File Thunder — Architecture & Design Guide V2

NexGate Media Engine | Version 2.0

> v2 integrates the media-security refinements worked out in design review. Every change from v1 is tagged inline as **[v2 FIX]** so it can be reviewed individually. Section 0 lists all of them in one place — read that first.

---

0. What changed in v2 (read this first)

These are the careful corrections. Each is explained in full in its section.

1. **Signing is three tiers, not two.** "Public = unsigned, private = signed" was wrong. Correct model: trivial public assets unsigned; *public content* (full images/videos) **signed even though public**; private/paid signed + auth. → §8
2. **Public content gets signed too.** A signed URL is anti-scrape / anti-hotlink, *not* only access control. TikTok signs even public videos. → §8, §14
3. **The visibility-bypass hole must be closed.** In v1, media URLs were permanent and unsigned, so a PRIVATE post's bytes were reachable by anyone with the link — the post was hidden by the API but the file was not. Fixed by locking the bucket (no public-read) and routing everything through signed CDN URLs. → §14
4. **Backend signs, CDN validates, they never talk at sign time.** Signing is pure math on `(path + expiry + secret)`. No file, no CDN contact. CDN fetches from MinIO lazily on first view. → §9
5. **Domain is `media.nexgate.co`, not `.com`.** v1 mixed both. One domain everywhere. → §10
6. **Object keys must be owner-correct.** v1 had one user's avatar stored under another user's prefix. The owner segment in the key must match the real owner. → §3
7. **Store the `objectKey`, never the signed URL.** Signed URLs are minted per request and thrown away. Never persisted in the DB or feed cache. → §13, §22
8. **MP4 and HLS use different token strategies.** Short MP4 = one signed URL. Long HLS = dual-token (short manifest + long segments) or prefix-signing, so playback never dies mid-video. → §11
9. **Single-use is for downloads only, never streams.** A stream is many requests; single-use would break playback. Streams use short expiry + user-binding instead. → §11, §12

10. **Watermarked file is NEVER in the feed response.** Feed = clean playback URLs only. Watermarked download comes from a *separate* endpoint, minted on tap. → §12, §13
11. **Cache gotcha: never cache a signed object keyed by its signature.** A cached object can outlive its stamp; private/paid content must bypass shared cache. → §10, §11
12. **Bot gate is the missing half of anti-scraping.** Signed URLs stop *permanent* theft; the bot gate stops the *act* of mass-pulling. v1 had no bot gate section. → §14
13. **Open decision parked: NextGate's moat (content vs commerce)** drives how hard to stamp public media and how heavy the bot gate needs to be. → §30

---

1. What is File Thunder?

File Thunder is NexGate's dedicated media processing engine — a standalone Spring Boot microservice responsible for all file and media operations across the platform.

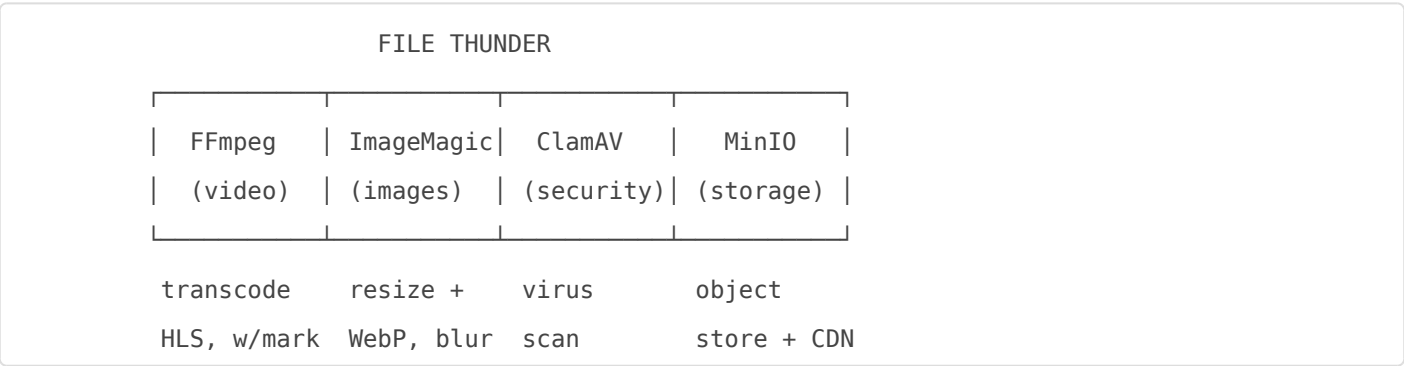
Core principle: The Main Backend never touches raw files. It delegates all media operations to File Thunder and acts as a thin client.

Responsibilities: receive uploads, validate (type/size/quota), virus scan, transcode, generate variants (thumbnails, WebP, HLS, MP4), watermark, store in MinIO, serve via CDN, mint signed delivery URLs, publish media-ready events.

Not: AI/ML, recommendation, social graph, payments, auth, business logic.

---

2. The Four Wheels



- **FFmpeg (Jaffree)** — transcode, HLS segmenting, thumbnail/best-frame, 3s preview, watermark overlay, blur-pad, fMP4, format conversion, audio extract/normalize.
- **ImageMagick (IM4Java)** — resize variants, format→WebP, EXIF strip (privacy), auto-orient, dominant color, LQIP, OG image, GIF→WebP, PDF→preview. Plus `blurhash-java`.
- **ClamAV** — virus scan every upload (TCP `clamav:3310`), hash-cache, digital products scanned twice.

