WP-2026-001 — SynthIQ: The Smart DICOM Load Balancer (Whitepaper)

Document number: DOC-2026-431 **Whitepaper:** WP-2026-001 **Version:** 5 **Category:** Marketing Material **Product family:** fleet (SynthIQ) **Owner:** Synthology Healthcare Solutions Group, LLC (SHSG) **Audience:** Radiology IT decision-makers, CIOs, PACS administrators, and imaging informaticists evaluating a DICOM-aware load balancer in front of a PACS or Router pool. **Regulatory posture:** Non-device software under **FD&C Act §520(o)(1)(D)**. No clinical or diagnostic function; nothing in this whitepaper describes a medical device. **Companion:** Renders on the marketing site at `/whitepaper/synthiq` ("SynthIQ — The Smart DICOM Load Balancer"). Published 2026-05-13.

Executive summary

PACS deployments have grown past the point where a single server can handle peak load. Most sites now run three to five backend Routers or PACS nodes in a pool, fronted by a generic load balancer (F5, NetScaler, HAProxy). That generic LB has no idea what a DICOM Study Instance UID is. It hashes the TCP 5-tuple and scatters instances of a single study across multiple backends. No single node holds the whole exam, so re-sending or forwarding that study duplicates fragments at the destination, per-node counts disagree, and there is no single device to recover the study from.

SynthIQ is a drop-in replacement for that generic LB. It speaks DICOM, reads the Study Instance UID from each C-STORE, and keeps a study together on one backend — across the inbound association, across separate associations and late re-sends hours or days later, and across SynthIQ restarts. It routes on Study Instance UID, not TCP 5-tuple, and is a drop-in replacement for F5, NetScaler, or HAProxy in front of any PACS or DICOM Router pool.

This paper covers the problem, the design, the operational characteristics, the regulatory posture, and an honest comparison against the alternatives.

§1 — The problem

1.1 What modern PACS deployments look like

A typical mid-to-large radiology operation runs:

- **Inbound modalities** — CT, MR, US, CR, mammography. Tens to hundreds of devices, each pushing DICOM C-STOREs to a single configured destination AE title + IP.
- **A backend Router or PACS pool** — 3 to 5 nodes running identical Storage SCPs. Sized so the pool absorbs peak load with one node out for maintenance.
- **A generic load balancer in front of the pool** — F5 BIG-IP, NetScaler ADC, HAProxy, Citrix, AWS NLB. Modalities are configured to point at the LB's VIP.

This topology is right. Pooling is the only way to scale, and putting an LB in front gives a single VIP for modality configuration plus the ability to roll nodes for maintenance. The problem isn't pool-and-LB. The problem is *which LB*.

1.2 What generic load balancers actually do

L4 (transport-layer) load balancers distribute TCP connections based on a hash of the connection 5-tuple: source IP, source port, destination IP, destination port, protocol. Some allow weighting and persistence by source IP. None parse the DIMSE payload above TCP. This is fine for stateless HTTP; it is structurally wrong for DICOM.

A modality sending a 1,500-instance CT study to the pool's VIP typically opens one or more associations. The LB hashes each new TCP connection, so — depending on the modality's connection-pooling behavior and the LB's hashing function — the 1,500 instances get distributed across two, three, or all five backends. The Study Instance UID (the clinical identifier that says these instances are one exam) is invisible to the LB. The LB scatters the study because it has no way to know it shouldn't.

1.3 The downstream consequences

A study split across the pool shows up in three operational places:

- **On re-send and re-route.** No single backend holds the whole exam, so when that study has to be re-sent — or forwarded to a second destination (an AI vendor, a VNA, a downstream archive) — every node holding a fragment re-transmits its own portion. The receiving system gets overlapping, duplicated instances, and the operator has no single place to re-drive the study from.
- **In counts and reconciliation.** Each node reports a different partial instance count for the same Study Instance UID. Worklist aggregators, QA dashboards, and RIS-PACS reconciliation either show the study several times with conflicting totals, or show one entry with whichever node reported first — silently hiding the rest. Both impose manual review.
- **In support and failure recovery.** When a transfer partially fails, there is no device that is the study's source of truth. Answering "did the whole study arrive, and where?" means querying every backend and reconciling fragments by hand.

