File & Media Thunder

- File Thunder — Architecture & Design Guide

File Thunder — Architecture & Design Guide

File Thunder — Architecture & Design Guide

NexGate Media Engine | Version 1.0

Table of Contents

---

1. What is File Thunder?

File Thunder is NexGate's dedicated media processing engine. It is a standalone Spring Boot microservice responsible for all file and media operations across the entire NexGate 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 file upload requests
- Validate files (type, size, quota)
- Virus scanning
- Transcoding and processing
- Generating variants (thumbnails, WebP, HLS, MP4)
- Watermarking
- Storing files in MinIO
- Serving files via CDN
- Publishing media-ready events

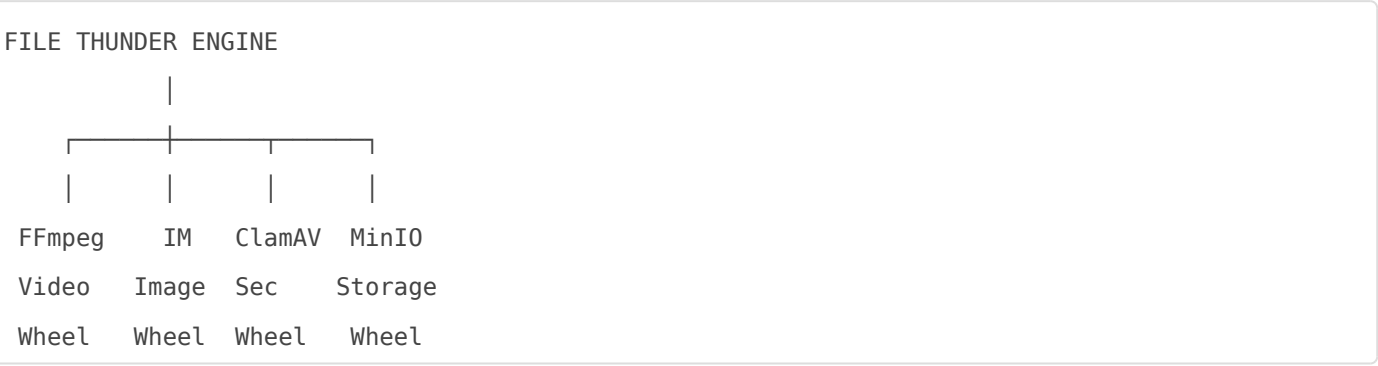
What it is NOT:

- Not an AI/ML service
- Not a recommendation engine
- Not a social graph service
- Not a payment service

---

2. The Four Wheels

File Thunder is powered by four core engines:



Wheel 1 — FFmpeg (Video)

- Video transcoding (any format → H.264 MP4 or HLS)
- Audio extraction (for Rec Engine)
- HLS segment generation
- Thumbnail extraction (best frame detection)
- 3-second preview clip generation
- Watermark overlay (logo + username)
- Aspect ratio handling and blur padding
- Fragmented MP4 (fMP4) generation
- GIF → MP4 conversion
- Format conversion (MOV, MKV, AVI, WEBM → MP4)
- Audio normalization

Java wrapper: Jaffree (calls `/usr/bin/ffmpeg` via ProcessBuilder)

Wheel 2 — ImageMagick (Images)

- Resize to multiple variants
- Format conversion (JPEG, PNG, HEIC → WebP)
- EXIF stripping (privacy — removes GPS data)
- Auto-orientation (fix rotated phone photos)
- Quality compression
- Dominant color extraction
- LQIP generation (10×10 base64)
- OG image generation (1200×630)
- GIF → WebP animated conversion
- PDF → preview image conversion

Java wrapper: IM4Java (calls `/usr/bin/convert` via ProcessBuilder)

Additional library: `blurhash-java` for BlurHash generation

Wheel 4 — MinIO (Storage)

- Stores ALL file variants across 4 buckets
- S3-compatible object storage (self-hosted)
- Receives direct uploads via presigned URLs (client never touches File Thunder bandwidth)
- Serves as origin for Cloudflare CDN
- Supports multipart uploads (files > 5MB)
- Supports byte-range requests (progressive streaming, resumable downloads)
- Lifecycle policies (auto-delete raw uploads after 24h)
- Storage class tiering (hot → warm → cold)
- Bucket event notifications (upload complete events)
- Never exposed publicly (only Cloudflare and File Thunder can reach it)

Java SDK: MinIO Java SDK (`io.minio:minio`)

Wheel 3 — ClamAV (Security)

- Virus and malware scanning for every uploaded file
- Hash-based scan cache (skip re-scanning known clean files)
- Runs as separate Docker container (daemon-based)
- File Thunder connects via TCP socket (`clamav:3310`)
- Digital products scanned twice (on upload + before download)

Java client: clamd4j or custom TCP socket client

---

3. Storage Architecture (MinIO Buckets)

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

```
nexgate-raw/          ← temporary upload landing zone
nexgate-public/       ← CDN cached, social content
nexgate-private/      ← signed URLs, DMs + docs
nexgate-digital/      ← purchase-gated, digital products
```

nexgate-raw (Temporary)

```
nexgate-raw/
  uploads/{userId}/{fileId}/
    original.ext     ← raw upload lands here
```

- Auto-delete lifecycle policy: 24 hours
- Incomplete multipart uploads aborted: 24 hours
- Never served to any client
- Deleted by File Thunder after processing completes

nexgate-public

```
nexgate-public/
  profiles/{userId}/{fileId}/
    avatar_400.webp
    avatar_150.webp
    avatar_50.webp
    cover.webp

  posts/{userId}/{fileId}/
    360p_clean.mp4
    720p_clean.mp4
    1080p_clean.mp4
    360p_watermarked.mp4
    720p_watermarked.mp4
    1080p_watermarked.mp4
    preview_3s.mp4
    hls/master.m3u8
    hls/360p/segment_000.ts ...
    hls/720p/segment_000.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

  categories/{categoryId}/{fileId}.webp
```

nexgate-private

```
nexgate-private/
  messages/{conversationId}/{fileId}/
    original.jpg
    thumb.webp

  audio/{fileId}.wav      ← temp for Rec Engine (deleted after transcription)

  documents/{userId}/{fileId}/
    original.pdf
    preview.webp

  kyc/{userId}/{fileId}/
    id_front.jpg
    id_back.jpg
```

nexgate-digital

```
nexgate-digital/
  products/{shopId}/{productId}/{fileId}/
    original/
      file.pdf           ← actual purchased product
      checksum.txt       ← SHA-256 hash
    preview/
      preview.webp       ← low-res watermarked preview
      cover.webp         ← product cover image
```

Object Key Pattern (Consistent Rule)

