Netsocs

OPERATIONAL INTELLIGENCE LAYER

TECHNICAL BROCHURE · EDITION 2026

Netsocs Failover Architecture

Application-Layer High Availability with Keepalived, NFS, Docker Stack & Virtual IP.

Confidential · Netsocs LLCTechnical Brochure · · Edition 2026 Document v1.1

A resilient active–passive architecture designed to keep the Netsocs application layer available through automated VIP failover, shared storage, and controlled Docker stack orchestration.

Active–Passive | Keepalived · VRRP | Docker Compose | NFS Shared Storage | Virtual IP

Application failover ≠ database backup

01 · EXECUTIVE SUMMARY

Keep the application layer available, even when a node fails.

Netsocs Failover Architecture is an active–passive design that keeps the Netsocs application layer reachable through a single Virtual IP, while Keepalived continuously monitors node health and shifts traffic to a healthy peer when needed. Shared application files are kept consistent between nodes via NFS, and the Docker stack is started, restarted, or stopped automatically as a node enters or leaves the MASTER state.

The architecture is designed to **reduce service interruption during application-layer node failure** and to provide a controlled, repeatable operating model for production environments. Failover behavior depends on network, storage, and service health conditions.

~3–6 s
VIP migration on node loss

2 ×
Active / passive nodes

1 ×
Virtual IP (single entrypoint)

8 ×
NFS-shared data volumes

Important Scope Note — Database Backup Is Not Included
This failover architecture does **not** back up, replicate, or protect the databases. MySQL, MongoDB, Redis, or any other database services are assumed to be hosted and managed externally. A separate database backup, replication, and disaster recovery strategy is required.

What it solves

Single-server outages no longer take the Netsocs application offline. Operators get a predictable, observable failover path that protects the client-facing endpoint without invasive changes to the stack.

Business value at a glance

Operational continuity

Service answering on the VIP keeps responding even if one node is lost.

Predictable behavior

VIP, Keepalived, and Docker lifecycle hooks codify the failover sequence.

Stack-aware

Docker Compose is started, restarted, or stopped automatically per node state.

Operator-friendly

Logs, status, and manual failover commands designed for runbook execution.

— 02 · ARCHITECTURE OVERVIEW

A single VIP, two nodes, and a clearly bounded scope.

Clients reach a single Virtual IP. Behind that VIP, two nodes run Keepalived in an active–passive configuration. Only the MASTER node runs the Netsocs Docker stack. Shared application files live on NFS so both nodes operate on identical recordings, certificates, and static assets. Databases are external to this architecture and follow their own protection strategy.

