CDN & File Delivery — Full Walkthrough

This is the learning-session writeup for how Cloudflare fits into File Thunder's delivery story. It complements `FILE_THUNDER.md` §8 (which is the terse reference version) with full end-to-end walkthroughs, diagrams, and setup steps.

Nothing in this doc changes the processing pipeline.
FFmpeg/ImageMagick/ClamAV/watermarking stay exactly as they are. This is only about the *last mile* — how a finished file reaches a viewer.

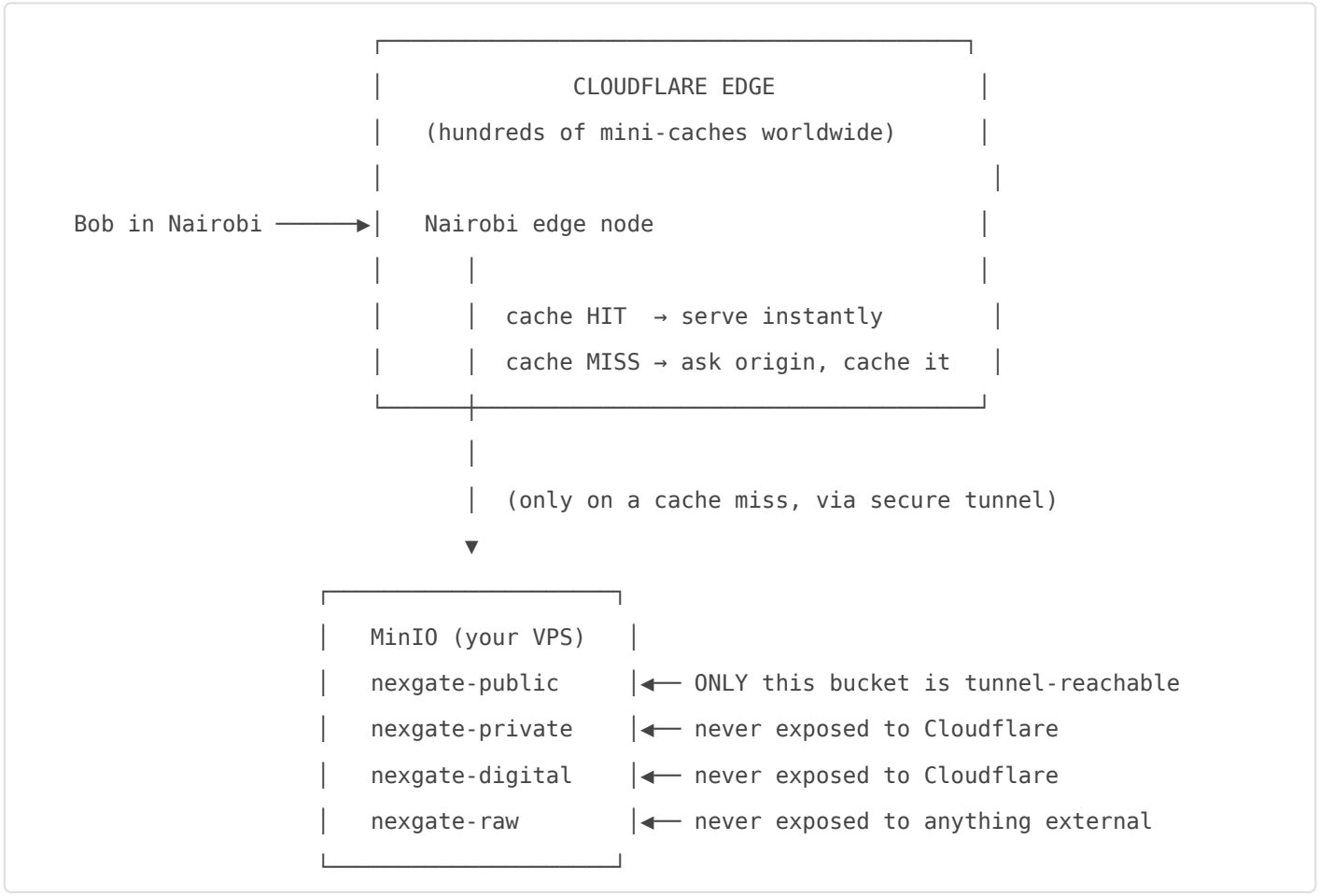
Table of Contents

---

1. The Core Idea, In One Picture

Cloudflare is **not storage**. It's a worldwide network of caches sitting in front of your one real origin (MinIO). Think of MinIO as a single warehouse, and Cloudflare as thousands of small local libraries

that keep copies of the popular stuff close to whoever's asking.



