

DNS APUs

Two PC Engines APU2 boxes (apu01 / apu02) run Unbound as a household anycast recursive resolver at **10.53.53.53**. FRR advertises that address over OSPF to the CCR2116 core, which ECMP-hashes clients onto whichever APUs are currently advertising. This book is the operator view: architecture, the anycast self-heal that closes the Sep 2026 OSPF-Full / empty-FIB blackholes, and the HOST-APU-01 listen map plus how the hp02 dashboard still scrapes metrics after Prometheus and node_exporter moved to localhost.

- [Architecture](#)
- [Anycast self-heal](#)
- [Hardening and dashboard scrape](#)

Architecture

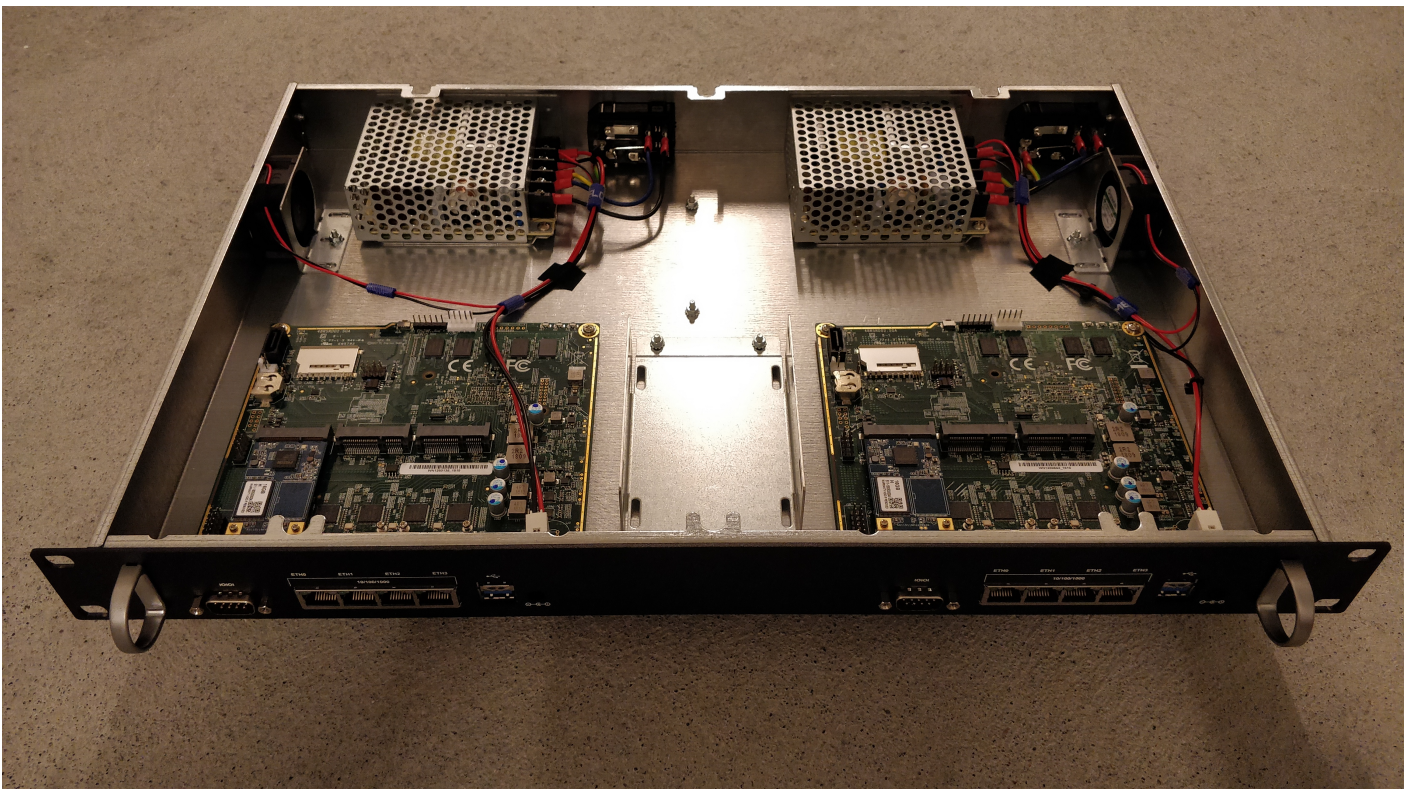
DNS APUs — architecture

