ADR-022 — Document Archive Strategy

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

ADR-022: Document Archive Strategy for Paperless-ngx Integration

Status: Proposed **Date:** 2026-05-08 **Author:** Skybound (ALAI SaaS Architecture) **Related:** MC #100025, MC #100004 (IMAP→Paperless pipe)

Context

Business Need

Bilko generates high-value, low-frequency documents requiring long-term archival in a centralized, searchable repository:

- **Signed contracts** (customer/vendor onboarding)
- **Invoices** (generated PDF with QR code, pdfkit)
- **Care plan PDFs** (if Bilko expands to healthcare use cases)
- **Incident reports** (audit trail documentation)
- **Signed onboarding documents** (scanned receipts, identity verification)

Current state: documents generated in-app (PDF via pdfkit), stored in Cloudflare R2 (configured, see BUILD-BLUEPRINT.md line 64), but **no archival pipe to Paperless-ngx** at archive.alai.no.

CEO question (2026-05-08): "Does Bilko have email→Paperless integration?" Answer: NO. This ADR selects the archival pattern before implementation begins.

Paperless-ngx Environment

- **URL:** `https://archive.alai.no`
- **Access:** Behind Cloudflare Access (service token required)
- **Credentials:** Paperless API token in Bitwarden (`Paperless API Token — anvil`, user=alembasic, created 2026-05-03)
- **Hosting:** Separate Azure VM (not GCP like Bilko)
- **Cross-cloud path:** GCP Cloud Run (europe-north1) → Azure VM (westeurope assumed)
- **IMAP pipe (MC #100004):** Daemon polls `alem@alai.no`, uploads attachments to Paperless. BookStack runbook page #2862. **Operational, general-purpose.**

Bilko Technical Constraints

From BUILD-BLUEPRINT.md:

- **Multi-tenancy:** Organization-scoped (`organizationId` discriminator). Every DB record carries `organizationId`. Middleware (`org-scope.ts`) extracts from JWT. No cross-tenant data leak.
- **Stack:** Kotlin/Ktor backend (apps/api/, port 8080), Next.js 15 frontend, PostgreSQL 15, Cloudflare R2 (S3-compatible), SendGrid (SMTP), GCP Cloud Run (multi-region).
- **Auth:** JWT (access token 15min, refresh token 7d httpOnly).
- **File storage:** Cloudflare R2 bucket (AWS_S3_BUCKET, S3-compatible API).
- **Document volumes:** Low-frequency, high-value (estimated <100 docs/day across all tenants at MVP scale, 10–50 orgs).
- **Regions:** EU residency for GDPR (data must stay in EU).
- **Deployment:** GCP Cloud Run (apps/api/ + apps/web/), Cloud SQL PostgreSQL, Terraform IaC.

Paperless-ngx Multi-Tenant Capabilities

Paperless-ngx is NOT multi-tenant at the DB schema level. Tenant isolation MUST be enforced via:

1. **Tags** (e.g., `org:uuid-abc123`) 2. **Correspondent** field (one correspondent per tenant, e.g., "Org: Firma AS") 3. **Document Type** field (e.g., "Invoice", "Contract", "Care Plan") 4. **Custom Fields** (optional key-value metadata)

All three can be set via `POST /api/documents/post_document/` API.

Decision

Recommended Pattern: Pattern 3 — App→Shared Blob→Archiver Job (Batch)

Bilko will write documents to a **Cloudflare R2 bucket** (already in use) with metadata attached (organizationId, documentType, timestamp). A separate **Cloud Run job** (or Cron Worker, TBD in implementation phase) reads the queue and uploads to Paperless-ngx via direct API call, applying multi-tenant tags (org:uuid-xxx), correspondent, and document type.

Fallback during outages: If archiver job fails or Paperless is unavailable, documents remain in R2 with idempotent retry semantics. Bilko user experience is never degraded by Paperless downtime.

