

LOGBRANIK

FROM LOGS TO CONTROLLED ACTION

NGINX LOG SECURITY

Build a local IP guard,
signed policies, and a
reliable analysis pipeline



Aleksandar Popovic

NGINX OPEN SOURCE / PYTHON LAB

OPEN BOOK: CC BY 4.0 / CODE: MIT

Contents

Nginx Log Security	4
1. Understand the feedback loop	8
2. Draw the trust boundaries	11
3. Design around failure	14
4. Run the laboratory	17
5. Produce logs you can safely analyze	20
6. Move events without losing their meaning	23
7. Commit events before acknowledging them	26
8. Connect ordinary Nginx to the guard	29
9. Know whose address you are blocking	32
10. Build a detector you can explain	35
11. Sign policy, not transport assumptions	38
12. Order updates across retries and restores	41
13. Make expiry work without the central server	44
14. Give operators precise controls	47
15. Observe the guard as an operating system	50

16. Move durable processing to PostgreSQL	53
17. Preserve the fast path in a Go agent	56
18. Test behavior at the boundary where it matters	59
19. Diagnose failures in the right order	62
20. Release a reproducible system	65
Appendix A. Laboratory runbook	68
Appendix B. From lessons to sixty implementation tasks	71
Appendix C. Glossary and sources	75

Nginx Log Security

Build a local IP guard, signed policies, and a reliable analysis pipeline

LogBranik - From logs to controlled action

Author: [Aleksandar Popovic](#)

Open-source publication candidate, edition 0.2. Prepared October 2026.

Copyright and open licenses

Copyright (c) 2026 Aleksandar Popovic. The complete book text, exercises, answers, prompts, and original LogBranik visual assets are licensed under [Creative Commons Attribution 4.0 International](#). The [complete CC BY 4.0 legal code](#) governs that grant. Original code, code snippets and configuration examples are licensed under the [MIT License](#). The source package includes the full MIT notice and a detailed licensing map.

You may share, modify and redistribute these materials, including commercially, under the applicable license. Attribute content to Aleksandar Popovic, link CC BY 4.0, and identify changes. Retain the MIT copyright and permission notice with code copies. Third-party fonts and dependencies retain their own licenses. An optional paid reading edition does not restrict access to the open sources. This edition has not yet been published to a public repository.

MIT notice for original code

MIT License. Copyright (c) 2026 Aleksandar Popovic.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Introduction

An ordinary Nginx server already knows a great deal about the traffic reaching an application. Its access logs tell us which paths clients request, how the application responds, and how much work each request causes. Turning those observations into an automatic defense is possible without buying Nginx Plus. The difficult part is deciding what the observations mean, delivering a trustworthy decision, and keeping the application usable when a component fails.

This book builds that system from explicit contracts. An edge host keeps request checks local. A central service receives completed access events, identifies a narrow pattern of suspicious activity, and publishes short-lived IP decisions. An agent accepts signed policies and answers Nginx authorization subrequests. The design distinguishes an operator's intention from what an edge has actually applied.

The companion implementation is deliberately compact. Python and SQLite let you inspect persistence, deduplication, detection, signatures, revision ordering, and expiry without first operating several infrastructure services. Its live transport exercises use mutually authenticated HTTPS. The deployment design then extends those ideas to Vector, FastAPI, PostgreSQL, and a Go agent. That production design is a specification and migration path, not a claim that the small laboratory is production software.

Who should read this book

You should be comfortable with Linux processes, HTTP, a basic Nginx configuration, and reading Python. You do not need to know public-key cryptography. The relevant operations are introduced through a concrete policy protocol. The PostgreSQL and Go lessons assume that you can read SQL and understand a simple service boundary; they explain the architecture rather than teach either language from scratch.

Work on servers you own or have permission to administer. All example addresses use documentation ranges or loopback. The traffic exercises target a local echo application. They are designed to study defensive behavior without contacting another organization's systems.

What you will build

The laboratory includes a durable ingestion store, an explainable 404-path detector, a signed full-state snapshot, a local authorization handler, a separate mTLS control handler, and a periodic delivery process. You will observe allow, would-block, block, and revoke transitions. You will test old-policy rejection, duplicate ingestion, clock changes, and expiry without a central server.

Each of the twenty lessons ends with an exercise and an answer. The source of every lesson is a Markdown file in `lessons/`. The same content is exported to Leanpub manuscript files. The appendices contain a runbook, a bridge to the sixty-task implementation pack, a glossary, and source references.

Evidence and edition status

The delivered verification directory records the actual Python test run and environment. The core and live loopback mTLS tests were executed during preparation. The host does not support creating an `AF_UNIX` socket, so that integration test is explicitly skipped. Nginx, Vector, PostgreSQL, and Go were unavailable in the preparation environment. Their configuration examples and production exercises therefore require execution on the reader's deployment host. No end-to-end Nginx performance measurement or production readiness claim is made.

This distinction is part of the engineering method. A plausible configuration is not a verified deployment. A unit test of a policy function is not a test of an Nginx access phase. A successful loopback HTTPS exchange is not a fleet rollout. The acceptance gates in later lessons tell you which evidence belongs to which claim.

AI transparency

AI was used to generate the manuscript, companion code, exercises, and publishing materials. Automated checks were executed for the scope documented above. This candidate has not yet received an independent human technical review or an author's final editorial approval. Before commercial publication, the responsible author should review the text, reproduce the deployment exercises, and update this statement to describe the work actually completed. Do not replace this statement with a claim that AI was used only for proofreading.

Reading route

Read lessons 1-3 for the system model. Run lesson 4 before changing a server. Lessons 5-12 cover trustworthy data and control. Lessons 13-17 cover operations and the production migration. Lessons 18-20 provide testing, troubleshooting, and a reproducible release workflow. The appendix runbook is useful at the keyboard, but it does not replace the explanations.

LogBranik is a proposed project brand. It is not an official Nginx product and does not imply endorsement by Nginx, Leanpub, or any dependency author. The name has not received trademark clearance. There is no fabricated author biography, endorsement, benchmark, or customer story in this edition.

1. Understand the feedback loop

Start with the event that actually exists

Imagine that a client requests five nonexistent paths from your application. Nginx proxies each request, the application returns 404, and Nginx writes an access record. An analyzer can now see five completed observations. It cannot travel backward and prevent those requests. Its first useful action affects a later request from the same address.

That timing determines the system's scope. Access-log analysis can interrupt repeated activity after evidence accumulates. It does not inspect a request before it starts, and it does not terminate an already established stream. A slow response delays the corresponding access event. A long-lived connection may not produce its terminal record for a long time. Network capacity can be exhausted before this application-level loop has enough evidence.

Instead of promising instant protection, define a measurable reaction interval. Start it when the detector has enough eligible events to create a decision. End it when the edge has persisted and published the policy containing that decision. Separately measure the delay from request completion to ingestion. Those two intervals reveal whether collection or control delivery is the bottleneck.

Three different things called a block

An IP decision in a database is desired state. A signed message containing it is delivery state. A revision published by an agent is applied state. Calling all three "blocked" makes an operator dashboard misleading. A central service can create a ban while an edge remains disconnected. An HTTP sender can finish transmitting while the receiver rejects an epoch mismatch. An agent can accept a policy whose decisions are already expired.

The first vocabulary rule is to name the state precisely. A decision may be active centrally and unapplied locally. A delivery may be pending, retrying, or acknowledged. A local policy may be applied,

but contain no active bans. The edge's answer to a real check is the ultimate observation of request-time enforcement.

The laboratory exposes this distinction in its demo:

```
cd companion
python3 -m logbranic demo
```

The command prints an initial allow, an ingestion result, a monitor outcome, an enforcement outcome, and an allow after revocation. It uses a temporary directory and generated keys, so it does not alter an existing Nginx configuration or retain credentials after completion.

The two paths

The request path contains Nginx, a local authorization check, and the protected application. It should remain usable when the central analyzer is unavailable. The analysis path contains logs, transport, storage, detection, and policy delivery. It is slower, retryable, and permitted to touch disks and databases.

Moving a database query into every authorization check joins those paths. A database maintenance operation then becomes an application outage. Moving signature verification into every check creates unnecessary CPU work and makes verification latency part of every request. Perform those operations when a policy changes. Publish a complete in-memory view afterward.

The architecture diagram in the exported edition shows both paths. Arrows represent actual information transfers, not an assumption that all transfers succeed. Every arrow across a machine boundary needs authentication, a maximum size, a timeout, and a definition of success.

Write the first invariants

An invariant describes behavior that must remain true despite retries and failures. For this system, expired decisions must stop denying requests even while central services are offline. A replayed older policy must not resurrect a revoked decision. A collector must not choose another node's identity by putting it in a JSON field. A repeated event must not be counted as additional evidence. A lost acknowledgement must not force a decision's lifetime to restart.

These statements are more useful than a feature list because they tell you what to test. "Supports retries" says little. "An exact duplicate policy returns its existing acknowledgement without reapplying or extending bans" defines a reproducible behavior.

Exercise

Write a timeline for five suspicious requests, an event retry, a central decision, a failed delivery, and a later successful delivery. Mark the earliest request that could be denied. Which interval should an operational alert measure?

Answer

The five original requests finish before their access observations exist. A retry adds no evidence after deduplication. The earliest denied request is one arriving after the agent has published the policy, assuming the site is in enforce mode and the decision has not expired. Alert on excessive desired-to-applied delay and on collection lag independently. Combining both delays into one number hides the failing component.

2. Draw the trust boundaries

Begin with actors, not ports

A visitor requests application content. A collector sends observations from an edge. A publisher distributes policy. An agent evaluates policy locally. An operator changes modes, allowlists, and manual decisions. Each actor needs a distinct authority. A valid certificate is proof of possession of a credential; it does not, by itself, prove permission to perform every action.

The companion mTLS server therefore checks a certificate fingerprint against an explicit role registry after the TLS handshake. The collector credential is accepted at ingestion. The publisher credential is accepted at policy updates. A different certificate signed by the same laboratory CA cannot publish just because its chain validates. The transport tests include this exact case.

In a managed deployment you will probably use a certificate identity with a registration database rather than manually maintained fingerprints. Preserve the second authorization step. Transport identity must map to a role and allowed scopes. Do not derive role from a visitor-controlled header or from a body field that happens to say collector.

The local edge boundary

Nginx sends a deployment-configured site ID and a resolved client IP to the authorization handler. It overwrites the headers carrying those values. The handler must not receive arbitrary visitor headers or request bodies. Its socket permissions restrict which local users can connect.

A Unix socket avoids opening another TCP port, but it is not magical authentication. A compromised process with access to the socket can submit checks. A compromised edge root account can change Nginx configuration, replace agent state, or bypass the service entirely. This design does not claim protection against a hostile administrator of the enforcement host.

The lab has a loopback TCP option for hosts that cannot create Unix sockets. That option allows other local processes to connect and is for teaching. Use the Unix profile and operating-system permissions for the production boundary; verify it on the actual host.