Every request after the first one, from anyone near that edge node, never touches your VPS at all.

---

2. The Three Delivery Lanes

Every file File Thunder ever produces falls into exactly one lane. The lane is decided by **who is allowed to see the file**, nothing else.

	LANE 1: PUBLIC	LANE 2: PRIVATE	LANE 3: PROTECTED
Who can view it	Anyone	The 1-2 people in the conversation	Only the verified buyer

```
├──────────────────┼──────────────────────┼──────────────────────┼──────────────────────
┤
│ Cloudflare cache?  │ YES — cached forever  │ NO — always bypassed   │ NO — always
bypassed   │
├──────────────────┼──────────────────────┼──────────────────────┼──────────────────────
┤
│ URL type           │ Plain permanent URL   │ Signed, 15-min expiry  │ Signed, 15-min
expiry  │
│                    │ cdn.nexgate.com/key   │ straight from MinIO    │ + audit log + re-
scan  │
├──────────────────┼──────────────────────┼──────────────────────┼──────────────────────
┤
│ Bucket             │ nexgate-public        │ nexgate-private        │ nexgate-
digital        │
├──────────────────┼──────────────────────┼──────────────────────┼──────────────────────
┤
│ Who assembles URL  │ Main backend, no call │ Main backend calls FT  │ Main backend calls
FT  │
│                    │ to File Thunder needed│ generateDownloadUrl()  │
generateDownloadUrl()  │
└──────────────────┴──────────────────────┴──────────────────────┴──────────────────────
┘
```

Why Lane 2/3 can never be cached: their URLs are unique per request (a signature + expiry baked into the query string). Caching is keyed on the URL — a signed URL is never requested twice, so there's nothing to cache, and worse, a shared cache is the last place private bytes should sit.

---

3. Category ? Lane Mapping

Every `MediaContext` value maps to exactly one lane:

```
LANE 1 — PUBLIC / CDN-CACHED (nexgate-public)
├── SOCIAL_IMAGE, SOCIAL_VIDEO
├── PROFILE_PICTURE, COVER_PHOTO
├── PRODUCT_IMAGE, PRODUCT_VIDEO
├── PRODUCT_PREVIEW_IMAGE, PRODUCT_PREVIEW_VIDEO   ← teasers, meant to be public
├── SHOP_BANNER, SHOP_LOGO
└── EVENT_COVER, EVENT_GALLERY
```

```
LANE 2 — PRIVATE / SIGNED, NEVER CACHED (nexgate-private)
├── DM_IMAGE
├── DM_VIDEO
└── DM_DOCUMENT

LANE 3 — PROTECTED / SIGNED + AUDITED, NEVER CACHED (nexgate-digital)
├── DIGITAL_PRODUCT
└── PRODUCT_PREVIEW_DOCUMENT   ← a document preview is still gated like a purchase
```

This matches `DOWNLOAD_CONTEXTS` already defined in `MediaQueryServiceImpl` — Lane 2 + Lane 3 together are exactly the contexts that go through `generateDownloadUrl()`. Everything else is Lane 1.

