mac-to-mac file system: NFS is faster than native SMB for dev env

For tiny-file dev trees on macOS, tuned NFSv3 beats native SMB — if you pick the right client profile for a wired desktop vs a laptop that roams.

Yiğit Konur · Apr 13, 2026 · 13 min read

▶ The useful bits

i bought a Mac Mini like a lot of people did for “openclaw” (lol).

last year i also over-sold the value of local LLMs for a few days and spent a few grand for no reason. but it worked out: with subagents + heavier MCP clients, the Mac Mini became my always-on local build machine, and i mostly “vibe code” from my MacBook into it.

the problem: macOS file sharing via SMB is ok for big files, but it’s painfully slow for dev trees full of tiny files (TypeScript, config, node_modules). i switched the “live filesystem” part to NFS and tuned it until it was both fast on the LAN *and* survived my MacBook leaving the LAN — which turns out to be the harder half of the problem, and the part most NFS-on-macOS blogs get wrong.

why NFS over SMB (for this workload)

SMB on macOS can be rough on small-file workloads because:

- Finder and friends do extra metadata work (including extended attributes)
- [Skip to content] er-operation overhead (and macOS’s implementation doesn’t always feel optimized for “60k tiny files”)
- directory listings devolve into a lot of little round trips

NFS is simpler. on my LAN, tuned NFS took “listing big dirs” from “why is this taking forever” to “ok, usable”.

the setup i tested

- server: Mac Mini (apple silicon), wired LAN, `192.168.1.200`
- client: MacBook (macOS), same network, roams on and off the LAN
- workload: ~60,000 files / ~8GB, mostly TypeScript + config + node_modules
- rtt: ~4ms (Wi-Fi → switch → ethernet)

pick your client profile before you tune anything

two wildly different recipes hide under the same “NFS client” label:

- **wired workstation** — always on, wired ethernet, never moves, never closes its lid. can be aggressive because nothing ever disappears.
- **mobile laptop** — sleeps, changes networks, walks into rooms without Wi-Fi. every client-side knob has to assume the server *will* disappear.

most blogs recommend workstation settings and call it a day. on a laptop that’s how you get Raycast to crash, Finder to beachball, and `launchd` hammering your radio every minute forever. server tuning is the same for both; client tuning is where they split.

server (Mac Mini): exports + nfsd tuning

/etc/exports

```
/Users/yigitkonur -alldirs -mapall=501:20 -network 192.168.1.0 -mask 255.255.255.0
```

Skip to content

wh[illegible] matters.

- `-alldirs`: mount subdirectories, not only the export root
- `-mapall=501:20`: maps all client access to a single local uid/gid (convenient for single-user dev)
- `-network ... -mask ...`: restricts access to your local subnet

note: `501:20` is common on macOS (first user + staff), not guaranteed. swap for your own uid/gid via `id -u` / `id -g`.

/etc/nfs.conf (server)

```
nfs.server.mount.require_resv_port = 0
nfs.server.require_resv_port = 0
nfs.server.nfsd_threads = 16
nfs.server.async = 1
nfs.server.fsevents = 0
nfs.server.wg_delay = 0
nfs.server.wg_delay_v3 = 0
nfs.server.reqcache_size = 512
nfs.server.request_queue_length = 512
nfs.server.export_hash_size = 256
nfs.server.tcp = 1
nfs.server.udp = 0
nfs.server.user_stats = 0
nfs.server.bonjour = 0
nfs.server.verbose = 0
```

quick “why” table:

parameter	default	value	why
require_resv_port	1	0	macOS clients often mount from non-privileged ports; avoids silent mount pain
nfsd_threads	8	16	more concurrency for lots of small file ops
async	0	1	faster writes; fine for dev where git is the source of truth
fsevents	1	0	reduces server-side overhead
wg	000 / 0	0 / 0	lower latency for small-file writes
req	4	512	better duplicate-request handling under retransmits
request_queue_length	128	512	avoids queue bottlenecks under bursts
export_hash_size	64	256	faster export lookup under load

Skip to content

parameter	default	value	why
bonjour	1	0	i connect by IP anyway

enable the daemon:

```
sudo nfsd enable
```

client — wired workstation profile

mount at `/Volumes/yigitkonur`, with aggressive tuning and `hard` mounts. this profile assumes the server is never absent.

