Yesod Observability

Field Guide

Stephen

August 2026

Contents

A compact operational reference for humans and agents. Where to look, how to query, and how to operate the telemetry pipeline. Normative contract: `ys-yes-rklm` and `reference/observation-framework.md`; narrative: book chapter "Seeing the Whole System"; exporter runbook: `docs/factory-observability-runbook.md` in the yesod repository. Current as of fleet release `541189d3` (promoted 2026-08-18); reviewed 2026-08-20.

The one rule that governs everything here: PostgreSQL is the lifecycle authority and replay source. SigNoz/ClickHouse is diagnostic only — it may lag, sample, expire, or be replaced without changing factory state. Never infer a factory transition from the telemetry backend; never let a telemetry outage block factory work. If SigNoz and the ledger disagree, the telemetry is wrong by definition.

Part I — Where to Look

The surfaces

Surface	Address	What it answers
SigNoz UI	`http://obs-vultr:8080` (Tailscale only)	trends, rates, history at a glance
`Yesod Factory` dashboard	SigNoz > Dashboards	pipeline health + factory activity in one page
Timeline/explain API	`GET /api/notes/{id}/timeline`, `/explain` (dertog)	authoritative per-note causal history
PostgreSQL ledger	`factory_events` + subsystem tables (192.168.0.155)	ground truth for everything
ClickHouse	on `obs-vultr`, not exposed; root SSH for operators only	raw telemetry storage

Choose by question class: *"what happened to note X"* => timeline/explain or ledger. *"how much / how often / when did it start"* => SigNoz. *"is this true"* => ledger, always.

Signal inventory

Logs — one per factory event, `service.name=yesod-exporter`. Parsed attributes (usable in filters and group-by): `event.type`, `workflow.stage`, `outcome.class`. Everything else (note id, run id, SHAs, actor, payload, causation) lives in the JSON body — searchable with `body CONTAINS '...'`, deliberately not an attribute (cardinality discipline).

Traces — six span types, derived from events, linked with `FOLLOWS_FROM` across async boundaries rather than one multi-day trace: `yesod.plan`, `yesod.dispatch`, `yesod.child.run`, `yesod.refinery.attempt`, `yesod.gate`, `yesod.deploy`.

Metrics — seven names, all `yesod.factory.*`: `exporter.lag` (gauge, s), `microvm.active` / `microvm.capacity` (gauges), `queue.age`, `run.stuck`, `heartbeat.stale`, `deployment.skew` (histograms, fire on trigger conditions). Labels are finite enums only (`environment`, `service`, `workflow_stage`, `outcome_class`, `model_tier`, `runner_pool`, `host_role`, `target_branch`). IDs, SHAs, hostnames, and model names are forbidden labels.

Per-note LLM telemetry (ys-yes-egm8, live in `541189d3`) — a *second*, parallel stream, distinct from the exporter path above. Every agent process the runner spawns is launched with OpenTelemetry resource attributes (`OTEL_RESOURCE_ATTRIBUTES` and `OPENCODE_RESOURCE_ATTRIBUTES` both carry them): `yesod.note.id`, optional `yesod.run.id`, `yesod.stage` (`plan` / `execute`), and `yesod.lane` (the effective dispatch target). The model dimension comes from the provider's own model attributes. This is what lets token usage be

filtered by note and grouped by pipeline stage and lane — the per-stage token attribution the factory prices work with. Note the boundary: this rides the provider's telemetry, not `factory_events`, so it is diagnostic like the rest of SigNoz and never a lifecycle authority. The purpose-built SigNoz LLM view over it is still to be built (ys-yes-qe1g).

Query cookbook

SigNoz filter expressions reference log attributes with an `attribute.` prefix; group-by keys use the bare name. The asymmetry is easy to trip on.

