

Cloud Computing Examples: Patterns, Strengths, and Pitfalls



Course promise: evaluate real cloud examples end to end—not recite definitions



Every example follows one sequence:
pattern → scenario → strengths → pitfalls → takeaway



Architects and IT pros build pattern fluency;
managers learn to judge trade-offs and failure modes



Scope: IaaS, PaaS, SaaS, containers, serverless,
event-driven, hybrid, and multi-cloud



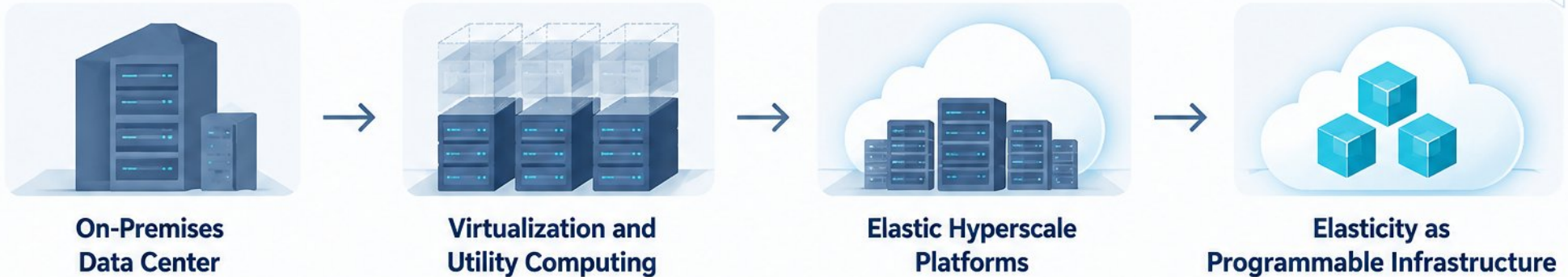
Ground rules: workload fit, cost model, operational
maturity, compliance, data boundaries



You will produce a defensible choice, its failure modes,
and an exit criterion



Background: From Data Centers to Cloud-Native Patterns



Inherited distributed-systems discipline

Where genuinely new bets appear



Evolution path

Evolution path: virtualization and utility computing → elastic hyperscale platforms → elasticity as programmable infrastructure



Patterns matter

Patterns matter: named, reusable solutions let teams debate known trade-offs, not reinvent designs



Cloud-native principles carried through this deck

Cloud-native principles carried through this deck: loosely coupled services, containers, declarative infrastructure, automation, observability, resilience



Operating model shifts

Operating model shifts: pay-as-you-go, commitment discounts, egress pricing, responsibility moving to customer configuration



2026 reality

2026 reality: 98% use cloud-native techniques, 82% of container users run Kubernetes in production, hybrid dominant at 73%

Seven Comparison Criteria for Judging Any Cloud Example



- Pattern structure: problem, forces, solution shape, resulting context, failure modes



- Scalability handles planned load; elasticity matches changing demand



- Total cost includes egress, observability, idle capacity, retries, migration friction



- Lock-in hides in managed services, proprietary networking, data gravity, commitments