Central observations are scoped

Suppose collector A is registered for site-demo. Its certificate maps to node-a and that site. A body containing site-other is not permission to ingest for another site. A body containing a fake node ID should be rejected by the schema, which has no node field. In the companion server, node identity comes from the certificate registry and is passed separately to the ingestion function.

The event schema is strict because accidental flexibility becomes authority. Ignoring unknown fields permits an old client to believe a new field is being enforced. Interpreting unknown site IDs as global scope creates cross-site consequences. Treat unknown versions, sites, and mutation types as explicit errors.

Decisions are data

A remote policy may describe an exact IP, a site, an expiry, and a reason code. It may not describe a shell command, executable path, firewall expression, arbitrary URL, or file to overwrite. The agent does not translate strings from a policy into commands. It evaluates a data structure.

This gives the control-plane compromise a defined blast radius. Someone who steals a signing key can still issue harmful valid decisions within the protocol's scopes and limits. They cannot obtain arbitrary code execution merely by adding a command field. Independent TTL caps, a local emergency bypass, and pinned site scopes reduce that damage without pretending it disappears.

A small threat table

Situation	Boundary that matters	Required response
Visitor spoofs guard headers	Nginx to agent	Overwrite identity headers
Collector sends another site	Ingestion authorization	Reject or quarantine scoped record
Valid CA certificate has wrong role	mTLS application authorization	Deny endpoint access
Old signed policy is replayed	Agent revision state	Reject lower revision
Signed policy has excessive TTL	Independent agent validation	Reject the policy
Root on the edge is hostile	Host administration	Outside this protocol's guarantee

Exercise

You are told to save time by sharing one certificate between collector, publisher, and operator. Explain the failure this creates and identify one test that would expose it.

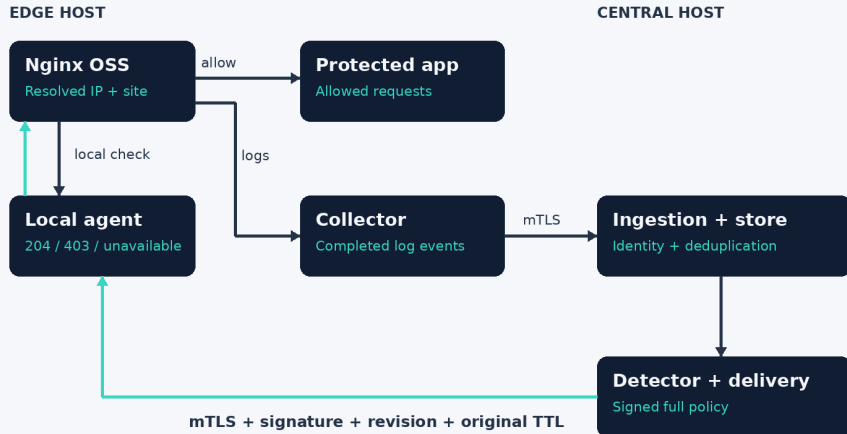
Answer

A stolen collection credential would gain mutation rights, so a less privileged component could become an enforcement controller. Use separate credentials and role authorization. Connect to the snapshot endpoint using a CA-valid collector certificate and require denial. The companion's wrong-role test demonstrates the same boundary with a dedicated wrong-role certificate.

3. Design around failure

Two paths, separate responsibilities

Local request checks remain independent of central analysis.



Deployment design; the compact lab substitutes Python/SQLite and finite captures.

Two paths: local authorization and central analysis.

Two machines, different responsibilities

Edge A runs Nginx, the collector, and an agent. Central B runs ingestion, durable storage, a detector, and delivery. The edge's request checks do not call Central B. The central side can inspect events and prepare policy without being in the path of every application request.

At a larger scale, several edges belong to one organization and serve registered sites. Each agent receives only its assigned sites. The initial design has one organization boundary. Adding customer tenancy later changes authorization, data retention, key ownership, and database constraints. Treat it as a new design task rather than adding a `tenant_id` field and declaring isolation solved.

Central outage

An already synchronized agent has a saved policy. If Central B becomes unavailable, the agent continues evaluating that policy. Each ban ends at its original expiry. The collector accumulates events only within its configured finite capacity. Once capacity is exhausted, behavior depends on the collector's backpressure and retention configuration; there is no infinite queue.

An outage therefore affects new knowledge, delivery, and stored observations. It does not necessarily affect ordinary request availability. The important exception is a first-boot agent without verified state. That agent cannot honestly say it has synchronized an empty policy. It reports not-ready until it receives a valid snapshot, including a signed empty one.

Agent outage

An absent local agent is a different case. The baseline protected Nginx location treats authorization errors as a failure. An allow response is 204. A deliberate deny is 403. A backend error is not an allow. Depending on configuration and build, the public result of an authorization error is generally an Nginx error response, often 500.

Some sites need a deliberate availability profile that permits requests if the local guard is down. That is a business decision about the protected application. Implement it inside the internal authorization location, where only selected backend error codes can be converted to an allow response. Never put a broad fallback on the public application location: it could transform a real application error or a real deny into success.

The production-design folder contains the optional fail-open example from the source pack. It is a candidate configuration until a real Nginx test verifies its status handling, request method, body preservation, and location routing. Do not enable it because the text looks plausible.

Database and filesystem failures

Ingestion returns success after its transaction commits. If storage fails, the response must invite a retry instead of pretending the data was saved. Policy publication similarly acknowledges only after persistent state and the live view agree. The agent test injects a disk-write failure and verifies that the previous view remains published and survives a new agent object reading the existing state.

A crash between persistence and live publication creates a short uncertainty window. On restart, the verified saved state becomes the source of truth. The sender may retry the exact policy and receive an idempotent acknowledgement. A crash before persistence means the sender has no valid acknowledgement and must try again.

Filesystem permissions also matter. State belongs in a directory an unprivileged service can write without granting arbitrary application users access. Replacing a file in a protected directory is different from writing through a path an attacker can redirect. Configure the directory deliberately and avoid world-writable runtime paths.

Availability is a policy

Do not label a system "fail-safe" without naming the failure. A central outage and an agent outage have different consequences. Monitor mode and emergency bypass have different purposes. Clock uncertainty and an ordinary expired ban must not be treated as the same event. Document each response in a small failure matrix and test the responses that matter to the deployment.

Exercise

An agent is offline from central services for thirty minutes. It previously applied a fifteen-minute ban. What should happen to requests from that IP after minute fifteen? What happens if the agent process itself disappears at minute five?

Answer

The live agent allows the address at its original expiry. A missing agent produces an authorization error under the baseline profile. These failures need separate alerts and recovery actions.

4. Run the laboratory

Set up an isolated working directory

Use Python 3.12 or a compatible newer release on Linux for the full laboratory. The preparation run used Python 3.12 and Cryptography 50.0.1. The requirements file provides a compatible major-version range, while the verification report identifies the version actually used. Reproduce tests with that recorded version before upgrading dependencies.

From the extracted book package:

```
cd companion
python3 -m venv .venv
. .venv/bin/activate
python3 -m pip install -r requirements.txt
python3 -m unittest discover -s tests -v
python3 -m logbranik demo
```

The demo runs in a temporary directory. Its stable outcomes are:

```
before (204, 'allow')
ingest {'accepted': 5, 'duplicates': 0, 'quarantined': 0}
monitor (204, 'would_block')
enforce (403, 'block')
revision 3
revoked (204, 'allow')
```

The request ID and decision ID are implementation data, not secrets in this example. Do not turn the example's fixed IP into a real-world allowlist or threat label. It is a documentation address used to make policy transitions reproducible.

Move from functions to processes

Initialize persistent lab state and create disposable loopback certificates:

```
python3 -m logbranik init
python3 -m logbranik certificates
```

These commands refuse to overwrite an existing state or certificate directory. The state directory contains a central signing key and an agent's public verification key because this is a single-machine lesson. In a two-machine deployment, copy only the public key, agent

identity, provisioned epoch, and authorized-site configuration to the edge. The central signing private key must not travel with it.

In terminal A, start the agent:

```
python3 -m logbranik agent
```

If your teaching host prohibits AF_UNIX sockets, use `python3 -m logbranik agent --tcp-auth`. That binds the authorization handler to loopback port 9101. It does not change the mTLS control listener, which uses loopback port 9443.

In terminal B, start ingestion:

```
python3 -m logbranik central-server
```

In terminal C, start periodic delivery:

```
python3 delivery_loop.py
```

The delivery process computes a full policy and sends it over mTLS. It checks the returned revision and payload hash. Its one-second loop is a teaching mechanism. The production design replaces repeated whole-state generation with durable desired-state changes, coalescing, a delivery outbox, and independent reconciliation.

Feed real schema-shaped events

Generate local fixture events without accessing another server:

```
python3 - <<'PY'
import time
from pathlib import Path
from logbranik.__main__ import fixture_event
from logbranik.protocol import encode
Path('events.ndjson').write_bytes(
    b'\n'.join(encode(fixture_event(i,time.time())) for i in
    range(1,6)))
PY
```

Send them through the mTLS ingestion endpoint:

```
curl --cacert lab-state/tls/ca.crt \
--cert lab-state/tls/collector.crt \
--key lab-state/tls/collector.key \
-H 'Content-Type: application/x-ndjson' \
--data-binary @events.ndjson \
https://localhost:9444/v1/events:batch
```

Inspect decisions with `python3 -m logbranic decisions`. Switch the registered site to enforce mode with `python3 -m logbranic mode enforce`. The delivery loop carries the updated mode to the agent. Check through the Unix socket:

```
curl --unix-socket lab-state/auth.sock -i \
-H 'X-Guard-Site: site-demo' \
-H 'X-Guard-IP: 198.51.100.23' \
http://localhost/v1/check
```

For the TCP teaching profile, use the same headers at `http://127.0.0.1:9101/v1/check`. Confirm a 403 only after enforcement mode arrives. A 204 with `would_block` is correct in monitor mode.

What this proves

The function demo proves the central-to-agent logic in one process. The process exercise proves transport, credentials, persistence, and live publication. Neither proves that Nginx calls the handler for every protected route. Lesson 8 adds that deployment gate.

Exercise

Submit the same `events.ndjson` twice. Then edit one field in a record while preserving its request ID and submit it again. Compare the ingestion counters.

Answer

The first retry is reported as duplicates and adds no evidence. The conflicting record is quarantined; it does not overwrite the original. The laboratory saves only the quarantine reason, avoiding a second raw copy of potentially sensitive input. A production quarantine should also have explicit retention, capacity, and access controls.

5. Produce logs you can safely analyze

Choose an observation contract

The analyzer needs an event identity, an event time, a site, a resolved client address, a query-free path, a method, a status, a duration, a byte count, and the guard outcome. It does not need authorization headers, cookies, request bodies, query strings, or arbitrary visitor metadata for the baseline detector.

That choice is both a privacy decision and an engineering constraint. A query string may contain a password-reset token. A body may contain personal data. A session cookie is an authentication credential. Collecting those fields first and promising to remove them later expands the number of systems that hold them. Start with a minimal record and require a reason for each addition.