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

Examples:
nexgate-public/posts/usr_123/post_456/file_789/thumb.webp
nexgate-private/messages/conv_abc/file_789/original.jpg
nexgate-digital/products/shop_xyz/prod_123/file_789/original.pdf
```

---

4. Accepted File Formats

Images

Accept	Reject
JPEG/JPG	BMP
PNG	TIFF
HEIC/HEIF	RAW
WebP	SVG (security risk)
GIF	

Output: Always WebP

Videos

Accept	Reject
MP4	WMV
MOV	FLV
MKV	VOB
WEBM	
AVI	
3GP	

Output: H.264 MP4 (short) or HLS H.264 (long)

Digital Products

PDF, DOCX, XLSX, PPTX, GLB, OBJ, FBX, STL, MP3, WAV, FLAC, ZIP, RAR, PSD, AI, PNG (stock art)

---

5. Upload Flow

Step-by-Step

```
CLIENT
  ↓
[1] Intelligent client-side compression
    - Detect: resolution, bitrate, codec, network type, device tier
    - Compress if: over-bitrated, slow network, large file
    - Never compress if: already optimized
    - Target: max 1080p, CRF 18 (light)
    - HEIC → JPEG conversion (iOS)
  ↓
[2] POST /media/upload-request
    {
      fileName, fileSize, mimeType,
      directory, clientMeta: {
        clientApp, networkType,
        deviceTier, wasCompressed
      }
    }
  ↓
[3] File Thunder validates:
    - MIME type in allowed list?
    - File size within per-upload limit?
    - User quota not exceeded? (atomic SQL check)
    - Creates DB record: status PENDING
    - Generates presigned MinIO URL → nexgate-raw
    - Returns { fileId, uploadUrl, expiresIn: 1800 }
  ↓
[4] Client uploads directly to MinIO
    - TUS resumable protocol for large files
    - Multipart upload for files > 5MB
```

```
    - Progress tracked client-side
    - Upload starts during caption writing (parallel)
  ↓
[5] Client confirms: POST /media/confirm { fileId }
    (or cleanup job recovers if confirm missed)
  ↓
[6] Processing pipeline starts
```

Quota + Duration Enforcement

Size check → at presigned URL generation (before upload):

```
Atomic SQL:
  UPDATE user_storage_quota
  SET used_bytes = used_bytes + fileSize
  WHERE user_id = ?
  AND (used_bytes + fileSize) <= quota_bytes
  RETURNING used_bytes

  0 rows affected → quota exceeded → reject 403
  1 row affected  → quota reserved → proceed
```

Duration check → after upload (FFprobe analysis):

```
Client sends estimated duration in clientMeta (hint only)
FFprobe confirms actual duration after upload

If actual duration > plan limit:
  → delete file from nexgate-raw
  → reject with 403: { error: "DURATION_EXCEEDED", plan: "FREE", maxDuration: 300 }
  → release quota reservation
  → notify user: "Upgrade plan to upload longer videos"
```

Why duration checked after upload (not before):

```
Client-reported duration = not trusted
  = could be manipulated
FFprobe = ground truth
  = server side, reliable
  = only way to confirm actual duration
```

Per-Upload Size + Duration Limits

Plan	Image	Short Video (social)	Long Video	Digital Product Video	Duration
FREE	20MB	200MB	Not allowed	Not allowed	Max 5 min
PRO	20MB	500MB	2GB	5GB per file	Max 60 min
BUSINESS	20MB	2GB	5GB	20GB per file	Unlimited

Why duration matters as much as size:

```
Same 200MB could be:
  3 minute 1080p reel   → acceptable 
  45 minute long video  → not acceptable for FREE 

Size check alone = not enough
Duration check = required alongside size

Enforcement:
  Size  → checked at upload request (before presigned URL)
  Duration → checked AFTER upload (FFprobe analysis)
             client also sends estimated duration in clientMeta

  If duration exceeds plan limit after FFprobe:
    → delete from nexgate-raw
    → reject processing
    → notify user: "Upgrade to upload longer videos"
    → refund quota reservation
```

Duration limits by plan:

```
FREE:
  Reels/short video → max 5 minutes
  Long video        → not allowed

PRO:
  Reels/short video → max 5 minutes (treated as short)
  Long video        → max 60 minutes

BUSINESS:
  All video         → unlimited duration
```

6. Processing Pipelines

6.1 Image Pipeline

```
Upload confirmed
  ↓
RabbitMQ: SCAN job
  ↓
ClamAV scan → VIRUS: quarantine + notify | CLEAN: continue
  ↓
SHA-256 hash check (deduplication)
  Exists + clean → skip processing, reuse variants
  New → continue
  ↓
RabbitMQ: PROCESS job → Image Worker
  ↓
[1]  Auto-orient (apply EXIF rotation before stripping)
[2]  Extract useful EXIF (orientation only)
[3]  Strip ALL EXIF (privacy — remove GPS, device info)
[4]  AI moderation check (NSFW detection)
     UNSAFE → quarantine | SAFE → continue
[5]  Extract dominant color (resize to 1×1 pixel)
[6]  Generate LQIP (resize to 10×10, base64 encode)
[7]  Generate BlurHash (blurhash-java library)
[8]  Generate variants by content type:
     Profile   → 400px, 150px, 50px (WebP)
     Post      → 1600px, 800px, 300px (WebP)
     Product   → 1000px, 500px, 200px (WebP, 1:1 square)
     Event     → 1200×630px, 800×420px (WebP)
     Story     → 1080×1920px (WebP)
[9]  Convert all variants → WebP
[10] Generate OG image (1200×630, WebP):
     - og_clean.webp (no play button)
     - For non-16:9 → blur pad background to fill 16:9
[11] Store all variants → nexgate-public
[12] Delete original from nexgate-raw
[13] Update DB: status READY, populate variants JSONB
```

```
[14] Publish to Kafka: FILE_READY event
[15] Publish to RabbitMQ: MEDIA_READY event → Main Backend
```

Variant sizes per content type:

Content	Variants
Profile picture	400px, 150px, 50px
Post image	1600px (large), 800px (medium), 300px (thumb)
Product image	1000px, 500px, 200px (always 1:1 square)
Event banner	1200×630, 800×420, 400×210
Category	400px wide

6.2 Short Video Pipeline (MP4)

Threshold: Duration < 3 minutes → MP4

```
Upload confirmed
  ↓
RabbitMQ: SCAN job
  ↓
ClamAV scan → VIRUS: quarantine | CLEAN: continue
  ↓
SHA-256 hash check (deduplication)
  ↓
FFprobe analysis:
  - resolution, bitrate, fps
  - codec, duration, aspectRatio
  - hasAudio, qualityScore
  ↓
Adaptive transcode decision:
  - Never upscale (never exceed input resolution)
  - Never inflate (if output > input size, use original)
  - Only generate eligible variants
  ↓
RabbitMQ: PROCESS job → Video Worker
  ↓
[FAST LANE — runs first for UX]:
  Quick 360p transcode → post goes LIVE_PARTIAL
```

