FIELD NOTES FROM A PRESALES SOLUTIONS ARCHITECT

Three Data Centres and No High Availability — Why a Second Site Without a Load Balancer Is Not Redundancy

A presales war story about an infrastructure review where three data centres looked like the safest design in the room — until I asked which box the traffic passes through first — and the four-layer check I now run before I let anyone say the word "redundant".

Daanial Farrukh · 31 October 2026 · https://quabin.com/three-data-centres-no-ha-load-balancer/

The slide had three buildings on it.

"We run three data centres. If one goes down, the others carry the load. We are fully redundant."

The audience nodded. It is a good slide. Three sites in different parts of the city, their own power, their own fibre, their own racks. An auditor had been happy with it. A customer's procurement team had been happy with it. Three is more than two, and two is already more than most.

I asked one question.

"When a visitor opens the portal, which device receives the packet first?"

"The load balancer."

"Where is the load balancer?"

Silence.

"Site A."

"And if Site A is the one that goes dark?"

Longer silence. Then the honest answer that every good engineer eventually gives.

"Then nothing reaches Site B or Site C. The load balancer is the only way in."

Three data centres. One front door. One key.

The Trap, Named

That's the trap. A single point of failure does not show up as a single point of failure when it sits in front of a lot of redundancy. It hides in plain sight, because every diagram draws the load balancer as a small box at the top and the three big data centres underneath, and the eye goes to the big boxes.

Redundancy is counted at the narrowest point of the path, not the widest. Three sites behind one load balancer give you exactly the availability of that one load balancer. Not more.

What High Availability Actually Requires

High availability is a property of a path, not of a building. For a request to survive the loss of a site, every hop it touches must have a living twin somewhere else, and the switch between the twins must happen without a human.

Draw the path and the problem is obvious:

Visitor → DNS → Load balancer → Web / app tier → Database

Each of those four layers has to span sites on its own terms:

- **DNS** has to be able to point visitors somewhere else, or to point them at an address that is itself not tied to one site.
- **The load balancer** has to exist in more than one place, or be a service that lives outside all your sites.
- **The web and app tier** has to be running, warm and current in the second site, not "we can restore it from backup in four hours".
- **The database** has to be replicated to the second site with a defined way to promote the replica, or the app tier there has nothing to talk to.

Lose any one layer and the other three do not matter. In the review I described, the provider had done the hardest part — the buildings — and skipped the cheapest part, the front door.

This pattern is common in three places:

- Providers and enterprises that grew from one site to two or three, and kept the original load balancer because "it works".
- Hardware load balancer deployments where a second appliance was quoted, judged expensive, and quietly dropped from the purchase order.
- Virtual load balancers that were deployed as one VM in one cluster and never moved to an HA pair because nobody owned that line of the design.

Why Nobody Sees It in Testing

Failover tests are almost always run from inside. Someone powers off a web server, watches the load balancer mark it down, and sees the site stay up. That proves the load balancer works. It proves nothing about what happens when the load balancer is the thing that is gone.

The honest test — pull the uplink of the site that hosts the load balancer — is one almost nobody schedules, because it takes the whole platform down for the duration if the design is wrong. So the design is never proven wrong. It is simply believed.

Absence of a failed test is not evidence of a passing one.

There is a second reason the gap stays hidden. Monitoring usually runs from the same site as the load balancer. When that site dies, the monitor dies with it, and the dashboard that would have shown red is itself unreachable. The first alert comes from a customer.

How to Actually Check This

Before anyone promises redundancy, I run four read-only checks. None of them change anything. All of them fit in ten minutes.

Step 1: Find the single address everything resolves to.

```
dig +short www.example.com
```

If the answer is one IP address, and that address is a physical or virtual device in one building, you have found the front door. Write down the address and the site it lives in. If the answer is a set of Cloudflare or other anycast addresses, the front door is already outside your sites, and the question moves to how the origins behind it are configured.

Step 2: Ask where that address is announced from.

```
traceroute -n <that IP>
```

Read the last three hops. If they are routers in Site A every time, the address is reachable only through Site A. Some teams answer "we can re-announce it from Site B" — ask how long that takes and who does it at three in the morning. A manual re-announcement is a recovery procedure, not high availability.

Step 3: Count the load balancers.

On a Linux load balancer pair, this one command tells you if a virtual IP is in play at all:

```
ip addr show | grep -A1 "inet "
```

If the public-facing address appears as a secondary address on an interface and a keepalived or haproxy process is running, you are looking at one member of a pair. If the address is the only one on the box and there is no VRRP process, you are looking at a single device. Then check its twin:

```
systemctl status keepalived
```

No such unit, no twin.