Paths themselves may contain identifiers. A query-free path is therefore not automatically anonymous. Inventory your application's routes before enabling retention. If `/customers/123456/profile` is sensitive, replace selected path segments with a route pattern at a trusted normalization stage or exclude the route. Preserve enough information for the chosen detector without pretending all paths have equal sensitivity.

Nginx format

The source-pack example uses `log_format ... escape=json` inside the HTTP context. Quoting variables without JSON escaping allows quotes and control characters in a request to damage the event boundary. The example emits numeric values as strings so that Nginx's variable expansion remains predictable; normalization converts those strings before application validation.

The essential shape is:

```
{
  "schema_version": 1,
  "request_id": "00000000000000000000000000000001",
  "event_time": "2026-10-10T00:00:00+02:00",
  "site_id": "site-demo",
  "client_ip": "198.51.100.23",
  "method": "GET",
  "path": "/missing-one",
  "status": "404",
  "request_time_seconds": "0.002",
  "bytes_sent": "64",
  "enforcement": "allow"
}
```

Use Nginx's own request identifier, rather than accepting a visitor's X-Request-ID as the deduplication identity. You may forward the generated identifier to the application for correlation. If the application maintains its own identity, keep the two identities distinguishable.

Original path and completed outcome

An internal redirect can change the current URI and the final logging location. The example derives a query-free original path from `$request_uri` and keeps it separate from routing variables. Test internal redirects, error pages, and any special application locations in your configuration. A top-level access log can be replaced by a location-specific logging rule, so reading only the server block is insufficient.

The guard outcome is equally important. An application 403 and a guard-generated 403 have different meanings. If the detector counts its own denied requests as fresh hostile evidence, a short ban can sustain itself through feedback. The production contract records allow, block, would_block, bypass, unavailable, or unknown. The teaching rule admits only events whose outcome is allow.

Disable authorization-subrequest logging. One public request should not become a second security observation merely because the guard performed a local check. The separate agent metrics can count checks without putting them back into the suspicious-traffic detector.

Time units and normalization

Nginx's request duration is expressed in seconds. The ingestion contract uses milliseconds. Multiplying once by 1000 is correct; multiplying in both the collector and API is not. A two-millisecond request would otherwise become a two-second observation.

The production design keeps event timestamps as normalized UTC RFC3339. The compact teaching protocol uses integer Unix seconds. This is an intentional simplification and means that the two wire formats are not interchangeable. `normalize_logs.py` converts the Nginx example into the laboratory's format:

```
python3 normalize_logs.py guard-access.json > events.ndjson
```

Do not use that file as a production streaming collector. It reads a finite input and stops on invalid data. It is useful for inspecting a captured fixture and making a one-time lab submission. The Vector lesson addresses continuous collection and rotation.

Exercise

Capture one local request whose path contains a quote and whose query contains a disposable token. Confirm that the resulting event parses as JSON, that the query is absent, and that the request duration has the expected units.

Answer

The path should remain a single escaped JSON string. The query should not appear anywhere in the generated security event. After normalization, the duration should be milliseconds. If the event fails JSON parsing, inspect the log format before adding a tolerant parser. A tolerant parser can hide a broken producer and silently lose observations.

6. Move events without losing their meaning

Retry is normal

A collector sends a batch, the database commits it, and the response is lost. The collector cannot know whether the commit happened. Resending is the correct action. The receiver must make that retry harmless. Network delivery and application effects are separate concerns; a transport's acknowledgement does not remove the need for application deduplication.

The production collector is Vector. It tails a known JSON access-log path, normalizes fields, and sends bounded NDJSON batches over authenticated HTTPS. Its HTTP sink offers retry-oriented delivery behavior, but durable end-to-end acknowledgement depends on the capabilities of the complete source-to-sink topology. Validate the exact file-source and buffer behavior you deploy rather than turning a sink label into a guarantee about every log line.

A buffer has a physical size

Choose a capacity based on an outage you intend to tolerate. If normalized events average 800 bytes and you expect 100 events per second, one hour represents approximately 288 MB of payload before overhead. Filesystem metadata, collector records, variable event sizes, and operational headroom increase the real requirement. A capacity estimate is a planning calculation, not a measured buffer utilization figure.

Set a finite disk buffer and alert on occupancy and age. The oldest buffered event tells you how stale central knowledge has become. The buffer's percentage tells you how close you are to capacity. Both matter: a quiet site may have low occupancy while carrying hours-old observations.

Backpressure is not always loss-free. A collector that stops consuming may let the local log grow until rotation removes unread data. A collector that drops oldest records preserves freshness while losing history. A collector that drops newest records preserves old

history while delaying knowledge of current activity. Make that tradeoff explicit and monitor the corresponding drops.

NDJSON and batch boundaries

NDJSON is one JSON object per line. A JSON array is a different format. A final newline is harmless if the receiver supports it. Empty batches, oversized lines, and an oversized decoded body need a defined response. Keep collector batch limits below the API's limits, including normalization and framing overhead.

The lab contract caps the body at 1 MiB, the batch at 500 records, and a line at 16 KiB. The teaching HTTP handler accepts Content-Length framing and does not implement chunked bodies or gzip. That scope is documented rather than hidden behind a broad HTTP claim. A production ingress can support compression, but must enforce the decoded maximum to avoid a small compressed request expanding into unbounded work.

The original Vector configuration remains in `companion/production-design/examples/vector/`. It is an adaptation starting point for the production RFC3339 protocol. It is not directly connected to the lab's integer-time server. Use the finite normalizer or write an explicitly tested lab transform before connecting them.

Log rotation

Rotation can rename the current file and ask Nginx to reopen a new one. A collector may track the old file by fingerprint while discovering the new file. Copy-and-truncate creates a different set of races. Your deployment test should keep traffic flowing through rotation and compare request IDs observed before and after it.

Testing a cold startup against one small file does not prove rotation behavior. Restart the collector with pending data, rotate a file while it is reading, and then restore the central endpoint after an outage. Count unique request IDs. Record duplicates separately from missing IDs. Duplicates are expected under retries; missing IDs are a different problem.

Permanent failures

A retryable 503 and a malformed 400 must not look identical in dashboards. A permanently invalid batch can occupy a retry loop and starve valid traffic. The ingestion design quarantines individual invalid records while committing valid neighbors. Entire-request errors, such as an invalid content type or decoded-size violation, remain explicit request failures.

Check the deployed collector's handling of every response class in this contract. Do not assume it retries every non-2xx status or that every permanent drop generates an operator-visible signal.

Exercise

Plan a ten-minute central outage at 100 events per second with 800-byte normalized events. What payload capacity does the arithmetic suggest, and what must a deployment measurement add?

Answer

The payload estimate is 48 MB: 100 multiplied by 600 multiplied by 800. Add framing, buffer overhead, filesystem headroom, and the observed event-size distribution. Measure disk consumption and backlog-drain time. A nominal 48 MB setting would leave no margin and would not prove ten minutes of operational tolerance.

7. Commit events before acknowledging them

Validate at the boundary

An ingestion API must bound work before parsing the entire request. Limit body bytes, line bytes, event count, and field lengths. Check version, request-ID shape, authorized site, valid address, method, path, status, duration, byte count, and outcome. Reject nonfinite numbers and booleans where the schema requires an integer.

Why distinguish a boolean from an integer? Python considers `True` an instance of `int`. A loose revision or status validator can therefore accept a JSON boolean as an apparently valid numeric value. The laboratory's integer helper checks the exact type. Protocol schemas must describe wire types, not just the language's inheritance hierarchy.

The parser also rejects duplicate JSON keys. Two fields with the same name can be interpreted differently by separate components. A signature may authenticate the bytes while one component sees the first value and another sees the last. Removing that ambiguity is a protocol decision, not merely a stylistic preference.

Define deduplication identity

The uniqueness key is authenticated node identity plus generated request ID. A request ID alone may collide across independently configured producers. A node field in the body is not authenticated identity. The TLS registry supplies the node separately.

For a first event, persist its validated representation and an immutable digest. For an identical retry, count a duplicate and preserve the original. For a retry with changed content, quarantine the conflict and retain the original. Changing content under an existing identity is not a normal retry; it is evidence of a broken or hostile producer.

The laboratory hashes a stable encoding of the validated incoming JSON before normalizing the client IP. This means reordered object keys do not cause a conflict, while changed values do. IPv4-mapped

address normalization still gives the detector and agent a common address identity.

Transaction scope

The teaching implementation serializes an ingestion transaction using a process lock and SQLite's immediate write transaction. It inserts new observations, updates detector effects, and commits before returning counters. If an entire batch is too large, no transaction begins. Invalid individual lines become bounded reason records; valid neighbors can commit.

This model intentionally favors clarity over throughput. A production system usually separates ingestion from detector work, while preserving an atomic connection through durable job rows. The transaction inserts the event and the pending job together. A notification can wake a worker, but the durable row is what survives a restart.

The important invariant is that success means the contract's required data is durable. A successful process-local queue insertion is not enough. A lost response is handled by deduplication; a lost in-memory event after a successful response cannot be repaired by a collector that believes its work is complete.

Distinguish storage and processing

The event's ingest status should not be confused with detection status. An accepted event may be too old for a fresh ban. It may have an outcome excluded by the detector. It may be valid history without triggering any rule. Saving every valid event does not mean every event is allowed to affect enforcement.

Persist the reason for processing ineligibility when it helps investigation. In the compact schema, the `eligible` flag is sufficient for the demonstration. A production schema should retain a bounded reason such as stale event, future timestamp, guard feedback, or unsupported observation context.

Keep poison data bounded

Quarantine is not a limitless second database. Cap record count, sample size, and retention. Do not copy credentials into an error message. Avoid returning detailed server internals to a producer. The laboratory stores only a short validation reason and timestamp. Production operations may need a redacted sample, but that is a deliberate addition with access controls.

Exercise

The API inserts ten events, returns 200, and then crashes before creating detector jobs. Is retrying the same batch enough to recover?

Answer

Not if duplicate handling skips the already inserted events and no durable pending work exists. The system can permanently miss their detection effects. Insert events and durable jobs in one transaction, or make detector discovery of unprocessed events a durable, repeatable scan. The lab avoids the gap by completing detection effects within its ingestion transaction.

8. Connect ordinary Nginx to the guard

Verify the module before editing configuration

Nginx Open Source can authorize a request using an HTTP subrequest when the `auth_request` module is present. It treats successful 2xx authorization responses as allows and 401/403 as denials. Other authorization statuses are errors. The module is not enabled in every possible build, so inspect the build you actually run:

```
nginx -V
```

Check the configure arguments or distribution documentation for `--with-http_auth_request_module`. Do not infer support from a different host or from an Nginx Plus example. The system in this book needs no Plus management API and does not change the Nginx configuration for each ban.

