COMPLETE WALKTHROUGH · NOVEMBER 2026

How DataNivra works: a complete walkthrough

An end-to-end guide — from sign-up to CI/CD automation — written in plain language for everyone on your team. No engineering background required.

Windows · macOS · Linux | Agent 0.6.1 | CLI 0.3.0 | Parts A–I fully covered · Parts J–L included

CONTENTS

A Tour without installing anything
B Create account & invite approver
C Hosted sandbox: nine missions
D Install Docker & the agent
E Practice databases & connect
F Industry pack, sensitive data, masking
G Build: request, approve, deliver
H Synthetic data (custom schemas)
I CLI, GitHub Actions, AI assistants
J Cloud sources (optional)
K Governance & audit evidence
L Plans, pricing & how to buy

What DataNivra is, in plain words

Every software team needs realistic data to test with. The fastest way to get it has always been to copy the production database into a test environment. That copy now contains real customers' names, card numbers or medical records, sitting in a place that was never meant to protect them.

DataNivra is a test data management platform built the other way round. Its tagline is "Realistic test data. Production data stays home." It has two parts:

- **DataNivra Cloud** (the website, the console and the API) is where people define rules, approve requests and see evidence. It only ever handles descriptions of your data: table and column names, counts, rule versions, job status and checksums.
- **The DataNivra agent** is a small program you run inside your own network. It does all the work that touches actual rows: reading, masking, subsetting, generating synthetic data, checking quality and delivering the result into your test databases.

> **Why it matters**
>
> Your production rows are processed only by the agent inside your network. DataNivra Cloud never sees your customers' records — a breach of their cloud would not expose your data.

Before you start: what you need

- A work email address for your own account
- A second person or second email address you control, to act as approver (DataNivra refuses to let one account approve its own work)
- A Windows 10 or 11 PC with administrator rights and about 10 GB free disk space
- A GitHub account and a test repository you can change (Part I)
- Optional: access to Amazon S3, Google BigQuery or Databricks (Part J)
- About two working days in total — you can stop and restart at any point

> **About the approver account**
>
> For this evaluation it is fine to use a second mailbox you control. In real use, every person should have exactly one account; otherwise one person could approve their own requests, which breaks the protection.

A Look around without installing anything

~45 min

A1 The website

1. Open **www.datanivra.com** in your browser.
2. Read the home page. Note the two ways to work: the console for your data team, and the Test Data API for developers and AI agents.
3. Click **Product → How it works** and **Architecture**. The architecture page shows exactly what crosses from your network to DataNivra Cloud.
4. Click **Security**, then open **Trust and verification** from the footer.
5. Click **Industry packs** and open the pack closest to your business (for example Financial services or Healthcare & life sciences).
6. Click **Pricing** and read the plans. Part L explains them.

A2 The Synthetic Playground (runs in your browser)

1. In the footer, under Product, click **Synthetic playground** (or go to **www.datanivra.com/demo**).
2. Pick an industry and generate a few rows. Everything runs in your browser; nothing is sent anywhere.

> **Expected result**
>
> You see realistic-looking but entirely made-up records, ready to download.

A3 Runnable examples & public repository

1. On the home page, find **Explore runnable examples** and click **View runnable examples**.
2. Browse the scenarios: healthcare, banking, insurance, retail, CI/CD recipes and the AI-assistant example.
3. The same files are public at **github.com/DataNivra/datanivra-examples** (MIT licence).

B Create your account & invite an approver

~15 min

B1 Sign up

1. On **www.datanivra.com** click **Start free**, then click **Start free** again on the next page.
2. Enter your name and work email, accept the Terms of Service and Privacy Policy, and submit.
3. Open the email from DataNivra (sender **hello@datanivra.com**) and click **Confirm my email**. The link works once and expires after 30 minutes.
4. Name your organization. Write down the short tenant identifier — you will need it in Part I. **You cannot change it later.**
5. Click **Confirm**. You are now the Organization owner.

> **Expected result**
>
> You land in the DataNivra console at **console.datanivra.com**. Your organization starts on the Free plan with a one-time 30-day Pro Trial, which unlocks everything in this guide. No credit card needed.

> **How to sign in later**
>
> Go to **console.datanivra.com** and choose **Email me a sign-in link**. DataNivra never asks for a password. Sessions end after inactivity — just request a new link.

B2 Invite your approver

1. In the left menu, open **Administration → Users & Roles**.
2. Click **Invite user**, enter the second email address and choose the role **Security admin**.
3. Ask your colleague (or open the second mailbox) to accept the invitation and sign in with an email link. Use a separate browser or a private window for the second account so both can be signed in at the same time.

> **Why this matters**
>
> Throughout this guide, DataNivra will refuse to let the person who created something also approve it. This is called separation of duties. When you see "Separation of duties" as an error — that is the expected, correct behaviour, not a fault.

C The hosted sandbox: nine guided missions

1–2 hours

The sandbox gives you a synthetic company estate and an agent run by DataNivra. Nothing to install. Each mission completes only when the product has really done the work.

1. In the left menu, open **Overview → Sandbox** and click **Start the sandbox**.
2. Wait one or two minutes while the sandbox is prepared.
3. Work through the nine missions in order.

#	MISSION	WHAT IT PROVES
1	Discover the synthetic estate	DataNivra can list tables and relationships without reading values.
2	Review sensitive-data findings	Personal and financial columns are detected automatically.
3	Create a masking policy from the pack template	Policies are versioned and need a second person's approval.
4	Request a masked subset	A small, consistent slice of the data is requested and approved.
5	Break it safely	A dataset with an unprotected sensitive column is refused.
6	Get a dataset certified	Automatic quality and privacy gates pass; a signed certificate is issued.
7	Provision to the sandbox QA environment	Only certified data can be delivered.
8	Inspect the evidence	You can download proof of what was done and verify it.
9	Schedule a refresh	Test data can refresh on a calendar without manual work.

