PRODUCT DOCUMENTATION · JUST RUN IT

Enterprise Automation Platform User Guide

This guide describes what the platform can do, walks through the web UI with real screenshots, explains the technology stack, deployment models, and how agents execute work. Lab default login: `admin` / `change-me` at `http://localhost:8080`.

Product
Lean workload automation control plane for jobs, workflows, schedules, and agents.

Audience
Operators, platform engineers, and product owners evaluating or running the MVP.

Version snapshot
Spring Boot 3.5.x · Java 21 · UI screenshots from lab stack (July 2026).

Contents

1 Overview & philosophy

The **Enterprise Automation Platform** is a web-based control plane for defining work, running it where files and scripts live, scheduling it, and observing outcomes. Product stance is **Just Run It**: reliability and clarity over feature bloat (no visual designer tax, no adapter catalog for its own sake).

> **What you get in the MVP**
> Define jobs (shell, HTTP, no-op, Git release binaries) → chain them in workflows → trigger on cron or on demand → execute locally or via out-of-process agents on
> **Linux, Windows, or other Java-capable hosts**
> → search/page execution history, cancel/kill runs, and optionally email on failure.

Object model (control plane):

OBJECT	ROLE
Job definition	Atomic unit of work (type + params + limits)
Workflow definition	Ordered steps over jobs with conditions
Schedule	Cron expression targeting a job *or* workflow by numeric ID
Agent	Registered execution endpoint (heartbeat + capacity)
Job / workflow execution	Runtime instance with status, logs, error message
User / audit	RBAC accounts and change trail

2 Capabilities

2.1 Jobs

TYPE	BEHAVIOR	WHERE IT RUNS
NOOP	Succeeds immediately (wiring / smoke tests).	Local server or agent host
COMMAND	Real shell: `/bin/sh -c` (Linux) or `cmd.exe /c` (Windows). Capture stdout/stderr and exit code.	**Executor host** (local = server; agent mode = agent host)
HTTP	HTTP GET against a URL; status and body recorded.	Executor host
GIT_RELEASE	Download release asset from GitHub/GitLab → cache → execute. See `docs/GIT_RELEASE_JOBS.md`.	Executor host (cache on that node)

- Create / edit / enable / disable jobs in the UI or REST API.
- Optional preferred agent; per-job concurrency limits.
- List shows **numeric database IDs** (required when building schedules).
- **Schedule** button prefills the new schedule form with the job ID.

2.2 Workflows

- Ordered steps referencing job definitions.
- Step conditions: `ALWAYS`, `ON_SUCCESS` (default fail-fast style), `ON_FAILURE`, optional expression.
- Optional `continueOnFailure` behavior when you need non–fail-fast paths.
- Restart from a failed step (UI + `POST .../workflows/executions/{id}/restart`).
- Schedule button and ID column same as jobs.

2.3 Schedules

- Cron-based triggers targeting either a **job ID** or a **workflow ID** (numeric, not name).
- Enable / disable without deleting the definition.
- Edit existing schedules from the UI.

2.4 Executions & observability

- Job and workflow execution history with search, filters, and paging (default 20, max 100).
- Status lifecycle (queued → running → success / failed / cancelled).
- **Failed only** filter; **Started** timestamp; error/reason preview on list rows.
- Detail page: “Why failed”, error message, log output.
- Cancel / kill running or queued work.
- Dashboard: counts, online agents, recent executions with deep links.

2.5 Agents

- Register, heartbeat, claim work, report completion.
- Capacity as a server-side concurrent-claim limit (see §6 for sequential vs parallel nuance).
- **Linux and Windows agents are both supported** (same JAR; shell differs by OS). Other OS hosts with Java 21+ are possible where a POSIX shell exists—see §6.3.
- Remove stale agents from the UI/API.

2.6 Security & notifications

- Local users with roles: VIEWER, OPERATOR, ADMIN.
- Optional OIDC (modes: local / hybrid / oidc) for Okta, Entra, Keycloak, etc.
- Admin user management page.
- Agents authenticate with HTTP Basic (typically the local operator account).

- Optional email on failure (and optional on success); SMTP off by default so empty config does not mark health DOWN.

2.7 UX polish

- Three color themes: **Dark** (default), **Medium**, **Light** — stored in browser local storage.
- Consistent nav: Dashboard, Jobs, Workflows, Schedules, Agents, Executions, Users (admin).

3 User interface (screenshots)

Screenshots were captured from a running lab instance (Docker Compose, agent mode). Your data and counts will differ; layout and navigation match the product.

3.1 Login

Sign in with a local account or, when OIDC is enabled, use the identity provider path configured by operations. Theme can be switched before or after login.

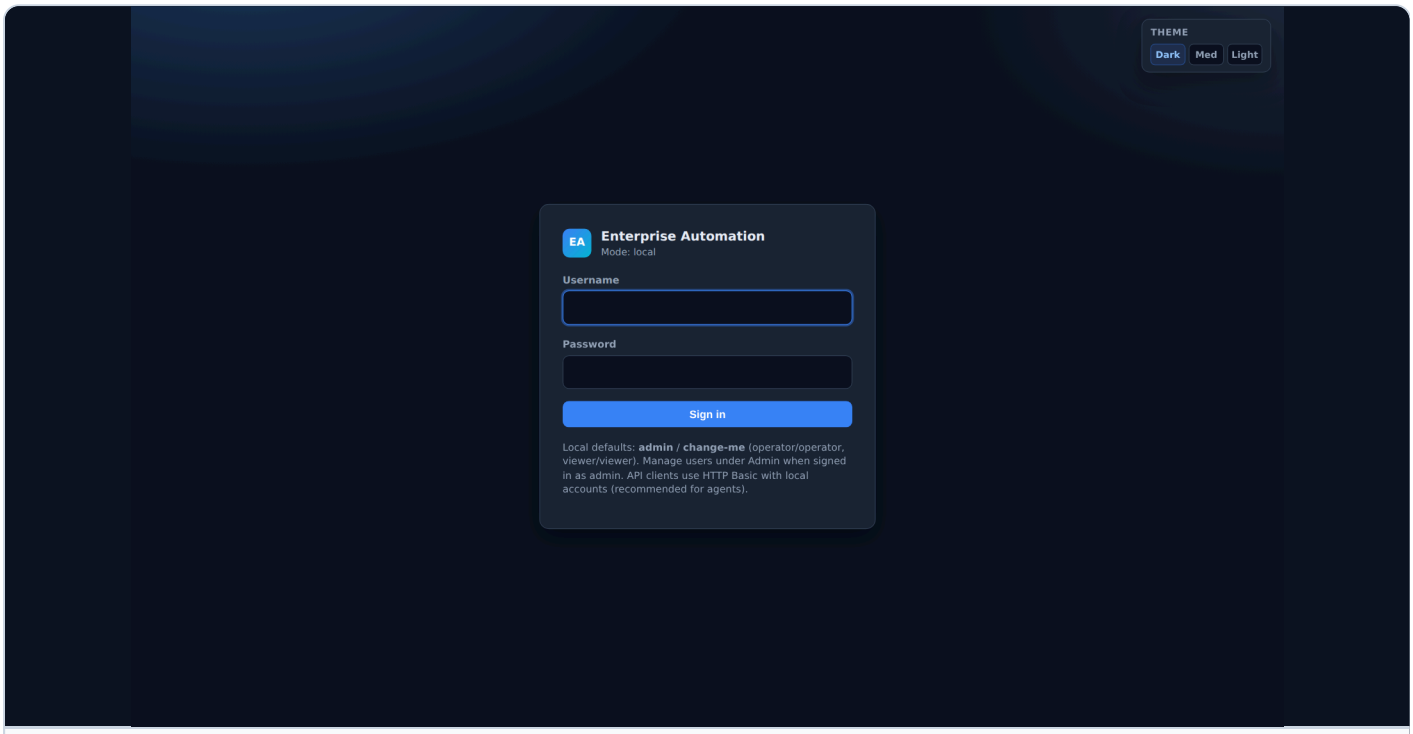


Figure 1 — Login (`/login`). Lab admin: admin / change-me.