Decision Drivers

Criterion	Weight	Pattern 1 (Email)	Pattern 2 (Direct API)	Pattern 3 (Blob Queue)
---	---	---	---	---
Multi-tenant scoping	HIGH	3/5	4/5	5/5
Bilko coupling	HIGH	5/5	2/5	5/5
Paperless coupling	HIGH	4/5	1/5	5/5
Retry/idempotency	HIGH	2/5	3/5	5/5
Auth model	MED	5/5	2/5	4/5
Dev velocity	MED	5/5	4/5	3/5
Ops surface	MED	4/5	5/5	3/5
Cross-cloud friendliness	MED	5/5	3/5	5/5
Dedup strategy	LOW	2/5	4/5	5/5
Scalability (>1k docs/day)	LOW	2/5	5/5	5/5
TOTAL (weighted sum)	—	**3.6/5**	**3.2/5**	**4.6/5**

Scoring rationale:

- **Multi-tenant scoping:** Pattern 3 allows worker to read `organizationId` from R2 metadata and apply consistent Paperless tags (org:uuid-xxx) + correspondent. Pattern 1 must

encode tenant in email subject or attachment metadata (fragile). Pattern 2 requires Bilko backend to hold tenant-to-Paperless-tag mapping (extra logic in hot path).

- **Bilko coupling:** Pattern 3 decouples Bilko completely (fire-and-forget to R2). Pattern 2 tightly couples Bilko to Paperless availability (degraded UX if archive.alai.no is down).
- **Paperless coupling:** Pattern 3 isolates Paperless availability from Bilko runtime. Pattern 2 makes Paperless a hot-path dependency.
- **Retry/idempotency:** Pattern 3 uses R2 versioning + worker retry (Cloud Run job cron or queue). Pattern 1 has weak email delivery guarantees (no DLQ). Pattern 2 requires Bilko to implement retry logic (failed upload = user sees error).
- **Auth model:** Pattern 1 reuses existing IMAP→Paperless pipe (zero new auth surface). Pattern 3 requires worker to hold CF Access token + Paperless API token (already exists in Bitwarden, see MC #100004). Pattern 2 requires Bilko backend to hold CF Access creds (rotation surface, Bilko team must manage Paperless tokens).
- **Dev velocity:** Pattern 1 is fastest (SMTP send, zero new code in Bilko). Pattern 3 requires worker provisioning + monitoring.
- **Ops surface:** Pattern 2 is simplest (no worker). Pattern 3 adds worker component.
- **Cross-cloud friendliness:** Pattern 3 is cloud-agnostic (R2 bucket is S3-compatible, worker can run anywhere). Pattern 2 crosses GCP→Azure directly (network latency, no queue).
- **Dedup:** Pattern 3 can use R2 object key = SHA256 of doc (idempotent). Pattern 1 relies on email Message-ID (can duplicate if retry). Pattern 2 requires Bilko to track uploaded doc IDs.
- **Scalability:** Pattern 1 has email attachment size limits (SendGrid = 30MB total, one.com Dovecot = unknown). Pattern 3 and 2 scale to multi-GB PDFs if needed.

Consequences

Positive

1. **Bilko never blocks on Paperless downtime.** User uploads document, gets immediate success (R2 write ~50ms), archival happens async. 2. **Idempotent retry semantics.** Worker crashes mid-upload? R2 object still there, retry on next cron run (dedupe via object key or Paperless custom_fields SHA256). 3. **Multi-tenant isolation enforced at archival layer.** Worker reads `organizationId` from R2 metadata → applies `tags=org:uuid-abc123` + `correspondent="Firma AS (uuid-abc123)"` in Paperless. Search in Paperless UI: filter by tag = instant tenant-scoped results. 4. **Scales to additional archive targets.** Worker can fan-out to Paperless + S3 Glacier + OneDrive (future). Bilko unchanged. 5. **Zero cross-cloud hot-path latency.** Bilko writes to R2 (same Cloudflare edge region as app), worker polls async. 6. **Reuses existing R2 bucket.** No new storage provisioning. R2 lifecycle policy can auto-delete after N days post-archive (cost optimization).

Negative

1. **Eventual consistency.** Document archived 1–15 minutes after user upload (depends on worker cron interval). If CEO searches Paperless 30 seconds after upload, doc not yet there. 2. **Additional ops surface.** Worker must be monitored (cron health check, dead-letter queue for failed uploads). 3. **Dev velocity slower than Pattern 1.** Must scaffold worker + deploy pipeline + monitoring.