Want	Expression / query
Hide any residual reconciler noise (bug ys-yes-x0yk, fixed in `541189d3`)	`attribute.event.type NOT LIKE 'reconcile.root_close%'`
One note's events	`body CONTAINS 'ys-yes-XXXX'`
One run's events	`body CONTAINS 'run-XXXXXXXXXXXX'`
Failures only	`attribute.outcome.class != 'success'`
One pipeline stage	`attribute.workflow.stage = 'reconcile'`
Rate by type	aggregation `count()`, group by `event.type`
Contract integrity	compare counts: `reconcile.child_close.intended` vs `.observed` — they must track

Ledger-side equivalents (from `yesod-dispatch`, via the controller env DSN):

```
-- what kinds of events, how many, how fresh
SELECT event_type, event_role, count(*), max(occurred_at)
FROM factory_events GROUP BY 1,2 ORDER BY 3 DESC;

-- everything about one note, in causal order
SELECT event_id, occurred_at, event_type, event_role, actor, outcome
FROM factory_events WHERE note_id = 'ys-yes-XXXX' ORDER BY event_id;

-- exporter delivery state (the checkpoint IS the delivery proof)
SELECT sink_id, event_id, updated_at FROM factory_telemetry_sink_checkpoints;
SELECT max(event_id) FROM factory_events;   -- compare: gap = undelivered
```

Reading rules for agents

- Query the constrained read surfaces (timeline/explain, dashboard queries); do not screen-scrape the SigNoz UI.
- A missing event is not evidence of a missing action — check emitter coverage (`complete` / `partial_legacy` / `empty_current` / `unknown` in explain responses) before concluding anything from absence.

- Duplicated telemetry is normal (delivery is at-least-once after replays); dedupe on `event_id` when counting from ClickHouse.
- Telemetry never authorizes action. Verify against the ledger before any mutation.

Part II – Operating the Pipeline

Topology

```
runners / conductor / dispatch / CLI writers
        |  (events written in the same transaction as, or bracketing,
        |   the authoritative mutation)
        v
PostgreSQL factory_events  (authority + replay source, retained indefinitely)
        |
        v
yesod-factory-exporter.service      (ONE instance, on yesod-dispatch)
        |  OTLP/HTTP, Tailscale grant, no app credentials
        v
obs-vultr:4318 -> SigNoz collector -> ClickHouse   (diagnostic, disposable)
```

Workers emit events (their writers record actor identity) but never speak OTLP and hold no telemetry credentials. Exactly one exporter exists, checkpointed per `sink_id` in PostgreSQL; the checkpoint advances only after the sink acknowledges every derived signal.

Commission / verify / stop

Commission (from the landed release on `yesod-dispatch` — full detail in the runbook):

```
sudo systemctl enable --now yesod-factory-exporter.service
```

Config: `/etc/yesod/refinery/factory-exporter.env` (root:yesod-control 0600); enable-marker `/etc/yesod/refinery/factory-exporter-enabled`. Bounds fail closed: batch 1-1000 (default 256), poll 0.1-60s (default 1), drain 0-60s (default 10). Keep `YESOD_FACTORY_MICROVM_CAPACITY` in sync with the real gate fleet — it feeds a gauge, and a stale value is a false claim on a chart (it drifted 8 vs 16 once already).

Verify (three independent checks, strongest first):

1. checkpoint vs head in PostgreSQL (SQL above) — equal or within a batch;
2. `systemctl is-active` + journal free of credential content;

3. SigNoz: logs arriving under `service.name=yesod-exporter`, only the six span names, only the seven metric names.

Stop without losing anything: `sudo systemctl disable --now yesod-factory-exporter.service` — keep the checkpoint and events. Restart resumes from the last acknowledgment. Never move a checkpoint forward to conceal lag.

