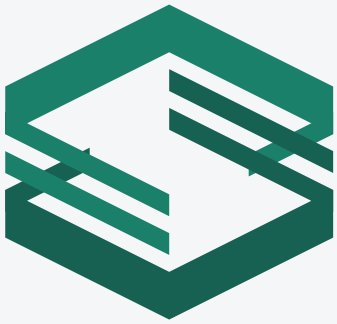
Manual 06 · watching the runs of a whole fleet

Monitor



The Stilton Monitor collects the events of every run from every runner and shows them in a browser — what happened, when, where and on whose authority. This manual covers starting it, connecting runners, reading the interface, keeping and reopening logs, and its security and limits.

CONTENTS

01 Two ways of watching a run

02 Starting the monitor
- 2.1 As a service
- 2.2 Settings

03 Connecting runners

04 The interface
- 4.1 Reading a failed run — an example

05 Where logs are kept

06 Saving and reopening a log
- 6.1 The log format

07 Provenance of a run

08 Security and limits

09 HTTP interface

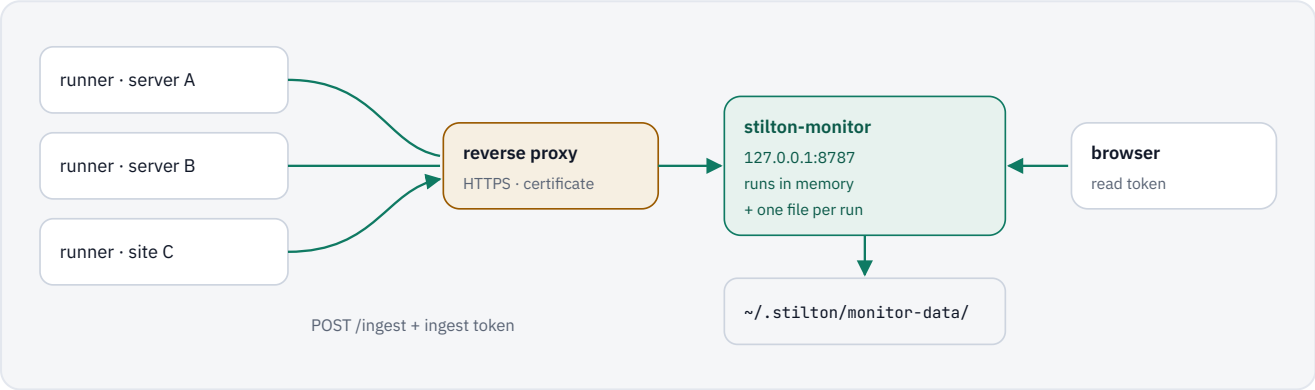
10 Troubleshooting

Copyright © 2025–2026 Marco Dongu. All rights reserved.

01 Two ways of watching a run

Stilton has two observation tools with different purposes. They share only the format of the events.

	Monitor panel in the studio	Stilton Monitor
Purpose	a debugger: understand a flow you are building	a flight recorder: understand what happened in production
When	while you design and run in the studio	always on, receiving from the runners of the fleet
Detail	maximum: memory, rows, fine timeline	filtered by the artifact's log level — no data rows by default
Where	inside the desktop application	on a server, reached with a browser



The monitor is a single program with no other dependency: no database, no web server to install. It needs no graphical environment.

---

02 Starting the monitor

Linux — bash

```
export MONITOR_TOKEN=$(openssl rand -hex 32)        # runners → monitor
export MONITOR_READ_TOKEN=$(openssl rand -hex 32)   # people → monitor
stilton-monitor                 # port 8787, listening on 127.0.0.1 only
stilton-monitor 9000            # another port
```

Without the two tokens the monitor does not start. For a quick experiment on your own computer you can do without them, but you have to say so:

Linux — bash

```
MONITOR_INSECURE_LOCAL=1 stilton-monitor
```

At start-up the monitor prints how it is configured: address and port, data folder, how many runs it reloaded from disk, whether ingest and reading are protected by tokens, and warnings such as identical tokens or plain HTTP on the network. Read those lines: they answer most questions.