> **Approvals in missions 3 and 4**

> These must be done by your approver account (from Part B2). If you try to approve with your own account, DataNivra refuses with "Separation of duties" — that is correct behaviour.

D Install the DataNivra agent on your Windows PC

~45 min

In real use the agent runs on a server inside your network, next to your databases. For this evaluation your PC plays that role. The agent is a Linux container, so on Windows you run it inside Docker Desktop using a Linux environment called WSL (Windows Subsystem for Linux).

D1 Install WSL and Docker Desktop (one time)

1. Click the Start menu, type **PowerShell**, right-click **Windows PowerShell** and choose **Run as administrator**.
2. Type `wsl --install -d Ubuntu` and press Enter. When it finishes, restart your PC.
3. After the restart an Ubuntu window opens and asks you to create a Linux user name and password. Choose anything you will remember — this password is only for your PC.
4. Download Docker Desktop for Windows from **docker.com** and install it with the default options.
5. Open Docker Desktop, go to **Settings → Resources → WSL integration**, switch on Ubuntu and click **Apply & restart**.
6. Open the **Ubuntu** app from the Start menu. This black window is where you type every command in this guide. Type `docker version` and press Enter.

> **Expected result**
>
> `docker version` prints a Client and a Server section without errors. If it says it cannot connect, make sure Docker Desktop is running (look for the whale icon in the taskbar) and try again.

> **Pasting into the Ubuntu window**

> To paste, right-click inside the Ubuntu window (or press Ctrl+Shift+V). Paste one block at a time and press Enter. If a command asks for your password, type your Linux password from step 3 — nothing appears on screen while you type, which is normal.

D2 Create the practice network

The agent and your practice databases will talk over a private Docker network. Create it once:

```
docker network create datanivra-lab
```

> **If you see "network with name datanivra-lab already exists"**
> That is fine — the network already exists from a previous run. Continue to D3.

D3 Create a one-time enrollment token in the console

1. In the console, open **Connect → Agents** and click **Install agent**.
2. Choose **Docker** as the install method.
3. Enter an Agent name, for example **ceo-laptop**. Use exactly this name again in step D4.
4. Click **Create enrollment token** (15 minutes validity is enough) and copy the token. **It is shown only once.** Keep this console page open: it will show the agent's heartbeat once running.

D4 Start the agent

Paste the following into the Ubuntu window. The first command creates a private folder for secrets; the second asks you to paste the token; the third starts the agent. If you chose a different agent name, replace `ceo-laptop` in both places.

```
sudo install -d -m 0700 -o 10001 -g 10001 "$HOME/datanivra-secrets"
read -rsp 'Enrollment token: ' T && echo && printf %s "$T" | sudo install -m 0600 -o 10001
docker run -d --name datanivra-agent --restart unless-stopped --network datanivra-lab \
  --read-only --tmpfs /tmp --cap-drop ALL --security-opt no-new-privileges:true \
  -e DATANIVRA_AGENT_CONTROL_PLANE_URL=https://api.datanivra.com \
  -e DATANIVRA_AGENT_AGENT_NAME=ceo-laptop \
  -e DATANIVRA_AGENT_SECRETS_DIR=/etc/datanivra/secrets \
  -e DATANIVRA_AGENT_ENROLLMENT_TOKEN_REF=file://enrollment-token \
  -v "$HOME/datanivra-secrets:/etc/datanivra/secrets:ro" \
```

```
  -v datanivra-agent-state:/var/lib/datanivra-agent \
  ghcr.io/manuelbomi/datanivra-agent@sha256:642b51b13d077cf0bb4e3d6bbb857b49d5c38712dbace6
```

Important: verify the agent joined the network

After the `docker run` command finishes, run this check immediately. The agent must be on the same Docker network as your practice databases or it will not be able to reach them in Part E.

Run this command:

```
docker network inspect datanivra-lab --format '{{range .Containers}}{{.Name}} {{end}}'
```

Expected result

You should see `datanivra-agent` listed. It may also list `lab-source` and `lab-target` if you have already created them.

If datanivra-agent is NOT in the list

The agent container did not join the network automatically. Connect it manually:

```
docker network connect datanivra-lab datanivra-agent
```

Then run the inspect command again to confirm it now appears.

Expected result in the console

Within a minute, the console page shows a heartbeat and the agent appears under **Connect → Agents** as **ACTIVE**. In the Ubuntu window, `docker ps` lists `datanivra-agent` as healthy.

If something goes wrong

- `docker logs datanivra-agent` shows what the agent is doing.
- If the token expired: create a new token (D3), then run `docker rm -f datanivra-agent` and repeat D4.
- The built-in health check `docker exec datanivra-agent datanivra-agent doctor` lists every check with PASS, WARN or FAIL.

Deploying on a Linux server? Use the quick installer

Part D walks through Docker Desktop on Windows. For a **bare-metal or VM Linux host** (the typical production setup), the signed quick-install script handles all the steps — Docker Compose download, system user, secrets directory, and systemd service — in one command:

```
# Read the script first: https://www.datanivra.com/install.sh
curl -fsSL https://www.datanivra.com/install.sh | sudo bash
```

Or pin a version and pass the enrollment token non-interactively (CI, Ansible, Terraform user-data):

```
sudo DN_VERSION=0.6.1 DN_ENROLLMENT_TOKEN="<token>" \
  bash <(curl -fsSL https://www.datanivra.com/install.sh)
```

The installer creates the `datanivra` system user (UID 10001), writes the token to `/run/datanivra-secrets/enrollment-token` (mode 0600), and registers a systemd unit that starts the agent on boot. Useful commands after installation:

```
journalctl -u datanivra-agent -f   # live logs
systemctl status datanivra-agent   # service status
```

For Kubernetes, Helm, or cloud-provider managed infrastructure see the download packages.

