Manual 04 · the machine where jobs run

Runtime Operations

Everything an administrator needs on a run machine — how the runner decides whether to run an artifact, how a run executes inside the engine, trust store, runtime profile, configuration and secrets, scheduling and services, logs and exit codes.

CONTENTS

01 **What happens when you start a run**
- 1.1 Inside a run

02 **Preparing a run machine**

03 **The trust store**

04 **The runtime profile**
- 4.1 Example profiles

05 **Configuration and secrets**
- 5.1 Configurable variables
- 5.2 Secrets

06 **Scheduling**
- 6.1 Linux — systemd timer (recommended)
- 6.2 Linux — cron
- 6.3 Windows — Task Scheduler
- 6.4 Chaining jobs

07 **Continuous artifacts as services**

08 **Logs and exit codes**
- 8.1 Exit codes
- 8.2 Environment variables of the runner

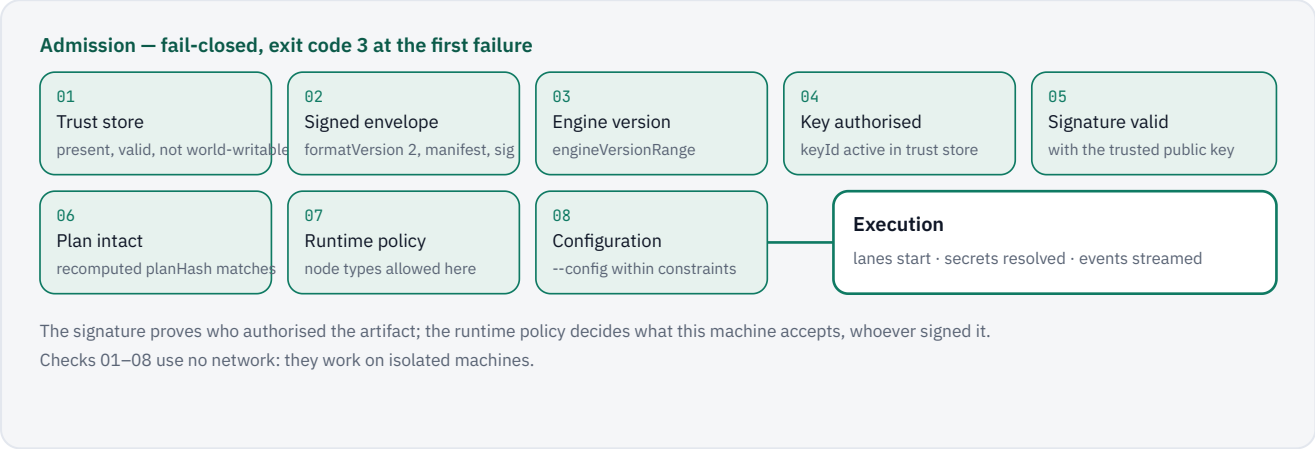
09 **Updating artifacts and the runner**

10 **Troubleshooting**

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

01 What happens when you start a run

`stilton-runner orders-daily.ffart` does two things, always in this order: **admission**, then **execution**. Nothing of the artifact runs until admission has passed every check.



On success the runner writes one line on the error stream and starts:

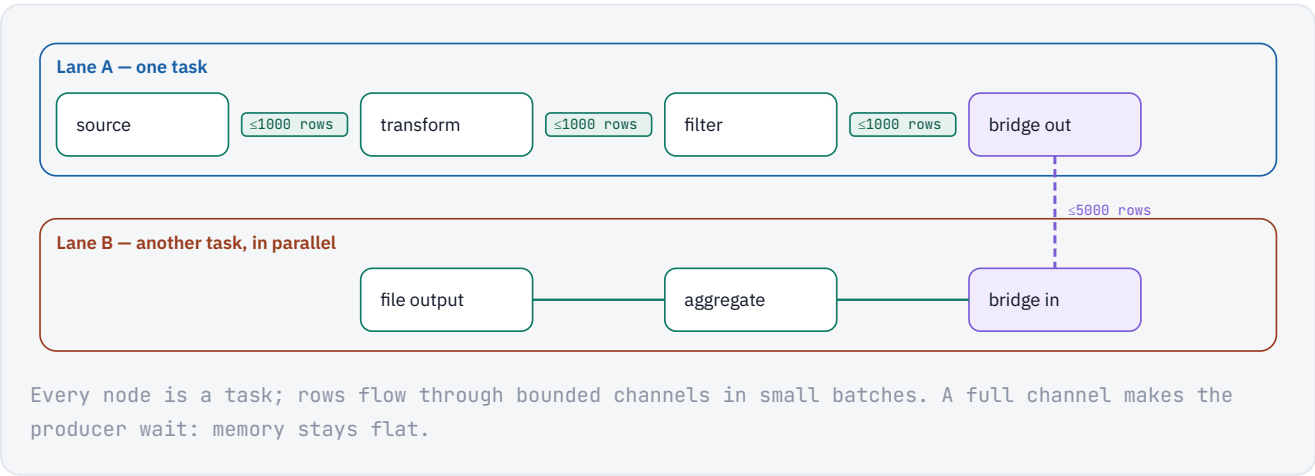
```text
signature OK: artifact signed by mark (ed25519:9f3c51a0…)
```

