Tomelet

WHITE PAPER



Engineering and operations

HOW WE BUILD AND RUN TOMELET

Overview

Tomelet ships a reader on six platforms and an encrypted sync service, and both change every week. Each change is an opportunity to improve the product and, if handled carelessly, a way for a mistake to reach readers.

Keeping a fast pace while protecting readers' privacy depends on making security part of how every change is made, reviewed, released and operated. Tomelet's engineering practice rests on five foundations:

A written engineering standard

Every change follows a written engineering standard, and the responsible teams approve scope, releases, spending and credentials.

Independent review

Every code change we make is reviewed independently of its author, with the deepest review reserved for encryption, sync, sign-in and payments.

Automated guardrails

The rules that matter most are checks that fail the build, so they apply to every change automatically.

A careful supply chain

Outside code is admitted only after a waiting period, pinned to an exact version and recorded with evidence.

Least-privilege operations

Separate environments, narrow permissions, short-lived credentials and logs that expire limit what any single mistake can reach.

This white paper sets out how each of these works in practice. The encryption that keeps a reader's library private is covered in the companion Tomelet Sync security white paper. Where this paper and the Privacy Policy or Terms differ, those documents govern.

Contents

Overview 2

How Tomelet is built 4

Challenges of building a private product quickly 5

Engineering done right 5

Our design tenets 6

How a change reaches readers 7

Automated guardrails 8

Testing and regressions 8

Securing the supply chain 9

Where the service runs 10

Protecting data and the service 11

Access to production 12

Logging and retention 13

Monitoring and recovery 14

Deleting your data 15

Requests from public authorities 16

Responding to vulnerabilities 17

Conclusion 18

How Tomelet is built

Tomelet is made by TOMELET PTE. LTD., based in Singapore. Every change to Tomelet follows a written engineering standard kept in the source repository.

The standard sets out how bugs are fixed, how quality is maintained and how dependencies are kept current, and it reserves every decision about new features, spending, production releases, credentials, privacy and legal wording for the teams responsible for them. Because the standard is written down and enforced by automated checks, the same safeguards apply to every change, every day.

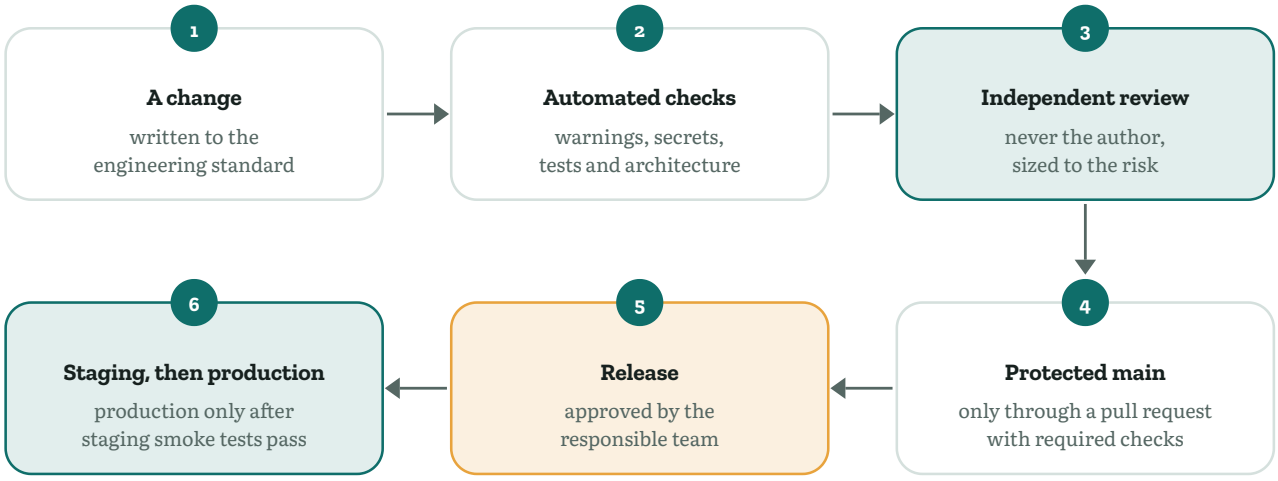


Figure 1. *Every code change we make passes the same gates on its way to readers. The sync service's release pipeline runs stage 6 automatically.*

Challenges of building a private product quickly

1. MOVING FAST WITHOUT LOSING CONTROL

A product on six platforms changes constantly. Each change has to be reviewed with care proportionate to what it touches, without slowing every change to the pace of the most sensitive one.

