ETL Orchestrator Documentation

Rook IT ApS — v1.0.0

Rook IT ApS

Table of contents

1. ETL Orchestrator

A web app that orchestrates ETL Engine pipelines: upload Rabbit-in-a-Hat mapping files (RiaHs), wire them into a DAG by what they read and write, run them against an OMOP CDM, and watch every step unfold live.

The orchestrator replaces the hand-written shell scripts and Excel sheets that glue multi-RiaH ETL loads together today. One web UI for configure + run + history.

Get started → **Build your first pipeline →**

Download docs (PDF)

1.1 What it does

- **Pipeline configuration.** Upload one RiaH per step, attach any Usagi / CSVMap lookup files, set step ordering. The orchestrator auto-wires the DAG by reading each RiaH — `GetCDMTableId(@table)` becomes a consume edge to whichever sibling step stores `index/<table>`; multiple writers of the same `globalSeq/<seq>` get serialized so they don't race.
- **Environments.** Each `(pipeline, env)` pair is a separate target: source DB, CDM DB, license, log level, per-env `etl.conf` overrides, engine env vars + image overrides. Run the same pipeline against `dev` and `prod` with different connections — no copy-paste.
- **Running.** Live DAG view with per-step status colours, per-table row counts streaming as the engine emits them, full engine log capture, abort signal that crosses worker boundaries.
- **Selective re-run.** Re-run only the steps that share a CDM table with your target — the design-correct **re-run cascade**. Pre/post automation stay out of scope; unrelated dimension data is untouched.
- **Save / Load.** Export a pipeline (or an env) as a single bundle file (`.etlopipeline` / `.etloenv`) and load it on another instance — perfect for backup, sharing, or moving an env between machines.

1.2 Who it's for

People running OHDSI OMOP CDM loads from one or more source-data extracts, using the Rook IT ETL Engine to do the per-step mapping work. If today you hand-edit `run-N.sh` scripts and `etl.conf` files, this gives you a configure-once UI on top.

1.3 Quick links

- Install in 60 seconds
- Build your first pipeline
- Re-run cascade — what it is and why
- Bundle formats (`.etlopipeline`, `.etloenv`)

2. Getting Started

2.1 Getting Started

Two pages here:

1. **Installation** — get the orchestrator running on a host (docker or podman).
2. **First Pipeline** — create a pipeline, upload a RiaH, define an environment, run.

If you just want to play with the demo: install first, then in the UI hit **Pipelines → ⤒ Import** and upload the synthmas bundle that ships in the release.

2.2 Installation

The release ships as a single `.tar.gz` bundle containing the docker image, install + lifecycle scripts, and a README. Works with **docker** or **podman** out of the box.

2.2.1 Requirements

OS	Linux x86_64
Runtime	docker ≥ 20.10 **or** podman ≥ 4.0
Disk	~1 GB free for the image; data dir grows with usage (the synthmas demo data is ~ 500 MB)
Port	`8000` free (override via `ORCHESTRATOR_LISTEN_PORT`)

2.2.2 Install

```
tar xzf etl-orchestrator-X.Y.Z.tar.gz
cd etl-orchestrator-X.Y.Z
./bin/install-etl-orchestrator.sh
```

When it finishes you'll see a block with the URL and an **initial admin password** (random, generated once on first boot — copy it before clearing your terminal):

```
================================================================
  ETL-Orchestrator 0.7.0 is running.
================================================================
  URL:        http://your-host:8000/
  Local URL:  http://localhost:8000/

  Initial admin login:
    username: admin
    password: <random>
  (You'll be asked to change it on first sign-in.)
================================================================
```

Open the URL, sign in as `admin` with the printed password, and you'll be forced through a password change before reaching the app.

2.2.3 Final install step: upload the license

The orchestrator boots and lets you sign in without a license, but it **can't actually run a pipeline** without one — every engine spawn needs a valid license file. So before you do anything else:

1. Make sure you have a `license.lic` file (request a trial at https://rook-it.com/licenses).
2. **Settings → License → Upload** → pick your `.lic` file.
3. The page now shows the licensed-to name + expiry.