E Create two practice databases & connect DataNivra ~45 min

Instead of touching a company system, you will create two small PostgreSQL databases in Docker: **lab-source** (filled with DataNivra's synthetic sample) and **lab-target** (an empty database that plays the role of a QA test environment).

E1 Run the setup script

1. In the Ubuntu window type `nano datanivra-lab-setup.sh` and press Enter. A simple editor opens.
2. Copy the whole script from Appendix A of this guide and paste it into the editor (right-click).

3. Press **Ctrl+O**, then Enter to save, and **Ctrl+X** to close the editor.

4. Run it: `bash datanivra-lab-setup.sh`

> **Expected result**
>
> The script prints five steps and ends with a table of row counts, for example `commerce.orders 12`, `crm.customers 5`. Ten tables in total. It also creates read-only and loader database accounts.

E2 Give the agent the connection details

DataNivra Cloud never receives database passwords. Instead you store small connection files next to the agent and tell the console only their names. Paste this whole block at once — it sources the passwords file and then writes the connection files in a single step:

```
. "$HOME/datanivra-lab/passwords.env"
printf '{"kind":"POSTGRESQL","host":"lab-source","port":5432,"dbname":"practice","user":"t
for env in qa synth ci; do
  printf '{"kind":"POSTGRESQL","host":"lab-target","port":5432,"dbname":"practice","user":
done
```

> **Paste the whole block at once**
>
> The first line sources the passwords file, so the lines that follow can use those passwords. If you paste each line separately, the later lines will have empty password variables and the files will be written incorrectly.

Verify the files were written correctly:

```
ls -la ~/datanivra-secrets/
```

> **Expected result**
>
> You should see four files: `lab-source.json`, `lab-target-qa.json`, `lab-target-synth.json` and `lab-target-ci.json`. All owned by uid 10001, permissions `-rw-------`.

Now run the doctor check to confirm everything is connected:

```
docker exec datanivra-agent datanivra-agent doctor \
  --source-ref file://lab-source.json \
  --connector POSTGRESQL \
  --target-ref file://lab-target-qa.json
```

Expected result

Among the checks you should see: `SOURCE_REACHABLE PASS`, `SOURCE_READ_ONLY PASS` and `TARGET_REACHABLE PASS`.

If you see SOURCE_REACHABLE FAIL or TARGET_REACHABLE FAIL (after the secret files are found)

This almost always means the agent container is not on the `datanivra-lab` Docker network, so it cannot reach the database containers by their names. Fix it with one command:

```
docker network connect datanivra-lab datanivra-agent
```

Then run the doctor check again.
Why this happens: The agent should have joined the network when it was started (Part D4), but Docker sometimes does not apply the `--network` flag correctly on first run, or the agent container was recreated without that flag. Connecting it manually is the right fix.

Understanding SKIP and DEPENDENCY_FAILED in doctor output

The doctor command runs checks in sequence. If an earlier check fails, checks that depend on it are shown as `SKIP` or `DEPENDENCY_FAILED` rather than being attempted. This is expected behaviour — it tells you which check to fix first.
For example: if `SOURCE_REACHABLE` fails, the check `SOURCE_READ_ONLY` will show `DEPENDENCY_FAILED` because there is no point testing read-only access on a database that cannot even be reached. Fix the root-cause failure (`SOURCE_REACHABLE`) and re-run; the downstream checks will then run normally.
A clean output should show only `PASS` lines. Any `WARN`, `FAIL`, `SKIP` or `DEPENDENCY_FAILED` line means something is not ready.

E3 Register the source in the console

Open **Connect → Data Sources** and click **Add source**. The registration wizard has five steps:

The 5-step registration wizard

1. **Connector type** — choose **PostgreSQL**. Set Source name to **Practice ERP**, Agent to your agent name, and declare the data as **Production data** (DataNivra applies the strictest rules to production sources).
2. **Read-only account** — the console shows the SQL commands to create a read-only database role. **For this lab the setup script already created the `tdm_reader` role in step E1**, so you can skip the SQL. If you ever register a real source, run the SQL that the console shows against that database before continuing. The commands look like this:

   ```
   CREATE ROLE tdm_reader WITH LOGIN PASSWORD 'your-secure-password';
   GRANT CONNECT ON DATABASE practice TO tdm_reader;
   GRANT USAGE ON SCHEMA public, commerce, crm, hr, finance TO tdm_reader;
   GRANT SELECT ON ALL TABLES IN SCHEMA public, commerce, crm, hr, finance TO tdm_reader
   ALTER DEFAULT PRIVILEGES IN SCHEMA public, commerce, crm, hr, finance
     GRANT SELECT ON TABLES TO tdm_reader;
   ```

3. **Secret reference** — enter `file://lab-source.json` (the connection file you wrote in E2).
4. **Validate** — click **Register and validate read-only access** and wait for **Read-only access verified**. This confirms the agent can reach the database and that the account is genuinely read-only.
5. **Discovery** — if prompted, review the list of fields that cross the boundary, tick **I reviewed what will be sent to DataNivra Cloud**, and click **Run metadata-only discovery**.

Discovery may run automatically

Depending on your version of the product, step 5 may happen automatically — DataNivra runs discovery as soon as read-only access is verified, without waiting for you to click. To check: open **Connect → Catalog**. If you see your five schemas and ten tables listed with their column details, discovery has already completed successfully. You do not need to run it again.

If you see "A source with this name already exists"

This happens if you clicked Back during registration — DataNivra already created the source before you went back. **Do not try to register it again.** Instead, go to **Connect → Data Sources** and click on the existing "Practice ERP" source. It is already registered; just continue from there.