- **MinIO** — S3-compatible storage, presigned direct uploads, origin for Cloudflare, multipart + byte-range, lifecycle policies. **Never publicly exposed** (only Cloudflare + File Thunder reach it).

---

3. Storage Architecture (MinIO Buckets)

Rule: 4 buckets total. Never one bucket per user.

```
nexgate-raw/       ← temporary upload landing (auto-delete 24h)
nexgate-public/    ← social content (avatars, posts, stories, events, shops)
nexgate-private/   ← DMs, documents, KYC, temp audio
nexgate-digital/   ← purchase-gated products
```

Object Key Pattern

```
{bucket}/{domain}/{ownerId}/{entityId}/{fileId}/{variant}


nexgate-public/posts/usr_123/post_456/file_789/720p_clean.mp4
nexgate-private/messages/conv_abc/file_789/original.jpg
nexgate-digital/products/shop_xyz/prod_123/file_789/original/file.pdf
```

> **[v2 FIX] Owner-correct keys.** The `{ownerId}` segment MUST be the real owner of the file. v1 had cases where one user's avatar lived under a different user's prefix — that leaks identity and misattributes ownership. The owner in the key = the owner in the DB. Keys are otherwise opaque (no usernames, no PII).

nexgate-public layout (unchanged from v1)

```
profiles/{userId}/{fileId}/  avatar_400.webp, avatar_150.webp, avatar_50.webp, cover.webp
posts/{userId}/{fileId}/     {360,720,1080}p_clean.mp4, {360,720,1080}p_watermarked.mp4,
                             preview_3s.mp4, hls/master.m3u8, hls/{360,720,1080}p/seg_*.ts,
                             thumbnail.webp, og_clean.webp, og_play.webp, og_preview.mp4
stories/{userId}/{fileId}/   720p_clean.mp4, thumbnail.webp
events/{accountId}/{eventId}/{fileId}/  banner.webp, banner_mobile.webp, banner_thumb.webp
shops/{shopId}/{productId}/{fileId}/    large.webp, medium.webp, thumb.webp
```

(See §12 for the watermarked-variant storage decision — store one rung, not all three.)

nexgate-private / nexgate-digital

As v1: messages, documents, kyc, temp audio (private); products with `original/` (never exposed) + `preview/` + `cover/` (digital).

---

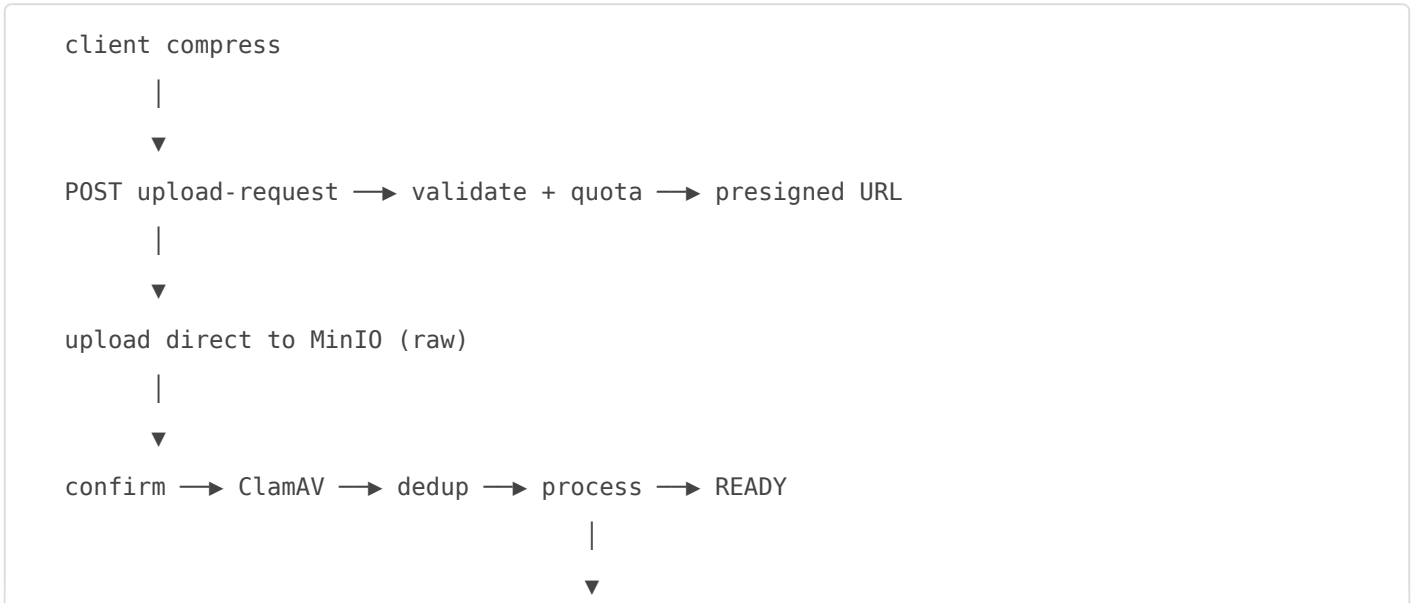
4. Accepted File Formats

- **Images** accept JPEG/PNG/HEIC/WebP/GIF; reject BMP/TIFF/RAW/SVG (SVG = security risk). Output always **WebP**.
- **Videos** accept MP4/MOV/MKV/WEBM/AVI/3GP; reject WMV/FLV/VOB. Output **H.264 MP4** (short) or **HLS H.264** (long).
- **Digital** PDF, DOCX/XLSX/PPTX, GLB/OBJ/FBX/STL, MP3/WAV/FLAC, ZIP/RAR, PSD/AI, PNG.