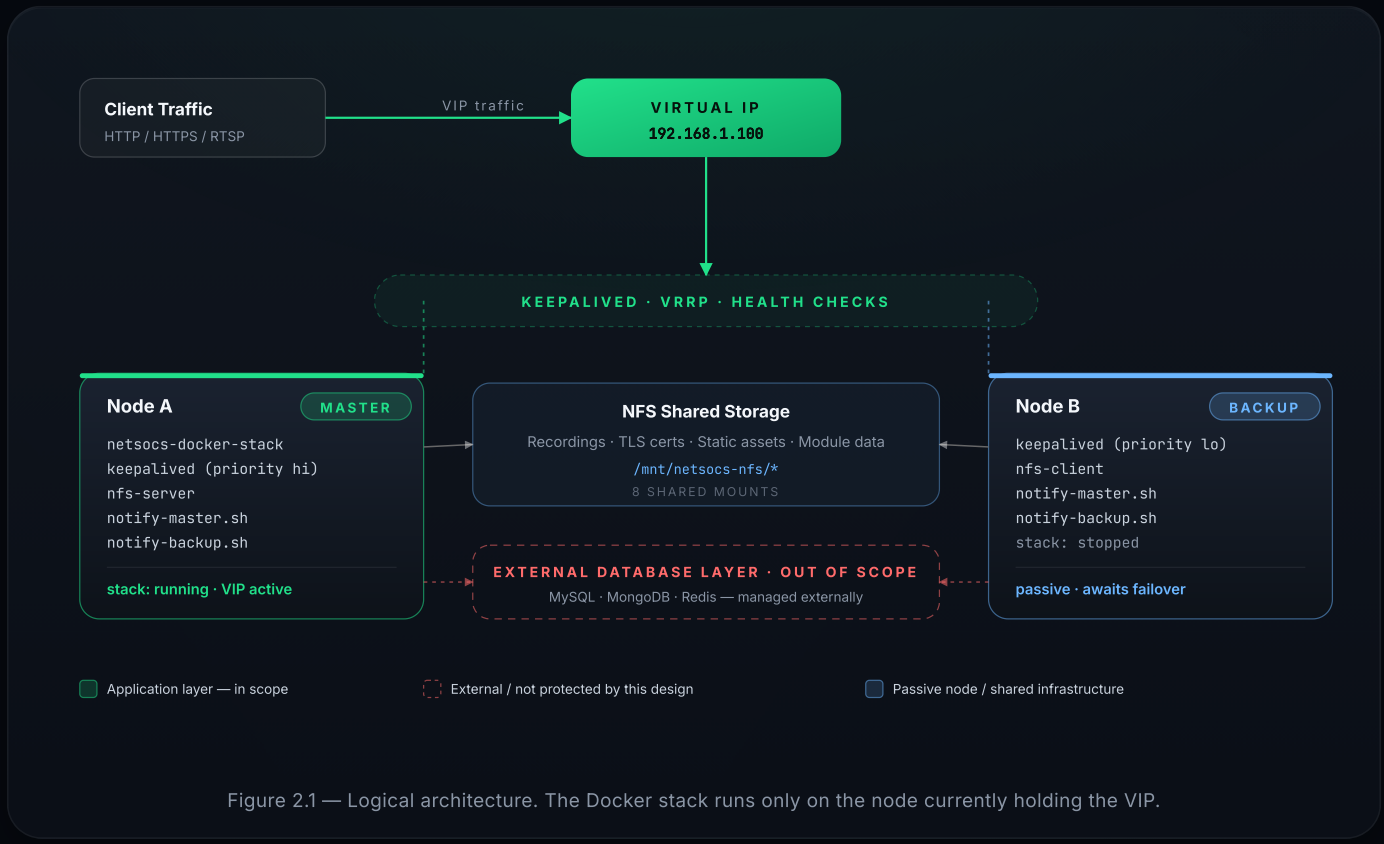

Figure 2.1 — Logical architecture. The Docker stack runs only on the node currently holding the VIP.

Database Backup Is Not Included · Application failover ≠ database backup

This architecture protects the **application layer only**. MySQL, MongoDB, Redis, and any other database services are assumed to be managed externally with their own backup, replication, and disaster recovery strategy.

03 · ACTIVE–PASSIVE LIFECYCLE

One node serves traffic.
The other is ready to take over.

Only the MASTER node runs the Netsocs Docker stack. The BACKUP node stays passive until Keepalived determines a failover is required. Lifecycle hooks start, restart, or stop the stack as nodes change state.

On becoming MASTER → `notify-master.sh`

If fewer than 3 containers are running, the stack is started with `docker compose up -d --no-recreate`. If the stack is already running, it is restarted to cleanly assume the VIP.

On becoming BACKUP → `notify-backup.sh`

The script detects the running stack and executes `docker compose down`. The node remains inactive until it is promoted to MASTER again.

Lifecycle states

Three reference states walk through a complete failover and recovery cycle.

01 · Initial State

Steady operation

Node A	MASTER · VIP
Stack on A	Running
Node B	BACKUP
Stack on B	Stopped

02 · Failover

Node A fails

Node A	Down
Node B	MASTER · VIP
Hook fires	notify-master
Stack on B	Starting → Running

03 · Recovery

Node A returns

Node A	MASTER · preempt
Stack on A	Running
Node B	BACKUP
Hook fires	notify-backup