Neutral

1. **Auth surface expands slightly.** Worker holds CF Access token + Paperless API token. Rotation = worker redeploy or Secret Manager update (already standard for GCP Cloud Run). 2. **R2 becomes queue.** If worker stops (VM crash, deployment), R2 accumulates unprocessed docs. Recovery = restart worker, process backlog.

Alternatives Considered

Pattern 1 — App?Email?Paperless (Relay)

How it works: Bilko backend sends document as attachment to dedicated inbox (e.g., `bilko-archive@alai.no`). Daemon (MC #100004 pipe) polls inbox, uploads to Paperless.

Pros:

- **Zero code in Bilko backend.** Just `sendgrid.send({ to: 'bilko-archive@alai.no', attachment: pdfBuffer })`. Reuses existing SendGrid integration.
- **Reuses MC #100004 pipe 1:1.** Daemon already operational.
- **Low coupling.** Bilko unaware of Paperless API.
- **Cross-cloud friendly.** Email = universal transport.
- **Easy to add more sources.** Any system can email attachments to dedicated inbox.

Cons:

- **Email is a weak queue.** No ordering guarantees, delivery can fail silently, dedup harder (Message-ID not unique across retries).
- **Attachment size limits.** SendGrid = 30MB total per email. Large invoice batches or scanned multi-page contracts may exceed.
- **Latency.** IMAP daemon polls every N minutes (configured in MC #100004). User uploads doc at 10:00, daemon polls at 10:15 → 15min delay.
- **Multi-tenant scoping fragile.** Must encode `organizationId` in email subject (e.g., "Archive Invoice | org:uuid-abc123") or attachment filename. Daemon must parse

subject/filename to apply Paperless tags. Parsing errors = wrong tenant tag.
- **Dedup complexity.** If Bilko retries email send (network timeout), daemon sees 2 emails with same attachment. Must SHA256 attachments and dedupe in Paperless query before upload.

Rejection rationale: Multi-tenant scoping via email subject/filename parsing is fragile. Email attachment size limits block future use cases (e.g., scanned multi-page contracts = 50MB PDF). No idempotent retry (email duplicates on send retry).

Pattern 2 — App?Direct Paperless API (Push)

How it works: Bilko backend calls `POST https://archive.alai.no/api/documents/post_document/` directly with app-scoped CF Access service token + Paperless API token. Synchronous upload during user request.

Pros:

- **Synchronous feedback.** User uploads doc, Bilko immediately gets Paperless document ID, can display "Archived as #12345" in UI.
- **Full metadata control.** Bilko sets `correspondent`, `document_type`, `tags`, `custom_fields` in single API call. No parsing.
- **Strong dedup.** Bilko can query Paperless `GET /api/documents/?custom_fields__sha256=abc123` before upload to skip duplicates.
- **Simplest ops surface.** No worker. Bilko backend + Paperless only.

Cons:

- **Bilko must hold CF Access credentials.** New secret in Bilko backend (Secret Manager entry, rotation burden). If CF Access token leaks, attacker can access Paperless directly.
- **Paperless becomes hot-path dependency.** If archive.alai.no is down (Azure VM maintenance, network partition), Bilko document upload **fails**. User sees error: "Failed to archive invoice". Degrades UX.
- **Tight coupling.** Bilko backend must know Paperless API contract (`POST /api/documents/post_document/`, multipart/form-data with `document` + `title` + `correspondent` + `tags` fields). API change in Paperless = Bilko backend update required.
- **Cross-cloud latency in user hot path.** GCP Cloud Run (europe-north1) → Azure VM (westeurope assumed) = 20–50ms network RTT + Paperless processing ~200ms = 250ms added to user upload response time.
- **No retry buffer.** If Paperless returns 500, Bilko must decide: fail user request, or queue retry internally (adds complexity).

Rejection rationale: Paperless availability becomes Bilko UX blocker. User uploads signed contract, archive.alai.no is down, user sees "Upload failed" even though contract PDF saved to R2. Unacceptable UX degradation for external dependency. Cross-cloud latency (250ms) in hot path for

low-value sync feedback.

