The Layers Under a Platform: Infrastructure Practice for Business Platforms

Mohammed Irfan, Ajmal Abubacker, Avinash KS, Elango and Avinash Nishant

Cloudnix Software Labs, Bengaluru, India

CNX-TR-2026-10 · Version 1.0 · November 2026

Abstract

Business platforms rarely fail in their code; they fail in layers nobody owned: a cache that serves stale prices, a database whose slow queries no one reads, an index that drifts from its source, messages that land in spam, and restores done in the wrong order. We set out the infrastructure practice we apply to the commerce, workflow, SaaS and AI platforms we design, build and operate, layer by layer: regions and clouds, isolation, content delivery and object storage, media, caching, web and application servers, databases, vector search and on-demand model servers, scheduled operations, monitoring, staging and releases, messaging across channels, and recovery order. For each layer we state the problem, our practice, the settings we use in production and the best practices we recommend.

Keywords: *infrastructure; multi-cloud; CDN; object storage; caching; web performance; database tuning; vector search; GPU; scheduled jobs; monitoring; observability; staging; transactional messaging; disaster recovery*

Contents

1. Executive summary
2. Regions and clouds
3. Isolation by platform
4. Content delivery and object storage
5. Image and media pipeline
6. Layered caching
7. Web and application servers
8. Databases
9. Vector search and model servers
10. Scheduled operations: the night shift
11. Monitoring, metrics and logs
12. Staging and releases
13. Messaging across channels
14. Backups and recovery order
15. A checklist for your own platform
16. References

1 Executive summary

A business platform is judged by its pages, but it lives or dies by the layers under them. A page that loads in a second on a quiet morning and times out on a sale day; an order email that lands in spam; a vector index that drifts from the database it describes; a backup that restores, but in an order that leaves checkout down for hours. None of these is a coding error. Each is an infrastructure layer that nobody owned.

This report sets out the infrastructure practice we apply to the platforms we design, build and operate: multi-tenant commerce, workflow and SaaS platforms, and AI systems with retrieval and model serving. It covers thirteen layers, from regions and clouds to recovery order, and for each one states the problem, the practice we follow, the settings we use in production, and the checks we run. Figures are taken from our own servers.

Principles.

1. **Every layer has an owner and a number.** If nobody can say what a layer is set to, nobody is running it.
2. **Separate what fails separately.** One platform per server group; staging apart from production.
3. **Serve from the closest, cheapest layer that is correct.** Edge before cache, cache before database, but never stale where money or identity is involved.
4. **Size to the machine, not the default.** Defaults are written for nobody in particular.
5. **Watch for slowness, not only for failure.** Slow queries and full disks become outages.
6. **Do the housekeeping every night.** Clutter is a performance problem with a delay.
7. **Plan recovery in business order.** Restore what lets customers buy first.

Layer	What goes wrong without it	Our practice in one line
Regions and clouds	One provider's outage or price change is yours	Three compute regions on two clouds
Isolation	One platform's spike is everyone's outage	Servers per platform; separate staging
CDN and storage	Slow pages far from the server; costly spikes	Object storage behind a CDN with edges near users
Media	Heavy, shifting pages on phones	WebP in three sizes, lazy loading, reserved space
Caching	Every visit hits the database; or stale prices	Five layers, with a list of what is never cached
Web and application	Workers run out when traffic doubles	Event-driven server, HTTP/2, sized worker pools
Databases	Slow queries unnoticed; no way to rewind	Sized buffers, slow-query log, binary log
Vector and model servers	Stale answers; idle GPUs billed by the hour	Synced index, daily validation, on-demand GPUs
Scheduled jobs	Clutter slows the platform; feeds go stale	A night shift before 4 am; reminders every 2 h
Monitoring	Customers report problems first	Metrics, application tracing, uptime checks, daily log review
Staging and releases	Releases break checkout; forks per tenant	Staging copy of each platform; one release for all tenants
Messaging	OTPs and orders in spam; missed approvals	Routing by type across email, mobile, WhatsApp, Slack and in-app
Recovery	Restores in the wrong order	A tested restore order: sign-in, catalogue, checkout, orders

2 Regions and clouds

Region choice decides latency for users, the jurisdiction of data, and exposure to any one provider. We run three compute regions on two clouds, each chosen for the workload it hosts.