3.2 Dashboard

At-a-glance health: job/workflow/schedule counts, online vs total agents, and recent executions with links into detail (including failure context).

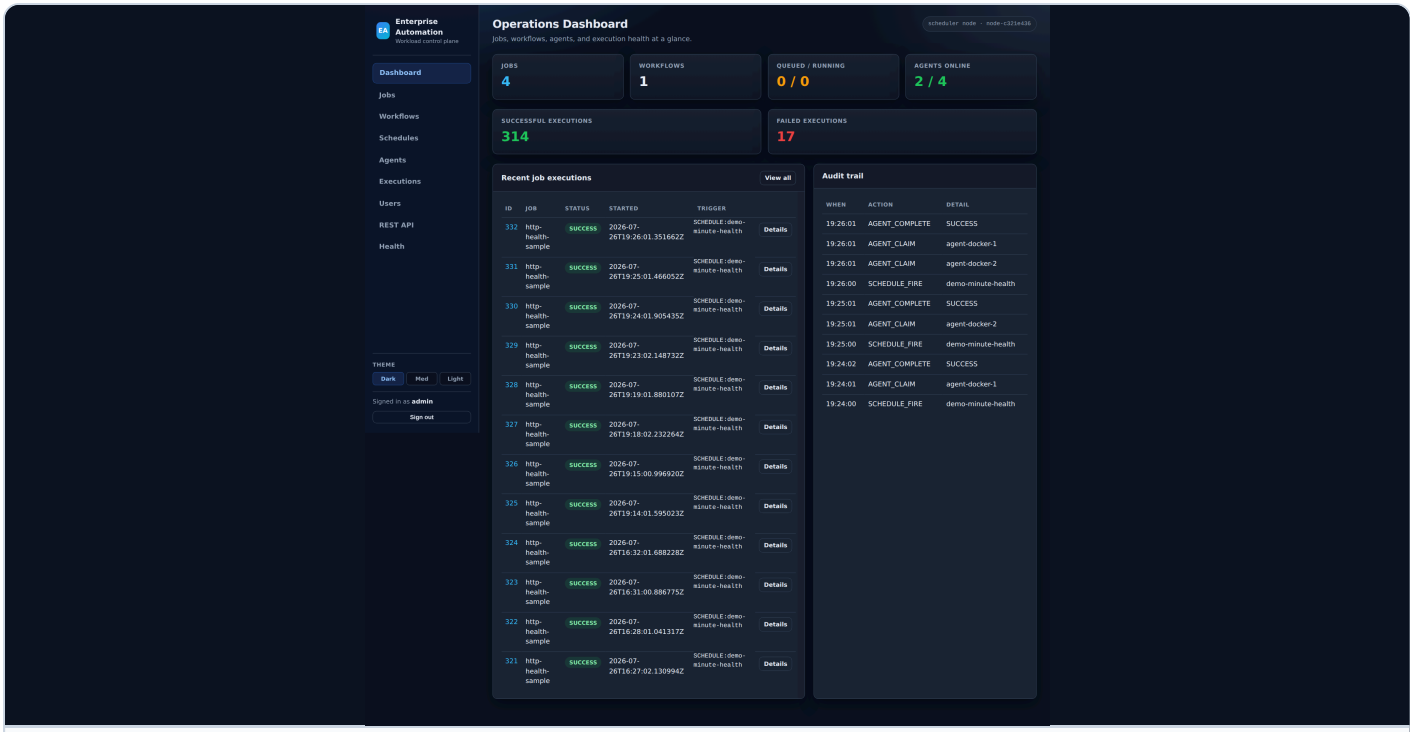


Figure 2 — Dashboard home after authentication.

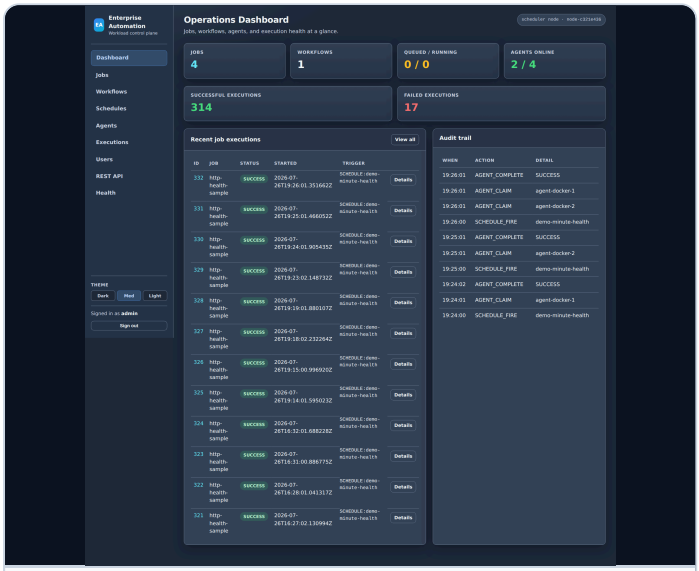


Figure 3a — Theme: Medium

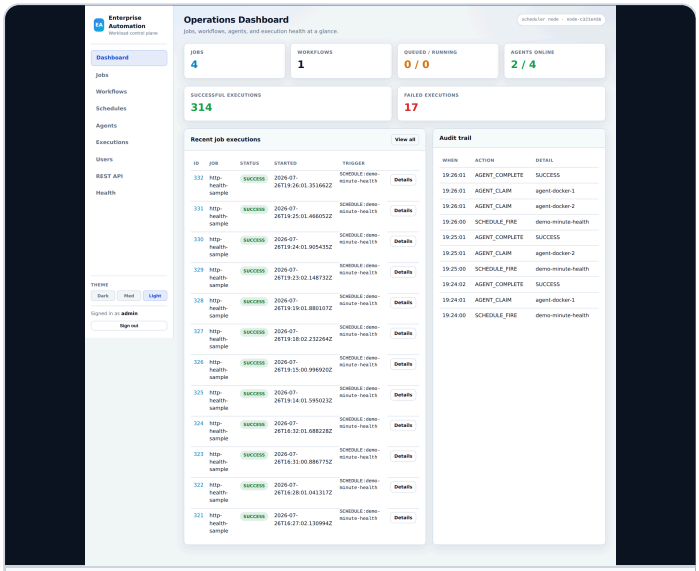


Figure 3b — Theme: Light

3.3 Jobs

List, search, filter, and page job definitions. Columns include the numeric **ID** used by schedules. Actions: Run, Edit, Schedule, enable/disable.