---

4. How Cloudflare Physically Sits in the Middle

Three ingredients, none of which touch your Java code:

1. **A subdomain**: `cdn.nexgate.com` (and `cdn-staging.nexgate.com` for staging).
2. **DNS pointed at Cloudflare** for that subdomain, "proxied" (orange cloud ON) — this is what makes Cloudflare the middleman instead of a pass-through record.
3. **A Cloudflare Tunnel** (`cloudflared`, a small background process on your VPS) that opens an *outbound* connection from your server to Cloudflare. This is the important safety property:

```
WITHOUT a tunnel                          WITH a tunnel
─────────────────                         ─────────────

Internet ──▶ your VPS:9000                Internet ──▶ Cloudflare ──▶ tunnel ──▶ your VPS
(port must be open to the world,          (port 9000 is never opened to the world —
 firewall rules, real attack surface)      the VPS "calls out", nobody calls in)
```

The tunnel is configured to route **only** to `nexgate-public`. `nexgate-private`, `nexgate-digital`, and `nexgate-raw` are not part of the tunnel's routing table — Cloudflare has no path to them even if it wanted one.

---

5. Walkthrough A — Social Video (Lane 1, Public/Cached)

Alice posts a 40-second clip. Bob, her friend, watches it later.

```
ALICE (upload)                                  BOB (view, later)
──────────────                                  ─────────────────

1. Tap "post"                                   6. Opens app, scrolls to Alice's post
        │                                               │
        ▼                                               ▼
2. Main backend → File Thunder:                 7. Main backend reads post from its DB,
   "need upload slot for SOCIAL_VIDEO"             sees context = SOCIAL_VIDEO (Lane 1)
        │                                               │
        ▼                                               ▼
3. File Thunder → presigned PUT URL             8. No call to File Thunder needed —
   → nexgate-raw (private, temp)                   main backend just glues:
        │                                           cdn.nexgate.com + stored objectKey
        ▼                                               │
4. Phone uploads raw bytes directly to MinIO            ▼
   (File Thunder never touches the bytes)       9. Bob's phone requests that URL
        │                                               │
        ▼                                               ▼
5. MinIO event → File Thunder:                  10. Nearest Cloudflare edge (e.g. Nairobi):
   PENDING → UPLOADED → VideoWheel                   - cache MISS (first viewer ever)
     - FFmpeg: short-path transcode                    → tunnel → MinIO → fetch once
       (360p/720p/1080p, watermark, outro)             - cached at that edge from now on
     - thumbnail + blurhash + preview clip                  │
     - LIVE_PARTIAL after 360p, READY after all             ▼
     - raw original deleted                     11. Every later viewer near Nairobi:
     - keys written to Postgres (not URLs)          cache HIT — instant, MinIO untouched
```

Key point: steps 1–5 (upload + processing) are entirely unrelated to CDN — that pipeline already works today. The CDN only enters at step 7 onward, and it's a pure "glue the domain onto the key" operation. No signing, no expiry, no per-viewer logic, because it's public by design.

If the clip had been ≥3 minutes (HLS path), step 10/11 repeats per segment: the player first fetches `master.m3u8` from the CDN, then a stream of `.ts` segments, each cached the same way.

---

6. Walkthrough B — Direct Message Image (Lane 2, Private/Bypassed)

Alice sends Bob a private photo in DM.

```
1. Alice sends photo → uploads to nexgate-raw (same presigned-upload pattern as Lane 1)
2. ImageWheel processes it → variants written to nexgate-private (NOT nexgate-public)
3. Bob opens the conversation
       │
       ▼
4. Main backend checks: is Bob actually a participant in this conversation? (authorization)
       │
       ▼
5. Main backend calls File Thunder → generateDownloadUrl(fileId, requesterId=Bob)
       │
       ▼
6. File Thunder issues a PRESIGNED GET URL — 15 min expiry, straight from MinIO
       │
       ▼
7. Bob's phone fetches directly from MinIO — the CDN is never involved in this request at all
       │
       ▼
8. URL expires after 15 minutes — if Bob reopens the chat later, a fresh one is issued
```

If a stranger somehow obtained this URL, it would already be expired or tied to a bucket that Cloudflare's tunnel can't even route to — two independent layers of protection.

---

7. Group Chats — Same Lane, Different Scale

Group chat media (photos/videos/documents shared in a group conversation, not a 1:1 DM) still belongs in **Lane 2** — same bucket (`nexgate-private`), same `generateDownloadUrl()` call, never cached. File Thunder doesn't actually know or care whether it's a 1:1 DM or a 300-person group; it only trusts whatever `requesterId` the main backend passes in.