/etc/auto_nfs

```
/Volumes/yigitkonur -
vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,rsize=1048576,wsize=1048576,readahe
ad=16,noatime,rdirplus,actimeo=10 192.168.1.200:/Users/yigitkonur
```

/etc/auto_master

append (otherwise `auto_nfs` is ignored):

```
/-                      auto_nfs
```

apply:

```
sudo automount -cv
```

Skip to content

/etc/nfs.conf (client)

```
nfs.client.access_for_getattr = 1
nfs.client.nfsiod_thread_max = 32
nfs.client.allow_async = 1
nfs.client.access_cache_timeout = 60
nfs.client.statfs_rate_limit = 10
nfs.client.tcp_sockbuf = 16777216
nfs.client.readlink_nocache = 2
nfs.client.max_async_writes = 128
nfs.client.iosize = 1048576
```

do **not** add `nfs.client.is_mobile = 0`. even on a wired machine, `auto` is the correct default — forcing it off gives you nothing here and is catastrophic if you ever move the recipe to a laptop.

mount options (what actually mattered)

option	why
vers=3	NFSv3 is the most stable on macOS. NFSv4 (macOS supports 4.0, not 4.1) has been flaky across releases
hard,intr	hard mounts don't fail with random I/O errors; intr lets you ctrl+c stuck ops
noresvport	common "why won't NFS mount on macOS" fix
nfc	unicode normalization correctness on macOS
locallocks	avoids NLM overhead and stale lock weirdness
nonegnamecache	avoids phantom ENOENT when files churn
rsize / wsize	bigger buffers, fewer trips for big reads/writes
noatime	avoids a write RPC on reads
actimeo=10	fewer metadata RPCs; still reasonable freshness for dev
rdirplus	big win for large dirs: fetch attrs with directory entries

the biggest-win knobs (workstation)

parameter	default	value	why
nfs.client.access_for_getattr	0	1	single biggest perf win: merges permission checks into getattr calls (less RPC spam)
nfs	16	32	more concurrent I/O for lots of small files
nfs	0	1	makes the async mount option actually do something

Skip to content

client — laptop profile (the one i actually run)

four things change from the workstation recipe. each one matters.

1. mount path: `~/mnt/mini`, not `/Volumes/anything`.

this is the single most impactful change on a laptop. `/Volumes` is scanned constantly by Finder, Spotlight, Raycast, Dock, LaunchServices, Time Machine, and every backup tool you install. a hung mount under `/Volumes` spreads the hang to all of them — that's the Raycast crash and the Finder beachball. a hung mount under `~/mnt/` only affects processes that explicitly walked into it.

2. mount options: keep `hard,intr`, add short `timeo` + `retrans`, add `nobrowse`, drop `deadtimeout`.

```
-
vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,rsize=1048576,wsize=1048576,readahe
ad=16,noatime,rdirplus,actimeo=10,timeo=30,retrans=3,nobrowse
```

`hard,intr` keeps data integrity intact while letting you ctrl+c stuck interactive ops. short `timeo=30,retrans=3` (~90s budget per RPC) is the real change: on a dead server, processes bounce off the mount in seconds instead of the 10 minutes `deadtimeout=600` buys you. `nobrowse` keeps Finder from enumerating the mount at all — belt-and-braces in case you ever put it under `/Volumes` anyway.

3. `/etc/nfs.conf` (client) — same as the workstation block, except you explicitly leave `nfs.client.is_mobile` alone.

the macOS default (`auto`) enables "auto-unmount unresponsive network volume on a laptop" behavior. setting it to `0` *disables* that safety net — that's the root cause of beachballs on disconnect, and the reason most "NFS on Mac" blogs produce a machine that's fast on the LAN and unusable off it. leave it at the default. don't even pu

Skip to content

e.