```
  User notified: "Almost ready!"
  ↓
[FULL PROCESSING — continues async]:
[1]  Detect aspect ratio (FFprobe):
     9:16  (vertical)  → native reel format → no padding needed
     16:9  (landscape) → needs blur pad for reel pool
     1:1   (square)    → needs blur pad for reel pool
     4:5   (portrait)  → needs blur pad for reel pool
     Other             → preserve original, blur pad if reel eligible

[2]  Transcode CLEAN variants (H.264, faststart):

     For NATIVE 9:16 videos:
       Transcode directly to 9:16 variants
       360p_clean.mp4  (360×640,  CRF 28)
       720p_clean.mp4  (720×1280, CRF 23)
       1080p_clean.mp4 (1080×1920, CRF 21, if eligible)

     For NON-9:16 videos (isReelEligible = true):
       Apply BLUR PAD to fill 9:16 frame (TikTok/Instagram style)

       What blur pad looks like:
       ┌─────────────────┐
       │▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒│  ← blurred background (scaled up + blurred)
       │▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒│
       │┌───────────────┐│
       ||               ||
       ||  original     ||  ← actual video centered (no cropping)
       ||   video       ||
       ||               ||
       │└───────────────┘│
       │▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒│
       │▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒│
       └─────────────────┘

       Why blur pad (not black bars, not crop):
         Black bars → looks amateur □
         Crop → destroys content □
         Blur pad → professional, intentional □
                    TikTok + Instagram do this □
```

No content lost □

FFmpeg blur pad command:

```
ffmpeg -i input.mp4 \
  -filter_complex \
  "[0]scale=720:1280:force_original_aspect_ratio=increase,
   crop=720:1280,
   boxblur=20:5[bg];
   [0]scale=720:1280:force_original_aspect_ratio=decrease,
   pad=720:1280:(ow-iw)/2:(oh-ih)/2[fg];
   [bg][fg]overlay=(W-w)/2:(H-h)/2" \
  output_9_16_blurpad.mp4
```

Steps:

- bg layer → scale to FILL 9:16, crop excess, apply blur (20px radius)
- fg layer → scale to FIT in 9:16, letterbox, center
- overlay → sharp video (fg) on top of blurred background (bg)

For NON-9:16 videos (isReelEligible = false):

- Keep original aspect ratio
- No blur pad
- Feed player handles letterboxing client-side

[3] Generate WATERMARKED variants:

FFmpeg overlay — MOVING watermark (TikTok style):

- Watermark jumps between corners every 3 seconds:
  - 0-3s → top left (10px, 10px)
  - 3-6s → top right (W-w-10, 10px)
  - 6-9s → bottom right (W-w-10, H-h-10)
  - 9-12s → bottom left (10px, H-h-10)
  - repeat...
- Logo PNG: 80×80px, 60% opacity
- Username text: same position as logo (below it)
- Based on clientApp field:
  - NEXGATE_ANDROID / NEXGATE_IOS / NEXGATE_WEB → "NexGate"
  - NEXGATE_LITE → "NexGate Lite"

Why moving watermark:

- Static corner = easy to crop out □
- Moving = appears in all corners = impossible to crop □

```
       TikTok confirmed uses this strategy

     FFmpeg command (moving watermark):
       overlay=
         x='if(lt(mod(t,12),3), 10,
            if(lt(mod(t,12),6), W-w-10,
            if(lt(mod(t,12),9), W-w-10, 10)))':
         y='if(lt(mod(t,12),3), 10,
            if(lt(mod(t,12),6), 10,
            if(lt(mod(t,12),9), H-h-10, H-h-10)))'

     Variants generated:
     360p_watermarked.mp4
     720p_watermarked.mp4
     1080p_watermarked.mp4

[4]  Generate extras:
     thumbnail.webp     → best frame (brightness + sharpness scored)
     thumbnail LQIP     → 10×10 base64
     thumbnail BlurHash → blurhash-java
     thumbnail dominantColor → hex
     preview_3s.mp4     → first 3 seconds, 360p, muted, watermarked
     og_clean.webp      → 1200×630, clean thumbnail
     og_play.webp       → 1200×630, NexGate play button burned in
     og_preview.mp4     → 360p, watermarked, permanent CDN URL

[5]  Generate Fragmented MP4 (fMP4):
     ffmpeg flags: frag_keyframe+empty_moov+faststart
     1-second fragment intervals (keyframe every 30 frames at 30fps)
     Enables byte-range requests for progressive streaming

[6]  Extract audio for Rec Engine:
     ffmpeg -vn -acodec pcm_s16le -ar 16000 -ac 1 audio.wav
     Store → nexgate-private/audio/{fileId}.wav
     (Rec Engine fetches, transcribes, deletes)

[7]  Store all variants → nexgate-public
[8]  Delete original from nexgate-raw
[9]  Update DB:
     status: READY
```

```
     isReelEligible: true (duration < 3min)
     streamingFormat: MP4
     variants: JSONB with all paths
     aspectRatio, qualityScore, duration
[10] Publish to Kafka: FILE_READY (with audioRef)
[11] Publish to RabbitMQ: MEDIA_READY → Main Backend
     Main Backend:
       → update post status LIVE
       → index into reel pool
       → trigger fan-out to followers
```

Reel pool eligibility:

```
duration < 3 minutes → isReelEligible: true → indexed in reel pool
duration ≥ 3 minutes → isReelEligible: false → feed only
```

MP4 variant resolutions by aspect ratio:

Aspect	360p	540p	720p	1080p
9:16 (vertical)	360×640	540×960	720×1280	1080×1920
16:9 (landscape)	640×360	—	1280×720	1920×1080
1:1 (square)	360×360	—	720×720	1080×1080

Why 1080p is max (not 4K):

All major platforms compress 4K down to 1080p on delivery anyway:

- TikTok → accepts 4K, serves 1080p max
- Instagram → accepts 4K, serves 1080p max
- YouTube Shorts → 1080×1920 optimal

Accepting 4K but processing/serving max 1080p = correct strategy. Above 1080p = no visible benefit for social content, just wasted storage.

Platform max resolution reference (2026):

Platform	Max Resolution	Aspect	Max File Size
TikTok	1080×1920	9:16	287MB mobile / 4GB web
Instagram Reels	1080×1920	9:16	4GB
YouTube Shorts	1080×1920	9:16	—
Facebook Reels	1080×1920	9:16	1GB

Platform	Max Resolution	Aspect	Max File Size
Twitter/X	1280×1024	any	512MB

Safe zone for 9:16 reels (1080×1920):

```
┌──────────────────┐
│▒▒▒ top 120px ▒▒▒│ ← UI bar — avoid placing content here
│                  │
│   SAFE ZONE      │ ← faces, text, important content here
│   860×1550px     │
│                  │
│▒ right 180px ▒▒▒▒│ ← action buttons (like, comment, share)
│▒▒ bottom 250px ▒▒│ ← caption area
└──────────────────┘
```

Watermark placement must respect safe zones.

6.3 Long Video Pipeline (HLS)

Threshold: Duration ≥ 3 minutes → HLS

Same as short video EXCEPT:

```
Format → HLS adaptive streaming (not MP4)
No fast lane (too long, post stays PROCESSING)
Post goes live only when fully processed

Transcode to HLS:
  hls/360p/  → segments + 360p.m3u8
  hls/720p/  → segments + 720p.m3u8
  hls/1080p/ → segments + 1080p.m3u8
  hls/master.m3u8 → points to all quality manifests

Segment duration: 2 seconds
Codec: H.264, AAC audio
Container: MPEG-TS (.ts chunks)

Still generated:
  preview_3s.mp4     ← for feed inline autoplay
```

```
  thumbnail.webp
  LQIP, BlurHash, dominantColor
  og_clean.webp, og_play.webp, og_preview.mp4

isReelEligible: false
streamingFormat: HLS
```

HLS Cache-Control:

```
.ts chunks:    Cache-Control: public, max-age=31536000
master.m3u8:   Cache-Control: public, max-age=31536000
```

6.4 Digital Products Pipeline

Upload Processing

```
Seller uploads product file
  ↓
Validate:
  - File type allowed?
  - Seller quota OK?
  - Product exists + owned by seller?
  ↓
ClamAV scan (DOUBLE scan — extra safety)
  ↓
SHA-256 checksum generated + stored
  ↓
Light processing by file type:

  PDF:
    Extract page 1 → preview image (72 DPI)
    Watermark "PREVIEW" across image
    Store: preview/preview.webp

  Video course (MP4):
    Extract first 2 minutes
    Transcode to 480p
    Watermark burned in
    Store: preview/preview.mp4
```

```
  Audio (MP3/WAV/FLAC):
    Extract first 30 seconds
    Lower bitrate (96kbps)
    Store: preview/preview.mp3

  3D Models (GLB/OBJ/FBX):
    Render thumbnail from multiple angles
    Store: preview/thumb_*.webp

  Images (stock art, PNG):
    Resize to low resolution (600px max)
    "PREVIEW" watermark burned in
    Store: preview/preview.webp

  Software/ZIP:
    No preview file
    Cover image only

  ↓
Store → nexgate-digital/products/{shopId}/{productId}/{fileId}/
  original/ ← actual product (never exposed publicly)
  preview/  ← shown to everyone
  cover/    ← product listing thumbnail
  ↓
Update DB: READY
Publish to RabbitMQ: DIGITAL_PRODUCT_READY → Main Backend
```

Download Flow (Buyer)

```
Buyer clicks Download
  ↓
POST /digital/download { productId, orderId, fileId }
  ↓
File Thunder validates:
  [1] orderId exists in DB?
  [2] orderId belongs to requesting user?
  [3] Order status = PAID?
  [4] download_count < download_limit?
  [5] Current time < expires_at?
```

```
  [6] User account in good standing?
  ↓
All pass:
  Generate single-use signed URL:
    10-minute TTL
    Order bound
    Device bound
    Encrypted token
    Stored in Redis (single-use enforcement)
  ↓
Log download event:
  orderId, buyerId, timestamp
  IP address, deviceId, country
  ↓
Increment download_count
  ↓
Return signed URL to client
Client downloads (chunked, resumable via byte-range)
  ↓
URL marked as used in Redis → 403 on reuse
```

No forensic watermarking for now. Planned for future phase when piracy becomes a real problem.

6.5 DM Attachments Pipeline

Same core pipeline with different rules:

	Social Posts	DM Attachments
Bucket	nexgate-public	nexgate-private
CDN	✅ Cloudflare	❌ No CDN
Access check	Public/followers	Conversation membership
Variants	Full (all sizes)	Thumb + original only
Watermark	✅ On download	❌ Never
Reel pool	✅ If eligible	❌ Never
Virus scan	✅ Yes	✅ Yes
URL type	Permanent CDN	Signed 5min TTL

	Social Posts	DM Attachments
Processing	Full	Minimal

Access control for DM files:

```
Client requests DM file URL
  ↓
File Thunder checks:
  Is requesting user a member of this conversation?
  YES → generate signed URL (5min TTL)
  NO  → 403 Forbidden
```

---

7. Watermarking Strategy

When Applied

- **Processing time** (server-side) — NOT at serve time, NOT client-side
- Done ONCE when video is processed
- Stored as separate `_watermarked` variant

When Served

Scenario	Variant Served
Watching on NexGate (in-app/web)	Clean variant
Downloading via NexGate download button	Watermarked variant
Accessing via CDN URL directly	Clean variant (unavoidable)
Sharing via savefromnet-style tools	Clean variant (acceptable)

Watermark Position — Moving (TikTok Style)

```
Jumps between 4 corners every 3 seconds:
  0-3s  → top left
  3-6s  → top right
  6-9s  → bottom right
  9-12s → bottom left
```

```
  repeat for full video duration

Logo: 80×80px, 60% opacity
Username: directly below logo, same position

Why moving:
  Static corner → easy to crop out □
  Moving → appears everywhere → impossible to crop □
  TikTok confirmed uses this strategy
```

App-based Watermark

```
clientApp = NEXGATE_ANDROID → "NexGate" logo
clientApp = NEXGATE_IOS     → "NexGate" logo
clientApp = NEXGATE_WEB     → "NexGate" logo
clientApp = NEXGATE_LITE    → "NexGate Lite" logo
```

Determined at upload time. Burned at processing time. Never changes.

Pre-Watermarked Content (Uploaded from Other Platforms)

The Problem:

```
User downloads TikTok video (has TikTok watermark)
Re-uploads to NexGate
NexGate adds NexGate watermark on top
= two watermarks visible = bad UX □□
```

What major platforms do:

Platform	Approach
Instagram	Algorithm suppresses watermarked content from discovery
TikTok	No detection, just adds own watermark on top
YouTube Shorts	Suppresses competitor watermarked content

Instagram head Adam Mosseri confirmed: videos with competitor watermarks receive significantly reduced algorithmic reach.

NexGate approach by phase:

```
Phase 1 (launch — current):
  No watermark detection at all
  NexGate watermark burned on top of everything
  Pre-existing watermarks = overwritten/overlaid
  = zero complexity 
  = same pipeline for all videos 
  = double watermark acceptable at this stage 

Phase 2 (growth):
  Client-side warning before upload:
  "This video appears to have a watermark
   from another platform. Remove it for
   better reach on NexGate."
  User can proceed anyway
  = user educated, NexGate protected 

