OBSERVABILITY GUIDE

SYNLYNK

Observing dispatched agent work across TUI, Vizor, and Slack.

v0.13.0 · Telemetry & Fleet Operations

```
$ synlynk tui # Terminal curses dashboard
$ synlynk viz --serve # Web visualizer (port 8721)
$ synlynk notify slack --webhook-url <url>

✓ Unified uxcore data engine · Zero external DB
```

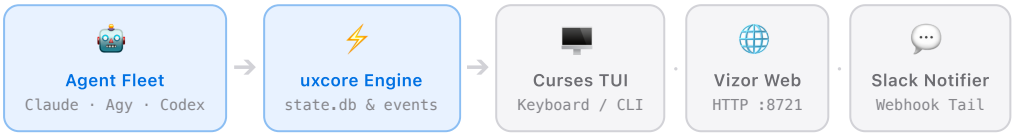
Curses TUI · Vizor Web Dashboard · Slack Notifier · Unified uxcore Layer

synlynk.com · github.com/nikhilsoman/synlynk · 2026

ARCHITECTURE & DATA FLOW

One engine. Three monitoring surfaces.

Synlynk provides a multi-surface observability suite designed for every operational context. Whether you are running interactive sessions in a terminal, operating a multi-repo wallboard in a web browser, or receiving asynchronous event alerts in team channels, all surfaces read from the exact same `uxcore` data foundation.



The Three Surfaces at a Glance

Curses TUI (Terminal)

Standard-library curses dashboard for developers who stay in the CLI. Zero dependencies, hotkeys for Fleet/Jobs/Costs/Review, and interactive job control.

Vizor Web Dashboard

Local web server on port 8721 serving interactive Gantt timelines, Architect Map repo graphs, cost charts, and web-based write endpoints.

Slack Notifier

Minimal one-way event tail posting dispatch, approval, and kill events directly into team Slack channels with direct deep-links to Vizor.

Unified State Model

No database drift or sync issues. All three surfaces query the single SQLite canon (`.synlynk/state.db`) via `uxcore`.

SHARED DATA ENGINE

Vizor and the TUI both read from the same `uxcore` data layer — anything you can do in one surface (inspecting jobs, approving PRs, killing tasks), you can do in the other.

CURSES TERMINAL UI

Terminal-native dashboard. Zero dependencies.

Launch the TUI inside any initialized Synlynk project to open a lightweight, curses-based split view. It uses Python's standard `curses` module with zero third-party packages required.

synlynk tui

```
$ synlynk tui
# Must be run from inside an initialized repository root
```

Panel Navigation Hotkeys

Switch between the four main observability views with single keypresses:

KEY	PANEL NAME	CONTENTS & OPERATIONAL DETAILS
`1`	Fleet Panel	Displays per-agent breakdown: completed tasks (done), active tasks, success rate (%), and total USD spend per agent.
`2`	Jobs Panel	Shows real-time execution list with timestamp, agent name, exit code status, duration in seconds, cost, and selection cursor.
`3`	Costs Panel	Displays aggregate financial telemetry: total spend, estimated spend, and itemized actual vs estimated spend per agent.
`4`	Review Panel	Lists system capability checklist showing enabled (`✓`) vs disabled (`✗`) capability flags for current workspace actor.
`q`	Quit	Exits the curses TUI session cleanly and returns to shell prompt.

Visual Mockup — TUI Fleet View (Key 1)

TUI — Fleet Panel

```
FLEET agy
agy done=14 active=1 success=93% $4.12
claude done=28 active=0 success=96% $8.45
codex done=9 active=0 success=88% $2.10
grok done=4 active=0 success=100% $0.95

[1]Fleet [2]Jobs [3]Costs [4]Review [q]uit
```

JOB CONTROL & KEYBINDINGS

Interactive job control. Approve and kill in TUI.

The Jobs panel (key `2`) provides full line selection and interactive job lifecycle management without leaving your terminal.

⬡ **NEW CAPABILITY — SHIPPED IN PR #851**

The TUI Jobs panel now supports interactive selection with arrow keys, one-key PR approval (`a`), and graceful job termination (`k`) with safety prompts.

Jobs Panel Keybindings

↑↓ **Navigate Job List (KEY_UP / KEY_DOWN)**

Moves the selection cursor (`>`) up and down through historical and currently executing jobs.

a **Approve Pending PR (Approve)**

If the selected job is in a `pending_approval` state with an attached PR, pressing `a` calls `uxcore.approve_pr()`. If ineligible, shows status message: *"No pending-approval job selected"*.

k **Kill In-Flight Job (Kill)**

If selected job is `in_flight` (queued/running/active), prompts `Kill selected job? [y]es [n]o`. Confirming with `y` invokes `uxcore.kill_job()`. If ineligible, shows message: *"No in-flight job selected"*.

