

Eugene Health Route — End-to-End Flow

Developer walkthrough: HTTP probe → FastAPI router → Neo4j driver `verify_connectivity` → JSON status envelope → container orchestrator gating

Audience

Engineers wiring up container healthchecks, k8s probes, or external monitors against `eugene-ws`. The health route is the smallest router in the codebase (two endpoints, fifty-three lines) but it is the contract that `docker-compose` and any future orchestrator use to decide whether the API is alive and whether traffic can be routed to it. Every reference uses the form `path/to/file.py:line`.

Reference endpoints

- `GET /health` — liveness probe. Always returns 200 as long as the FastAPI process is accepting requests.
- `GET /health/ready` — readiness probe. Verifies the Neo4j driver can reach the configured `NEO4J_URI` and returns 200 only when every dependency check passes; otherwise 503 with a per-check diagnostic envelope.

Probe model

Eugene splits the two concerns deliberately:

- **Liveness** answers *is the Python process wedged?* — no external IO, no driver calls, no env lookups. A failure means restart the container.
- **Readiness** answers *can this replica serve traffic right now?* — it actually round-trips to Neo4j. A failure means stop routing traffic, but do *not* restart (a flaky Neo4j or an unset env var is not fixed by a bounce).
- Both endpoints are tagged `health` and live on the unprefix router (`APIRouter(prefix="", tags=["health"])`). No authentication is required — probes must work before any Entra ID middleware is exercised.

0. Probe model (read this first)

Unlike the other Eugene guides this route has no Neo4j schema of its own; the "model" is the contract between the HTTP probe and the orchestrator. Three rules:

- **Liveness is cheap and side-effect free.** GET /health returns a constant dict. It must not allocate driver connections, read env vars, or touch the filesystem — any of those would couple liveness to a downstream failure mode and turn restarts into a self-inflicted outage.
- **Readiness exercises the real dependency.** GET /health/ready calls `driver.verify_connectivity()` which opens a Bolt session against Neo4j. A passing probe therefore proves not just "URI is set" but "credentials work and the server answered".
- **Status envelope is structured.** The response always includes status, service, and a checks map. Per-check entries carry ok, optional latency_ms, and optional error so a human reading `{h.c('curl /health/ready')}` can see exactly which dependency is sour.

Status values

```
GET /health      -> { status: "OK",          service: "eugene-ws" }
GET /health/ready -> { status: "OK"|"DEGRADED",
                      service: "eugene-ws",
                      checks: { neo4j: { ok, latency_ms?, error? } } }
```

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

1. HTTP entry

`src/router/health_router.py`

- Router declared at line 11 with no prefix and tag `health`: routes mount at the service root.
- Two coroutines: `liveness()` at line 15 and `readiness(response)` at line 20.
- No DI, no module-load side effects. The driver is constructed lazily inside the readiness handler (see section 2) so importing the module is safe even without env vars set — important because the router is registered unconditionally at boot.

Verbatim liveness (`health_router.py:14-16`)

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

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

Verbatim readiness signature (`health_router.py:19-20`)

```
@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_ws.py:33, 58`)

```
def add_routers(app: FastAPI) -> None:
    from router.auth import auth_router
    from router import root_router, health_router, release_notes_router
    ...
    routers = [
        auth_router,
        root_router,
        health_router,
        ...
    ]
    for router in routers:
        app.include_router(router.router)
```

`health_router` is included third — after `auth` and `root` — but because it carries no prefix, the order does not affect routing. It is listed early as a signal that the probes are first-class.

2. Handler internals — what gets checked

`readiness()` is the only handler that does any work. It is small enough to read verbatim:

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

    # Neo4j reachability via existing connection factory
    uri = os.environ.get("NEO4J_URI", "")
    if uri:
        start = time.perf_counter()
        try:
            from graph.infra.db.graph_db_connection_factory import (
                GraphDbConnectionFactory,
            )

            driver = GraphDbConnectionFactory.remote_neo4j_instance_from_env()
            driver.verify_connectivity()
            checks["neo4j"] = {
                "ok": True,
                "latency_ms": int((time.perf_counter() - start) * 1000),
            }
        except Exception as exc:
            checks["neo4j"] = {"ok": False, "error": f"{type(exc).__name__}: {exc}"}
            overall_ok &= checks["neo4j"]["ok"]
    else:
        checks["neo4j"] = {"ok": False, "error": "NEO4J_URI not set"}
        overall_ok = False

    if not overall_ok:
        response.status_code = status.HTTP_503_SERVICE_UNAVAILABLE

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

What it actually verifies

- **NEO4J_URI env var presence.** An unset/empty URI short-circuits to `ok: False, error: "NEO4J_URI not set"` without attempting any IO. This catches the common "forgot to mount `docker.env`" failure mode immediately.
- **Neo4j Bolt reachability.** `GraphDbConnectionFactory.remote_neo4j_instance_from_env()` (at `src/graph/infra/db/graph_db_connection_factory.py:53`) also reads `NEO4J_USERNAME` and `NEO4J_PASSWORD` from the environment — so a `KeyError` there is caught by the same `try/except` and surfaced as the error string. The driver is cached on the factory, so repeated probes do not reopen pools.
- **Connectivity round-trip.** `driver.verify_connectivity()` is a synchronous Bolt handshake against the server. It validates credentials, routing, and TLS (if configured). Any `neo4j-driver` exception class is caught by the broad `except Exception` — this is deliberate so a probe never 500s, only 503s.
- **Latency capture.** `time.perf_counter()` bookends the call; the integer millisecond reading lands in `checks.neo4j.latency_ms` on the happy path. It is omitted on failure (the timer end-point is never reached).

Note: `overall_ok &= checks["neo4j"]["ok"]` uses a bitwise AND on booleans, which is equivalent to logical AND but evaluates both sides — intentional, so new checks added later cannot accidentally short-circuit and skip their diagnostics.

3. Response model

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

Liveness, healthy (always)

HTTP/1.1 200 OK

Content-Type: application/json

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

Readiness, healthy

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "status": "OK",
  "service": "eugene-ws",
  "checks": {
    "neo4j": { "ok": true, "latency_ms": 7 }
  }
}
```