That's the install done. You're now free to import / build pipelines, create environments, and run.

2.2.4 Customising port / data dir / container name

All three are env-var overrides on the installer:

```
ORCHESTRATOR_LISTEN_PORT=9000 \
ORCHESTRATOR_DATA_DIR=/srv/etlo \
```

```
ORCHESTRATOR_CONTAINER_NAME=etlo-staging \
    ./bin/install-etl-orchestrator.sh
```

The installer persists your chosen values to `bin/etlo.config` so the lifecycle script (`run-etl-orchestrator.sh`) finds the right container without you re-passing env vars.

2.2.5 Upgrading

Just re-run `./bin/install-etl-orchestrator.sh` with the newer tarball. The script removes the old container, loads the new image, restarts. Your data dir is untouched, so pipelines / envs / run history all carry over.

If the new release includes a schema change, the entrypoint's bootstrap step applies the alembic migration on first start automatically.

2.2.6 Lifecycle

After install, manage the container with:

```
./bin/run-etl-orchestrator.sh status      # container + /healthz
./bin/run-etl-orchestrator.sh logs        # follow logs
./bin/run-etl-orchestrator.sh stop        # data preserved
./bin/run-etl-orchestrator.sh start
./bin/run-etl-orchestrator.sh restart
./bin/run-etl-orchestrator.sh uninstall   # remove container; data + image kept
```

2.2.7 What the installer does, in order

1. **Detects docker or podman.** Hard error if neither is installed; clear error if the daemon isn't reachable from the current user (you need to be in the docker/podman group, or root).
2. `$tool load -i data/etl-orchestrator.docker` then re-tags as `:latest` in the same repo namespace the image landed in.
3. **Creates the data directory** (defaults to `/var/lib/orchestrator`).
4. **Probes the host for a container socket** (`/var/run/docker.sock`, `/run/podman/podman.sock`, `$XDG_RUNTIME_DIR/podman/podman.sock`) and bind-mounts the first hit to `/var/run/docker.sock` inside the container. The orchestrator uses it to spawn engine containers for each pipeline step.
5. **Removes any existing container of the same name** (the upgrade path).
6. `$tool run -d --restart unless-stopped` with port + volume + socket mounts; environment vars stripped to the essentials.
7. **Waits for** `/healthz` (up to 60 s) so failures surface immediately.
8. **Greps the bootstrap log for the initial admin password** and prints URL + password in the block above.
9. **Writes** `bin/etlo.config` with the chosen settings so the lifecycle script auto-loads them.

2.2.8 Uninstalling completely

```
./bin/run-etl-orchestrator.sh uninstall              # removes container
docker rmi rook-it.com/etl-orchestrator:latest       # frees ~ 300 MB
rm -rf /var/lib/orchestrator                          # DESTROYS ALL STATE
```

2.3 Your First Pipeline

End-to-end walkthrough — create a pipeline, add a step with a RiaH, define an environment, run.

2.3.1 1. Create a pipeline

Pipelines → + New.

Fields:

Field	Purpose
Name	Globally unique; appears in the menubar picker.
OMOP version	The CDM version every RiaH in this pipeline must target. Uploaded RiaHs are rejected if their `cdmDb.dbName` (`CDMV5.4` etc.) doesn't match.
Engine image	Default container image to spawn for steps (e.g. `etl-engine:latest`). Per-env override available later.

2.3.2 2. Add a step

Open the new pipeline (Configure mode). The DAG canvas shows the fixed **Pre-automation** and **Post-automation** nodes. Click **Add step**, name the step (e.g. `location`), and you land on its Configure page.