Two PC Engines **APU2** boards in a 1U dual-slot 19" chassis are the household recursive DNS. Clients do not pick a box. DHCP (and the core intercept of rogue DNS) hands out **10.53.53.53**. Each APU puts that address on `lo` and FRR advertises it as an OSPF stub. The **CCR2116** core installs an ECMP pair and hashes each source/destination pair onto one arm.

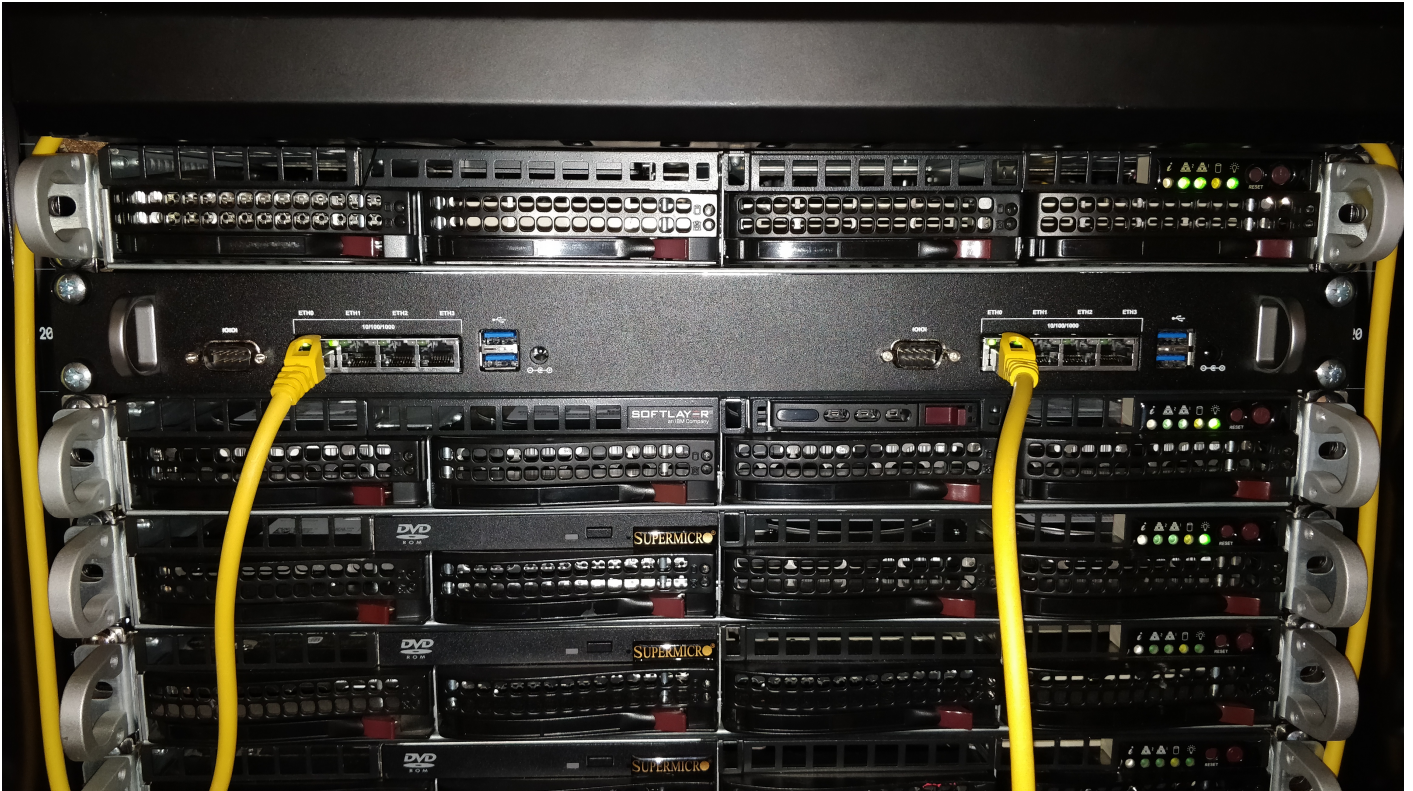
This page is the map. The self-heal that withdraws a sick arm is [Anycast self-heal](#). What still listens where, after HOST-APU-01, is [Hardening and dashboard scrape](#). The rest of the household fabric (islands, VLANs, dual WAN, VRRP) is [MikroTik fabric](#).