Phase 3 (scale):
  Server-side AI detection:
    Scan first frame for known platform logos
    Flag: hasExternalWatermark: true
    Suppress in reel pool (isReelEligible: false)
    Show post-level label: "Cross-posted content"
    = same approach as Instagram 
```

Note: NexGate never BLOCKS pre-watermarked uploads. Only suppresses reach in discovery/recommendations. Content still visible to uploader's followers.

Username Change Policy

Old videos keep old username watermark (same as TikTok). Re-watermarking = future premium feature.

---

8. File Protection Layers

Layer	Method	Stops
1	`Content-Disposition: inline` header	Casual right-click downloaders

Layer	Method	Stops
2	Dynamic URL via JS (never in HTML)	HTML source inspection
3	Signed expiring URLs (private content)	URL sharing between users
4	Session + device binding	Cross-device URL reuse
5	Disable right-click on player (frontend)	Non-technical users
6	Watermark burned on download	Casual sharing without credit
7	HLS chunking for video	Download and reassembly

Philosophy: Goal is not impossible to download. Goal is not worth the effort for 99% of users + traceable if they do.

---

9. URL Strategy

Public Content ? Permanent CDN URLs

```
Thumbnail:
https://media.nexgate.com/posts/usr_123/vid_789/thumb.webp

Video (public, clean):
https://media.nexgate.com/posts/usr_123/vid_789/720p_clean.mp4

OG preview (permanent):
https://media.nexgate.com/og/vid_789/og_preview.mp4

Profile picture:
https://media.nexgate.com/profiles/usr_123/avatar_400.webp
```

- No signing, no expiry
- Cache-Control: public, max-age=31536000
- Cloudflare CDN caches globally

Private Content ? Signed Expiring URLs

```
DM attachment:
https://media.nexgate.com/private/...?
  x-expires=1716305400
  &x-sig=hmac_signature
  &x-uid=usr_xyz
  &x-sid=sess_abc

Digital product download:
https://media.nexgate.com/digital/...?
  x-expires=1716305400
  &x-sig=hmac_signature
  &x-order=enc_orderId
  &x-device=device_abc
```

Signed URL Params Explained

Param	Purpose	Without It
`x-expires`	URL dies after TTL	Permanent link = unrevokable
`x-sig`	HMAC tamper protection	Anyone can forge expiry
`x-uid`	Bound to user session	URL shareable between users
`x-sid`	Bound to session	Works after logout

Who Generates vs Validates

- **Main Backend / File Thunder** → generates signed URL (knows secret key)
- **Cloudflare CDN** → validates signature at edge (configured with same secret)
- **MinIO** → never exposed to clients

TTL by Content Type

Content	TTL
Stream URL (video)	10 min
DM attachment	5 min
Digital product download	10 min (single-use)
Public CDN content	Permanent
OG preview video	Permanent

10. OG (Open Graph) Strategy

For Video Posts

```
<meta property="og:type" content="video.other" />
<meta property="og:url" content="https://nexgate.com/reels/reel_abc123" />
<meta property="og:title" content="@username on NexGate" />
<meta property="og:description" content="caption text..." />

<!-- Clean thumbnail (permanent CDN) -->
<meta property="og:image"
  content="https://media.nexgate.com/posts/usr_123/vid_789/thumb.webp" />
<meta property="og:image:width" content="1200" />
<meta property="og:image:height" content="630" />
<meta property="og:image:type" content="image/webp" />

<!-- Permanent OG preview video (NOT signed, NOT expiring) -->
<meta property="og:video"
  content="https://media.nexgate.com/og/vid_789/og_preview.mp4" />
<meta property="og:video:type" content="video/mp4" />
<meta property="og:video:width" content="360" />
<meta property="og:video:height" content="640" />

<meta property="og:site_name" content="NexGate" />

<!-- Twitter Card -->
<meta name="twitter:card" content="player" />
<meta name="twitter:image"
  content="https://media.nexgate.com/posts/usr_123/vid_789/thumb.webp" />
<meta name="twitter:player"
  content="https://nexgate.com/embed/reel_abc123" />
```

For Image Posts

```
<meta property="og:type" content="website" />
<meta property="og:image"
```

```
  content="https://media.nexgate.com/posts/usr_123/img_789/og.webp" />
<!-- No og:video tag = no play button shown -->
```

Play Button Behavior

Platform	Play Button Source
WhatsApp	Added by WhatsApp (reads og:type)
Telegram	Added by Telegram
Twitter/X	Added by Twitter
SMS/Email	Must burn into og:image (og_play.webp)
Unknown apps	Must burn into og:image (og_play.webp)

NexGate serves og_clean.webp to known platforms, og_play.webp to unknown platforms (detected via crawler User-Agent).

OG Video URL — Why Permanent

- WhatsApp/Telegram crawl on share, user opens hours later
- Signed URL expired = "Error loading video"
- Solution: permanent `og_preview.mp4` in CDN (360p, watermarked, small)
- Access control = server-side (private posts return 403)

---

11. Compression & Transcoding

Client-Side Intelligent Compression

Before upload, client analyzes:

```
Inputs:
  resolution, bitrate, codec
  network type (WiFi/4G/3G/2G)
  device tier (high/mid/low)

Decision:
  Already optimized + good network → upload direct
  Over-bitrated → compress
```

```
  Slow network → compress regardless

Target (if compressing):
  Max 1080p
  CRF 18 (light — preserve quality for server)
  H.264

Never upscale on client either.
```

Server-Side Adaptive Transcoding

After upload, FFprobe analyzes:

```
Extract: width, height, bitrate, fps, codec, duration

Decision tree:
  height ≥ 1080 → generate 1080p, 720p, 360p
  height ≥ 720  → generate 720p, 360p
  height ≥ 480  → generate 480p, 360p
  height < 480  → keep original resolution

Rules:
  NEVER upscale (never exceed input resolution)
  NEVER inflate (if output > input size → use original as-is: -c:v copy)
  CRF adapts to input quality
```

CRF Values (H.264)

Variant	CRF	Use
1080p	21	High quality
720p	23	Standard
480p	25	Medium
360p	28	Low / slow networks
OG preview	30	Very small file

Video Codec Strategy

Now	Future
H.264 (universal support)	AV1 (50% smaller, open source)
AAC audio 128kbps	Opus audio
MP4 container	CMAF container

Image Format Strategy

Input	Output
JPEG	WebP (quality 80%)
PNG	WebP (quality 85%)
HEIC	WebP (quality 80%)
GIF	WebP animated + MP4 loop

Critical FFmpeg Flags

```
-movflags +faststart  → metadata at start of file
                        = video starts playing before fully downloaded
                        = essential for web streaming

-movflags frag_keyframe+empty_moov+faststart  → fragmented MP4
                        = byte-range requests work
                        = reel prefetch works
```

Generation Loss (Double Compression)

- Client compresses lightly (CRF 18) → preserves quality
- Server compresses per variant (CRF 23-28) → optimizes delivery
- Two passes but first is near-lossless → acceptable tradeoff
- Benefit: fast uploads on Tanzania mobile networks

---

12. BlurHash, LQIP & Dominant Color

All three generated for every image and video thumbnail:

Dominant Color

```
ImageMagick resize to 1×1 pixel
→ average color of entire image
→ stored as hex: "#2D7BC8"
→ shown instantly as CSS background (0ms, 0 bytes extra)
```

BlurHash