---

5. Upload Flow

```
[1] Client-side intelligent compression (max 1080p, CRF 18 light, HEIC→JPEG)
[2] POST /media/upload-request { fileName, fileSize, mimeType, directory, clientMeta }
[3] File Thunder: validate MIME + size + quota (atomic SQL) → DB record PENDING
    → presigned MinIO URL (nexgate-raw) → { fileId, uploadUrl, expiresIn: 1800 }
[4] Client uploads directly to MinIO (TUS resumable / multipart >5MB)
[5] Client confirms POST /media/confirm { fileId }   (cleanup job recovers if missed)
[6] Processing pipeline starts
```



```
RabbitMQ media.ready ──► Main Backend
```

Quota checked at presigned-URL time (atomic `UPDATE ... WHERE used+size <= quota`). **Duration** checked AFTER upload via FFprobe (client value is a hint, not trusted) — over limit → delete from raw, release quota, 403 `DURATION_EXCEEDED`.

Per-Upload Limits

Plan	Image	Short Video	Long Video	Digital Video	Duration
FREE	20MB	200MB	□	□	5 min
PRO	20MB	500MB	2GB	5GB	60 min
BUSINESS	20MB	2GB	5GB	20GB	Unlimited

---

6. Processing Pipelines (summary)

All pipelines: **ClamAV scan → SHA-256 dedup check → process → store → delete raw → DB READY → publish events.** Watermarking and all placeholders are generated **here, at processing time**, never at serve time.

6.1 Image

Auto-orient → strip EXIF (privacy) → NSFW check → dominant color + LQIP + BlurHash → size variants per content type (Post: 1600/800/300; Profile: 400/150/50; Product: 1000/500/200 square; Event: 1200×630/800×420) → WebP → OG image → store public → READY.

6.2 Short Video (MP4, duration < 3 min)

FFprobe → adaptive transcode decision (never upscale, never inflate) → **fast lane** (quick 360p → `LIVE_PARTIAL`) → full processing:

- **Clean** variants `{360,720,1080}p_clean.mp4` (faststart). Non-9:16 reel-eligible → **blur-pad** to 9:16 (scale-fill+blur background, scale-fit sharp foreground, overlay).
- **Watermarked** variant(s) — **moving** watermark (jumps corners every 3s; 80×80 logo 60% opacity + username; app-based label from `clientApp`). **[v2 FIX] store one download rung (720p), not all three** (§12).
- Extras: `thumbnail.webp` (best-frame scored) + its LQIP/BlurHash/dominantColor, `preview_3s.mp4` (360p muted watermarked), `og_clean.webp`, `og_play.webp`, `og_preview.mp4`.
- fMP4 (`frag_keyframe+empty_moov+faststart`) for byte-range.
- Extract `audio.wav` → nexgate-private (Rec Engine).

- READY: `isReelEligible` (<3min), `streamingFormat: MP4`, variants JSONB, aspectRatio, qualityScore, duration.

6.3 Long Video (HLS, duration ? 3 min)

Same as short EXCEPT: HLS adaptive (no fast lane — stays PROCESSING until done). `hls/{360,720,1080}p/` segments + `master.m3u8`. Segment = 2s, H.264 + AAC, MPEG-TS. Still makes preview_3s, thumbnail, placeholders, OG. `isReelEligible: false`, `streamingFormat: HLS`.

6.4 Digital Products

Double ClamAV scan → SHA-256 checksum → light preview by type (PDF p1 watermarked, video first 2min 480p watermarked, audio 30s 96kbps, 3D multi-angle thumbs, image 600px watermarked) → store `original/` (never exposed) + `preview/` + `cover/` → READY.

6.5 DM Attachments

nexgate-private, **no CDN**, conversation-membership check, thumb+original only, no watermark, no reel pool, signed 5-min URLs. Virus scanned like everything.

---

7. Watermarking Strategy

- **Applied at processing time, once.** Stored as separate `_watermarked` variant. Never at serve time, never client-side.
- **Moving watermark** (corners every 3s) — static is croppable, moving is not.
- **Served:** in-app playback → **clean** variant; download button → **watermarked** variant (§12). Direct CDN URL → clean (unavoidable; accepted, same tradeoff every platform has).
- Pre-watermarked uploads (from other platforms): Phase 1 = no detection, just overlay ours. Phase 3 = AI logo detection → suppress reach (Instagram model), never block.
- Username change: old videos keep old username (like TikTok).