Protect a real content location

The internal authorization location proxies to the local agent. The public application location invokes that subrequest and then proxies allowed traffic to the application. The core structure is:

```
location / {
    auth_request /_guard_auth;
    auth_request_set $guard_outcome
        $upstream_http_x_guard_outcome;
    proxy_pass http://127.0.0.1:8081;
}

location = /_guard_auth {
    internal;
    auth_request off;
    access_log off;
    proxy_pass http://unix:/run/logbranic/auth.sock:/v1/check;
    proxy_method GET;
    proxy_pass_request_body off;
    proxy_pass_request_headers off;
    proxy_set_header Content-Length "";
    proxy_set_header Host localhost;
    proxy_set_header X-Guard-IP $remote_addr;
    proxy_set_header X-Guard-Site "site-demo";
    proxy_next_upstream off;

    proxy_cache off;
}
```

This is a structural excerpt. The delivered configuration files add logging and timeout settings. Socket paths and permissions must match your deployment. Use the generated lab configuration described in the runbook instead of pasting this excerpt as a whole server.

Do not use `return 200` in the public location as your proof that authorization works. Nginx processing phases matter, and a `rewrite-phase` `return` can finish a request before the `access` phase you intend to test. The companion `echo` application supplies an actual content upstream so the test exercises the relevant path.

Keep bodies and identity separate

The authorization subrequest needs an address and site, not the visitor's upload. Clearing its body prevents a guard call from waiting on content it does not inspect. This does not remove the body from the main application request. Verify that a permitted POST still reaches the upstream with its method and body intact.

Overwriting guard headers is essential. A visitor may send `X-Guard-IP` claiming to be an allowlisted address. That input must never become the address the agent evaluates. The site ID similarly comes from

deployment configuration rather than the visitor's Host header or a body parameter.

Audit every route

Location selection can route static files, error pages, API paths, and named locations differently. An existing authentication integration may already use `auth_request`; simply replacing it would remove application authentication. Compose the checks deliberately or preserve the existing mechanism through a supported integration design.

Also inspect `satisfy` settings and IP access rules. A combination that allows one access module to satisfy authorization may bypass the guard when another module permits the request. The acceptance matrix should include every route class and every preexisting authentication rule.

Reload once, then update policy

Installing the integration requires configuration validation and a controlled Nginx reload. After installation, bans and revocations change agent state rather than Nginx files. The claim "no reload for a ban" applies to policy changes, not to the initial integration or future routing edits.

Exercise

On a disposable Linux host, test a permitted GET, a denied GET, a POST body, an attempted direct request to `/_guard_auth`, and a spoofed guard header. Record the public response and whether the upstream received each request.

Answer

Allowed requests reach the upstream. A guard denial returns 403 without contacting it. A permitted POST preserves method and body. The internal location is not publicly callable as an ordinary route. A spoofed header does not alter the resolved identity. These observations require actual Nginx execution; the preparation environment did not run this deployment gate.

9. Know whose address you are blocking

Direct edges are the easy case

When a visitor connects directly to Nginx, the peer address is the simplest identity available. Ignore a visitor-supplied forwarding header unless an explicitly trusted proxy stands between the visitor and the edge. A header is data; it becomes identity only through a configured trust relationship.

When Nginx sits behind a load balancer or CDN, the peer address belongs to that intermediary. Blocking it would block many users. The correct solution is to identify the actual ingress topology and trust only the intermediary's legitimate peers when replacing the address. The real-IP module provides the mechanisms, but the deployment supplies the trust list.

Model the complete path

List each hop in order: visitor, CDN, load balancer, edge, application. For each hop, write who can connect and which party sets or appends the forwarding field. If the origin is reachable directly from the internet, a visitor may bypass a CDN's header discipline. Restrict origin connectivity as part of the topology rather than relying only on the expected path.

A trust configuration with `0.0.0.0/0` or `::/0` says that every peer may choose the replacement address. That defeats the identity boundary. Similarly, trusting an entire private network can grant unrelated services the ability to impersonate clients. Use the actual ingress peers and maintain their changes as controlled configuration.

Preserve the original peer address where useful for audit. It helps distinguish a client identity resolved through a trusted chain from the TCP peer that actually connected. The request-time check should use the resolved address; investigation may need both.

Normalize once

IPv6 addresses have multiple textual representations. The agent and detector should compare parsed addresses, not strings copied from separate serializers. The laboratory maps IPv4-mapped IPv6 addresses to their IPv4 identity. It rejects zone identifiers such as `%eth0`, hostnames, and malformed addresses.

A zone identifier is meaningful to local interface addressing, but is not an internet client identity for this protocol. Allowing it into an exact-IP ban key can create an address that the producer and consumer interpret differently. Document your normalization rules and use the same test fixtures across languages when migrating the agent to Go.

An IP is not a person

Many clients can share one public address through NAT. A corporate proxy may represent an entire office. A mobile network can change a device's address. An attacker may distribute requests across many addresses. The system is controlling traffic attributed to an address, not proving a person's identity or intent.

This explains several conservative defaults: automatic decisions are site-scoped, exact-address, and short-lived. A global ban or a broad network ban has a larger blast radius. It requires separate operator intent and evidence. A detector that is accurate for one site's workload may be wrong for another site's clients.

Detection and enforcement must agree

Suppose the collector stores the resolved client address, but the guard check receives the original load-balancer peer. Central can create a perfectly valid decision that never matches a local check. The opposite mismatch can accidentally ban the intermediary. The system's logs may look healthy while protection is ineffective or harmful.

Add a deployment test that records a known client's resolved address in both the access event and agent check. Include direct connections, trusted proxy connections, and malicious forwarding values from an untrusted peer. A unit test of `ipaddress` cannot establish Nginx's trust chain.

Exercise

Your origin uses a CDN and is also reachable directly. A request arrives directly with a forwarding header containing an allowlisted IP. What address should be evaluated, and what other deployment control would simplify the boundary?

Answer

The direct peer is not a trusted CDN peer, so its forwarding claim must not replace the client identity. Evaluate the actual peer. Restrict origin access to the intended ingress peers when the architecture requires CDN-only entry, then test both accepted proxy traffic and refused direct traffic. Keep an intentional management path separate from public visitor access.

10. Build a detector you can explain

Begin with a narrow rule

The laboratory rule counts distinct paths that returned 404 from one site and one address in the last sixty seconds. At five distinct paths it creates a fifteen-minute decision. Those numbers are teaching defaults, not universal threat thresholds. A normal browser can encounter several missing assets after a broken release. A search engine may revisit old links. A customer behind a corporate proxy may share an address with a crawler.

The point of the rule is to make the decision observable. An operator can inspect the paths, times, address, and rule version. A more complicated rule without explanations is harder to calibrate and harder to revoke confidently. Add complexity after you can reproduce both valid and false-positive cases.

The lab deliberately excludes unknown and guard-generated outcomes. Only an allowed request contributes to the demonstration's evidence. This is conservative: an unavailable guard or an uninstrumented route should not silently become the same trusted observation as an allowed protected request.

Event time and processing time

Event time says when the request completed. Processing time says when the analyzer receives it. A collector outage can separate them by hours. If the detector uses processing time for every arriving event, restoring a collector can turn old traffic into a new ban.

The teaching implementation stores old events but permits a fresh decision only from events within its freshness policy. It also rejects excessively future-dated events as detection evidence. The time-window query itself excludes events later than the current processing time. Events in the small future-skew tolerance are retained but do not immediately inflate the current window.

In production, make freshness and allowed clock skew explicit configuration with metrics. A high stale-event count can indicate

transport backlog. A high future-event count can indicate clock configuration problems or a hostile producer. Both should be visible without automatically punishing visitor addresses.

Deduplication precedes counting

The detector operates after event identity has been accepted exactly once for its effects. Counting then deduplicating raw history is too late: the earlier count may already have created a decision. A retry must not increment a counter, create another detection, or extend an active decision.

The lab's transaction handles this ordering directly. A production worker should mark a job's completion and commit its counter changes and decision creation together. If a worker crashes after updating a counter but before marking the event processed, repeating that job must not count it twice.

Do not renew by accident

An active decision has an original expiry. New delivery messages carry the same expiry. Repeated activity does not automatically reset it in the teaching model. The detector also keeps a short persisted cooldown so that still-visible old window evidence cannot immediately create a replacement decision after a revoke or a very short TTL.

A production renewal policy, if needed, must be a separate decision rule. It should require fresh evidence after a defined boundary, have a maximum cumulative duration, and remain distinguishable from transport retries. Operators should be able to tell whether they are seeing a first decision, a deliberate renewal, or a delivery retry.

Measure false positives in monitor mode

Before enabling enforcement, collect would-block outcomes and compare them with known application behavior. Review the top affected paths, addresses with many legitimate requests, shared network exits, and events around deployments. Monitor mode tests the rule's consequences without denying those requests.

Do not infer false-positive rate from a test fixture designed to trigger the rule. The fixture proves implementation behavior. It says nothing

about your application's normal traffic. Use representative owned traffic and labeled operational cases for calibration.

Exercise

Create five 404 events with different request IDs but the same path. Then create five with different paths. Retry both batches. Finally submit five events older than the freshness limit. What should create a decision?

Answer

The single-path batch does not meet the distinct-path rule. The different-path batch can create a decision. Replaying either batch adds duplicates rather than evidence. Old events remain valid stored history but cannot create a fresh ban. To isolate each case, use a new state directory or different documentation addresses so that previous decisions and cooldowns do not affect the next scenario.

11. Sign policy, not transport assumptions

Two different protections

mTLS authenticates both ends of a transport connection and protects data in transit. A policy signature authenticates the policy bytes independently of the connection that carries them. The two mechanisms answer different questions. A valid TLS connection does not prove that the receiver got the intended policy from the intended signing authority. A valid signature does not decide which network peer is permitted to use the update endpoint.

The laboratory uses Ed25519 signatures from the Cryptography library. It does not implement a signature algorithm. The central process holds the private signing key. The agent receives pinned public verification keys through provisioning. An incoming policy cannot introduce its own trusted verification key.

Define the signed bytes

The envelope contains a key identifier, a base64 payload, and a base64 signature. The signature covers a fixed domain separator followed by the exact decoded payload bytes:

```
DOMAIN = b"nginx-guard/snapshot/v1\x00"  
signature = private_key.sign(DOMAIN + raw_payload)  
public_key.verify(signature, DOMAIN + raw_payload)
```

The domain separates this use from other signed message types. The zero byte is part of the protocol. Changing it changes the message. Do not replace the payload bytes with a newly serialized JSON object before verification. Whitespace and key ordering can change the bytes even when an object seems equivalent.

The sender may choose a deterministic JSON encoding when generating a new policy. The receiver still verifies the received bytes. Deterministic generation helps produce reproducible fixtures; it is not permission for the verifier to rewrite an incoming message.

Verification is only the first gate

After verifying the signature, validate the decoded policy. Check the version, agent identity, provisioned epoch, positive revision, issuance and delivery interval, complete authorized site map, mode values, allowlist scope, decision count, address syntax, and independent ban lifetime limits. A malicious but correctly signed policy remains possible if a signing credential is compromised.

Reject unknown fields. The teaching policy deliberately has no command field. Reject duplicate keys and ambiguous numeric types. Enforce byte limits before parsing and collection limits before constructing the live view. These are security and capacity boundaries, not merely convenient error messages.