Expected result

Under **Connect → Catalog** you now see five schemas and ten tables with their columns and relationships — but no actual data values, because DataNivra only received the structure.

E4 Register three test environments

1. Open **Trust → Provisioning** and click **New environment**.
2. Fill in: Environment name **lab-qa**, tier **QA**, provisioning agent (your agent), target secret reference `file://lab-target-qa.json`. Leave Ephemeral unticked. Save.
3. Repeat for **lab-synth** (tier **DEV**, `file://lab-target-synth.json`) and **lab-ci** (tier **QA**, `file://lab-target-ci.json`).

Expected result

The Provisioning page lists all three environments with your agent. The note "No certified datasets — Only certified datasets can be provisioned" is expected at this stage. You have not created any datasets yet. That happens in Part G.

F Protect: industry pack, sensitive data & masking policy

~30 min

F1 Activate industry packs

1. Open **Overview → Industry Packs**, choose **General enterprise** and click **Activate this pack**. Follow the steps on screen.
2. Also activate **Financial services**: Part I uses one of its scenarios.

The pack activation wizard has 12 steps

The activation wizard walks you through connecting a source, choosing settings and creating policies. Here is what to select at each key step:

- **Step 4 — Connect a source:** Select **Practice ERP (Postgresql, Production)** from the list. It will show "Read-only access verified; its agent can run this pack."
- **Step 5 — Entity model and scenario:** Accept the default entity model the pack suggests. For Financial Services, choose the Everyday Banking scenario.
- **Step 6 — Size, closure and masking:** See the values below.

- **Step 7 — Policies from templates:** Before clicking **Create 5 drafts from templates**, you must fill in the **Masking key reference** field. See the box below.

Step 7 — Masking key reference is required

The form will not let you proceed without a masking key reference. The key ensures the same real customer always masks to the same fake name across tables, so joins still work in test data.

First, find which directory your agent actually reads secrets from — it may differ from `~/datanivra-secrets/`:

```
docker inspect datanivra-agent --format '{{range .Mounts}}{{.Source}} → {{.Destination}
```

Look for the line ending in `→ /etc/datanivra/secrets`. The path before the arrow is your secrets directory. Common values are `/home/yourusername/datanivra-secrets` or `/run/datanivra-secrets`.

Write the masking key to that directory (replace the path with yours). The agent requires a specific **keyset format** — anything else causes the errors shown below:

```
# Use the path shown by docker inspect above — example uses /run/datanivra-secrets
KEY=$(python3 -c "import os, base64; print(base64.urlsafe_b64encode(os.urandom(32)).deco
printf '{"primary":1,"keys":{"1":"%s"}}' "$KEY" | sudo install -m 0600 -o 10001 -g 10001
```

Verify it:

```
docker exec datanivra-agent cat /etc/datanivra/secrets/masking-key
# Should print: {"primary":1,"keys":{"1":"base64url..."}}
```

Why this exact format — and what breaks if you get it wrong

- **Plain string** (e.g. the hex output of `openssl rand -hex 32`) → `SECRET_NOT_JSON_OBJECT`
- `{"key": "..."}` (wrong field names) → `KEYSET_INVALID`
- **Tink or JWK keyset JSON** → `KEYSET_INVALID`
- **Correct format:** `{"primary": 1, "keys": {"1": "base64url-32-bytes"}}` → success

The `primary` field (integer) selects which version is used when no version is specified. The `keys` object maps version numbers (as strings) to base64url-encoded key bytes of at least 32 bytes each. This design supports key rotation: add version 2, move `primary` to 2, keep version 1 for older datasets that must stay linkable.

Then type this into the **Masking key reference** field on the console:

```
file://masking-key
```

> **All secret files must go to the same directory**
>
> Every secret file (connection files, masking key) must be written to the directory your agent actually reads from — the one shown by `docker inspect`. Writing to `~/datanivra-secrets/` when the agent reads from `/run/datanivra-secrets/` causes SECRET_NOT_FOUND errors even though the file exists on your PC.

> **Step 6 — what to enter for Size, closure and masking**
>
> The tutorial does not specify these values clearly. Use these for the evaluation:
>
> - **What kind of dataset:** Hybrid (recommended) — this tests both masking and synthetic generation together
> - **Start the subset from:** Choose the customers table (or the financial services equivalent — whatever the dropdown shows as the root entity)
> - **Root records to keep:** 100 — small is fine for evaluation, the default 1,000 takes longer
> - **Relationship depth:** 3 — leave the default
> - **Scenario records to generate:** 100
> - **Target environment:** lab-qa

F2

Review sensitive-data findings

1. Open **Protect → Sensitive Data**.
2. The page shows a flat list of every detected column across all your connected sources, displayed as schema.table.column. There is no dropdown to filter by source name — all sources appear together in one list.
3. Look at the findings: names, email addresses, phone numbers, bank and tax identifiers, salaries. Each has a confidence level and a reason.
4. Confirm a few findings and dismiss one you disagree with. Every decision is recorded in the audit trail.

> **Why there is no source-name filter here**
>
> Protect → Sensitive Data is an org-wide view: DataNivra shows all detected sensitive columns from every source so you can review and act on them in one place. If you want to

browse the structure of one specific source, use **Connect → Catalog** instead, which is organised by source. Use your browser's Ctrl+F search on the Sensitive Data page to quickly jump to columns from a particular schema.

F3 Create and approve a masking policy

1. During pack activation (F1 step 7), click **Create from pack templates**. The new policies appear under **Protect → Policies**.
2. Open the new masking policy, review its rules, then click **Submit for review**.
3. Switch to your **approver account**, open the same policy and click **Review and approve**. Give a reason, for example Evaluation.

