ALLIED BizTech SOLUTIONS

FIELD GUIDE · 2026

Rebuild Cloudinary yourself.

The plan we follow.

For teams paying $89 to $3,000+ a month on Cloudinary who want to own their image URLs, transforms and delivery, and stop paying per credit.

- → Is it worth it, at your size
- → What to rebuild and what to drop
- → The architecture, in three layers
- → Migration and go-live checklists
- → The asset model
- → A week-by-week build plan
- → Transforms, caching, URLs
- → Where teams get stuck

Cloudinary bills in credits. *AWS bills in gigabytes.*

A Cloudinary credit is roughly one transformation, 1GB delivered or 1GB stored, bundled into tiers that hide what you pay for. A rebuild pays S3, CloudFront and a Cloudflare Worker at their published prices. Transformations make the gap: Cloudinary's per-credit price is about 10× the underlying Worker or Lambda cost.

USAGE A MONTH	CLOUDINARY A YEAR	REBUILT: YEAR ONE / AFTER	SAVED IN YEAR ONE
Plus · 75K credits · 5TB delivered	~$1,068	~$12,140 / $540	Stay on Cloudinary
Advanced · 225K credits · 15TB	~$3,348	~$13,040 / $1,440	Stay on Cloudinary
Custom · 750K credits · 50TB	~$11,000+	~$15,200 / $3,600	Pays back in 19 months
Custom · 2M credits · 150TB	~$28,000+	~$20,000 / $8,400	~$8,000
Enterprise · 5M+ credits	~$60,000+	~$26,600 / $15,000	~$33,400

Cloudinary list prices, May 2026. "Rebuilt" is S3 ($0.023/GB stored), CloudFront ($0.085/GB delivered for the first 10TB, less after) and Cloudflare Workers ($0.50 per million requests), all-in, plus our fixed build price of $11,600. If your own team builds it, put your engineering cost in that place.

THE RULE OF THUMB

Below about 750K credits a month, stay on Cloudinary and negotiate: the build does not pay back. At 750K it pays back in 19 months. **At 2M credits a month, the rebuild pays back in 8 months, then ~$19K a year stays with you: ~$47K saved over three years.** At 5M+, year one alone saves ~$33K.

Who you need

- **A senior engineer** at home with S3, CDNs, edge workers and image processing (sharp or imgproxy). Our team ships this in 3–4 weeks; one engineer working between other priorities should plan for longer.
- **Your front-end or ecommerce lead**, two hours a week: the URL patterns in use, the pilot category, the go-live call.
- **An owner after launch**: cache tuning, storage lifecycle, cost alerts, and on-call.

Rebuild the 15 transforms your URLs use. *Drop the rest.*

Cloudinary advertises 100+ transformation operators. Your front-end uses about 15. Those are the project, with URL syntax that matches; the long tail is where rebuilds stall.

REBUILD ONE FOR ONE

- **Resize, crop, fit, fill, pad**: `w_`, `h_`, `c_fill`, `c_fit`, `c_pad`, identical params
- **Format negotiation**: `f_auto` serves WebP or AVIF from the Accept header
- **Quality auto**: `q_auto`, on par with Cloudinary's
- **DPR and responsive images**: `dpr_auto`, srcset, art direction with the picture element
- **Watermark overlays**: `l_text`, `l_image`
- **Uploads**: multipart to S3 with progress, signed URLs, resumable for large video
- **Signed URL access control** for paid or gated assets: HMAC signing with a TTL

REBUILD, SIMPLER

- **Smart crop** (`g_auto`): face and saliency detection with OpenCV or SAM2, in the Worker or at ingest
- **Background removal and generative fill**: remove.bg or SAM2; DALL·E or Flux Pro, direct
- **Auto-tagging and AI search**: CLIP embeddings
- **Video**: thumbnails and HLS through MediaConvert or Mux

DROP ON PURPOSE

- **100+ operators your URLs never use**
- **MediaFlow and creative review**: Frame.io or Filestage
- **Full DAM, player analytics, moderation**: Bynder, Mux Data, Rekognition

ADD WHAT PER-CREDIT PRICING HIDES

- A transform budget per URL prefix, with an alert at 80%
- Direct S3 access for Athena, Bedrock and more

Three layers. *URL-compatible with Cloudinary.*

Your front-end keeps the same URL pattern, `/w_400,h_400,c_fill/asset.jpg`. A Cloudflare Worker reads it, transforms the original from S3 and caches the result at the edge. You swap a domain; the front-end does not change.

