High-Availability Reference Architecture

MessageFoundry runs reliably as a single process, and scales to a highly available pair when you need to survive a host failure. This reference describes the supported HA model — **active/passive** (one leader, one or more warm standbys) — how leadership is coordinated, what the operator supplies, and what failover looks like.

The defining property of this design is that **all coordination lives in the shared SQL store**. There is no separate clustering broker, no Windows Server Failover Clustering (WSFC), and no Active Directory dependency. If you can stand up a server-class database and a basic load balancer, you can run MessageFoundry highly available.

Single-node vs. two-node active/passive

	Single node	Two-node active/passive
Processes	One engine	Two (or more) identical engines
Store	SQLite, PostgreSQL, or SQL Server	PostgreSQL **or** SQL Server (shared)
Who runs the message graph	The one node	The **leader** only
Failure of a host	Interface is down until restarted	A standby is promoted and resumes
Operator load balancer	Optional	**Required** (floating VIP / L4 LB)
Coordination	None needed	App-level leader lease in the store

Single-node operation is the default and is the same whether or not clustering exists in the build — clustering is strictly opt-in. Turn it on by enabling the cluster on a server-class store; every node then coordinates so that **exactly one** node — the leader — runs the message graph, and a standby takes over on failure.

Two-node active/passive is the supported HA topology. (A horizontal active/active scale-out model is **not** offered.)

Requirements

A clustered deployment requires, and enforces at configuration load:

- A **server-class shared store** — PostgreSQL or SQL Server. SQLite is a single-file, single-node store and cannot back a cluster; the cluster needs the shared membership table and row leases that a server

database provides. Both PostgreSQL and SQL Server run the same active/passive leadership lease and are at parity.

- **Every node points at the same database** (same server, database, and schema) and runs the **same configuration directory**.
- A **connection-pool size with headroom** — a clustered node drives concurrent background work (membership and lease renewal, the leader reclaim sweep, and the per-stage workers), so give the pool room above its working connections.

The same `serve` command starts on each host. Nodes self-identify; you only need to pin a node identity for a stable identity across restarts.

The leader lease (no WSFC, no AD)

Leadership is an **application-level lease stored as a row in the shared database** — not an OS-level cluster service. This is what removes the WSFC/AD requirement: the database is the single source of truth for who leads.

How it works at a high level:

- **Exactly one node holds the leader lease** and is the leader. The leader renews the lease on a short heartbeat interval; the expiry is computed on the **database's own clock**, so node-to-node clock skew never affects who may lead.
- **A standby acquires leadership only after the lease has expired.** A standby never preempts a healthy leader.
- **A leader that cannot renew self-fences.** If the current leader cannot refresh its lease within its fence timeout — shorter than the lease TTL — it stops acting as leader *before* the lease can expire and a standby can acquire it. This is the split-brain guard: a network-partitioned or stalled old leader provably stops processing before any standby is promoted.
- **On a clean stop, the leader releases its lease immediately**, so a standby takes over promptly rather than waiting out a timeout.

Three timing values govern this, and the configuration enforces the invariant `heartbeat < fence timeout < lease TTL`. The defaults trade a roughly half-minute crash-failover window for ample tolerance of a momentarily slow database or garbage-collection pause; you can scale all three down proportionally for faster failover at the cost of less slack.

What the leader does (and the standby doesn't)

The wired message graph — **all** inbound listeners (MLLP, TCP, File, and the rest) **and** the router, transform, and delivery workers — runs **only on the leader**.

A **warm standby binds no listener ports and runs no workers.** It stays warm by maintaining its membership heartbeat and converging shared state (reference data, configuration version, transform state), and it brings the full graph up the moment it acquires leadership — and tears it down if it loses leadership.

The graph supervisor polls leadership on a short interval, so a demotion or fence promptly stops a node from accepting new inbound work.

Singleton background work — retention purges, the lease-reclaim sweep, and polling of any **shared** external source (a watched directory, a database-poll table, a remote directory) — is **leader-gated** so it never double-executes.