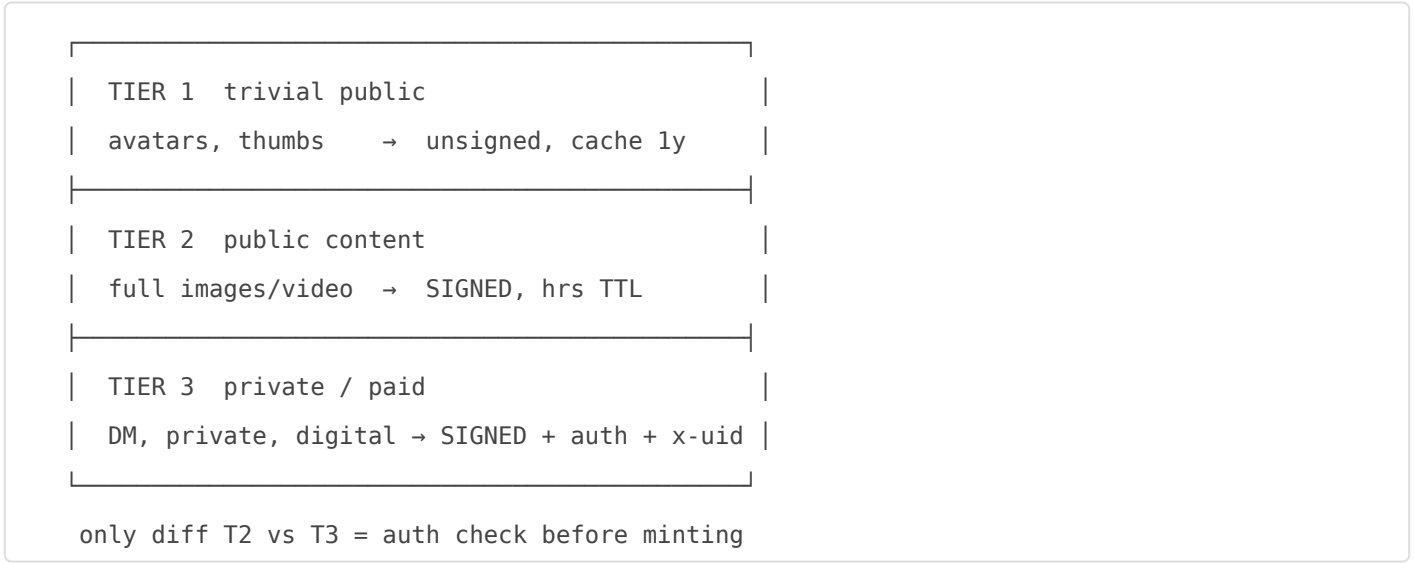
> **[v2 FIX] Honest scope.** A watermark does NOT prevent download — it survives download. Its job is attribution + traceability, not prevention. The clean file is reachable by a determined ripper via the playback URL; that's accepted. Goal = "not worth the effort for 99% + traceable if they do."

---

8. The Three Delivery Tiers **[v2 FIX — core correction]**

v1 assumed two levels (public unsigned / private signed). Correct model is **three**:

Tier	Content	Login to view?	URL signed?	Expiry	CDN cache
1 — trivial public	avatars, thumbnails, posters, blur previews	no	**no**	n/a	aggressive, 1y
2 — public content	full post images, video rungs	no	**YES**	hours	careful (§10)
3 — private / paid	DM media, private posts, digital products, KYC	**yes**	YES	minutes	none / private



The key insight: **"public to view" and "signed URL" are independent.** Tier 2 content is open to watch by anyone (no login), but the *file* is still signed — because signing is **anti-scrape / anti-hotlink**, not access control. This is exactly TikTok's model (public video, yet the raw file link expires and 403s).

The only difference between tier 2 and tier 3 in code is **whether an auth/visibility check runs before minting** — the signing itself is identical.

> Tier 1 is the only unsigned tier. Anything a scraper would actually *want* (full media) is tier 2+, i.e. signed.

9. Signed URLs — how they actually work **[v2 — new, the mechanism]**

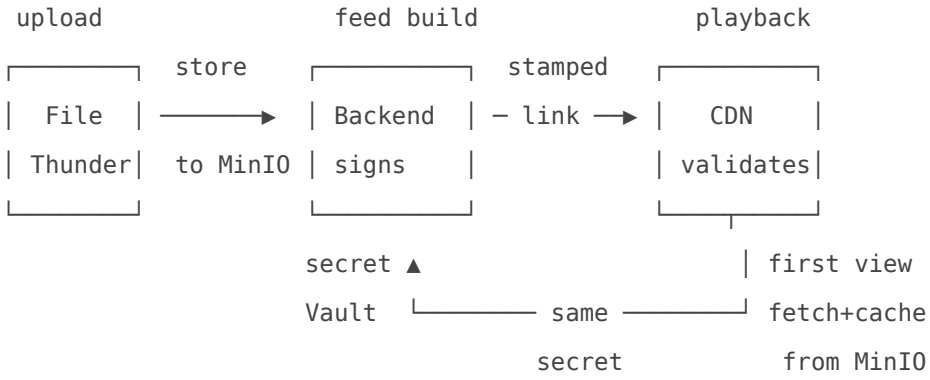
A signed URL = a normal link with a **stamp** made from three things:

```
stamp (x-sig) = HMAC(secret, path + expiry)
```

- `path` — the object key being requested
- `expiry` (`x-expires`) — unix time the link dies
- `secret` — a key known ONLY to File Thunder/backend AND the CDN. Never in the URL, browser, or JSON. Lives in Vault.

