SPEC-022 — Document Archive Implementation

> **MC #100025** | Published 2026-05-08 | Status: Approved (Pattern 3 — Skybound)
> **Related:** ADR-022 • COMPLIANCE-022

SPEC-022: Document Archive Implementation — Pattern 3 (Blob Queue)

Status: Draft — awaiting CodeCraft + FlowForge dispatch **Date:** 2026-05-08 **Author:** CodeCraft (MC #100025, Subtask 2) **ADR:** ADR-022-document-archive-strategy.md (Pattern 3 selected, 4.6/5 weighted score) **CEO Decisions baked in:** D1 (5min cron), D2 (delete immediately on success), D3 (one correspondent per org) **Related:** MC #100025 (this task), MC #100004 (IMAP→Paperless pipe, BookStack #2862)

---

1. Overview

Pattern 3 (App→Shared Blob→Archiver Job) is the selected architecture for Bilko→Paperless-ngx document archival. Bilko backend writes generated PDFs plus a `.meta.json` sidecar to a dedicated Cloudflare R2 staging bucket (`bilko-archive-queue`). A separate Cloud Run job (`archiver-worker`) polls the queue on a 5-minute cron (per CEO decision D1), uploads each document to Paperless-ngx at `archive.alai.no` via the Paperless REST API, and deletes the R2 object immediately upon confirmed upload (per CEO decision D2). Multi-tenant isolation is enforced by the worker reading `organizationId` from R2 object metadata and applying an `org:` tag plus a per-org correspondent (per CEO decision D3) on every Paperless document. Bilko user experience is never degraded by Paperless downtime; R2 is the authoritative queue and all retry semantics live in the worker. See ADR-022 §Decision Drivers for the full pattern comparison and rejection rationale for Patterns 1 and 2.

---

2. Components

Component	Location	Type	Purpose
---	---	---	---
`ArchiveService`	`apps/api/src/main/kotlin/no/alai/bilko/services/ArchiveService.kt`	New Kotlin service	Writes PDF + `.meta.json` sidecar to R2 `bilko-archive-queue` bucket; returns `ArchiveJobId`
R2 bucket `bilko-archive-queue`	Cloudflare R2 (separate from existing `AWS_S3_BUCKET`)	New bucket	Staging queue for pending Paperless uploads
R2 bucket `bilko-archive-dlq`	Cloudflare R2	New bucket	Dead-letter queue for objects that failed 3 upload attempts
`archiver-worker`	`apps/archiver-worker/`	New Cloud Run job (Node.js — see §10)	Polls R2 → uploads to Paperless → deletes R2 objects
Cloud Scheduler trigger	GCP Cloud Scheduler `bilko-archiver-cron`	New scheduler job	Fires `archiver-worker` Cloud Run job every 5 minutes (per CEO decision D1)
Flyway migration `V_archive_status`	`apps/api/src/main/resources/db/migration/`	New migration	Adds `archive_status`, `archive_job_id`, `paperless_doc_url`, `archived_at` columns to `invoices` and future document tables
`ArchiveAuditLog`	`apps/api/src/main/kotlin/no/alai/bilko/model/ArchiveAuditLog.kt` + Flyway migration	New DB table	Per-document archive status: `pending`, `archived`, `failed`
Bilko DB table `org_paperless_cache`	PostgreSQL, Flyway migration	New table	Caches `organizationId → paperless_correspondent_id` and `organizationId → paperless_org_tag_id` to avoid repeat API calls

---

3. Interfaces

3.1 ArchiveService — Kotlin signature

```
// File: apps/api/src/main/kotlin/no/alai/bilko/services/ArchiveService.kt
// Package: no.alai.bilko.services
```

data class SourceDoc(val organizationId: UUID, val organizationName: String, val documentType: DocumentType, // enum: INVOICE | CONTRACT | CARE_PLAN | INCIDENT_REPORT | ONBOARDING val documentBuffer: ByteArray, // raw PDF bytes val sha256: String, // hex SHA-256 of documentBuffer val metadata: Map // e.g. { "invoiceNumber": "2024-001", "contractId": "abc" })

data class ArchiveOptions(val priority: ArchivePriority = ArchivePriority.NORMAL // NORMAL | HIGH (for future urgency flag))

// Return type — opaque job ID (R2 object key) typealias ArchiveJobId = String

// Primary entry point — called by InvoiceService, ContractService, etc. // Throws ArchiveWriteException (wraps R2 S3 error) on R2 write failure. // NEVER throws on Paperless unavailability (async path). suspend fun archive(sourceDoc: SourceDoc, options: ArchiveOptions = ArchiveOptions()): ArchiveJobId

Callers (e.g. `InvoiceService.generatePDF()`) catch `ArchiveWriteException` and return HTTP 503 to the user with body `{"error": "Document archived pending. Retry in 5 minutes.", "code": "ARCHIVE_QUEUE_FAILURE"}`. The Bilko UX is decoupled per ADR-022 §Consequences (Positive #1): R2 write failure is the only user-visible failure; Paperless unavailability is invisible to the user.

