# TestPulse Pro Beta — Virtual Lab Administration and Troubleshooting

**For:** the network/AAA engineer evaluating the Pro Beta Virtual-Lab Appliance.

**The short version:** you should not need to administer Kubernetes. The appliance ships an AI administrator that owns the cluster, and this guide tells you what to ask it, how to read what it tells you, and — most importantly — **how to tell a broken lab apart from a real diagnostic finding.**

---

## 1. What you are evaluating, and what is holding it up

TestPulse diagnoses AAA failures from real wire evidence. To show you that, this appliance carries a complete 802.1X/RADIUS/TACACS lab so you can see the full evidence pipeline without wiring up your own switch and servers.

That lab is a single-node Kubernetes cluster. **You are not expected to operate it.** Every cluster action is performed by the AI administrator that ships with the appliance, and you talk to it in plain language:

> *"the lab looks broken"* · *"no packets were captured"* · *"RADIUS keeps rejecting"* · *"is the lab healthy?"*

If you ever find yourself typing `kubectl` to make the product work, that is a defect in this appliance — please report it.

---

## 2. One lab, two planes

Knowing which plane is which will save you most of the confusion available here.

### Container plane — 13 workloads

`tac_plus` · `freeradius` · `openldap` · `dns` · `isc-dhcp` · `kea-ddi` · `hostapd-nas` · `hostapd-mab` · `mock-nac` · `supplicant` · `alpine-supplicant` · `ostinato` · `passthru-syslog`

**The entire AAA path lives here**, and the traffic is genuinely real — a real `wpa_supplicant` on the supplicant pod produces real EAPOL frames on the lab bridge, and real DHCP DORA follows (roughly 28 EAPOL + 4 DHCP frames per method). Nothing is simulated.

### KubeVirt VM plane — 3 virtual machines

`mikrotik-chr` (RouterOS switch) · `endpoint-alpine-1/2` (Linux supplicants) · `endpoint-win10` (**not shipped in v1**)

This plane needs hardware virtualization passed through to the appliance. It adds the switch evidence path. The NAS is a container and is always there.

> **The one thing worth memorising:** if the VM plane will not start, **802.1X still works.** The AAA path is entirely in the container plane. You have lost the switch and VPN evidence paths, not the product.

---

## 3. The most important skill: lab fault vs product finding

TestPulse is a diagnostic tool, so a broken lab and a real finding can look alike. Getting this wrong is the fastest route to a wrong conclusion about the product.

| What you see | Almost certainly a lab fault when… | A real finding when… |
|---|---|---|
| RADIUS rejects `testuser` | **EAP-TLS still passes** while PEAP/TTLS/PAP fail — certificate auth skips the users file, so this is the users database, not RADIUS | all methods fail together, with a cause family and evidence behind it |
| Directory lookups fail | the LDAP pod is `Running` but the directory is **empty** — it restarted and lost its data | the directory answers, and answers wrongly or slowly |
| "Transport fault" on a healthy run | **no pcap was captured** — the sub-agent is reporting its own blindness | packets were captured and show loss, jitter or retransmits |
| Everything fails at once | the node ran out of disk and evicted the lab — AAA faults are selective, eviction is total | (it isn't) |
| A test suite "passed" with nothing run | **"no tests collected" is a failure**, not a pass | a suite ran and reported results |

**Ask the administrator directly: *"is this a lab problem or a real finding?"*** It is instructed to answer that question plainly, and never to present a lab fault as a product result.

---

## 4. The three health checks

Ask for these by name. Each answers a different question — a green answer from one says nothing about the others.

| Ask for | It answers |
|---|---|
| **lab check** (`/testpulse-lab-check`) | *Is the lab ready at all?* Disk, core AAA pods, the CoA listener, supplicant tooling, the lab CA. |
| **vlab check** (`/k8s-vlab-check`) | *Are components up and reachable?* API, viewer, every pod, RADIUS on :1812/:3799, the NAS on the 802.1X network. |

| Ask for | It answers |
|---|---|
| **pcap check** (`/k8s-vlab-pcap-check`) | *Can every device be captured?* Proves wire evidence will land. |

**Why the third one matters more than it sounds.** A test that runs with no capture does not fail — it produces a *confident diagnosis from absent evidence*. That is worse than an error, because it looks like a result. Run the pcap check before you trust a run.

---

## 5. The failure you are most likely to hit: hardware virtualization

**Symptom:** the VM plane never starts. The administrator will report VM pods stuck pending with `Insufficient devices.kubevirt.io/kvm`.

**Cause:** your hypervisor is not exposing hardware virtualization to this appliance. It cannot be fixed from inside — it is a setting on *your* hypervisor, with the appliance powered off:

| Hypervisor | Setting |
|---|---|
| VMware Workstation / ESXi | **Expose hardware assisted virtualization to the guest OS** (VM settings → Processors) |
| VirtualBox | `VBoxManage modifyvm "<vm-name>" --nested-hw-virt on` |
| Hyper-V | `Set-VMProcessor -VMName "<vm-name>" -ExposeVirtualizationExtensions $true` |

Also confirm virtualization is enabled in the host's firmware (BIOS/UEFI) — it is off by default on many machines.