On failure it writes the reason, runs nothing and exits with code **3**:

```text
REFUSED: runtime policy: node types not allowed on this machine — shell_exec (Clean archive)
[~/.stilton/runtime.json]
```

Inside a run

Once admitted, the plan runs in the same engine as the studio.



Every node is a task; rows flow through bounded channels in small batches. A full channel makes the producer wait: memory stays flat.

- **Lanes run in parallel**, each as an independent task. Bridges are the only links between them.

- **Every node starts at once** and waits for its input. Rows travel in batches of up to 64 through channels holding up to 1 000 rows (5 000 on a bridge). When a channel is full, the producer waits — this *backpressure* keeps memory bounded however large the source.
- **Streaming nodes** (sources, transform, filter, TMap, file and database outputs) hold only the rows in flight. **Dataset nodes** — aggregate, sort-based windows, pivot, join's lookup side, materialize — hold their whole input in memory. Size the machine for them (see *Performance & Best Practices*).
- **Secrets** are resolved when a node opens its connection, from the runner's environment or the keychain. They are never written to logs.
- **Transactions** commit when their lane ends successfully, and roll back if any member failed.
- **The run ends** when every lane has ended. The run fails if any lane failed.

02 Preparing a run machine

A checklist for a new server, in order.

01 **A service account** — the jobs run as this account, never as `root` or an administrator. Its home directory holds the trust store, the runtime profile and the run logs.

02 **The runner** and its system libraries (*Getting Started*, chapter 4), and the database drivers the artifacts need (chapter 6).

03 **The trust store**: authorise the keys of the developers whose artifacts this machine may run.

04 **A runtime profile**, recommended in production: which node types this machine accepts.

05 **Secrets**: the values of the `requiredSecrets` of each artifact.

06 **Monitor access**, if used: `MONITOR_URL` and `MONITOR_TOKEN`.

07 **A schedule** or a service definition.

08 **A dry run**: `stilton-runner --admit-only` on each artifact, then one real run.

Linux — service account — bash

```
sudo useradd --system --create-home --home-dir /var/lib/stilton --shell /usr/sbin/nologin stilton
sudo -u stilton mkdir -m 700 -p /var/lib/stilton/.stilton
sudo mkdir -p /opt/stilton/jobs && sudo chown stilton: /opt/stilton/jobs
```

03 The trust store

The trust store lists the public keys whose artifacts this machine runs. It belongs to the account that runs the runner: `~/.stilton/trust-store.json`.

Linux — as the service account bash

```
sudo -u stilton stilton-runner trust add mark-entry.json --owner "Mark Rossi"
#   key authorised
#     keyId (fingerprint): ed25519:9f3c51a0…
sudo -u stilton stilton-runner trust list
```

output of trust list text

```
trust store: /var/lib/stilton/.stilton/trust-store.json
active   ed25519:9f3c51a0…  Mark Rossi
active   ed25519:41d0e7b2…  Anna Bianchi
revoked  ed25519:c8a91f33…  Mark Rossi (old laptop)
```

Command	Effect
`trust add <entry.json> --owner NAME`	authorises a key; `--owner` is required and appears in every audit line
`trust add --public-key <B64> --owner NAME`	the same, from the bare public key
`trust add - --owner NAME`	reads the entry from standard input
`trust list`	lists keys and their status
`trust show <keyId>`	shows one entry
`trust revoke <keyId>`	revokes a key; the line stays, for audit
`--store <path>`	on any command: works on another trust store file

Important. Before `trust add`, read the keyId to the developer on a second channel — phone or in person — and check it matches the one shown in their studio. It is the only step that binds a key to a person.