Readiness, degraded (Neo4j unreachable)

HTTP/1.1 503 Service Unavailable

Content-Type: application/json

```
{
  "status": "DEGRADED",
  "service": "eugene-ws",
  "checks": {
    "neo4j": {
      "ok": false,
      "error": "ServiceUnavailable: Could not connect to bolt://neo4j:7687"
    }
  }
}
```

Readiness, misconfigured (URI unset)

HTTP/1.1 503 Service Unavailable

Content-Type: application/json

```
{
  "status": "DEGRADED",
  "service": "eugene-ws",
  "checks": {
    "neo4j": { "ok": false, "error": "NEO4J_URI not set" }
  }
}
```

- `status`: literal OK or DEGRADED — the only two values. Easy to grep, easy to alert on.
- `service`: the constant "eugene-ws" — lets downstream aggregators distinguish this probe from the sibling agent / MCP services 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.

4. Use in k8s / docker-compose

docker-compose (docker-compose.yml:92-97)

```
eugene_ws:
  ...
  depends_on:
    neo4j:
      condition: service_healthy
  healthcheck:
    test: ["CMD-SHELL",
      "python -c \"import urllib.request; \"
      \"urllib.request.urlopen('http://localhost:8000/health')\""]
    interval: 10s
    timeout: 5s
    retries: 5
    start_period: 15s
```

- Compose hits /health (liveness), not /health/ready. The reason: in local-stack mode the compose file already gates eugene_ws on neo4j: condition: service_healthy, so by the time Python starts, Neo4j is up. Liveness is enough to know the uvicorn process bound the port.
- Downstream services (eugene_mcp, eugene_agent_ws) then chain depends_on: eugene_ws: condition: service_healthy — this is how the agent tier waits for the API tier without polling.
- start_period: 15s gives the Python interpreter and FastAPI router-import block (which is non-trivial — see eugene_ws.py: 31-78) time to come up before failed probes start counting against the retry budget.

Kubernetes (recommended pattern)

There is no checked-in k8s manifest yet, but the route is shaped for the canonical liveness/readiness split. /health maps to livenessProbe; /health/ready maps to readinessProbe:

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

- Readiness gating: a 503 from /health/ready causes the kubelet to drop the pod from Service endpoints, so kube-proxy / ingress stop sending it user traffic. The pod keeps running — no restart — and the next successful probe re-adds it to the rotation. This is the right behaviour when Neo4j is briefly unreachable: do not restart-storm the API tier.
- Liveness gating: a sustained failure on /health (which only fails if the Python event loop is wedged) triggers a pod restart. The endpoint is intentionally trivial so this signal is unambiguous.
- Probes are exempt from auth because the router carries no auth dependency — do not put them behind the global get_current_user dependency or the kubelet will see 401 and consider the pod unhealthy.

5. Suggested debugging walk

- Bring the stack up: `docker-compose up eugene_neo4j eugene_ws`. Hit `curl -i http://localhost:18000/health` — expect a 200 within ~15s of the container starting.
- Verify readiness: `curl -sS http://localhost:18000/health/ready | jq ..` Expect status: OK with a `latency_ms` field on the `neo4j` check — that confirms the Bolt round-trip actually happened.
- Simulate Neo4j outage: `docker-compose stop eugene_neo4j`, then re-hit `/health/ready`. Expect HTTP 503 and a `checks.neo4j.error` like "ServiceUnavailable: ...". Liveness (`{h.c('/health')}`) should still be 200 — confirms the split.
- Simulate misconfiguration: `unset NEO4J_URI` in `docker.env` and restart `eugene_ws`. `/health/ready` should return 503 with error: "NEO4J_URI not set" immediately (no Bolt attempt).
- Trace the driver path: breakpoint at `src/router/health_router.py:34` (`driver.verify_connectivity()`). Step into the factory at `graph_db_connection_factory.py:53` to confirm the process-wide driver is reused (it is — the factory caches in `cls._NEO4J_DRIVER`, so repeated probes do not leak Bolt connections).

6. Reference index

HTTP entry

`src/router/health_router.py`
`src/eugene_ws.py`

GET /health (L14), GET /health/ready (L19)
`add_routers()` L31, registered L58

Dependencies pinged

`src/graph/infra/db/graph_db_connection_factory.py`

env vars: `NEO4J_URI`, `NEO4J_USERNAME`, `NEO4J_PASSWORD`

`remote_neo4j_instance_from_env` (L53)
caches driver in `cls._NEO4J_DRIVER` (L49)
read by the factory

Probe consumers

`docker-compose.yml`

`eugene_ws` healthcheck L92-97 (uses `/health/ready`)
`eugene_mcp` depends_on `service_healthy` L11

(no k8s manifest checked in yet; pattern in section 4)

Related routers (for contrast)

`src/router/root_router.py`
`src/router/release_notes_router.py`

GET / -- service banner
GET /release-notes