Region	Cloud	Workloads
Bangalore	Cloud B	Commerce, workflow and web platforms; staging copies
Singapore	Cloud A	AI application tier, vector search, object storage
San Francisco	Cloud B	Operations; GPU model servers created on demand
Mumbai	Cloud A	Object storage

2.1 *Best practices*

- **Place compute near the majority of users,** and serve everyone else from a CDN rather than adding regions early.
- **Keep at least one copy of data on a second cloud.** A single account or provider is a single point of failure for backups too.
- **Record data residency per platform.** Know which jurisdiction each database and backup sits in before a customer asks; add regional copies (United States, European Union, United Kingdom) on demand.
- **Avoid provider-specific services in the core path** unless they earn their lock-in; a platform that can move is cheaper to run and safer to own.

3 Isolation by platform

Unrelated platforms on one server share every failure: a runaway report on one fills the CPU for all; a security issue in one exposes the others; none can be patched or moved alone. We run each platform on its own servers: commerce, workflow, web content, the AI application and the vector service are separate machines, and staging lives on its own server with a copy of each platform.

3.1 *Best practices*

- **Isolate by blast radius,** not by team: whatever must not fail together should not share a machine.
- **Keep tenants of one platform together** (multi-tenancy on shared servers is efficient), but keep platforms apart.
- **Never run staging on production servers.** A test import or load test should not be able to touch live data.
- **Name and inventory every server** with its purpose, region, owner and backup schedule.

4 Content delivery and object storage

Static files are most of a page's weight. Served from the application server they cost CPU, bandwidth and latency; stored on its disk they tie the platform to one machine. Files live in object storage and are served by a CDN whose edges in our platforms' markets include Mumbai, Delhi, Chennai, Singapore, London, New York and Sydney [1].

4.1 *Best practices*

- **Keep application servers stateless for files.** Uploads go to object storage, so a server can be replaced without copying anything.
- **Version file names** (or add a version parameter) so long cache lifetimes are safe: a change produces a new name.
- **Set long cache headers on versioned assets** (a year for scripts, styles and fonts; a month for images) and short ones on anything that changes in place.

- **Serve the HTML from the origin** unless it is safe to cache; pages with carts or prices are not.
- **Restrict who can read storage buckets;** serve through the CDN rather than making buckets public.

5 Image and media pipeline

On mobile networks, images decide how fast a page feels and how it scores on search engines' page-experience measures [2]. Every uploaded image is converted to WebP in three sizes (small, medium and full) at upload; the browser chooses the size it needs; images below the fold load as the visitor scrolls; and each image has its dimensions reserved so the layout does not shift.

5.1 *Best practices*

- **Process at upload, not at request.** Resizing on the fly costs CPU at the worst moment.
- **Use responsive image sets** so phones never download desktop images.
- **Always set width and height** (or an aspect ratio) to avoid layout shift.
- **Lazy-load below the fold, never above it;** the main image should load immediately.
- **Keep originals** in storage so new sizes or formats can be generated later.

6 Layered caching

Caching is the main defence against sale-day spikes and the main source of subtle bugs. We cache in five layers, each for what is safe to keep there.

Layer	Holds	Invalidated by
CDN edge	Images, files, themes	New file name or version
Page cache	Rendered product, category and home sections	Catalogue or content change
Code cache	Compiled application code in memory	Release
Object cache	Frequently read objects, in memory	Write to the underlying record; expiry
Database buffer	Hot rows and indexes in RAM	Managed by the database

Never cached. Carts, prices and stock at checkout and payment, anything tied to a signed-in person, and any response that depends on permissions. A cached price is a pricing error; a cached personal page is a data leak.