1.4 Why this hasn't been fixed before

DICOM-aware load balancing has been a known need for fifteen years. Commercial DICOM routers handle session-level routing and provide buffering, but none ship study-affinity routing as a default tier with a persistent cache that survives proxy restarts. Vendor-specific PACS products do something similar internal to their own clusters, but you must buy the entire PACS to get it, and they don't help with a heterogeneous pool. The gap SynthIQ fills is a stand-alone, vendor-neutral, drop-in product that does study-affinity routing as its first-class behavior.

§2 — The solution

2.1 The core idea

When a C-STORE arrives at SynthIQ, the proxy:

1. Parses the DICOM dataset enough to read the Study Instance UID.
2. Looks up the SUID in a persistent affinity cache.
3. If found, routes the C-STORE to the backend that previously held this study.
4. If not found, picks the least-busy healthy backend, records the binding, and routes there.
5. Forwards the C-STORE to the chosen backend, returning the backend's status verbatim to the modality.

Every subsequent instance of that study — whether it arrives in the same modality association, on a later association, as a delayed re-send hours or days afterward, or after a SynthIQ service restart — goes back to the same backend. The clinical study stays together. Affinity keys on the Study Instance UID alone: a separate exam carries a different UID, so SynthIQ groups the instances of one study, not a patient's separate studies — and it persists no MRN or demographics to do otherwise.

2.2 Why the study is the right unit

A DICOM study is, by definition, the unit of clinical work. The DICOM standard requires every instance of a study to share a Study Instance UID, generated once at study creation and never rewritten — so the SUID is the one identifier that reliably says "these objects are one exam." Routing on it means a study is never split across the pool: it is received, stored, re-sent, and forwarded as a single coherent unit on one backend, with one device as its source of truth.

The cache window matters because a study's instances don't always arrive in one burst. Additional or corrected series, a delayed re-send after a network interruption, and prior-reconciliation transfers can land hours or days after the first instance. As long as they carry the same Study Instance UID, SynthIQ routes them to that study's original backend, so the study never fragments. The default retention is a **96-hour window** (`StudyCacheWindowHours`, configurable per deployment); once it lapses on a completed study the binding expires to bound cache growth. This is study-level, not patient-level: SynthIQ does not link a patient's separate exams — each is a distinct UID — and stores no MRN or demographics that would let it.

2.3 What happens when a backend is unhealthy or busy

SynthIQ polls each backend's health endpoint every 5 seconds (configurable). Each backend reports its queue depth — how many DICOM operations are currently in flight or held. SynthIQ uses this to make two distinct decisions:

- **For new studies (no affinity binding yet)**, SynthIQ picks the backend with the lowest queue depth among healthy nodes. This avoids piling new work onto an already-saturated node. A generic LB has no view into backend load and will continue sending traffic to a backend that's quietly drowning.
- **For studies with an existing affinity binding**, SynthIQ honors the binding even if that backend is currently slow or unhealthy. The study's already-received instances live there; moving a study mid-stream fragments it across two nodes — exactly the problem SynthIQ exists to prevent. If the bound backend is truly down, the modality sees a transient error and retries (standard DICOM behavior), which surfaces the outage to the operator.

§3 — Architecture

Modalities push C-STOREs into the SCP listener. The routing engine consults the affinity store, picks the right backend, and forwards via a pooled SCU. Heartbeats run independently every 5 seconds against each

backend's `/api/health` endpoint to keep queue-depth state fresh. Operators inspect the live state via the Admin SPA.

3.1 Inbound SCP (the DICOM front door)

SynthIQ exposes one DICOM Storage SCP listener per configured **pool**. A pool binds (TCP port, AE title) to a set of backend nodes. Each pool runs independently — different pools can implement different routing policies, accept different AE titles, or front different backend node sets. A typical deployment runs a single pool with the AE title `SYNTHIQ`.

The SCP accepts any Storage SOP class plus Verification (C-ECHO). It is intentionally class-agnostic — the routing decision is about the study's identity, not its modality. CT, MR, US, mammography, PT/CT, DICOM SR, structured reports — all route the same way.