Rack photos

Hardware photos are the original 2019 gallery shots from [racked the apu2s](#). The software on the boards has moved on (Unbound + FRR anycast, not Pi-hole/Traefik); the metal is the same.



These are my apu2 DNServers running pihole on docker with traefik 2.0



Dual APU2 19" rack unit in the cabinet (yellow uplinks on ETH0)

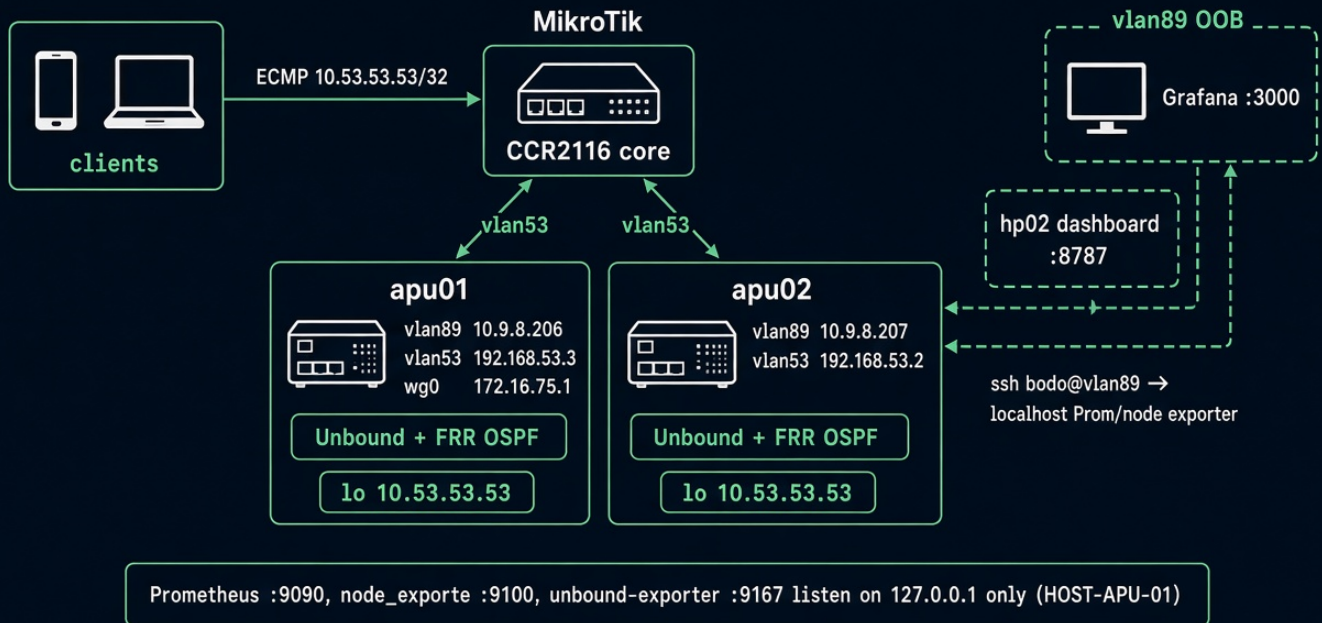
Case parts

- the bare case (326744)
- the power supplies (326747)
- two fans (SUN-HA40201V4-1)
- two fan mounts (311857)

The boards were remounted from the standard APU case; the original aluminium heat spreader does not reuse — use APUCOOL. The 326744 kit includes the power socket that still needs to be soldered to the APU2; polarity matters.