6.1 *Best practices*

- **Cache sections, not whole pages,** where pages mix shared and personal content.
- **Invalidate on write** for anything users edit, and use expiry only as a safety net.
- **Key caches by tenant** on multi-tenant platforms; a shared key is a cross-tenant leak.
- **Measure hit rates;** a cache with a low hit rate is only extra memory and extra bugs.

7 Web and application servers

We use an event-driven web server with HTTP/2 and Brotli and gzip compression, a separate FastCGI process manager for application workers, and compiled code cached in memory. AI services sit behind a lightweight reverse proxy. Worker pools are dynamic, up to 50 workers per pool on a commerce server, a figure derived from memory per worker and the memory left after the database.

```
-- sizing the application worker pool
workers = (RAM − database buffer − OS and cache headroom) / memory per worker
-- example: (3 GB − 1 GB − 0.6 GB) / ~28 MB  ≈ 50 workers
```

Listing 1. Example exchange or schema.

7.1 *Best practices*

- **Size worker pools from measured memory per worker,** never from defaults; too many workers swap, too few queue.
- **Use an event-driven server** so slow clients do not each hold a worker.
- **Turn on HTTP/2 and compression** for text responses; serve pre-compressed static files where possible.
- **Keep long work out of the request.** Imports, reports and messages go to background jobs.
- **Set timeouts at every hop** so a stuck request fails fast instead of holding resources.

8 Databases

Databases listen only inside their server. The buffer pool is sized to the machine (1 GB on a 3 GB commerce server; 4 GB on a 15 GB application server). Every query slower than one second is logged, and binary logging is enabled where point-in-time recovery is needed [3].

8.1 *Best practices*

- **Size the buffer pool to hold the working set,** leaving room for workers and the operating system.
- **Log slow queries and read the log weekly;** add indexes or rewrite before users feel it.
- **Index for the tenant key first** on multi-tenant schemas; every query filters by it.
- **Keep binary logs** on platforms where a mistake must be rewound to a minute, not to last night.
- **Never expose the database port;** connect from the application host or through a tunnel.
- **Keep heavy reports off the primary** as volume grows, on a replica or a reporting copy.

9 Vector search and model servers

AI features add three kinds of infrastructure: a vector index, the services that call models, and GPU servers. Our vector search runs in containers on its own server. Scheduled jobs add new records and remove deleted ones so the index tracks the source database; a daily job validates results; and a separate job cleans old logs. GPU model servers are created from an operations host when a workload needs them and stopped when it finishes.

9.1 *Best practices*

- **Keep the index in step with the source** with incremental sync plus a separate deletion pass; an index that only grows returns answers from deleted records.
- **Validate retrieval daily** against known questions, so drift is caught before users notice.
- **Run vector search apart from the main platform** so heavy indexing cannot slow transactions.
- **Treat GPUs as on-demand.** Idle GPU servers are among the most expensive resources a platform can leave running.
- **Watch disk on vector hosts;** indexes and snapshots grow faster than expected.

10 Scheduled operations: the night shift

Time	Job	Why
01:00	Clean up records; compress yesterday's logs; remove old requests	Keeps tables and disks small
02:30	Detect and remove bot traffic and sessions	Keeps analytics and sessions honest
03:30	Rebuild sitemaps for every store	Search engines see current catalogues
03:45	Sync catalogues with shopping feeds; pull approval status	Listings stay valid
Every 2 h	Abandoned-cart reminders	Recovers sales, day and night
Weekly	Longer maintenance	Runs on the quietest night

10.1 *Best practices*

- **Schedule heavy jobs for the lowest traffic,** and stagger them so they do not compete.
- **Make jobs idempotent** so a rerun after a failure does no harm.
- **Log every run with its duration;** a job that doubles in time is an early warning.
- **Alert on jobs that do not run,** not only on jobs that fail.

11 Monitoring, metrics and logs

Every server reports CPU, memory and disk metrics. Application performance monitoring traces slow requests and errors to the code on the busiest platforms. Uptime is checked on every store and admin. Alerts fire on errors, failed logins, traffic spikes and downtime. Yesterday's logs are compressed, summarised and read every morning, and slow queries are reviewed [4].