Step 4: Ask the database the same question.

This is the layer teams forget. A second site with a load balancer and an app tier but a database that lives only in Site A is still a single site. Confirm replication is real and current:

```
mysql -e "SHOW REPLICA STATUS\G" | grep -E
"Replica_IO_Running|Replica_SQL_Running|Seconds_Behind_Source"
```

Both Running values must be Yes, and the lag has to be a number you can live with. On older MySQL and MariaDB the command is SHOW SLAVE STATUS\G and the fields are Slave_IO_Running and Seconds_Behind_Master. If this returns nothing, there is no replica, and no amount of load-balancer work will give you a second site.

This step is where most tutorials stop and where mine starts. The checks above only tell you the shape of the problem. The fix is below.

How to Actually Fix the Front Door

There are two honest answers. Which one is right depends on whether you already use Cloudflare.

Option A — Move the load balancer outside your buildings (Cloudflare Load Balancing)

If the site is already proxied through Cloudflare, this is the fastest path, because it adds no hardware and no new network design. The provider in my story was already on Cloudflare for WAF and SSL, which is exactly why I recommended it.

The idea is simple. Cloudflare's edge becomes the load balancer. Each of your sites becomes an **origin** inside a **pool**. A **health monitor** probes every origin from the outside, and when Site A stops answering, traffic is steered to Site B and Site C with no device of yours in the path.

Visitor → Cloudflare edge (LB + health checks) → Origin in Site A | Site B | Site C

Step 1: Give each site its own public origin address. Each data centre needs an address that is reachable without passing through another data centre. If your three sites today share one address that lives in Site A, this step is the real work: you need a web or app entry point per site. It can be a small HAProxy or Nginx VM in each site, or a direct address on the app tier.

Step 2: Create a health monitor. Cloudflare → Traffic → Load Balancing → Manage Monitors → Create. Use an HTTPS probe to a path that touches the application, not just the web server — a /health endpoint that opens a database connection is far more honest than /. Set the expected code to 200.

Step 3: Create a pool per site, attach the monitor. Cloudflare → Traffic → Load Balancing → Manage Pools → Create. One pool per data centre, with that site's origin inside it. Attach the monitor and set a notification email, so a site going red tells a human even though failover is automatic.

Step 4: Create the load balancer on the hostname. Cloudflare → Traffic → Load Balancing → Create Load Balancer. Hostname www.example.com, pools in failover order, proxied on. Choose a steering policy — "Off" gives strict failover in pool order, which is the right starting point for an active–standby design. Save.

Step 5: Verify from outside — twice.

```
curl -sI https://www.example.com | head -5
```

A 200 and a cf-ray header prove you are reaching the application through Cloudflare. Now the test that nobody ran before: block the health-check path on Site A's origin for two minutes, or simply stop its web service, and run the same curl again. The response must still be 200, and Cloudflare → Traffic → Load Balancing must show Site A's pool as Unhealthy and the others Healthy. Bring Site A back and confirm the pool returns to Healthy.

This error is not a bug. It is the verification. If the curl fails while Site A is down, the second site is not yet an independent origin, and you have found the real work.

Option B — Keep it on-premises (keepalived + virtual IP)

If the traffic cannot leave your network — a private portal, a regulatory constraint, an internal API — the on-premises answer is a pair of load balancers sharing a **virtual IP** with VRRP, implemented by keepalived. Two HAProxy or Nginx nodes hold the public address between them. One is MASTER and answers; the other is BACKUP and takes the address within a few seconds when the master disappears.

Visitor → Virtual IP (VRRP) → HAProxy-1 (MASTER) | HAProxy-2 (BACKUP) → App tier

Step 1: Install on both nodes.

```
apt install keepalived haproxy
```

Step 2: Configure VRRP on the first node. This is the entire file for a minimal pair. Mark this as a change — it adds an IP address to the box.

```
vrrp_script chk_haproxy {
    script "/usr/bin/pgrep haproxy"
    interval 2
    weight -20
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 101
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass <SHARED_SECRET>
    }
    virtual_ipaddress {
        203.0.113.10/24
    }
    track_script {
        chk_haproxy
    }
}
```

On the second node, the same file with state BACKUP and priority 100. The track_script block is the part people leave out: without it, keepalived only fails over when the whole node dies, not when HAProxy itself has crashed while the node is still up.

