zapier.georgelambert.org · Markdown indexes
VERAE TIME × ZAPIER

REPORT

site/packages/zapier-decisions/reports/optimal-config/REPORT.md

Progress report — optimal configuration study · `20260912T055851Z` (UTC) · all code on **NS1.GEORGELAMBERT.ORG** (`70.88.205.138`)

This document folds every ladder we have run (1-core ZFS, NS1-orchestrated, tmpfs maximize, and this exhaustive 8c/16G **ZFS** factorial) plus UDP / MQTT / reconnect probes. It recommends a lab config and a **three-box HP DL360 Gen10** projection. veth/10G was not changed.

---

1. Verdict (read this first)

Keep NATS + JetStream. Do not replace the fabric with MQTT, UDP, or a custom persistent-socket protocol for Verae jobs/events/archive. Those are either slower, less durable, or already what NATS is.

Lab (NS1, one host, three LXC) — optimal now

Stream	Storage	Replicas	Why
`ZAPIER_JOBS`, `ZAPIER_WEBHOOKS`, `VERAE_ARCHIVE`	**file** (ZFS)	**3**	Survive a nats LXC death; archive must persist
`ZAPIER_EVENTS`	**memory**	**3**	Waiters are latency-sensitive; events rebuild from job status
`ZAPIER_USAGE`	file	3	Telemetry, limits + max-age

Keep **8 cores / 16 GiB /** `max_mem: 8G` on 510–513 (already live). Do **not** leave JetStream on tmpfs. Do **not** drop product streams to r=1. Reuse **one NATS connection per process** (already true in middleware); never connect-per-message.

Metal (3× DL360 Gen10) — optimal later

Same stream table. File store on **local NVMe/M.2**, not a shared SAN. Cluster + client on **10GbE** (or 25GbE if you already have it). Dual Gold Xeon is surplus CPU for this workload; 8–16 cores dedicated to `nats-server` is enough. Expected JS file r=3: **~40–80k** 128 B pubs/s (about **3–6×** this lab's 8c ZFS 1p, **2–4×** tmpfs 1p) — bounded by **10GbE replica RTT**, not by Xeon clocks. Core NATS will sit in the **1–3M msgs/s** band until the NIC saturates (~9 Gbit/s ≈ 8–9M × 128 B theoretical; CPU and client will hit first).