Visual Mockup — TUI Jobs Panel with Interactive Hint

TUI — Jobs Panel

```
JOBS claude
> 2026-08-09T14:10:00 claude pending_approval 45.2s $0.35
2026-08-09T14:02:15 agy exit=0 12.1s $0.12
2026-08-09T13:45:30 codex exit=0 89.4s $0.80
2026-08-09T13:30:10 grok exit=1 15.0s $0.10

[1]Fleet [2]Jobs [3]Costs [4]Review [q]uit [a]pprove [k]ill
```

VIZOR LOCAL WEB SERVER

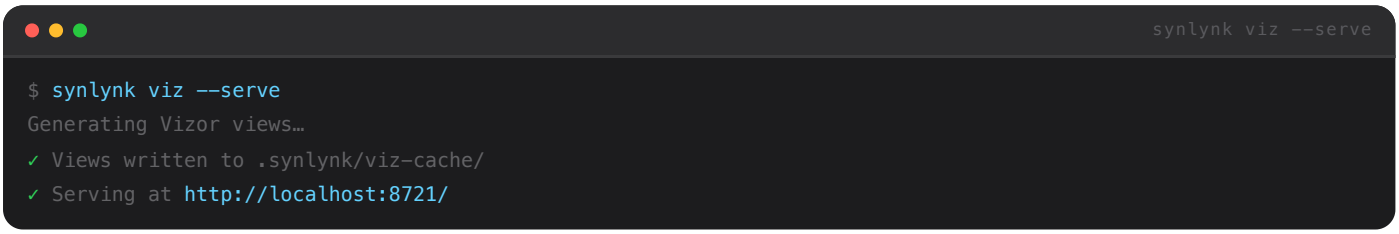
Rich web dashboard. Port 8721 background server.

Vizor generates an offline-first HTML/JS dashboard from local workspace telemetry and serves it over a lightweight Python HTTP background server.

COMMAND	DESCRIPTION & OPERATIONAL BEHAVIOR
`synlynk viz --serve`	Starts the Vizor background HTTP server on default port `8721` and keeps serving workspace updates.
`synlynk viz --port 8900`	Overrides the default port, running the HTTP server on custom port `8900` instead.
`synlynk viz --generate`	Regenerates static HTML views in `.synlynk/viz-cache/` without starting a server or launching a browser.
`synlynk viz --open`	Opens the existing cached Vizor dashboard (`http://localhost:8721/index.html`) in your default browser.
`synlynk viz --stop`	Stops the running Vizor background HTTP server process gracefully.

📌 **DEFAULT PORT CORRECTION — FIXED IN PR #850**
Vizor defaults to port **8721** (`DEFAULT_PORT = 8721` in `synlynk/viz.py`). An earlier legacy document erroneously cited port 8420 — port 8721 is authoritative.

Server Startup Visual Mockup



SETUP & NAVIGATION TABS

Interactive setup. Five visual monitoring views.

First-Run FTUE (First-Time User Experience)

On initial interactive launch, Vizor prompts for project preferences and stores them under `vizor.*` in `.synlynk/config.json`:

1. **User-Facing UX? (y/n)**
   Picks second navigation tab view: `journeys` for user-facing projects, or `tube` (Architect Map) for internal/library repos.
2. **Browser Notifications? (y/n)**
   Enables desktop browser notifications (`notify_on_refresh`) when background data updates land.
3. **Auto-Refresh Interval? (15 / 30 / off)**
   Sets auto-polling interval in minutes (`refresh_interval_minutes`). Non-interactive runs default to off / `tube`.

Five Navigation Tabs

1. Gantt (Default)
Timeline view displaying agent execution overlap, task duration bars, and parallel job concurrency.

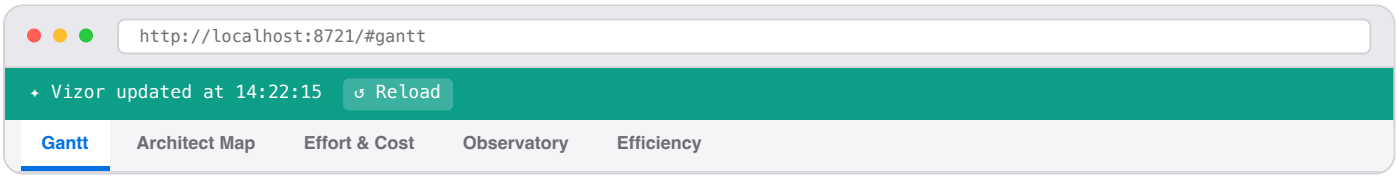
2. Journeys / Architect Map
User journey flow diagram or multi-repo tube map based on FTUE project type choice.