3.2 R2 Object Schema

Object key convention:

```
org///.pdf
org///.meta.json
```

Example:

```
org/550e8400-e29b-41d4-a716-446655440000/invoice/a1b2c3d4...ef.pdf
org/550e8400-e29b-41d4-a716-446655440000/invoice/a1b2c3d4...ef.meta.json
```

Using SHA-256 as the object key suffix provides idempotent R2 writes: re-upload of identical PDF bytes overwrites the same key (R2 last-writer-wins), preventing queue bloat on Bilko retry paths.

`.meta.json` schema:

```
{
  "schemaVersion": "1",
  "r2Uuid": "",
  "organizationId": "550e8400-e29b-41d4-a716-446655440000",
  "organizationName": "Firma AS",
  "documentType": "invoice",
  "bilkoDocumentId": "",
  "invoiceNumber": "2024-001",
  "contractId": null,
  "timestamp": "2026-05-08T10:30:00Z",
  "sha256": "a1b2c3d4...ef",
  "retryCount": 0,
  "lastAttemptAt": null,
  "lastError": null
}
```

- `retryCount` and `lastError` are mutated in-place by the worker on failure (R2 PUT of updated `.meta.json`).
- `r2Uuid` (= `sha256`) is the Paperless dedup key: `bilko-source-uuid:` tag on the Paperless document.
- `bilkoDocumentId` allows the worker to write back the Paperless doc URL to the Bilko DB audit row.

Content-type: PDF object → `application/pdf`. `.meta.json` → `application/json`.

Retention class: Standard (no Infrequent Access — objects are short-lived by design).

3.3 Worker ? Paperless API call

The worker calls `POST https://archive.alai.no/api/documents/post_document/` as `multipart/form-data`:

```
POST /api/documents/post_document/
Host: archive.alai.no
CF-Access-Client-Id:
CF-Access-Client-Secret:
Authorization: Token
Content-Type: multipart/form-data
```

Fields: document — PDF binary (required) title — " — " (e.g. "Invoice — Firma AS 2026-05-08") correspondent — (integer, pre-resolved by worker — see §5) document_type — (integer, mapped

from documentType enum) tags — [, , ,] created — custom_fields — [{"field": , "value": ""}, {"field": , "value": ""}, {"field": , "value": ""}]

The worker resolves `correspondent_id`, `document_type_id`, and tag IDs prior to the upload call, using the `org_paperless_cache` Bilko DB table (see §5). All IDs are integers assigned by Paperless on creation.

Reuse of paperless-upload.js: The worker is Node.js (see §10 for language decision). It may directly import or inline logic equivalent to `~/system/tools/paperless-upload.js` (MC #100004). The worker should NOT import the file at runtime from the system tools path — instead, the logic (multipart form construction, CF Access header injection, retry) is copied into `apps/archiver-worker/src/paperlessClient.js` with full ownership by the Bilko repo. This avoids coupling the Bilko Cloud Run container to the ALAI system tools directory.

Worker side-effect on success (D2):

1. `DELETE` R2 PDF object key. 2. `DELETE` R2 `.meta.json` sidecar key (per CEO decision D2 — delete immediately, no buffer). 3. `UPDATE` Bilko DB `archive_audit_log` row: `archive_status = 'archived'`, `paperless_doc_url = `, `archived_at = NOW()`. 4. `UPDATE` Bilko DB source document table (e.g. `invoices`): `archive_status = 'archived'`, `paperless_doc_url = `, `archived_at = NOW()`.

Worker updates the Bilko DB via a thin internal HTTP endpoint on `bilko-api` Cloud Run service (authenticated with a shared internal API key stored in Secret Manager), OR directly via Cloud SQL connection if the worker runs in the same GCP VPC. **Recommendation:** internal HTTP endpoint on `bilko-api` is safer (no direct DB access from worker, follows existing service boundary). Endpoint: `PATCH /internal/v1/archive-audit/{bilkoDocumentId}` — worker-to-api call, not user-facing.