```
Algorithm: DCT-based image encoding
Output: ~30 character string "LGF5]+Yk^6#M@-5c,1J5@[or[Q6."
Library: blurhash-java
Components: 4x3 (width x height)
→ decoded client-side to blurry placeholder
→ no network request
→ shows shape/composition of image
```

LQIP (Low Quality Image Placeholder)

```
ImageMagick resize to 10×10px
Quality: 20%
Convert to base64 string
→ embedded in feed JSON
→ client scales up (naturally blurry)
→ more accurate than BlurHash
→ no network request
```

Progressive Loading Stages

```
Stage 0 (instant, 0ms):  dominantColor CSS background
Stage 1 (instant, 0ms):  BlurHash decoded → blurry shape
Stage 2 (instant, 0ms):  LQIP scaled up → accurate blur
Stage 3 (CDN, 50-200ms): thumb.webp loads → sharp thumbnail
Stage 4 (on demand):     large.webp loads → full quality (on tap)
```

---

13. Deduplication

Hash-Based (Exact Match)

```
File uploaded
  ↓
Compute SHA-256 of raw file
  ↓
Check file_hashes table:
  HASH EXISTS + was CLEAN → skip processing entirely
                            create reference to existing variants
                            increment reference_count
  HASH NOT FOUND → process normally
                   save hash + object_key to file_hashes
```

Benefits

- Same viral video uploaded 1000x = stored once
- Known clean files skip ClamAV scan (hash cache)
- Known virus files rejected instantly (hash cache)
- Storage savings up to 99% for viral content

Reference Counting (Deletion)

```
User deletes file
  ↓
Decrement reference_count
reference_count > 0 → keep file (others reference it)
reference_count = 0 → delete from MinIO
```

Near-Duplicate Detection

- Exact SHA-256 only (now)
- Perceptual hashing (pHash) = future feature for copyright detection

---

14. Quota Management

Enforcement Point

At presigned URL generation — never after upload.

```
Request upload
  ↓
Atomic check:
  UPDATE user_storage_quota
  SET used_bytes = used_bytes + fileSize
  WHERE user_id = ?
  AND (used_bytes + fileSize) <= quota_bytes
  RETURNING used_bytes

  0 rows affected → quota exceeded → reject with 403
  1 row affected  → quota reserved → proceed
```

What Counts Against Quota

Content	Charged
Social posts	Original upload size only
Digital products	Actual stored bytes (all variants)

Plan Limits

Plan	Storage	Max Image	Max Video
FREE	1GB	20MB	200MB, 5min
PRO	20GB	20MB	1GB, 60min
BUSINESS	100GB	20MB	2GB, unlimited

Warning System

```
80% used → email + in-app notification
95% used → in-app banner
100% used → uploads blocked, existing content untouched
```

Quota Release

- On ABANDONED: immediate release (file never stored)
- On soft delete: NOT released (file still exists)

- On hard delete (30 days after soft): quota freed

Quota Cache (Redis)

```
GET quota:usr_123 → used_bytes (cached)
INCRBY quota:usr_123 fileSize → atomic increment
Sync to PostgreSQL every 5 minutes
```

---

15. Storage Lifecycle

nexgate-raw

```
Auto-delete: 24 hours (MinIO lifecycle policy)
Incomplete multiparts: aborted after 24 hours
```

nexgate-public (Social Content)

```
All variants: keep FOREVER (never delete)
Move to cold storage tier based on access frequency:
  Hot  (< 30 days or high views)   → NVMe SSD
  Warm (30-365 days, moderate)     → HDD
  Cold (1yr+, rare access)         → Cold object tier

CDN serves from cache after first request
MinIO barely hit after content warms up

Smart lifecycle (access-based, not just age):
  < 100 views/day for 30 days → move to warm
  < 10 views/day for 90 days  → move to cold
  Viral spike on old content  → auto-promote back to hot
```

nexgate-private

```
DM attachments: 1 year
Audio temp files (for Rec Engine): 24 hours OR deleted by Rec Engine after transcription
```

```
KYC documents: per legal requirement
```

nexgate-digital

```
Original product files: FOREVER (seller's product)
Preview files: FOREVER
Cover images: FOREVER
```

Deleted Content (Soft Delete)

```
User deletes post
  ↓
DB: soft delete (mark deleted, keep record)
  ↓
30-day grace period:
  User can appeal/restore
  Legal holds possible
  CDN cache still valid (purge takes time)
  ↓
30 days → hard delete from MinIO
            Cloudflare cache purge
            DB record updated
            Quota freed
```

---

16. Abandoned Upload Handling

The Problem

```
File uploaded to nexgate-raw □
Client never sends POST /media/confirm □
Reasons: app crash, network drop, user closed app
→ file stuck in raw bucket = wasted storage
```

Three-Layer Solution

Layer 1 — Client Confirm (Primary fast path)

```
Client sends POST /media/confirm after upload
→ processing triggered immediately
→ best UX (instant)
```

Layer 2 — Cleanup Job (Safety net, every 30 min)

```
// Runs every 30 minutes
// Finds PENDING records > 1 hour old
// Checks if file exists in MinIO

File exists → RECOVER (upload done, confirm missed)
  → update status UPLOADED
  → trigger processing
  → Tanzania network drop scenario handled 

File not exists → ABANDONED (user closed app)
  → update status ABANDONED
  → release quota reservation
```

Layer 3 — Raw Bucket Lifecycle (Final safety)

```
nexgate-raw auto-delete: 24 hours
Incomplete multiparts aborted: 24 hours
= catches anything missed by layers 1 + 2
```

Cleanup Job Scaling

```
Add DB index:
  CREATE INDEX idx_media_cleanup
  ON media_files (created_at)
  WHERE status = 'PENDING'

Process in batches of 100
Parallel MinIO checks (parallelStream)
Redis distributed lock (prevent overlap):
  SET lock:cleanup_job "running" NX EX 2100
```

fileId in Object Key (Enables Recovery)

```
objectKey = uploads/{userId}/{fileId}/original.mp4
                                ↑
                          fileId encoded in path
→ cleanup job extracts fileId from objectKey
→ can recover without client confirm
```

---

17. Progress Tracking (SSE)

Why SSE (not polling, not WebSocket)

	Polling	SSE	WebSocket
Direction	Both	Server→Client only	Both
Connection	New each time	Persistent	Persistent
Auto-reconnect	Manual	Built-in □	Manual
Complexity	Medium	Simple	Complex
Tanzania networks	Wasteful	Resilient □	Complex

Upload progress = one direction only → SSE is perfect fit.

Status Flow

```
Redis key: upload_status:{fileId}

PENDING      → "Upload requested"
UPLOADING    → "Uploading..."
UPLOADED     → "File received"
SCANNING     → "Checking file..."
PROCESSING   → "Processing video..."
LIVE_PARTIAL → "Almost ready!" + { available: ["360p"] }
READY        → "Your post is live!" + { liveUrl: "..." }
FAILED       → "Something went wrong"
QUARANTINED  → "File failed security check"
ABANDONED    → "Upload timed out"
```

Spring Boot SSE