Upload three things (only the RiaH is mandatory):

- **RiaH** (`.json.gz`, exported from Rabbit-in-a-Hat). The orchestrator parses it to auto-populate:
- **expected tables** (every CDM table the RiaH writes via `tableToTableMaps`)
- **produces** (named `StoreSequence` / `StoreIndexMap` artifacts)
- **consumes** (cross-step `GetCDMTableId(@<table>)` lookups, resolved against sibling steps' `produces`)
- **Usagi** CSV(s) — vocab mapping lookups the RiaH's `UsagiMap` rules reference. Usagi files are plain CSV in a specific column layout (`sourceCode`, `sourceName`, `sourceFrequency`, `targetConceptId`, `mappingStatus`, …) — the same format the OHDSI Usagi tool exports.
- **CSVMap** csv(s) — plain CSV lookups the RiaH's `CSVMap` rules reference.

The DAG view updates the moment the upload succeeds: the new node appears, and any cross-step `consumes` show up as arrows from the producer to the new node.

2.3.3 3. Define an environment

Each pipeline targets one or more **environments** (= "where the source data lives + where the CDM lives + what license to use"). Add one:

Pipeline → Environments → + New.

Tabs:

- **Source database** — `csv` (a per-pipeline source set) or one of the RDBMS backends (postgres, mysql/mariadb, sqlserver). For RDBMS, fill in host/port/db/schema/user/password.
- **CDM database** — same shape; `csv` writes per-table CSV files to a per-env directory.
- **Settings** — log level, vocab cache size, worker threads, optional engine image override.

When source is `csv`, you'll need a **source set** — a named collection of CSV files at the pipeline scope. Upload them via **Pipeline → Source sets**.

2.3.4 4. Run

Switch the env's URL to Run mode (`/run`). Press ▶ **Run pipeline**. The DAG starts colouring per step:

- **white border** = not yet run.
- **blue** = currently running.
- **green** = success (every expected CDM table loaded).
- **amber** = partial (engine exited 0 but some expected tables didn't load).
- **red** = failed.
- **grey** = skipped (intentional — e.g. steps outside a re-run cascade, or `post_automation` when nothing it would produce is enabled).

The right-side strip lists currently-running steps with per-table progress bars. Each step's full state is one click away — tap the node, the modal opens with status, per-table row counts, the engine log (⬇ Log to download), and the **Rerun (cascade)** button.

2.3.5 5. Make a change, re-run just the affected steps

This is where the orchestrator earns its keep. Say your `conditions` RiaH had a bug and you've fixed + re-uploaded it. Right-click `conditions` in the DAG → **Rerun (cascade)**. A confirm dialog tells you exactly which steps will be re-run (the cascade — every step that shares a CDM table with `conditions`, transitively) and which CDM tables will be wiped before the re-run. Click through and only those steps run; everything else stays as it was.

→ How the cascade is computed.

2.3.6 6. Save a pipeline you're happy with

Configure → ⇩ Save. Downloads a `.etlopipeline` bundle (zip) containing the pipeline config, every step file (RiaH/Usagi/CSVMap, sha256-deduped), all wiring (resolved by step name so it re-links cleanly), and optionally the vocabulary zip. Hit **Pipelines → ⇧ Import** on another instance (or on the same one as a duplicate) and the bundle recreates the whole pipeline in one click.

→ Bundle formats.

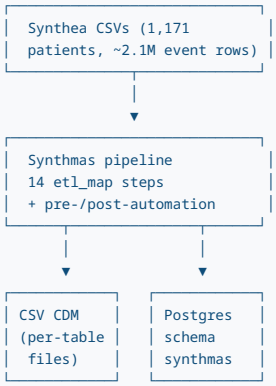
3. Demo (Synthmas)

3.1 Synthmas demo

End-to-end walk-through: get a running ETL Orchestrator with the Synthmas pipeline loading the Synthea synthetic-patient dataset into an OMOP CDM.

You'll do this **twice** — once with a CSV CDM target (self-contained, no database required) and once with a PostgreSQL CDM (closer to production: FK enforcement, downstream OHDSI tool compatibility). Both runs produce **row-for-row identical CDM tables** on engine 1.4.1+ — the parity isn't approximate, it's exact.

3.1.1 What you'll build



The demo covers all of:

- License upload + Settings tour.
- Athena vocab upload + what tables it must contain.
- Source CSVs (Synthea generation or download).
- CSV environment + run.
- PostgreSQL environment + run (same output, RDBMS-shaped for OHDSI tools + FK enforcement).
- Reading the DAG, the ribbon bar, the per-step modal.