2. TRUSTING OUTSIDE CODE

Modern software depends on hundreds of open-source packages and build tools. A newly published or compromised package can reach an app through an ordinary update, so outside code needs to be admitted deliberately.

3. KEEPING PRODUCTION NARROW

The people and tools that build a service also need to diagnose it. Access has to be enough to fix problems and no more, and every credential has to be as short-lived as possible.

4. LEARNING FROM EVERY BUG

A bug a reader reports is a gift of information. The challenge is to turn each one into a permanent safeguard, so the same problem, and others like it, cannot return.

> **ENGINEERING DONE RIGHT**
>
> These challenges shape how Tomelet is built and run. Our goal is a product that improves every week while every change meets the same standard of care for readers' privacy. This white paper sets out the practices that make that possible, so that readers, researchers and partners can see how Tomelet is made.

Our design tenets

Four tenets guide every engineering and operations decision at Tomelet.

HOLD AS LITTLE AS POSSIBLE

The free reader needs no account, and Tomelet Sync stores what a reader's devices have already encrypted. The less we hold, the less there is to protect.

MAKE THE SAFE PATH THE ONLY PATH

A rule that matters is enforced by a check that fails the build, so following it is automatic.

LIMIT THE BLAST RADIUS

Separate environments, narrow permissions and a token-signing key that never leaves its key service mean a single mistake reaches as little as possible.

FIND AND FIX QUICKLY

Alarms, crash reports and a published disclosure policy bring problems to light quickly, and every fix arrives with a test that keeps it fixed.

How a change reaches readers

INDEPENDENT REVIEW, SIZED TO THE RISK

Every change we make, except a documentation-only edit, is reviewed by an independent reviewer, never its author, before it can merge. Routine dependency updates from automated tools merge only after every required check passes. A tool reads each change and sets the depth of its review:

TIER	APPLIES TO	REVIEW
Documentation	Documentation and records only, within a size limit.	The author checks every statement and link.
Light	Small changes to a few files, away from sensitive areas.	One independent reviewer, one round.
Standard	Everything else.	One independent reviewer working through a full checklist.
High risk	Encryption, sync, sign-in, accounts, payments, the database, the network layer, the server, build pipelines and release tooling.	A reviewer in its own copy of the code at the exact commit, re-running every check itself.

Reviewers focus where a mistake would cost readers most and automated checks cannot see: personal data, payments, privacy and the sync protocol.

Automated guardrails

- **Zero warnings:** Code must finish with no analyzer, compiler or linter warnings, and comments that would silence a warning are rejected.
- **Commit checks:** Before a commit is accepted, a local check rejects secrets, credential files and warning suppressions. The build system repeats the same checks on every change.
- **A protected main branch:** Code reaches the main branch only through a pull request that passes the required checks, including a supply-chain scan of new dependencies.
- **Architecture rules:** A check enforces the app's layering, keeping background services independent of screens and each platform SDK inside the one module that wraps it.

Testing and regressions

Changes ship with unit and widget tests. The sync service is tested against a local emulation of the AWS services it uses, and each app release is driven end to end on every platform it ships to.

A bug a reader reports gets a test that reproduces it before it gets a fix. The fix makes the test pass, and the test stays in the suite permanently; changing one needs the reviewer's recorded approval. Each fix also covers the whole class of bug, with a guard that fails if a new instance appears.

Securing the supply chain

Outside code reaches Tomelet only through a deliberate intake: it waits, it is pinned, and it is recorded.

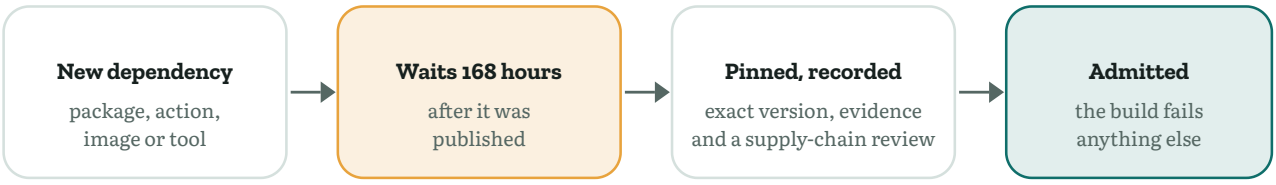


Figure 2. *The intake every new package, build action, container image or tool passes.*

- **A waiting period for outside code:** A new package, build action, container image or tool must have been published for at least 168 hours, be pinned to an exact, immutable version and be recorded in an evidence ledger with a supply-chain review attempted. The build fails a dependency that is too new or not pinned.
- **Pinned build actions:** Build actions are pinned to a full commit hash. Automated update requests arrive weekly or monthly after a cooling-off period, and small updates merge only after every required check passes.
- **Secret scanning:** Every pull request and every change to the main branch is scanned for committed secrets. Full-history secret scans, vulnerability scans of our dependency lock files and software bills of materials run on demand.
- **Trusted and untrusted build machines:** Any build step that holds a credential or can write to a release runs on GitHub's hosted machines. Our own build computers run only work that holds no secret.
- **Signed apps:** Android builds are signed with our release key, Mac builds are signed with our Apple developer identity and notarized by Apple, and Apple and Microsoft sign the apps they distribute through their stores.