Must be approved by the other account

Your own account cannot approve a policy it submitted. Use your approver account (from Part B2) in a separate browser window or private tab.

What you have proved so far

By the end of Part F you have: a source connected and catalogued, three test environments registered, an industry pack active, sensitive columns identified, and a masking policy approved
by a second person. You are ready to build and deliver your first dataset in Part G.

G Build: request, approve, certify & deliver test data ~45 min

G1 Request a masked subset

Open **Build → Dataset Requests** and click **New dataset request**. The wizard has 12 steps. Here is what to enter at each one:

Step 1 — Business entity

- **Request title:** `General Enterprise QA dataset`
- **Business entity or domain:** type `general-enterprise` (listed as a known domain)
- **Industry pack:** General Enterprise

- **What kind of dataset:** Hybrid

Step 2 — Source scope

- **Source:** Practice ERP — this is where the source appears (not Step 1)

Step 3 — Subset size & strategy

- **Subset name:** `customers-100` (tells the audit trail "100 customers and everything related to them")
- **Business entity (root table):** `crm.customers`
- **How should records be selected:** Fixed count
- **Number of root records:** `100`
- **Relationship depth:** `3`
- **Keep both Include dependent records and Include referenced records ticked** — this pulls in orders, invoices and payments for those customers so foreign keys still work

Step 4 — Scenario coverage

- **Scenario:** Seasonal peak (GE_SEASONAL_PEAK) — the most representative normal scenario; gives you the richest dataset to inspect

Step 5 — Masking policy

- **Masking policy:** General Enterprise Enterprise standard PII masking — v1 (Approved)
- You will see the checksum pinned. The agent refuses the job if the policy is changed after you submit — that is the tamper-proof guarantee working correctly.

Step 6 — Target environment

- **Deliver to:** `lab-qa` — the QA environment you created in Part E4 specifically for masked subset delivery

Steps 7–9 — Refresh, retention, synthetic augmentation

- **Refresh cadence:** None (manual for evaluation)
- **Retention:** 30 days (default)
- **Synthetic augmentation:** leave at default

Step 10 — Certification policy

- **Certification policy:** General Enterprise certification: standard

Steps 11–12 — Estimated footprint & Review

- Step 11 shows an estimate of rows and storage — review it and click Next.
- Step 12 is the review screen. Add a **Justification** such as `Evaluation run — testing masked subset delivery to lab-qa`, then click **Submit**.

Then switch to your **approver account** and open **Build → Dataset Requests** to approve the request.

Expected result

The request runs on your agent. Under **Build → Datasets** a new dataset version appears and goes through certification. Open **Trust → Certification** to see each gate: policy coverage, masking completion, referential integrity, orphans, schema and row-count reconciliation.

G1b

Optional: entity resolution metadata columns

If you configured an **identity graph** and want to surface the cross-system entity linkage directly in your provisioned tables, check **Include entity resolution metadata** on the dataset request form (or add `include_entity_resolution_metadata: true` to your test-data profile so every invocation includes it automatically).

DataNivra appends two columns to every table that participates in the identity graph:

- `_dn_entity_resolution_key` — a stable, deterministic pseudonym that identifies the same real entity across tables and datasets. Use it as a join key in tests that exercise cross-system entity matching.
- `_dn_entity_resolution_sources` — a JSON array of the source system names that contributed identity evidence (for example `["ehr","billing"]`).

Both columns are `NULL` for rows not covered by the identity graph. Requires agent ≥ 0.6.2.

G2 The certificate and the evidence

1. Open **Trust → Certificates** and select the new certificate. You will see all 16 certification gates listed with Pass/Fail results.
2. The evidence files are stored inside your agent container (not downloadable from the console in the current version). The certification page lists their exact file paths and SHA-256 checksums under **Evidence for this report**. To copy them out of the container:

```
docker cp datanivra-agent:/var/lib/datanivra-agent/datasets/<dataset-id>/v1/ ~/datanivra-e
```

> **No "Download report" button in this version**
>
> The original guide mentions Download report (HTML) and Download evidence package (JSON) buttons — these do not exist in the current console. The evidence is stored inside your agent container at the paths shown on the certification page. The console page itself, with its SHA-256 checksums, serves as the auditable record.

G3 Deliver and check the result

1. In **Trust → Provisioning**, click **Provision a certified dataset**. Select the dataset version and **lab-qa** as the target, then click **Provision**.
2. The status shows **Queued**. Click the provisioning job link (not the browser refresh) to see the job progress. It should reach **Succeeded** in under a minute.
3. Check the masked data in the target database. In the Ubuntu window:

```
docker exec -it lab-target psql -U postgres -d practice -c "\dt qa_data.*"
docker exec -it lab-target psql -U postgres -d practice -c "SELECT * FROM qa_data.customer
```

> **Expected result**
>
> All 10 tables appear in `qa_data`. Customer names, emails, phone numbers and addresses are replaced with realistic fictional values. The `notes` column shows `[REDACTED]`. Dates, segments and other non-sensitive fields are preserved. All 959 rows (94 masked + 875 synthetic) are present with referential integrity intact.

H Synthetic data shaped like your own tables

~30 min

Industry packs cover 15 industries. If your systems look different, DataNivra can generate synthetic data from a description of your own tables. It never needs your values.

H1 Create a custom schema

1. Open **Automate → Custom Schemas** and click **New custom schema**.
2. Give it any name (for example `farm-ops`) and click **Insert example** to populate it with a sample 10-table structure, or paste your own JSON structure.
3. Click **Create custom schema**. The page returns to the list — **click the schema name** to open it and see its tables and relationships.

> **You must click the schema name after creating it**
>
> The console does not automatically navigate into the schema after creation. Click the name in the list to open it before proceeding.