4. no reconnect LaunchDaemon. ever.

a `StartInterval` daemon that pings the server every 30–60s is the thing your Console.app will fill with “attaching network” events forever — whether you’re home or in a café. on a laptop it’s never the right shape. two alternatives:

on-demand automount — accessing `~/mnt/mini` triggers the mount, idle unmounts release it:

`/etc/auto_mini`:

```
/Users/yigitkonur/mnt/mini -
fstype=nfs,vers=3,tcp,rw,hard,intr,noresvport,nfc,locallocks,nonegnamecache,rsize=1048576,wsize=1048
576,readahead=16,noatime,rdirplus,actimeo=10,timeo=30,retrans=3,nobrowse
192.168.1.200:/Users/yigitkonur
```

`/etc/auto_master` (append):

```
/-                      auto_mini
```

apply:

```
sudo automount -cv
```

or manual — a tiny `mount-mini` you run when you actually want the share:

```
# ~/bin/mount-mini
#!/bin/bash
set -e
MOUNT="$HOME/mnt/mini"
mkdir -p "$MOUNT"
sudo mount -t nfs -o
v                    r,noresvport,nfc,locallocks,nonegnamecache,rsize=1048576,wsize=1048576,readahe
a                    ,actimeo=10,timeo=30,retrans=3,nobrowse \
  192.168.1.200:/Users/yigitkonur "$MOUNT"
```

Skip to content

you run it when you want the share. you don't run it when you're on coffee-shop Wi-Fi that has nothing to do with your LAN.

optional macOS overhead to disable

```
# stop creating .DS_Store files on network shares
defaults write com.apple.desktopservices DSDontWriteNetworkStores -bool TRUE
# disable Spotlight indexing on the NFS mount
sudo mdutil -i off ~/mnt/mini
```

don't disable Gatekeeper's quarantine as a "perf tweak". some recipes recommend `defaults write com.apple.LaunchServices LSQuarantine -bool NO`. that flag is not an NFS knob — it disables the quarantine prompt for *all* downloaded files, everywhere. the perf win is noise; the security impact is not. skip it.

security: don't hand-wave this

NFSv3 + `sec=sys` authenticates by uid/gid. no encryption, no signing, no Kerberos. the honest threat model: any machine on your LAN that can spoof uid `501` (trivial on any Mac they control) can read and write every file in your exported home directory.

that's fine on a wired home network with only your own devices. it's *not* fine on a network with guest Wi-Fi, IoT devices on the same subnet, a housemate, or anything resembling a coworking space. treat the share like an unlocked git remote — useful because nobody else is on the wire.

making it reboot-safe

server side (Mac Mini) — always-on, wired, doesn't move

Skip to content

optional: a tiny watchdog so the daemon comes back if it crashes.

```
# /usr/local/bin/nfsd-watchdog.sh
#!/bin/bash
if ! pgrep -x nfsd > /dev/null 2>&1; then
    nfsd enable && nfsd start
fi
if ! showmount -e localhost 2>/dev/null | grep -q "/Users/yigitkonur"; then
    nfsd update
fi
```

root crontab:

```
* * * * * /usr/local/bin/nfsd-watchdog.sh >> /tmp/nfsd-watchdog.log 2>&1
@reboot sleep 10 && /usr/local/bin/nfsd-watchdog.sh >> /tmp/nfsd-watchdog.log 2>&1
```

client side (MacBook) — mobile, roams

nothing. no LaunchDaemon, no cron, no polling. the automount direct map above covers “mount on access, unmount when idle” without a retry loop. if the LAN isn’t there when you `cd ~/mnt/mini`, you get a fast mount error in seconds instead of a ten-minute hang that takes Raycast and Finder with it.