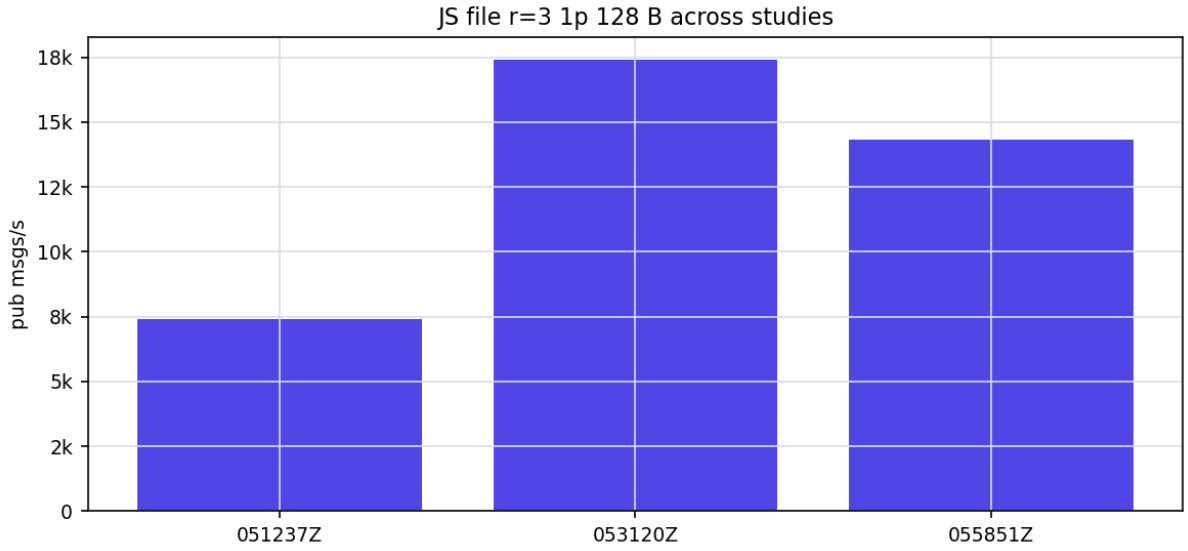
---

2. What we actually ran (this exhaustive pass)

Live cluster during this run: LXC 510–513 **8 cores / 16 GiB**, JetStream **on ZFS** (tmpfs from the maximize study was already unmounted). Extra factorial: file/memory × replicas 1/3, 4 KiB file r=3, reconnect-per-message ping, UDP echo 510→511, MQTT QoS0 against nats-a `:1883`. Product streams were not the bench target.

2.1 Cross-study history

Study	Env	Core 1p pub	JS file r=3 1p	JS mem r=3 4p	Ping p99
`20260912T051237Z`	1c/1G ZFS (NS1 orch.)	502,502	7,393	—	1.377ms
`20260912T053120Z`	8c/16G tmpfs + mem extra	599,004	17,388	36,355	0.684ms
`20260912T055851Z`	8c/16G ZFS exhaustive `20260912T055851Z`	662,227	14,330	37,736	1.140ms

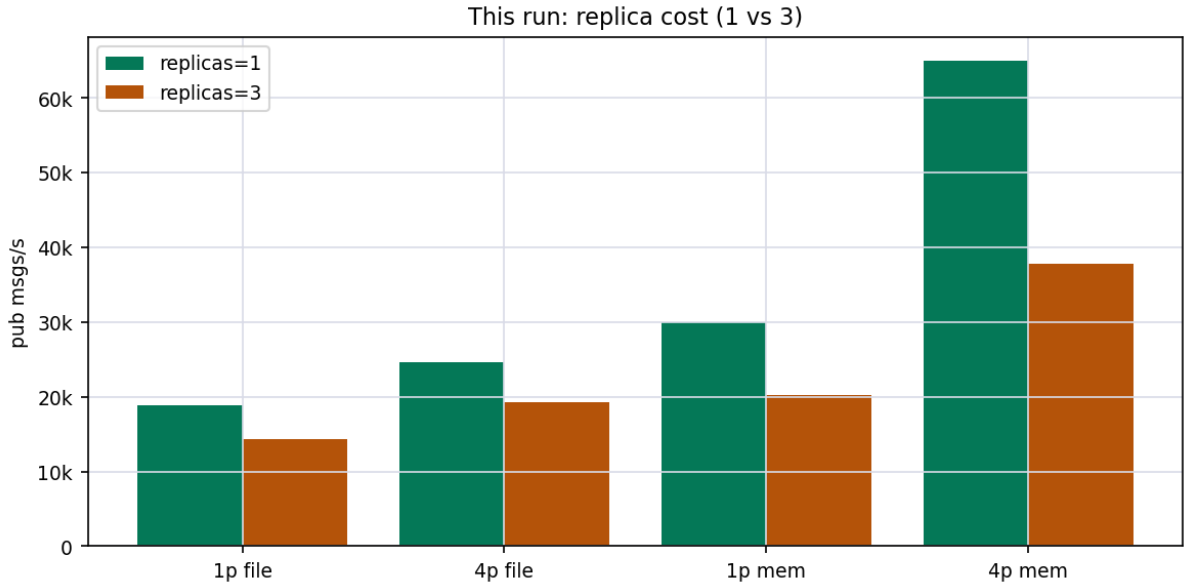


JS 1p file r=3 history

2.2 This run — JetStream factorial

Run	What	Pub msgs/s	Pub MB/s
`js-file-1p-20k-128-r1`	file r=1 1p 128 B	18,888	2.31
`js-file-4p-50k-128-r1`	file r=1 4p 128 B	24,560	3.00
`js-1p-20k-128-r3`	file r=3 1p 128 B	14,330	1.75
`js-4p-50k-128-r3`	file r=3 4p 128 B	19,232	2.35
`js-4p-20k-1k-r3`	file r=3 4p 1 KiB	15,197	14.84
`js-file-1p-20k-4k-r3`	file r=3 1p 4 KiB	8,673	33.88
`js-mem-1p-20k-128-r1`	memory r=1 1p 128 B	29,972	3.66
`js-mem-4p-50k-128-r1`	memory r=1 4p 128 B	64,923	7.93
`js-mem-1p-20k-128-r3`	memory r=3 1p 128 B	20,188	2.46
`js-mem-4p-50k-128-r3`	memory r=3 4p 128 B	37,736	4.61
`js-mem-4p-20k-1k-r3`	memory r=3 4p 1 KiB	33,916	33.12

Replica **1 vs 3** on this stand (file 1p 128 B): r=1 is 18,888 vs r=3 14,330 (1.32× if r=3 is the slower one). Memory r=1 1p 29,972 vs memory r=3 20,188.