11.1 *Best practices*

- **Monitor from outside as well as inside;** a server can be healthy while users cannot reach it.
- **Alert on symptoms users feel** (errors, slow pages, failed checkouts) before causes (CPU).
- **Summarise logs daily;** nobody reads raw logs, everyone reads a one-page summary.
- **Rotate and compress logs** so disks do not fill; keep enough history for incidents and audits.
- **Trace requests end to end** on the busiest platforms, so a slow page can be traced to a query or call.

12 Staging and releases

A staging server keeps a copy of each platform. Every change is tried there before it is released, and releases go to every tenant together from one codebase, so no tenant is left on an old version and no customer-specific fork exists.

12.1 *Best practices*

- **Keep staging close to production** in software versions and configuration; differences are where releases break.
- **Use anonymised or sample data** on staging; production data on staging is a privacy risk.
- **One codebase for all tenants;** configure differences, do not fork them.
- **Release at known times** with someone watching the metrics afterwards.

13 Messaging across channels

Messages are part of the platform's infrastructure: an OTP that does not arrive is a failed login, and an order email in spam is a support call. Each platform routes messages by type and channel.

Channel	Used for	Practice
Email via delivery services	Orders, invoices, resets, briefs	Dedicated transactional services; signed sender domains (SPF, DKIM); bounce handling
Email via local relay	Server mail, notices	Queues and retries on each server
Mobile	OTPs	Expiring codes; limits per number
WhatsApp	Support, updates, reminders	Support groups, including outside working hours
Slack	Team alerts and approvals	Per-channel routing
In-app	Every notification	A record the user can always find

13.1 *Best practices*

- **Never send from the request.** Queue messages so a slow provider never slows a page.
- **Authenticate sender domains** with SPF, DKIM and DMARC, and route bounces back to the platform [5].
- **Separate transactional and marketing streams** so a campaign cannot hurt OTP delivery.
- **Log every send** with its status, so "I never got it" can be answered.

14 Backups and recovery order

Backups and their schedules are described in our security report. Infrastructure adds the recovery order: after a failure, each platform is restored in the order that lets customers transact soonest: sign-in, then catalogue, then checkout and payments, then orders, then everything else. The order is tested in a monthly drill [6].

15 A checklist for your own platform

1. If your main region failed now, where would your platform run, and how long would it take?
2. Do any two unrelated platforms share a server?
3. Are images and files served from your application server?
4. Do phones download desktop-sized images?
5. What exactly is cached, and what must never be?
6. How many application workers can each server run before it swaps?
7. Which queries take more than a second, and who reads that list?
8. Does your vector index remove records deleted from the source?
9. Are GPU servers running right now with nothing to do?
10. What runs every night, and would you know if it did not?
11. Would you hear about downtime before a customer does?
12. Does staging match production?
13. Do your OTPs and order emails reach the inbox?
14. In what order would you restore your platform?

16 References

1. M. Thomson and C. Benfield. *HTTP/2*, RFC 9113, IETF, 2022.
2. Google. *Core Web Vitals*, web.dev documentation.
3. Oracle. *MySQL 8.4 Reference Manual: InnoDB Buffer Pool; The Slow Query Log; The Binary Log*.
4. B. Beyer et al. *Site Reliability Engineering: Monitoring Distributed Systems*. O'Reilly, 2016.
5. M. Kucherawy and E. Zwicky. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*, RFC 7489, IETF, 2015.
6. M. Swanson et al. *Contingency Planning Guide for Federal Information Systems*. NIST SP 800-34 Rev. 1.
7. Cloudnix. *Defence in Depth for Business Platforms*, Technical Report CNX-TR-2026-09, 2026.

About this report. Prepared by the Cloudnix engineering team from systems we have designed, built and operated. Client and product names are withheld; illustrative values are marked as examples. Cloudnix Software Labs, Bengaluru, India · cloudnix.com. This report may be shared freely in its entirety.

Document control

Report	CNX-TR-2026-10	Version	1.0
Issued	November 2026	Status	Published
File	cloudnix-infrastructure-practices-v1.0.pdf		