3.2 Routing decision (the three-tier waterfall)

When a C-STORE arrives, SynthIQ runs three tiers in order. The first tier that matches wins:

- **Tier 1 — Persistent affinity.** If the SUID is in the affinity cache and bound to a backend that's still in the pool and still enabled, use that backend. This honors clinical-identity continuity regardless of current load.
- **Tier 2 — Health-aware least-busy.** Otherwise, pick the backend with the smallest queue depth among healthy nodes. Ties broken deterministically by backend ID so repeat decisions converge.
- **Tier 3 — Last-resort fallback.** If no backend is currently healthy (cold start before heartbeat warms, or transient pool-wide outage), pick the first enabled node. Logs the fallback so the operator sees it.

Once the decision is made, SynthIQ records the binding in the persistent store and uses it for every subsequent same-SUID instance.

3.3 Outbound SCU (the proxy forwarding)

SynthIQ holds one pooled SCU association per (inbound association, backend) pair. The DICOM library's association linger timeout keeps the underlying connection open across back-to-back C-STOREs. A 1,500-instance study reuses one outbound association instead of negotiating 1,500 times — a measured throughput improvement of **roughly 24× over the unpooled baseline**. When the inbound association releases (or aborts), SynthIQ drains the pooled SCU connections cleanly so backends aren't left with dangling associations.

3.4 Affinity store

The persistent cache is backed by one of three databases, chosen by the operator at deployment:

- **SQLite** (default) — in-process, no external DB. Comfortable to ~1,000 instances/second on commodity SSD. Right for the first decade of customers and for single-host SynthIQ deployments.
- **PostgreSQL** — the same instance most VNA customers already operate. Right when you want HA across multiple SynthIQ instances or 10×+ scale.
- **SQL Server** — for shops standardized on the Microsoft data tier.

All three implement the same schema (`study_uid`, `pool_id`, `backend_id`, `first_seen_utc`, `last_seen_utc`, `instance_count`, `committed_utc`). Migration scripts move the cache between providers without changing application code.

3.5 Heartbeat service

A background service polls each enabled backend's `/api/health` endpoint every `HeartbeatIntervalSeconds`. SynthIQ computes the backend's effective queue depth as `received + hold + processing` (deliberately excluding `failed` — backends with a lot of failed-but-stuck items are unhealthy, not "busy"). A backend that misses heartbeats for `HeartbeatTimeoutSeconds` is marked unhealthy and dropped from Tier-2 candidate selection.

3.6 Vendor-neutral health tiering (three doors, no dead ends)

The Tier-2 least-busy decision assumes a backend can tell SynthIQ how busy it is. Every Synthology product can — it serves the full `/api/health` contract. But a vendor-neutral pool may also hold third-party PACS, research VNAs, or cloud endpoints that report load in their own format, or not at all. SynthIQ grades each backend by **load-signal fidelity** and routes to it accordingly — never assuming a load it wasn't given. Four health modes, from richest signal to thinnest:

- **Native** (Synthology products) — the full `/api/health` contract: queue depths, a normalized load score, active associations, disk headroom, throughput. Feeds least-busy today and weight-aware predictive routing as it lands.
- **Certified — "SynthIQ-Ready"** (any vendor, self-service) — the backend implements the published *SynthIQ Health Contract*: one lightweight endpoint returning, at minimum, `status` (accepting work?), `queue_depth` or a normalized load, and `capacity` or `max_associations` (the denominator that makes load comparable across heterogeneous nodes). A real load number means real least-busy — native-grade on the essentials.
- **Adapter** (any vendor, no vendor effort) — the backend exposes its own existing health/metrics in its own shape, and Synthology writes a per-vendor normalizer that maps them onto the common load scale. Same least-busy quality; the integration work is ours, built on first real demand.
- **Liveness** (any DICOM node) — no parseable load signal at all. SynthIQ runs an up/down probe, preferring the standards-native DICOM **C-ECHO** Verification SOP class (every conformant node answers it) and falling back to TCP or HTTP. A live node is distributed to by its configured **weight** (weighted round-robin); a node that fails the probe drops out.

For a third-party backend, that is **three doors** — implement the contract, let us adapt, or run heartbeat-only — and **none of them is a dead end**. Any DICOM-conformant node is a first-class pool member; only how much intelligence its load signal unlocks varies.