H2 Create and approve a test data profile

1. Scroll to **Create a draft profile** at the bottom of the schema page. Enter Profile name `farm-qa`, choose Target environment **lab-synth**, and click **Create draft profile**.
2. On the profile page, click **Submit for review**.
3. Switch to your **approver account**. Go to **Protect → Policies** (not Automate → Test Data Profiles — the profile only appears there after it is approved). Find `farm-qa` and click **Review and approve**.

> **Approver goes to Protect → Policies, not Automate → Test Data Profiles**
>
> The profile only appears under Automate → Test Data Profiles once it has been approved. Before approval, the approver must go to **Protect → Policies** to find and approve it.

H3 Request and check the synthetic data

1. Switch back to your main account. Open **Automate → Test Data Profiles**. The `farm-qa` profile now appears.

2. Leave all scenario counts at 100 and click **Request test data**. The request is **approved automatically** — no second person needed for the actual data request (only the profile definition required approval).
3. Click **Open the request** to see the job progress. Once it shows Succeeded, check the data in Ubuntu:

```
docker exec -it lab-target psql -U postgres -d practice -c "\dt synth_data.*"
docker exec -it lab-target psql -U postgres -d practice -c "SELECT * FROM synth_data.custo
```

Expected result

Customer IDs are prefixed `SYN-` so synthetic rows are always identifiable. Names, emails, addresses and notes are entirely fabricated — DataNivra only saw the column names and types, never any production values. All foreign-key relationships across the 10 tables are intact.

I Automation: CLI, GitHub Actions & AI assistants

1–2 hours

I1 Install the command-line tool (CLI)

First check your Python version:

```
python3 --version
```

Python 3.12 required

The CLI requires Python ≥ 3.12. Ubuntu 22.04 ships Python 3.10 — if you see `3.10.x` you must install 3.12 first:

```
sudo add-apt-repository ppa:deadsnakes/ppa -y
sudo apt update
sudo apt install -y python3.12 python3.12-venv
```

Install pipx (if not already installed) and the CLI:

```
sudo apt update && sudo apt install -y pipx && pipx ensurepath
```

Close and reopen the Ubuntu window, then install the CLI pinned to the Python 3.12 interpreter:

```
pipx install datanivra-cli==0.3.0 --python python3.12
datanivra --version
```

It must say **0.3.0**. Verify four commands work:

```
datanivra sources list
datanivra datasets list
datanivra test-data profiles
datanivra certificates list
```

All four should return a (possibly empty) list without errors.

12 Sign the CLI in

```
datanivra login --url https://api.datanivra.com
```

1. The CLI prints a code (like `BCDF-GHJK`) and a link. Open the link in the browser where you are signed in to the console.
2. Check that the page shows the same code as your terminal, tick the confirmation box and click **Approve**.
3. Back in the terminal, `datanivra whoami` shows your name and organization.

> **Always include `--url`**
>
> The `--url https://api.datanivra.com` flag is required. The CLI has no built-in default URL.

> **Security note**
>
> If someone sends you such a link out of the blue, choose Deny: approving it would sign their computer in as you. Every approval appears in the audit trail.

13 GitHub Actions: test data for every pipeline run

What this integration does — and why it matters

Many teams fall into one of two traps when testing code in CI:

- **Shared static test database** — data drifts over time, parallel runs interfere with each other, and nobody knows whose change broke what.
- **Production snapshot copied into CI** — real customer names, account numbers, and balances flow into GitHub runners and test logs. A data-exposure incident waiting to happen.

DataNivra's GitHub Actions integration solves both problems. Every CI run gets its own fresh, policy-governed slice of realistic test data, and **production data never leaves your environment**.

How the flow works (with a concrete example)

Imagine your team is building a banking mobile app. Your test suite needs 100 customer accounts with realistic balances, transactions, and linked cards — but you cannot use real accounts. Here is exactly what happens when a developer pushes code and the pipeline starts:

STEP	WHO ACTS	WHAT HAPPENS
1	GitHub Actions runner	The runner starts on GitHub's servers. It has no DataNivra password stored anywhere. Instead, GitHub issues it a short-lived cryptographic identity token (valid for a few minutes only) that proves "this is a run of your-org/banking-app, triggered by this commit."
2	DataNivra CLI on the runner	The CLI sends that token to `api.datanivra.com` and says: "I am pipeline `github-ci-eval`. I need 100 rows of `FS_EVERYDAY_BANKING` data for purpose `EVAL-1`."
3	DataNivra cloud (policy check)	DataNivra checks the `ci-eval` policy a human approved in advance: Is this pipeline allowed? Is 100 rows within the approved envelope? Is the purpose valid? If yes, it grants a reservation and signals your agent.
4	DataNivra agent (inside your environment)	Your agent — running as a Docker container on your own server or VM — receives the instruction. It pulls masked or synthetic data (never raw production rows) and provisions exactly 100 customer accounts into **lab-ci**, your CI lab

STEP	WHO ACTS	WHAT HAPPENS
		database. This database lives entirely inside your infrastructure.
5	GitHub Actions runner	The runner connects directly to **lab-ci** and runs your test suite against those 100 accounts. The tests see realistic data: proper account numbers, plausible balances, linked cards — but none of it is real.
6	DataNivra CLI on the runner	When tests finish (pass or fail), the CLI releases the reservation. The 100 accounts are cleared from lab-ci, making room for the next run. The release happens even if the tests fail, via `if: always()`.

What never happens

- Real production rows never reach the GitHub runner or its logs.
- No DataNivra password, API key, or secret is stored in GitHub — the runner authenticates with GitHub's own ephemeral token.
- The pipeline cannot silently take more data than approved — asking for 500 rows when the policy allows 100 returns `TOTAL_EXCEEDS_ENVELOPE` and provisions nothing.
- The data does not "build up" between runs — each reservation is explicitly released, so lab-ci stays clean.