The lab schema differs from the production protocol in timestamp representation and decision metadata. Both use the same signing concept and domain, but they are not interchangeable clients. A production implementation must match the full RFC3339 contract and use a shared test-vector suite. Do not advertise compatibility based only on a similar JSON shape.

Key rotation

Provision a new public key before sending policies signed by its private counterpart. During a controlled overlap, the agent can pin both old and new key identifiers. Confirm that every intended agent accepts the new key before retiring the old one. Do not remove a key merely because new delivery succeeded on one edge.

Retiring a compromised key introduces a second concern: a saved policy may have been signed by it. Define how verified persisted state is refreshed and how the old key is removed without silently leaving an agent unable to restart. Local operator recovery and a new clean snapshot should be part of the rotation procedure.

A production signing key belongs in central secret storage with deliberate access and backup rules. The lab's raw key file is for an isolated example. The book package contains source that creates keys; it must not contain generated private keys or populated lab-state directories.

Exercise

Save a valid envelope. Change one payload byte without updating the signature. Then generate another valid envelope with the same revision but a different site mode. Why are both rejected, and by which gates?

Answer

The byte edit fails signature verification. The second envelope may pass signature verification because it was signed by the correct key, but fails the revision-conflict rule after a policy with that revision has already been applied. Cryptographic authenticity and safe state ordering are independent requirements.

12. Order updates across retries and restores

A signature does not make a message recent

A revoked ban can still exist inside an old valid signature. Without ordering, replaying that envelope could reintroduce it. The agent therefore remembers the highest applied revision for its provisioned stream epoch. An incoming lower revision is rejected even if its signature is valid and its delivery interval looks acceptable.

A full snapshot is a complete replacement of remote desired state. It lists every assigned site, the central allowlist, and the active decisions. Missing decisions are removed. An empty decision list is a legitimate policy that clears remote bans. Revision gaps are harmless because the agent does not need every intervening change to reconstruct the latest state.

Three ordering cases

Incoming revision	Payload digest	Result
Lower than applied	Any	Reject stale policy
Equal to applied	Equal	Return existing acknowledgement
Equal to applied	Different	Reject revision conflict
Higher than applied	Valid new policy	Persist and publish replacement

An exact duplicate can be acknowledged after its delivery window expires because it changes nothing. A new revision cannot use that exception. Its delivery window must still be valid when applied. This distinction lets a sender recover a lost response without reopening a stale update path.

The acknowledgement contains agent identity, epoch, revision, and exact payload digest. A generic 200 is not enough. The sender checks that the receiver applied the content it intended to deliver. The

companion's HTTPS push function performs those checks before reporting success.

Persist the high-water mark with state

The revision, payload hash, and policy must survive a restart together. Saving a revision in one file and a ban list in another creates inconsistent crash states. The lab persists the signed envelope atomically and derives its verified high-water mark on restore. Production may use a different on-disk representation, but it must preserve that atomic relationship.

Update handlers are serialized. Request checks read one published immutable view. During a replacement, they can observe the old view or the new view, but never a partially mutated decision map. The test suite runs checks during publication and accepts only complete old or new outcomes.

Push and pull have different jobs

Push reduces reaction time. Periodic authenticated pull reconciliation repairs a missed push and supports outbound-only edges. Both routes should pass through the same policy validator. Implementing one validator for push and a looser one for pull creates a second update protocol by accident.

The production design starts with a jittered fifteen-second reconciliation interval. That is a design default, not a measured service guarantee. The compact lab uses periodic push and does not implement an independent pull endpoint. Treat pull as a production task with its own timeout, authorization, revision, and expiry tests.

When a pending envelope expires before application, recompute current desired state and issue a new revision. Keep each existing decision's original expiry. Refreshing an envelope's delivery interval must not refresh the contained bans. Retrying a still-fresh envelope can reuse its exact stored bytes.

Recover from central restore

A database restored from backup may have a revision counter lower than one already applied at an edge. The agent is correct to reject those older messages. Do not fix the error by disabling revision checks. Recover authenticated edge high-water marks and reconcile central counters under operator control, or explicitly re-enroll an agent with a new locally provisioned epoch.

The epoch is a stream identity, not a random field central may replace on every restart. The agent does not auto-accept a different epoch from a remote policy. Losing the edge's local state similarly requires a controlled fresh bootstrap; it is not evidence that an older snapshot should be trusted.

Exercise

An edge has applied revision 42. Central is restored to revision 30 and sends a freshly signed revision 31. What should the agent do? What must an operator recover?

Answer

The agent rejects revision 31. A fresh signature and timestamp do not make it newer than applied state. The operator must restore a central revision high-water mark consistent with authenticated edge reports, or deliberately provision a new epoch through the controlled recovery procedure. The lab has no automatic epoch reset command because that would conceal the trust decision.

13. Make expiry work without the central server

Two clocks, two purposes

Wall time connects a policy to its UTC issuance and expiry. A monotonic clock measures elapsed time within one running process. Wall time can move because of a synchronization correction or a configuration mistake. Monotonic time does not provide a timestamp you can carry across a restart.

When accepting a ban, the lab computes a local monotonic deadline from its remaining wall-time lifetime. A check requires both wall-time and monotonic conditions to say that the decision is active. If the wall clock moves backward, the monotonic deadline prevents a running process from extending the ban indefinitely.

This is a useful local safeguard, not a complete time-service implementation. The lab also compares elapsed wall and monotonic time against an anchor and returns unavailable after a large divergence. A production agent needs monitored time health and a documented recovery policy. Simply having two clocks does not prove that the initial wall time was correct.

Envelope freshness is not ban lifetime

An envelope may be eligible for delivery for two minutes. A contained decision may last fifteen minutes. The delivery interval limits when a new policy can be accepted. It does not invalidate already applied state at minute two. On restart, a saved signed policy can still contain a live decision even though its original delivery interval ended.

Conversely, creating a fresh envelope does not extend a decision. The ban's original expiry is carried forward. Reissuing the same address under a new decision ID is a new decision, with a separate detection and operator policy. It must not occur as an accidental side effect of delivery retries.

The test suite advances an injected clock without sleeping. It checks local expiry, refresh without extension, restore after delivery expiry, and unavailability after backward wall-time movement. These tests

are deterministic because they do not depend on a fifteen-minute real wait.

Restart with uncertainty

After a restart, the monotonic clock has a new origin. Reconstruct remaining lifetime only when wall time is trustworthy. The production design requires unavailable state and an alert if time health cannot establish remaining TTL safely. Central reconciliation can provide new policy, but the edge still needs a credible time basis for independently enforcing maximum lifetimes.

The compact Agent accepts a supplied clock and rejects saved policy issued too far in the future. It cannot independently certify NTP health. That responsibility belongs to deployment supervision or a fuller production agent. Do not convert the injected-clock tests into a claim that all host clock failures are handled automatically.

Local policy precedence

Emergency bypass and locally provisioned emergency addresses come first. They exist so that an operator can recover a host even if a remote policy is harmful. Central exact-address allowlists come next. Active decisions are evaluated only after those exemptions. Monitor mode records a would-block rather than denying.

The lab exposes a process-local bypass flag for testing, not an unauthenticated network mutation endpoint. Production operations should expose a controlled local mechanism with ownership, audit, and visible health state. A forgotten bypass is a security configuration change that should remain obvious to operators.

Prune without requiring delivery

Request checks ignore expired bans even if their entries remain in the saved policy. Background pruning can reduce memory and clean derived indexes, but expiry correctness must not depend on a successful pruning thread or a new central snapshot. This makes the behavior robust during network outages and scheduling delays.

An agent also needs limits on active policy size. Full snapshots simplify ordering at the cost of payload and memory use. The teaching limit is deliberately smaller than the source pack's

production target. Measure both serialization cost and request-view size before increasing it.

Exercise

Stop the delivery process after applying an enforced decision. Advance the test clock to its expiry. Then try a wall-clock correction backward while leaving the monotonic clock unchanged. What are the distinct expected outcomes?

Answer

At ordinary expiry the address is allowed. After a large clock divergence, the lab reports unavailable because time health is uncertain. The deployment's Nginx availability profile decides how an unavailable guard affects the public request. Expired policy and unhealthy time should have separate metrics so operators do not mistake a clock incident for a normal ban ending.

14. Give operators precise controls

An automatic system still needs intervention

An operator must be able to inspect evidence, change a site mode, revoke a decision, protect known infrastructure through an allowlist, and activate a local recovery mechanism. Those are different operations with different authority. A button labelled "unblock" can be ambiguous if it merely edits desired state while an edge remains disconnected.

Use status language that tells the operator what is known. A revocation can be committed centrally, queued for delivery, acknowledged by one edge, and pending on another. The central audit should record the intention. The applied-revision record should show the effect. Do not overwrite one with the other.

Practice revocation in the lab

Create the lesson 4 fixture, switch to enforce mode, and inspect the decision:

```
python3 -m logbranik decisions
```

Copy its actual ID into the revoke command:

```
python3 -m logbranik revoke ACTUAL_DECISION_ID
```

The token above denotes the ID you just inspected; it is not a literal example ID that the program knows. With the delivery loop running, a new full policy omits that decision. Check the local handler again and confirm allow. If the control connection is offline, central revocation alone does not remove already applied state. Local expiry remains the maximum lifetime boundary.

The lab does not provide a full operator HTTP API or a fleet dashboard. Its CLI is an intentionally small administration surface for one demonstration site. The production pack specifies authenticated roles, bounded queries, idempotency keys, audit records, and desired/applied status separation.

Manual decisions

A production manual block needs an exact target, an explicit site or global scope, a reason, an operator identity, and a mandatory TTL. A global scope must be explicit rather than inferred from a missing site. Automatic network-range decisions are excluded from the baseline design because their blast radius is larger than an exact-address decision.

Make manual requests idempotent. A CLI command can time out after the central transaction commits. Retrying with the same idempotency key should return the original effect rather than creating another ban with a new lifetime. A repeated command and a deliberate new intervention are different actions.

Mode and allowlist changes benefit from optimistic concurrency. An operator should know if another person changed the configuration between reading it and saving it. Silent last-writer-wins behavior can remove a newly added exemption or re-enable enforcement after a maintenance decision.

Keep emergency controls local

A remote signed policy must not be able to remove the edge's independent emergency address list or bypass setting. Central may control ordinary policy, but recovery authority should remain available when central credentials or logic are the source of the incident.

Local bypass needs visible state. Include it in health responses and heartbeat reports, display it prominently in the operator interface, and alert if it remains active beyond the intended maintenance period. A recovery mechanism that becomes a forgotten permanent mode defeats the reason for the guard.

Evidence retention

An explanation may outlive raw event retention. Store bounded evidence references and a concise detection summary when making a decision. Otherwise an operator can see a ban but find that its explanation disappeared during cleanup. Retain only what the investigation and policy require; keeping every raw observation indefinitely creates cost and access risks.

The teaching decision table has a reason code and timestamps. Its evidence is found through the associated site/address/window observations. The production schema should attach explicit evidence identities or summaries so that historical explanations remain stable after unrelated database maintenance.