The safety principle. A backend enters the Tier-2 least-busy ranking *only* when its load is genuinely known (Native, Certified, or Adapter). A Liveness backend's load is *unknown* — represented as null, never as zero — so it is excluded from least-busy and balanced by weight instead. This formalizes and fixes the classic generic-balancer failure mode, where a silent backend looks like the emptiest node and gets flooded: SynthIQ never balances a backend on a number it didn't report, and never floods one it can't measure.

Status. The full health-mode model ships today — Native, Certified (the published SynthIQ Health Contract + self-service SynthIQ-Ready certification), and Liveness (the standards-native C-ECHO probe) — together with the null-load safety rule that excludes an unmeasured backend from least-busy. The one path built on demand is the per-vendor **Adapter**, where Synthology normalizes a specific vendor's existing metrics onto the common load scale when an integration calls for it.

3.7 Admin SPA + JSON APIs

A single-file dashboard at `http://<synthiq>/` polls four endpoints every 2 seconds:

- `GET /api/status` — aggregate counters, listener health, cache size, routing decisions vs cache hits, per-backend health snapshot.
- `GET /api/pools` — live pool config + per-backend queue depth + commitment state.
- `GET /api/affinity` — paged SUID→backend bindings, searchable, with commitment status.
- `GET /api/activity` — recent first-instance routing decisions with the chosen tier (persistent-affinity-hit, least-busy, fallback).

Plus two endpoints for upstream visibility:

- `GET /api/affinity/{suid}/commitment` — has the backend reported durable commitment?
- `POST /api/affinity/{suid}/commitment` — backend reports it has durably committed the study.

The dashboard color-codes routing-decision pills so an operator can see at a glance what's happening: **affinity hit** (persistent binding matched), **least busy** (new study, picked lowest-queue backend), **association cached** (reused open SCU connection), and **fallback** (heartbeat not warm yet or pool-wide outage).

§4 — Operational impact

SynthIQ is a routing layer, not a viewer or an archive — it never caches or stores the pixel data (objects stream through a transient spool and are forwarded; only the routing record persists). Its value is therefore operational: what changes when every instance of a study reliably lands on one backend instead of scattering across the pool.

4.1 Clean, non-duplicating re-sends and re-routes

This is the differentiator that matters most in day-to-day support. When a study is scattered across several backends and it has to be re-sent — or forwarded to an additional destination (an AI vendor, a VNA, a downstream archive) — every backend holding a fragment re-transmits its own portion, and someone has to reconcile the overlapping, duplicated instances. With SynthIQ, the whole study lives on one backend: a re-send or re-route is a single, clean transmission from one device — no duplicate fragments, and one obvious place for support to re-drive the study from.

4.2 Consistent counts and reconciliation

Any system that counts or deduplicates by Study Instance UID — worklist aggregators, QA dashboards, RIS-PACS reconciliation — depends on each study being whole in one place. A scattered study makes these systems either show the study several times with conflicting instance totals, or show one entry with whichever backend reported first (silently hiding the rest). The first imposes explicit manual review; the second is worse because it is silent and can mask a partial transfer. Co-locating every instance of a study on one backend makes the count unambiguous.

4.3 Balanced backends

New studies route to the backend with the lowest real reported load (health-aware least-busy), so work spreads evenly across the pool instead of piling onto whichever node a connection hash happened to pick. A backend whose load SynthIQ cannot measure is never assumed empty — it is balanced by its configured weight and watched by an up/down probe, so a quiet or heartbeat-only node is never flooded (see §3.6). The result is even

utilization and headroom that a connection-level balancer cannot deliver, because it has no view into a DICOM backend's ingest queue.

4.4 Dedicated lanes for priority traffic

Each SynthIQ pool is an independent listener + AE title + VIP + backend set, so an operator can stand up a separate pool for urgent traffic pointed at a specialized or faster backend. The modality operator chooses the lane — sending an urgent study to that pool's AE title puts it on a path that runs independently of routine volume, with its own affinity and least-busy balancing, so a flood of routine work cannot delay it. The clinical decision of what is urgent stays with the operator; SynthIQ provides the dedicated pool and faithfully delivers to it.