The same workflow works for any app that needs a database

The banking example above applies equally to an e-commerce order service, a healthcare records app, or an HR system. The `--scenario` flag selects the industry pack (e.g. `FS_EVERYDAY_BANKING=100` for 100 banking customers). DataNivra provisions the right shape of data for your domain every time.

a) Find your tenant slug

```
datanivra org tenant
```

This prints your tenant slug — a short lowercase identifier like `dadamoses-org`. You will need it below. **Do not use the UUID** that appears in browser URLs.

b) Register a workload identity in the console

1. Go to **Settings → Workload Identities → Register**.
2. Name it (e.g. `github-ci-eval`), set the audience to `datanivra:<your-slug>` (e.g. `datanivra:dadamoses-org`), and set the trust to your GitHub repository (`repo:YourOrg/your-repo:*`).
3. Copy the **Workload Identity ID** (a UUID).

c) Set repository variables in GitHub

In your GitHub repo: **Settings → Secrets and variables → Actions → Variables → New repository variable**. Create three variables:

NAME	VALUE
`DATANIVRA_TENANT`	Your tenant slug, e.g. `dadamoses-org`
`DATANIVRA_AUDIENCE`	`datanivra:dadamoses-org` (prefix + slug)
`DATANIVRA_WORKLOAD_IDENTITY_ID`	The UUID from step b)

> **Type the values — do not paste**
>
> Pasting from certain tools can silently append a carriage-return (`\r\n`) to the value. That hidden character causes login to fail with `USAGE: Invalid WorkloadTokenExchange: check tenant_slug`. If you must paste, immediately delete to end-of-line and retype the last character.

d) Create a ci-eval policy

1. Go to **Automate → Test Data Profiles → New Profile**.
2. Name it `ci-eval`. Set Automatic Approval, then add the workload identity and profile settings.
3. Have an approver go to **Protect → Policies** to approve it. (They will not find it under Automate — that tab only shows already-approved profiles.)

e) The workflow file

Create `.github/workflows/datanivra-eval.yml` in your repository:

```
name: datanivra-eval
on: workflow_dispatch
permissions:
  contents: read
jobs:
  test-data:
runs-on: ubuntu-24.04
permissions:
  contents: read
  id-token: write     # must be at job level, not workflow level
env:
  DATANIVRA_BASE_URL: https://api.datanivra.com
  DATANIVRA_CONFIG_DIR: /tmp/datanivra
steps:
  - uses: actions/checkout@v4
  - uses: astral-sh/setup-uv@v6
  - run: uv tool install datanivra-cli==0.3.0
  - name: Sign in as this pipeline
    run: |
      TENANT=$(printf '%s' "${{ vars.DATANIVRA_TENANT }}" | tr -d '[:space:]')
      AUDIENCE=$(printf '%s' "${{ vars.DATANIVRA_AUDIENCE }}" | tr -d '[:space:]')
      WI_ID=$(printf '%s' "${{ vars.DATANIVRA_WORKLOAD_IDENTITY_ID }}" | tr -d '[:space:]'
      datanivra login --url https://api.datanivra.com \
        --workload-identity "$WI_ID" \
        --tenant "$TENANT" \
        --audience "$AUDIENCE" \
        --make-default
  - name: Request test data within the approved profile
    id: req
    run: |
      datanivra test-data request \
        --profile-ref ci-eval@1 \
        --scenario FS_EVERYDAY_BANKING=100 \
        --purpose-ref EVAL-1 \
        -o json > req.json
      echo "id=$(python3 -c 'import json;print(json.load(open("req.json"))["id"])')" \
        >> "$GITHUB_OUTPUT"
  - name: Wait until the data is ready
    run: datanivra test-data wait "${{ steps.req.outputs.id }}" --timeout 600 --interval 1
  - name: Your tests would run here
    run: echo "Test data is ready in lab-ci"
  - name: Release the data
    if: always()
    run: |
      TENANT=$(printf '%s' "${{ vars.DATANIVRA_TENANT }}" | tr -d '[:space:]')
      AUDIENCE=$(printf '%s' "${{ vars.DATANIVRA_AUDIENCE }}" | tr -d '[:space:]')
      WI_ID=$(printf '%s' "${{ vars.DATANIVRA_WORKLOAD_IDENTITY_ID }}" | tr -d '[:space:]'
```

```
      datanivra login --url https://api.datanivra.com \
        --workload-identity "$WI_ID" \
        --tenant "$TENANT" \
        --audience "$AUDIENCE" \
        --make-default
      datanivra test-data release "${{ steps.req.outputs.id }}" \
        --reason-code TEST_RUN_FINISHED
```

Why `id-token: write` at job level

GitHub only injects the OIDC token environment variables (`ACTIONS_ID_TOKEN_REQUEST_URL` and `ACTIONS_ID_TOKEN_REQUEST_TOKEN`) when the permission is declared on the job, not just at the workflow level.

Node.js 20 deprecation warning (harmless)

You may see: "Node.js 20 is deprecated ... actions/checkout@v4, astral-sh/setup-uv@v6". This is a GitHub notice about their runners, not a DataNivra issue. It does not affect the workflow and will resolve automatically when those actions release Node.js 24 versions.

Over-limit protection to test

Try asking for more than the approved limit: change `=100` to `=500` in the workflow. The request is DENIED with the reason `TOTAL_EXCEEDS_ENVELOPE` and nothing is created. This is the protection you are buying: pipelines cannot quietly take more than you approved.

J Cloud sources (optional)

~1 hour each

Each connection follows the same pattern as Part E: create a read-only identity in the cloud service, store its details in a connection file next to the agent, register the source, and run discovery. Use only test buckets, datasets or catalogs during the evaluation.