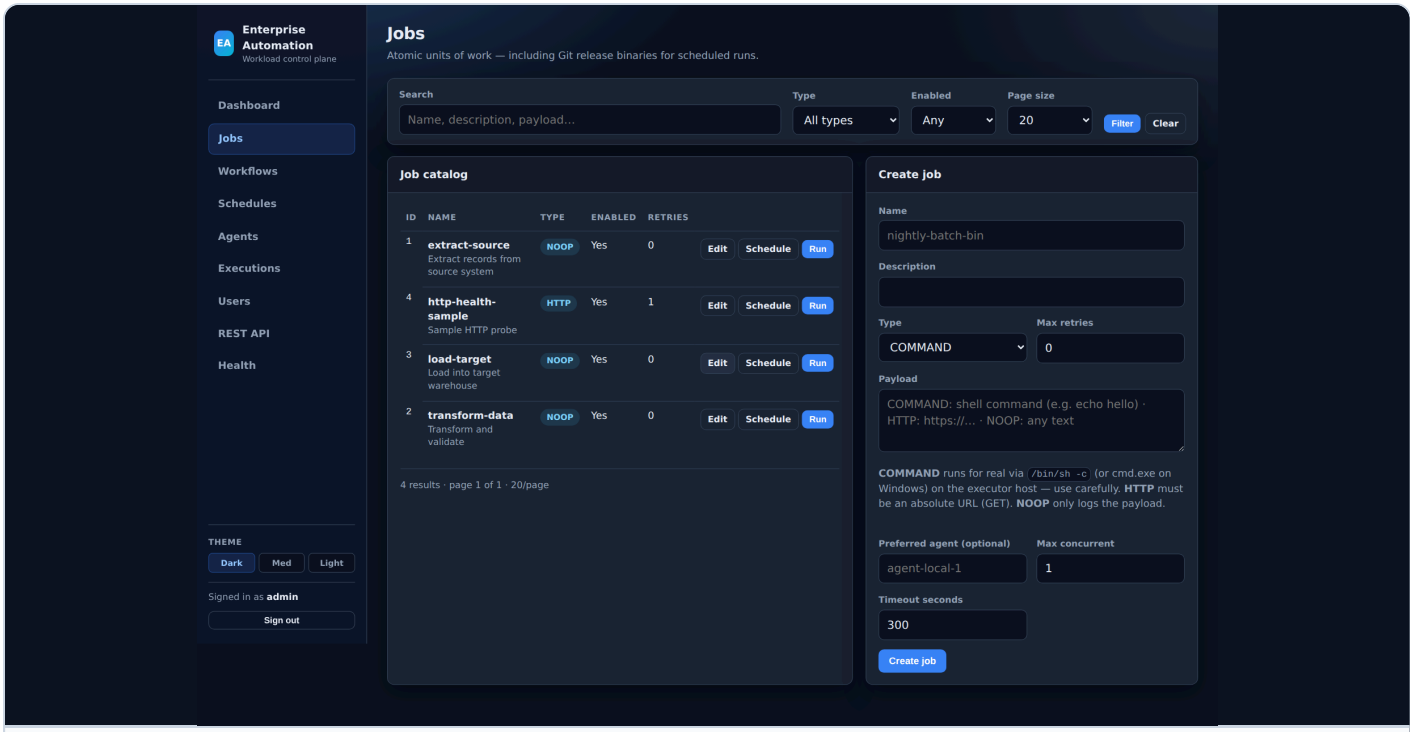


Figure 4 — Jobs list.

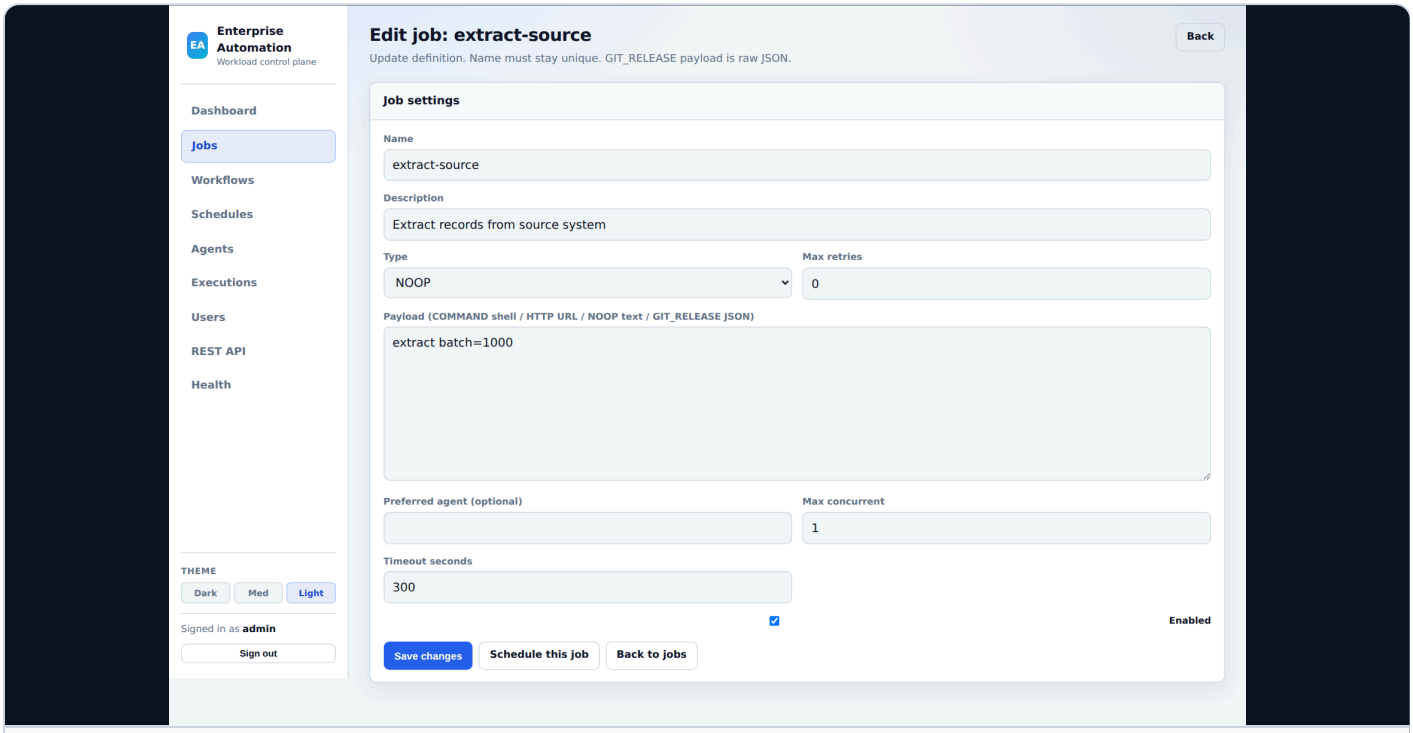


Figure 5 — Create / edit job (type, parameters, preferred agent, concurrency).

> **Schedule IDs are numeric**
> When creating a schedule, use the job or workflow *database ID* shown on the list (or prefilled by the Schedule button) —not the display name.

3.4 Workflows

Define multi-step orchestrations, conditions, and failure behavior. Same ID / Schedule patterns as jobs.