What actually changes is who's allowed to ask:

```
1:1 DM:        "is requesterId one of the 2 people in this conversation?"
Group chat:    "is requesterId a CURRENT member of this group?"
```

That authorization check lives entirely in the main backend, before it ever calls File Thunder — nothing to design differently on the File Thunder side for correctness.

The real new problem: fan-out

In a 1:1 DM, at most 2 people will ever request a given file — cheap no matter how it's delivered. In a group chat, one photo sent to a 300-person group means up to 300 people opening it:

```
300 members → 300 separate calls to generateDownloadUrl()
            → 300 separate presigned URLs generated
            → 300 separate direct-to-MinIO fetches
            → none of it cached (Lane 2 is always cache-bypassed) — all of it hits the one VPS
```

That's a materially different load profile than a private DM, and it's worth designing around before group chat ships — but the fix is about *reducing redundant origin calls*, not about caching private files at a shared edge.

Option 1 — do nothing special (fine for small/medium groups)

A few dozen members fetching a photo once is trivial load for MinIO on a VPS. If groups stay small (family/friend-sized, not broadcast channels), Lane 2 as-is is genuinely fine — no need to build anything extra ahead of time.

Option 2 — share ONE signed URL instead of one per member (for large groups)

Instead of every member independently triggering `generateDownloadUrl()`, main backend generates the presigned URL **once** — when the message is sent, or on first view — caches that single URL (e.g. in Redis, keyed by message/file id), and hands the *same* URL to every member who opens the chat until it expires (15 min). This turns "300 origin hits" into "1 origin hit, 300 people reusing the same short-lived link":

```
Member 1 opens chat ⟶ main backend: URL cached? NO ⟶ call generateDownloadUrl() ⟶ cache
it
Member 2 opens chat ⟶ main backend: URL cached? YES ⟶ reuse same URL, no MinIO call
Member 3-300 ...      ⟶ same — reuse until the 15-min expiry, then regenerate once
```

Still never touches Cloudflare's shared cache — this is purely about cutting redundant presigned-URL generation and redundant MinIO traffic, which is a much simpler problem than "how do we safely cache private files."

When a "group" stops being private

If a group grows into "public broadcast channel" territory (thousands of members, more like a public feed than a private conversation), that's usually a sign the content has drifted out of Lane 2 entirely — worth revisiting whether it should be public (Lane 1) or a members-only public tier, rather than forcing an increasingly public thing through the private/signed lane. Not a decision to make now — just a pattern to recognize if it comes up.

---

8. Voice Notes & Live Streaming — Do the Same Rules Apply?

Neither exists in File Thunder today (no `MediaContext` for either, no live-ingest infra). They split sharply: one fits the existing model perfectly, the other breaks a core assumption.

Voice notes — yes, exactly the same rules

A voice note is just a short, finished audio file. It doesn't break anything this guide relies on: upload once → process once → immutable key → serve. It drops straight into the existing lanes:

```
Voice note in a DM/group chat  → Lane 2  (same as DM_IMAGE/DM_VIDEO — private bucket, 15-min
                                          presigned URL, never cached)
Voice note on a public post    → Lane 1  (same as SOCIAL_IMAGE — public bucket, plain CDN URL,
                                          cached forever)
```

The only new work is a new `MediaContext` (e.g. `DM_VOICE_NOTE`) and a small audio wheel (FFmpeg re-encode to a consistent format/bitrate, maybe waveform-peaks generation for the UI — same spirit as ImageWheel's blurhash). **Delivery-wise, there is nothing new to design.**

Live streaming — no, this breaks a core assumption

Everything so far relies on one idea: **a file is finished, then served.** Upload → process → immutable key → cache forever. Live streaming violates that at its core — the "file" is still being created while people are watching it. That changes several things at once:

1. **No presigned-PUT-to-MinIO upload.** A live stream needs a real-time ingest protocol (RTMP, SRT, or WebRTC) hitting an ingest server continuously — not a client uploading one finished file.
2. **The manifest never stabilizes.** VOD HLS's `master.m3u8` flips once (`LIVE_PARTIAL` → `READY`) and never changes again — that's why "short-cache the manifest, cache segments

forever" (§10) works for VOD. A *live* HLS playlist is a **sliding window** — rewritten every few seconds to point at only the newest 3-5 segments, continuously, for the whole stream duration. That manifest needs a near-zero cache TTL (1-2s) the entire time, not a one-time flip.
3. **Individual segments are still cacheable** — each 2-6 second `.ts` chunk is immutable once written, so Cloudflare still helps a lot for segment traffic, just not for the manifest.
4. **Private/restricted live streams** (e.g. a paid live-shopping event — Lane 2/3 equivalent) can't use one 15-minute signed link the way a DM photo does, because the session usually runs longer than that. This needs rotating/refreshing tokens or signed cookies for the stream's duration — new machinery, not something the current design already provides.

Because of points 1 and 4, this genuinely isn't "reuse the existing lanes" — it needs its own ingest + packaging pipeline. Building that in-house (RTMP ingest server + real-time FFmpeg transcode + rolling manifest) is a serious lift, well beyond a delivery-lane decision. The pragmatic path most platforms take at this stage is to **not build live ingest in-house** and use a managed service (Cloudflare Stream Live, Mux, Agora, etc.) that hands back an HLS URL to embed — which then *does* slide back into the Lane 1 viewing model, just with someone else running the real-time part.

Bottom line: voice notes are a non-event — add the context, reuse the lanes. Live streaming is a separate architectural track that shouldn't be forced into this document's model; buying that piece is worth strongly considering before building it in-house.

---

9. Video/Audio Calls — Not a File Thunder Concern

This is the sharpest split of all. Everything else in this guide — even live streaming — is about media that gets **stored** (however briefly) and then **fetched by someone else, later or from elsewhere**. A 1:1 or group call is neither.

1. **Nothing is ever "delivered" through the storage layer.** A call is devices talking to each other in as close to real time as possible — 100-300ms round trip matters, not "eventually arrives." Routing it through MinIO → Cloudflare → viewer would add multiple seconds of latency. Wrong tool entirely.
2. **The transport is peer-to-peer or SFU, not HTTP/CDN.** Calls use **WebRTC**: media streams connect device-to-device directly when possible, and go through a lightweight **SFU (Selective Forwarding Unit)** relay when a direct connection isn't possible (NAT/firewall issues) — the SFU forwards packets, it doesn't store them. A completely different network model from "upload a file, cache it, serve it."
3. **Normally nothing to process, scan, or watermark.** No FFmpeg wheel, no ClamAV, no thumbnails — there's no file, just a live signal that's gone the instant it's forwarded.

4. **Cloudflare's role, if any, is different too**: not caching — it would be **TURN/STUN relay** (helping two devices behind NATs find each other, or relaying when they can't connect directly). Cloudflare Calls / Cloudflare Realtime is their product here, but it conceptually replaces the SFU, not the CDN edge cache used everywhere else in this guide.

Calls need their own **separate service**, not a feature of File Thunder — worth calling out explicitly so this doesn't get scoped into "file delivery" work by mistake later.

The one place calls do touch File Thunder: if a call gets recorded and saved afterward (e.g. a missed-call "voicemail," or a saved video-call recording), it's no longer a live call at that point — it's a finished file, and it goes right back into the existing model: probably `DM_VIDEO` or a new `CALL_RECORDING` context, Lane 2, same rules as any other private media.

