

Eugene Agent Health Route — End-to-End Flow

Developer walkthrough: HTTP probe → FastAPI router → LLM key + MCP /health probe + JWT secret → JSON status envelope → k8s readiness gating for the chat UI

Audience

Engineers wiring up container healthchecks or k8s readiness probes against eugene-agent-ws — the WebSocket / REST agent tier that the Next.js chat UI talks to. This is the sibling of the eugene-ws health route, but it has a different dependency surface: there is no Neo4j driver here, the agent instead depends on an LLM provider and on the eugene-mcp server. Every reference uses the form `path/to/file.py:line`.

How this differs from the src (eugene-ws) health route

- Same two-endpoint split (liveness vs readiness) and same OK/DEGRADED envelope shape, so orchestrator wiring is identical.
- Service identifier is eugene-agent-ws, not eugene-ws — useful when scraping mixed logs.
- Readiness checks three things instead of one: LLM provider key (ANTHROPIC_API_KEY or OPENAI_API_KEY), MCP reachability (EUGENE_MCP_SERVER_URL), and JWT signing secret (EUGENE_CLIENT_SECRET).
- FastAPI app mounts under `root_path="/agent/api"`, so the URLs exposed to the outside are `/agent/api/health` and `/agent/api/health/ready` (the router itself still declares unprefixes `/health` paths).

Reference endpoints

- GET `/health` — liveness probe. Returns 200 with a constant dict as long as the uvicorn process is accepting requests. No env reads, no IO.
- GET `/health/ready` — readiness probe. Returns 200 only when (a) at least one LLM API key is present, (b) the MCP server at EUGENE_MCP_SERVER_URL answers its own `/health` with a status code below 500, and (c) EUGENE_CLIENT_SECRET is configured. Otherwise 503 with a per-check diagnostic envelope.

0. Probe model (read this first)

This route has no Neo4j schema of its own; the "model" is the contract between the HTTP probe, the orchestrator, and the agent's external dependencies. Four rules:

- **Liveness is side-effect free.** GET /health returns a constant dict. It must not call MCP, must not check env vars, must not reach out to the LLM provider — any of those would couple liveness to a downstream failure and turn transient outages into restart-storms.
- **Readiness exercises real dependencies.** GET /health/ready actually opens an httpx client and GETs the MCP server's own health endpoint. A passing probe therefore proves not just "URL is set" but "the MCP container is up and answering on the configured URL".
- **LLM key presence, not LLM call.** The readiness probe checks ANTHROPIC_API_KEY/OPENAI_API_KEY *existence* only. It does not round-trip to Anthropic or OpenAI — that would cost money on every probe and couple readiness to a third party's uptime. A wrong key still passes readiness; the first chat turn will surface it.
- **Status envelope is structured.** The response carries status, service, and a checks map. Per-check entries carry ok plus check-specific fields (anthropic/openai booleans, status_code/latency_ms, or error).

What the agent service actually depends on

- **LLM provider** — at least one of ANTHROPIC_API_KEY or OPENAI_API_KEY must be set. The Strands agent inside eugene-agent-ws is provider-agnostic; readiness only insists that *some* credential is available.
- **eugene-mcp** — the agent calls MCP tools over HTTP at EUGENE_MCP_SERVER_URL. Without it the chat tier cannot answer any biomedical query.
- **JWT signing secret** — EUGENE_CLIENT_SECRET signs the short-lived tokens minted by the auth router; without it the WebSocket handshake cannot authenticate the UI.
- **(implicit) the Next.js UI** — not probed; the UI is the consumer, not a dependency.

Status values

```
GET /health      -> { status: "OK",          service: "eugene-agent-ws" }
GET /health/ready -> { status: "OK"|"DEGRADED",
                      service: "eugene-agent-ws",
                      checks: { llm_provider: {...},
                                mcp:           {...},
                                jwt_secret:    {...} } }
```

DEGRADED is paired with HTTP 503 Service Unavailable; OK with 200. Orchestrators that only look at the status code do the right thing; humans get the JSON for triage.