**Meanwhile, keep testing.** The container plane carries the whole AAA path, so 802.1X, RADIUS, TACACS, DHCP and the directory all work without this. You lose the RouterOS switch evidence path only. The NAS is a container and is always there.

---

## 6. Reading a run you can trust

Before treating a run as evidence of anything:

1. **Was the wire captured?** Ask for the pcap check. No capture, no wire claims.
2. **Did the suite actually run?** "No tests collected" is a red result.
3. **Which planes were up?** A run with the VM plane down cannot speak to switch or VPN behaviour — the administrator will tell you which planes were live.
4. **Is the diagnosis grounded?** TestPulse should give you an observed outcome, a named cause family, the specific evidence, and a fix direction. A verdict with no evidence behind it is a bug worth reporting.

# 7. Lab credentials

**These are published on purpose.** The lab is self-contained inside this appliance — RADIUS, TACACS+, the directory and the supplicants all talk to each other over virtual bridges that never reach your network. The shared secrets are between components that both live in this VM, so knowing them gains an attacker nothing they would not already have by being on the appliance itself.

They are printed here because **you need them**: to point your own supplicant at the lab, to run `radtest` by hand, to add a user, or to read a capture and understand what you are looking at.

## AAA

| What | Value |
|---|---|
| RADIUS shared secret | `testing123` |
| RADIUS CoA secret | `testing123` |
| TACACS+ key | `tac_lab_key` |

## Identities

| What | Value |
|---|---|
| 802.1X / PAP test user | `testuser` |
| Second test identity | `testuser1` |
| Endpoint / appliance login | `testpulse` |
| Directory admin bind | `adminpass` |

## The one credential that is NOT published

**Your RDP password is generated on this appliance at first boot and shown once**, on the console, at the end of the first boot. It is never written to a log or the journal. That one is different because RDP is the single port this appliance exposes to *your* network — the risk is yours, so the credential has to be yours too, not one shared by every copy of this image.

Change it whenever you like with `passwd`.

## If you rotate the AAA secrets

You can, but change them **everywhere at once**. A shared secret is shared: the RADIUS server, the NAS, the supplicant, the switch config and the testbed YAML all carry it, and if they disagree every authentication fails — as `auth_semantic_failure`, which looks exactly like a real finding. A half-rotated lab does not merely break; it lies about what it found.

Ask the administrator to do it rather than editing by hand — it knows every site.

---

# 7. How deployments work here — and what ArgoCD is

**This appliance does not run ArgoCD.** Its lab manifests are applied directly, and nothing you do requires it. This section is here because testers ask what ArgoCD is for in a Kubernetes AAA lab, and because the lesson it taught is worth carrying into your own environment.

**What ArgoCD does:** it is a GitOps continuous-delivery controller. You declare the desired state of a Kubernetes application in a git repository, and ArgoCD continuously reconciles the live cluster against it. Nobody deploys by hand; you commit, and the cluster converges.

**Why the internal TestPulse lab uses it:** the AAA lab is a fleet of interdependent workloads — RADIUS, TACACS, LDAP, DNS, DHCP, the NAS, the supplicants — and hand-applied changes drift. Git as the source of truth makes the lab reproducible, and makes "what changed before this broke?" answerable.

**The lesson worth taking away, whether or not you use ArgoCD:**

> An out-of-band image change does not stick. Under a GitOps controller, pushing a new image with `kubectl set image` reports a **successful rollout** and is then quietly **reverted to the tag committed in git**. The pod runs the old code while every symptom points at the new code being broken — routes appear missing, files appear absent, and your local build was clean the whole time.

So the deployment discipline is:

1. Build the image and make it available to the cluster's container runtime.
2. **Bump the pin in git and commit it** — this is the step that makes the change real.
3. Apply, then **verify the running pod's image matches the pin**, and that the change is actually present in the running pod.

Never infer a successful deployment from a successful rollout message. A plain restart re-pulls the *same pinned tag*, so it does not pick up new code at all.

Two configuration choices from the internal lab, both learned the hard way: **automatic self-healing is turned off**, because an aggressive reconcile fights an operator who is mid-debug; and **sync and health changes are pushed to a notification channel**, so a silent revert cannot go unnoticed.

---

# 8. Sizing

**These are minimums, not a recommended tier.** The lab was specified at this size and does not run correctly below it.

| Resource | Minimum |
|---|---|
| vCPU | **6** |
| RAM | **32 GB** |
| Disk | **200 GB SSD** |

Running both planes is what drives these numbers. Below them the KubeVirt VMs and the container plane contend, and **the symptom is not a clean failure — it is inflated latency that reads as a transport fault in a diagnosis.** So an undersized host does not crash in a way you would notice; it produces a confident, wrong answer. **Suspect the host before the lab when a latency budget fails.**

Keep an eye on disk. If the node runs out, the kubelet evicts the entire lab at once, and recovery needs a deliberate sequence — ask the administrator rather than restarting things yourself.

---

## 9. When to report a bug

Please report:

- Anything that forced you to run `kubectl` yourself.
- A diagnosis presented confidently with no evidence behind it.
- A lab fault presented to you as a product finding.
- A suite reporting success while having run nothing.

These matter more than cosmetic issues — each one is a case where the appliance told you something untrue about its own state.

---