Where the service runs

The sync service runs on Amazon Web Services in the United States (us-east-1). The website and the browser app are served by Cloudflare.

The service is built from serverless functions behind Amazon CloudFront, with a DynamoDB database, S3 file storage and AWS Key Management Service. Development, staging and production each run in their own AWS account, and the deployment tools refuse to deploy a stage into the wrong account.

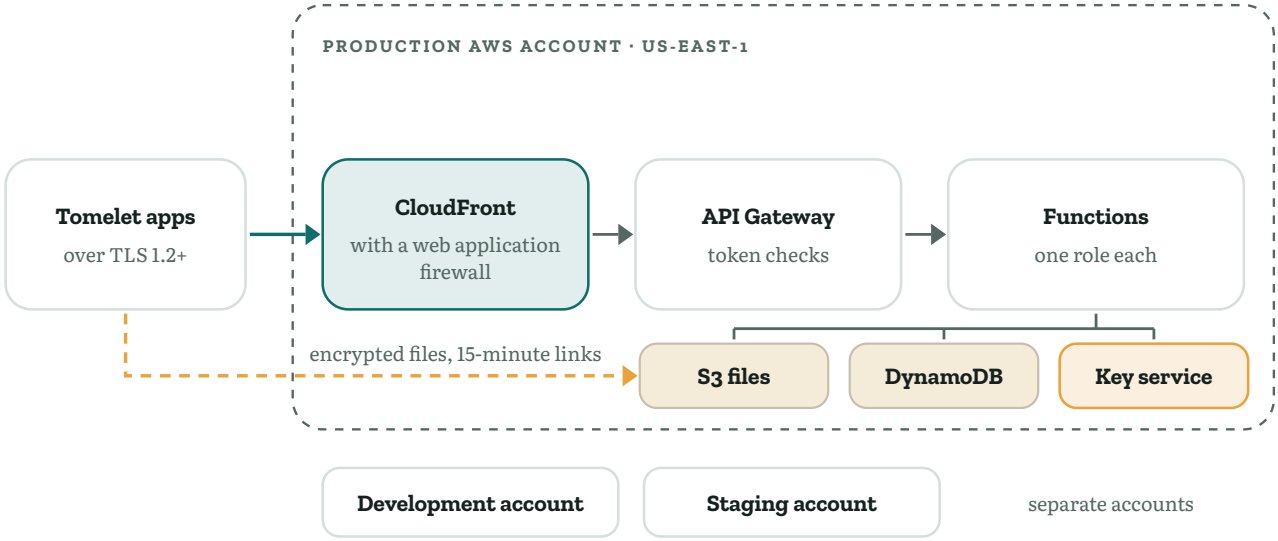


Figure 3. *The production sync service, isolated in its own AWS account. Sync requests pass the edge firewall; encrypted files travel straight to and from storage over 15-minute links.*

Protecting data and the service

- **Encryption in transit:** Connections use TLS 1.2 or newer. Plain HTTP is redirected to HTTPS, our domains send HTTP Strict Transport Security, and storage refuses any request that is not encrypted.
- **Encryption at rest:** AWS encrypts the database and file storage at rest. A reader's synced library is encrypted on their device before that, so what AWS stores is already ciphertext.
- **Edge protection:** A web application firewall with managed rule sets and per-address rate limits sits in front of the sync API, and the API accepts requests only through it, apart from the public listing of the keys that verify its sign-in tokens.
- **Short-lived links:** The apps upload and download encrypted files through links that are valid for 15 minutes and name a single file.
- **Protected signing keys:** Sign-in tokens are signed by a key held in AWS Key Management Service, which never leaves it. The Tomelet Sync security white paper describes tokens and sessions in detail.
- **Least privilege inside the service:** Each function has its own role, reads only its own stage's settings and may use each encryption key only for the one operation it needs. The infrastructure build fails if a new broad permission appears without a written justification.

Access to production

Production is reached three ways, each for one purpose.