Anycast architecture

DNS APUs — anycast 10.53.53.53



Addressing

Box	vlan89 (OOB)	vlan53 (OSPF / Unbound)	Other	Grafana
apu01	10.9.8.206	192.168.53.3	wg0 172.16.75.1 (home WireGuard)	http://10.9.8.206:3000/
apu02	10.9.8.207	192.168.53.2	—	http://10.9.8.207:3000/
anycast	—	10.53.53.53 on lo of each APU	advertised via FRR OSPF	—
CCR2116 core	10.9.8.252	192.168.53.1	ECMP for 10.53.53.53/32	—
hp02 dashboard	10.9.8.253	—	also 172.16.62.253; UI :8787	—

OSPF/BFD peers on vlan53: CCR2116 192.168.53.1 ↔ apu01 192.168.53.3 and apu02 192.168.53.2. Both APUs run Ubuntu 24.04, Unbound (DNSSEC hardened), FRR, and BFD.

How a query lands

```
flowchart LR
    C[Clients / DHCP stubs] -->| "UDP/TCP 53 → 10.53.53.53" | CORE[CCR2116 core]
    CORE -->| "ECMP 10.53.53.53/32" | A1[apu01 Unbound]
    CORE -->| "ECMP 10.53.53.53/32" | A2[apu02 Unbound]
    A1 --- L01["lo 10.53.53.53"]
    A2 --- L02["lo 10.53.53.53"]
    CORE -->| "vlan53 OSPF + BFD" | A1
    CORE -->| "vlan53 OSPF + BFD" | A2
```

RouterOS hashes ECMP **per source/destination pair**. One source address samples one arm. That is why `/dns` probes the anycast from more than one hp02 address (`172.16.62.253` and `10.9.8.253`) — a single probe cannot prove both APUs.

The core also intercepts rogue DNS (dst-nat of UDP/TCP 53 not already aimed at 10.53.53.53, from sources not on `DNS-SERVERS`) and sends it to the anycast. Production DHCP DNS is 10.53.53.53, not a vlan89 Grafana address.

What each box runs

- **Unbound** — recursive resolver, DNSSEC (`harden-glue` , `harden-dnssec-stripped`). After HOST-APU-01 it is **not** bound to `0.0.0.0` ; it listens on the fabric addresses (vlan89, vlan53, the anycast on `lo` , and on apu01 also `wg0`). ACL is fabric prefixes, not all of RFC1918.
- **FRR OSPF** — advertises `10.53.53.53/32` as a stub of `lo` . Removing the address from `lo` withdraws it from OSPF within about a second; putting it back re-advertises. Both boxes are hardened with `no zebra nexthop kernel enable` after the Sep 2026 kernel-NHG blackholes.
- **Grafana :3000 — vlan89 only** (10.9.8.206 / .207). Not on vlan53, not on `wg0` .
- **Prometheus :9090, node_exporter :9100, unbound-exporter :9167 — 127.0.0.1 only**. The hp02 dashboard scrapes them over SSH as `bodo@vlan89` to `127.0.0.1` (see the hardening page). LAN `:9090` / `:9100` connection-refuses on purpose.

Operator surfaces

Surface	URL	What it is
Grafana apu01	http://10.9.8.206:3000/	On-box dashboards, vlan89 only
Grafana apu02	http://10.9.8.207:3000/	Same
hp02 dashboard	<code>:8787</code> on hp02 (<code>/apu</code> , <code>/dns</code> , <code>/docker</code> , <code>/sites</code>)	Fleet view. Metrics over SSH after HOST-APU-01

Surface	URL	What it is
/apu	DASH-32 / REQ-DASH32	Three angles: box (SSH), core (REST ECMP/OSPF), wire (UDP probes)
/dns	REQ-D5	Unbound PromQL via SSH → 127.0.0.1:9090
/docker	REQ-DF5	APU pulse via SSH → 127.0.0.1:9100 (no Docker on the APUs)

SSH to the APUs is as `bodo`, pubkey only, password auth off, root login off. Use **vlan89** (`10.9.8.206 / .207`) — that path kept answering when vlan53 had no return route.