Who does what, and when — three separate moments:

1. **Upload time:** File Thunder makes the variants, stores them in MinIO. CDN not involved.
2. **Feed-build time (signing):** backend computes `x-sig` from `(path + expiry + secret)`. **Pure string math — no file opened, no CDN contacted, no storage touched.** It's just writing a stamped link. (You can sign a path whose bytes don't even exist yet — the math only sees the string.)
3. **Playback time (serving):** the phone requests the stamped URL → CDN re-computes the stamp with its copy of the secret. Match + not expired → serve. First viewer: CDN lazily fetches the file from MinIO and caches it. Later viewers: served from cache.



Tamper resistance (why a stripped/edited link fails):

- No sig → CDN rejects (paths require a valid sig) → 403.
- Fake sig → recompute doesn't match → 403.
- Edited expiry → sig was computed *from* the old expiry, no longer matches → 403.
- Swapped path → sig bound to original path → 403.
- Untouched + in time → serve. After `x-expires` → even the real link 403s.

Forging anything requires the **secret**, which never leaves backend + CDN. Protect the secret (Vault) and the whole scheme holds.

You don't write the HMAC. The MinIO SDK (or CDN signer) does it in one call:

```
// ── Mint a signed delivery URL ──
String url = minioClient.getPresignedObjectUrl(
    GetPresignedObjectUrlArgs.builder()
        .method(Method.GET)
        .bucket(bucket)
        .object(objectKey)
        .expiry(seconds, TimeUnit.SECONDS)   // → x-expires + x-sig
        .build());
```

---

10. URL Strategy

Tier 1 — trivial public ? permanent, unsigned, cache hard

```
https://media.nexgate.co/profiles/usr_123/avatar_400.webp
https://media.nexgate.co/posts/usr_123/vid_789/thumbnail.webp
```

`Cache-Control: public, max-age=31536000`. Scraping these gains nothing.

Tier 2 — public content ? signed, short-ish expiry

```
https://media.nexgate.co/posts/usr_123/vid_789/720p_clean.mp4?x-expires=1780297200&x-sig=8f3a9c
```

Tier 3 — private / paid ? signed + user-bound + auth check first

```
https://media.nexgate.co/private/messages/conv_abc/file_789/original.jpg
   ?x-expires=...&x-sig=...&x-uid=usr_xyz
```

> **[v2 FIX] Domain is `.co`.** All URLs use `media.nexgate.co`. v1 mixed `.com`/`.co` — that produces broken links. One domain everywhere.

> **[v2 FIX] Cache-key gotcha.** Never let the CDN cache a signed object using a cache key that includes `x-sig`/`x-uid` — you'd fragment cache per user, and a cached object can outlive its stamp (origin checks expiry, but a cache hit never reaches origin). Rules: tier 1 cache hard; tier 2 cache by **path only** (strip query from cache key), short edge TTL; tier 3 **never** on shared cache.

Signed URL params

Param	Purpose
`x-expires`	link dies after TTL
`x-sig`	HMAC tamper protection
`x-uid`	bound to user (tier 3) — shared link won't work for someone else

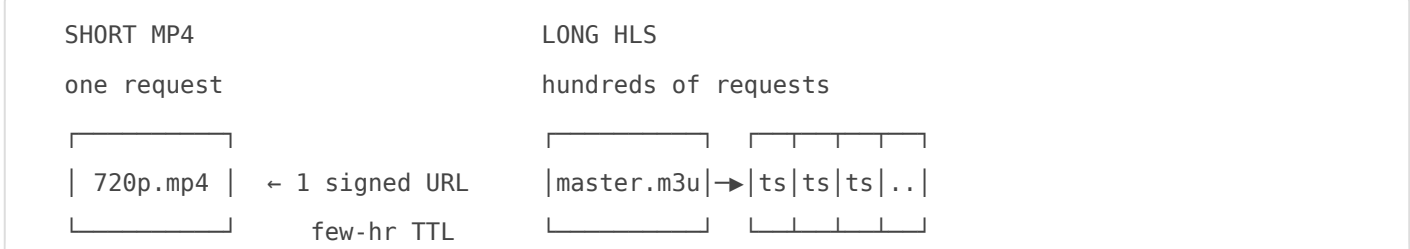
Who generates / validates

Backend (has secret) **generates**. Cloudflare edge (same secret) **validates**. MinIO never seen by clients.

---

11. Streaming Token Strategy: MP4 vs HLS **[v2 — new]**

Expiry is checked **when each request opens**, not continuously. An in-flight transfer completes even if the clock passes mid-download. The two formats differ because of how many requests a single playback makes.



```
opens once,                    short token     long token
rides to end                   (~1 min)        (> video len)
                               per-session      one, shared
```

Short video (MP4) — one signed URL

One file = one request (or a few byte-ranges, all opened up front). A few-hour expiry is plenty; the in-flight request completes, so mid-play expiry basically never happens. → mint one signed URL per rung, few-hour TTL. **No dual-token, no single-use.**

Long video (HLS) — dual-token (or prefix-signed)

HLS = hundreds of segment requests (one every ~2s), so expiry is re-checked constantly. A single short token would die mid-video. Industry-standard fix (Google Media CDN / CloudFront):

- `master.m3u8` → **short** token (~1 min). Per-session entry gate. Fetched once at start.
- **child manifests +** `.ts` **segments** → **long** token (> full video length, up to ~1 day), carried by **signed cookie** or **path-prefix signing** (sign the folder, not each segment).
- Segments stay **cacheable** (same bytes for everyone); only the manifest is per-session. Preserves cache hit rate.