LAYER 1 · STORAGE

S3, your bucket

- Originals in S3 Standard
- Versioning and lifecycle rules
- Cross-region replication if you need it
- Asset metadata in Postgres
- Upload API: direct PUT or signed URL

LAYER 2 · TRANSFORM

Cloudflare Worker

- Parses Cloudinary's URL params
- Resize, crop, quality, format via imgproxy or sharp
- Caches each result at the edge
- ~$0.50 per million requests, 95%+ cache hits
- The URL grammar your team knows

LAYER 3 · DELIVERY

Cloudflare or CloudFront

- A global CDN
- WebP and AVIF negotiation
- Brotli and Gzip on metadata
- HTTP/3 and 0-RTT
- Per-asset analytics in your dashboard

Decisions to make in week one

DECISION	OUR DEFAULT	WHY
Where it runs	Storage in your AWS, the Worker in your Cloudflare	Your bill, your assets; AWS credits apply
Image domain	Your own, such as `images.yourbrand.com`	The last URL migration you will ever do
URL grammar	Cloudinary's, parsed in the Worker	Swap a domain, not the front-end
Transform engine	imgproxy or sharp, behind the Worker	Change the engine later without touching a URL
Cache key	Normalized params plus the asset version	High hit rates and no purges
Failover	Multi-region S3 and multi-CDN	The origin stays in your control

Cloudinary runs on AWS too: when AWS has a bad day, so does Cloudinary. On your own accounts you get the same uptime profile, with the origin under your control. Optional fourth layer: an asset dashboard for marketing on the same Postgres metadata.

S3 holds the files. *Postgres knows what they are.*

Model these before the migration starts. Keep Cloudinary's `public_id` as the S3 key, so every old URL maps to its file one for one and the migration can run twice without duplicates.

TABLE	KEY COLUMNS
`assets`	id, public_id, s3_key, folder, resource_type (image, video, raw), format, bytes, width, height, sha256, version, created_at, deleted_at
`asset_meta`	asset_id, alt_text, focal_x, focal_y, credit, attributes (jsonb)
`asset_tags`	asset_id, tag, source (person, CLIP), created_at
`named_transforms`	name (Cloudinary's `t_` name), params, owner, updated_at
`uploads`	id, asset_id, method (signed PUT, multipart), uploader, status, started_at, completed_at
`signing_keys`	key_id, secret reference, ttl_seconds, active, rotated_at
`transform_budgets`	url_prefix, monthly_limit, used, alert_at (80%), owner
`usage_daily`	date, url_prefix, requests, cache_hit_ratio, transforms, bytes_out
`migration_log`	public_id, cloudinary_version, s3_key, bytes, checksum_match, status, migrated_at

TWO RULES THAT SAVE A REBUILD

Put the version in the URL and in the cache key: a replaced image gets a new URL, so nothing stale is ever served and you never purge a CDN. **Store originals once and derive everything else**: transformed variants live in the cache, not in S3, so storage grows with your catalog, not with your breakpoints.

Four weeks, *pilot included.*

This is the order we build in. Each week ends with something your front-end lead can load in a browser, and the last week is a pilot on a single product category before the full cutover.

WEEK	BUILD	DONE WHEN
1	**Inventory and storage.** Collect the URLs in use from CDN logs and a crawl; list the operators and named transformations they call. S3 bucket, versioning, the metadata tables. Start the background sync from Cloudinary's bulk download API.	Every operator in use is listed; the sync is running
2	**The Worker.** The URL parser for the operators on your list, imgproxy or sharp behind it, `f_auto` and `q_auto`, the edge cache and its key, signed URLs.	Your sample of production URLs renders the same on both
3	**Uploads and front-end.** The upload API and signed-URL helpers; helpers for Next/Image, Shopify Liquid or WooCommerce. Budgets per prefix, monitoring, failover, a load test, the delta sync.	Counts and checksums match Cloudinary
4	**Pilot and cutover.** One product category on the new domain for a week, real-user latency before and after, a daily fix list, then the domain swapped everywhere.	The pilot category is as fast or faster

AFTER CUTOVER

Keep Cloudinary **serving until your renewal date**, then cancel. Image URLs live in sent emails, marketplace feeds and social posts you cannot edit; give them time to age out, and nobody is in a hurry.

Every image URL is an API call. *Make each one cheap.*

Resizing an image is a solved problem. The real work is a URL parser that matches Cloudinary's exactly, a cache that answers almost every request, and limits that stop anyone from generating variants you pay for.