As a service

/etc/systemd/system/stilton-monitor.service — ini

```
[Unit]
Description=Stilton Monitor
After=network-online.target

[Service]
User=stilton-monitor
EnvironmentFile=/etc/stilton/monitor.env
ExecStart=/usr/local/bin/stilton-monitor 8787
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

/etc/stilton/monitor.env (chmod 600) — ini

```
MONITOR_TOKEN=4f9c…
MONITOR_READ_TOKEN=b71e…
MONITOR_DATA_DIR=/var/lib/stilton-monitor/data
MONITOR_MAX_RUNS=500
```

Settings

Variable	Default	Effect
`MONITOR_TOKEN`	**required**	ingest token (runners), at least 16 characters
`MONITOR_READ_TOKEN`	**required**	read token (browser and API), at least 16 characters, different from the ingest token
`MONITOR_BIND`	`127.0.0.1`	listening address; `0.0.0.0` to listen on the network (prefer a TLS reverse proxy)
`MONITOR_DATA_DIR`	`~/.stilton/monitor-data`	folder of the stored runs, one file per run
`MONITOR_MAX_RUNS`	`200`	runs kept **in memory**; older ones leave the list, their files stay
`MONITOR_MAX_EVENTS`	`50000`	events per run kept in memory; the file stays complete
`MONITOR_MAX_BODY_BYTES`	8 MiB	largest accepted push (larger → 413)
`MONITOR_RATE_BYTES_PER_MIN`	64 MiB	ingest volume per IP per minute (more → 429); `0` = no limit
`MONITOR_AUTH_FAIL_MAX`	`20`	wrong read passwords per IP per minute before 429; `0` = off
`MONITOR_WORKERS`	`16`	requests served in parallel (queue full → 503)

Variable	Default	Effect
`MONITOR_INSECURE_LOCAL`	off	`1` = tokens optional, only when listening on loopback

03 Connecting runners

The monitor's address is chosen in the studio's **Compile** panel and signed into the artifact:

In the Compile panel	The runner pushes to
a literal URL, `https://monitor.internal/ingest`	that URL
`${MONITOR_URL}`	the value of the environment variable `MONITOR_URL` on the run machine — the same artifact can report to a different monitor at each site
empty	`MONITOR_URL` if set on the run machine; otherwise no monitor

The token never travels in the artifact: each run machine has `MONITOR_TOKEN` in its environment.

Linux — on the run machine — bash

```
export MONITOR_URL=https://monitor.internal/ingest
export MONITOR_TOKEN=4f9c…
stilton-runner orders-daily.ffart
```

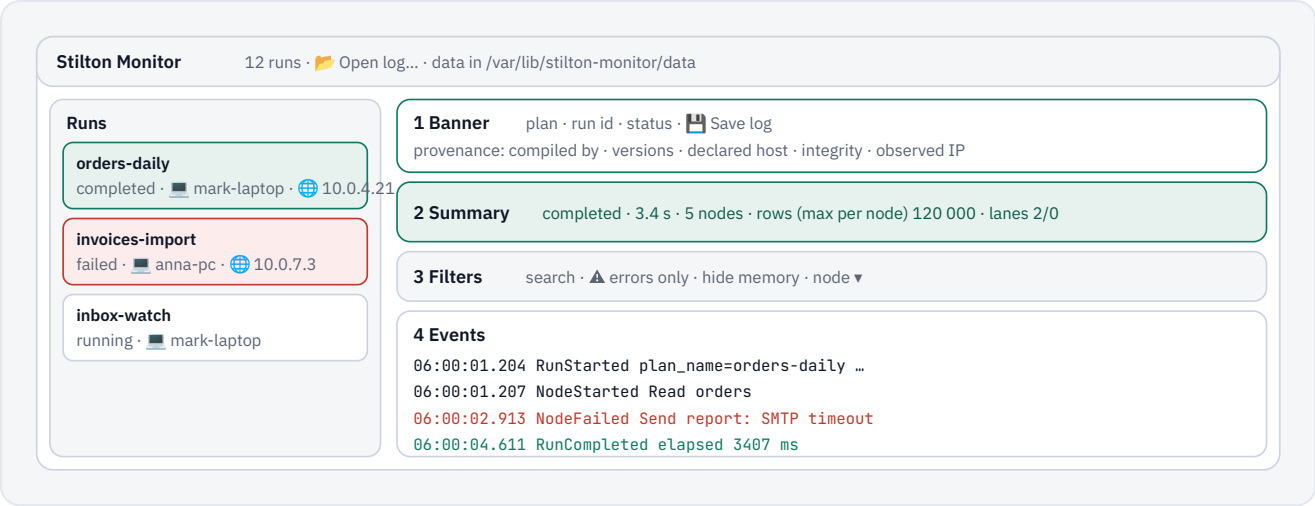
Pushing never slows down or stops a run. Events go in batches with a short timeout. If the monitor is unreachable or refuses a batch (401, 413, 429), the events are appended to `stilton-monitor-fallback.ndjson` in the runner's working directory. Open that file later in the monitor (chapter 6), or send it again: events already received are recognised and not duplicated.

Linux — resend a fallback file — bash