- Name the primary constraint first, then score candidates against it

Pattern Example: Lift-and-Shift Migration to Cloud VMs



Solves rapid data center exit, hardware refresh, and compliance deadlines



Legacy three-tier app on cloud VMs behind a load balancer



Strengths: speed, low change risk, familiar ops, cost transparency



Pitfalls: old cost profile, unused elasticity, preserved technical debt



Parallelism exposes deadlocks; managed databases break cross-DB links



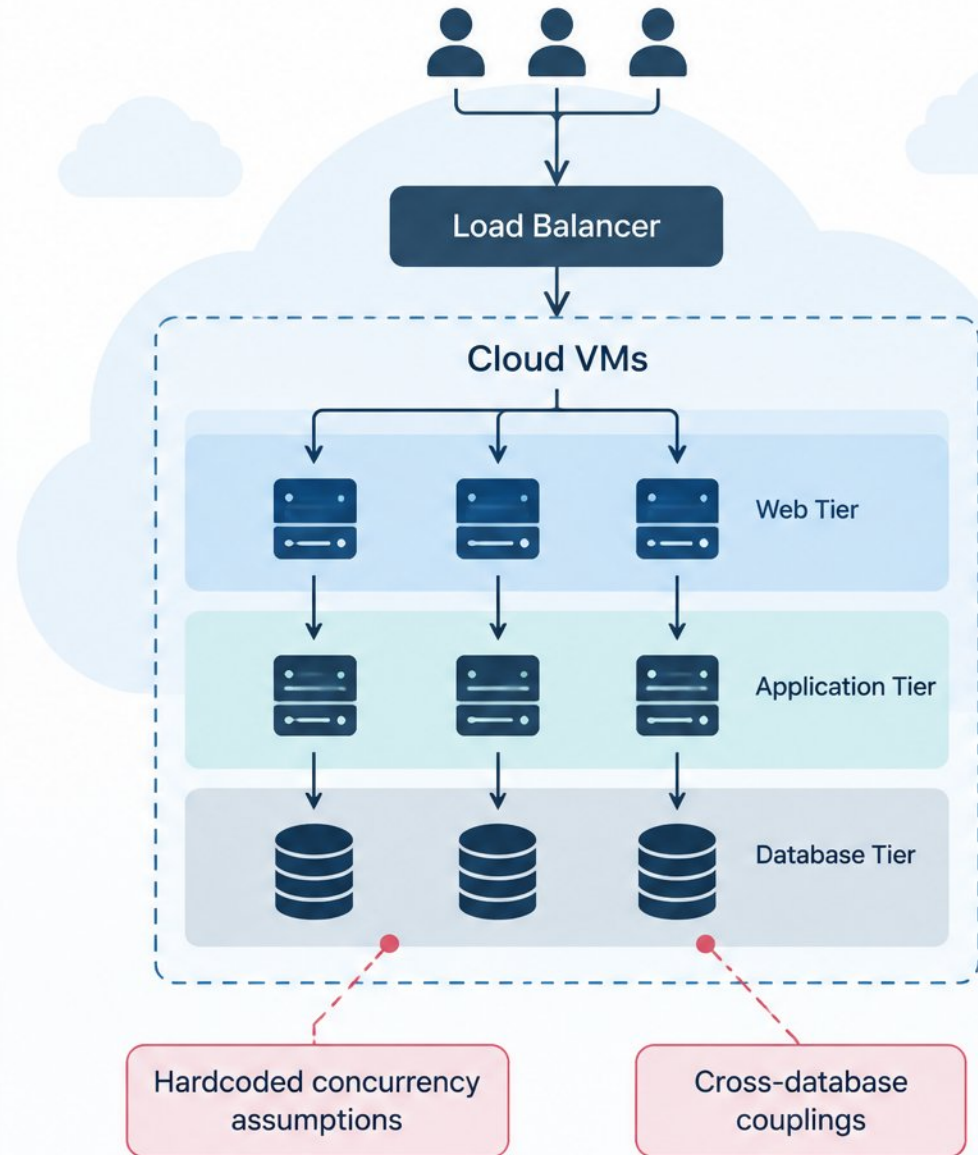
Use only when the system is healthy and only its location changes



Migrate non-production first, use IaC, write a cutover runbook



Watch for orphaned resources and unowned post-migration optimization



Pattern Example: Containers and Microservices on Kubernetes



- Independent deployability, per-service scaling, team autonomy, faster release cadence



- Managed Kubernetes plus CI/CD, API gateway, and database per service



- Strengths: runtime portability, horizontal scaling, resource efficiency, team alignment



- Pitfalls: distributed complexity, service sprawl, config drift, operational overhead, multi-cloud fragmentation



- Top incident impacts: missing requests/limits, confused probes, no PDB, mutable tags, broad RBAC