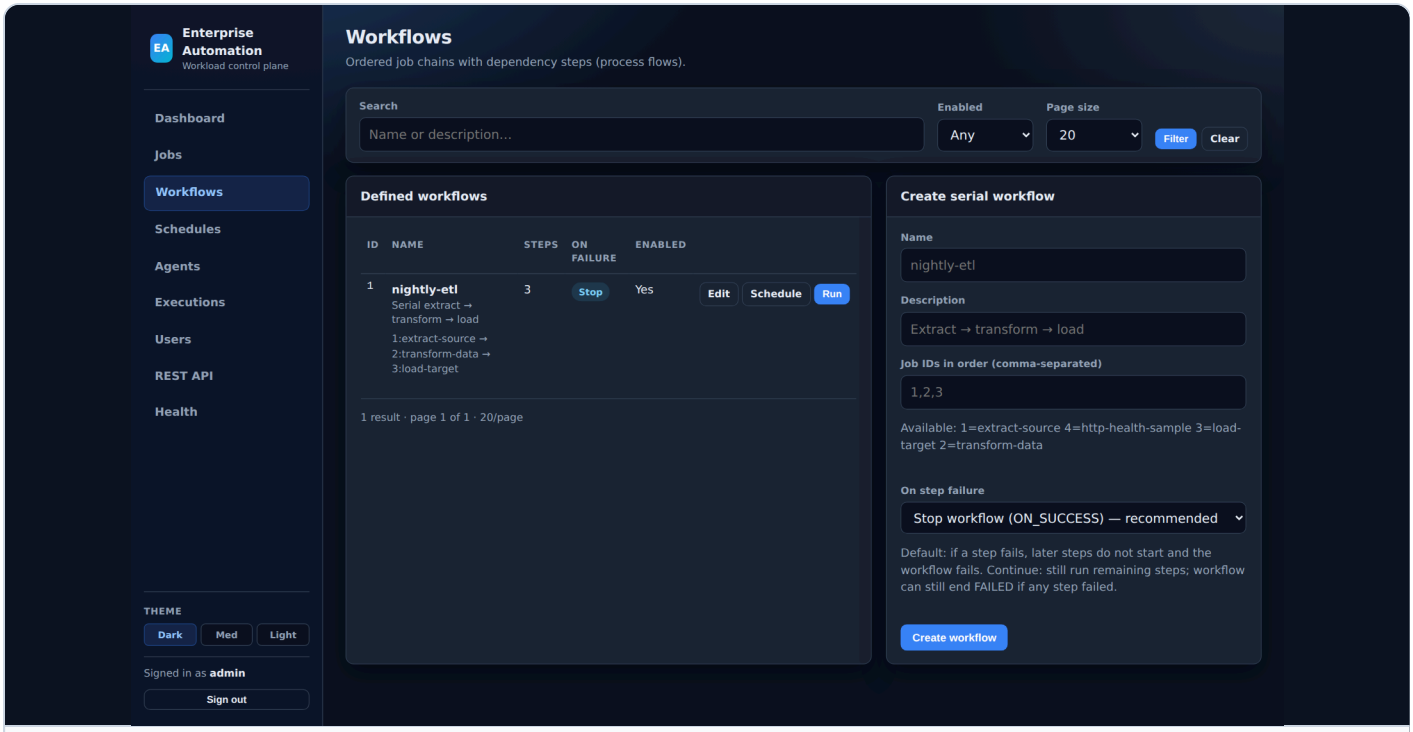


Figure 6 — Workflows list.

3.5 Schedules

Cron definitions targeting a job or workflow ID; enable/disable and edit in place.

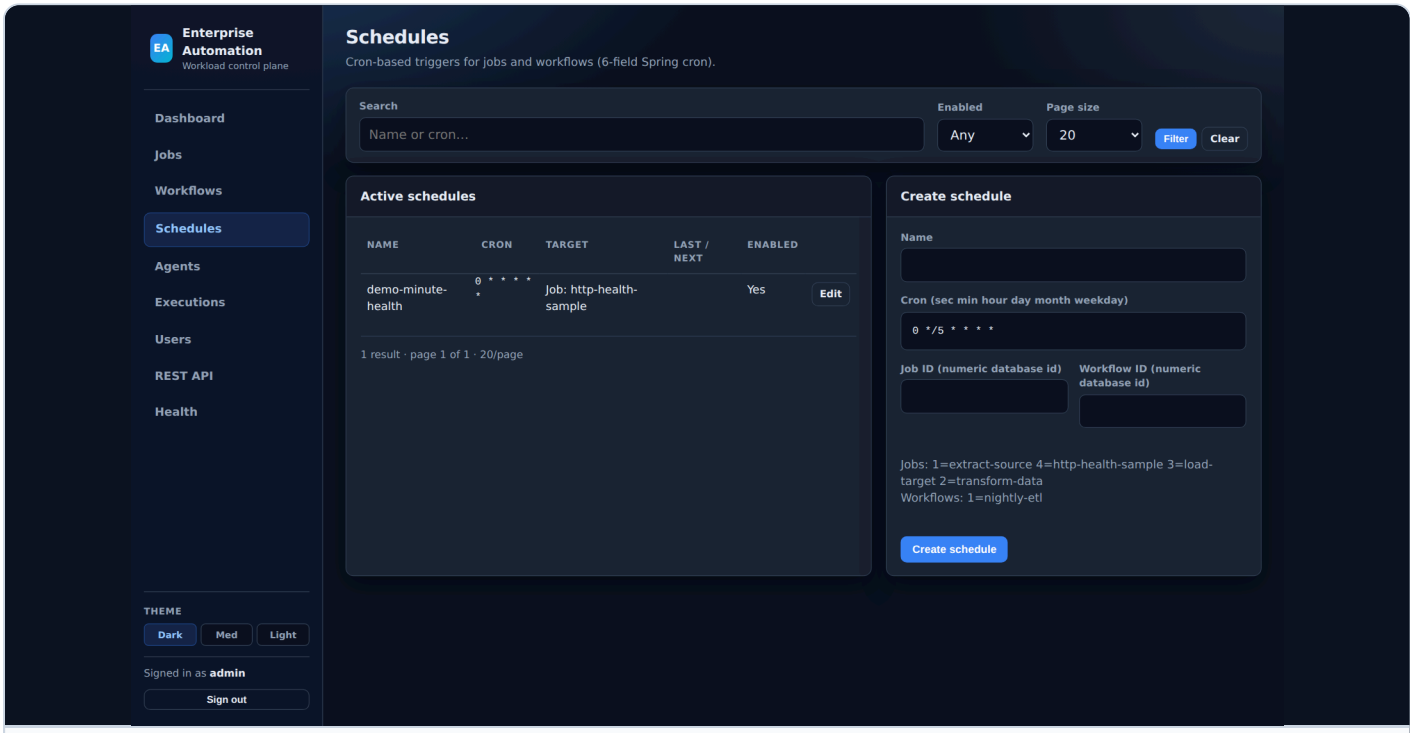


Figure 7 — Schedules list.

3.6 Agents

Registered workers show name, host, capacity, last heartbeat, and online state. Remove agents that are permanently retired.