> At our stage on Cloudflare Free/Pro, do the simpler version: backend mints a **long prefix-signed** token for the whole `hls/` folder when the post loads. Graduate to true edge dual-token (short→long exchange) if/when we move to a heavier CDN.

Single-use — downloads ONLY, never streams

A stream is many requests; making the token single-use would 403 the viewer one segment in. **Streams use short expiry +** `x-uid` **binding instead.** Single-use (Redis) is correct only for one-file-one-request **downloads** (§12, digital products).

Edge case (both formats)

User pauses past expiry, then seeks → player fires a fresh request with a dead URL → 403. **Client safety net:** catch the 403, silently re-fetch a fresh URL from backend, resume.

	Short MP4	Long HLS

Requests / play	one (few byte-ranges)	hundreds
Token	single signed URL	dual-token / prefix-signed
TTL	few hours	> watch duration
Single-use	no	no

---

12. Watermark & Download Path **[v2 FIX]**

Decision: **Option B — downloads are watermarked.** (Full build, not the launch shortcut.)

The feed NEVER contains a watermarked URL. Feed = clean playback URLs only. Watermarked download is a **separate endpoint**, minted on tap.

```
User taps Download
  → GET /media/{fileId}/download?quality=720p   (quality optional, default 720p)
  → backend: check canDownload (post setting) + tier (private/paid → auth check)
  → mint signed URL → {quality}_watermarked.mp4
      short TTL (~10 min), x-uid bound, single-use (Redis x-once)
  → log download event (orderId/buyerId/ip/device/country if digital)
  → return { downloadUrl, expiresIn, singleUse: true }
Client downloads. URL marked used in Redis → 403 on reuse.
```

Default quality, no picker (short video). Serve one default rung (**720p**) — matches TikTok/IG, sane on TZ data. Endpoint takes optional `quality` so a picker is an additive change later (long-form only, where the size gap matters).

Storage decision: store **one** watermarked rung (720p), not all three. If a picker is added later, **lazy-generate** other rungs on first request and cache — only spend CPU/storage on a quality someone actually downloads.

Timing: watermarked 720p is generated at **processing time** (pre-ready), so download is instant — no first-user FFmpeg penalty.

```
// ── Download endpoint (picker-ready, single-use, watermarked) ──
@GetMapping("/media/{fileId}/download")
ResponseEntity<?> download(@PathVariable String fileId,
                           @RequestParam(defaultValue = "720p") String quality,
                           @AuthenticationPrincipal AccountEntity user) {
```

```
    MediaEntity m = mediaService.findReady(fileId);

    // ── access gate (tier 2 = open; tier 3 = ownership/visibility) ──
    if (!m.isCanDownload())                 return ResponseEntity.status(403).build();
    if (m.isRestricted() && !accessService.canAccess(user, m))
                                            return ResponseEntity.status(403).build();

    // ── mint watermarked, user-bound, single-use URL ──
    String key = m.watermarkedKey(quality);                 // lazy-generate if absent
    String url = mediaService.mintDownloadUrl(key, user.getId(), Duration.ofMinutes(10));
    downloadLog.record(fileId, user, request);              // audit
    return ResponseEntity.ok(Map.of("downloadUrl", url, "expiresIn", 600, "singleUse", true));
}
```

---

13. Feed Response Shape **[v2 — new]**

Rules baked in:

- **Batch-mint inline.** While building the page (already looping posts), mint every signed URL right there — no per-item round trip from the client.
- **Store `objectKey`, mint URL per request.** Never persist a signed URL (it expires).
- **Tier 1 unsigned** (avatar, thumb, poster, preview3s); **tier 2 signed** (full image, video rungs).
- **Placeholders filled** (blurhash, lqip, dominantColor) + `width`/`height`/`duration` present.
- **No watermarked URL here** (§12). `canDownload` flag tells the client whether to show the button.


```
{
  "data": [{
    "id": "0e67aef4-...",
    "author": {
      "id": "ad03431a-...", "userName": "juliusdev_", "verified": true,
      "profilePictureUrl": "https://media.nexgate.co/profiles/ad03431a-.../avatar_400.webp"
    },
    "content": "Testing video media post □□",
    "shareCode": "waBbP82h",
    "media": [{
      "id": "436bb62b-...",
      "mediaType": "VIDEO",
```

```
      "status": "READY",
      "width": 1080, "height": 1920, "aspectRatio": "9:16", "duration": 28,
      "isReelEligible": true,
      "streamingFormat": "MP4",
      "canDownload": true,
      "dominantColor": "#1a1a2e",
      "blurhash": "L6Pj0^i_.AyE_3t7t7R**0o#DgR4",
      "lqip": "...",
      "poster":    "https://media.nexgate.co/posts/ad03431a-.../436bb62b-.../thumbnail.webp",
      "preview3s": "https://media.nexgate.co/posts/ad03431a-.../436bb62b-.../preview_3s.mp4",
      "variants": {
        "360p":  "https://media.nexgate.co/posts/ad03431a-.../436bb62b-.../360p_clean.mp4?x-
expires=1780297200&x-sig=a1f2b3",
        "720p":  "https://media.nexgate.co/posts/ad03431a-.../436bb62b-.../720p_clean.mp4?x-
expires=1780297200&x-sig=b2e3c4",
        "1080p": "https://media.nexgate.co/posts/ad03431a-.../436bb62b-.../1080p_clean.mp4?x-
expires=1780297200&x-sig=c3f4d5"
      },
      "order": 1
    }],
    "privacySettings": { "visibility": "PUBLIC" }
  }]
}
```

(HLS video: `streamingFormat: "HLS"`, replace `variants` with a single signed/prefix-signed `master.m3u8`.)

