Deployment & Network Exposure Guide

This is the consolidated reference for **how every MessageFoundry network channel binds, whether it supports TLS, and how it is authenticated and gated.** If you are about to expose the engine beyond `127.0.0.1`, read the checklist and the channel matrix first.

Related references: design rationale for the off-loopback posture and the engine's transport-security and strong-auth model lives in the project's architecture docs; PHI-in-transit handling is in PHI.md; clustering topology is in CLUSTERING.md.

On-premises by default

MessageFoundry runs **on-premises** and binds **loopback (`127.0.0.1`) by default** — the engine API, and every inbound listener via `[inbound].bind_host`. In that posture there is **no off-host network exposure**: nothing PHI-bearing crosses a wire. Everything below is about what changes when you deliberately bind a channel to a routable address.

Fail-closed rule: a non-loopback **API** or **MLLP inbound** bind is *refused at startup* unless TLS is configured (or, for the API, an upstream TLS terminator is trusted). The `serve --allow-insecure-bind` flag is a **loud dev-only escape**, not a supported production setting.

Trust boundary — inside your organization's private network

MessageFoundry's supported deployment is *inside a single healthcare organization's private, trusted network* (on-prem or the org's private cloud), behind that org's perimeter controls — firewall, network segmentation, VPN/NAC. It is *never* placed directly on the public internet. This is the model every clinical interface engine assumes; state it explicitly in your own runbook, because it is the assumption every control here depends on and the first thing a security reviewer will ask for.

"Inside the network" is a statement about the **trust boundary**, *not* about which interface the engine binds. Three planes sit at different exposure levels:

Plane	What it is	Where it binds	Posture
Management	console / IDE → engine API	loopback by default (or a restricted management subnet)	required auth + RBAC + full audit; smallest surface — keep it off general-user VLANs
Data	inbound feeds you *receive* (MLLP, TCP/X12, DB-poll)	the **internal network interface** — feeds come from other systems on your LAN, not `127.0.0.1`	**TLS on the wire** (enable MLLP-over-TLS) + the `[egress]` /ingress allow-lists + your network segmentation. PHI must not cross the LAN in cleartext
Inbound web service	a partner *calls into* MessageFoundry (SOAP/REST source)	its own network socket	a distinct surface — see the caveat below

The **management plane** is what you keep most contained; the **data plane is network-bound in any real install** (an EHR's MLLP feed is not on localhost) — which is exactly why MLLP-over-TLS and the fail-closed bind-guard exist. With TLS enabled on the data plane, PHI never crosses the LAN in cleartext.

Off-loopback security controls — delegate to your environment (and write it down)

Because the trust boundary is your private network, the controls that only become material once the engine leaves loopback are **satisfied by your organization's existing infrastructure** — *provided you document the delegation* and turn on the engine-side floor. This is an accepted way to meet the deployment-conditional OWASP ASVS items.

Control (ASVS)	Delegate to your environment	Engine-side option
Transport encryption	— *enable* the shipped native API/WSS TLS + MLLP-over-TLS	built in
MFA / multi-layer admin	your **directory (AD / Entra)** via SSO federation enforces MFA for enterprise users	**built-in MFA is required for local accounts**; enterprise/AD users get MFA through their identity provider
TLS client-cert / mTLS	your **PKI**	API mTLS is built in (`tls_client_ca_file` , opt-in) — enable mTLS + a console client cert
Certificate revocation	your **proxy / PKI** (OCSP/CRL at the terminator)	document the delegation, or add OCSP/CRL to the TLS contexts
Off-box log shipping	forward the audit + operational logs to your **SIEM/syslog**	supported via your log-forwarding tooling

Write the delegation into your deployment runbook. "We run MessageFoundry inside our network behind \ <perimeter / IdP / PKI / SIEM>" is what turns these from open gaps into *addressed-by-environment* — for your own risk posture and for any ASVS-scoped review.

On multi-factor authentication

MessageFoundry **requires multi-factor authentication for local accounts** — it is built in and on by default. For **enterprise / Active Directory users**, MFA is enforced by **your own identity provider through SSO federation**: your directory (AD / Entra) is the identity source, and MFA applies via Kerberos / Windows SSO (the workstation logon was already MFA'd), Conditional Access on a federated-SSO front, or an MFA-terminating reverse proxy. For an MFA-required deployment of enterprise users, prefer AD / SSO so the second factor is enforced at your identity provider rather than relying solely on the engine's local-account MFA.

> **Note for security reviewers:** a back-channel **LDAP simple-bind validates the password but does not itself prompt a second factor.** For directory accounts, drive sign-in through Kerberos / Windows SSO or a federated-SSO front so MFA is enforced by your IdP.

Caveat — accepting inbound web-service calls

If you intend to use MessageFoundry to **accept** web-service calls (a partner POSTs *into* the engine), treat that inbound listener as a **distinct network surface even inside your LAN**: it must carry its own partner authentication, TLS, and ingress allow-list, and it lives in the connector layer with its own bind/host/port, **separate** from the management API. Harden it for partner-facing traffic before relying on it. The shipped REST and SOAP connectors are *outbound* (the engine is the client); FHIR and X12 connectors are likewise dialed by the engine.