Step 3: Start and verify the address is held.

```
systemctl enable --now keepalived

ip addr show eth0 | grep 203.0.113.10
```

The virtual IP must appear on the MASTER and not on the BACKUP.

Step 4: Fail it and watch. Stop HAProxy on the master and run the same ip addr show on the backup. The address must move within about three seconds. Start HAProxy again and it moves back. If it does not move, the track script is wrong or the two nodes cannot see each other's VRRP multicast — check that the switch and any firewall between them allow protocol 112.

A word of honesty about Option B: a VRRP pair protects you against the loss of one load balancer. It does not by itself protect you against the loss of a whole site, because both nodes normally sit on the same subnet in the same building. To span sites with a virtual IP you need a stretched layer 2 network or a routed design with BGP, and at that point Option A is usually cheaper and safer.

The Bigger Lesson — the Four-Layer Redundancy Checklist

Before the word "redundant" goes on a slide, each of these must have an answer and a command that proves it.

1. **DNS.** Can visitors be steered away from a dead site without a human editing a record? Check: dig +short www.example.com — is it a single site-bound address?
2. **Load balancer.** Is there a second one, somewhere else, that takes over on its own? Check: systemctl status keepalived on-prem, or Cloudflare → Traffic → Load Balancing → pools listed as Healthy.
3. **App tier.** Is the second site running the current release right now? Check: curl -sI https://<site-B-origin>/health — it must be 200 today, not "after a restore".
4. **Database.** Is the replica current and is promotion documented? Check: SHOW REPLICA STATUS\G — both threads Yes, lag acceptable.
5. **Monitoring.** Does at least one probe run from outside all your sites? Check: where does the monitor that would page you actually live?
6. **The real test.** Has the site that hosts the primary load balancer ever been taken offline on purpose, with customers watching? If not, the design is a belief.

Plain takeaway: **three sites behind one door is one site with two very expensive spare rooms.**

What I Tell Every Customer Now

In the next infrastructure review, ask one question:

"If the building with the load balancer loses power, what is the first thing a visitor sees — and who had to do anything to make that happen?"

If the answer is "the same page as always, and nobody", they're in good shape. If the answer is "we re-announce the address from the other site" or "we have never tried", we're not done yet, and the fix is a health check and a pool away.

I write about the questions I ask in real customer rooms — the ones that change architectures and surface the problems no one else noticed. If you found this useful, drop your email below and I'll send the next one when it's ready.

FAQs

Is a second data centre enough for high availability?

No. A second data centre is a place where redundancy can live, not redundancy itself. The request path has four layers — DNS, load balancer, application tier and database — and each one has to exist and fail over in the second site on its own. If any one of them lives only in the first site, the second site cannot serve a single request when the first is down.

Why is a single load balancer a single point of failure even with three data centres?

Because every request has to pass through it before it reaches any of the three sites. Availability is set by the narrowest hop in the path, and a lone load balancer in one building is exactly as available as that building. Three sites behind it inherit its failure, not the other way round.

How does Cloudflare Load Balancing provide high availability across data centres?

Cloudflare runs the load balancer at its own edge, outside all of your sites. You define each data centre as an origin in a pool, attach a health monitor that probes it from the internet, and set a steering policy. When a site stops passing the health check, Cloudflare routes new requests to the remaining healthy pools automatically, with no device of yours in the path and no DNS change by a human.

What is keepalived and when should I use it instead of a cloud load balancer?

keepalived implements VRRP on Linux so two load balancer nodes share one virtual IP address. The backup node takes over the address within seconds when the master or its HAProxy process fails. Use it when traffic must stay inside your own network. Remember that a standard VRRP pair sits on one subnet in one site, so it protects against a dead load balancer, not a dead building.

How do I test whether my setup really fails over?

Take the primary load balancer, or the whole site hosting it, offline on purpose and run an external check at the same time. A plain curl -sI https://www.example.com from outside your network must still return 200 while the primary is down. If it does not, you have not tested failover before; you have only tested a healthy system.

Does the database need to be in both data centres too?

Yes. An application tier in the second site with no database to talk to is a web page that returns errors. The database must be replicated to the second site, the replica must be current, and there must be a documented, preferably automated, way to promote it. Check with SHOW REPLICA STATUS\G that both replication threads are running and the lag is acceptable.

Read online: https://quabin.com/three-data-centres-no-ha-load-balancer/