Exercise

Design the response to a manual revoke request for an address enforced on two edges, one of which is offline. Which states should the operator see immediately and later?

Answer

Immediately, the central decision is revoked and delivery is pending for both relevant edges. Later, the connected edge reports the new revision and the revocation becomes applied there. The offline edge remains pending until reconciliation or the original local expiry. The response must not claim fleet-wide enforcement has already changed merely because the central transaction succeeded.

15. Observe the guard as an operating system

Metrics should answer an action question

A count of requests is useful context. It does not tell you whether a desired ban reached the edge. An agent health status does not tell you whether log observations are current. Start each metric with the question an operator needs to answer.

For collection, ask whether observations are arriving and how old the oldest pending event is. For ingestion, ask whether commits succeed, duplicates rise unexpectedly, or validation conflicts appear. For detection, ask how many events are eligible and which rule is creating decisions. For delivery, ask how long desired state has remained unapplied. For the agent, ask which revision is active, whether time is healthy, and whether emergency bypass is enabled.

Avoid high-cardinality labels such as every client IP or request ID in metrics. Those belong in bounded event queries or logs. Metrics describe rates, capacity, and state classes. Unbounded labels can turn a defensive service into its own memory problem.

Separate latency distributions

Measure request-check latency at the edge, not only inside the authorization function. Nginx pays for socket transport, scheduling, and response handling as well as a map lookup. Use a baseline with the guard disabled and compare an enabled run under the same workload and host conditions.

Measure completion-to-ingest delay and decision-to-applied delay separately. A detector may finish quickly while the collector batches for several seconds. A delivery worker may transmit quickly while an overloaded agent takes longer to persist policy. A single average hides both bottlenecks and tail behavior.

The source pack proposes initial targets of decision-to-applied p95 at most five seconds at 100 events per second and added authorization p95 at most two milliseconds at 1,000 requests per second. These are

provisional acceptance targets. This edition has not measured them and does not use them as marketing claims.

Capacity has several dimensions

Events per second, active decisions, assigned sites, edges per fleet, snapshot size, and retention length scale different components. A system handling many requests with few bans can have a cheap request view and a large history store. A fleet with many active bans can have low event traffic but expensive snapshot fanout.

Full snapshots simplify convergence. They also serialize the entire desired state for an edge. Coalesce rapid changes, cap bytes and entries, and measure serialization and transfer cost. Do not introduce a delta protocol until its ordering, missing-delta recovery, and compaction contracts are clear.

The lab's periodic push demonstrates safe replacement but creates a new revision and outbox record repeatedly. It is a teaching process, not a storage-efficient production dispatcher. Its database will grow during a long session. Stop it when the exercise is complete. The production retention and outbox cleanup tasks are part of the sixty-step pack.

Retention is operational work

Set a short raw-event retention aligned with troubleshooting needs. Index the queries you actually run. Observe database disk use, cleanup duration, and how deletion affects active processing. Retaining history and retaining enough explanation for a decision are related but separate policies.

Do not put credentials in debug logs while diagnosing TLS failures. Report a bounded identity or fingerprint when appropriate, without serializing private keys or entire certificates into routine logging. Treat operator audit records as privileged evidence and scope access accordingly.

Exercise

An operator sees no new bans, low CPU, and a healthy agent. Collection lag is twenty minutes and the future-event counter is zero. What should they investigate first, and why would increasing detector sensitivity be a poor response?

Answer

Investigate the collector, transport, and ingestion backlog first. The analyzer's knowledge is stale even though the local agent is healthy. Increasing sensitivity does not make missing fresh observations arrive and may amplify false positives when the backlog drains. The freshness policy should keep those old observations from creating fresh bans.

16. Move durable processing to PostgreSQL

Preserve the contract when changing storage

SQLite is useful for an inspectable single-node laboratory.

PostgreSQL becomes appropriate when ingestion, detector workers, delivery workers, and operators need controlled concurrent access. A database migration should preserve behavior before it improves throughput.

Keep event uniqueness on authenticated node and request ID. Preserve immutable event digests and original decision expiries. Add durable jobs and delivery intent as transactional records. Test the same retry and replay scenarios against the new storage implementation. If an optimization removes one of those invariants, it is a behavior change rather than a transparent migration.

The delivered `companion/sql/production-schema.sql` is a teaching schema for events, detector jobs, decisions, and delivery work. It is not a tested migration for an existing database. Production migrations need versioning, rollback or forward-repair strategy, representative data, and execution against the actual PostgreSQL version.

Claim a job, then commit its effect

Several workers can claim available rows using row-level locking with `SKIP LOCKED`. This is useful for a queue-like workload, but it does not automatically serialize every shared counter those jobs will update. Two different events for the same site and address may be claimed by different workers.

A transaction outline is:

```
BEGIN;  
SELECT id, node_id, request_id  
FROM detector_jobs  
WHERE completed_at IS NULL  
ORDER BY id  
LIMIT 1  
FOR UPDATE SKIP LOCKED;  
  
-- Lock or atomically update the shared detector state.  
-- Write a detection and decision when appropriate.  
-- Write delivery intent in the same transaction.  
-- Mark the claimed job completed.  
COMMIT;
```

The comments identify required application work; this listing is not a complete worker. Hold the claim lock until the effects and completion commit together. A worker crash before commit leaves the job eligible. A crash after commit leaves its effects recorded once.

Serialize the detector's shared key

The relevant shared state is usually site, address, rule version, and time window. You can use a dedicated row lock, a carefully defined advisory lock, or an atomic SQL design. The choice must handle concurrent events for the same key and independent events for different keys.

An in-process mutex does not coordinate separate worker processes. A queue row lock coordinates that row, not every event from the same visitor. Process-local counters make behavior dependent on which worker happens to receive an event. Start with one detector worker, demonstrate correctness, then add concurrency with a race test designed to expose lost or repeated increments.

Window processing also needs a policy for out-of-order events. A simple fixed bucket boundary can miss a burst split across two buckets. A rolling window can require more retained state. The lab uses a direct query because the scale is small. Production should choose a window model deliberately and preserve freshness exclusion before counting.

Outbox as a transaction boundary

Create desired decisions and delivery intent together. Otherwise a crash can leave a committed ban with no durable instruction to publish it. A delivery worker can later derive a full per-agent snapshot, allocate a revision, sign it, store its bytes, and retry those bytes while fresh.

Signing can involve external secret infrastructure. Do not hold a high-contention detector lock while waiting indefinitely for it. Define how unsigned intent becomes signed, persisted delivery work and how a crash at that boundary is repaired. Revision allocation and exact stored bytes still need a coherent transaction design.

Notifications are a latency optimization. A worker must also scan durable pending work after restart or a missed notification. Relying on a transient message as the only work record recreates the ingestion gap that the database was meant to close.

Exercise

Two workers claim different event rows from the same site and address. Each reads a shared counter value of four and each writes five. What is lost, and which lock did not solve the problem?

Answer

One increment is lost, and both workers may make an inconsistent threshold decision. Locking each queue row did not serialize the shared detector key. Use a shared-state lock or a correct atomic update, and commit the decision and job completion with the counter effect. A real PostgreSQL concurrency test is required for this migration.

17. Preserve the fast path in a Go agent

Why separate the production implementation

The Python agent demonstrates semantics and is easy to inspect. A production edge service has additional responsibilities: bounded concurrency, lifecycle supervision, metrics, platform-specific socket ownership, key rotation, independent reconciliation, and predictable request latency. The source architecture chooses Go for that service, while retaining the same policy invariants.

This lesson is a porting design. The package does not contain a completed Go agent or a measured claim about its performance. The original sixty-task pack breaks implementation into verified milestones. Treat the Python tests as behavioral fixtures to reproduce in Go, not as evidence that an unwritten port works.

Publish an immutable view

Represent the request-time state as one object containing modes, normalized allowlist keys, normalized active decision keys, revision identity, and expiry deadlines. Construct a new object during an update, then publish one pointer. Do not mutate a map that active request goroutines are reading.

A conceptual shape is:

```
type View struct {
    Revision uint64
    // Maps are private and immutable after publication.
    Modes map[string]string
    Bans  map[Key]Decision
}

var current atomic.Pointer[View]
```

This excerpt is explanatory, not a compilable service: Key, Decision, imports, validation, and lifecycle are deliberately omitted. A pointer publication primitive does not make the maps immutable. That guarantee comes from ownership and code discipline. Keep update

construction separate from the check path and use race tests on the finished implementation.

Normalize address keys

Use an address type rather than arbitrary text. Apply the same IPv4-mapped IPv6 rule as the collector and central detector. Reject zone identifiers and unknown sites. Build composite keys from site and address; a missing site must not become global scope.

For each request, read the published view once, apply local bypass and allowlist precedence, test active deadlines, and return the documented status and outcome. No database access, disk read, central call, or signature verification belongs in that path.

Serialize updates

One update coordinator validates the envelope, checks ordering, persists state, and publishes the view. If several control requests arrive simultaneously, the coordinator serializes them so that revision comparisons and publication remain coherent. Expensive work can occur before the final lock when safe, but recheck ordering at the commit boundary.

Bound the envelope body before allocating it. Bound policy entries before building indexes. Reject duplicate decisions and revision conflicts. Keep errors machine-readable enough for a dispatcher to distinguish stale data, invalid content, unauthorized identity, and transient storage failure.

Supervise the process

Run the agent as an unprivileged service account. Provision a runtime directory with deliberate Nginx access to the authorization socket. Place durable state in a separate writable state directory. Apply systemd hardening only after verifying that it permits the required filesystem and network operations.

Do not have a remote policy execute `nginx -s reload` or edit arbitrary configuration. Initial deployment changes are operator-controlled. Runtime decisions update the local view. Service restart and policy update are different events and should have distinct operational procedures.

Reconciliation and health

Add an outbound authenticated pull loop independent of push. The loop reports or uses the applied revision and receives only the agent's authorized snapshot. Apply the same validator and persistence sequence. Heartbeats include revision, version, time health, capacity, and emergency bypass state.

Keep request readiness distinct from process liveness. A process can be listening while it has no verified policy. A healthy local agent can have old but still valid state during a central outage. A new empty policy can establish readiness without any bans. These distinctions should survive the language migration.

Exercise

A developer stores a pointer to a new view, then updates its ban map in place to remove expired entries. Why is this unsafe, and what alternatives preserve the model?

Answer

Readers may race with map mutation or observe a partial change. Build and publish a new immutable view, or keep expiry evaluation in each check and prune by publishing another complete view later. The finished Go port needs a race-detector run and concurrent-check/update tests, plus the signature, replay, restart, and TTL fixtures from the lab.

18. Test behavior at the boundary where it matters

A useful test can falsify a claim

A test that repeats an implementation's constants is weak evidence. A test that retries the same event and observes no additional decision checks a real invariant. A test that injects a persistence failure and observes the old published state checks the acknowledgement boundary. Design tests around observable outcomes and failure mechanisms.