1. HTTP entry

`agents/eugene-agent-ws/src/router/health_router.py`

- Router declared at line 16 with no prefix and tag `health`: `APIRouter(prefix="", tags=["health"])`. Routes mount at the service root; the app-level `root_path="/agent/api"` then prepends `/agent/api` externally.
- Two coroutines: `liveness()` at line 23 and `readiness(response)` at line 28.
- Module-level imports include `httpx`, `os`, and `time`. No driver/client is constructed at import time — the `httpx` client is created per-request inside `readiness` (see section 2) so import-time is safe even when env vars are not set.

Verbatim `liveness` (`health_router.py:22-24`)

```
@router.get("/health")
async def liveness() -> dict[str, Any]:
    return {"status": "OK", "service": "eugene-agent-ws"}
```

Three lines, no IO. The route returns a fresh dict on every call; FastAPI serializes it with the default JSON encoder.

Verbatim `readiness` signature (`health_router.py:27-28`)

```
@router.get("/health/ready")
async def readiness(response: Response) -> dict[str, Any]:
```

The handler takes a FastAPI `Response` so it can mutate `response.status_code` in place — this is how it flips to 503 without raising an exception (which would lose the structured body).

Registration (`eugene_chat_ws.py:30-37, 53-57`)

```
def add_routers(app: FastAPI) -> None:
    from router.auth import auth_router
    from router import root_router, health_router
    from query.router import chat_query_agent_router

    routers = [auth_router, root_router, health_router,
                chat_query_agent_router]
    for router in routers:
        app.include_router(router.router)

app = FastAPI(
    root_path="/agent/api",
    title="Eugene Agent WS",
    version="2.0.0",
)
```

- `health_router` is included third — after `auth` and `root` — but because it carries no prefix the order does not affect routing.
- The app-level `root_path="/agent/api"` means an external probe must hit `http://host:port/agent/api/health/ready` (the ingress strips this prefix before handing the request to uvicorn). Container-local healthchecks that bypass the ingress use the raw `/health` path.
- The router carries no auth dependency — probes must succeed before any JWT verification is exercised.

2. Handler internals — what gets checked

`readiness()` runs three independent checks sequentially. Verbatim from `health_router.py:27-78`:

```
@router.get("/health/ready")
async def readiness(response: Response) -> dict[str, Any]:
    checks: dict[str, dict[str, Any]] = {}
    overall_ok = True

    # 1. LLM provider: at least one API key must be present
    has_anthropic = bool(os.environ.get("ANTHROPIC_API_KEY"))
    has_openai = bool(os.environ.get("OPENAI_API_KEY"))
    checks["llm_provider"] = {
        "ok": has_anthropic or has_openai,
        "anthropic": has_anthropic,
        "openai": has_openai,
    }
    overall_ok &= checks["llm_provider"]["ok"]

    # 2. MCP server reachability
    mcp_url = os.environ.get("EUGENE_MCP_SERVER_URL", "")
    if mcp_url:
        start = time.perf_counter()
        try:
            probe_url = mcp_url.rstrip("/")
            if probe_url.endswith("/mcp"):
                probe_url = probe_url[:-4]
            verify_ssl = os.environ.get(
                "EUGENE_MCP_VERIFY_SSL", "true"
            ).lower() != "false"
            async with httpx.AsyncClient(
                verify=verify_ssl, timeout=3.0
            ) as client:
                resp = await client.get(f"{probe_url}/health")
            checks["mcp"] = {
                "ok": resp.status_code < 500,
                "status_code": resp.status_code,
                "latency_ms": int((time.perf_counter() - start) * 1000),
            }
        except Exception as exc:
            checks["mcp"] = {"ok": False,
                             "error": f"{type(exc).__name__}: {exc}"}
        overall_ok &= checks["mcp"]["ok"]
    else:
        checks["mcp"] = {"ok": False,
                         "error": "EUGENE_MCP_SERVER_URL not set"}
        overall_ok = False

    # 3. JWT signing secret configured
    secret_configured = bool(os.environ.get("EUGENE_CLIENT_SECRET"))
    checks["jwt_secret"] = {"ok": secret_configured}
    overall_ok &= secret_configured

    if not overall_ok:
        response.status_code = status.HTTP_503_SERVICE_UNAVAILABLE

    return {
        "status": "OK" if overall_ok else "DEGRADED",
        "service": "eugene-agent-ws",
        "checks": checks,
    }
```

