FREE CHECKLIST

12 AWS Mistakes Killing Your Budget

The most expensive mistakes I see in cloud setups — and how to fix each one.

---

1 NAT Gateway data transfer costs

One of the most surprising line items. Every byte from private subnets to the internet goes through NAT Gateway at $0.045/GB + hourly charge.

> **Quick fix:** Use VPC endpoints for S3 and DynamoDB. Review if all traffic actually needs NAT.

2 No Reserved Instances or Savings Plans for predictable workloads

On-demand pricing for workloads that run constantly is like paying hotel rates to live somewhere. You're leaving 30-60% on the table.

> **Quick fix:** Commit to 1-year Savings Plans for your baseline compute.

3 Running outdated versions of managed services

RDS, EKS, and other managed services charge extended support fees once a version reaches end-of-life. Fees aren't huge, but the upgrade is usually easy and brings new features, performance, and security patches.

> **Quick fix:** Check end-of-support dates for RDS engines, EKS, and other managed services. Plan upgrades before extended support fees kick in.

4 Not using Spot instances for non-critical workloads

CI runners, batch jobs, stateless workers, and non-production environments don't need on-demand pricing. Spot instances run the same EC2 hardware at 70-90% discount — interruptions are the trade-off.

> **Quick fix:** Identify interruption-tolerant workloads (CI, batch, dev). Move them to Spot via EKS managed node groups, Karpenter, or EC2 Fleet. Use mixed instance types to reduce interruption risk.

5 Running oversized EC2 instances 24/7

You picked an instance type during a spike and never downsized. Check CloudWatch CPU/memory — if you're consistently under 30%, you're overpaying. Many workloads also run fine on t3 or t4g (Graviton) instances at a fraction of the cost — but nobody tested the switch.

> **Quick fix:** Right-size using AWS Compute Optimizer. Consider t4g Graviton instances for 20% savings with no code changes.

6 Oversized RDS instances with no autoscaling

Your database is provisioned for peak load but runs at 10% most of the day. Slow queries, missing indexes, and inefficient schema design force you to compensate with bigger instances. Multi-AZ on dev/staging environments doubles the waste.

> **Quick fix:** Right-size to t4g instances for small workloads. Use Aurora Serverless v2 only for highly variable load. Tune slow queries and add proper indexes before upsizing. Remove Multi-AZ from non-production.

7 Running Kubernetes nodes at low utilisation

Your EKS cluster has 10 nodes but pods only use 30% of capacity. Cluster Autoscaler is either misconfigured or missing.

> **Quick fix:** Set proper resource requests and limits on pods. Enable Karpenter for automatic node scaling. Adjust scaling rules to match actual usage patterns.

8 Forgotten resources — idle load balancers, unattached EBS volumes, old snapshots

They don't show up in your app, but they show up on your bill. Every month.

> **Quick fix:** Run AWS Trusted Advisor or Cost Explorer to find zombie resources.

9 Logging everything to CloudWatch at full retention

Default log retention is "forever." Terabytes of debug logs you'll never read, costing hundreds per month.

> **Quick fix:** Set log retention policies (7-30 days for most environments). Archive to S3 if needed.

10 No S3 lifecycle policies

Uploads pile up. Old versions accumulate. Multipart uploads fail and leave fragments. S3 costs creep up invisibly.

> **Quick fix:** Add lifecycle rules — use Intelligent-Tiering for unpredictable access, move to Infrequent Access after 30 days for rarely accessed data, Glacier after 90. Delete incomplete multipart uploads and expire old object versions.

11 Cross-region and cross-AZ data transfer

Services chatting across regions or availability zones rack up transfer costs silently. A chatty microservice architecture makes this worse.

> **Quick fix:** Co-locate services that talk frequently. Use VPC peering instead of public endpoints.

12 No cost alerts or budgets set up

You only find out about cost spikes when the invoice arrives. By then you've been overpaying for weeks.

> **Quick fix:** Set up AWS Budgets with alerts at 50%, 80%, and 100% of your expected spend.

Bonus Tips

- 💰 **Ask AWS for credits.** Migration credits, PoC credits, startup credits via AWS Activate and MAP. Most companies never ask — your account manager can often unlock thousands in free credits.
- 🏷️ **Tag everything.** Without proper tagging, you can't attribute costs to teams, projects, or environments. Untagged resources are invisible waste.
- ⚡ **Add caching layers.** Redis or ElastiCache in front of your database reduces reads, lowers RDS load, and lets you run a smaller instance. CloudFront for static assets cuts data transfer costs.
- ⏰ **Schedule non-production environments.** Dev and staging don't need to run 24/7. Auto-stop outside business hours and you save 65%+ on those resources.
- 🔧 **Use Graviton across the board.** Not just EC2 — RDS, ElastiCache, and Lambda all support Graviton. 20-40% cheaper with no code changes.

Want someone to find these in your account?

Book a free 30-minute call and I'll show you where the waste is hiding.

Book a Free Call

or email me directly: viktar@patotski.com

Viktar Patotski — Software Engineer & Architect · 20+ years experience · **viktar@patotski.com**