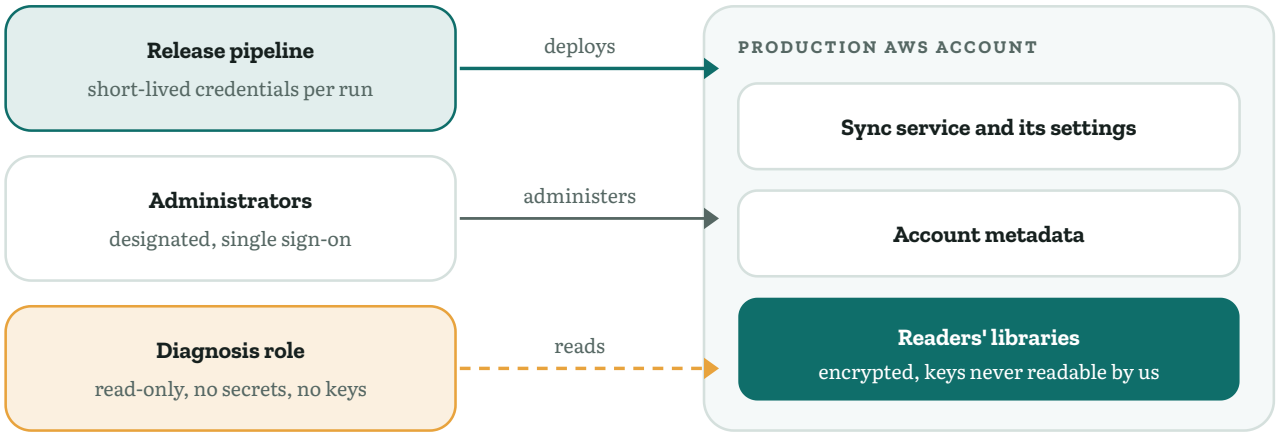


Figure 4. *Each route into production has one purpose, and none of them reaches the contents of a reader's library.*

- **Releases:** The production sync service changes only through the automated release pipeline, started when the responsible team publishes a release. The pipeline deploys to staging, runs its smoke tests and continues to production only when they pass. AWS deployments use short-lived credentials issued for that run, so no AWS key is stored in the build system.
- **Administration:** Administrative access to the production AWS account is limited to designated administrators, through single sign-on.
- **Diagnosis:** Production problems are investigated through a read-only role that cannot change anything, read our secret store or use our signing and encryption keys. It sees the same account metadata the service holds, such as account identifiers, e-mail addresses, device records, sizes and timestamps, and it is used under a standing rule to read the minimum, prefer counts to records and never copy anyone's data.
- **Test accounts:** When a change has to be confirmed in production, the check uses only the company's own account, never a reader's.

Logging and retention

The service never logs request bodies, tokens or ciphertext, and our engineering rules keep personal data out of logs. Each record expires automatically.

RECORD	WHAT IT HOLDS	KEPT
Security log	Sign-ins, newly linked sign-in methods, sign-outs, device removals and account-deletion requests, with IP address and device, to detect account abuse.	**Up to 12 months**
Service logs	One line per request: the account identifier, a shortened device tag, the route, the result, the app version and platform. The gateway's access log adds the IP address and user agent.	**3 months**
Edge and storage logs	Requests as they reach our content network and file storage, including IP address.	**90 days**
Crash reports	Errors from the apps, scrubbed of file names, book details, e-mail and IP addresses, tokens, ciphertext and keys, which readers can turn off in Settings. Errors from the service, with request headers, bodies and IP addresses removed.	**30 days**

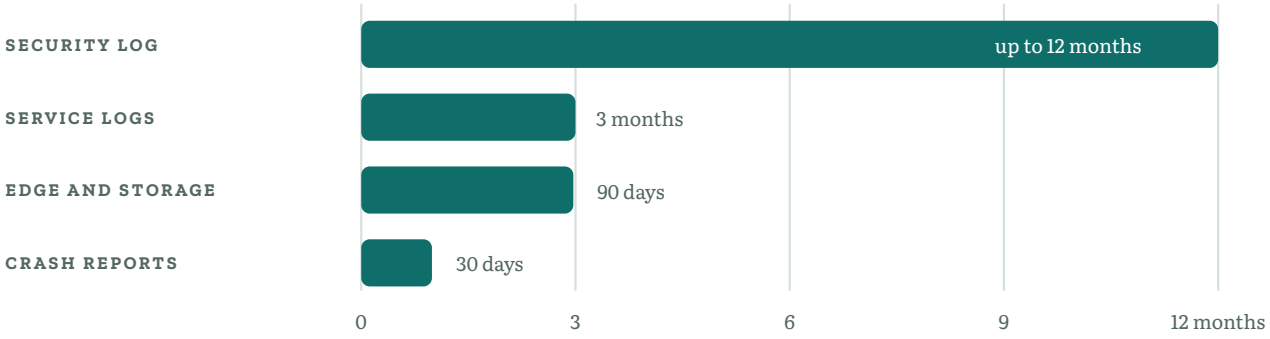


Figure 5. *Every record expires on a fixed schedule. The longest, the security log, exists to detect abuse of a reader's account.*

Monitoring and recovery

Problems are found early, and a reader's library is never at the mercy of the service.