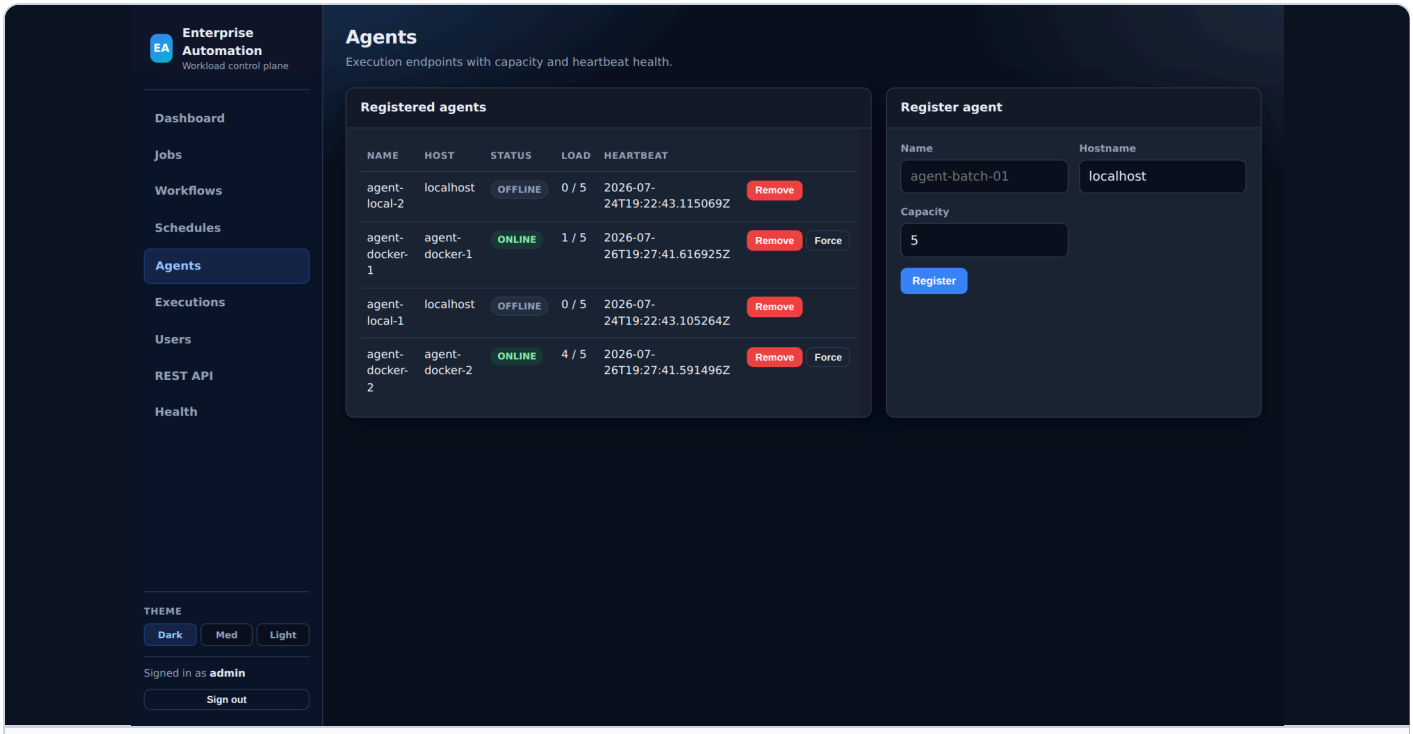


Figure 8 — Agents inventory.

3.7 Executions

Primary ops surface for “what ran and why did it fail.” Use status filters (including Failed only), search, and paging. Open a row for full logs.

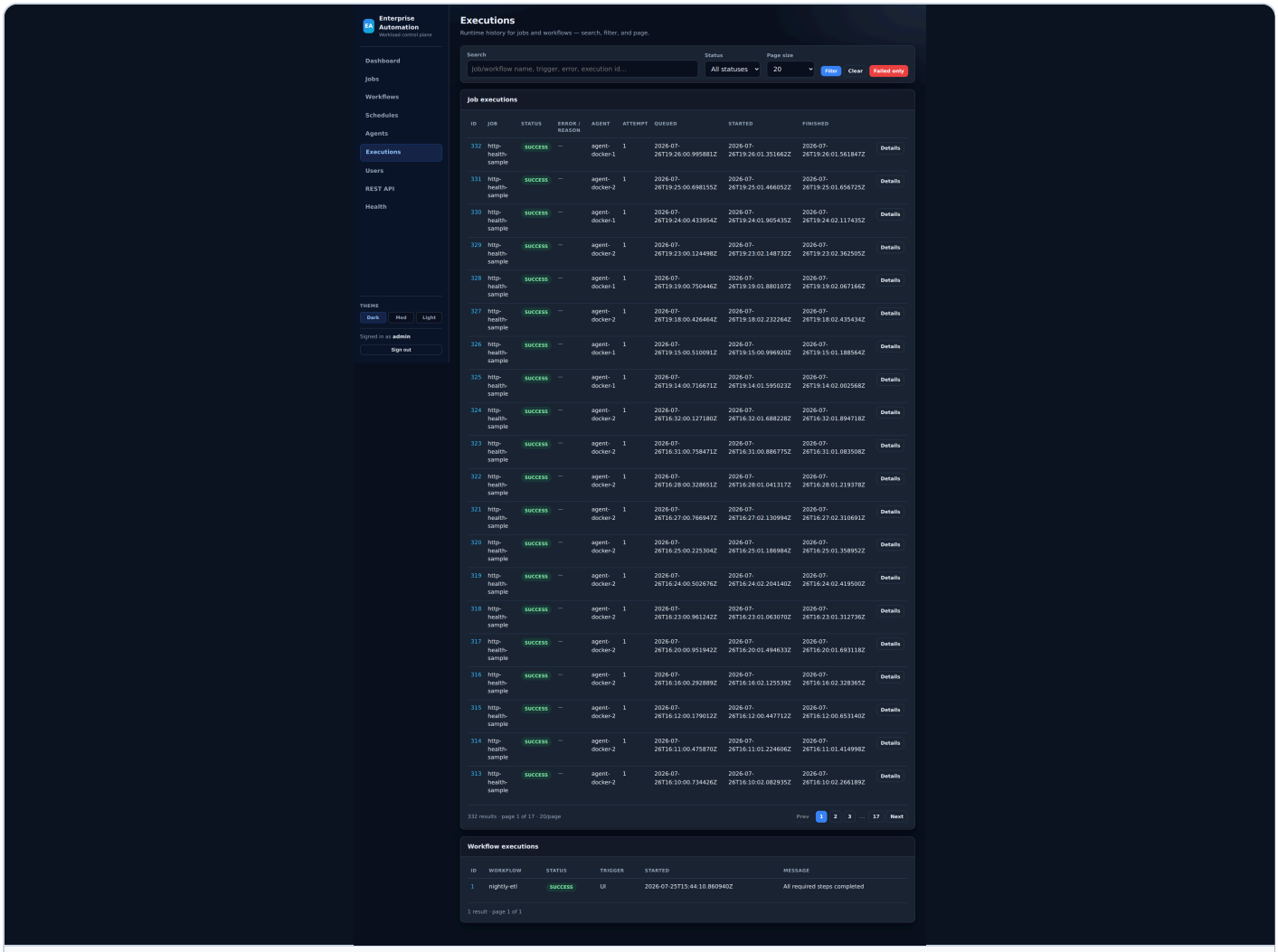


Figure 9 — Job executions list.

