OLLAMA_HOST Resolution Strategy — ALAI Fleet (MC #8476)

OLLAMA_HOST Resolution Strategy — ALAI Fleet

Owner: Skillforge | **Verified:** 2026-07-28 | **Re-verified:** 2026-08-16 | **MC:** #8476 **Source of truth:** `~/system/tools/ollama-host.sh` (implemented 2026-04-20, CodeCraft) + `~/system/architecture/distributed-ai-factory-plan.md` §4

---

1. Why This Exists

ALAI agent scripts call Ollama for local inference (classifiers, embeddings, session summarization). Different machines reach Ollama differently:

- **ANVIL** (Mac Studio, `makinja-sin-mac-studio` / hostname `makinja.local`) — Ollama runs locally on `localhost:11434`.
- **Remote/client hosts** (e.g. ab-mac) — must reach ANVIL over Tailscale at `100.103.49.98:11434`, or fall back to **FORGE** (`10.0.0.2:11434`, LAN-only) if ANVIL is unreachable.
- **Neither reachable** — script must degrade gracefully (warn, do not hang, do not crash agents that have a Claude API fallback).

`ollama-host.sh` centralizes this so scripts don't hardcode an endpoint. Instead they read `$OLLAMA_HOST` (or, in JS, `process.env.OLLAMA_HOST`), which the wrapper resolves and exports once per shell session.

---

2. Resolution Order (as implemented in `ollama-host.sh`)

The live script's actual order is **cache-first**, then hostname, then a network probe ladder — this is slightly more defensive than the original architecture-plan pseudocode (§4 of `distributed-ai-factory-plan.md`), which omits the cache and the local-probe step. Documenting the **real** script here:

```
1. Explicit override   — if $OLLAMA_HOST is already set in the environment, use it
                         as-is. No probing. (Lets a human or CI job pin an endpoint.)
2. Cache check         — if /tmp/ollama-host.cache exists and is < 5 min old:
                           - cached value present and != "NONE"  → export it, done
                           - cached value == "NONE"              → warn, leave unset, done
3. Hostname match      — if `hostname` contains "makinja-sin-mac-studio" or
                         "makinja.local" (i.e. we ARE ANVIL) → localhost:11434
4. Local probe         — curl localhost:11434/api/version, 1s timeout
                         (catches "Ollama running locally for some other reason")
5. Tailscale ANVIL     — curl http://100.103.49.98:11434/api/version, 2s timeout
6. FORGE (LAN only)    — curl http://10.0.0.2:11434/api/version, 1s timeout
                         (fast-fails on non-LAN hosts, e.g. remote/cloud machines)
7. Graceful degrade    — write "NONE" to the cache, print one WARNING to stderr,
                         leave $OLLAMA_HOST unset. Agents with a Claude API fallback
                         continue; Ollama-only agents fail explicitly downstream
                         rather than hanging on an unreachable host.
```

Every successful resolution (steps 3–6) writes its result to the cache file before exporting, so the *next* shell in the same 5-minute window skips the probe ladder entirely (step 2 short-circuits).

---

3. Cache: `/tmp/ollama-host.cache`

- **Path:** `/tmp/ollama-host.cache` (flat text file, single line: either a resolved URL like `http://100.103.49.98:11434`, or the literal string `NONE`).
- **TTL:** 300 seconds (5 minutes), enforced by comparing `stat` mtime against `date +%s`. The script tries macOS `stat -f %m` first, falls back to GNU `stat -c %Y` — safe on both ANVIL (macOS) and any Linux host that sources the same file.
- **Purpose:** avoids re-running the curl probe ladder (up to ~4s of sequential timeouts in the worst case: local 1s + Tailscale 2s + FORGE 1s) on every new shell/subprocess.

- **Failure caching:** a `NONE` result is cached too — so a host with no reachable Ollama doesn't re-probe every session, it just re-warns from cache until the TTL expires or the cache is flushed.

`ollama_flush_cache()`

```
ollama_flush_cache() {
  rm -f "$_OLLAMA_CACHE_FILE"
  echo "[ollama-host] Cache cleared. Run 'source ~/system/tools/ollama-host.sh' to re-
resolve."
}
```

Deletes `/tmp/ollama-host.cache` and prints a reminder to re-source. Call this manually after:

- Tailscale reconnects (e.g. after sleep/wake or VPN flap) and ANVIL becomes reachable again
- Ollama is (re)started on ANVIL or FORGE
- You changed `$OLLAMA_HOST` by hand and want the wrapper to re-probe instead of trusting a stale cached failure

It is exported as a shell function (`export -f ollama_flush_cache`) so it's callable from any subshell in the sourcing session, not just interactively.