- Avoid decomposing too finely; a service should justify a small team



- Under ~10 services with predictable traffic, simpler platforms often win on total cost

A City of Services



Edge Layer



Application Layer



Messaging Layer

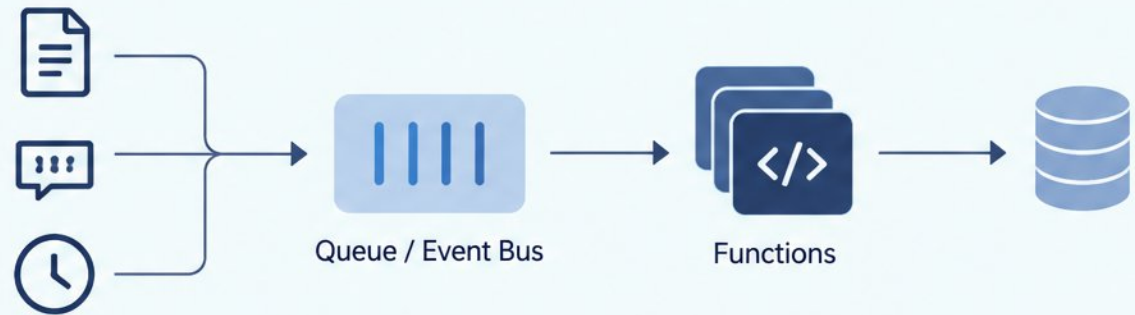


Data Layer

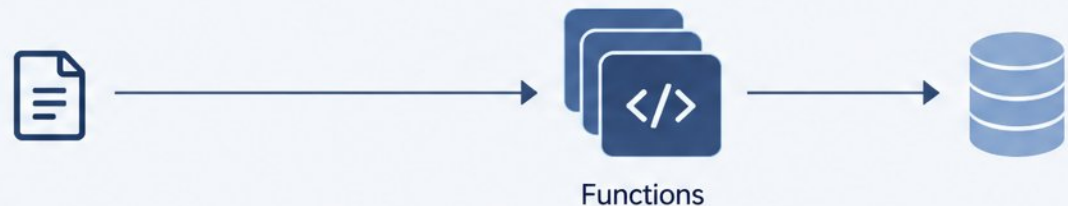
Pattern Example: Serverless and Event-Driven Workloads

- Solves spiky demand without capacity planning; bursts become backlogs, not outages
- Typical: object uploads, queues, scheduled jobs, thin APIs on an event bus
- Cold-start reality by runtime: Go/Node ~100ms; Java/.NET seconds
- Mitigations: trim artifacts, raise memory, provisioned concurrency, async queues
- Pitfalls: silent dropped events, provider-specific contracts, hard local debugging
- Decision cues: workload duration, traffic variance, latency tolerance, portability

Event-Driven Flow (with buffer)



Synchronous Path (no buffer)



? Problem

Spiky demand and unpredictable traffic

⚙ Forces

Variable traffic, cost efficiency, and operational simplicity

💡 Solution Shape

Event-driven, serverless functions with a buffer (queues or event bus)

⚠ Failure Modes

Silent dropped events, provider-specific contracts, hard local debugging

Pattern Example: Hybrid, Multi-Cloud, and Data Gravity

PLACEMENT PATTERNS FOR REAL-WORLD CONSTRAINTS

Steady-State On-Prem + Elastic Cloud



Pitfall:
Doubled operations

Geographic Split



Pitfall:
Inconsistent identity

Edge + Core



Pitfall:
Weak portability

AI GPU Splits



Pitfall:
Drift across substrates

Hybrid Across Data Jurisdictions



Pitfall:
Egress and data gravity



Solves regulatory limits, immovable legacy systems, single-vendor risk, latency needs



Proven patterns: steady-state on-prem + elastic cloud, geographic split, edge + core, AI GPU splits



Strengths: placement flexibility, cross-provider resilience, negotiating leverage, data locality



Pitfalls: doubled operations, inconsistent identity, weak portability, drift across substrates



Egress and data gravity are first-class constraints, not billing surprises



Most large multi-cloud estates still lack unified governance and coherent incident response



Decision cue: does the real benefit justify a second control plane?