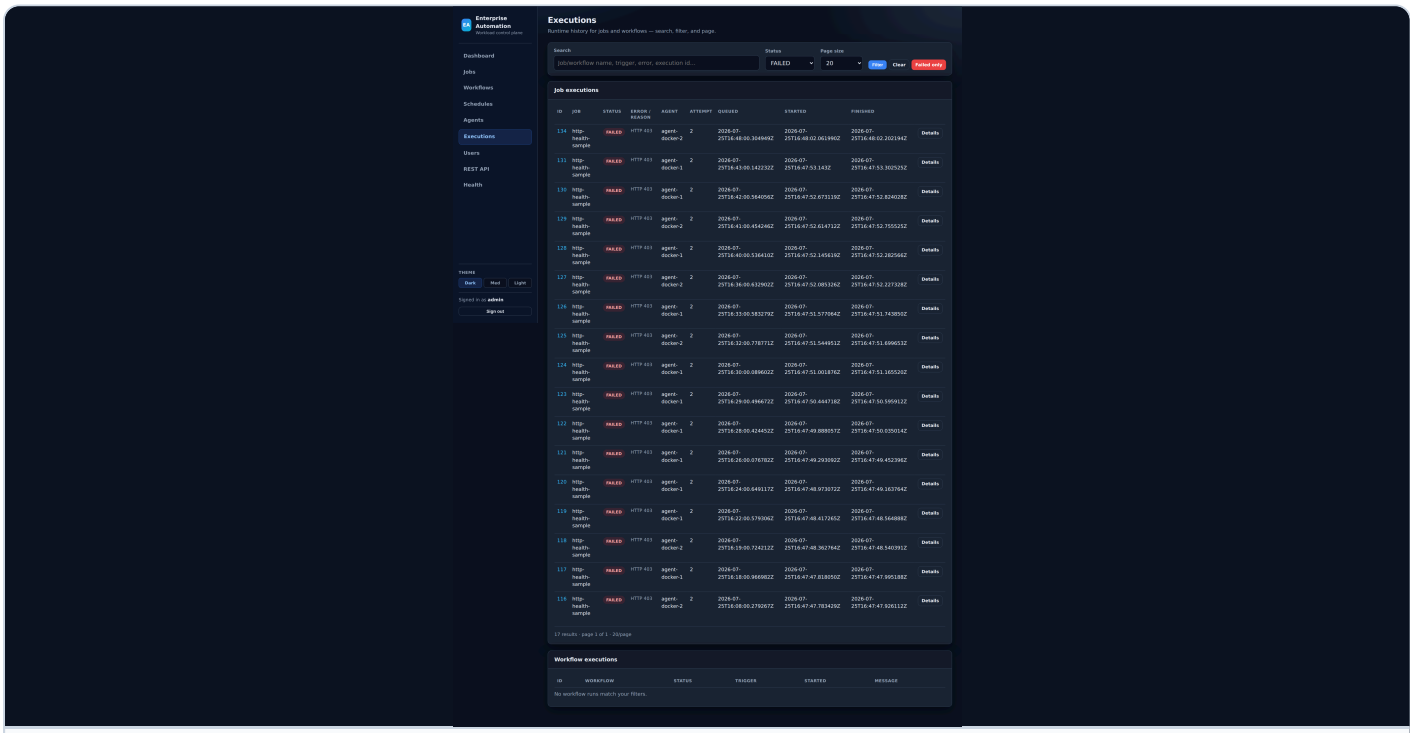


Figure 10 — Failed-only filter and error preview columns.

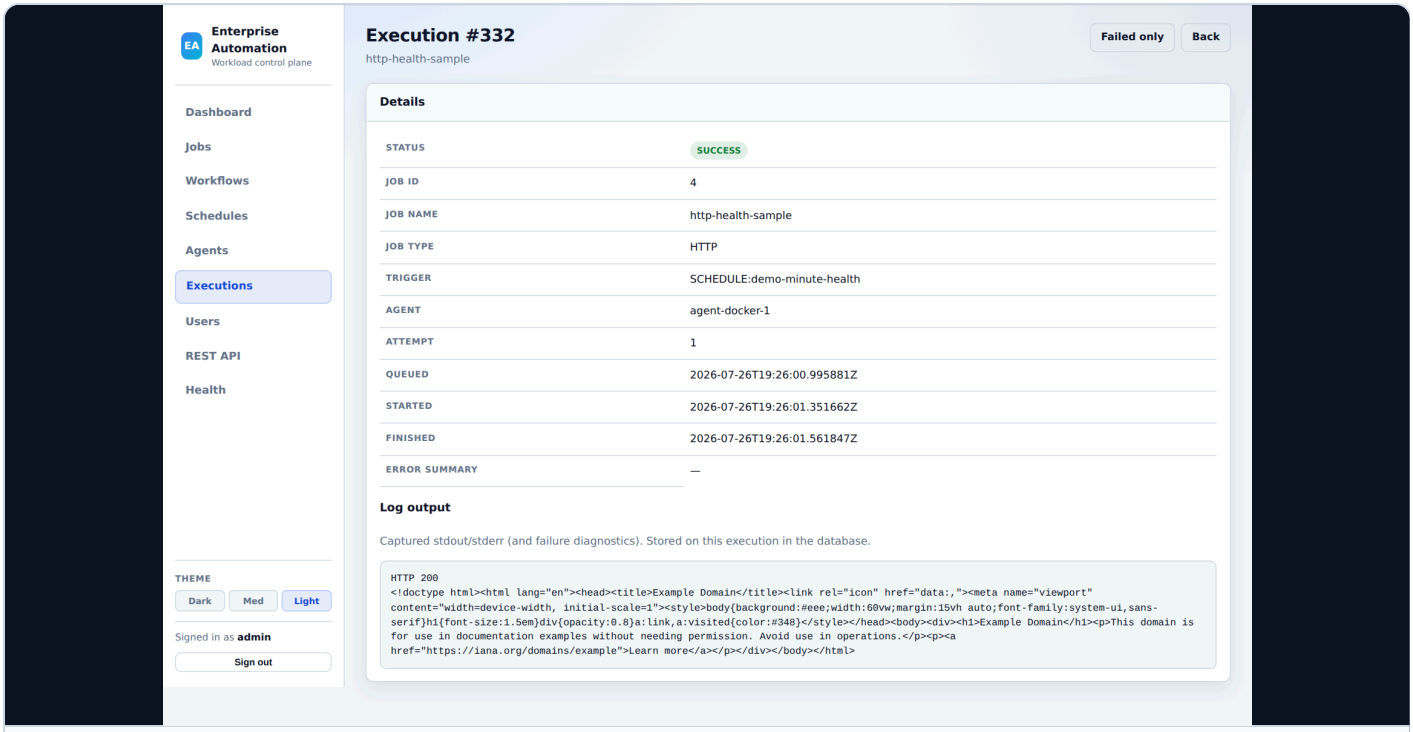


Figure 11 — Execution detail: status, timing, error message, log output, cancel/kill when applicable.

3.8 Users (admin)

Administrators manage local accounts, roles, enablement, and passwords.