§5 — Operational characteristics

5.1 Deployment shape

SynthIQ ships as a single self-contained executable. Three deployment options:

- **Windows Service** (most common for hospital IT) — Installer MSI, service runs as a configurable user.
- **Linux systemd** (for cloud-hosted or Linux-shop deployments) — same binary self-contained, packaged as a tarball.
- **Docker container** (roadmap) — same binary in a minimal base image, intended for Kubernetes operators who want rolling-deploy semantics.

Configuration is in a single JSON config file. Operators edit it directly or through a future admin UI; the current admin SPA is read-only and uses the config as authoritative.

5.2 Monitoring surface

- All JSON endpoints return standard 200/4xx/5xx with JSON bodies. Prometheus exporter is on the roadmap; for now, scrape `/api/status` JSON.
- Structured console + rolling file logging. Each routing decision emits one structured Info line with pool, source AE, source IP, SUID, SOPUID, modality, patient ID, chosen backend, and decision reason.
- Per-association summary log line on release covers the whole association in one row: `rx=N fwd=N failed=N studies=N cache-hits=N backends=N`.

5.3 Scale envelope

Measured on commodity server hardware:

Metric	Measured
End-to-end throughput (modality → SynthIQ → backend → response)	~514 study-instances/sec mean over five runs (range 419-550; 20,000 / 20,000 delivered on every run; 2 vCPU / 8 GB SynthIQ across a two-node Router pool, shipped defaults — DOC-2026-490 v4)
Routing-decision latency (first instance of a study)	~5–15 ms (cache miss)
Routing-decision latency (subsequent instances)	~0.1 ms (in-memory cache hit)
Affinity-cache write rate	~1,000+ writes/sec sustained
Heartbeat cost	One HTTP GET per backend per interval

A single SynthIQ instance comfortably handles a 5-backend pool receiving from 50–100 modalities at typical clinical rates. For larger envelopes, the path is multi-instance SynthIQ behind a generic L4 LB (yes, the irony) sharing a PostgreSQL or SQL Server affinity backend.

5.4 Failure modes

Failure	Behavior
Backend goes down	Heartbeat marks unhealthy. New studies route to other backends. Affinity-bound studies still try the bound backend; the modality sees a transient error and retries — intentional, because moving a study mid-stream fragments it.
Backend goes slow	Queue depth on `/api/health` rises. Tier-2 decisions route new studies to other (lower-queue) backends. Tier-1 affinity-bound studies stay where they are.
SynthIQ crashes / restarts	Affinity cache is persistent — restarts come up with the cache intact. In-flight C-STOREs in the brief restart window error to the modality; modality retries are unaffected.
Affinity DB unreachable (PostgreSQL/SQL Server)	Cache lookups fail; SynthIQ falls back to Tier-2 (least-busy) for all decisions. Logs the DB-unreachable state. New bindings won't persist until the DB is back.
Disk fills (SQLite case)	Affinity writes fail; subsequent same-SUID studies don't get persistent affinity, fall through to per-association cache. Logs the disk-full state.
Cold start	Tier-3 fallback runs (first-enabled-node), routes proceed normally. Switches to Tier-2 once heartbeat completes its first tick.

SynthIQ does not durably store DICOM payloads. C-STORE bytes flow through and are dropped from memory once forwarded. This is a deliberate design choice (see §6 and §8).

§6 — Regulatory posture

6.1 Classification

SynthIQ is **non-device software** under **FD&C Act §520(o)(1)(D)**, the 21st Century Cures Act §3060 carve-out that excludes from the FDA device definition software functions intended for transferring, storing, converting, or displaying device data, provided the software does not interpret or analyze that data. The classification rests on three properties:

1. **No storage of clinical data.** DICOM payloads pass through; SynthIQ does not retain images, reports, or any clinical-content payload. The affinity cache holds routing metadata only — Study Instance UID, pool ID, backend ID, timestamps, instance count, commitment timestamp. No PHI body.
2. **No transformation of payload.** Bytes received from the modality equal bytes forwarded to the backend. SynthIQ does not anonymize, re-encode, or modify the DICOM dataset.
3. **Routing decisions on identity, not clinical content.** The SUID is a unique identifier, not a clinical measurement. SynthIQ's routing logic does not interpret, classify, or make clinical determinations about the imaging data.