the big lesson: don’t rsync through the NFS mount

this surprised me at first, but the math is unforgiving for small files:

```
LOOKUP → CREATE → WRITE → COMMIT = 4 RPCs × 4ms RTT ≈ 16ms minimum per file
```

for 60,000 files, you’re paying minutes of pure protocol overhead before any real work. for

Skip to content

eam it instead:

```
tar cf - --exclude='.git' --exclude='.DS_Store' -C /local/project . \
  | ssh mini "tar xf - -C ~/remote/project/"
```

for me: 60,000 files in 1 min 57 sec. rsync-over-nfs was not close.

helper:

```
# /usr/local/bin/nfs-sync.sh
#!/bin/bash
# usage: nfs-sync.sh <local-dir> <remote-relative-dir>
LOCAL_DIR="${1:?Usage: nfs-sync.sh <local-dir> <remote-dir>}"
REMOTE_DIR="${2:?Usage: nfs-sync.sh <local-dir> <remote-dir>}"
FILE_COUNT=$(find "$LOCAL_DIR" -not -path '*/.git/*' -not -name '.DS_Store' | wc -l | tr -d ' ')
echo "syncing $FILE_COUNT files: $LOCAL_DIR → mini:~/$REMOTE_DIR"
ssh mini "mkdir -p ~/$REMOTE_DIR"
tar cf - --exclude='.git' --exclude='.DS_Store' -C "$LOCAL_DIR" . \
  | ssh mini "tar xf - -C ~/$REMOTE_DIR/"
```

benchmark snapshot

healthy LAN, same test suite across tuning passes:

test	original	after tuning	after research	total gain
create 1000 files (1–10KB)	101.7s (9/s)	84.3s (11/s)	61.9s (16/s)	+78%
read 1000 files	25.1s (39/s)	20.2s (49/s)	10.3s (97/s)	+149%
stat 1000 files	2.8s (363/s)	1.9s (533/s)	1.9s (535/s)	+47%
overwrite 500 files	31.3s (15/s)	18.8s (26/s)	10.5s (47/s)	+213%
ls -la (1000 files)	42.3s	7.5s	4.4s	9.6x

biggest contributors on a healthy LAN:

- Skip to content `ss_for_getattr = 1`
- `actimeo=10` + `ldirplus` (metadata efficiency)
- bigger `rsize`/`wsize` for the occasional big file

the number *not* in this table is the one that predicts whether your laptop stays usable when the LAN goes away: "`ls` on the mount with the server unreachable." with the naive workstation settings applied to a laptop (`is_mobile=0`, `hard,intr,deadtimeout=600`, mount under `/Volumes`) that's a ten-minute beachball. with the laptop profile above it's a few seconds of ENOENT and you move on.

NFSv3 vs NFSv4 on macOS: don't bother with v4

- macOS supports NFSv4.0, not 4.1
- `vers=4.1` often falls back to v3 anyway
- `vers=4` has had regressions on some Sonoma / Sequoia builds
- tuned v3 is already very good for dev workloads

NFS vs SMB vs alternatives

protocol	small file perf	setup complexity	laptop-friendly	macOS support
NFS (tuned)	good	medium	yes, with the laptop profile	stable with v3
SMB	meh for small files	easy	yes (built-in reconnect)	supported, often sluggish
SSHFS	moderate	easy	yes	project status varies
Mutagen	often strong	medium	yes	active development
Syncthing	async sync	easy	yes	good

the perf test script i used

drop this in `/tmp/nfs-perftest.sh` (edit `NFS_TARGET` to point inside your mount):