Pattern 3 — App?Shared Blob?Archiver Job (Batch) [RECOMMENDED]

How it works: Bilko writes document to **Cloudflare R2 bucket** (`alai-bilko-archive-queue/` prefix or separate bucket) with metadata:

```
{
  "organizationId": "uuid-abc123",
  "organizationName": "Firma AS",
  "documentType": "invoice",
  "invoiceNumber": "2024-001",
  "timestamp": "2026-05-08T10:30:00Z",
  "sha256": "abc123...def"
}
```

Separate **Cloud Run job** (cron every 5 minutes, or Cloud Tasks queue) reads R2 objects, uploads to Paperless via `POST /api/documents/post_document/` with:

- `correspondent` = "Firma AS (uuid-abc123)"
- `document_type` = "Invoice"
- `tags` = "org:uuid-abc123,invoice,bilko"
- `custom_fields` = `{ "sha256": "abc123", "invoiceNumber": "2024-001", "uploadedAt": "2026-05-08T10:30:00Z" }`

After successful upload, worker **deletes R2 object** (or moves to `archived/` prefix). On failure, object remains, retry on next cron run.

Pros:

- **Full decoupling.** Bilko writes to R2 (fire-and-forget, <50ms). Worker handles Paperless upload async. Bilko unaware of Paperless downtime.
- **Idempotent retry.** R2 object key = `{organizationId}/{documentType}/{sha256}.pdf`. Duplicate upload (network retry) = same key, R2 overwrites. Worker can query Paperless `custom_fields__sha256` before upload to skip duplicates.
- **Multi-tenant tagging trivial.** Worker reads `organizationId` from R2 metadata → applies `tags=org:{organizationId}` in Paperless. No parsing, no guessing.
- **Scalable.** R2 = unlimited objects. Worker can batch-process 1000+ docs/run if needed. Paperless bulk upload API available.
- **Platform-agnostic.** R2 is S3-compatible. Worker can run on GCP Cloud Run, Azure Container Apps, AWS Lambda, Cloudflare Workers (D1 queue). No vendor lock-in.

- **Future-proof.** Add OneDrive archival target? Worker fans out to Paperless + OneDrive + S3 Glacier. Bilko unchanged.
- **Audit trail in R2.** If worker crashes mid-upload, R2 object = source of truth. Re-run = idempotent.

Cons:

- **Eventual consistency.** User uploads doc at 10:00, worker cron runs at 10:05 → doc visible in Paperless at 10:05:30. 5.5min delay.
- **Additional ops component.** Worker must be deployed, monitored (cron health check via Cloud Monitoring uptime check, alert on 3 consecutive failures).
- **Dev velocity slower.** Must scaffold worker (Cloud Run job + cloudbuild-worker.yaml + Terraform module), deploy pipeline, monitoring dashboard.
- **R2 becomes queue.** If worker stops (VM crash, deployment), R2 accumulates unprocessed docs. Must monitor queue depth (R2 ListObjectsV2 count).

Why this pattern wins:

1. **Bilko UX never degrades.** Paperless down? User still uploads doc successfully (R2 write). Worker retries until Paperless recovers. 2. **Multi-tenant isolation enforced structurally.** Worker applies `org:{uuid}` tag from R2 metadata. No chance of cross-tenant leak (Paperless search by tag = instant tenant filter). 3. **Scales to 10,000 orgs × 100 docs/day.** R2 = unlimited storage, worker processes batch (100 docs/run = 6 seconds at 60ms/doc). 4. **Idempotent by design.** R2 object key = content hash. Worker crash mid-upload? Re-run processes same doc, Paperless dedupes via `custom_fields.sha256`. 5. **Reuses existing Bilko infrastructure.** R2 bucket already configured (BUILD-BLUEPRINT line 64). Worker = new Cloud Run service (Terraform module = 20 lines).

Implementation complexity accepted because:

- Bilko is a **B2B SaaS** with multi-tenant data sovereignty requirements. Eventual consistency (5min delay) is acceptable for archival. Real-time feedback ("Archived as #12345") is nice-to-have, not must-have.
- Pattern 2 (direct API) makes Paperless a **hot-path dependency** → UX risk unacceptable.
- Pattern 1 (email) has **multi-tenant scoping fragility** (parsing subject lines) + attachment size limits (30MB SendGrid).