What this book is not

It is not a runbook to rewrite FRR, nftables, or Unbound on the live boxes. The applied shape lives in:

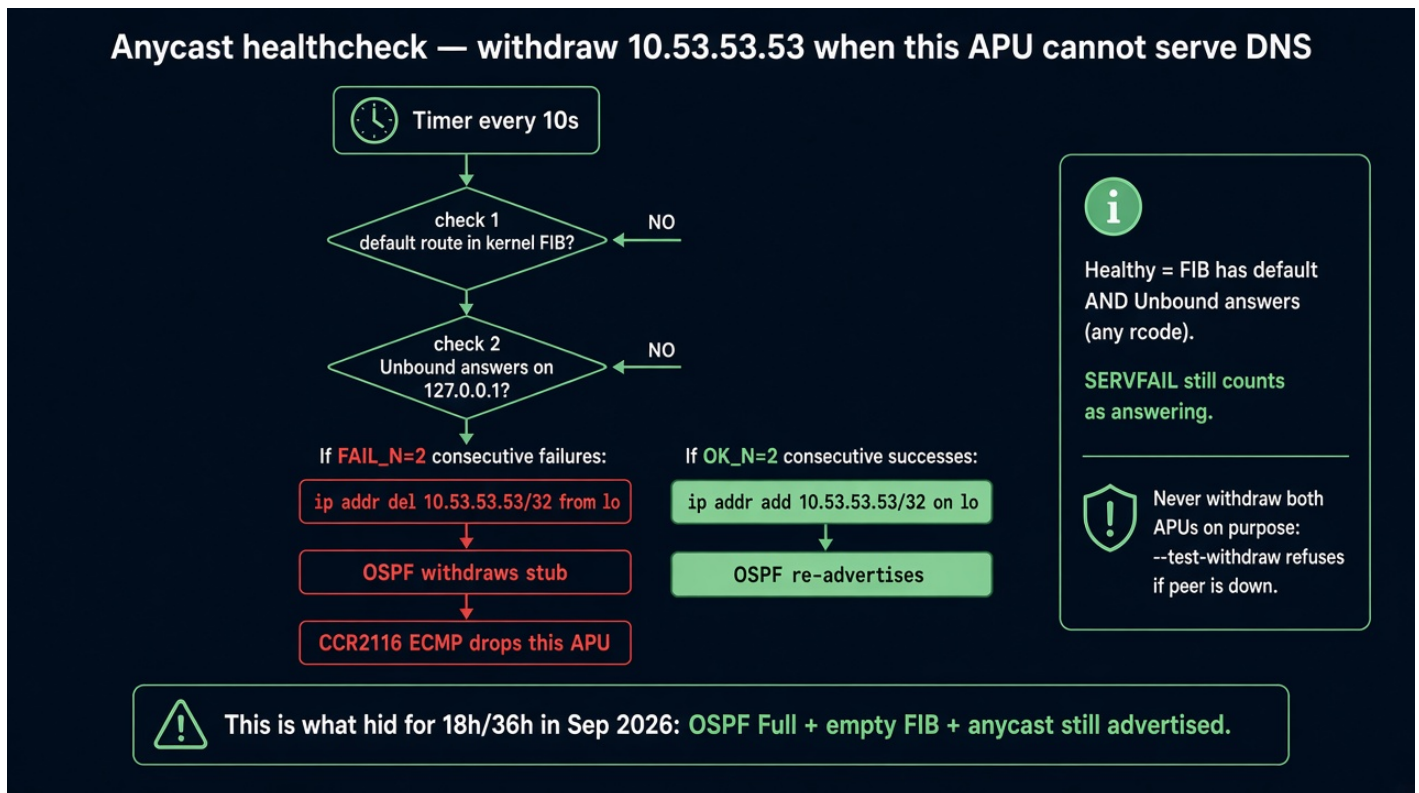
- `mikrotik-workspace/tools/handover/2026-09-15-apu-hardening.sh` (HOST-APU-01)
- `mikrotik-workspace/scratch/anycast-hc/anycast-healthcheck.sh` and `tools/handover/anycast-hc/`
- `mikrotik-workspace/tools/handover/2026-09-13-apu-anycast-healthcheck.sh` (install)
- hp02 `mikrotik-dashboard` (`apu.py`, `dns_collect.py`, `docker_fleet.py`)