Standing artifacts on the backend

Artifact	Content
Dashboard `Yesod Factory`	overview: exporter lag; MicroVM active vs capacity; signal events by type (noise filtered); root_close noise rate; note transitions; child-close intent vs observed; runs started vs completed; workflow-stage activity; outcome classes; deploy/release observations
Dashboard `Yesod Contract Integrity`	contract checks: close triplet (intended/observed/completed); non-success outcomes; agent claims vs ledger confirms; arms vs route changes; worktrees vs runs; refinery/gate witness panel; plan jobs; deploy observations
Alert `Yesod factory exporter lag high`	warning when avg `exporter.lag` > 300s over 5m; channel `stephen-email` (delivery requires SMTP on the SigNoz host; the Triggered Alerts view works regardless)

Every panel carries its own tooltip (the `i` on the panel header): what it measures, what good looks like, what bad looks like, and the first response. Read the tooltip before interpreting an unfamiliar panel — the operating knowledge lives in the dashboard itself, not only in this guide.

How to read the overview at a glance: *Exporter Lag* flat at zero (any climb = exporter/sink problem, factory unaffected); *MicroVMs* bursting under the capacity line (pinned at or above capacity = investigate); *Signal Events* bursts matching known work; *root_close Noise* — high in the screenshot above, which predates the fix; `ys-yes-x0yk` has since landed in `541189d3` and the panel shows the cliff and a flat floor thereafter; the paired lines in the lower rows tracking each other.

On the integrity page, the rule is uniform: **paired lines must track**. Any persistent gap between an intent series and its confirmation series is a contract question, answered in the ledger, never in ClickHouse. The *Refinery & Gate* panel is expected empty until the first post-release admission — merged is not proven.