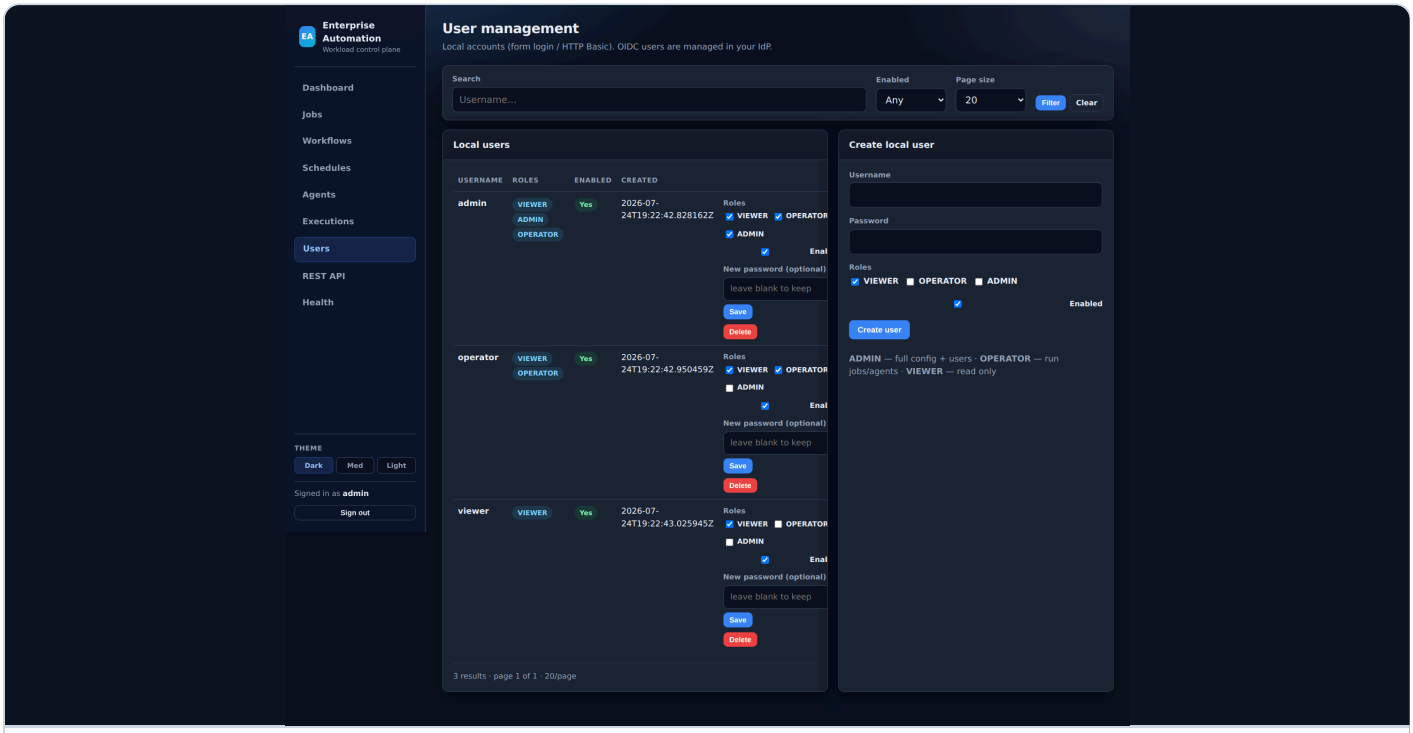


Figure 12 — Admin → Users.

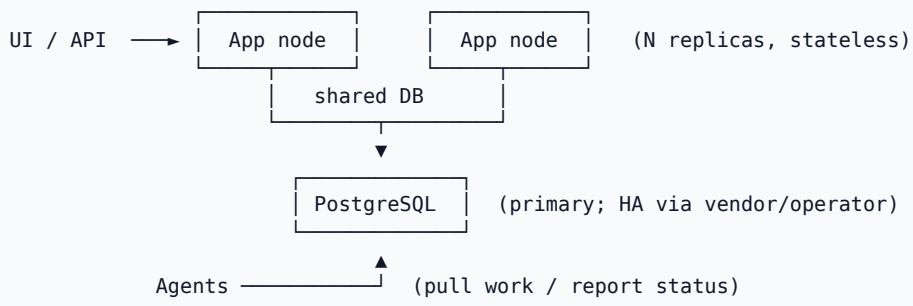
4 Technologies leveraged

LAYER	TECHNOLOGY	ROLE
Language / runtime	**Java 21**	LTS runtime for server and agent modules
Application framework	**Spring Boot 3.5.x**	HTTP, DI, config, packaging; control plane and agent process
Web UI	**Thymeleaf** + server-rendered HTML/CSS/JS	Ops console (no separate SPA build required)
API	Spring MVC REST (`/api/v1/...`)	Programmatic control; agents claim/complete here
Persistence	**Spring Data JPA / Hibernate**	Definitions, executions, users, audit
Databases	**PostgreSQL 16** (prod/lab compose); **H2** (local JAR dev)	Source of truth; shared state for HA app nodes
Security	**Spring Security** + optional **OIDC**	Form login, Basic (agents/API), SSO hybrid/oidc modes
Scheduler	In-process scheduling with **DB lease**	Only one active scheduler owner across HA replicas
Observability	**Spring Boot Actuator**	Liveness/readiness health (Compose/K8s probes)
Build	**Maven** multi-module	`server`, `agent` artifacts
Containers	**Docker** multi-stage builds	Maven build inside image; slim runtime jar
Local multi-node	**Docker Compose**	Postgres + 2 app nodes + 2 agents
Kubernetes	**Helm** chart (`charts/automation-platform`)	AKS-oriented deploy of server HA + agents + optional Postgres
Notifications	Optional JavaMail / SMTP	Fail (and optional success) email; disabled by default

```
server/   # control plane UI + REST + scheduler + security + DB
agent/    # out-of-process worker (claim / execute / complete)
charts/   # Helm
docs/     # briefs, PDFs, this user guide
```

5 Deployment model

The platform is a **stateless application tier** in front of a **shared database**, with optional **out-of-process agents** that pull work. Prefer agent mode for production so scripts and secrets stay on worker hosts, not on the control plane.



5.1 Execution modes

MODE	PROPERTY	WHEN TO USE
local	AUTOMATION_EXECUTION_MODE=local	Single-node dev / lab JAR. Server process executes COMMAND/HTTP/etc. on itself.
agent	AUTOMATION_EXECUTION_MODE=agent	Compose, K8s, production. Agents claim jobs and run on agent hosts.

5.2 Option A — Local JAR (developer)

```
cd develop/automation-platform
mvn -pl server -DskipTests package
java -jar server/target/automation-server-0.1.0-SNAPSHOT.jar
# UI http://localhost:8080  (H2, typically local execution mode)
```

5.3 Option B — Docker Compose (lab / demo HA)

Ships **Postgres 16**, **two control-plane nodes** (`:8080`, `:8081`), and **two agents** in agent mode.

```
cd develop/automation-platform
./bin/docker-compose up --build -d
# or: docker compose up --build -d
# open http://localhost:8080
```

SERVICE	ROLE
db	PostgreSQL 16, volume pgdata
app1 / app2	Server replicas; liveness via Actuator
agent1 / agent2	Workers polling the control plane with Basic auth

> **Lab credentials are not production secrets**
> Default bootstrap passwords (admin / operator / viewer) are for local demos only. Override via environment variables and integrate secrets managers in real environments.

5.4 Option C — Kubernetes (Helm)

Chart path: `charts/automation-platform`. Deploys server HA, agents, and optional Postgres for AKS-style clusters. Configure image registry, replicas, resources, ingress, and secrets per environment. Use Actuator `/actuator/health/liveness` and readiness probes (not the aggregate health if optional mail is misconfigured in older builds—current builds keep mail conditional).

5.5 Scaling shape

- **App nodes:** scale horizontally; share one Postgres; scheduler lease ensures a single active cron owner.
- **Agents:** scale out for more host coverage and higher parallel claim capacity (see §6).
- **Database:** production HA/DR via Postgres operator, multi-AZ, tested restore—not reinvented in the app.

6 Agent types & execution model

“Agent” in this product means an **execution endpoint** that can run job payloads. There are two operational models (local vs out-of-process), full support for **Linux and Windows** workers, and a practical path for other Java-capable hosts. Capacity rules apply on every OS.

> **Linux and Windows agents are first-class**
> The same `automation-agent` JAR runs on both. The control plane can stay on Linux (Compose/Kubernetes) while teams place agents on Windows boxes that own scripts, drives, and Windows services—and on Linux hosts that own POSIX scripts. Pin each job with
> **preferred agent**
> and OS-appropriate payloads.

6.1 Local executor (no separate agent process)

ASPECT	DETAIL
When	`AUTOMATION_EXECUTION_MODE=local`
Process	Jobs run **inside the server JVM** on the control-plane host
Best for	Dev, unit demos, single-box PoCs
Limits	Couples scripts/binaries to the server host; weaker isolation for production COMMAND work

6.2 Out-of-process agent (worker module)

ASPECT	DETAIL
Module	`agent/` → `automation-agent` JAR / container
When	`AUTOMATION_EXECUTION_MODE=agent` on the server
Loop	Register → heartbeat → **claim** next job → execute → **complete** with status/logs
Auth	HTTP Basic against control plane (typically local `operator` user)
Config (examples)	`AUTOMATION_SERVER`, `AUTOMATION_USER`, `AUTOMATION_PASSWORD`, `AUTOMATION_POLL_MS`, `AUTOMATION_CAPACITY`, `AUTOMATION_AGENT_NAME`
Best for	Production and multi-host script execution (“run where the files live”)

```
# Conceptual agent loop
while true:
  heartbeat()
  job = claimWork(capacity)   # server enforces concurrent claims ≤ capacity
  if job:
    result = execute(job)     # NOOP | COMMAND | HTTP | GIT_RELEASE on this host
    complete(job, result)
  sleep(pollMs)
```

6.3 Operating systems: Linux, Windows, and other hosts

The agent is a **pure Java 21** process. It is not Linux-only. Packaging in the lab (Docker Compose / Helm) is Linux-oriented; bare-metal or VM agents on other OSes are deployed by running the JAR where Java is installed.

