CLOUD & INFRASTRUCTURE PLAYBOOK · PART 4

Cloud Deployment Archetypes: Matching Availability to Cost

You don't choose five nines — you buy them. Three nines is 43 minutes of downtime a month; five nines is 5 minutes a year. A working guide to the six cloud deployment archetypes, and what each one costs.

FROM BERENBERG & CALDER, ACM COMPUTING SURVEYS (GOOGLE)

- **Zonal — one fault domain**
- **Regional — survives a zone failure**
- **Multi-regional — survives a region failure**
- **Global — five-nines territory**
- **Hybrid & Multi-cloud — compositions, not rungs**

Tom Ward
Enterprise Architect · Cloud & Infrastructure

linkedin.com/in/tomward
July 2026

Availability usually gets discussed as an aspiration — everyone wants to be highly available. But the nines are a budget, and they convert to something uncomfortably concrete: three nines allows 43 minutes of downtime a month; four nines allows 52 minutes a *year*; five nines allows about five. You cannot manage your way into that gap. Berenberg and Calder's survey makes the point plainly: each deployment archetype has a cost to achieve a given number of nines, and the more nines you want, the more it costs. Not every application needs the top of the ladder — the paper separates business-critical applications from line-of-business and internal ones, and only the first group is usually worth the climb.

THE INSIGHT

Availability is a purchase, not an aspiration. You don't decide to have five nines — you choose a deployment archetype capable of delivering them, and pay what it costs. Every rung up buys availability and lower latency; every rung up costs more.

RUNG 1 Zonal

"One fault domain"

Every component runs inside a single zone. If the zone goes down, the workload is restarted elsewhere from its last checkpoint or fails over to a standby. That makes a zone a single failure domain — for software faults as much as for fire or power — so this is not a high-availability posture. It remains the right answer for a real set of cases: high-performance computing and TPU pods that need supercomputer-like bandwidth inside one zone, developer test workloads, and commercial off-the-shelf software whose per-instance licences make redundant copies prohibitively expensive. A primary-zone-with-failover-zone variant raises availability, but roughly doubles cost if the standby is kept hot.

WHAT IT BUYS

- Lowest cost, simplest operations
- No cross-zone egress charges between VMs
- Viable for HPC, dev/test, licence-bound COTS

WHERE IT BREAKS

- × A zone outage is an application outage
- × Recovery is scripted or manual, not instantaneous
- × Not sufficient for most production workloads

RUNG 2 Regional

"Survives a zone failure"

The application is replicated across three or more zones inside one region and all of them actively serve traffic — that active-everywhere posture is what distinguishes regional from zonal, which serves from one zone and fails over. Best practice is to replicate across every zone in the region and size each roughly equally, so capacity genuinely exists elsewhere when a zone is lost. Data is typically replicated synchronously within the region. A single-region-with-failover-region variant adds disaster recovery across regions, usually asynchronously.

WHAT IT BUYS

- Survives the loss of a zone with no user impact
- Latency optimised for users in one geography
- Keeps data inside a single country or jurisdiction

WHERE IT BREAKS

- × A region-wide outage still takes the application down
- × Cross-zone traffic and replication add cost
- × A warm DR region costs money and may not be ready

RUNG 3 Multi-regional

"Survives a region failure"

The serving stack is stitched across several regions. Three models dominate: isolated regional stacks with data sharding, where users are mapped to a region and their data stays there (the paper's example is banking — *us.hsbc.com* for US customers, *hsbc.co.uk* for UK ones); DNS load balancing across isolated regional stacks, the common choice for worldwide serving; and DNS load balancing with a custom multi-region load balancer, which is expensive because you own and operate the routing and authentication layer yourself.

WHAT IT BUYS

- Survives the loss of an entire region
- Lower latency by serving users from nearby regions
- Data residency per jurisdiction, where regulation demands it

WHERE IT BREAKS

- × Costs 3×N — three zones in every region, for zonal redundancy
- × DNS TTL delays cap the availability actually achievable
- × In the sharded model, a region down means those users are down

RUNG 4 Global

"Five-nines territory"

One stack spread worldwide behind global anycast load balancing — a single virtual IP mapped everywhere — with a globally available database. The stronger of its two models is the Global Services Stack, where each microservice is load-balanced globally and can fail over independently. It is also the cheapest of the multi-region and global options, because it needs only one zone per region rather than three, and it expands into new geographies incrementally.

WHAT IT BUYS

- Instantaneous recovery — the requirement for five nines
- Per-microservice failover instead of whole-region failover
- Up to threefold resource savings versus multi-regional

WHERE IT BREAKS

- × The anycast VIP and global database become global failure domains
- × Global components have no regional isolation to fall back on
- × Needs backup VIPs and serious investment in the global tier

Hybrid & Multi-cloud

Hybrid connects on-premises to public cloud, so on-prem effectively becomes another connected zone or region — used both as a transition path and as a permanent home for what cannot move. **Multi-cloud** runs across two or more public clouds so that one provider's failure is survivable, which is the highest theoretical availability because the failure domains are genuinely independent and don't share software bugs. It demands cross-cloud APIs and portable products, and the paper calls it "in its infancy."