Anycast self-heal

Anycast self-heal

FRR on each APU advertises `10.53.53.53/32` as a stub of `lo`. The core therefore load-shares onto **whatever is currently advertised**, not “whatever has OSPF Full”. If an APU cannot actually serve DNS, it must take that address off `lo` itself. That is `anycast-healthcheck.sh`, run every 10 s by `anycast-healthcheck.timer`.

Without this, OSPF Full + an empty kernel FIB still looks like a live resolver. That is exactly the Sep 2026 blackholes.



What healthy means

Healthy is **both**:

1. A **default route exists in the main kernel FIB** (`ip -4 route show default` matches `^default`). An empty FIB is what an OSPF-Full blackhole looks like from the box.

2. **Unbound answers on 127.0.0.1** — `dig +time=1 +tries=1 @127.0.0.1 example.com A` prints a `status:` line. **Any rcode counts**, including SERVFAIL. The resolver is up; upstream may be down. That condition hits both APUs equally. Withdrawing both would turn a degraded cache into no DNS at all.

Hysteresis: **FAIL_N=2** consecutive failures withdraw; **OK_N=2** consecutive successes restore. The timer fires every 10 s (`OnUnitActiveSec=10s`), so a real fault takes ~20 s to leave the ECMP set, and a flap of a single check is held.

flowchart TD

```
T["Timer every 10s"] --> F{"Default route in kernel FIB?"}
F -->|no| FAIL
F -->|yes| U{"Unbound answers on 127.0.0.1? any rcode"}
U -->|no| FAIL
U -->|yes| OK
FAIL["FAIL_N=2 consecutive"] --> DEL["ip addr del 10.53.53.53/32 dev lo"]
DEL --> W["OSPF withdraws the stub"]
W --> ECMP["CCR2116 ECMP drops this APU"]
OK["OK_N=2 consecutive"] --> ADD["ip addr add 10.53.53.53/32 dev lo"]
ADD --> ADV["OSPF re-advertises"]
```

Why this exists — Sep 2026 blackholes

When	Box	Issue	What happened	How long
2026-09-11	apu02	#497	OSPF Full, kernel FIB empty, anycast still advertised	~18 h
2026-09-12	apu01	#523 / #515	Same shape	~36 h

Trigger: a glibc upgrade made needrestart bounce `systemd-networkd`. `networkd` deleted FRR's kernel nexthop objects; the kernel dropped every route that used them. **FRR 8.4.4 never reinstalled them.** Unbound kept answering locally. The adjacency stayed Full. The core kept both next hops in `10.53.53.53/32`. Every client whose hash landed on the sick APU had **no DNS**, while pings, OSPF, and “unbound up” all looked fine.

Two other fixes sit beside the healthcheck (they do not replace it):

- `no zebra nexthop kernel enable` on both APUs (classic nexthops; the #515 exposure was kernel NHG).
- `needrestart` / `networkd` drop-ins so that bounce does not wipe the FIB.

The healthcheck is the last line: if the FIB is empty anyway, **stop advertising the anycast**.

Known interplay: `netplan` owns `lo`. A `networkd` reconfigure can put `10.53.53.53` back on an unhealthy APU; the next check (≤ 10 s) removes it again.