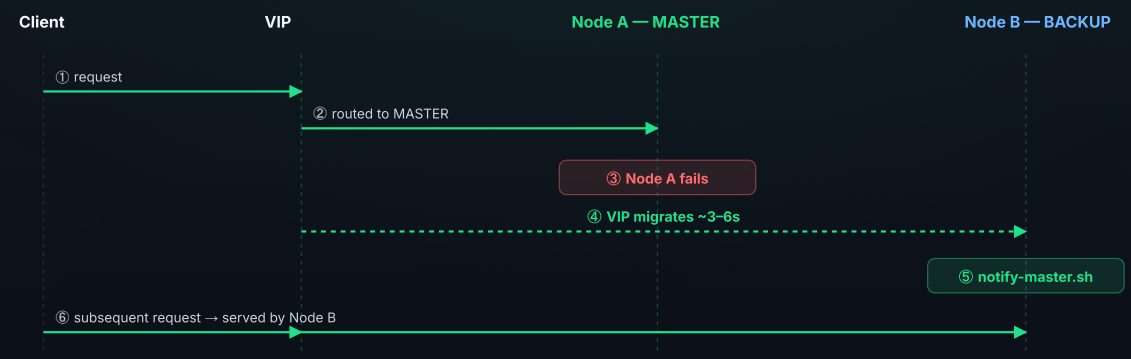

Figure 3.1 — Failover sequence. The VIP migrates to Node B; the stack is started by `notify-master.sh`.

— 04 · TECHNICAL COMPONENTS

Small, well-defined parts. Predictable behavior.

Each component has one job. Together they implement a clean, observable failover loop. No component requires custom build steps; everything ships as configuration files and shell scripts maintained in the Netsocs repository.

Keepalived

Implements VRRP between the two nodes, runs health checks, and triggers notify hooks on state transitions.

Role: VIP ownership, health, lifecycle hooks.
Best practice: use unicast in cloud environments.

Virtual IP (VIP)

The single, floating address clients use to reach Netsocs. Reserved and never assigned to a real host.

Role: stable client entrypoint.
Best practice: reserve in DHCP / IPAM to avoid collisions.

Docker Compose

Defines the Netsocs application stack. Started, restarted, or stopped automatically by Keepalived hooks.

Role: stack orchestration.
Best practice: keep Compose versions identical across nodes.

NFS Shared Storage

Serves recordings, TLS certs, and module assets to both nodes via 8 mounts under `/mnt/netsocs-nfs`.

Role: shared application data.
Best practice: validate mounts before any failover test.

Health Check Script

Validates Docker daemon, critical containers (`netsocs-caddy`, `traefik`), and port 80.

Role: service-aware liveness.
Best practice: tune thresholds to local latency.

Notify Master Script

Runs when the node takes the VIP. Starts the stack if down; restarts it if already running.

Role: promotion hook.
Best practice: log to `/var/log/keepalived-failover.log`.

Notify Backup Script

Runs when the node loses the VIP. Detects the running stack and executes `docker compose down`.

Role: demotion hook.
Best practice: verify stop completes before declaring backup state.

External Database Layer

MySQL, MongoDB, Redis hosted on dedicated infrastructure. Both nodes connect to the same external endpoints.

Role: shared state — **not** failed over here.
Best practice: separate DR plan.

05 · IMPLEMENTATION PHASES

Eight phases, sequenced for repeatable deployment.

Each phase has a clear owner, exit criteria, and validation step. Follow them in order. Skipping NFS or VIP verification before testing failover is the most common source of operational surprises.

PHASE 01

Prepare Nodes

Ubuntu / Debian, Docker Engine ≥ 24.x, Compose plugin, static IPs, mutual ping, ports 112 (VRRP) and 2049 (NFS) open.

PHASE 02

NFS Server on Node A

Run `setup-nfs-server.sh`. Confirm shares via `showmount -e localhost`.

PHASE 03

NFS Client on Node B

Run `setup-nfs-client.sh`. Verify 8 mounts under `/mnt/netsocs-nfs`.

PHASE 04

NFS Compose Override

Symlink `compose.override.nfs.yml` so `docker compose up` uses NFS volumes automatically.

PHASE 05

Keepalived

Install on both nodes. Apply master/backup configs, substitute placeholders, install notify scripts.

PHASE 06

Validate VIP

Confirm the VIP appears on Node A. `systemctl status keepalived`, watch the VRRP journal.

PHASE 07

Test Manual Failover

Stop Keepalived on Node A. Verify VIP migrates to Node B and stack starts. Recover Node A and confirm reclaim.

PHASE 08

Document & Operationalize

Capture runbook, logs, contacts. Train operators. Schedule periodic failover drills under maintenance windows.

Reference variables

Captured once per environment. Never committed to the repository.