```
curl -sS -X POST https://monitor.internal/ingest \
  -H "Authorization: Bearer $MONITOR_TOKEN" -H "Content-Type: application/x-ndjson" \
  --data-binary @stilton-monitor-fallback.ndjson
```

04 The interface

Open the monitor's address in a browser. It asks for a user and a password: any user name, and the **read token** as password. The page refreshes by itself every two seconds.



The list (left) shows the runs, newest first: plan name, status, number of events and, when known, the studio that compiled the artifact and the IP address the events came from.

The banner shows the selected run's plan, id, status and provenance (chapter 7), and **Save log**.

The summary answers "how did it go?" at a glance: outcome, duration, nodes, rows, lanes succeeded/failed. If the run failed, the error message is shown here instead of being buried in the events.

> **Note.** *Rows (max per node)* is the real size of the flow — the largest number of rows a single node handled. The *sum of node rows* adds every node, so the same rows are counted once per node they cross: it is labelled as a sum on purpose.

Filters survive the automatic refresh:

- **search** — any text in the event type or content, case-insensitive; matches are highlighted, and a summary tells how many events match, how many are errors, the time span they cover, and counts per event type and per node;
- **errors only**;
- **hide memory** — on by default; memory samples, when shown, read like `RSS 18.6 MB · PSS 15.5 MB`;
- **node** — isolate the events of one node.

Events show the time to the millisecond (the time on the run machine, not the arrival time), the type and the content. Errors are red, warnings amber, completions green.

Reading a failed run — an example

The list shows *invoices-import* in red. Select it.

01 The summary says **failed** after 12.8 s, with the message `DB Sink "Write invoices": duplicate key value violates unique constraint "invoices_pkey"`.

02 Press **errors only**: two events remain — `NodeFailed` for *Write invoices* and `RunFailed`.

03 Clear the filter and choose the node *Write invoices*: its `NodeProgress` events show it had written 4 200 rows before failing; the lane's transaction rolled them back.

04 Search for `invoices_pkey` to see whether the same error appeared in earlier runs of the plan.

05 Where logs are kept

Where	What	Written by
`~/.stilton/runs/<run_id>.ndjson` **on the run machine**	the **complete** log of the run, always	the runner
`<MONITOR_DATA_DIR>/<run_id>.ndjson` **on the monitor**	what the monitor received, filtered by the log level	the monitor, as events arrive
`stilton-monitor-fallback.ndjson` **next to the runner**	only events that could not be pushed	the runner

Consequences worth remembering:

- **The complete log is never lost**, even if the monitor receives only the essentials: it stays on the machine that ran.
- **The monitor reloads its files at start-up**, so a restart loses nothing.
- **Duplicates are discarded**, in memory and on disk.
- **Retention** (`MONITOR_MAX_RUNS`, `MONITOR_MAX_EVENTS`) trims only memory. The monitor never deletes files: cleaning the disk is your decision.

> **Warning.** Deleting files does not empty the list of runs, because the list lives in memory and the disk is read only at start-up. The right order is: **stop the monitor → delete the files → start it again**.

Linux — keep 90 days of run files — bash

```
find /var/lib/stilton-monitor/data -name '*.ndjson' -mtime +90 -delete   # monitor stopped, or then restart it
```

06 Saving and reopening a log

- **Save log** (in the banner) downloads the run as an NDJSON file.
- **Open log…** (at the top) opens an `.ndjson` file **from your computer** and shows it like a run: banner, summary, filters and search all work. The file is not imported into the monitor: it is a temporary view, closed with **close**.

You can open any log in the Stilton format: downloaded from the monitor, copied from `~/.stilton/runs/` of a run machine, or a fallback file. Logs that start with the runner's header line also show the artifact's metadata.

The log format

One JSON object per line. The first line describes the run; each following line is an event with the time it happened on the run machine.

run.ndjson json