1. **Parse the grammar exactly.** Components are separated by slashes and params by commas; chained components apply in order; the version segment and named transformations (`t_`) come along. Build a test corpus from real production URLs and compare dimensions, format and a visual diff against Cloudinary.
2. **Normalize before you cache.** Sort params inside each component and fill in defaults, so `w_400,h_400` and `h_400,w_400` share one cache entry. Put the asset version in the key. That is how the cache-hit rate stays above 95%.
3. **Cap what the world can ask for.** An open transform URL lets anyone request endless variants, at your cost. Allow-list sizes or named transforms, sign URLs that carry arbitrary params, and set a budget per URL prefix with an alert at 80%.
4. **Get format negotiation right.** `f_auto` reads the Accept header. Put the chosen format in the cache key, or a browser without AVIF support receives an AVIF from cache.
5. **Keep heavy work off the edge.** Edge workers have memory and CPU limits. Run video, very large originals and smart crop at ingest or on a regional worker, and cache the result.

Measure the saving

Track the cache-hit ratio, transform p95, origin requests and bytes delivered per URL prefix, next to the AWS and Cloudflare bills. Those numbers prove the saving and show which page to fix first.

Own the domain

Serve images from your own domain from day one. The next time you change engine or CDN, no URL changes with it.

Move it, launch it, *run it.*

Migration

- ☐ Collect the URLs in use from CDN logs and a crawl
- ☐ List named transformations and eager transforms
- ☐ Keep `public_id` as the S3 key, with the version
- ☐ Sync in the background while Cloudinary keeps serving
- ☐ Carry alt text, tags and metadata across
- ☐ Check counts and checksums, then run a delta at cutover
- ☐ Render a sample of real URLs on both and compare
- ☐ Switch eager transforms to on-the-fly with caching

Go-live

- ☐ The domain swapped in every site, app and template
- ☐ Signing secrets in the Worker's secrets, not in code
- ☐ Budgets per prefix live, with alerts
- ☐ Failover tested: second region, second CDN
- ☐ A restore from versioned S3 proven
- ☐ Rollback ready: the Cloudinary domain one config change away

Running it

- ☐ **Always on:** error and cache-hit alerts, budget alerts at 80%
- ☐ **Weekly:** top prefixes by transforms and bytes, new URL patterns
- ☐ **Monthly:** imgproxy or sharp updates through staging, edge cache tuning, lifecycle rules
- ☐ **Quarterly:** cost review, the AWS and Cloudflare bills against the Cloudinary bill you replaced
- ☐ **Yearly:** a restore drill, a key rotation and an access review

Who owns it

Name one person before launch. Every product page loads through this pipeline: someone has to answer when images break at 2 a.m. or a new template starts minting variants.

Five ways a rebuild *stalls.*

1. Switching to another wrapper

ImageKit is about 30% cheaper than Cloudinary today, with the same wrapper economics and a pricing path that tracks Cloudinary's. imgix is competitive at low volume; above 100K images delivered a day it reaches Cloudinary-tier pricing, with the same lock-in.

2. A CDN on its own

Bunny is genuinely cheap and worth using as the CDN layer. It solves about 30% of what Cloudinary does: you still need the transforms, the asset tooling and the migration.

3. The $5K freelance build

It gets you an S3 bucket and a basic worker. It does not get you URL compatibility, an edge caching strategy, format negotiation, migration tooling or production debugging, which is where the value is.

4. Self-hosting imgproxy and calling it done

imgproxy is excellent; it is what we run under the hood. Wiring it up with cache, CDN, S3, monitoring and CI is 3–5 weeks of senior engineering, and the running of it is yours from then on.

5. Leaving transform URLs open

Without allow-lists, signing and budgets, one crawler or one new template mints thousands of variants. The bill moves from Cloudinary to your Worker and your CDN.

And one that is not a mistake: negotiating

Vendr and Tropic typically get 10–20% off Cloudinary annual contracts. Do it if the rebuild is a year away; it captures about an eighth of what a rebuild does, and the pressure resets every renewal.

RATHER HAVE IT BUILT?

We build this exact plan. *Fixed price.*

The same scope, architecture and four weeks, delivered by a senior team that has shipped 250+ products.

$11,600 fixed: a senior engineer, a coder and a half-time delivery manager for 3½ weeks, agreed before we start; video +$4,200–$6,600, asset dashboard +$2,500–$4,200, AI assets +$5,000–$7,500

Live in 3–4 weeks, with a one-week pilot on a single product category first

---

URL-compatible: your front-end keeps its URLs; assets move in about 48 hours for 1–3M, with no downtime

---

90-day warranty, and the code is in your GitHub from day one

We can run it for $1,800 a year: Worker monitoring, edge cache tuning and storage lifecycle

alliedbiz.tech/rebuild/cloudinary-alternative

Allied BizTech · 15+ years · ISO 9001:2015 · 250+ products shipped. Cloudinary is a trademark of its owner, named here for identification only; this guide is independent and not endorsed by Cloudinary. Prices as of May 2026.