Because only one node ever processes a given lane, **per-lane FIFO ordering is naturally preserved**, and it survives failover: a promoted leader reclaims a crashed predecessor's stranded in-flight work before delivering anything newer on that lane, so a later message can never jump ahead of an interrupted earlier one.

The floating VIP / L4 load balancer (operator-supplied)

Clients reach "the engine" through a **floating VIP or Layer-4 load balancer that you supply**, not a fixed hostname — so failover is transparent to senders apart from a reconnect. MessageFoundry does not ship this component; it is part of your network.

The mechanism is deliberately simple and relies on the leader-gating above:

- **For each inbound port, configure a TCP health check** — a plain TCP connect to that listener port.
- **Only the leader binds the port**, so the health check passes **only on the leader**. The VIP therefore routes inbound traffic to the leader automatically.
- **On failover, the new leader binds the port** and the old leader's socket closes, so the VIP follows the leader with no manual intervention.

For MLLP/TCP senders, configure partners to **reconnect on drop** (standard MLLP client behavior): on failover they see a connection drop and reconnect through the VIP to the new leader.

The engine API (the console / IDE control and read plane) is available on **every** node, so an API-facing VIP can health-check the unauthenticated liveness endpoint. To pin operations to the leader, the cluster exposes read-only status endpoints (this node's role — leader, standby, or single-node — and the cluster-wide membership with the live leader and lease holder); the monitoring console surfaces the current leader from these.

DB-tier HA is delegated to the database

MessageFoundry does **not** replicate the store itself. **High availability of the database tier is delegated to the database**, using its native mechanism:

- **PostgreSQL** — streaming replication with your chosen failover tooling.
- **SQL Server** — Always On availability groups.

Point all engine nodes at the database's HA endpoint (the replication/failover address), and the database's own failover handles store-tier resilience independently of engine-tier leadership.

Failover behavior

Failover is **not instantaneous** — plan for a promotion window rather than zero downtime:

- **Clean stop (planned switchover or graceful shutdown).** The leaving leader releases its lease, so a standby is promoted on its next heartbeat — failover is prompt.
- **Crash or network partition.** The dead leader's lease ages out, and a standby acquires once it expires (up to the lease TTL). A partitioned old leader self-fences within its fence timeout — shorter than the TTL — so it stops processing before the standby is promoted.

Across the window, no message is lost:

- **In-flight rows are protected by per-row leases** — a standby reclaims only rows whose lease has expired.
- **On promotion, the new leader immediately recovers the predecessor's stranded in-flight work**, so delivery resumes at once rather than waiting out per-row lease timeouts.
- **Ordering is preserved** — the stranded head of each lane is reclaimed before newer work on that lane.
- Delivery is **at-least-once with idempotent re-runs**: a message interrupted mid-delivery is re-delivered once its lease frees, so **downstream endpoints should be idempotent**.

Operational guidance

- **Keep node clocks synced (NTP).** Leadership is decided on the database clock, but per-row leases use node wall-clock, so reasonably synchronized clocks keep lease expiry consistent across nodes.
- **Deploy identical configuration to every node.** Each node loads its graph from its own configuration directory; the cluster coordinates the reload *version*, not the files. Keep the directories identical.
- **Apply configuration changes as a coordinated restart, not a rolling one**, so nodes never run divergent graphs across the change window.

Transport security still applies

HA does not change the transport posture. MessageFoundry binds loopback by default and **fails closed** on a non-loopback API or MLLP bind without TLS. When you expose a listener off loopback — which any real data-plane feed requires — enable TLS on that channel (the API supports in-process TLS or a trusted upstream terminator; MLLP supports per-connection TLS and optional mutual TLS). Inter-node cluster coordination rides the shared store connection and carries **no separate node-to-node socket**, so encrypting the store connection encrypts the cluster traffic with it. Raw TCP and X12 listeners have no transport TLS of their own — keep them on loopback or front them with a TLS-terminating proxy or an isolated trusted segment. This is deployment guidance for the environment, not a limitation of the HA model.

Related

- CLUSTERING.md — clustering reference.

- DEPLOYMENT.md — deployment and network-exposure guide.
- CONFIGURATION.md — the full store and cluster settings catalog.