---

14. Security & Anti-Scraping **[v2 — expanded]**

14.1 Close the visibility-bypass hole

v1's permanent unsigned URLs meant a PRIVATE post's bytes were reachable by anyone with the link — the API hid the post, the file did not. **Fix:**

1. Remove **public-read** from MinIO buckets (already a stated rule — enforce it).
2. Route all access through signed CDN URLs.

3. Private/followers/DM/paid → tier 3 (auth check before minting + `x-uid`). After this, a hidden post's media returns 403 to anyone but the authorized viewer.

14.2 Two layers, both required

- **Signed URLs** stop *permanent* theft: scraped links die in hours → no permanent mirror, no hotlinking. **They do NOT stop a one-time download** — a valid link is a valid link.
- **Bot gate** stops the *act* of mass-pulling: it guards the **feed API door** so a bot can't cheaply pull fresh links over and over.
- **Neither alone is enough. Together** = scraping isn't worth it. (You cannot make it impossible — the goal is "not worth the effort for 99%, traceable if they do.")

14.3 Bot gate (the missing half) — sized to moat (§30)

- **App attestation** — the real app computes a hard-to-fake token (TikTok's `X-Bogus` / `msToken` equivalent). A plain script can't produce it without running app code.
- **Rate limiting** — one "user" pulling thousands of posts/min = bot → throttle/block.
- **Auth + behaviour** — require login; flag inhuman scroll/pull patterns.
- **Escalating friction** — suspicious traffic → CAPTCHA / slowdown / ban.

> **Build depth depends on the moat decision (§30).** Commerce-moat → lighter bot gate is fine for launch. Content-moat → invest earlier.

14.4 Protection layers (reference)

`Content-Disposition: inline` · dynamic URL via JS · signed expiring URLs · session/device binding · disable right-click (frontend) · watermark on download · HLS chunking.

---

15. OG (Open Graph) Strategy

OG images + `og_preview.mp4` are **permanent, unsigned, tier 1** — because crawlers (WhatsApp/Telegram) fetch on share and the user may open hours later; a signed URL would expire → "Error loading." Access control for private posts is enforced **server-side** (the post page returns 403), not via the OG asset. Serve `og_clean.webp` to known platforms, `og_play.webp` (burned play button) to unknown crawlers (User-Agent detection).

---

16. Compression & Transcoding (reference)

Client: light pre-compress (max 1080p, CRF 18, never upscale). Server: FFprobe → adaptive ladder (never upscale, never inflate → `-c:v copy` if output > input). CRF 21/23/25/28 for 1080/720/480/360, 30 for OG preview. 1080p max (all platforms serve ≤1080p anyway). `-movflags +faststart` always; `frag_keyframe+empty_moov+faststart` for fMP4. Now: H.264 + AAC. Future: AV1 + Opus + CMAF.

---

17. BlurHash, LQIP & Dominant Color

Generated for every image + video thumbnail, at processing time. Progressive load: dominantColor (0ms) → BlurHash (0ms) → LQIP (0ms) → thumb.webp (CDN) → full on tap. All three stored in `media_content_signals` and shipped inline in the feed JSON.

---

18. Deduplication

SHA-256 on raw file → `file_hashes`. Exists+clean → skip processing, reference existing variants, `reference_count++`. Delete → decrement; 0 → remove from MinIO. Near-dup (pHash) = future (copyright).

---

19. Quota Management

Enforced at **presigned-URL generation** (atomic SQL). Social = original size charged; digital = all stored bytes. Plans: FREE 1GB / PRO 20GB / BUSINESS 100GB. Warn 80%/95%/100%. Release on abandoned (immediate) and hard-delete (30d after soft). Redis cache, sync to PG every 5 min.

---

20. Storage Lifecycle

raw: auto-delete 24h. public: keep forever, access-based tiering (hot NVMe → warm HDD → cold; viral old content auto-promotes). private: DM 1y, audio 24h, KYC per law. digital: forever. Soft delete → 30d grace → hard delete + Cloudflare purge + quota free.

21. Abandoned Upload Handling

Layer 1: client confirm (fast path). Layer 2: cleanup job every 30 min (PENDING >1h → file exists = recover/process; not = abandoned, release quota). Layer 3: raw lifecycle 24h. `fileId` in object key enables recovery without confirm. Redis distributed lock on the job.

---

22. Progress Tracking (SSE)

SSE (one-way, auto-reconnect, resilient on TZ networks). Redis key `upload_status:{fileId}`: PENDING→UPLOADING→UPLOADED→SCANNING→PROCESSING→LIVE_PARTIAL→READY (or FAILED/QUARANTINED/ ABANDONED). Multi-instance via Redis Pub/Sub bridging emitters.

---

23. Communication Architecture

No direct HTTP between services. **RabbitMQ** (FT ↔ Main Backend, topic `nexgate.media`): `media.upload.request`, `media.upload.url.ready`, `media.ready`, `media.live.partial`, `media.failed`, `media.quarantined`, `digital.product.ready`. **Kafka** (FT → Rec Engine / Analytics, topic `content.new`).