Cross-Cutting Strengths That Show Up in Every Pattern



Name the trade-off explicitly in reviews

Cross-Cutting Pitfalls: Cost Drift, Misconfiguration, and Reliability Anti-Patterns



Cost drift: idle resources, over-provisioning, and unpredictable egress



Unallocated cost is unowned cost—tag and assign ownership at creation



Misconfiguration drives 80–98% of cloud incidents across major providers



Provider risk profiles differ: AWS storage, Azure identity, GCP IAM



Weak IAM rises from 87% of SMEs to 98% of large enterprises

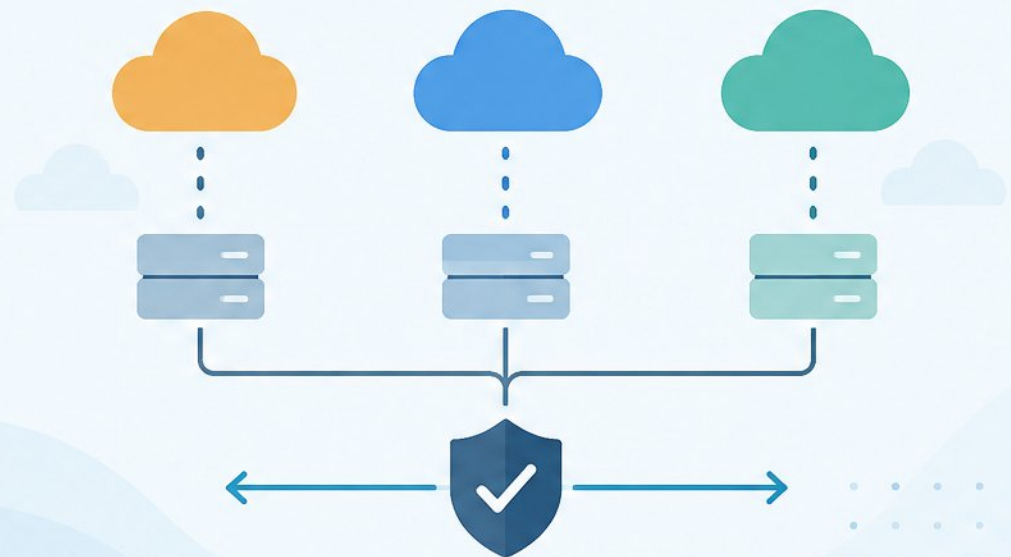
Cost Drift



Misconfiguration Exposure



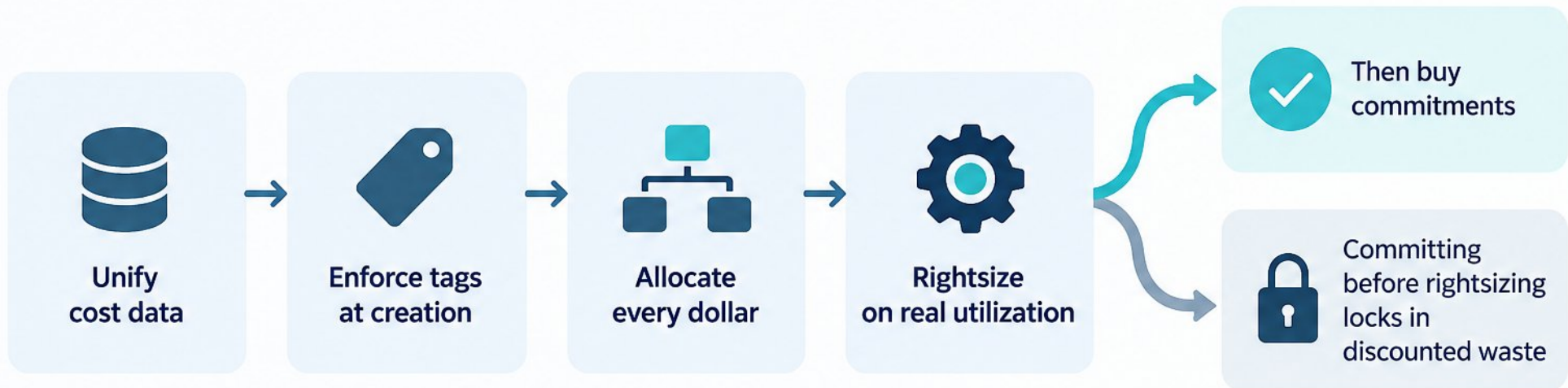
Provider Risk Profiles



Cost and Governance Discipline: FinOps in Practice