SERVICE	READ-ONLY IDENTITY TO CREATE	NOTES
Amazon S3	IAM user or role with only `s3:GetObject` and `s3:ListBucket` on one prefix	Upload a few CSVs from the Appendix download to a test bucket.

SERVICE	READ-ONLY IDENTITY TO CREATE	NOTES
Google BigQuery	Service account with BigQuery Data Viewer on the dataset and Job User on the project	Reading uses the table-data API, not billed query scanning.
Databricks	Non-admin service principal with an OAuth secret	Needs Unity Catalog and a SQL warehouse.

K Governance, evidence & preview features

~30 min

K1 Audit and security review

1. Open **Trust → Audit**. Every action you took is listed. The trail is hash-chained: tampering is detectable.
2. Open **Trust → Security Review**. Choose a date range and click **Download audit evidence (JSON)**.

K2 Preview features

FEATURE	WHERE	STATED LIMIT
Compliance Guard	Trust → Compliance Guard	Reports codes and counts; does not certify compliance.
Document sets	Build → Document Sets	No PDF; rule-based detection; does not prove anonymisation.
Model Provider Policy	Protect → Model Provider Policy	No model adapter ships yet, so every model call is refused.

L Plans, pricing & how to buy

~15 min

PLAN	PRICE	WHAT IT IS FOR
Free	$0, no card	Permanent, synthetic data only: hosted sandbox, all 15 industry packs, custom schemas in sandbox. No own agents or sources.
Pro Trial	$0 for 30 days, once	Included with a new organization: the real workflow with one agent, three sources, CI/CD and custom schemas.
Team	From $499/ month	Small teams: 3 agents, 10 sources, 25 profiles, 5 CI identities, 1 year of audit history.
Business	From $1,499/ month	Larger estates: 10 agents, 50 sources, single sign-on, priority support.
Enterprise	Custom quote	Negotiated limits and terms, contract, invoice payment.

Questions to discuss with the DataNivra team

- Which of our production databases are supported today, and which are "coming next"?
- What support response times will be written into our contract (SLA)?
- Where will our agent run, and who in our team owns it?
- Do we need the self-hosted control plane, or is the hosted service acceptable to our security team?

Ready to get started?

Sign up free — no credit card, no installation required to try the hosted sandbox. Add your own agent when you're ready.

Start free →

Already have an account? Sign in to the console

After rebooting your PC

Secret files disappear after a reboot — you must restore them

On some systems the agent's secrets directory is mounted on `/run/datanivra-secrets/`, which is a **tmpfs** — a RAM-based filesystem that is wiped clean every time the PC restarts. If your `docker inspect` output shows `/run/datanivra-secrets → /etc/datanivra/secrets`, all secret files (connection files and masking key) must be restored after every reboot.

After a reboot, do these two things in the Ubuntu window before continuing your evaluation:

1. **Restore the secret files** — re-run the E2 paste block:

```
. "$HOME/datanivra-lab/passwords.env"
printf '{"kind":"POSTGRESQL","host":"lab-source","port":5432,"dbname":"practice","u
for env in qa synth ci; do
  printf '{"kind":"POSTGRESQL","host":"lab-target","port":5432,"dbname":"practice",
done
```

   Also restore the masking key (re-run the F1 masking-key block with your stored key).

2. **Reconnect the agent to the lab network** — Docker sometimes disconnects containers from custom networks on restart:

```
docker network connect datanivra-lab datanivra-agent
```

After these two steps, run the doctor check (E2) again to confirm everything is green before continuing.

Troubleshooting

SYMPTOM	WHAT TO DO
Sign-in link says it is invalid	Links work once and expire after 30 minutes. Request a new one.
"Separation of duties" when approving	Expected: approve with the other account (your approver).
Agent not ACTIVE	Check Docker Desktop is running; run `docker logs datanivra-agent` ; create a new enrollment token if the old one expired.
`SECRET_NOT_FOUND` in doctor	The connection files in `~/datanivra-secrets/` are missing. Re-run the E2 paste block.
`SOURCE_REACHABLE FAIL` or `TARGET_REACHABLE FAIL`	The agent is not on the `datanivra-lab` network. Run `docker network connect datanivra-lab datanivra-agent` .
"A source with this name already exists" when registering	The source was already created. Go to Connect → Data Sources and use the existing one.
CLI says the code expired	Run `datanivra login --url https://api.datanivra.com` again and approve within 10 minutes.
`USAGE: Invalid WorkloadTokenExchange: check tenant_slug` in GitHub Actions	The `DATANIVRA_TENANT` variable was pasted with a hidden `\r\n` appended. Delete the variable and re-enter the slug by typing it manually.
`INVALID_CREDENTIALS` (exit code 3) in GitHub Actions	The `--url` flag is missing on the login command, or `DATANIVRA_TENANT` contains the UUID instead of the slug. Run `datanivra org tenant` locally to get the correct slug.
GitHub run waits and times out	Your PC or Docker Desktop was off, or the agent is not ACTIVE. Start it and re-run the workflow.
Doctor shows `SKIP` or `DEPENDENCY_FAILED`	A prerequisite check failed. Fix the first `FAIL` line you see (most often `SOURCE_REACHABLE FAIL`), then

SYMPTOM	WHAT TO DO
	re-run the doctor command. Dependent checks will run once the root cause is resolved.
Secret files missing after reboot (`SECRET_NOT_FOUND`)	Your secrets directory is on a `/run/` tmpfs that clears on reboot. Re-run the E2 paste block to restore the connection files, then run `docker network connect datanivra-lab datanivra-agent` to reconnect the agent.

Help

support@datanivra.com (product questions) · **sales@datanivra.com** (plans and contracts) · **security@datanivra.com** (security questions) · Phone +1 (607) 361-3823, Mon–Fri 9am–5pm ET