Implementation Spec (High-Level)

Phase 1: Bilko Backend Changes (CodeCraft)

1. **Add R2 archive write function** in `apps/api/src/main/kotlin/no/alai/bilko/services/ArchiveService.kt`:

```
suspend fun archiveDocument(
    organizationId: UUID,
    organizationName: String,
    documentType: String,  // "invoice" | "contract" | "care_plan"
    documentBuffer: ByteArray,
    metadata: Map  // { "invoiceNumber": "2024-001", ... }
): String {
    val sha256 = documentBuffer.sha256()
    val objectKey = "archive-queue/${organizationId}/${documentType}/${sha256}.pdf"
```

s3Client.putObject(bucket = "alai-bilko-files", key = objectKey, body = documentBuffer, metadata = mapOf("organizationId" to organizationId.toString(), "organizationName" to organizationName, "documentType" to documentType, "timestamp" to Instant.now().toString(), "sha256" to sha256) + metadata)

return objectKey }

2. **Call `archiveDocument()` after invoice PDF generation** in `InvoiceService.generatePDF()`:

```
val pdfBuffer = pdfGenerator.generate(invoice)
s3Client.putObject(...)  // existing code
archiveService.archiveDocument(
    organizationId = invoice.organizationId,
    organizationName = organization.name,
    documentType = "invoice",
    documentBuffer = pdfBuffer,
    metadata = mapOf("invoiceNumber" to invoice.number)
)
```

3. **Same pattern for contracts, care plans, onboarding docs.**

Phase 2: Archiver Worker (CodeCraft + FlowForge)

1. **New Cloud Run service** `bilko-archiver-worker` (Kotlin/Ktor or Node.js, TBD):

```
// apps/archiver-worker/src/main/kotlin/no/alai/bilko/archiver/Main.kt
```

fun main() { val s3Client = S3Client(/* R2 config */) val paperlessClient = PaperlessClient(baseUrl = "https://archive.alai.no", cfAccessClientId = System.getenv("CF_ACCESS_CLIENT_ID"), cfAccessClientSecret = System.getenv("CF_ACCESS_CLIENT_SECRET"), apiToken = System.getenv("PAPERLESS_API_TOKEN"))

runBlocking { val objects = s3Client.listObjectsV2("alai-bilko-files", prefix = "archive-queue/") objects.forEach { obj -> try { val metadata = obj.metadata val documentBuffer = s3Client.getObject(obj.key)

// Check if already uploaded (dedup) val existing = paperlessClient.searchBySHA256(metadata["sha256"]!!) if (existing != null) { logger.info("Document ${obj.key} already archived as Paperless #${existing.id}, skipping") s3Client.deleteObject(obj.key) return@forEach }

// Upload to Paperless val paperlessDoc = paperlessClient.uploadDocument(document = documentBuffer, title = "${metadata["documentType"]} - ${metadata["organizationName"]}", correspondent = "${metadata["organizationName"]} (${metadata["organizationId"]})", documentType = metadata["documentType"]!!.capitalize(), tags = listOf("org:${metadata["organizationId"]}", metadata["documentType"]!!, "bilko"), customFields = mapOf("sha256" to metadata["sha256"]!!, "uploadedAt" to metadata["timestamp"]!!, "organizationId" to metadata["organizationId"]!!))

logger.info("Archived ${obj.key} → Paperless #${paperlessDoc.id}") s3Client.deleteObject(obj.key)

} catch (e: Exception) { logger.error("Failed to archive ${obj.key}: ${e.message}", e) // Leave object in R2, retry on next run } } } }

2. **Deploy as Cloud Run job** (triggered by Cloud Scheduler every 5 minutes):

infrastructure/gcp/terraform/modules/archiver-worker/main.tf

resource "google_cloud_run_v2_job" "bilko_archiver_worker" { name = "bilko-archiver-worker" location = var.region

template { template { containers { image = "europe-north1-docker.pkg.dev/${var.project_id}/bilko/archiver-worker:latest"

env { name = "CF_ACCESS_CLIENT_ID" value_source { secret_key_ref { secret = "cf-access-client-id" version = "latest" } } } env { name = "CF_ACCESS_CLIENT_SECRET" value_source { secret_key_ref { secret = "cf-access-client-secret" version = "latest" } } } env { name = "PAPERLESS_API_TOKEN" value_source { secret_key_ref { secret = "paperless-api-token" version =