6.2 HIPAA posture

Per HIPAA's transport-layer accommodations, SynthIQ operates as a covered transport conduit. The standard Synthology BAA applies. The metadata cache holds the Study Instance UID — which is considered an identifier only in context (a SUID alone, with no patient name, no medical record number, no date of birth, no demographic data, does not identify a patient under HIPAA's narrow construction). Customer threat models that consider SUID metadata sensitive can promote the cache to encrypted-at-rest using Synthology's standard fleet-wide key escrow architecture.

6.3 Cybersecurity posture

SynthIQ inherits the Synthology fleet's at-rest crypto architecture for any future encryption promotion. The persistent affinity database is at-rest unencrypted by default (the trade-off is operator inspectability via SQL/SQLite tools); customers who require encryption deploy SynthIQ under the Synthology master+subkey tree, identical to how every other Synthology product handles its at-rest state. Network posture: DICOM TLS is supported; production deployments should enable TLS and configure modalities accordingly.

6.4 Regulatory posture detail

SynthIQ is **not a device** within the meaning of FD&C Act §201(h) by operation of §520(o)(1)(D). The firm maintains ISO 13485–aligned QMS practices. Reclassification path: if a future SynthIQ release adds a clinical-interpretation or recommendation function, the carve-out no longer applies and the product becomes a device subject to the usual classification analysis. Today's SynthIQ has no such function (see §6.1).

§7 — Comparison

An honest comparison against alternatives. None of these products are bad — they're built for different problems. The table illustrates where SynthIQ fits.

Capability	SynthIQ	F5 BIG-IP	NetScaler	HAProxy	Mirth Connect	Forcare	Vendor PACS LB
L4 connection load balancing	✓	✓	✓	✓	✓ basic	✓	✓
Health-checked backends	✓	✓	✓	✓	✓	✓	✓
SUID-keyed study affinity	✓	✗	✗	✗	partial	✓ (proprietary)	✓ (intra-vendor)
Persistent affinity across restarts	✓	✗	✗	✗	✗	partial	✓
Honors affinity to unhealthy backend	✓	✗	✗	✗	✗	✗	✓
Backend queue-depth-aware routing	✓	partial	partial	partial	✗	✗	✓
Safe with a silent / no-load backend (won't flood)	✓	✗	✗	✗	✗	partial	partial
Mixed-vendor backends graded by load fidelity	✓	✗	✗	✗	✗	✗	✗
Drop-in replacement for L4 LB	✓	n/a	n/a	n/a	✗	✗	n/a
Vendor-neutral (heterogeneous pool)	✓	✓	✓	✓	✓	✓	✗
Open / inspectable affinity store	✓	✗	✗	✗	partial	✗	✗
Approximate cost (mid-size)	$$	$$$$	$$$$	$ (OSS)	$$	$$$	bundled

- **F5 and NetScaler** are excellent products for their domain. They don't read DICOM payload above TCP. Customers running them in front of PACS pools live with the scatter problem or accept the engineering cost of writing custom iRules / NetScaler responder policies that parse DICOM in flight — possible but operationally expensive.
- **HAProxy** has the same L4-only constraint; in its case, the open-source price point typically wins the deployment regardless of the scatter cost.
- **Mirth Connect** is a healthcare integration engine that includes DICOM channels. It can do session-scoped routing, but it doesn't ship a persistent SUID-keyed affinity store as a first-class behavior. Customers who use Mirth for DICOM routing typically build the affinity layer themselves.
- **Forcare** is the closest commercial peer — a complete DICOM router with proprietary affinity logic, built for the same customer profile. Forcare is expensive and ships as an appliance with vendor-specific operational expectations.
- **Vendor PACS internal LB** — Sectra, Visage, GE Centricity, etc. Generally good at this within their own product, useless for cross-vendor pools.

SynthIQ's positioning is "the missing tier between generic L4 LB and a full DICOM router." If you already run Forcare and it works, you don't need SynthIQ. If you run F5/NetScaler/HAProxy in front of a DICOM pool and you've felt the scatter pain, you do.