- Cloud spend is variable and shaped by daily engineering decisions
- Sequence first: unify cost data, enforce tags, allocate, then optimize
- Rightsize on real utilization before buying commitments—avoid discounted waste
- Track unit economics: cost per customer, transaction, or request
- Automate governance: anomaly alerts, budget guardrails, non-production shutdown

DISCIPLINE
TURNS CLOUD
INTO VALUE



Decision Framework: Selecting a Pattern for a Real Scenario



Start from the dominant constraint: latency, residency, cost, simplicity, or capacity



Classify workload shape first, then score candidates against the seven criteria



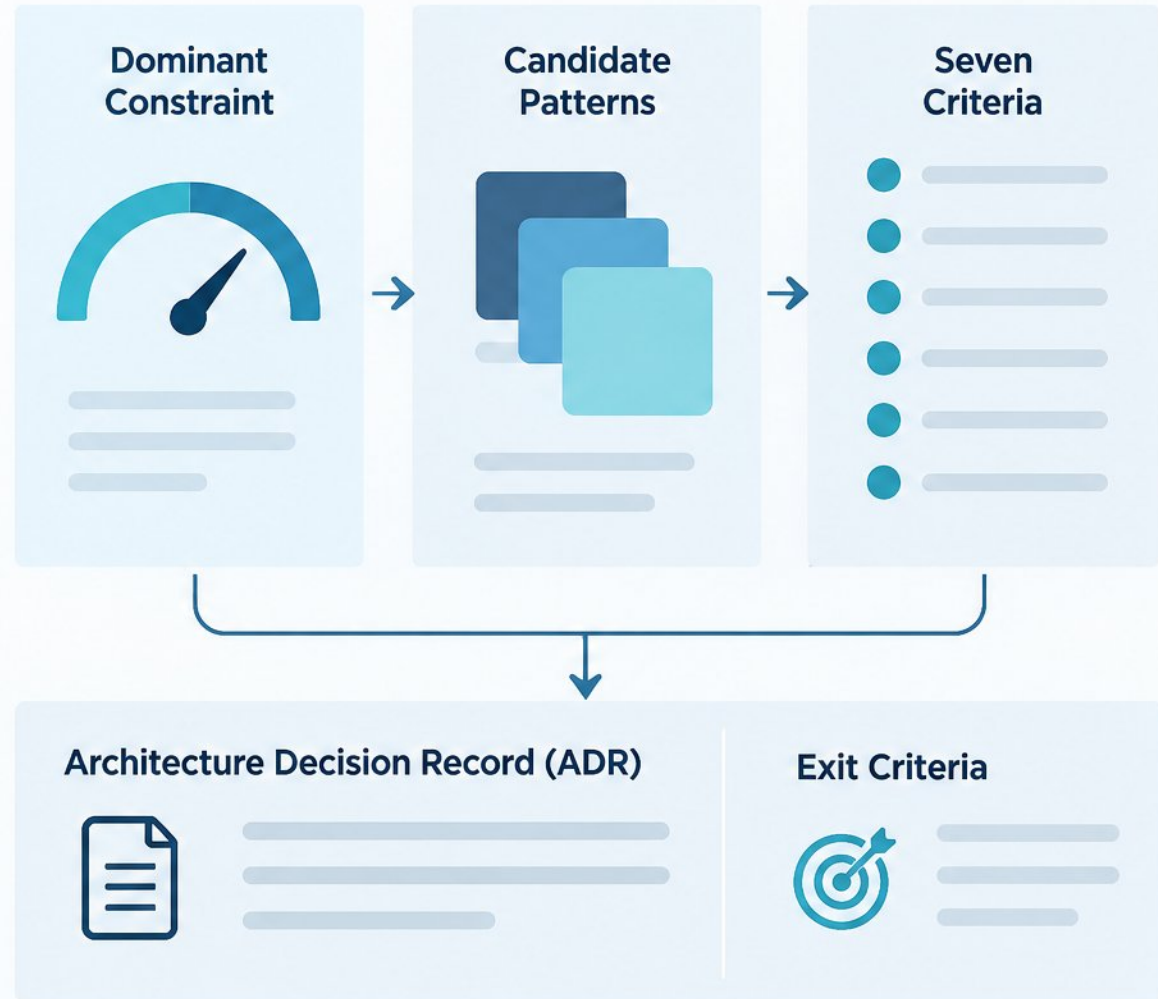
Sequence decisions: retention path, service model, data strategy, then operating model