"latest" } } } }

timeout = "600s" # 10min max } } }

resource "google_cloud_scheduler_job" "archiver_trigger" { name = "bilko-archiver-cron" schedule = "*/5 * * * *" # Every 5 minutes time_zone = "Europe/Oslo"

http_target { uri = "https://${var.region}-run.googleapis.com/apis/run.googleapis.com/v1/namespaces/${var.project_id}/jobs/${google_cloud_run_v2_job.bilko_archiver_worker.name}:run" http_method = "POST"

oauth_token { service_account_email = google_service_account.archiver_worker.email } } }

3. **Monitoring dashboard** (Cloud Monitoring): - Queue depth (R2 objects in `archive-queue/` prefix) — alert if >500 - Worker success rate — alert if <95% over 1h - Worker execution time — alert if >300s - Paperless API error rate — alert if >5% over 15min

Phase 3: Paperless-ngx Configuration (FlowForge + Proveo)

1. **Create Paperless correspondents** (one per Bilko org, OR dynamic via worker): - Option A: Worker auto-creates correspondent if not exists (`POST /api/correspondents/` with name="Firma AS (uuid-abc123)"). - Option B: Manual setup (CEO creates correspondent in Paperless UI for each new Bilko customer). **Recommend Option A** for scalability.

2. **Create Paperless document types**: - Invoice - Contract - Care Plan - Onboarding Document - Incident Report

3. **Create Paperless custom fields**: - `sha256` (text, unique identifier for dedup) - `organizationId` (text, Bilko tenant UUID) - `uploadedAt` (datetime, original upload timestamp) - `invoiceNumber` (text, optional) - `contractId` (text, optional)

4. **Tag taxonomy**: - `org:{uuid}` (one tag per Bilko tenant, e.g., `org:abc-123-def`) - `invoice` | `contract` | `care-plan` | `onboarding` | `incident` - `bilko` (source system tag)

Phase 4: Retention Policy (Dr. Sarah Chen — Healthcare Compliance)

Question for CEO:

1. **How long to keep docs in R2 after successful Paperless upload?** - Option A: Delete immediately (worker deletes R2 object after Paperless confirms upload). - Option B: Keep 30 days (R2 lifecycle policy auto-deletes after 30d). Allows re-upload if Paperless doc accidentally deleted. - **Recommendation:** Option A (immediate delete). Paperless is source of truth post-archival. R2 =

queue only.

2. **Paperless retention policy?** - Invoices: 7 years (Norway Bokføringsloven, Serbia/Croatia equivalent) - Contracts: Indefinite (until contract expires + 5 years) - Care plans: 10 years (HIPAA if US expansion, GDPR Article 17 deletion rights) - **Recommendation:** Configure per-document-type in Paperless via workflow rules (out of scope for this ADR).

3. **GDPR Article 17 (Right to Erasure) handling?** - When Bilko org deletes account (GDPR erasure request), worker must: 1. Query Paperless `GET /api/documents/?tags__name=org:{uuid}` 2. Delete all matching docs `DELETE /api/documents/{id}/` 3. Delete correspondent `DELETE /api/correspondents/{id}/` - **Recommendation:** Separate MC for GDPR compliance (erasure worker). Out of scope for archival MVP.

Stakeholders

- **CEO (Alem Basic):** Final approval on pattern choice + retention policy decisions.
- **CodeCraft (Petter Graff, Hadi Hariri):** Bilko backend changes + archiver worker implementation.
- **FlowForge (Kelsey Hightower):** GCP Cloud Run job + Cloud Scheduler + Terraform IaC.
- **Proveo (Angie Jones):** End-to-end validation (upload invoice in Bilko → verify appears in Paperless with correct tags/metadata).
- **Dr. Sarah Chen (Healthcare Compliance):** HIPAA/GDPR retention policy review if Bilko expands to care plan archival.
- **Skillforge:** BookStack runbook page for archiver worker (operational playbook, troubleshooting).