```
{"kind":"flowpilot-log-header","logFormat":1,"runner":
{"os":"linux","arch":"x86_64","version":"0.9.0","host":"etl-01"},"artifact":{"planName":"orders-
daily","profile":"prod","studio":{"label":"mark-laptop"},"planHash":"sha256:5e1f…"}}
{"timestamp":1762495201204,"event":{"type":"RunStarted","payload":
{"run_id":"run-…","lane_count":2,"plan_name":"orders-daily"}}}
{"timestamp":1762495201207,"event":{"type":"NodeStarted","payload":{"node_id":"node_1","label":"Read
orders"}}}
```


07 Provenance of a run

The banner separates information by **how much you can trust it**:

Kind	Fields	Source
Declared	studio that compiled and its label, studio version, plan version, host declared by the runner, plan hash (*integrity*), log level	the signed artifact and the runner, reported as received
Observed	IP address	read by the monitor from the network connection — nobody declares it

The studio's label is set in the Compile panel (*Compiled by*) and identifies the installation, not a person. The plan hash is the one in the signed manifest: the runner ran the artifact only after verifying it. Behind a reverse proxy the observed IP is the proxy's.

08 Security and limits

The monitor is **closed by default**: nothing opens by accident.

Token	Used by	How
ingest (`MONITOR_TOKEN`)	runners	`Authorization: Bearer <token>` (or `X-Stilton-Token`) on `POST /ingest`
read (`MONITOR_READ_TOKEN`)	people, scripts	browser password (any user name), or `Authorization: Bearer` on the API

An ingest token does not allow reading, and vice versa: the ingest token sits on every run machine, and whoever steals one must not be able to read everybody's logs.

Defences of ingest: requests coming from a web browser are refused (403), only NDJSON/JSON is accepted (415), body size (413) and volume per IP (429) are limited, and the observed IP can only be written by the monitor.

Defences of reading: repeated wrong passwords from one IP block reads from that IP for a minute (429). The web page runs under a strict content security policy and treats every field of every event as untrusted text.

Limits, stated plainly

- **No built-in TLS.** On a network, tokens and events travel in clear unless a TLS reverse proxy (or a VPN) is in front. See *Security & Certificates* for a proxy configuration.
- **Tokens per role, not per person**: there are no user accounts and no record of who looked at what.
- **The monitor reports what runners send.** It does not re-verify artifact signatures, and no log proves on its own that a run happened as told. The only value no runner declares is the observed IP.

09 HTTP interface

Method	Path	Use	Token
POST	`/ingest`	runners push NDJSON here (`Content-Type: application/x-ndjson`)	ingest
GET	`/`	the web interface	read
GET	`/api/runs`	list of runs, and the data folder	read
GET	`/api/runs/<run_id>`	events and provenance of one run	read
GET	`/api/runs/<run_id>/download`	the run as an NDJSON file	read

Linux — failed runs of the last list, as JSON — bash

```
curl -s -u any:$MONITOR_READ_TOKEN http://127.0.0.1:8787/api/runs | jq '.runs[] |
select(.status=="failed")'
```

10 Troubleshooting

Symptom	Cause and remedy
The monitor does not start: *required tokens missing*	set `MONITOR_TOKEN` and `MONITOR_READ_TOKEN`, two different values of at least 16 characters
The browser asks for a user and a password	that is the read token: any user, password = `MONITOR_READ_TOKEN`
Unreachable from another machine	it listens on 127.0.0.1: publish it through a TLS reverse proxy, or set `MONITOR_BIND`
Pushes answer 401	the runner's `MONITOR_TOKEN` differs from the monitor's ingest token (the read token is not valid for ingest)
Pushes answer 413 or 429	batch too large or too much volume from one IP: raise the limits. The events are in the runner's fallback file
Nothing arrives	check the URL in the artifact (Compile panel) or `MONITOR_URL`; look for `stilton-monitor-fallback.ndjson` next to the runner
Runs I deleted are still listed	the list is in memory: stop → delete → start

Symptom	Cause and remedy
Few events in a run	the artifact's log level filters them — the banner says so — or a filter is active
The plan name or the provenance is missing	the artifact was compiled by an older studio: compile it again