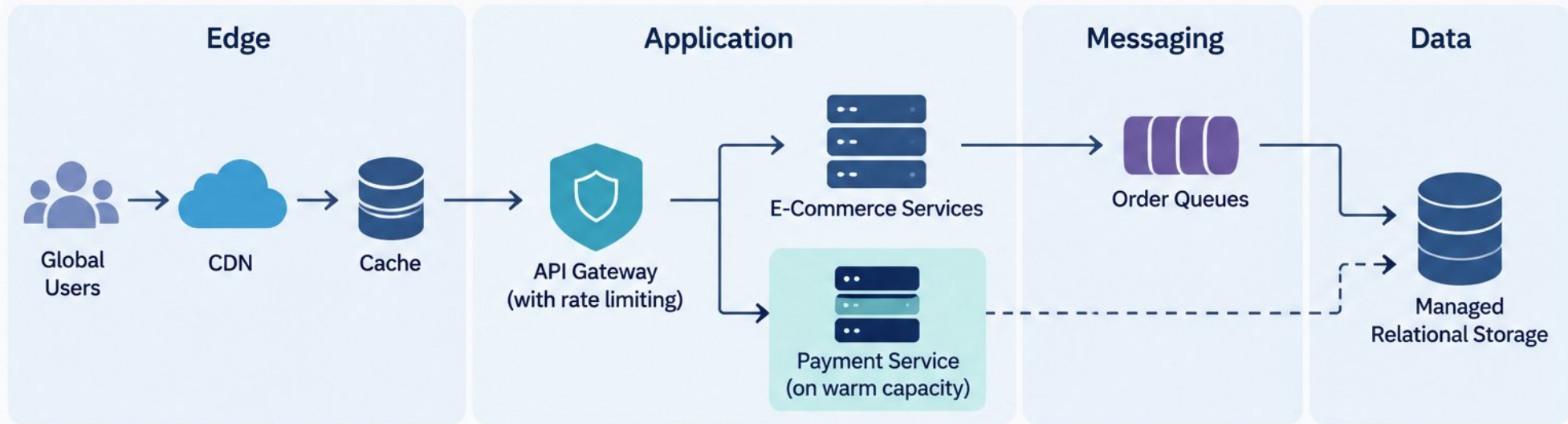
Compare fully loaded cost—egress, lost commitment leverage, second control plane



Write an ADR; define exit criteria before committing



Case Walkthrough 1: Elastic E-Commerce Under Spiky Demand



Scenario: campaign-driven retail spikes, global low-latency users, image-heavy catalogs



Strengths realized: spikes absorbed, distant-region latency low, payment retries stay idempotent