3.1.2 Prerequisites

- A Linux host with **docker** or **podman** installed.
- Internet egress (initial image pull, Athena download, optional Synthea generation).
- For the PostgreSQL run: a reachable PostgreSQL ≥ 14 instance you can create a schema in (the demo creates `synthmas`).

3.1.3 Steps

1. **Setup** — request a temp license, install the orchestrator, log in, import the Synthmas pipeline.
2. **Vocabulary** — what Athena bundle you need + how to upload it.

3. **Source data** — get the Synthea CSVs.
4. **Run on CSV** — create the CSV env, kick off the pipeline, watch it complete.
5. **Run on PostgreSQL** — create the schema + the RDBMS env, run again.
6. **Reading the run-mode UI** — DAG colors, ribbon bar, per-step modal, what the numbers mean.

3.2 Setup

3.2.1 1. Request a trial license

The orchestrator drives the **ETL Engine**, which is licensed software. You need a license with the **ETL** feature to run the Synthmas demo (it uses `StoreIndexMap` + `StoreSequence` rules that aren't available in the free tier).

To get a **5-day free trial license**:

1. Go to **https://rook-it.com/licenses** and create an account.
2. Request your one-time 5-day trial. You'll receive a `license.lic` file by email.

If 5 days isn't enough — for example, you're evaluating a longer production pilot — write to **licenses@rook-it.com**. We'll arrange a short conversation about your use case first, then issue a longer license tailored to it. We don't issue extended licenses unattended, just so we can match the license terms to the work you're actually doing.

3.2.2 2. Install the orchestrator

Download the latest release tarball from your Rook IT release page (installer comes inside):

```
tar xzf etl-orchestrator-<version>.tar.gz
cd etl-orchestrator-<version>
sudo bin/install-etl-orchestrator.sh
```

The installer:

- Probes your host for docker / podman + their sockets.
- Loads the image from `data/etl-orchestrator.docker`.
- Starts the container with `--restart unless-stopped`.
- Waits for `/healthz` to come up.
- Prints **a randomly-generated initial admin password to stdout**. This is shown **once** — copy it somewhere safe before closing the terminal.

Default URL: `http://localhost:8000/`. Log in as `admin` with the random password the installer printed.

> **Different host port**
>
> Set `ORCHESTRATOR_LISTEN_PORT=8765` (or similar) in the environment before running `install-etl-orchestrator.sh` — both the container and the host port mapping pick it up.

3.2.3 3. Upload the license

This is the last install step before you can do anything else — without a license, the orchestrator boots and lets you in, but every engine spawn will fail with a clear "no valid license" error.

In the UI: **Settings → License → Upload** `license.lic`.

A valid license with the ETL feature unlocks the StoreIndexMap / StoreSequence rules every Synthmas RiaH uses. The license page shows the licensed-to name + expiry once it's installed.

3.2.4 4. Import the Synthmas pipeline

Download the pipeline bundle:

⤓ synthmas-pipeline-1.0.0.etlopipeline

In the UI: **Pipelines → Import** → select the `.etlopipeline` file → **Import**.

You should now see a "Synthmas" pipeline on the **Pipelines** page with 14 etl_map steps wired into a DAG of cross-step `StoreIndexMap` / `StoreSequence` edges.

3.2.5 5. (Optional) Switch on TLS

If the orchestrator is reachable beyond `localhost`, **Settings → TLS** to pick a mode:

- **Self-signed** for a quick lab cert (Rook IT branded; valid 13 months).
- **Self-managed** to upload a real cert + key + chain.

Either way the change takes effect on the next container restart (`bin/run-etl-orchestrator.sh restart`); gunicorn doesn't hot-reload TLS.

→ Next: **Vocabulary**.

3.3 Vocabulary

OMOP mappings (`VocabMap("SNOMED")` etc.) resolve source codes against the OMOP **Athena** vocabulary tables. You need to download those once and upload the zip to the orchestrator.