The companion's core tests cover duplicate and conflicting events, stale and future observations, guard-feedback exclusion, exact IPv4 and IPv6 behavior, monitor/enforce/revoke transitions, allowlist precedence, TTL expiry, refresh without extension, invalid signatures, epoch mismatch, stale revisions, equal-revision conflicts, malformed JSON, excessive input, restart, and concurrent checks.

Live loopback tests cover authenticated HTTPS push, acknowledgement identity, certificate-derived ingestion identity, missing client certificates, wrong-role certificates, and authorization routes that cannot mutate policy. The Unix-socket route test is explicitly skipped in the preparation environment because `AF_UNIX` creation is prohibited there. Run it without that skip on a Linux deployment host.

Keep evidence levels separate

Evidence	What it establishes	What it does not establish
Core policy test	Function and state semantics	Nginx route coverage
Live mTLS test	Transport and role boundary	Production certificate lifecycle
Unix socket test	Local handler transport	Filesystem policy for all host users
Nginx integration test	Access-phase behavior	Fleet delivery reliability
Collector rotation test	Tested file/buffer topology	Unlimited outage tolerance
Load measurement	Recorded host/workload performance	Performance on every deployment

One test can be valuable without proving the entire system. The mistake is attaching a larger claim to it than the observed behavior supports.

Build the Nginx acceptance matrix

Test ordinary requests, special routes, named locations, redirects, static resources, existing authentication, and satisfy combinations. Confirm that a denied request never reaches the protected upstream. Confirm that permitted POST requests preserve their bodies. Confirm that application 403 and 500 responses are not confused with guard outcomes.

Kill the local agent and observe the public result under the chosen availability profile. Kill central services while leaving the agent live and verify local expiry. Spoof forwarding and guard headers from an untrusted client. Rotate logs while traffic continues. Restart the collector with pending records and compare unique generated request IDs.

These are deployment tests, not extra unit tests of a configuration string. The original pack's acceptance documents define the milestones and evidence expected from the real system.

Test recovery, not only failure

After an outage, restore transport and check that stale log replay does not create fresh bans. Reconcile a newer policy and verify that old pending deliveries cannot restore a revoked decision. Restart an agent from saved state after envelope delivery expiry while a decision remains live. Restore central storage behind the edge's high-water mark and observe rejection instead of silent rollback.

Recovery paths often exercise a different state sequence from initial startup. A successful cold demo does not cover them. Use deterministic fixtures where possible so the test can identify whether a failure concerns time, ordering, or data identity.

Record the environment

A useful verification record includes interpreter or compiler version, dependency version, operating system, configuration digest, workload, command, result, skipped cases, and limitations. For a performance claim, add hardware, concurrency, duration, distributions, and baseline conditions. Do not substitute a target number for measured evidence.

The delivered `verification/python-tests.txt` is an actual command log. The release report identifies unexecuted services. Updating the manuscript should regenerate its evidence after meaningful code changes rather than preserving an old green result beside new source.

Exercise

A pull request changes only the fail-open Nginx profile. Existing Python policy tests all pass. Is the change sufficiently verified?

Answer

No. The concrete remaining risk is Nginx's internal error routing and preservation of the main request. Run actual Nginx tests for backend availability errors, real 403 denial, application errors, and POST method/body behavior under that profile. Broader unrelated tests would not resolve the missing evidence.

19. Diagnose failures in the right order

Start with the visible outcome

A visitor receives 403. First determine whether Nginx's guard denied the request or the protected application returned it. Use the event's guard outcome and an upstream correlation identifier. If the outcome is block, inspect the local revision and matching decision. If the outcome is allow, investigate application authorization rather than deleting guard state.

A visitor receives an error while the agent is unavailable. Check local process state, socket existence and permissions, first-policy readiness, and time health. Do not begin by tuning detector thresholds. The detector's intent cannot fix a missing request-time dependency.

Follow an absent ban through the pipeline

If an expected block does not happen, ask a sequence of concrete questions. Did the request finish and produce an access event? Did the collector read the correct file? Did normalization retain the correct site and address? Did ingestion commit it? Was it fresh and eligible? Did the rule threshold actually meet distinct-path criteria? Did central create a decision? Did an edge apply the corresponding revision? Was the site in enforce mode? Was an allowlist or bypass taking precedence?

This sequence separates missing observation from missing delivery and from correct non-enforcement. The answer may simply be that the site is in monitor mode. It may be that a shared address has a deliberate exemption. Removing safeguards to make a demo "work" obscures the actual cause.

Read errors as protocol information

A signature failure suggests altered bytes, the wrong signing key, an incorrect domain, or a bad encoding. An epoch failure suggests mismatched provisioning. A stale revision after database restore suggests a high-water recovery problem. A same-revision conflict suggests a sender reused a revision for different content.

These are different incidents. Retrying an epoch mismatch forever is not useful. Disabling verification to accept a policy is not a repair. The lab delivery process stops on non-retryable control errors and asks for inspection; a production dispatcher should surface the same distinction in delivery state and alerts.

Safe local recovery

If the guard is harming availability, activate the documented local bypass through the controlled administrative path. Record why it was activated and make its state visible. Preserve evidence before changing central policy. Revoke harmful decisions or correct mode and allowlist configuration, then reconcile and confirm local application before removing bypass.

Do not delete the state file as a routine "unblock" command. That loses revision history and can leave a new agent not-ready. Do not reset the epoch without a controlled provisioning action. Do not edit the stored signed payload in place: it invalidates the signature and destroys the evidence of what was applied.

Diagnose collector backlog

When central ingestion recovers, watch buffer occupancy, oldest-event age, accepted-event rate, duplicates, and stale-event exclusions. A rapidly falling buffer is not automatically healthy if valid records are being permanently dropped. A burst of duplicates may be normal recovery from lost acknowledgements. A burst of conflicts is not.

Check rotation and disk capacity before changing retry settings. If unread logs have already been removed, retrying transport cannot reconstruct them. Use the unique request-ID reconciliation from the collection lesson to distinguish duplicate delivery from actual observation loss.

A concise incident record

Record the affected site, visible outcome, local revision, desired revision, clock health, mode, bypass state, relevant decision expiry, transport condition, and changes made. This small set of facts makes the next investigation faster without dumping raw secrets into the incident channel.

The appendix runbook gives command-oriented steps for the laboratory. A production runbook should add service names, ownership, escalation, backup locations, certificate rotation, and a tested recovery path for your host distribution. Keep it versioned with the deployment artifacts.

Exercise

Central shows a revoked decision, but one edge still returns 403. List three plausible explanations and the evidence that distinguishes them.

Answer

The edge may not have applied the newer policy; compare desired and local revision. A different active decision may match the same address; inspect the full desired and local explanation. The application may be returning 403 while the guard allows; inspect the guard outcome and upstream trace. Each explanation leads to a different action, so a generic "clear cache" step is inappropriate.

20. Release a reproducible system

Freeze a reviewable edition

A release joins code, configuration, protocol version, tests, deployment notes, and operator procedures. Tagging code alone leaves readers uncertain about which manuscript and fixtures describe it. The book's release tooling exports lesson sources, checks structure, builds reading formats, records evidence, and creates a checksum manifest.

Each lesson is maintained in a Markdown file. Leanpub manuscript text is generated from those files, so there is one editorial source. The sample selects the preface and first four lessons. The full PDF and EPUB are built locally for review. Leanpub can generate its own preview when the manuscript repository is connected; that platform-specific output still needs inspection.

Avoid the production-readiness shortcut

The teaching implementation is useful because it executes the central invariants. It does not implement every production requirement. It has one demonstration site, a compact integer-time wire format, a small rule, a simple periodic push process, and a SQLite store. It does not include a fleet operator API, independent pull reconciliation, mature retention, a complete Go port, or production load measurements.

Those are concrete next milestones, not vague warnings. The source pack provides sixty implementation tasks with acceptance evidence. Map each one to a lesson and complete it against your actual repository. A task is done when its criteria have evidence, not when a coding assistant emits a plausible patch.

Assistants as implementation tools

The included Claude Code pack has native project instructions and progress tracking. Its tasks specify intended behavior and dependencies. Keep the application repository's existing instructions, review changes, and record commands and results. An assistant-generated summary should distinguish code written, tests executed, and deployment gates pending.

Model effort can help with complex reasoning, but it cannot replace real service execution. Use stronger reasoning for protocol ordering, threat boundaries, and concurrency review when available. Use ordinary effort for editorial cleanup and mechanical exports. The final evidence comes from the code, the environment, and human judgment about the observed behavior.

Publish responsibly

Before commercial book publication, identify the actual responsible author or pen name, perform a human technical and editorial review, reproduce unexecuted deployment exercises, inspect PDF and EPUB, and complete the platform's author and payout setup. The package intentionally contains no invented biography or legal identity.

The Leanpub AI policy expects quality and human curation rather than mass-produced low-quality material. The manuscript's transparency statement describes the current preparation process. Update it honestly after review. Do not claim AI was used only for copyediting when it generated substantial text and code.

Pricing in the publishing notes is an editorial starting suggestion. It is not a revenue forecast, a market benchmark, or a platform fee quotation. Check the current account terms before selecting formats, payment details, and publication settings.

Improve through evidence

The most valuable next changes are those driven by a observed limit. If rotation loses events, repair collection before adding more detectors. If false positives affect shared addresses, improve calibration and scope before lowering thresholds. If snapshots become too large, measure coalescing and fanout before creating a delta protocol. If concurrent workers miscount, fix state serialization before increasing worker count.

Keep release notes precise. Say that an epoch-recovery procedure was added and tested, or that a particular Nginx profile preserved POST bodies under specified errors. Avoid blanket statements such as "security improved" when the underlying behavior can be named.

Exercise

Choose the next three implementation-pack tasks after finishing the laboratory. For each, name the remaining risk, the expected observable behavior, and the evidence required to mark it complete.

Answer

A useful set is actual Nginx route integration, continuous collector recovery, and production worker transaction correctness. Their risks are bypassed access phases, lost or stale observations, and repeated or lost detection effects. Their evidence is a route/status/body matrix, unique request-ID reconciliation across outage and rotation, and a concurrent PostgreSQL crash/retry test. Completing those three would support stronger deployment claims than adding a cosmetic dashboard first.

Appendix A. Laboratory runbook

Files and responsibilities

File	Purpose
logbranic/protocol.py	Strict parsing, signatures, policy validation, agent state
logbranic/central.py	SQLite events, deduplication, small detector, revisions
logbranic/servers.py	Local checks, mTLS endpoints, role authorization, push
logbranic/certificates.py	Disposable loopback TLS fixtures
logbranic/__main__.py	Laboratory CLI and function demo
delivery_loop.py	Periodic signed push for the teaching exercise
normalize_logs.py	Finite Nginx-log conversion into lab NDJSON
render_nginx_config.py	Isolated Nginx lab configuration generation
echo.py	Local application used for real access-phase testing
tests/test_guard.py	Behavioral and live transport tests

Run commands from the companion directory unless a command says otherwise. No command in this runbook installs configuration into `/etc/nginx`. The generated Nginx configuration listens only on loopback and uses a directory you control.