Per-check semantics

- `checks.llm_provider`. Two boolean env reads. `ok` is the OR of `anthropic` and `openai`, so a single configured provider is sufficient. The individual booleans are surfaced so a human can see *which* provider is wired.
- `checks.mcp`. The MCP URL is normalised: trailing `/` is stripped, then a trailing `/mcp` suffix is removed so that `https://host/mcp` (the JSON-RPC endpoint the agent actually calls) becomes `https://host` (the FastAPI root that hosts `/health`). A 3-second timeout caps the probe latency; any exception is caught and surfaced as `error: "ExceptionClass: message"`. `verify_ssl` honours an opt-out env (`EUGENE_MCP_VERIFY_SSL=false`) for local self-signed deployments.
- `checks.mcp.ok`. Defined as `resp.status_code < 500` — meaning a 404 from the MCP server is treated as "healthy enough". Rationale: the goal is to prove the server is up and responding, not to assert that this exact path exists; some MCP deployments do not expose a sibling `/health` route.
- `checks.jwt_secret`. Single env existence check on `EUGENE_CLIENT_SECRET`. No format validation — an empty-but-set secret would pass here but later fail signature verification at the auth router.

Note: every `overall_ok &= ...` uses a bitwise AND on booleans, equivalent to logical AND but always evaluating both sides — intentional, so that adding a fourth check cannot accidentally short-circuit and skip its diagnostic.

3. Response model

There is no Pydantic model. Both handlers are typed -> `dict[str, Any]` and FastAPI serializes the dict directly. Four concrete shapes occur in practice:

Liveness, healthy (always)

HTTP/1.1 200 OK

Content-Type: application/json

```
{ "status": "OK", "service": "eugene-agent-ws" }
```

Readiness, healthy

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "status": "OK",
  "service": "eugene-agent-ws",
  "checks": {
    "llm_provider": { "ok": true, "anthropic": true, "openai": false },
    "mcp":          { "ok": true, "status_code": 200, "latency_ms": 12 },
    "jwt_secret":   { "ok": true }
  }
}
```

Readiness, degraded (MCP unreachable)

HTTP/1.1 503 Service Unavailable

Content-Type: application/json

```
{
  "status": "DEGRADED",
  "service": "eugene-agent-ws",
  "checks": {
    "llm_provider": { "ok": true, "anthropic": true, "openai": false },
    "mcp": {
      "ok": false,
      "error": "ConnectError: All connection attempts failed"
    },
    "jwt_secret": { "ok": true }
  }
}
```

Readiness, misconfigured (no LLM key, no MCP URL)

HTTP/1.1 503 Service Unavailable

Content-Type: application/json

```
{
  "status": "DEGRADED",
  "service": "eugene-agent-ws",
  "checks": {
    "llm_provider": { "ok": false, "anthropic": false, "openai": false },
    "mcp":          { "ok": false, "error": "EUGENE_MCP_SERVER_URL not set" },
    "jwt_secret":   { "ok": false }
  }
}
```

- status: literal OK or DEGRADED — the only two values.
- service: constant "eugene-agent-ws" — distinguishes this probe from eugene-ws and eugene-mcp when scraping mixed logs.

- `checks.<name>.error` is a free-form "ExceptionClass: message" string — useful for humans, not for machines. Programmatic gating should key off the HTTP status code, not the error text.
- All three check entries are always present on the GET `/health/ready` response, even when one short-circuits — consumers can index without defensive guards.

4. Use in production — k8s readiness gating for the chat UI

The agent tier sits between the Next.js chat UI and the MCP server. Readiness gating here directly controls whether the chat UI can complete its WebSocket handshake.

Kubernetes (recommended pattern)