Before you expose off-loopback

1. **API** — set `[api].tls_cert_file` + `[api].tls_key_file` (in-process TLS), *or* set `[api].tls_terminated_upstream = true` + `[api].trusted_proxies` (front it with a TLS terminator). Keep `[auth].enabled = true` (a non-loopback bind with auth disabled is refused).
2. **MLLP inbound** — set `tls = true` + `tls_cert_file` / `tls_key_file` per connection. A non-loopback MLLP bind without `tls` is refused.
3. **Raw TCP / X12 inbound** — **no transport TLS exists** on these channels. Keep them on loopback, or front them with a TLS-terminating proxy / restrict to a trusted segment. See no-TLS hazards.
4. **Outbound connectors** — they default to verified TLS where the protocol supports it. Do **not** set `MEFOR_ALLOW_INSECURE_TLS` in production (it is what lets you weaken verification — see the escape hatch).
5. **Lock down egress** — populate the relevant `[egress].allowed_*` allow-lists so a transform can only send to approved destinations (see egress allow-lists).
6. **Off-box logs + MFA** — confirm audit and operational logs forward to your SIEM/syslog, and confirm your MFA enforcement (local-account MFA on by default; enterprise users via your IdP) before a production cutover.

Channel × TLS posture matrix

Legend: **Bind** = default bind/connect posture · **TLS** = transport encryption support · **Auth** = authentication on the channel · **Egress gate** = the `[egress]` allow-list that confines it · **Off-loopback guarded?** = whether a non-loopback bind is refused without TLS.

Inbound (listeners — the engine binds a socket)

Channel	Bind default	TLS support	Auth	Ingress/egress gate	Off-loopback guarded?
Engine API	`[api].host` = `127.0.0.1`	**Yes** — in-process via `tls_cert_file` / `tls_key_file`, *or* upstream via `tls_terminated_upstream` + `trusted_proxies`; `tls_min_version` (≥1.2); opt-in mTLS via `tls_client_ca_file`; HSTS over https	Session RBAC with required local-account MFA; interface auth via mTLS, OAuth 2.0 client-credentials, and SMART-on-FHIR Backend Services	— (auth-gated)	**Yes** — refused without TLS or a trusted terminator; also refused if auth is disabled on a non-loopback bind
MLLP source	`[inbound].bind_host` = `127.0.0.1`	**Yes** — per-connection opt-in `tls=true` + `tls_cert_file` / `tls_key_file`; opt-in mTLS via `tls_ca_file`; ≥TLS 1.2. **Plaintext by default**	None (MLLP has no app auth)	—	**Yes** — non-loopback plaintext refused
Raw TCP source	`[inbound].bind_host` = `127.0.0.1`	**No** — plaintext only	None	—	**No transport guard** — keep loopback or proxy-terminate
X12 source (ISA/IEA framed)	`[inbound].bind_host` = `127.0.0.1`	**No** — plaintext only (same socket plumbing as raw TCP)	None	—	**No transport guard** — keep loopback or proxy-terminate
File source	local filesystem	n/a (no network)	n/a	—	n/a
Database poll source	connects to `[store]` DB	**Yes** — inherits the store DB connection TLS (`[store].encrypt` default true)	Store DB auth	`[egress].allowed_db`	n/a (outbound DB connection)

Outbound (the engine dials a destination)

Channel	Connect	TLS support	Auth	Egress gate
MLLP destination	dials host:port	**Yes** — per-connection `tls=true`; `tls_verify=true` **default**; client-cert mTLS via `tls_cert_file` / `tls_key_file` + `tls_ca_file`; ≥TLS 1.2	peer HL7 ACK	`[egress].allowed_mllp`
Raw TCP destination	dials host:port	**No** — plaintext only	None	`[egress].allowed_tcp`
X12 destination	dials host:port	**No** — plaintext only (optional TA1)	None	`[egress].allowed_tcp`
REST destination	dials URL	**HTTPS by default** — `verify_tls=true` default (downgrade refused without the escape); cleartext-credential `http` refused; 3xx redirects refused	optional `Authorization` (Basic/Bearer), refused over plaintext	`[egress].allowed_http`
SOAP destination	dials URL	**HTTPS by default** — reuses the REST client + no-redirect opener; per-connection client-cert mTLS; ≥TLS 1.2 in the mTLS context	optional WS-Security `UsernameToken` (Nonce + Timestamp)	`[egress].allowed_http`
FHIR destination	dials URL	**HTTPS by default** — verified TLS; cleartext refused without the escape	OAuth 2.0 client-credentials / SMART-on-FHIR Backend Services	`[egress].allowed_http`
DATABASE destination	dials server:port	**Yes** — SQL Server `Encrypt=yes` **default**, `TrustServerCertificate=false` default (weakened only via the escape)	ODBC `sql` / `integrated` / `entra`	`[egress].allowed_db`
File destination	local filesystem	n/a (no network)	n/a	`[egress].allowed_file_dirs`
RemoteFile destination + source (SFTP / FTPS / FTP)	dials remote host	**Protocol-dependent** — **SFTP** encrypted (SSH host-key verify on by default); **FTPS** explicit TLS; **FTP** plaintext (credentials refused without the escape)	username/password or SSH key	`[egress].allowed_remote`