3. Effort & Cost
Financial graphs, token counters, cost per agent, and estimated vs actual spending trends.

4. Observatory & 5. Efficiency
Live job process inspector (htop style) and agent performance success rate analytics.

Manifest Polling & Live Reload Banner

Vizor background script polls `/manifest.json` every 60s. When new data lands, it displays an overlay banner:



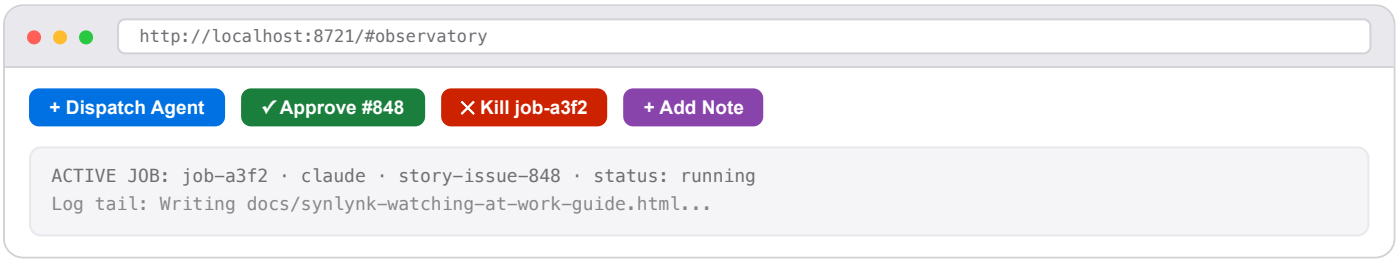
INTERACTIVE DASHBOARD CONTROL

Full web control. Dispatch, approve, kill, note.

Vizor is not just a read-only viewer. The web dashboard provides four interactive POST API endpoints that allow users to mutate state and control agents directly from browser components.

ACTION	HTTP ENDPOINT	WEB UI TRIGGER & BACKEND BEHAVIOR
`Dispatch Job`	`POST /dispatch`	Triggered via **[+ Dispatch Agent]** modal. Spawns a background agent job with specified task parameters.
`Approve PR`	`POST /approve`	Triggered via **[✓ Approve]** button on pending jobs. Calls `uxcore.approve_pr()` to merge approved code.
`Kill Job`	`POST /kill`	Triggered via **[✕ Kill]** button on active jobs. Calls `uxcore.kill_job()` to stop runaway processes.
`Add Note`	`POST /note`	Triggered via **[+ Note]** button. Saves operational annotations directly to `.synlynk/viz-notes.json`.

Visual Mockup — Vizor Web Action Panel



⚡ **FULL FEATURE PARITY**

Actions taken in Vizor immediately broadcast through `uxcore` and reflect in the CLI and TUI within the next refresh cycle.

SLACK WEBHOOK NOTIFIER

Asynchronous team alerts. Bring your own webhook.

The Slack notifier (`synlynk/notifiers/slack.py`) acts as a minimal, zero-footprint event tail consumer that posts agent activity into team Slack channels.

synlynk notify slack

```
$ synlynk notify slack --webhook-url https://hooks.slack.com/services/T00/B00/X00
```

BRING YOUR OWN WEBHOOK

There is no Slack app registration or OAuth configuration command in Synlynk. Users generate an Incoming Webhook URL directly from Slack's app console and pass it to Synlynk.

Subscribed Event Types (PR #850)

The notifier performs a one-shot tail of events filtered to three core action types:

- `dispatch` — Fires when an agent job is dispatched into the background queue.
- `approve_pr` — Fires when a pull request is approved by an agent or operator.
- `kill_job` — Fires when an in-flight job is cancelled or terminated.

READ-ONLY SURFACE WARNING

The Slack integration is strictly a one-way, read-only notifier. It posts alerts but does not support interactive Slack buttons or slash commands. Use TUI or Vizor for write actions.

Visual Mockup — Slack Message Output with Vizor Deep-Link

FEATURE MATRIX & PLAYBOOK

Choosing the right surface. Observability playbook.

CAPABILITY / FEATURE	CURSES TUI	VIZOR WEB DASHBOARD	SLACK NOTIFIER
INTERFACE & ENVIRONMENT			
Primary Surface	`Terminal (Curses)`	Web Browser (HTML5)	Slack Channel
Execution Mode	`Interactive foreground`	Background HTTP server	One-shot tail script
Default Port / Process	`No network port`	Port `8721`	Outbound HTTP POST
TELEMETRY & CAPABILITIES			
Live Updates	**`Real-time hotkeys`**	**60s manifest poll**	**Event push tail**
Interactive Write Actions	**`Yes (Arrow / a / k)`**	**Yes (POST endpoints)**	No (Read-only)
Gantt & Architect Map	No	**Yes (Full visual)**	No
Vizor Deep-Linking	N/A	**Native routes**	**Yes (Embedded links)**
SETUP & BEST OPERATIONAL USE CASE			
Setup Requirement	`Zero setup`	`synlynk viz --serve`	BYO Webhook URL
Recommended Use Case	`Developer active coding`	Team wallboard / PM	Asynchronous team alerts

🏆 OBSERVABILITY COMPLETE

Full Fleet Control Unlocked

You have complete visibility across terminal hotkeys, interactive web dashboards, and real-time team notifications.

📘 **RELATED DOCUMENTATION**

For quick onboarding see the **Quick Start Guide**; for CLI syntax see the **Command Reference**; for database schema details see **The Manual**.