---

4. Auth Model

4.1 Bilko backend ? R2 (write to `bilko-archive-queue`)

Reuses existing R2 credentials already in Bilko Cloud Run environment: `AWS_S3_ENDPOINT`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`, `AWS_S3_BUCKET`. A new env var `AWS_S3_ARCHIVE_BUCKET=bilko-archive-queue` routes `ArchiveService` writes to the separate archive bucket. The same Cloudflare R2 token is scoped to both buckets via R2 token policy. No new SA credential needed for Bilko backend.

4.2 Worker ? R2 (read + delete from `bilko-archive-queue`)

Recommendation: separate R2 API token for the worker, scoped to `bilko-archive-queue` and `bilko-archive-dlq` with read + delete permissions. Do NOT share the Bilko production R2 token (which has write access to the main receipts/PDF bucket) with the worker. Principle of least privilege: worker should not be able to touch the main Bilko file storage bucket.

Worker credential: new Cloudflare R2 API token stored in GCP Secret Manager as `bilko-archiver-r2-access-key-id` and `bilko-archiver-r2-secret-access-key`. Provisioned by FlowForge as part of the worker deployment Terraform module.

4.3 Worker ? Paperless (archive.alai.no)

Worker requires two credentials:

- `CF_ACCESS_CLIENT_ID` + `CF_ACCESS_CLIENT_SECRET` — Cloudflare Access service token
- `PAPERLESS_API_TOKEN` — Paperless-ngx API token

Recommendation: create a new dedicated Bitwarden item `Paperless API Token — bilko-archiver-worker` (separate from the existing `Paperless API Token — anvil` item referenced in MC #100004).

Rationale: the existing anvil token is shared with the IMAP→Paperless daemon (MC #100004). If the worker token is rotated (e.g. Bilko security incident), the IMAP daemon must not be affected. Separate tokens allow independent rotation. Both tokens have equal Paperless API access (same permissions) but are separate credentials with separate audit trails in Paperless and Bitwarden.

Similarly, create a new CF Access service token `bilko-archiver-worker` in Cloudflare Zero Trust, separate from any existing `archive-alai-no CF Access` token. Stored as: `bilko-archiver-cf-access-client-id` and `bilko-archiver-cf-access-client-secret` in GCP Secret Manager.

4.4 Bilko backend ? Paperless

FORBIDDEN. The Bilko backend NEVER calls Paperless directly. Pattern 3 rationale from ADR-022 §Pattern 2 rejection: "Paperless becomes hot-path dependency — if archive.alai.no is down, Bilko document upload fails. User sees error." The R2 queue decouples Bilko from Paperless availability entirely.

---

5. Multi-Tenant Scoping

5.1 Correspondent strategy (per CEO decision D3 — one per org)

Correspondent name pattern: `org-` (e.g. `org-550e8400-e29b-41d4-a716-446655440000`).

On first archive for a new organization, the worker calls:

```
POST https://archive.alai.no/api/correspondents/
{ "name": "org-", "match": "", "matching_algorithm": 0, "is_insensitive": false }
```

The returned `correspondent.id` is stored in Bilko DB table `org_paperless_cache`:

```
CREATE TABLE org_paperless_cache (
    organization_id UUID PRIMARY KEY REFERENCES organizations(id),
    paperless_correspondent_id INTEGER NOT NULL,
    paperless_org_tag_id INTEGER NOT NULL,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

On subsequent archives for the same org, the worker reads from `org_paperless_cache` (HTTP GET to `bilko-api` internal endpoint `GET /internal/v1/paperless-cache/{organizationId}`). Cache miss triggers correspondent + tag creation and cache write. This avoids one Paperless API round-trip per document after first archive.

The human-readable org name (`organizationName` from the `.meta.json`) is NOT used as the Paperless correspondent name — `org-` is canonical to prevent name collisions and to survive org renames in Bilko.

5.2 Tag strategy

Every archived document receives exactly these tags:

Tag	Purpose	Created by
`org:`	Tenant isolation — one tag per Bilko org	Worker on first archive for org

`doc-type:invoice` (or contract, care-plan, incident-report, onboarding)	Document type filter	Worker — static set, pre-created in Paperless during initial setup
`bilko-source`	Identifies all documents archived from Bilko (across all orgs)	Pre-created in Paperless during initial setup
`bilko-source-uuid:`	Idempotency dedup key — prevents duplicate Paperless documents	Worker — unique per document

The worker stores `paperless_org_tag_id` in `org_paperless_cache` alongside `paperless_correspondent_id`. Document-type tag IDs and the `bilko-source` tag ID are stored in worker environment config (`PAPERLESS_TAG_IDS_MAP` env var as JSON: `{"invoice": 12, "contract": 13, ...}`). These are set once during initial Paperless setup and do not change.

5.3 Tenant search in Paperless

To retrieve all documents for an org:

```
GET https://archive.alai.no/api/documents/?tags__id__in=&page=1&page_size=25
```

For filtering by doc type within an org:

```
GET https://archive.alai.no/api/documents/?tags__id__in=,
```

This is the canonical Paperless query pattern. Cross-tenant queries are impossible if the caller only has access to their own `org_tag_id`. (Note: Paperless does not natively enforce per-tag ACLs — isolation is enforced by the Bilko application layer controlling which `org_tag_id` each user can query.)

---

6. Retention Policy

6.1 R2 staging bucket (`bilko-archive-queue`)

- **On successful upload to Paperless:** DELETE immediately (per CEO decision D2). No buffer.

Rationale (ADR-022 §Open Questions, CEO decision D2): Paperless is source of truth post-archival. R2 is a queue, not a backup.

- **On failed upload (retry count < 3):** Object remains in R2. Worker increments `retryCount` in

`.meta.json` on each failure. Object will be retried on next cron invocation.

- **On failed upload (retry count = 3):** Worker moves object (COPY then DELETE) to `bilko-archive-dlq`

bucket and sends alert (Slack or email to `dev@alai.no`, the existing alert address per BUILD-BLUEPRINT line 302). Object in DLQ retained for 7 days then auto-deleted via R2 lifecycle rule.

- **Orphan protection:** R2 lifecycle rule on `bilko-archive-queue`: objects older than 7 days

trigger alert (Cloud Monitoring metric → alert policy). This catches worker failures that leave objects stranded without incrementing retry count.

6.2 Paperless retention

TBD — pending legal/compliance review by Dr. Sarah Chen (S3, healthcare compliance). Interim recommendations based on applicable law:

Document Type	Recommended Retention	Legal Basis
---	---	---
Invoices	7 years	Norway Bokføringsloven §13; Serbia Zakon o računovodstvu; BiH equivalent
Contracts	Indefinite until expiry + 5 years	Standard contract law (Norway, Serbia, BiH, Croatia)
Care plans	25 years	NHS/CQC standard (applicable if Bilko expands to UK healthcare)
Incident reports	7 years	General audit retention standard
Onboarding documents	5 years post-customer-offboarding	GDPR Art. 5(1)(e) storage limitation

Paperless retention enforcement is OUT OF SCOPE for this implementation phase. Configure via Paperless Workflow rules in a subsequent MC (FlowForge + Dr. Sarah Chen).

---

7. Retry Semantics

7.1 Worker retry loop

On each 5-minute cron invocation (per CEO decision D1), the worker:

1. Lists all objects under `org/` prefix in `bilko-archive-queue` (R2 `ListObjectsV2` equivalent). 2. For each `.meta.json` sidecar found (PDF existence implied by paired key): a. Read `retryCount`. If `retryCount >= 3`: move to DLQ, skip. b. Fetch corresponding PDF bytes. c. Attempt Paperless dedup

check: `GET /api/documents/?custom_fields__value=` — if document already exists in Paperless (Bilko double-run), skip upload, DELETE R2 object, update Bilko DB (idempotent cleanup). d. Attempt upload. On success: DELETE R2 objects, update Bilko DB audit log. e. On failure: increment `retryCount` in `.meta.json`, write updated `.meta.json` back to R2, log error. Leave PDF object in place.

7.2 Idempotency

R2 object key = `org///.pdf`. If Bilko backend calls `ArchiveService.archive()` twice for the same document (e.g. invoice regenerated), R2 write is idempotent (same key, same bytes). Worker sees one object, uploads once.

Paperless dedup via `bilko-source-uuid:` tag: if worker runs twice before completing a DELETE (e.g. crash between upload and delete), the second run finds the tag already present in Paperless and skips re-upload. Only deletes R2 + updates Bilko DB.

7.3 DLQ and alerting

Object moved to `bilko-archive-dlq` after 3 failures. Alert fires to `dev@alai.no` (Cloud Monitoring alert via existing `TF_VAR_alert_email` in BUILD-BLUEPRINT line 302). DLQ objects require manual triage — either re-queue by moving back to `bilko-archive-queue` (resets `retryCount`) or manually upload to Paperless and delete from DLQ.

---

8. Error Handling

8.1 Bilko backend — R2 write failure

`ArchiveService.archive()` throws `ArchiveWriteException`. Caller (e.g. `InvoiceService.generatePDF()`) catches and:

- Returns HTTP 503 to user: `{"error": "Document archived pending, retry in 5 minutes.", "code": "ARCHIVE_QUEUE_FAILURE"}`.
- Writes `archive_status = 'failed'` to `archive_audit_log` (allows re-trigger from admin UI in future).
- Does NOT fail the invoice PDF generation itself (PDF is already in main R2 bucket).

Per ADR-022 §Consequences (Positive #1): R2 write failure is a degraded but non-blocking UX state. The invoice PDF is already saved to the main Bilko R2 bucket. Only archival is deferred.

8.2 Worker — Paperless API errors

Error	Action
----------	----------
401 Unauthorized	Token expired/rotated. Alert `dev@alai.no` immediately. Worker stops processing (do not retry — all subsequent calls will also 401). Manual token rotation required.
403 Forbidden	CF Access token issue. Same action as 401.
429 Rate Limited	Exponential backoff within single cron run: wait 2s, 4s, 8s (cap at 30s). If still failing after 3 attempts, leave object in R2 for next cron.
500/502/503	Retry up to 3 times within cron run with exponential backoff (2s, 4s, 8s). If all fail, increment `retryCount` in `.meta.json`, leave for next cron.
Network timeout	Same as 5xx. Worker `fetch()` timeout = 30 seconds per request.

8.3 Worker — Cloud Run job retry policy

Cloud Scheduler retry policy: max 3 retries with 30s backoff on Cloud Run job invocation failure (distinction: this is job-launch failure, not Paperless upload failure — separate from §7 object-level retries). If the job crashes mid-run, objects remain in R2 and are processed on next cron invocation.

8.4 Worker — structured logging

All worker log lines emit JSON to stdout (Cloud Run log aggregation reads stdout). Required fields:

```
{
  "severity": "INFO|WARNING|ERROR",
  "timestamp": "",
  "r2Key": "
```

---

9. Observability

9.1 Worker metrics (Cloud Monitoring custom metrics or stdout-JSON)

Metric	Type	Description
---	---	---
`archive_jobs_processed_total`	Counter	Total R2 objects successfully uploaded to Paperless
`archive_jobs_failed_total`	Counter	Total R2 objects that failed upload (all retry attempts)
`archive_queue_depth`	Gauge	Count of objects currently in `bilko-archive-queue` (R2 ListObjectsV2 at job start)
`archive_e2e_latency_seconds`	Histogram	Time from R2 object `timestamp` in `.meta.json` to confirmed Paperless upload
`archive_dlq_depth`	Gauge	Count of objects in `bilko-archive-dlq` (alert if > 0)

Metrics emitted as structured log lines (Cloud Run → Cloud Logging → Log-based metrics) OR via Cloud Monitoring custom metric API from worker. **Recommendation:** log-based metrics (simpler, no extra SDK dependency in worker). Cloud Monitoring log-based metric filter on `action` field.

9.2 Alert policies

Condition	Severity	Channel
---	---	---
`archive_dlq_depth > 0`	P1	`dev@alai.no` (existing Cloud Monitoring alert email)
`archive_queue_depth > 500` for 15 minutes	P2	`dev@alai.no` — worker may have stopped
Worker job not invoked in >10 minutes	P2	Cloud Scheduler missed execution alert
`archive_jobs_failed_total` > 5 in 1 hour	P2	`dev@alai.no`
Paperless 401 in worker logs	P1	`dev@alai.no` — token rotation required

9.3 Bilko DB audit log

Every document that passes through `ArchiveService.archive()` gets a row in `archive_audit_log`:

```
CREATE TABLE archive_audit_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    organization_id UUID NOT NULL REFERENCES organizations(id),
    bilko_document_id UUID NOT NULL,     -- FK to invoices.id, contracts.id, etc.
    document_type VARCHAR(50) NOT NULL,
    r2_object_key TEXT NOT NULL,
    sha256 TEXT NOT NULL,
    archive_status VARCHAR(20) NOT NULL DEFAULT 'pending',  -- pending | archived | failed
    paperless_doc_id INTEGER,
    paperless_doc_url TEXT,
    archived_at TIMESTAMPTZ,
    retry_count INTEGER NOT NULL DEFAULT 0,
    last_error TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
CREATE INDEX idx_archive_audit_log_org ON archive_audit_log(organization_id);
CREATE INDEX idx_archive_audit_log_doc ON archive_audit_log(bilko_document_id);
CREATE INDEX idx_archive_audit_log_status ON archive_audit_log(archive_status) WHERE
archive_status != 'archived';
```

The worker updates this table via `PATCH /internal/v1/archive-audit/{bilkoDocumentId}` on the `bilko-api` service (authenticated internal call). The `bilko-api` internal endpoint is protected by a shared secret (`INTERNAL_API_KEY`) stored in Secret Manager, injected into both `bilko-api` Cloud Run service and `archiver-worker` Cloud Run job at deploy time.

---

10. Open Questions for Next Phase

1. **Worker language — Kotlin vs Node.js:** ADR-022 §Phase 2 lists "Kotlin/Ktor or Node.js, TBD." Recommendation: **Node.js**, reusing paperless-upload.js logic (inlined into `apps/archiver-worker/src/paperlessClient.js`). Rationale: (a) faster to ship — Node worker requires zero Gradle/JVM setup, shorter Docker image, simpler Cloud Run job config; (b) the heaviest logic (multipart Paperless upload) already exists in Node (MC #100004); (c) Kotlin adds value for domain-heavy Bilko services, not for a thin queue-poller. Downside: two runtimes in the Bilko repo (Kotlin + Node). Acceptable given worker is a standalone job in `apps/archiver-worker/`, isolated from `apps/api/`. **CodeCraft must confirm this choice before implementation starts.**

2. **Backfill for existing Bilko documents not yet archived:** Out of scope for first ship. All pre-existing invoices, contracts in Bilko DB are unarchived. A backfill worker (one-shot Cloud Run job, reads Bilko DB → writes to R2 queue → worker picks up) is a natural Phase 2 task. Create child MC when this phase ships.

3. **DR — Paperless VM outage >24h:** Worker retries indefinitely (R2 objects accumulate). At 24h queue backlog (estimated ~2,880 cron invocations), `archive_queue_depth > 500` alert fires to ops. Worker will self-heal on Paperless recovery without intervention. If VM is permanently lost: restore Paperless from Azure VM backup (existing backup schedule assumed — verify with FlowForge). R2 queue is the authoritative backlog; no documents are lost.

4. **GDPR Art. 17 erasure flow:** When a Bilko org deletes their account, all archived documents must be deleted from Paperless (`DELETE /api/documents/{id}`) and the correspondent deleted (`DELETE /api/correspondents/{id}`). This is a separate erasure worker, out of scope for this implementation phase. File child MC at same time as backfill MC (Phase 2).

5. **Bilko internal API endpoint auth (`/internal/v1/`):** The worker-to-api callback for updating `archive_audit_log` requires an internal auth mechanism. Shared secret (`INTERNAL_API_KEY`) is recommended for MVP. mTLS (Cloud Run service-to-service auth via OIDC token) is more secure and already supported by GCP — recommend upgrading to mTLS in Phase 2. Child MC for FlowForge.

References

- ADR-022-document-archive-strategy.md — pattern decision, rejection rationale, CEO decisions
- BUILD-BLUEPRINT.md line 64 — Cloudflare R2 existing configuration (`AWS_S3_BUCKET`, `AWS_S3_ENDPOINT`)
- BUILD-BLUEPRINT.md lines 192-193 — multi-tenancy model (`organizationId` discriminator)
- BUILD-BLUEPRINT.md line 302 — alert email (`dev@alai.no`, `TF_VAR_alert_email`)
- BUILD-BLUEPRINT.md §9 — GCP Cloud Run deployment model, Terraform IaC structure, Secret Manager
- MC #100004 — IMAP→Paperless pipe, `~/system/tools/paperless-upload.js`, BookStack #2862
- MC #100025 — parent task (ADR + spec)
- Paperless-ngx API: https://docs.paperless-ngx.com/api/
- Cloudflare R2 S3-compatible API: https://developers.cloudflare.com/r2/api/s3/api/
- GCP Cloud Run jobs: https://cloud.google.com/run/docs/create-jobs
- GCP Cloud Scheduler: https://cloud.google.com/scheduler/docs

---

Revision #3
Created 2026-05-08 19:29:45 UTC by John
Updated 2026-06-14 20:02:41 UTC by John