Internal

Channel	Transport	TLS
Inter-node cluster coordination (active-passive HA)	**Rides the shared `[store]` DB connection — no separate node-to-node socket.** Leadership lease, heartbeat (`last_seen`), and config-version bumps are reads/writes against cluster tables on the same pool.	**= the store DB connection's TLS** (PostgreSQL via asyncpg, SQL Server via aioodbc). Encrypt the store connection and the cluster traffic is encrypted with it.
Store DB connection (PostgreSQL / SQL Server)	asyncpg / aioodbc pool	**Yes** — `[store].encrypt` (default true) + `[store].trust_server_certificate` (default false); weakened only via the escape

---

No-TLS channels — hazards

These channels have **no transport encryption at all** — there is no per-connection `tls` option as there is for MLLP:

- **Raw TCP source/destination** — plaintext, arbitrary framing.
- **X12 source/destination** — plaintext ISA/IEA-framed EDI interchanges.
- **Plain FTP** (RemoteFile `protocol=ftp`, as opposed to SFTP/FTPS) — cleartext protocol; credentials and file contents cross the wire in the clear (the connector refuses credentials over plain FTP unless the escape is set).

Deployment requirement: run these on **loopback only**, or behind a **TLS-terminating proxy / on a trusted, isolated network segment**. If PHI flows over one of them off-host without that protection, it is exposed in cleartext. There is no startup guard that refuses a non-loopback raw-TCP/X12 bind, so this is an **operator responsibility**.

The `MEFOR_ALLOW_INSECURE_TLS` escape hatch

Several connectors **fail closed** on a weakened-TLS or cleartext-credential configuration unless the environment variable `MEFOR_ALLOW_INSECURE_TLS` is set. It exists for **dev / trusted-lab** use only. With it set, these otherwise-refused settings become permitted (each logs a loud warning):

- REST/SOAP/FHIR `verify_tls = false`; cleartext credentials over `http`.
- MLLP outbound `tls_verify = false`.
- DATABASE destination / store: `Encrypt=false` or `TrustServerCertificate=true` (SQL Server), `[store].trust_server_certificate=true` / `[store].encrypt=false`.
- RemoteFile SFTP: accepting an unknown host key.
- Plain-FTP credentials.

Never set `MEFOR_ALLOW_INSECURE_TLS` in production. Its presence is the single switch that turns the fail-closed TLS posture into best-effort.

Egress allow-lists

Outbound destinations are confined by per-protocol allow-lists in `[egress]`. **Default = empty = unrestricted** (backward-compatible). Once a list is **populated, it is fail-closed**: a destination of that type that does not resolve to a

listed `host:port` makes the config **fail at load / reload / start** (validated *after* environment-variable substitution, so dynamic addresses are checked against the resolved value).

Setting	Confines
`[egress].allowed_mllp`	MLLP destinations
`[egress].allowed_tcp`	raw TCP **and** X12 destinations
`[egress].allowed_http`	REST, SOAP, FHIR, and alert-webhook destinations
`[egress].allowed_db`	DATABASE destination + the DB poll source
`[egress].allowed_remote`	RemoteFile SFTP/FTPS/FTP (source + destination)
`[egress].allowed_file_dirs`	File destination directories

For an off-loopback deployment, populate the lists you use so a transform cannot exfiltrate to an unapproved address. Today the deny-by-default behavior is per-list, activated by populating it.

Bind-guard behavior (summary)

- **API:** a non-loopback `[api].host` is refused unless in-process TLS is configured, or `tls_terminated_upstream` + `trusted_proxies` are set; also refused if `[auth].enabled = false`. Override (dev only): `serve --allow-insecure-bind`.
- **MLLP inbound:** a non-loopback MLLP source without `tls=true` is rejected at wiring time, before the engine starts. Override (dev only): `serve --allow-insecure-bind`.

A note on compliance posture

MessageFoundry's controls **support a HIPAA-compliant deployment** when run inside your trusted network with the delegations above documented. MessageFoundry maintains an **internal OWASP ASVS 5.0 Level 3 self-assessment** (212 met / 0 failed / 0 partial / 133 not-applicable of 345 requirements). This is a self-assessment, **not an external audit**; an independent code review and penetration test are recommended at Level 3 but not required, and are planned. MessageFoundry makes no "certified" or "guaranteed" claims, and HIPAA compliance is a property of your overall deployment, not of the engine alone.

Mirth, Corepoint, Cloverleaf, Rhapsody, and Ensemble are trademarks of their respective owners. MessageFoundry is independent and unaffiliated.