There is also `ollama_available()`, a one-line guard scripts can call before an Ollama-only code path:

```
ollama_available() {
  [[ -n "${OLLAMA_HOST:-}" ]] && [[ "${OLLAMA_HOST}" != "NONE" ]]
}
```

---

4. Consumption in Node/JS Agent Scripts

Scripts that were refactored off hardcoded endpoints read the resolved value via `process.env.OLLAMA_HOST` (the shell wrapper exports it, so any JS process spawned from a shell that sourced `ollama-host.sh` inherits it).

Verified counts (`grep -rlE` over `~/system`, `.js/.sh/.ts/.mjs`):

Date	Files reading `process.env.OLLAMA_HOST`

2026-04-20 (task description estimate, unverified)	52
2026-07-28 (first live grep)	149
2026-08-16 (re-verified live grep)	**848**

⚠ **Correction to MC #8476 task description:** the task text (written 2026-04-20, the day `ollama-host.sh` was implemented) states "52 refactored files now read `process.env.OLLAMA_HOST`". No historical record (evidence/reports) of a "52 files" sign-off was found via `discover.js` search, so that original number cannot be reconstructed or verified — treat it as a same-day estimate, not a target to reconcile against.

The count itself is **not stable** — it grew 149 → 848 (5.7x) between 2026-07-28 and 2026-08-16, three weeks apart, tracking the system's overall growth rate rather than a single discrete refactor. Practical implication: do not cite an absolute count from this runbook as current without re-running the grep below; cite the *trend* (rapid, ongoing adoption of the wrapper pattern) instead.

```
grep -rlE "process\.env\.OLLAMA_HOST" ~/system --include="*.js" --include="*.sh" --include="*.ts" --include="*.mjs" | wc -l
```

Not every hardcoded reference is necessarily a bug: some are inside `ollama-host.sh` itself (the probe URLs), health-probe/monitoring scripts that intentionally target a specific host, or docs/comments. A hardcode count is a signal for follow-up, not an automatic defect list — each hit needs eyeballing before "refactor" is filed as a task.

---

5. Shell Integration

Expected integration (per `ollama-host.sh` header comment):

```
[ -f ~/system/tools/ollama-host.sh ] && source ~/system/tools/ollama-host.sh
```

added to `~/.zshrc`.

⚠ **Verified gap (re-checked 2026-08-16, unchanged since 2026-07-28):** `~/.zshrc` on this machine (ANVIL) still has **no reference to `ollama-host.sh` or "ollama" at all** (`grep -in ollama ~/.zshrc` returns nothing, 158-line file). The wrapper is NOT currently auto-sourced on interactive shell start. It must be invoked explicitly (`source ~/system/tools/ollama-host.sh`) or sourced by whatever launches an agent process today.

This is a real gap, not a documentation omission — filed as a follow-up rather than silently assumed fixed. See §6. Note: a *different* overlay mechanism exists (`~/system/config/role-overrides/workstation/env.sh`, added by the `workstation` role's `bootstrap.sh` for non-ANVIL machines like a Mac Air test host — MC #8562) which sets its own `OLLAMA_HOST` and is appended to `~/.zshrc`/`~/.bashrc` by that bootstrap path, but only when the `workstation` role is provisioned. It

targets a stale fallback IP (`100.104.164.86` for FORGE, offline since ~2026-06-14 per `~/system/CLAUDE.md`) and is a separate code path from `ollama-host.sh` — do not conflate the two when auditing shell integration.

---

6. Follow-ups Identified While Writing This Runbook

- **zshrc source line missing** — add the one-line source guard from §5 to `~/.zshrc` so every interactive shell resolves `$OLLAMA_HOST` automatically instead of relying on each agent-launch path to source it independently.
- **"52 files" figure stale** — treat as historical/unverifiable; 149 is the current verified count of `process.env.OLLAMA_HOST` consumers. Re-baseline if this runbook is used as a compliance reference.
- **Hardcode audit** — a full-tree count of remaining `localhost:11434`/`127.0.0.1:11434` references (excluding `ollama-host.sh`'s own probe code) would tell CodeCraft how much refactor work is actually left; not completed here due to `~/system`'s size (23G) making a full recursive grep slow — a scoped follow-up task should target it.
- **Tailscale ACL** — per architecture plan §4, ANVIL's Ollama port was not reachable from other Tailscale peers as of the plan's writing. This runbook does not re-verify that ACL state; treat §4's Tailscale ACL recommendation as still open unless confirmed otherwise.

Revision #3
Created 2026-07-28 07:14:29 UTC by John
Updated 2026-08-16 02:08:51 UTC by John