Open Questions for CEO

1. **Worker cron interval:** 5 minutes (recommended) vs 15 minutes (lower Cloud Run invocation cost)? - 5min = faster archival, users see docs in Paperless <6min after upload. - 15min = lower cost (~$0.50/month vs ~$1.50/month for Cloud Run invocations), acceptable delay for archival use case. - **Awaiting CEO decision.**

2. **R2 retention after upload:** Delete immediately (recommended) vs keep 30 days (safety buffer)? - Immediate = lower storage cost, cleaner queue. - 30 days = allows re-upload if Paperless doc accidentally deleted (rare edge case). - **Awaiting CEO decision.**

3. **Multi-tenant correspondent strategy in Paperless:** - Option A: One correspondent per Bilko org (e.g., "Firma AS (uuid-abc123)"). Pro: clean correspondent filter in Paperless UI. Con: 10,000

orgs = 10,000 correspondents (Paperless UI clutter). - Option B: Single correspondent "Bilko" + rely on `org:{uuid}` tags for tenant isolation. Pro: clean Paperless correspondent list. Con: must always filter by tag (cannot filter by correspondent alone). - **Recommendation:** Option A (one correspondent per org). Paperless search by correspondent is more intuitive than tag filter for non-technical users (CEO searching for customer docs). - **Awaiting CEO decision.**

References

- **MC #100025** — This task (pattern decision + ADR)
- **MC #100004** — IMAP→Paperless pipe (operational, BookStack #2862)
- **BUILD-BLUEPRINT.md** — Bilko tech stack, multi-tenancy model, R2 config (lines 64, 192–193)
- **Paperless-ngx API docs** — https://docs.paperless-ngx.com/api/
- **Cloudflare R2 docs** — https://developers.cloudflare.com/r2/api/s3/api/
- **GCP Cloud Run jobs** — https://cloud.google.com/run/docs/create-jobs
- **ADR-020** — Bilko backend canonical path (`apps/api/`)
- **ADR-021** — Bilko blueprint realignment (Kotlin/Ktor sole backend)

Next Steps (Child MCs)

Upon CEO approval of Pattern 3:

1. **MC #TBD (CodeCraft):** Implement `ArchiveService.kt` in Bilko backend + call from `InvoiceService.generatePDF()`. **Estimate:** 2h. **Priority:** M. 2. **MC #TBD (CodeCraft):** Scaffold archiver worker (`apps/archiver-worker/`) with R2→Paperless upload logic + dedup via SHA256. **Estimate:** 4h. **Priority:** M. 3. **MC #TBD (FlowForge):** Deploy archiver worker as Cloud Run job + Cloud Scheduler cron (Terraform IaC). **Estimate:** 3h. **Priority:** M. 4. **MC #TBD (FlowForge):** Provision CF Access service token for archiver worker + store in Secret Manager. **Estimate:** 1h. **Priority:** M. 5. **MC #TBD (Proveo):** End-to-end validation — upload test invoice in Bilko stage, verify appears in Paperless with `org:{uuid}` tag + correspondent. **Estimate:** 2h. **Priority:** M. 6. **MC #TBD (Skillforge):** BookStack runbook page for archiver worker (troubleshooting, monitoring dashboard links, manual queue drain). **Estimate:** 1h. **Priority:** L.

Total estimate: 13h across 3 specialists (CodeCraft 6h, FlowForge 4h, Proveo 2h, Skillforge 1h).

Decision Status: Awaiting CEO approval on:

1. Pattern 3 acceptance (vs Pattern 1 or 2) 2. Worker cron interval (5min vs 15min) 3. R2 retention policy (immediate delete vs 30d) 4. Paperless correspondent strategy (one-per-org vs single "Bilko" correspondent)

Next action: CEO review → approve → create 6 child MCs → dispatch to CodeCraft/FlowForge/Proveo/Skillforge.

Revision #4
Created 2026-05-08 19:29:45 UTC by John
Updated 2026-07-19 20:00:56 UTC by John