Quick function demonstration

```
python3 -m pip install -r requirements.txt
python3 -m unittest discover -s tests -v
python3 -m logbranic demo
```

For a repeatable dependency baseline, the preparation run used Cryptography 50.0.1. The requirements range permits compatible releases; upgrading within it still requires rerunning the tests.

Process laboratory

```
python3 -m logbranik init
python3 -m logbranik certificates
```

Run agent, central server, delivery loop, and echo application in separate terminals:

```
python3 -m logbranik agent
python3 -m logbranik central-server
python3 delivery_loop.py
python3 echo.py
```

These lines are four separate long-running processes, not a sequence to paste into one blocked terminal. Use the same initialized directory for this single-host lab. Disposable credentials expire after seven days. Regenerate fixtures in a new isolated directory when needed; do not disable TLS verification to work around expiration.

Nginx host acceptance

On a Linux host with the required Nginx module, generate the isolated configuration:

```
python3 render_nginx_config.py
nginx -t -c "$PWD/lab-state/nginx.conf" -p "$PWD/lab-state/"
nginx -c "$PWD/lab-state/nginx.conf" -p "$PWD/lab-state/"
```

The echo application must already be running. The agent must have a valid applied policy before public requests can be allowed. The delivery process supplies the initial signed empty policy.

If you chose the loopback TCP teaching option for the agent, generate the matching profile with `python3 render_nginx_config.py --tcp-auth`. Do not mix a TCP agent with a Unix proxy target. Both variants remain lab-only until deployment acceptance is recorded.

Create five actual local requests:

```
for i in 1 2 3 4 5; do
  curl -s -o /dev/null http://127.0.0.1:8080/missing-$i
done
python3 normalize_logs.py lab-state/guard-access.json >
  events.ndjson
```

Submit the finite capture with the collector credential:

```
curl --cacert lab-state/tls/ca.crt \  
  --cert lab-state/tls/collector.crt \  
  --key lab-state/tls/collector.key \  
  -H 'Content-Type: application/x-ndjson' \  
  --data-binary @events.ndjson \  
  https://localhost:9444/v1/events:batch  
python3 -m logbranik decisions  
python3 -m logbranik mode enforce
```

Wait for a newly acknowledged delivery revision, then request another path. The local requests originate from 127.0.0.1, so that is the address the lab rule will match. The fixture address from lesson 4 is a separate demonstration and does not match a real loopback curl request.

This finite-capture exercise closes the full path once. Continuous real-time tailing, rotation, and replay use the production collector design and require their own acceptance test. Do not claim that manually normalizing a file is a streaming collector.

Stop and preserve evidence

Stop the isolated Nginx instance using its generated configuration:

```
nginx -s quit -c "$PWD/lab-state/nginx.conf" -p "$PWD/lab-state/"
```

Interrupt the lab processes. Save the command output and relevant request IDs. The agent cleans its socket during orderly shutdown; a hard-killed process can leave a stale socket path. Verify that the process is absent and that the path is the lab socket before removing only that socket. The program refuses to overwrite an existing socket automatically.

Never archive a populated lab-state directory into a public code package. It contains private keys and may contain access observations. The distributed package contains generation code and empty configuration examples only.

Deployment evidence still required

Record the actual Nginx version and configure options, route coverage, body preservation, blocked-upstream exclusion, unavailable-agent profile, proxy trust behavior, Unix ownership, collector rotation, and end-to-end latency. The delivered preparation evidence does not substitute for this run on your server.

Appendix B. From lessons to sixty implementation tasks

The included Claude Code pack preserves all sixty original implementation tasks. The table below maps each task to its primary explanatory lesson. A lesson explains the design; it does not mark the production task complete. The original tracker remains pending and requires actual acceptance evidence.

Extract companion/nginx-guard-claude-code-pack.zip, read its README_SR.md and bootstrap instructions, and install it into your application repository according to the pack. Keep that repository separate from the book manuscript.

ID	Implementation task	Lesson
000	Repository inventory and project bootstrap	20
001	Version selection and reproducible dependencies	20
002	Typed contracts and golden fixtures	11
003	Local networks and development certificates	4
004	Database schema and initial migrations	16
005	Build commands and test foundation	18
006	Authenticated central service and role boundaries	2
007	Bounded NDJSON ingestion	7
008	Deduplication and durable processing queue	7
009	Structured Nginx access logging	5
010	Vector transport and disk buffering	6
011	First complete log ingestion demonstration	4
012	Go agent process and local listener	17
013	Immutable policy state and atomic persistence	12

ID	Implementation task	Lesson
014	Local authorization decision engine	13
015	Nginx auth_request integration	8
016	Availability profiles and local emergency bypass	3
017	Local expiry and offline enforcement milestone	13
018	Ed25519 signatures and cross-language vectors	11
019	Private mTLS snapshot receiver	11
020	Desired decisions and transactional snapshot outbox	12
021	Reliable push delivery and applied acknowledgements	12
022	Pull reconciliation and node heartbeat	12
023	Manual block lifecycle end-to-end milestone	14
024	Event-time aggregation and concurrency correctness	16
025	Sensitive-path scanning rules	10
026	Login-abuse rule with explicit endpoint semantics	10
027	Request-volume and error anomaly monitoring	10
028	Risk policy and safe automatic decision creation	10
029	Automatic detection-to-block demonstration	4
030	Operator roles and complete audit trail	14
031	Operator CLI for investigation	14
032	Operator CLI for bounded policy changes	14
033	Allowlist precedence and revocation race handling	14
034	Evidence search and decision explanations	14
035	False-positive recovery and emergency drills	19
036	Fleet membership and multiple sites per node	2

ID	Implementation task	Lesson
037	Snapshot coalescing and bounded delivery scaling	15
038	Raw-event retention and database maintenance	15
039	Metrics and structured cross-service diagnostics	15
040	Alert rules and operational health views	15
041	Measured throughput and latency baseline	15
042	Central outages, backpressure and collector recovery	6
043	Clock, TTL and restart fault matrix	13
044	TLS and signing-key lifecycle	11
045	Hostile inputs and cross-scope security tests	18
046	Nginx routing and existing-auth regression suite	8
047	Complete reliability regression gate	18
048	Production images and agent release binaries	20
049	Hardened systemd units and filesystem ownership	17
050	Idempotent installation and canary configuration	20
051	Database backup and tested restore	12
052	Upgrade, rollback and compatibility procedures	20
053	Deployment observability and recovery handbook	19
054	Continuous integration and release policy	20
055	Replay evaluation and production threshold calibration	10
056	Monitor-only installation readiness	10
057	Single-site enforcement canary	20
058	Staged fleet enablement and rollback drill	20
059	Final audit and release handoff	20

Milestone order

Start with repository and protocol foundations. Verify collection before detection. Verify local authorization before remote control. Verify signed control and expiry before automatic enforcement. Add operator recovery, then run reliability and deployment gates before enabling a canary. Fleet rollout comes after a single-site measured demonstration.

The compact Python lab covers behavior across several milestones using a smaller stack. It does not import its test result into the original production tracker. PostgreSQL worker correctness, Go race safety, continuous collector guarantees, real Nginx routing and staged deployment require their own evidence.

Resume safely

Use the pack tracker and handoff document to resume work. Preserve unfinished prerequisites. Do not mark a task done because an assistant summarized it as successful. Record the executed command, observed result, and relevant artifact. When a prerequisite changes, reopen dependent work according to the tracker rules.

Appendix C. Glossary and sources

Glossary

Access event: A terminal observation of one completed Nginx request. It is not a pre-request inspection.

Agent: A local service that validates remote policies and answers authorization checks.

Applied state: The complete policy view an agent has persisted and published.

Backpressure: A condition in which downstream capacity restricts upstream progress. Its effect depends on the whole collection topology.

Decision: A bounded data record describing an address, scope, reason, and expiry.

Delivery interval: The period during which a new envelope can be applied. It is separate from each contained decision's lifetime.

Desired state: The central configuration and decisions the fleet is intended to apply.

Epoch: A provisioned identifier for one agent policy stream. Changing it requires a controlled recovery action.

Event time: The time the observed request completed.

Idempotency: Repeating the same operation identity does not create additional effects.

Monitor mode: A mode that reports would-block while permitting requests.

mTLS: TLS with authenticated client and server credentials. Application authorization still determines permitted roles and scopes.

NDJSON: Newline-delimited JSON objects, distinct from a JSON array.

Outbox: Durable delivery intent or bytes stored with the state transition that requires them.

Policy snapshot: A complete replacement of an agent's remote policy state.

Processing time: The time an observation is handled by the analyzer.

Revision: A monotonically increasing integer within one provisioned policy stream.

TTL: A decision's mandatory time to live, bounded independently by the agent.

Primary technical references

The book's architecture is an original project design developed from the companion implementation pack. The following official documentation supports the external mechanisms used. Consult the documentation for your installed versions before adapting examples. References were checked in October 2026.

1. Nginx authorization-subrequest module:
nginx.org/en/docs/http/ngx_http_auth_request_module.html.
Module availability, authorization statuses, and directive semantics.
2. Nginx real-IP module:
nginx.org/en/docs/http/ngx_http_realip_module.html. Trusted peer configuration and resolved client addresses.
3. Nginx HTTP log module:
nginx.org/en/docs/http/ngx_http_log_module.html. JSON escaping, log contexts, and buffering.
4. Vector HTTP sink:
vector.dev/docs/reference/configuration/sinks/http/. Transport configuration and sink behavior.
5. Vector end-to-end acknowledgements:
vector.dev/docs/architecture/end-to-end-acknowledgements/.
Topology-dependent acknowledgement guarantees.

6. PostgreSQL SELECT and locking:
[postgresql.org/docs/current/sql-select.html](https://www.postgresql.org/docs/current/sql-select.html). Queue-like uses of row locking and SKIP LOCKED.
7. Cryptography Ed25519 interface:
cryptography.io/en/latest/hazmat/primitives/asymmetric/ed25519/. Library signing and verification interfaces.

Publishing references

8. Leanpub Git/GitHub writing mode:
help.leanpub.com/en/articles/2916385. Manuscript workflow and preview.
9. Leanpub manuscript ordering:
help.leanpub.com/en/articles/15408698. Book.txt ordering.
10. Leanpub AI quality policy:
help.leanpub.com/en/articles/11853585. Human curation and content quality.
11. Leanpub AI-use settings: help.leanpub.com/en/articles/9363893. Platform disclosure controls; retain the book's own transparency statement.
12. Leanpub terms: leanpub.com/terms. Current account and content terms.

How to use these sources

The references are mechanisms, not endorsements of LogBranik. The book does not copy a vendor manual or claim that a third-party feature proves this complete system works. Its policy ordering, role map, data contract, and operational boundaries are project decisions, supported by the tests recorded with this edition and by the deployment gates readers must execute.

If a reference changes, check whether the change affects the installed mechanism before revising the book. A new Nginx release does not automatically change an existing host. A different collector acknowledgement capability can change the end-to-end guarantee even when the ingestion code remains identical.