VARIABLE	DESCRIPTION	EXAMPLE
`NODE_A_IP`	Real IP of Node A	`192.168.1.10`
`NODE_B_IP`	Real IP of Node B	`192.168.1.11`
`VIP_ADDRESS`	Floating Virtual IP (must be unused)	`192.168.1.100`
`NETWORK_IFACE`	Network interface on both nodes	`eth0` / `ens3`
`AUTH_PASS`	VRRP authentication password (≤ 8 chars)	`Ns3c2024`
`NETSOCS_DIR`	Project directory on both nodes	`/opt/netsocs`

06 · BEST PRACTICES

Operational discipline that prevents quiet failures.

These practices reflect what experienced operators check before, during, and after going live. They are inexpensive and protect against the failure modes most likely to surface under real load.

01 **Use static IPs** for both Node A and Node B. DHCP leases that change mid-incident are the most common cause of silent failover regressions.

02 **Reserve the VIP.** Confirm it is not used by any other host and is excluded from DHCP scopes and IPAM.

03 **Keep Docker Compose versions identical** across nodes. Skew can cause the stack to start on Node A and fail on Node B.

04 **Keep environment variables out of the repository.** Source them from `failover-env.sh`; do not commit secrets.

05 **Test failover during a maintenance window** at least once before production exposure, and on a recurring schedule afterwards.

06 **Validate NFS mounts** before starting any production traffic. A missing mount silently breaks recordings, certificates, or static assets.

07 **Monitor Keepalived logs and Docker health.** Forward `/var/log/keepalived-*.log` and `docker logs` into your central observability platform.

08 **Confirm MediaMTX binds to all interfaces** when using `network_mode: host` (use `:8554` rather than a specific IP).

09 **Maintain a separate database backup and replication strategy.** This failover does not address database protection.

10 **Document rollback and recovery steps** alongside the runbook. Every operator should know exactly how to revert.

11 **Use cloud-specific failover mechanisms** when VRRP / VIP routing is not natively supported by the provider.

12 **Apply least privilege** to NFS, firewall, and SSH between the two nodes. Restrict to required ports only.

Validation hygiene

Treat NFS, VIP, and Keepalived validation as preconditions to production traffic — not as post-incident debugging tools. Every box on the checklist maps directly to a failure mode observed in the field.

07 · VERIFICATION & TESTING

A checklist that confirms each layer is doing its job.

Run this checklist in order. Each item maps to a specific layer of the failover stack. If any step fails, stop and resolve before proceeding.

- [x] **Ping the VIP** from an external client: `ping 192.168.1.100` must respond.
- [x] **Curl the VIP**: `curl http://$VIP_ADDRESS` must return the Netsocs page.
- [x] **Verify VIP assignment** on Node A: `ip addr show $NETWORK_IFACE | grep $VIP_ADDRESS`.
- [x] **Check Docker containers** on Node A: `docker compose ps`.
- [x] **Check Keepalived status** on both nodes: `systemctl status keepalived`.
- [x] **Tail failover logs**: `tail -f /var/log/keepalived-failover.log`.
- [x] **Simulate Node A failure**: stop Keepalived on A (`systemctl stop keepalived`).
- [x] **Confirm Node B takes over**: VIP appears on Node B, stack starts via `notify-master.sh`.
- [x] **Recover Node A**: restart Keepalived on A. VIP should preempt back in ~10 seconds.
- [x] **Confirm Node B releases the VIP** and stops the stack: `docker compose ps` returns no running containers.

What "success" means here

Success is reproducible failover and recovery, observed end-to-end. A single run does not prove production readiness — run the checklist on multiple occasions and across at least one full operator rotation.

— 08 · CLOUD DEPLOYMENT NOTES

Cloud networks change the rules around the VIP.

On-prem VRRP multicast does not map cleanly to AWS, GCP, or Azure. Each provider imposes constraints that affect how the VIP is realized. Treat this section as a starting point and validate against the provider's current networking model.

VRRP multicast

Usually unavailable in cloud VPCs. Use Keepalived in **unicast mode**, declaring each peer's real IP under `unicast_peer`.

AWS

Layer-2 VIPs are not routable. Reassign an **Elastic IP** via the AWS CLI from the notify scripts — see the AWS-specific runbook.

GCP & Azure

Similar constraints. Prefer a **provider-managed Load Balancer** or routing rule that targets the node currently holding the role.

Environment-specific consideration

The choice between unicast Keepalived, Elastic IP reassignment, and managed load balancers is a deployment decision. Validate latency, IAM permissions, and quota limits before relying on any of them in production.