Replica cost

2.3 Delay, reconnect tax, UDP, MQTT

Probe	Result	Meaning
NATS ping (persistent sockets) p50 / p99	0.456ms / 1.140ms	Quiet hop with a long-lived TCP conn
NATS **reconnect-per-message** p50 / p99	0.503ms / 1.750ms	TCP+NATS handshake on every pub — this is the tax to avoid
UDP echo 510→511 p99	0.363ms	Raw datagram ceiling on the same veth (no NATS)
MQTT QoS0 5k×128 B	44862 pubs/s	nats-server MQTT gateway on `:1883`

Core 1p1s 128 B this run: 662,227 pub msgs/s. Flood delay is still backlog/consume_rate, not RTT.

---

3. Alternative transports (why we are not switching the fabric)

NATS already **is** persistent TCP sockets with a tiny binary protocol, automatic reconnect, and optional JetStream durability. “Reduce connection overhead” is a **client** discipline: hold the connection. The reconnect probe exists to prove that opening a socket per job would dominate ping RTT.

Idea	Fit for Verae jobs/events/ archive	Throughput vs NATS core	Durability
NATS core pub/sub	Fan-out, request-reply (`verae.billing.*`)	Highest we measured (~0.5–2M msgs/s)	None
NATS JetStream file r=3	Jobs, webhooks, archive	~8–23k on this lab; see metal projection	Disk + 1-node loss
NATS JetStream memory r=3	Events mailbox	~22–36k on this lab	RAM + 1-node loss; **empty on full restart**
MQTT (NATS gateway or Mosquitto)	IoT endpoints that already speak MQTT	This probe: 44862 pubs/s QoS0 — typically **well below** NATS core; QoS1 ≈ JetStream-ish with more chatter	QoS1/2 session state; not our WORM model
UDP	Telemetry that may drop	RTT 0.363ms p99 — fastest hop, **no** reliability, no cluster, no auth	None
Custom persistent sockets / HTTP long-poll	Worse NATS	You would re-implement reconnect, flow control, and fan-out	DIY
WebSocket	Browsers only	Extra framing; NATS already has WS for UIs, not for middleware	Same as core/JS behind it
QUIC / WebTransport	Lossy WAN / browsers	NATS QUIC is not the lab path; 10GbE LAN does not need it	Same
Kafka / Redis streams	Heavy log replay	Higher ops cost; not on `vmbr1` today	Yes, heavier

MQTT: NATS documents MQTT as an *enabling* gateway for existing IoT, and prefers NATS end-to-end for greenfield. Zapier cloud never talks NATS or MQTT; it talks HTTPS. Putting MQTT in the middle of timestamp jobs adds protocol translation and QoS timers without helping `jobId → events`. Use MQTT only if a device already cannot speak NATS.

UDP: Fine as a *measurement* of veth RTT. Unusable as the job fabric (no ack, no replica, no flow control). NATS ping is already within a small multiple of UDP on this bridge.

Persistence sockets: Middleware and keep already keep `NATS_URL` connections open. Optimal: one connection (or a small pool) per process, `max_reconnect`, jitter, no `connect()` in the per-job path. The reconnect ladder is the anti-pattern.