Read 5 and 6 sideways, not upward

Hybrid and Multi-cloud aren't higher rungs so much as **compositions**. Each is assembled from the four archetypes above — you pick a deployment model within each cloud or on-premises estate, then combine them. Which is why the honest question isn't "should we be multi-cloud?" but **"which archetype are we running, in each place we run?"**

Four Things That Surprise People

Most of the ladder is intuitive: more redundancy, more availability, more cost. These four findings are not, and each one changes a decision.

01 Global can cost less than multi-regional

A multi-regional deployment needs three zones in every region, because you still need zonal redundancy inside each one — so the cost runs 3×N. A global deployment needs only one zone per region, so it runs at N, and every additional region is incremental rather than multiplied.

Why it matters: the cheaper option is also the more reliable one. That inverts how availability is usually budgeted, where "global" is assumed to be the expensive end of the menu.

02 DNS failover cannot deliver five nines

Five nines leaves roughly five minutes of unavailability a year, which rules out any manual mitigation — recovery has to be instantaneous. DNS can't provide that: TTL delays are built into the protocol, and clients such as browsers routinely disobey TTLs anyway.

Why it matters: if your failover story is "we'll repoint DNS," you have already chosen a lower availability tier. Five nines needs anycast load balancing with integrated health checking.

03 One sick microservice takes down a whole region

With a regional application stack, if any single service in the region is unhealthy the entire region has to be pronounced unhealthy and pulled from serving — even though the rest of the stack is fine. A global services stack doesn't work that way: each microservice is independently load-balanced, so only the affected service moves.

Why it matters: this, more than geography, is the real architectural argument for the global model — and it maps to how microservices are actually owned, by different teams.

04 Failover you never test isn't failover

Standby stacks sit unused most of the time, and there is always a risk the standby isn't ready when it's needed. Worse, the act of failing over exercises code paths that are rarely stressed — and those paths can themselves cause the outage. Load-balancing-based models sidestep the whole problem, because every zone and region is always taking traffic.

Why it matters: if you keep a standby, you must routinely promote it in production. An untested failover is an assumption, not a control.

The Archetype Selection Test

Ten checks before you commit an application to a deployment archetype. Score one point per honest "yes." Below seven, you have a preference rather than a decision — and you'll discover the gap during an incident.

- ☐ The availability target is stated **in minutes, not nines**, and the business has agreed to that number.
- ☐ You know which **application type** this is — business-critical, line-of-business, or internal — and only the first is climbing the ladder.
- ☐ **RPO and RTO are defined** per application, and the archetype was chosen to meet them rather than justified after the fact.
- ☐ **Data residency and regulatory constraints** were mapped before the archetype was picked, not discovered after.
- ☐ You can name **which fault domain each rung protects against** — zone, region, cloud — and which ones you remain exposed to.
- ☐ If you need five nines, **recovery is instantaneous via load balancing** — no manual step fits inside a five-minute annual budget.
- ☐ If you rely on **DNS for failover**, you have consciously accepted that five nines is out of reach.
- ☐ **Failover paths are exercised on a schedule** — a standby you never promote is a standby you cannot trust.
- ☐ Your **global dependencies** — anycast VIP, global database, global control plane — are themselves counted as failure domains.
- ☐ The cost is understood **all-in**: instances per zone, zones per region, cross-zone and cross-region egress, replication, redundant storage, and the skills to operate it.

Going Deeper: References

The source and the provider guidance behind this note. All links verified July 2026.

Deployment Archetypes for Cloud Applications
Berenberg & Calder, ACM Computing Surveys 55(3), Article 61. The survey this note is drawn from — and it's Open Access, so it's free to read in full.
dl.acm.org/doi/10.1145/3498336

Google Cloud — Deployment Archetypes
The Architecture Center's applied, illustrated version of the same six archetypes — the practitioner companion to the paper, kept current.
docs.cloud.google.com/architecture/deployment-archetypes

Google SRE — the books
Error budgets, availability targets, and the operational discipline behind the nines. Free to read online.
sre.google/books

Google Cloud — Disaster Recovery Planning Guide
Turning RPO and RTO into concrete deployment and data-replication choices.
cloud.google.com/architecture/disaster-recovery

Google Cloud — Load Balancing
Global anycast load balancing and integrated health checking — the mechanism behind instantaneous recovery.
cloud.google.com/load-balancing

AWS — Multi-Region Application Architecture
The same archetypes expressed in AWS terms, with a reference implementation.
aws.amazon.com/solutions/implementations/multi-region-application-architecture

Azure Front Door
Microsoft's global entry point and failover tier — the Azure counterpart to global anycast load balancing.
learn.microsoft.com/en-us/azure/frontdoor/front-door-overview

About the Author

Tom Ward is an Enterprise Architect with 20+ years across Fortune 500 financial services, retail, and government — including Wells Fargo and Bank of America — specializing in EA governance, reference architectures, hybrid cloud, and legacy modernization, with recent hands-on GenAI and mobile delivery work.

More field notes: archreviews.com/field-notes · Connect: linkedin.com/in/tomward