Safety rails

- **Never withdraw both APUs on purpose.** `--test-withdraw` first asks the **peer** (`ANYCAST_PEER` in `/etc/default/anycast-healthcheck`: `apu01` → `192.168.53.2`, `apu02` → `192.168.53.3`) for a `NOERROR`. If the peer is silent, the test refuses.
- `SERVFAIL` / upstream loss is **not** a withdraw. Only “no default in FIB” or “Unbound does not answer at all”.
- `--test-withdraw` writes a **hold file** so the timer cannot re-add the address mid-test. A hold older than 10 minutes is treated as leftover and removed.
- The script never restarts `FRR` or `Unbound`. It only adds/deletes one `/32` on `lo`.

```
anycast-healthcheck.sh           # one check (what the timer runs)
anycast-healthcheck.sh --status  # print state, change nothing
anycast-healthcheck.sh --test-withdraw [secs]  # default 20s; refuses if peer is down
```

Live proof (install `handover --apply --proof`, measured 2026-09-15 12:45Z on `apu02`): core dropped to one next hop for 11 samples, **0 DNS misses**, then both arms returned.

Where it lives

Piece	Path
Script on the APU	<code>/usr/local/sbin/anycast-healthcheck.sh</code>
Timer / unit	<code>anycast-healthcheck.timer</code> / <code>.service</code>
Peer env	<code>/etc/default/anycast-healthcheck</code> (<code>ANYCAST_PEER=...</code>)
State	<code>/run/anycast-healthcheck.state</code>
Workspace copy	<code>mikrotik-workspace/scratch/anycast-hc/anycast-healthcheck.sh</code>
Install handover	<code>tools/handover/2026-09-13-apu-anycast-healthcheck.sh</code>

The hp02 `/apu` page (REQ-DASH32) is the observer: timer state, `--status`, hold file, health-check journal, core ECMP set, and a per-APU verdict (`healthy` / `withdrawn` / `degraded` / `blackhole` / `unreachable`). It does **not** act. Withdrawal is the box's decision; repair is the operator's.

Hardening and dashboard scrape

Hardening and dashboard scrape

HOST-APU-01 (2026-09-15 fabric + APU review) closed the listen surface that the Sep 2026 anycast work did not touch. Grafana, Prometheus, node_exporter and unbound-exporter used to listen on **every address**. apu01 also terminates wg0 (172.16.75.1/24), so those sockets were reachable from every wg-home peer **without crossing the core firewall**.

After the bind change, LAN :9090 and :9100 **connection-refuse**. The hp02 dashboard still needs those numbers, so it scrapes **over SSH** as bodo to vlan89 and talks to 127.0.0.1 on the box. It never re-exposes those ports.

HOST-APU-01 listen map — what is reachable where

	apu01	apu02
Unbound DNS :53	vlan89 vlan53 wg0	vlan89 vlan53
Grafana :3000	vlan89 vlan89	vlan89 vlan89
Prometheus :9090	localhost	localhost
node_export :9100	localhost	localhost
unbound-exporter :9167	localhost	localhost
SSH :22	vlan89	vlan89

hp02 dashboard
SSH tunnels from hp02

- /dns → 127.0.0.1:9090 PromQL
- /docker → 127.0.0.1:9100 metrics
- /apu → 127.0.0.1:9090 + FIB/healthcheck

LEGEND

- Grafana only on vlan89 10.9.8.206/.207:3000
- Prometheus and exporters localhost only
- Unbound on vlan89 + vlan53 (+ wg0 on apu01)
- SSH bodo@vlan89

LAN :9090/:9100 connection-refuses after HOST-APU-01; dashboard scrapes over ssh, never re-exposes those ports.

Listen map (after HOST-APU-01)

Service	apu01	apu02
Unbound DNS :53	vlan89 + vlan53 + wg0 + anycast on lo — not 0.0.0.0	vlan89 + vlan53 + anycast on lo — not 0.0.0.0
Grafana :3000	vlan89 only (10.9.8.206)	vlan89 only (10.9.8.207)
Prometheus :9090	localhost	localhost
node_exporter :9100	localhost	localhost
unbound-exporter :9167	localhost	localhost
SSH :22	vlan89 (bodo, pubkey)	vlan89 (bodo, pubkey)

Unbound ACL is fabric prefixes (10.9.8.0/24, 10.53.53.0/24, 192.168.32.0/19, 192.168.53.0/24, 172.16.62.0/24, 172.16.75.0/24), not all of RFC1918. Reload keeps an old 0.0.0.0:53 socket; a **restart** is what actually drops it.