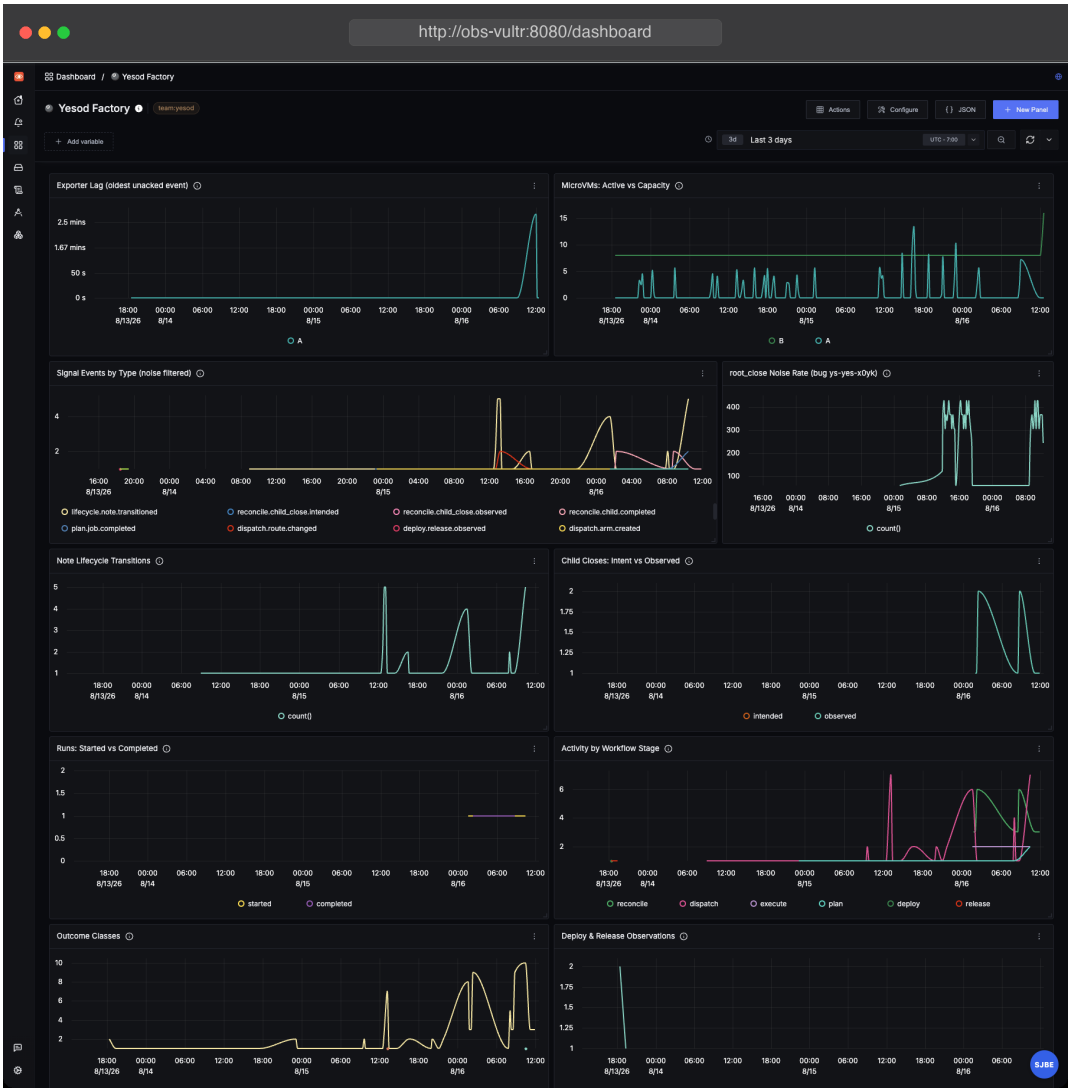

Figure 1: Yesod Factory dashboard, first three days of production.

Figure 2: Contract Integrity dashboard: paired intent/confirmation lines.

Figure 3: Tooltips carry per-panel operating knowledge.

All three artifacts were created via the API and can be recreated from this guide; they are diagnostic furniture, not factory state — losing them loses nothing authoritative.

Migration to the permanent host

The Vultr node is disposable by design. To move the backend:

1. stand up SigNoz on the new host; point a Tailscale grant at its OTLP port;
2. set `YESOD_OTLP_ENDPOINT` to the new address in the exporter env;
3. **new `sink_id`** => full replay: the new backend re-ingests the entire event history from PostgreSQL and starts with everything; **same `sink_id`** => continuation from the last acknowledgment;
4. restart the exporter; verify with the three checks above;
5. recreate dashboard + alert (or export/import their JSON first);
6. tear down the Vultr node. Export nothing from ClickHouse — it holds no authoritative state.

Known issues and follow-ups

Ref	What
`ys-yes-x0yk` (bug)	**Resolved, in `541189d3`.** `reconcile.root_close.observed` had been emitted per poll cycle over closed roots — 99% of all events; now emitted on change / first observation only. The dashboard's noise panel shows the cliff.
`ys-yes-d2le` (FR)	Open. No spans for release promotions (derive from `release_promotion_*` tables — timestamps already exist); no refusal-code events, so parked states are invisible to telemetry. The 2026-08-19/20 promotion campaign — three rolled-back attempts that left no trace in SigNoz — is concrete motivation.
`ys-yes-qe1g` (FR)	Open. A SigNoz LLM view over the per-note token telemetry now emitted (see the Signal inventory's LLM subsection); the attributes flow, the dashboard to consume them does not exist yet.
refinery/gate emitters	Merged; produce production events from the first post-release refinery attempt onward. Merged is not proven — witness the first admission.

Ref	What
alert email delivery	needs SMTP configuration on the SigNoz host, or replace the channel with Slack/webhook.

Do / don't

Do	Don't
Treat every SigNoz view as a lens over the ledger	Treat a dashboard as a source of truth
Alert on exporter lag (silence is a failure mode)	Let telemetry block or retry factory work
Keep IDs in log bodies, enums in labels	Add note ids / SHAs / model names as metric labels
Label test cells `deployment.environment=test`	Mix playground telemetry into production views
Replay via a fresh `sink_id`	Copy ClickHouse data between backends