```
@GetMapping(
  value = "/media/{fileId}/progress",
  produces = MediaType.TEXT_EVENT_STREAM_VALUE
)
SseEmitter trackProgress(@PathVariable String fileId) {
  SseEmitter emitter = new SseEmitter(7_200_000L); // 2hr
  emitterRegistry.register(fileId, emitter);
  // Send current status immediately
  emitter.send(redis.get("upload_status:" + fileId));
  return emitter;
}
```

Multi-Instance Scaling

```
File Thunder (Server A) updates Redis
File Thunder (Server B) has SSE connection
→ Redis Pub/Sub bridges them:
  PUBLISH media_status:{fileId} { status: "READY" }
  Server B receives → finds local emitter → pushes to client
```

---

18. Communication Architecture

Rule: No Direct HTTP Between Services

```
 No REST API calls between File Thunder and Main Backend
 No webhooks
 RabbitMQ for internal operations
 Kafka for event streaming to external services
```

RabbitMQ (File Thunder ? Main Backend)

Exchange: `nexgate.media` (topic exchange)

Routing Key	Direction	Consumer
`media.upload.request`	MB → FT	File Thunder
`media.upload.url.ready`	FT → MB	Main Backend

Routing Key	Direction	Consumer
`media.ready`	FT → MB	Main Backend
`media.live.partial`	FT → MB	Main Backend
`media.failed`	FT → MB	Main Backend
`media.quarantined`	FT → MB	Main Backend
`digital.product.ready`	FT → MB	Main Backend

Example MEDIA_READY event:

```
{
  "event": "MEDIA_READY",
  "fileId": "vid_789",
  "ownerId": "usr_123",
  "type": "SHORT_VIDEO",
  "status": "READY",
  "variants": {
    "360p_clean": "posts/usr_123/vid_789/360p_clean.mp4",
    "720p_clean": "posts/usr_123/vid_789/720p_clean.mp4",
    "thumbnail": "posts/usr_123/vid_789/thumb.webp",
    "blurhash": "LGF5]+Yk^6#M@-5c,1J5@[or[Q6.",
    "lqip": "...",
    "dominantColor": "#1a1a2e",
    "preview_3s": "posts/usr_123/vid_789/preview_3s.mp4",
    "og_clean": "posts/usr_123/vid_789/og_clean.webp",
    "og_preview_video": "og/vid_789/og_preview.mp4"
  },
  "isReelEligible": true,
  "streamingFormat": "MP4",
  "duration": 28,
  "aspectRatio": "9:16"
}
```

Kafka (File Thunder ? Rec Engine + Analytics)

Topic: `content.new`

```
{
  "event": "FILE_READY",
  "fileId": "vid_789",
```

```
  "type": "SHORT_VIDEO",
  "ownerId": "usr_123",
  "media": {
    "duration": 28,
    "aspectRatio": "9:16",
    "qualityScore": 75,
    "isReelEligible": true,
    "hasAudio": true,
    "streamingFormat": "MP4"
  },
  "userProvided": {
    "caption": "Check this Spring Boot setup",
    "hashtags": ["#tech", "#java", "#springboot"],
    "location": null
  },
  "audioRef": {
    "bucket": "nexgate-private",
    "objectKey": "audio/vid_789.wav",
    "expiresAt": "2026-05-23T10:00:00Z"
  },
  "thumbnailUrl": "https://media.nexgate.com/...",
  "processedAt": "2026-05-22T10:00:00Z"
}
```

Rec Engine consumes this, fetches audio.wav, runs Whisper transcription, classifies content, deletes audio.wav.

---

19. CDN Strategy

How It Works

```
First request (cache miss):
  Client → Cloudflare edge → MinIO → Cloudflare caches → serves

Every subsequent request (cache hit):
  Client → Cloudflare edge → serves from cache
  MinIO never touched
```

Cache-Control Headers

Content	Cache-Control	TTL
Post thumbnails	public, max-age=31536000	1 year
Video MP4 variants	public, max-age=31536000	1 year
HLS .ts chunks	public, max-age=31536000	1 year
HLS .m3u8 manifests	public, max-age=31536000	1 year
OG images	public, max-age=31536000	1 year
OG preview video	public, max-age=31536000	1 year
Profile pictures	public, max-age=86400	1 day
API responses	no-store	Never cached
Private content	private, no-store	Never cached

Cloudflare Phases

Phase	Setup	Cost
Now (launch)	Cloudflare Free	$0
Growth	Cloudflare Pro	$20/month
Scale	Migrate to Cloudflare R2 (zero egress)	Pay storage only

MinIO — Never Publicly Exposed

```
Public access: BLOCKED
Only Cloudflare IPs can reach MinIO
Client → media.nexgate.com (Cloudflare) → MinIO (internal)
MinIO URL never seen by clients
```

---

20. Shareable Links

Share URL Format

```
Generated client-side (no server call):
  nexgate.com/reels/reel_abc123?ref=whatsapp
```

```
ref values:
  whatsapp, telegram, twitter, sms, copy, email
```

Follow Prompt (On Link Click)

```
Sarah taps share link
→ NexGate loads post
→ Server looks up post owner from DB (postId → authorId)
→ Shows follow prompt for correct creator
→ No uid in URL (no manipulation possible)
→ No token needed (public content)
```

Analytics from ref Parameter

```
Track per share:
  Which platform shared from
  Click-through rate per platform
  Follow conversion rate per platform
  New signups from shares
  Watch completion rate from shares
```

---

21. Technology Stack

Component	Technology
Language	Java 21 (Spring Boot 3.x)
Video processing	FFmpeg (global install) via Jaffree
Image processing	ImageMagick (global install) via IM4Java
BlurHash	blurhash-java library
Virus scanning	ClamAV (Docker container, TCP :3310)
Object storage	MinIO (S3 compatible)
CDN	Cloudflare
Message queue (ops)	RabbitMQ
Event streaming	Kafka

Component	Technology
Cache / progress	Redis
Database	PostgreSQL
Resumable uploads	TUS protocol
Secrets	HashiCorp Vault (vault.qbitspark.com)
Container	Docker + Docker Compose
Progress delivery	SSE (Server-Sent Events)

Why FFmpeg + ImageMagick Global Install

```
Both are CLI tools (not daemons)
Called on demand via ProcessBuilder
Jaffree/IM4Java call /usr/bin/ffmpeg and /usr/bin/convert
Same filesystem = works perfectly
No network overhead
Simpler than separate containers

ClamAV = separate Docker container because:
  Runs as daemon (long-running process)
  Official Docker image exists
  Connects via TCP = natural for Docker
  Auto-updates virus definitions
```

---

22. Database Schema