What the runner enforces:

- the file must exist, be valid JSON and **not be writable by others** (`chmod 600`; the folder `700`), or nothing runs;
- an entry whose keyId is not the fingerprint of its public key is ignored, with a warning;
- a revoked key cannot be re-activated with `trust add`: revocation usually means compromise;
- revocation takes effect at the next run — there is no cache.

04 The runtime profile

The signature says **who** authorised an artifact, not **what** it does: an artifact signed by a trusted developer with a *Shell* node runs that command. The runtime profile lets each machine refuse node types it should never execute, whoever signed them.

~/.stilton/runtime.json — json

```
{
  "formatVersion": 1,
  "description": "production ETL server — no shell, no SSH",
  "policy": {
    "deniedNodeTypes": ["shell_exec", "ssh_exec"]
  },
  "secrets": { "onlyDeclared": true }
}
```

Key	Meaning
`policy.deniedNodeTypes`	node types refused on this machine
`policy.allowedNodeTypes`	if present, **only** these types are accepted; a denied type always wins
`secrets.onlyDeclared`	`${NAME}` in resources is resolved only for the secrets the artifact declares — not for any environment variable of the machine

The runner looks for the profile in this order: `--profile <file>`, the variable `STILTON_RUNTIME_PROFILE`, then `~/.stilton/runtime.json`. Without a profile there are no restrictions. With one, the runner is strict: invalid JSON, an unknown key, a misspelt node type, a `formatVersion` other than 1, or a file writable by others → refused.

Example profiles

Machine	Profile
Production ETL server	deny `shell_exec`, `ssh_exec`; `onlyDeclared: true`
DMZ machine that only receives webhooks	allow only `webhook_receiver`, `webhook_responder`, `transform`, `filter`, `sink_db`, `log`
Developer's test box	no profile

Check an artifact against the profile without running it:

Linux — bash

```
stilton-runner --admit-only orders-daily.ffart
```

output — json

```
{"admitted":true,"keyId":"ed25519:9f3c…","owner":"Mark Rossi","planHash":"sha256:5e1f…",
 "planName":"orders-daily","nodeTypes":["filter","sink_file","source_db","tmap"]}
```

05 Configuration and secrets

Configurable variables

If the developer made some variables configurable (*Build & Publish*, chapter 5), you can change their values on this machine with a file:

```
/etc/stilton/orders-prod.json                                                    json
{ "profile": "prod", "values": { "THRESHOLD": "250", "DB_HOST": "db2.internal" } }
```

```
Linux                                                                            bash
chmod 600 /etc/stilton/orders-prod.json
stilton-runner --config /etc/stilton/orders-prod.json orders-daily.ffart
# or: STILTON_CONFIG=/etc/stilton/orders-prod.json stilton-runner orders-daily.ffart
```

Without `--config` the values chosen at compilation apply. The artifact is refused if the file sets a variable that is not configurable, a value outside its constraints, a value containing `${…}` or control characters, or if the file is writable by others.

Secrets

Each artifact lists the secrets it needs (`requiredSecrets` , shown by the studio's Compile panel and in the manifest). For each `${NAME}` the runner looks, in order, at:

01 the **environment variable** `NAME` of the runner's process;

02 the **system keychain**, service `stilton`, entry `NAME` — Credential Manager on Windows, Secret Service on Linux.

If neither has it, `${NAME}` stays as it is — the connection then fails with a message that shows the unresolved name, never an invented value.

Machine	Recommended source
Linux server (no desktop session)	environment variables from a file readable only by the service account
Windows server	environment variables of the scheduled task or service, or Credential Manager of the service account
A workstation that also has the studio	the keychain: the studio's *Environments* → *Secrets* section writes there

```
/etc/stilton/orders.env (chmod 600, owner stilton)                               ini
DB_PASSWORD=S3cr3t-value
API_TOKEN=tok_live_6f1d…
MONITOR_URL=https://monitor.internal/ingest
MONITOR_TOKEN=9b2e…
```

Warning. Do not put secrets on the command line (`DB_PASSWORD=… stilton-runner …` ends up in the shell history and in the process list). Use an environment file, the service definition or the keychain.

06 Scheduling

The beta has no built-in scheduler (the *Stilton Scheduler* arrives with version 2.0). Use the scheduler of the operating system: the runner is an ordinary command that returns an exit code.