On **apu01 only**, nftables table inet host_apu_01 is extra belt-and-braces: allow Grafana/metrics on lo and enp2s0 (vlan89), **drop** tcp/3000,9090,9100,9167 on every other ingress (wg0 and vlan53). Persist via include "/etc/nftables.d/host-apu-01.nft" in /etc/nftables.conf and nftables.service enabled. nft -f alone dies at reboot; enable without the include would load Debian's flush-ruleset stub and wipe the live table.

Other HOST-APU-01 items: drop bodo from group lxd (membership is de-facto root); snap remove lxd if unused; apu02 systemctl drop-in (accept_redirects=0, send_redirects=0, rp_filter=2). Never run lxc / lxc list on these boxes — on Ubuntu 24.04 /usr/sbin/lxc is the lxd-installer stub and **installs the snap**.

Handover: mikrotik-workspace/tools/handover/2026-09-15-apu-hardening.sh (dry-run from hp02; --apply on the APU as root, with --dns / --nftables as needed).

How hp02 still sees the boxes

```
flowchart LR
    subgraph hp02 ["hp02 dashboard :8787"]
        DNS["/dns REQ-D5"]
        DOCKER["/docker REQ-DF5"]
        APU["/apu REQ-DASH32"]
        SITES["/sites"]
    end
```

```
DNS -->|"ssh bodo@10.9.8.206/.207"| P["127.0.0.1:9090 PromQL"]
DOCKER -->|"ssh bodo@vlan89"| N["127.0.0.1:9100 /metrics"]
APU -->|"one ssh round trip"| BOX["FIB, lo anycast, timer, Prom on localhost"]
SITES --> INV[inventory / live probes]
```

Requirement	Page	What it scrapes	What it must not do
REQ-D5	/dns	One SSH per APU to vlan89; remote helper queries <code>http://127.0.0.1:9090</code> for the existing PromQL (Unbound + node). Grafana links stay on vlan89.	Must not scrape a LAN <code>:9090</code> URL. That path refuses, and <code>/dns</code> used to paint PROM DOWN while Unbound was answering 9/9.
REQ-DF5	/docker	One SSH, <code>curl http://127.0.0.1:9100/metrics</code> on the box. APUs are <code>kind: node_exporter</code> (system pulse only — no Docker). A leftover LAN <code>node_url</code> is ignored when <code>ssh</code> is set.	Must not scrape <code>vlan89/vlan53 :9100</code> . That painted apu01/apu02 red (<code>Connection refused</code>) while the boxes were up.
REQ-DASH32	/apu	Same SSH path: default route, OSPF route count, nexthop mode, <code>10.53.53.53/32</code> on lo, healthcheck <code>--status / timer / journal</code> , a smaller Unbound Prom set. Plus core REST (ECMP + OSPF/BFD) and UDP probes.	Does not act; does not read root-only <code>frr.conf</code> / nft.

SSH details that matter for the poller: user `bodo`, **no sudo**, `ssh -n` / stdin closed so apu01 Defaults `use_pty` cannot hang the dashboard. Fail-soft: SSH failure is `prometheus unreachable` / `status: error`, never an exception — probes still run.

Dashboard pages to open

Path	Role
/apu	Anycast arms, self-heal, blackhole vs withdrawn vs healthy
/dns	Unbound QPS, cache, rcodes — Prom over SSH
/docker	APU load/mem from node_exporter over SSH (and Docker hosts elsewhere)
/sites	Site / inventory view of the same two boxes

Grafana remains the on-box UI: `http://10.9.8.206:3000/` and `http://10.9.8.207:3000/`. Prometheus and the exporters have **no** LAN URL after HOST-APU-01; do not add one back so the dashboard can

“just HTTP”.