---

10. Walkthrough C — Digital Product Purchase (Lane 3, Protected/Audited)

Alice sells a paid PDF course. Bob buys it.

```
 1. Bob completes payment → main backend confirms order in its own DB
        |
        ▼
 2. Main backend calls File Thunder → generateDownloadUrl(fileId, requesterId=Bob)
        |
        ▼
 3. File Thunder checks:
      - context is DIGITAL_PRODUCT? ✓ (only DIGITAL_PRODUCT / DM_DOCUMENT /
PRODUCT_PREVIEW_DOCUMENT
        are allowed through this endpoint at all)
      - file status == READY? ✓
        |
        ▼
 4. File Thunder writes a DownloadAuditEntity row:
      fileId, ownerId, requesterId=Bob, objectKey, requestedAt, expiresAt
      (a permanent record of exactly who downloaded what, and when)
        |
        ▼
 5. Presigned GET URL issued — 15 min expiry, bucket = nexgate-digital
        |
```

```
        ▼
6. Bob downloads directly from MinIO — never cached, never at a shared edge
```

The audit row is what lets you answer "who downloaded this file, and when" months later — important for a paid-goods platform if there's ever a dispute or a leak investigation.

---

11. Cache Invalidation — The Only Sharp Edge

Because object keys are unique per `fileId`/variant and are **never overwritten**, a cached copy at the edge can basically never go stale — new content always gets a new key. That removes 99% of the usual "how do I bust the cache" headache. Three known exceptions:

Case	Problem	Fix
HLS `master.m3u8`	Same key, first written when only 360p is ready (`LIVE_PARTIAL`), then rewritten when all qualities finish (`READY`)	Short cache lifetime on this one file only (10–30s). The `.ts` segments never change — cache those forever
User thumbnail overwrite	`user-poster.webp` reuses the same key on re-upload	Prefer versioning the key (`user-poster-v2.webp`) over a cache purge call
Profile picture / cover / shop logo	Same "identity slot gets overwritten" pattern	Same fix — version the key rather than relying on purge

Versioning the key is preferred over calling Cloudflare's purge API because it needs no extra network call and can never race (old URL still valid until the new one is swapped in by the app).

---

12. Cloudflare Setup Checklist

Concrete steps, in order, when it's time to actually turn this on:

1. **Add the domain to Cloudflare** (or subdomain, if the root domain is hosted elsewhere).
2. **Create `cdn.nexgate.com` DNS record**, proxied (orange cloud ON). Create `cdn-staging.nexgate.com` the same way for staging.
3. **Install `cloudflared` on the VPS**, authenticate it to your Cloudflare account, create a tunnel, and route it to `http://localhost:9000` (MinIO) — but only for requests that resolve to the `nexgate-public` bucket path.
4. **Re-enable anonymous read, but only on `nexgate-public`** (the other three buckets stay fully private — do not touch their bucket policy).

5. **Firewall the MinIO port** so it's not reachable from the raw internet at all — the tunnel is the only path in. (If a tunnel isn't used yet, at minimum restrict to Cloudflare's published IP ranges.)
6. **Set a Cache Rule** in Cloudflare for `cdn.nexgate.com/*`: cache everything, respect `Cache-Control: public, max-age=31536000, immutable` from origin — except `*/master.m3u8`, which gets its own short-TTL rule (10–30s).
7. **Flip the app config** (already scaffolded per `FILE_THUNDER.md` §8):

   ```
   ft.storage.mode=cdn
   ft.cdn.base-url=https://cdn.nexgate.com
   ```

8. **Smoke test**: upload a test image, confirm the CDN URL 200s, re-request it and confirm the `cf-cache-status` response header flips from `MISS` to `HIT` on the second request.

---

13. Video-at-Scale Caveat

Cloudflare's free/pro plans restrict using the CDN as a bulk *video* host at large scale (their terms are written around "no non-HTML heavy media" abuse). Images, thumbnails, blurhash/lqip, and HLS manifests are completely fine on the free tier — this whole design is ideal for those. But if social video volume grows large, the two Cloudflare-native upgrade paths to know about later are:

- **Cloudflare R2** — S3-compatible object storage, zero egress fee to Cloudflare's own edge. Natural fit since File Thunder already stores keys, not URLs — swapping the origin later doesn't change any application logic.
- **Cloudflare Stream** — paid, purpose-built for HLS/video delivery with its own player and encoding.

Not a decision to make now — just something to keep in mind as a later migration, not a redesign.

Revision #3
Created 5 July 2026 08:37:48 by Admin Qbit
Updated 5 July 2026 09:00:46 by Admin Qbit