Linux — systemd timer (recommended)

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

```ini
[Unit]
Description=Stilton job: orders-daily
Wants=network-online.target
After=network-online.target

[Service]
Type=oneshot
User=stilton
WorkingDirectory=/opt/stilton/jobs/orders
EnvironmentFile=/etc/stilton/orders.env
Environment=STILTON_CONFIG=/etc/stilton/orders-prod.json
ExecStart=/usr/local/bin/stilton-runner /opt/stilton/jobs/orders/orders-daily.ffart
StandardOutput=append:/var/log/stilton/orders.ndjson
```

/etc/systemd/system/stilton-orders.timer — ini

```ini
[Unit]
Description=Run orders-daily every day at 06:00

[Timer]
OnCalendar=*-*-* 06:00:00
Persistent=true

[Install]
WantedBy=timers.target
```

Linux bash

```
sudo mkdir -p /var/log/stilton && sudo chown stilton: /var/log/stilton
sudo systemctl daemon-reload
sudo systemctl enable --now stilton-orders.timer
systemctl list-timers stilton-*                # next run
journalctl -u stilton-orders.service -n 50     # last messages (stderr)
```

`WorkingDirectory` matters: relative paths in the artifact (`out/totals.csv`) are resolved from there.

Linux — cron

crontab -e (as the user stilton) text

```
0 6 * * *  cd /opt/stilton/jobs/orders && set -a && . /etc/stilton/orders.env && set +a &&
/usr/local/bin/stilton-runner orders-daily.ffart >> /var/log/stilton/orders.ndjson 2>>
/var/log/stilton/orders.err
```

Windows — Task Scheduler

Create a small wrapper script, then schedule it under the service account.

C:\Stilton\jobs\orders\run.ps1 powershell

```
Set-Location C:\Stilton\jobs\orders
$env:STILTON_CONFIG = "C:\Stilton\config\orders-prod.json"
C:\Stilton\stilton-runner.exe orders-daily.ffart | Out-File -Append -Encoding utf8
C:\Stilton\logs\orders.ndjson
exit $LASTEXITCODE
```

Windows (PowerShell, as administrator) powershell

```
$action  = New-ScheduledTaskAction -Execute "powershell.exe" `
           -Argument "-NoProfile -ExecutionPolicy Bypass -File C:\Stilton\jobs\orders\run.ps1"
$trigger = New-ScheduledTaskTrigger -Daily -At 6:00
Register-ScheduledTask -TaskName "Stilton orders-daily" -Action $action -Trigger $trigger `
           -User "DOMAIN\svc-stilton" -Password (Read-Host "password") -RunLevel Limited
```

Secrets for the service account: store them in its Credential Manager (log on once as that account and use the studio's *Environments → Secrets*, or set them as user environment variables of that account).

Chaining jobs

Because the exit code tells success from failure, chaining is a shell matter:

Linux bash

```
stilton-runner extract.ffart && stilton-runner load.ffart || echo "chain failed: $?"
```

07 Continuous artifacts as services

Some artifacts never end on their own: a *Dir Watcher* in watch mode, a *Webhook Receiver*, an *MQTT* or *ActiveMQ* source. They keep the run alive and process events as they arrive. Run them as a service, not as a scheduled task.

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

```ini
[Unit]
Description=Stilton service: invoices inbox watcher
After=network-online.target

[Service]
User=stilton
WorkingDirectory=/opt/stilton/jobs/inbox
EnvironmentFile=/etc/stilton/inbox.env
ExecStart=/usr/local/bin/stilton-runner /opt/stilton/jobs/inbox/inbox-watch.ffart
Restart=on-failure
RestartSec=30
StandardOutput=append:/var/log/stilton/inbox.ndjson

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

Stopping. `systemctl stop`, Ctrl+C or *End* in Task Scheduler ask the runner to stop cleanly — the same as **Stop** in the studio: listening nodes stop waiting, the lane finishes the rows already in flight, transactions are closed and the run ends with its outcome. A second request exits at once (exit code 130); then rows in flight are lost and the database rolls back open transactions.

in the [Service] section — ini