> **[v2 FIX] Events carry `objectKey`s + variants, not signed URLs.** `MEDIA_READY` ships storage keys (e.g. `posts/usr_123/vid_789/720p_clean.mp4`) + placeholders. The Main Backend mints signed URLs **per feed request** from those keys — it never stores a signed URL.

---

24. CDN Strategy

Cloudflare in front of MinIO. First request → edge → MinIO → cache → serve; thereafter from cache. Cache-Control: tier 1 = 1y; tier 2 = path-keyed short edge TTL; tier 3 / API = `private, no-store`. **MinIO never publicly exposed** (only Cloudflare IPs). Phases: Free → Pro ($20) → R2 (zero egress). See §10 cache-key gotcha.

---

25. Shareable Links

Client-side `nexgate.co/reels/{shareCode}?ref=whatsapp`. No `uid` in URL (server resolves owner from shareCode → follow prompt). `ref` drives share analytics. Public content = no token needed for the page; the **media** on the page still follows tier rules (§8).

---

26. Technology Stack

Java 21 / Spring Boot 3.x · FFmpeg (Jaffree) · ImageMagick (IM4Java) · blurhash-java · ClamAV (Docker, TCP 3310) · MinIO · Cloudflare · RabbitMQ · Kafka · Redis · PostgreSQL · TUS · **Vault (`vault.qbitspark.com`) — holds the signing secret** · Docker Compose · SSE.

---

27. Database Schema (key tables + v2 notes)

```
media_files (
  file_id UUID PK, owner_id UUID, directory ENUM, original_name TEXT,
  object_key TEXT,          -- canonical storage key  [v2: keys, never signed URLs]
  mime_type TEXT, file_size BIGINT,
  status ENUM,              -- PENDING..READY..QUARANTINED..ABANDONED
  variants JSONB,           -- variant object_keys (NOT URLs)
  metadata JSONB, scan_result TEXT, hash TEXT,
  is_reel_eligible BOOLEAN, streaming_format ENUM,
  can_download BOOLEAN DEFAULT true,   -- [v2] drives feed canDownload + download gate
  created_at TIMESTAMPTZ, ready_at TIMESTAMPTZ
);

media_content_signals (
  file_id UUID FK, has_audio BOOLEAN, audio_ref TEXT,
  dominant_color TEXT, blurhash TEXT, lqip TEXT,
  aspect_ratio TEXT, quality_score INT, width INT, height INT, duration_seconds DECIMAL
);

-- user_storage_quota, file_hashes, media_variants (HOT/WARM/COLD),
-- digital_product_files, digital_orders, digital_download_logs  (as v1)
```

Indexes: `idx_media_cleanup (created_at) WHERE status='PENDING'`; unique `file_hashes(hash)`; `media_files(owner_id, status, created_at DESC)`.

> **[v2 FIX] No `signed_url` column anywhere.** Signed URLs are minted per request and discarded. The DB stores only `object_key` + `variants` (keys).

---

28. Docker & Infrastructure (reference)

`eclipse-temurin:21` + ffmpeg + imagemagick(+webp/heic). ClamAV separate container (daemon, auto-updates). FT: 4 cores / 16GB (FFmpeg CPU-heavy). ClamAV: 1 core / 2GB. Env: PG, Redis, RabbitMQ, Kafka, MinIO, ClamAV, `VAULT_ADDR`.

---

29. What File Thunder Does NOT Do

 ML/AI classification · Whisper (Rec Engine) · CLIP · feed ranking · social graph · payments · auth · push · search · business logic · fan-out · recommendations. Extracts `audio.wav` temporarily; Rec Engine fetches/transcribes/deletes. FT never knows audio content.

---

30. Open Decisions (parked)

1. **Moat: content vs commerce?** *This drives §14.3 (bot-gate depth) and §8 (how hard to stamp tier-2 public media).* If commerce is the moat (shops/payments/events), public media scraping hurts less → lighter bot gate, tier-2 stamping is softer/optional. If content is the moat → invest in bot gate + stamp tier-2 firmly. **Decide before finalizing anti-scraping work.**
2. **Bot gate depth for launch** — minimum: login + rate limit. Full app-attestation later, sized to (1).
3. **HLS dual-token vs prefix-sign for launch** — prefix-sign on Cloudflare now; edge dual-token if/when heavier CDN (§11).
4. **Download quality picker** — default 720p now; picker + lazy-gen later (§12).

Summary — File Thunder v2 in one line

> Receives raw files → ClamAV → processes into clean + watermarked variants (FFmpeg/IM) with blurhash/lqip/dominant-color → stores **object keys** across 4

> buckets in MinIO (never public-read) → the Main Backend **mints signed delivery URLs per request** from those keys using a three-tier policy (trivial-public unsigned, public-content signed, private/paid signed + auth) → Cloudflare serves and caches by tier → streams use MP4-single-URL or HLS-dual-token, downloads use single-use watermarked URLs → signed URLs + a bot gate together make scraping not worth the effort — all without business logic, ML, or social ops.

File Thunder Architecture Guide v2.0 — NexGate / QBIT SPARK v2 integrates media-security design review. [v2 FIX] tags mark every change from v1.

Revision #1
Created 29 May 2026 10:22:23 by Admin Qbit
Updated 29 May 2026 10:22:44 by Admin Qbit