Skip to content

```
#!/bin/bash
set -e
NFS_TARGET="$HOME/mnt/mini/dev/some-project"
TEST_DIR="$NFS_TARGET/.nfs-perftest-$$"
t() { perl -MTime::HiRes -e 'print Time::HiRes::time()'; }
log() { printf "  %-35s %7.2fs  %s\n" "$1" "$2" "$3"; }
cleanup() { rm -rf "$TEST_DIR" 2>/dev/null; }
trap cleanup EXIT
mkdir -p "$TEST_DIR"
echo; echo "  NFS perf test"; echo "  $(printf '%.0s-' {1..45})"
t0=$(t)
for i in $(seq 1 100); do dd if=/dev/urandom bs=$((1024+RANDOM%9216)) count=1 of="$TEST_DIR/f$i"
2>/dev/null; done; sync
d=$(echo "$(t) - $t0" | bc); log "create 100 files (1-10KB)" "$d" "$(echo "100/$d"|bc) files/s"
t0=$(t)
for f in "$TEST_DIR"/f*; do cat "$f">/dev/null; done
d=$(echo "$(t) - $t0" | bc); log "read 100 files" "$d" "$(echo "100/$d"|bc) files/s"
t0=$(t)
for f in "$TEST_DIR"/f*; do stat -f "%z" "$f">/dev/null; done
d=$(echo "$(t) - $t0" | bc); log "stat 100 files" "$d" "$(echo "100/$d"|bc) ops/s"
t0=$(t)
for i in $(seq 1 50); do echo "mod $i $(date +%s%N)">"$TEST_DIR/f$i"; done; sync
d=$(echo "$(t) - $t0" | bc); log "overwrite 50 files" "$d" "$(echo "50/$d"|bc) files/s"
t0=$(t); ls -la "$TEST_DIR">/dev/null
d=$(echo "$(t) - $t0" | bc); log "ls -la (100 files)" "$d"
t0=$(t); dd if=/dev/zero of="$TEST_DIR/big" bs=1048576 count=10 2>/dev/null; sync
d=$(echo "$(t) - $t0" | bc); log "write 10MB sequential" "$d" "$(echo "10/$d"|bc) MB/s"
echo "  $(printf '%.0s-' {1..45})"; echo
```

appendix: if you post to Craft.do via API (auto-make markdown safe)

if you have a “github-flavored” version with triple-backtick code fences, you can convert it to Craft.do-safe markdown by turning fenced code blocks into indented code blocks before you POST. here’s the jq filter:

Skip to content

```
def craft_safe_md:
  gsub("\r\n"; "\n")
  | (split("\n")) as $lines
  | reduce $lines[] as $line (
      {out: [], in_code: false};
      if ($line | test("^[[:space:]]*```")) then
        .in_code = (.in_code | not)
        | .out += [""]
      else
        if .in_code then
          .out += ["    " + $line]
        else
          .out += [$line]
        end
      end
    )
  | .out
  | join("\n")
  | rtrimstr("\n") + "\n";
```

use it on the markdown string you're about to send (single-pass jq; don't round-trip through shell variables).

tldr

- if SMB feels slow for small files on macOS, try NFSv3 — server tuning is the same for everyone, client tuning splits by shape
- server side: `require_resv_port=0`, `nfsd_threads=16`, `async=1`, TCP only
- workstation client: `/Volumes/*`, `hard,intr,actimeo=10,rdirplus`, automount direct map
- laptop client: `~/mnt/*`, same options plus `timeo=30,retrans=3,nobrowse`, **leave** `nfs.client.is_mobile` **at its default**, no reconnect LaunchDaemon
- Skip to content t perf knob is `nfs.client.access_for_getattr = 1`; the single setting on a laptop is `nfs.client.is_mobile = 0`
- don't disable Gatekeeper's `LSQuarantine` as a "perf tweak"

- NFSv3 + `sec=sys` is uid/gid auth only; keep the share on a trusted subnet
- for bulk copies use `tar | ssh`, not rsync through the mount

Sources & further reading

1. Apple nfsd man page
2. Apple exports man page
3. Apple automount man page
4. Apple mount_nfs man page

Questions & answers

▶ Why did NFS feel faster than SMB for this setup?

▶ Is this safe to run on a MacBook that leaves the LAN?

▶ Should I use NFSv4 on macOS instead?

See you down the next rabbit hole.

More notes →