```
-- Core media file record
media_files
  file_id           UUID PK
  owner_id          UUID          -- accountId
  directory         ENUM          -- PROFILE, POSTS, STORIES, EVENTS, SHOPS, MESSAGES,
PRODUCTS
  original_name     TEXT
  object_key        TEXT          -- raw upload path in nexgate-raw
  mime_type         TEXT
  file_size         BIGINT
  status            ENUM          -- PENDING, UPLOADING, UPLOADED, SCANNING, PROCESSING,
```

```
                                -- LIVE_PARTIAL, READY, FAILED, QUARANTINED, ABANDONED,
EXPIRED
  variants          JSONB         -- all processed variant paths
  metadata          JSONB         -- dimensions, duration, fps, codec
  scan_result       TEXT          -- "clean" or virus name
  hash              TEXT          -- SHA-256 for deduplication
  is_reel_eligible  BOOLEAN
  streaming_format  ENUM          -- MP4, HLS
  created_at        TIMESTAMPTZ
  ready_at          TIMESTAMPTZ

-- Content signals for Rec Engine
media_content_signals
  file_id           UUID FK
  has_audio         BOOLEAN
  audio_ref         TEXT          -- nexgate-private/audio/{fileId}.wav
  dominant_color    TEXT          -- hex color
  blurhash          TEXT
  lqip              TEXT          -- base64
  aspect_ratio      TEXT          -- "9:16", "16:9", "1:1", "4:5"
  quality_score     INT           -- 0-100
  width             INT
  height            INT
  duration_seconds  DECIMAL

-- User storage quota
user_storage_quota
  user_id           UUID PK
  plan              ENUM          -- FREE, PRO, BUSINESS
  quota_bytes       BIGINT
  used_bytes        BIGINT
  file_count        INT
  updated_at        TIMESTAMPTZ

-- Per bucket breakdown
user_storage_breakdown
  user_id           UUID
  bucket            TEXT
  used_bytes        BIGINT
  file_count        INT
```

```
  updated_at        TIMESTAMPTZ

-- Deduplication hash index
file_hashes
  hash              TEXT PK       -- SHA-256
  object_key        TEXT          -- canonical storage path
  file_size         BIGINT
  mime_type         TEXT
  reference_count   INT
  created_at        TIMESTAMPTZ

-- Variant storage class tracking
media_variants
  file_id           UUID FK
  quality           TEXT          -- "360p", "720p", "1080p", "thumb", etc
  status            ENUM          -- HOT, WARM, COLD, FROZEN
  storage_class     TEXT
  object_key        TEXT
  file_size         BIGINT
  last_accessed     TIMESTAMPTZ
  access_count      INT

-- Digital products
digital_product_files
  file_id           UUID PK
  product_id        UUID FK
  file_type         ENUM          -- MAIN, PREVIEW, COVER
  original_name     TEXT
  mime_type         TEXT
  file_size         BIGINT
  checksum          TEXT          -- SHA-256
  object_key        TEXT
  status            ENUM          -- PROCESSING, READY, FAILED
  created_at        TIMESTAMPTZ

-- Digital orders
digital_orders
  order_id          UUID PK
  product_id        UUID FK
  buyer_id          UUID FK
```

```
  seller_id         UUID FK
  amount_paid       DECIMAL
  currency          TEXT
  status            ENUM          -- PAID, REFUNDED, DISPUTED
  download_count    INT DEFAULT 0
  download_limit    INT
  expires_at        TIMESTAMPTZ
  purchased_at      TIMESTAMPTZ

-- Download audit log
digital_download_logs
  log_id            UUID PK
  order_id          UUID FK
  buyer_id          UUID FK
  file_id           UUID FK
  downloaded_at     TIMESTAMPTZ
  ip_address        TEXT
  device_id         TEXT
  user_agent        TEXT
  country           TEXT
  download_number   INT           -- which download (1st, 2nd, etc)
```

Critical indexes:

```
-- Cleanup job performance
CREATE INDEX idx_media_cleanup
ON media_files (created_at)
WHERE status = 'PENDING';

-- Hash lookup for deduplication
CREATE UNIQUE INDEX idx_file_hashes_hash
ON file_hashes (hash);

-- Owner lookup
CREATE INDEX idx_media_files_owner
ON media_files (owner_id, status, created_at DESC);
```

---

23. Docker & Infrastructure

File Thunder Dockerfile

```
FROM eclipse-temurin:21-jdk-alpine

# Install FFmpeg (global, called by Jaffree)
RUN apk add --no-cache ffmpeg

# Install ImageMagick with WebP support (called by IM4Java)
RUN apk add --no-cache \
    imagemagick \
    imagemagick-webp \
    imagemagick-heic

# Copy Spring Boot jar
COPY target/file-thunder.jar app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Docker Compose (Relevant Services)

```
file-thunder:
  build: ./file-thunder
  ports: ["8081:8081"]
  environment:
    SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/nexgate
    SPRING_REDIS_HOST: redis
    RABBITMQ_HOST: rabbitmq
    KAFKA_BOOTSTRAP_SERVERS: kafka:9092
    MINIO_ENDPOINT: http://minio:9000
    CLAMAV_HOST: clamav
    CLAMAV_PORT: 3310
    VAULT_ADDR: https://vault.qbitspark.com
  depends_on: [postgres, redis, rabbitmq, kafka, minio, clamav]

clamav:
  image: clamav/clamav:stable
  ports: ["3310:3310"]
  volumes:
```

```
    - clamav_data:/var/lib/clamav
  # Auto-updates virus definitions via freshclam
```

VPS Resource Allocation

```
File Thunder: 4 cores, 16GB RAM (FFmpeg is CPU intensive)
ClamAV:       1 core,  2GB RAM
```

---

24. What File Thunder Does NOT Do

```
❌ ML / AI content classification
❌ Speech-to-text (Whisper) — that's Rec Engine
❌ Image visual analysis (CLIP) — that's Rec Engine
❌ Feed computation or ranking
❌ Social graph operations (follow/unfollow)
❌ Payment processing
❌ User authentication / authorization
❌ Push notifications
❌ Search indexing
❌ Business logic (post creation, comments, likes)
❌ Fan-out to followers
❌ Recommendation logic
```

File Thunder extracts audio.wav and stores temporarily. The Rec Engine fetches it, runs Whisper, classifies content, and deletes the audio file. File Thunder never knows what's in the audio.

Summary — File Thunder in One Sentence

> File Thunder receives raw files, validates and secures them via ClamAV, processes them into optimized delivery variants using FFmpeg and ImageMagick, stores everything in MinIO across four isolated buckets, serves public content efficiently via Cloudflare CDN, tracks progress via SSE and Redis,

> communicates with the Main Backend via RabbitMQ, and publishes content signals to Kafka for downstream services — without ever performing any business logic, ML, or social operations.

File Thunder Architecture Guide v1.0 — NexGate / QBIT SPARK
Document compiled: May 2026