ALARMS

Production alarms watch for server errors, slow responses, throttling, repeated refusals of the same request and other signs of trouble.

CRASH REPORTS

Reports from the apps and the service surface the failures readers meet, including a sync that keeps failing the same way.

BACKUPS

The production database has point-in-time recovery and daily backups, and stored files keep earlier versions for a limited time. Production resources are protected against accidental deletion, and restore procedures are documented.

LOCAL FIRST

A reader's library and notes live on their devices and keep working whenever the service is unavailable.

Deleting your data

Readers can delete their account in the app, on the website or by e-mail.

1. The deletion takes effect at once in the app. On the website it takes effect once the reader approves it on a device, or after a 14-day waiting period they can cancel.
2. The account is closed, every device is signed out and new uploads stop.
3. An hourly job deletes the account's encrypted files and records, and retries any step that fails on its next run.

Section 12 of the Privacy Policy sets out what is deleted and when, and the Terms describe how long encrypted data is kept after a subscription ends. The books and notes on a reader's own devices always stay with them.

Requests from public authorities

On 4 October 2026, TOMELET PTE. LTD. adopted a written procedure for requests for user data from public authorities.

1. Every request goes to the team responsible for legal requests, and no one else answers it.
2. We check that it is lawful: who sent it, its legal basis, and that it binds a Singapore company, taking legal advice where needed.
3. We challenge requests that are unlawful, too broad or unclear, and ask the authority to narrow or withdraw them before we provide anything.
4. We disclose only the minimum the law requires. Books, notes, highlights and reading positions are end-to-end encrypted, and we cannot read them.
5. We tell the affected reader before disclosing, unless the law forbids it.
6. We keep a record of every request (date, authority, legal basis, decision and what was disclosed) for at least five years.
7. We review the procedure once a year.

Anything we could be required to provide is limited to what we hold: account details such as identifiers, the e-mail address and purchase status, the metadata listed in the Tomelet Sync security white paper, and the logs described in this paper.

Responding to vulnerabilities

Security reports go to security@tomelet.app and are handled under our vulnerability disclosure policy at tomelet.app/trust/vulnerability-disclosure.

STEP	WHEN
Acknowledgement	Within 3 working days.
First assessment	Within 10 working days.
Fix or documented mitigation	Within 90 days, with critical issues handled at once.

Each report becomes a private issue at our highest priority, and its fix ships with a permanent regression test and an independent review.

Written runbooks guide the response to production alarms and the rotation of the token-signing key and the secrets the service holds. Production problems are fixed forward, so readers' data is never rolled back.

Conclusion

Tomelet improves every week, and every change meets the same standard of care for readers' privacy. That consistency comes from practices that are written down and enforced by automated checks.

A written engineering standard governs every change, and the responsible teams decide what is built, what is released and who holds which credential. Every code change we make is reviewed independently of its author, outside code is admitted only after it has waited, been pinned and been recorded, and production is reached only through three purpose-built routes. Throughout, a reader's library stays encrypted with keys our servers never hold in readable form.

IN SUMMARY

- **A written engineering standard** applies to every change, with the responsible teams approving releases, spending and credentials.
- **Independent review** of every code change we make, deepest for encryption, sync, sign-in and payments.
- **Automated guardrails** that fail the build: zero warnings, secret scanning, architecture rules and permanent regression tests.
- **A careful supply chain:** outside code waits at least 168 hours, is pinned to an exact version and is recorded with evidence.
- **Least-privilege operations:** separate AWS accounts, short-lived credentials, a read-only diagnosis role and logs that expire.
- **A written procedure** for requests from public authorities, and published response times for security reports.

We will continue to strengthen how Tomelet is built and run, and we will update this paper as those practices change. We welcome findings from the security community through our vulnerability disclosure policy, and questions about privacy at privacy@tomelet.app. The encryption that protects a reader's library is described in the companion Tomelet Sync security white paper.

Tomelet

Read the book the way it was printed.

TRUST CENTER
tomelet.app/trust

SECURITY
security@tomelet.app

PRIVACY
privacy@tomelet.app