The displacement, precisely. A generic L4 balancer routes a *TCP connection* and checks a backend with a *TCP-open or HTTP-200* probe. Neither tells it anything about a DICOM study or a backend's ingest queue — so it splits one exam across nodes and, worse, treats a silent or heartbeat-only backend as healthy and floods it. SynthIQ routes the *study*, balances on *real reported ingest load*, and never balances a backend on a number it didn't report. It displaces the generic balancer on the imaging path — and, because it fronts a mixed-vendor pool graded by load fidelity, it also removes the single-vendor balancer's lock-in: one software control point for a whole multi-vendor imaging estate, no appliance license.

§8 — Future direction

8.1 Immediate roadmap

- **Prometheus exporter** for `/api/status` so SynthIQ feeds standard observability stacks without scraping JSON.
- **Admin UI mutating endpoints** (pool edit, heartbeat tuning) behind authentication. Current admin SPA is read-only.
- **DICOM TLS on by default** for production deployments.

8.2 Conditional roadmap (ship when customer signal arrives)

- **Per-vendor health adapters.** Normalizers that map a third-party backend's own existing health/metrics onto the common load scale, so it earns first-class least-busy without implementing the SynthIQ Health Contract itself. The rest of the vendor-neutral health tiering in §3.6 — Native, Certified, and Liveness — ships today; the adapter path is built when a specific vendor integration calls for it.
- **In-memory transient queue** to absorb brief backend hiccups (seconds to minutes) without changing the regulatory posture. Bounded RAM-only queue per (pool, backend) with time-out and overflow semantics;

the modality retains the retry contract.

- **DICOM-native Storage Commitment SCP** for modalities that issue N-ACTION commitment requests. The current commitment surface is HTTP-based (backend → SynthIQ); the DICOM-native layer adds modality → SynthIQ → backend proxying.

8.3 Architectural directions under review

- **Durable buffering (spool-and-forward) inside SynthIQ.** This would alter the data-handling profile that today keeps SynthIQ within the §520(o)(1)(D) non-device carve-out, and is therefore deliberately out of scope. Customers who need durable buffering deploy XyDromatics Router (the Synthology fleet's durable-buffer DICOM product) in front of or behind SynthIQ.
- **Multi-region HA** with shared affinity backend across SynthIQ instances and active-active replication. Doable today with PostgreSQL or SQL Server as the backend; the operational pattern is documented but not yet a turnkey deployment.

8.4 What SynthIQ won't do

- **Anonymization or pseudonymization inside SynthIQ.** That's XyDromatics Router's role.
- **Clinical report generation, viewing, or interpretation.** SynthIQ is a transport. The regulatory + commercial positioning rests on this.
- **PACS replacement.** SynthIQ goes between modalities and a PACS pool. It is not a PACS, does not store images, does not query/retrieve by clinical content.

Conclusion

SynthIQ is what a load balancer for a PACS pool should be: aware of the DICOM payload it routes, sticky to the unit of clinical reading (the study), and operationally simple enough to deploy as a drop-in replacement for the F5 or NetScaler you have today. It doesn't try to be a PACS, a DICOM router, or a durable buffer. It does one thing: keep studies together on the right backend.

The product ships with an open inspectable affinity store (SQLite, PostgreSQL, or SQL Server), and a regulatory posture that lets a customer deploy it as non-device software under §520(o)(1)(D). POC engagements typically run **30 days** from initial config to bridge-and-decommission of the existing L4 LB.

Glossary

- **Affinity binding:** the persistent association between a Study Instance UID and the backend node that holds its instances.
- **DIMSE:** DICOM Message Service Element — the application-layer primitive that carries C-STORE, C-FIND, C-MOVE, etc.
- **§520(o)(1)(D):** FD&C Act subsection added by the 21st Century Cures Act of 2016 §3060. Excludes from the FDA device definition any software function intended for transferring, storing, converting, or displaying clinical-lab or other device data, provided the software does not interpret or analyze that data. The carve-out under which SynthIQ and every other Synthology product is classified as non-device software.

- **SCP / SCU:** Service Class Provider (server) / Service Class User (client). DICOM terminology.
- **SUID:** Study Instance UID. The DICOM-standard unique identifier for a clinical study.

> **Regulatory classification.** All Synthology and XyDromatics software described here is non-device software under the FD&C Act §520(o)(1)(D) software exclusion — not a medical device, and no clinical diagnostic function. Published by Synthology Healthcare Solutions Group, LLC as a controlled document; the authoritative version is the approved record in the Synthology quality-management system.