```ini
TimeoutStopSec=120      # time allowed for a clean stop before systemd kills the runner
```

Note. Design continuous flows so that each event is complete in itself — one file, one message, one transaction: then even a forced stop never leaves half an event written.

08 Logs and exit codes

A run produces three streams of information.



Stream	Content
stderr	the admission result (`signature OK …` or `REFUSED …`) and diagnostic messages
stdout	a header line (artifact, runner, provenance), then one event per line: `RunStarted` , `NodeStarted` , `NodeProgress` , `NodeCompleted` , `NodeLog` , `NodeFailed` , `RunCompleted` / `RunFailed` … filtered by the artifact's log level
local record	`~/.stilton/runs/<run_id>.ndjson` , every event unfiltered; change the folder with `STILTON_RUNS_DIR`
monitor	the same events as stdout, pushed to `MONITOR_URL` with `MONITOR_TOKEN` ; on failure appended to `stilton-monitor-fallback.ndjson` in the working directory

Each event line looks like this:

stdout — json

```
{"timestamp":1762502102512,"event":{"type":"NodeCompleted","payload":
{"run_id":"run-…","lane_id":"lane_a","node_id":"node_3","stats":
{"rows_in":1200,"rows_out":1187,"rows_rejected":13,"elapsed_ms":420,"error":null}}}}
```

Useful one-liners with `jq` :

Linux — bash

```
jq -c 'select(.event.type=="RunCompleted" or .event.type=="RunFailed") | .event' run.ndjson
jq -r 'select(.event.type=="NodeFailed") | .event.payload | "\(.node_id): \(.error)"' run.ndjson
jq -r 'select(.event.type=="NodeLog" and .event.payload.level=="error") | .event.payload.message'
run.ndjson
```

Exit codes

Code	Meaning	Typical reaction
`0`	completed	none
`1`	started and failed	read `RunFailed` / `NodeFailed` ; rerun after the fix
`2`	wrong usage or unreadable artifact	check the command line and the file
`3`	refused at admission — nothing ran	read stderr: trust store, signature, policy or configuration
`130`	stopped by a second termination request	the run was interrupted: check what it had written

Environment variables of the runner

Variable	Effect
`MONITOR_URL`	monitor endpoint (`…/ingest`); used if the artifact names none, or resolves `${MONITOR_URL}`
`MONITOR_TOKEN`	token to be accepted by the monitor
`STILTON_CONFIG`	configuration file, as `--config`
`STILTON_RUNTIME_PROFILE`	runtime profile, as `--profile`
`STILTON_RUNS_DIR`	folder of the local run records
`STILTON_RUN_ID`	forces the run id (useful to correlate with an external scheduler)
secret names	any `${NAME}` the artifact requires

09 Updating artifacts and the runner

- **A new artifact version**: copy it next to the old one, run `--admit-only`, then swap the file the schedule points to. Keep the previous file until the new one has run successfully once.
- **A new runner version**: install it next to the old one (`stilton-runner-0.9.1`), run `--admit-only` on every artifact the machine uses, then switch the link. Artifacts declare the runner versions they need; a runner that is too old refuses them at admission instead of failing halfway.
- **Removing a developer**: `trust revoke` their keyId on every machine. Their artifacts stop being accepted at the next run; recompile them under another key if they are still needed.

10 Troubleshooting

Message or symptom	Cause and remedy
`SIGNATURE REFUSED: trust store missing`	no key authorised for **this account**. Remember: `sudo -u stilton stilton-runner trust add …`
`SIGNATURE REFUSED: trust store writable by ANYONE`	`chmod 600 ~/.stilton/trust-store.json` and `chmod 700 ~/.stilton`
`REFUSED: key not authorised`	the artifact's keyId is not in the trust store, or is revoked
`REFUSED: planHash does not match`	the artifact was modified after signing: get a fresh copy from the developer
`REFUSED: runtime policy …`	the artifact uses a node type this machine denies. Change the profile only if that is really intended
`REFUSED: configuration …`	the `--config` file sets a non-configurable variable or a value outside its constraints

Message or symptom	Cause and remedy
A `${NAME}` in an error message	a secret is not set for the account that runs. With systemd, check `EnvironmentFile=`; with Task Scheduler, the account's environment
Works by hand, fails when scheduled	different account, environment or working directory. Compare `whoami`, `env` and `pwd` in both cases
The monitor shows nothing	`MONITOR_URL` / `MONITOR_TOKEN` not set for the account; look for `stilton-monitor-fallback.ndjson` in the working directory
Memory grows with the input size	a dataset node (aggregate, join lookup, materialize, pivot) holds its input: see *Performance & Best Practices*