HOST OS	SUPPORTED?	HOW YOU DEPLOY TODAY	COMMAND SHELL	NOTES
Linux	**Yes — primary**	Docker image / Compose / Helm pods, or `java -jar` on a Linux VM/host	`/bin/sh -c`	Default path in repo samples. Use Linux paths and shell syntax in job payloads.
Windows	**Yes**	**Manual**: install JRE/JDK 21, copy agent JAR, run as console, Task Scheduler, or a service wrapper (e.g. NSSM / WinSW). No MSI in-repo yet.	`cmd.exe /c`	Use Windows paths and commands (`.bat`, `powershell -File`, etc.). Pin jobs with preferred agent to the Windows host name.
Other OS (macOS, Unix-like JVM hosts)	**Possible**	Same JAR + Java 21+ where the vendor provides a JRE; manual process management	Non-Windows path uses `/bin/sh -c`	Works when the host has a POSIX shell and can run your scripts/binaries. Not a separately packaged product tier—validate per platform (AIX/z/OS etc. only if a suitable Java runtime exists).

All job types on any agent host

- **NOOP** — same everywhere (plumbing / smoke tests).
- **COMMAND** — runs on the agent host's OS (shell above). Payload must match that OS.
- **HTTP** — GET from the agent host's network (useful for "call from that site").
- **GIT_RELEASE** — downloads into the agent's cache and executes; ship an asset built for that OS/arch (Windows `.exe` vs Linux binary).

Linux agent (typical lab / K8s)

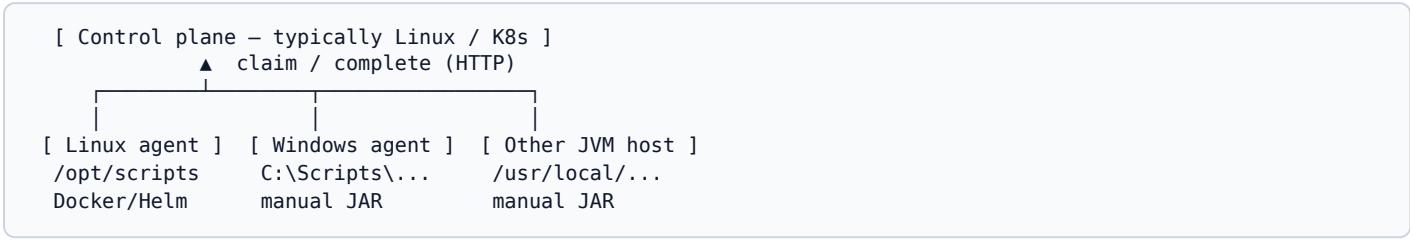
- Compose and Helm start Linux containers that already point at the control plane.
- Or on a bare Linux host: `java -jar automation-agent-*.jar --server http://control-plane:8080 --name linux-app-01 ...`

Windows agent (teams with Windows-only processes)

1. Control plane in `AUTOMATION_EXECUTION_MODE=agent` (Compose lab already uses this).
2. Build or obtain `agent/target/automation-agent-*.jar` (`(mvn -pl agent -am -DskipTests package)`.
3. On the Windows machine: install **Java 21**, place the JAR, set env (or flags): `AUTOMATION_SERVER`, `AUTOMATION_USER` / `AUTOMATION_PASSWORD`, stable `AUTOMATION_AGENT_NAME` (e.g. `win-app-server-01`), optional `AUTOMATION_ARTIFACTS_CACHE_DIR`.
4. Start: `java -jar automation-agent-*.jar`. Confirm the agent appears Online in the UI.
5. Create jobs with **preferred agent** = that name and Windows-native COMMAND payloads (e.g. `C:\Scripts\start.bat` or PowerShell `-File`).
6. Keep it running via Task Scheduler or a service wrapper for production-like always-on.

```
REM Example Windows env (lab values — use real secrets in production)
set AUTOMATION_SERVER=http://control-plane.example.com:8080
set AUTOMATION_USER=operator
set AUTOMATION_PASSWORD=********
set AUTOMATION_AGENT_NAME=win-app-server-01
set AUTOMATION_CAPACITY=5
java -jar C:\automation\automation-agent-0.1.0-SNAPSHOT.jar
```

> **Do not mix OS syntax across agents**
> A Linux-style payload (`/opt/app/start.sh`) will fail on a Windows agent, and `C:\Scripts\...` will fail on Linux. Prefer named agents per host class (e.g. `linux-batch-*`, `win-etl-*`). The UI does not yet label agents by OS type—use naming conventions and preferred agent.



6.4 Capacity vs parallelism (critical)

> **Capacity is not automatic multi-threading**
> Agent *capacity* is a **server-side claim limit** for that agent record. The **stock agent process is sequential**: one job at a time in its main loop. Setting capacity to 10 does *not* mean one process runs 10 COMMAND jobs in parallel. Parallel peak today ≈ **number of agent processes** (or a future multi-worker pool), each with capacity ≥ 1. This rule is the same on Linux and Windows.

GOAL	PRACTICAL APPROACH TODAY
More concurrent jobs on one machine	Run **N agent processes** (or pods) on that host, each capacity ≥ 1
Run only on a specific host / OS	Preferred agent on the job definition; ensure that agent is online
Company start/status/stop scripts	Install scripts on the agent host; model as COMMAND jobs in a workflow

6.5 Job assignment

- Server assigns claimable work with optional preferred-agent affinity.
- Without preference, online agents with remaining capacity can claim (round-robin style selection among eligible agents)—so always set preferred agent when Linux and Windows agents share one control plane.
- Offline / missing heartbeat agents stop receiving new claims.

7 Security & roles

ROLE	TYPICAL ACCESS
VIEWER	Read dashboards, definitions, executions
OPERATOR	Run jobs, manage schedules/agents as configured, agent Basic auth
ADMIN	User management, full configuration surface

MODE	USE
`local`	Form login + Basic only (default lab)
`hybrid`	Local accounts + OIDC SSO together
`oidc`	SSO-primary deployments

Details: `docs/SECURITY.md` and `docs/SECURITY-ANALYSIS-2026-07.md`.

8 Day-2 operations cheat sheet

TASK	HOW
Health probe	`GET /actuator/health/liveness`
List jobs (API)	`curl -u admin:… '/api/v1/jobs?page=0&size=20&q=…'`
Failed executions	UI Executions → Failed only, or `/api/v1/executions/jobs?status=FAILED`
Restart workflow step	Failed workflow execution detail, or restart API with `fromStepOrder`
Cancel / kill	Execution detail actions (or API) for running/queued work
Email on fail	Configure SMTP + notification flags (off by default)

Related documents

- `docs/CAPABILITIES.md` — full capability matrix vs Automic-class suites
- `docs/tech-stack-and-product-direction.pdf` — stack + F500 gaps + Just Run It
- `docs/platform-architecture-and-operations-guide.pdf` — deeper architecture/ops
- `docs/SESSION-BRIEF-2026-07.md` — current MVP snapshot and roadmap candidacy
- `docs/GIT_RELEASE_JOBS.md` — GIT_RELEASE job type

Enterprise Automation Platform — User Guide · Screenshots from lab UI · Just Run It: define work, run it where files live, schedule it, observe failures, secure it.