```
livenessProbe:
  httpGet: { path: /agent/api/health,      port: 8000 }
  initialDelaySeconds: 15
  periodSeconds: 10
  failureThreshold: 3      # -> restart pod
readinessProbe:
  httpGet: { path: /agent/api/health/ready, port: 8000 }
  initialDelaySeconds: 5
  periodSeconds: 5
  failureThreshold: 2      # -> remove from Service endpoints
```

- **Readiness gates chat UI traffic.** A 503 from GET /health/ready drops the agent-ws pod from its Service endpoints; the ingress in front of the Next.js UI then returns 503 for the WebSocket upgrade, and the UI surfaces a "chat unavailable" banner. No user is allowed to start a conversation that the agent cannot service.
- **Liveness only restarts on wedged process.** GET /health fails only if the uvicorn event loop has stopped responding — an unambiguous signal that a restart is the right remediation. MCP outages and missing LLM keys do *not* trigger restarts.
- **Probes are exempt from auth.** The router carries no auth dependency — do not put it behind the global JWT verification or the kubelet will see 401 and consider the pod unhealthy. `root_path="/agent/api"` does not affect this; auth middleware is the actual gate.
- **Probe path must include the root_path.** Because the app declares `root_path="/agent/api"`, the in-cluster probe path is `/agent/api/health/ready`, not `/health/ready`. Setting this wrong is the most common k8s wiring mistake on this service.
- **Boot ordering.** The agent does not block on MCP at startup; GET /health/ready is the only place that synchronously checks MCP. This means agent-ws can start before MCP is ready and will simply stay out of rotation until the dependency comes up — no explicit `depends_on` ordering required.

docker-compose (local stack)

The compose file points its container healthcheck at GET /health (cheap, never false-negative) and lets downstream gating in the Next.js UI rely on the actual WebSocket connect to discover whether MCP is up. The GET /health/ready endpoint is still useful for ad-hoc `curl` diagnosis during local development.

5. Suggested debugging walk

- **Confirm liveness.** `curl -i http://localhost:18001/health` (container-local) or `curl -i http://localhost:18001/agent/api/health` (through ingress). Expect a 200 with `service: "eugene-agent-ws"` within ~15s of container start.
- **Verify all three dependency checks pass.** `curl -sS ../health/ready | jq .checks`. Expect `llm_provider.ok`, `mcp.ok`, and `jwt_secret.ok` all true, plus a `mcp.latency_ms` field — that confirms the httpx round-trip to `EUGENE_MCP_SERVER_URL` actually happened.
- **Simulate MCP outage.** `docker-compose stop eugene_mcp`, then re-hit `GET /health/ready`. Expect HTTP 503 with `checks.mcp.error` like `"ConnectError: ..."`. Liveness should still be 200 — confirms the split.
- **Simulate missing LLM key.** Unset both `ANTHROPIC_API_KEY` and `OPENAI_API_KEY` in `docker.env` and restart `agent-ws`. `GET /health/ready` returns 503 with `llm_provider.ok: false` and both provider booleans false — no MCP call is skipped (the checks run independently).
- **Trace the MCP probe path.** Breakpoint at `agents/eugene-agent-ws/src/router/health_router.py:53` (`await client.get(f"{{probe_url}}/health")`). Confirm that `probe_url` equals the MCP host root, not the `/mcp` JSON-RPC endpoint — the strip-suffix logic at lines 48-50 is what makes this work for either form of `EUGENE_MCP_SERVER_URL`.

6. Reference index

HTTP entry

<code>agents/eugene-agent-ws/src/router/health_router.py</code>	<code># GET /health (L22), GET /health/ready (L27)</code>
<code>agents/eugene-agent-ws/src/eugene_chat_ws.py</code>	<code># add_routers() L30, app root_path L53</code>

Dependencies probed

<code>env: ANTHROPIC_API_KEY / OPENAI_API_KEY</code>	<code># llm_provider check (L33-40)</code>
<code>env: EUGENE_MCP_SERVER_URL</code>	<code># mcp check target (L43)</code>
<code>env: EUGENE_MCP_VERIFY_SSL (default true)</code>	<code># httpx verify= (L51)</code>
<code>env: EUGENE_CLIENT_SECRET</code>	<code># jwt_secret check (L67)</code>
<code>httpx.AsyncClient(timeout=3.0).get(probe_url + /health)</code>	<code># MCP reachability (L52-53)</code>

Probe consumers

<code>k8s livenessProbe -> /agent/api/health</code>	
<code>k8s readinessProbe -> /agent/api/health/ready</code>	<code>(gates chat UI traffic)</code>
<code>docker-compose healthcheck -> /health</code>	<code>(liveness only)</code>

Sibling routes (for contrast)

<code>src/router/health_router.py</code>	<code># eugene-ws variant: probes Neo4j only</code>
<code>agents/eugene-agent-ws/src/router/root_router.py</code>	<code># GET / -- service banner</code>
<code>agents/eugene-agent-ws/src/router/auth/</code>	<code># consumes EUGENE_CLIENT_SECRET</code>