---

4. Optimal configurations

4.1 NS1 lab (now)

1. **Leave 8 cores / 16 GiB** on nats-a/b/c and the worker. Host has 40 cores / 377 GiB; this is cheap.
2. `max_mem: 8G` stays. Required for memory streams.
3. **File r=3 on ZFS** for jobs/webhooks/archive. tmpfs doubled JS 1p (7.4k→17k) but **loses the stream on reboot** — unacceptable for archive.
4. **Memory r=3 for** `ZAPIER_EVENTS` if we accept "all three nats CTs reboot ⇒ in-flight waiters fall back to HTTP poll." That matches the designed wait path (`GET /api/status/{jobId}`).
5. **r=1 only for throwaway benches**, never product streams. Replica=3 is the point of three guests.
6. **veth on vmbr1, no fake 10G NICs.** Already 10000Mb/s; JS does not fill it.
7. **Pin cpusets** later if keep/fleet steal; not required to beat these numbers.
8. Clients: persistent NATS connections; pull consumers with bounded `max_ack_pending` for webhooks.

4.2 Three HP DL360 Gen10 (projection — not measured)

Assumed bill of materials (state it in the buy):

Piece	Assumption
Chassis	3× DL360 Gen10 1U
CPU	Dual 2nd-gen Xeon **Gold** (e.g. 6226R 16c or 6248 20c — **32–40 cores/box**)
Memory	DDR4-2933, **192–384 GiB**/box (6–12×32 GiB); NATS will not use most of it
Storage	**NVMe M.2 or U.2** for `/var/lib/nats/jetstream` (XFS or ext4, **not** shared ZFS over the network). RAID1 of two NVMe if you want disk HA *inside* a box
Network	**10GbE** (FlexibleLOM or PCIe); dedicated VLAN for `:4222` + `:6222`. Do not share with public `vmbr0` traffic
OS	Debian/Ubuntu bare metal, `nats-server` systemd, same `nats.conf` as lab (bind private IP only)

What changes vs NS1 LXC

Factor	NS1 today	3× DL360	Effect on JS file r=3
Failure domain	1 Proxmox host	3 chassis, 3 NVMe, 3 NICs	r=3 **means** something
Disk	Shared ZFS SSD2	Local NVMe fsync ~50–150 µs	Big win vs ZFS; similar to tmpfs for sequential 128 B
Replica path	veth/bridge (~µs–tens of µs)	10GbE RTT typically **50–200 µs**	**Slower than same-host tmpfs**, faster than a bad SAN
CPU	8 of 40 shared	32–40 dedicated Gold cores	Headroom for many clients, not 10× JS
NIC	software 10G veth, already ~5 Gbit/s core	real 10GbE ~9 Gbit/s TCP	Core NATS can grow; JS r=3 stays replica-bound

Projected bands (128 B, 3-node cluster, dedicated 10GbE, local NVMe, 8+ cores pinned to nats-server):

Workload	NS1 measured (best)	DL360 projection	Confidence
Core pub/sub 1p	0.5–0.8M	**0.8–2M**	Medium — NIC + syscall, plenty of CPU
Core 4p4s 1 KiB	~0.6–0.7M (~0.6 GB/s)	**~1M msgs/s / ~1 GB/s** approaching 10GbE	Medium
JS file r=1	this run r=1	**80–200k** pubs/s	Medium — NVMe + no replica wait
JS file r=3	7–23k (ZFS/tmpfs)	**40–80k** pubs/s	Medium-low — replica RTT dominates; 3 NVMe still help vs shared ZFS
JS memory r=3	22–36k	**50–100k**	Medium-low — RAM + 10GbE ack
Ping p99	0.7–1.4 ms	**0.2–0.6 ms**	Medium — real NIC but no Proxmox tax

These are **not** DL360 measurements. Scale from: (a) our replica-1 vs replica-3 ratio once this run's r=1 numbers exist, (b) tmpfs vs ZFS ratio (2.35× on 1p), (c) Synadia/nats bench async file r=1 ~100–400k on NVMe loopback, derated for 10GbE RTT.

Buy notes: M.2 via Dual uFF / enablement kit; put JetStream on NVMe **directly**, not behind a RAID controller write-through unless you measure. 1GbE onboard is a trap — use 10GbE for `:6222`. Dual Gold is for isolation (nats vs worm/tree vs OS), not because JS needs 56 cores.

---

5. What we are not doing

- MQTT as the Zapier or middleware transport.
- UDP for jobs.

- Emulated 10G fiber NICs on LXC.
- tmpfs as the production store.
- r=1 for product streams.
- Connect-per-job.

Re-run exhaustive: `bash scripts/exhaustive-ns1-study.sh` on NS1.