KEEPALIVED · UNICAST (CLOUD)

```
unicast_src_ip NODE_A_IP    # real IP of this node
unicast_peer {
    NODE_B_IP               # real IP of the other node
}
```

FIREWALL · VRRP & NFS

```
# On AMBOS nodos — allow VRRP between peers
ufw allow from $NODE_B_IP proto vrrp    # On Node A
ufw allow from $NODE_A_IP proto vrrp    # On Node B

# NFS — restrict to backup node only
ufw allow from $NODE_B_IP to any port 2049
ufw allow from $NODE_B_IP to any port 111
```

09 · OPERATIONAL RUNBOOK · SNAPSHOT

The commands operators actually run on the box.

This snapshot is meant for fast reference — not as a complete runbook. Operators should keep the full guide in the repository and rehearse the failover procedure during scheduled drills.

Verify VIP — ANY NODE

```
ip addr show $NETWORK_IFACE | grep $VIP_ADDRESS
ping $VIP_ADDRESS
curl http://$VIP_ADDRESS
```

Check Keepalived — BOTH NODES

```
systemctl status keepalived
journalctl -u keepalived -f
```

Check Logs — BOTH NODES

```
tail -f /var/log/keepalived-health.log
tail -f /var/log/keepalived-failover.log
```

Manual Failover Test — NODE A

```
# Option 1 — stop Keepalived
systemctl stop keepalived
# Option 2 — bring the interface down
ip link set $NETWORK_IFACE down
# Option 3 — stop the Docker stack
cd $NETSOCS_DIR && docker compose down
```

Confirm Node B Took Over — NODE B

```
ip addr show | grep $VIP_ADDRESS
docker compose ps
tail -20 /var/log/keepalived-failover.log
```

Recover Node A — NODE A

```
systemctl start keepalived
# VIP preempts back in ~10 seconds
ip addr show $NETWORK_IFACE | grep $VIP_ADDRESS
docker compose ps
```

Failover that respects least privilege and auditability.

The failover layer touches privileged interfaces — VRRP, NFS, Docker, system services. Apply the same controls you would apply to any production component: secrets management, restricted access, and traceable logs.

Protect the VRRP auth password

Keep `AUTH_PASS` outside the repository. Rotate it on operator turnover. Treat it like any other shared secret.

Never commit `failover-env.sh`

Environment files contain VRRP password, node IPs, and operational topology. Keep them in your secrets manager.

Limit NFS access

Export shares only to the backup node's IP. Avoid wildcard exports. Verify `/etc/exports` on every change.

Restrict firewall rules

Open only what node-to-node communication requires: VRRP, NFS, portmapper. Block lateral movement at the host firewall.

Keep logs for audit

Forward Keepalived, Docker, and NFS logs to your central platform. Retain history sufficient for incident review.

Least privilege & access

Only authorized operators have SSH to the nodes. Use bastion / jump hosts where required. Disable password auth in favor of keys.

— 11 · LIMITATIONS & SCOPE

Transparent about what this architecture does not do.

The Netsocs Failover Architecture is intentionally scoped to the application layer. Honest scope boundaries make production deployments safer and avoid misaligned expectations.

✓ COVERED · APPLICATION LAYER

- Docker Compose stack lifecycle
- Virtual IP (VIP) management via Keepalived
- Active–passive node orchestration
- NFS-shared application files (recordings, TLS, static assets)
- Health checks for Docker daemon & critical containers
- Manual and automated failover workflows

✕ NOT COVERED · OUT OF SCOPE

- Database backup (MySQL / MongoDB / Redis / others)
- Database replication and failover
- Object storage or large media retention strategy
- Provider-specific VIP behavior (AWS / GCP / Azure routing)
- Complete disaster recovery program
- Application-layer schema or data migration

Application failover ≠ database backup

This document describes **application-layer failover only**. Databases are not backed up, replicated, or recovered by this architecture. A separate strategy covering database backup, replication, and disaster recovery is required and must be designed, implemented, and validated independently.

END OF DOCUMENT

Netsocs Failover Architecture

Application-Layer High Availability with Keepalived, NFS, Docker Stack & Virtual IP.

DOCUMENT

Technical Brochure

EDITION

2026

VERSION

v1.1