Pattern: CDN and caching, gateway with rate limiting, queues for orders, managed relational storage



Pitfalls: cart concurrency under flash sales, invisible cross-region egress, chatty calls, cold starts at the burst edge



Fix: warm capacity for the revenue path, queues for latency-tolerant work, budget cross-region data flow



Takeaway: elasticity is an architectural property—autoscaling cannot fix an app never built to scale horizontally

Case Walkthrough 2: Regulated Data Analytics Across Jurisdictions



Scenario: multi-region analytics and AI, with regulated data confined to approved jurisdictions



Residency and sovereignty override provider, region, and tooling choices



Pattern: localized data zones, one control plane, private links, policy-based identity



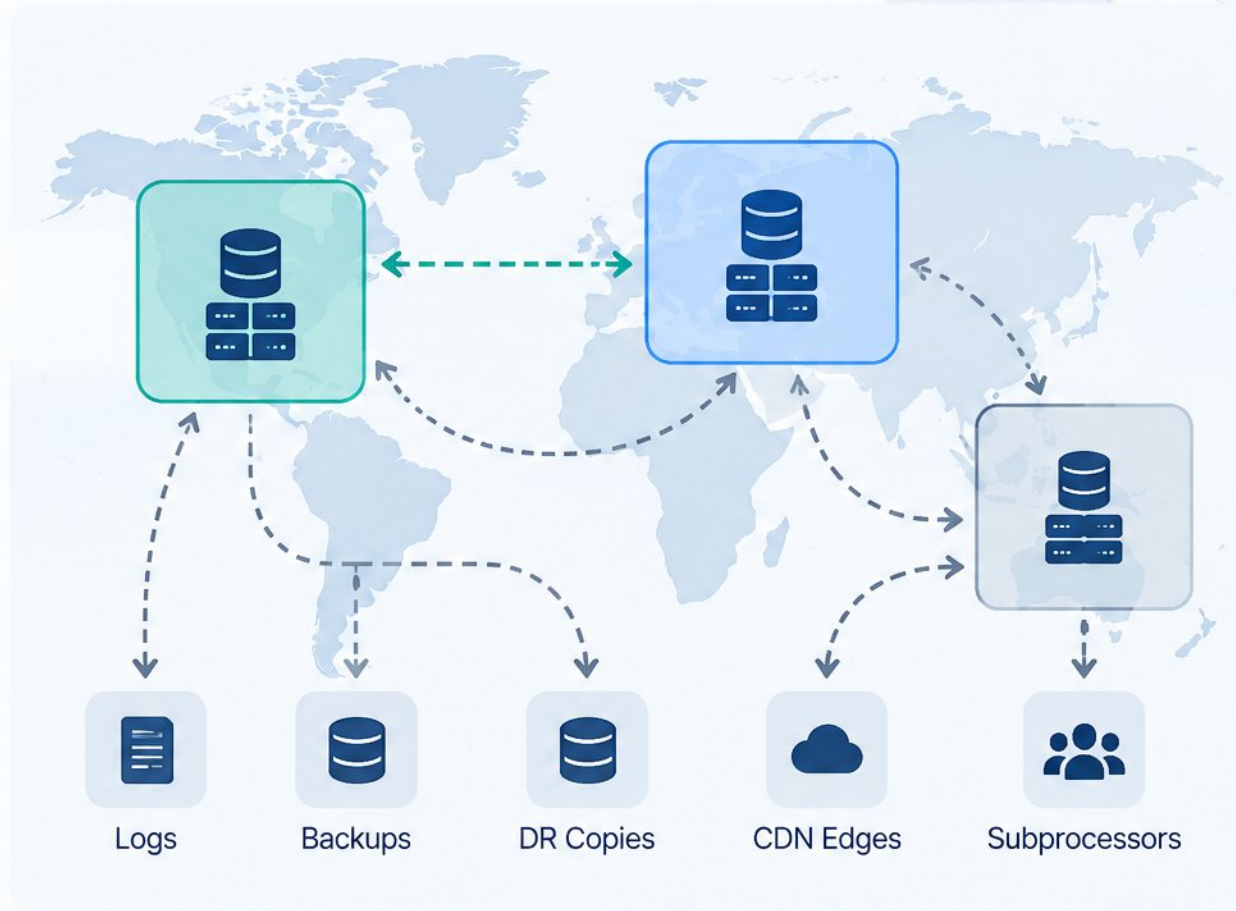
Pitfall: the residency illusion—logs, backups, DR copies, CDN edges, subprocessors