3.3.1 What the bundle must contain

The Synthmas mappings exercise these vocabularies:

Vocabulary	Used by
SNOMED	conditions, procedures, allergies, devices
LOINC	observations (clinical step), measurements
RxNorm	drug_exposure
CVX	drug_exposure (immunizations)

You can include more (LOINC Component Hierarchy, ATC, etc.) without hurting — the engine indexes the whole vocab on load and unused entries just sit there.

Required tables inside the Athena zip:

- `CONCEPT.csv`
- `CONCEPT_RELATIONSHIP.csv`
- `CONCEPT_ANCESTOR.csv`
- `CONCEPT_SYNONYM.csv`
- `CONCEPT_CLASS.csv`
- `VOCABULARY.csv`
- `RELATIONSHIP.csv`
- `DOMAIN.csv`
- `DRUG_STRENGTH.csv`

These are exactly the file set the engine's `vocabulary_loader` expects (filename prefix `vocabulary_download_v5_*.zip`).

3.3.2 How to download

1. Create an account on **https://athena.ohdsi.org/**.
2. **Download** → tick the vocabularies above → start the download.
3. Athena emails you when the bundle is ready (usually within an hour).
4. Click the email link, save the resulting `.zip` (typically named `vocabulary_download_v5_<date>_<id>.zip`).

Total uncompressed size: ~3.3 GB; compressed ~600 MB. The orchestrator keeps it as a single blob and never re-extracts at runtime.

3.3.3 Upload

In the UI: **Pipelines → Synthmas → Configure → Vocabulary** → upload the zip. Give it a version string (the Athena release date works: `v5_2026-01-15`). The orchestrator computes a content hash and won't re-extract if the same content is uploaded again.

Notes:

- The vocab is **pipeline-scoped** — once uploaded, every env using this pipeline shares it. Switch vocab by uploading a new version; envs see the change on their next pre-automation run.
- Loading time into the postgres CDM is **~10 minutes for the full Athena bundle** on a modest box; the CSV CDM path doesn't load into a database — it serves the vocab files directly from disk.

→ Next: **Source data**.

3.4 Source data

The Synthmas demo runs against **Synthea** synthetic-patient CSVs — open data with no PHI, perfect for demos. You need ~1,000 patients (the default Synthea run produces 1,171 with the recommended seed).

3.4.1 Option A — use the pre-generated bundle from the engine demos

Easiest: the etl-engine repo ships a small Synthea bundle under `demos/synthmas-single-RiaH/etc/etl-engine.d/db/`. Copy those CSVs into a local folder:

```
git clone https://github.com/rook-it/etl-engine
cp -r etl-engine/demos/synthmas-single-RiaH/etc/etl-engine.d/db ./synthea-source
ls synthea-source/
# allergies.csv  careplans.csv  claims.csv  claims_transactions.csv
# conditions.csv  devices.csv  encounters.csv  imaging_studies.csv
# immunizations.csv  medications.csv  observations.csv
# organizations.csv  patients.csv  payer_transitions.csv  payers.csv
# procedures.csv  providers.csv  supplies.csv
```

3.4.2 Option B — generate your own with Synthea

If you want a larger / different cohort:

```
# Synthea v3.x — uses Java 11+
git clone https://github.com/synthetichealth/synthea
cd synthea
./run_synthea -p 1000 -s 42 \
              --exporter.csv.export true \
              --exporter.csv.folder_per_run false \
              --exporter.fhir.export false
mv output/csv ../synthea-source
```

`-p 1000` = 1,000 living patients (plus deceased who lived during the sim window); `-s 42` = deterministic seed; the `--exporter.fhir.export false` flag skips the large FHIR JSON output we don't use.

3.4.3 Upload to the orchestrator

In the UI: **Pipelines → Synthmas → Configure → Source sets → New** → name it (`synthea-1k`), upload the CSVs (multi-select the whole folder).

You can keep multiple source sets per pipeline (different patient cohorts / different Synthea releases) and pick which one an env reads from on its environment page.

→ Next: **Run on CSV**.