.
- Workers show `Online` and can run a staged job.

Rollback guidance

If the upgrade fails basic checks:

1. Stop the affected service(s). 2. Restore the previous `lyftdata` binary. 3. Restart and re-check liveness and worker connectivity.

If an upgrade includes a data migration, restoring the previous binary may not be sufficient. In that case, restore the server data directory from your pre-upgrade backup.

Evaluation upgrades

For evaluation environments created with Evaluation Quickstart, the simplest upgrade is to download a fresh binary into your eval directory and restart the process. If you want to keep your jobs/history, keep the staging directory and only replace the binary.

Workers

Understand built-in vs external workers, enrollment, and where to install them

Source: install/workers.mdx

Workers execute jobs. The server stores job definitions, schedules runs, and deploys staged jobs to workers.

Built-in vs external workers

- **Built-in worker**: Runs alongside the server and is the default for evaluation. If you followed the Evaluation Quickstart, you already have a built-in worker and do not need to install anything else to run jobs end-to-end.
- **External workers**: Run on additional hosts so you can scale out, isolate workloads, or place execution closer to sources/destinations. External worker support depends on your license; see Licensing and Community Edition.

What a worker needs to connect

External workers need four core values:

- `LYFTDATA_WORKER_ID`: the worker's unique identity.
- `LYFTDATA_WORKER_API_KEY`: the API key used to authenticate to the server.
- `LYFTDATA_URL`: the base URL of the server (how the worker reaches it).
- `LYFTDATA_JOBS_DIR`: the worker's local data directory.

You typically create the worker ID in **Workers**, then create an API key in **Settings** → **API Keys**, and finally install the worker service on the target host.

Install an external worker

Choose the guide that matches the worker host OS:

- Linux: Worker installation
- macOS: Worker installation
- Windows: Worker installation

After the service starts, confirm the worker appears as `Online` in the UI. If it does not, the fastest checks are the server URL, API key, and firewall rules. See Networking & TLS and Troubleshooting.

After the worker is online

- Deploy a staged job to the worker to confirm end-to-end execution.
- Enable metrics collection for new workers (disabled by default) via Next steps.
- If you are scaling out, review System Requirements and the Scaling playbook.