Default geo-replication quietly creates violations nobody chose



Fix: classify data, map flows, lock replication regions, re-validate after change



Approved, in-jurisdiction data flow



Other data paths that can cross borders

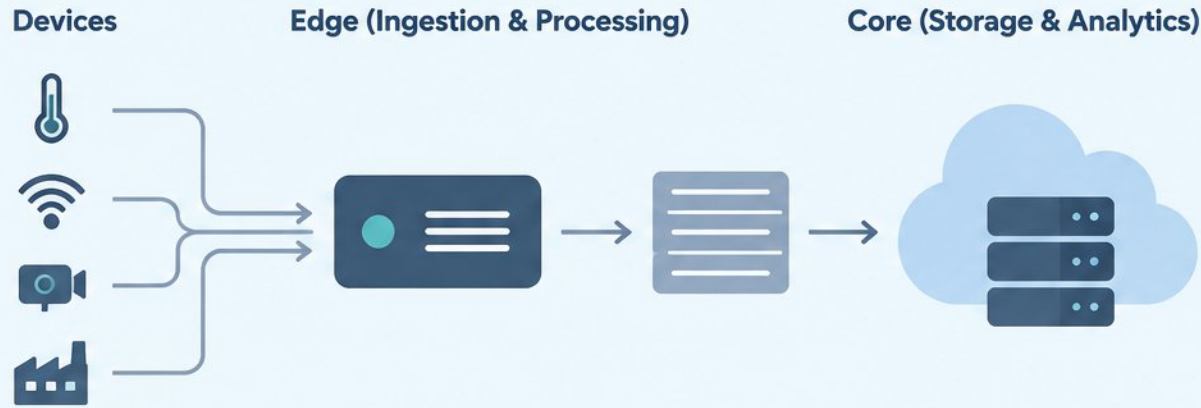


Approved data zone

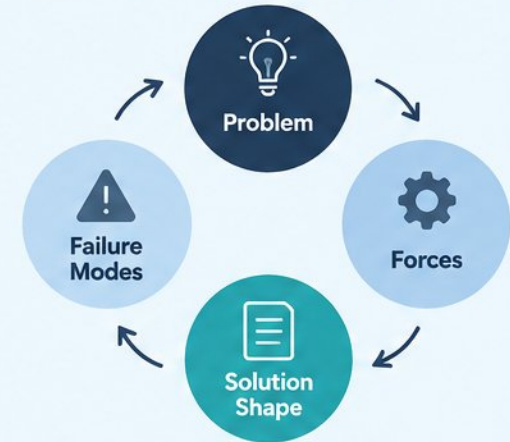


Other region

Case Walkthrough 3: IoT Ingestion, Comparison Table, and Key Takeaways



Pattern-to-Pitfall Loop



Comparison Table Across the Session

	Lift-and-shift	Containers and microservices	Serverless	Hybrid and multi-cloud
 Speed				
 Autonomy				
 Cost profile				
 Operational burden				

Key Points

- **Buffers**, not raw speed, keep bursty device telemetry durable
- **Keep** ingestion and signal processing near the source
- **Pitfalls**: no dead-letter path, edge logic creep, rising per-device cost
- **Trade-offs**: lift-and-shift cheapest; containers cost most to operate
- **Takeaways**: queue before bursts, rightsize before commitments, define exit criteria

PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/90794883/public