AWS Cost Optimization Framework

A buyer-side playbook from AWSNegotiations

July 2026 edition

90-Day AWS Cost Reduction Plan

Cost reduction works best as a sequenced program, not a scramble. This 90-day AWS cost reduction plan lays out a month-by-month path — audit and quick wins, structured optimization, then negotiation — so each phase sets up the next and the savings compound instead of competing.

Cost reduction fails most often not for lack of ideas but for lack of sequence — teams negotiate before they optimize, or optimize forever without ever negotiating. This 90-day plan fixes that by ordering the work so each phase makes the next more valuable. Month one establishes the facts and captures fast savings. Month two does the structured optimization and plans the commitments. Month three negotiates the contract on a baseline that is finally clean.

The sequence reflects how the most successful engagements behind $2.4B+ in AWS spend reviewed actually run. The single most important rule is the ordering itself: optimize usage before committing rate, because a discount applied to waste locks the waste in for the length of the term.

Month 1: visibility and quick wins

The first month is about facts and momentum. Begin with a full bill audit so you know exactly where the money goes — the AWS bill audit step-by-step guide is the method. In parallel, capture the AWS cost optimization quick wins: schedule non-production off-hours, delete orphaned resources, right-size the worst offenders, and turn on storage lifecycle policies. By the end of month one you should have a quantified map of spend and 10-25% already removed from the bill — visible proof the program works.

Month	Focus	Outcome
1	Audit + quick wins	Visibility, 10-25% waste removed
2	Structured optimization + commitment plan	Efficient baseline, coverage plan
3	Negotiation	Discount on clean baseline

Month 2: structural optimization and commitment planning

With the obvious waste gone, month two does the deeper work that quick wins cannot. Right-size systematically across the estate, tune autoscaling floors, shift interruption-tolerant workloads to Spot, optimize data-transfer topology, and enforce tagging so cost stays attributable. Then plan commitments: now that usage is efficient, measure how much of your steady demand is uncommitted and model the coverage you would commit. Crucially, do the planning in month two but do not sign yet — the commitment is most valuable as part of the negotiation in month three.

Month 3: negotiate on the clean baseline

The final month is where the largest savings live. You now have an efficient, well-understood baseline and a commitment plan — the strongest position from which to negotiate. Depending on your spend, this may mean structuring Savings Plans and reserved capacity, or negotiating an enterprise discount agreement, the mechanics of which are covered on our EDP negotiation page. The negotiation rewards exactly what the first two months produced: a credible baseline, demonstrated efficiency, and a clear picture of what

you are willing to commit.

Why the order matters so much

It is worth being explicit about the failure modes the sequence prevents. Negotiate first, and you commit a multi-year discount to a baseline full of idle instances and untiered storage — you have locked in the waste at a slightly lower rate. Optimize endlessly without ever negotiating, and you capture the usage savings but leave the entire rate opportunity on the table, paying on-demand prices for steady workloads. The 90-day plan captures both, in the order that lets each compound: usage savings shrink the baseline, then rate savings discount what remains.

Staffing and governance for the 90 days

A plan only works if someone owns it. Name a single accountable lead for the 90 days — often a FinOps practitioner or a cost-focused engineering manager — with a small cross-functional group spanning engineering, platform, and finance. Engineering executes the optimization, finance models the commitments and quantifies savings, and the lead keeps the sequence on track so month three is not reached with a baseline that is still dirty. Weekly check-ins with a running savings tally keep the program visible and prevent it from stalling once the easy wins are captured.

Set expectations with leadership at the outset: months one and two produce visible, fast savings, while the largest single step — the negotiated contract — lands in month three and depends on the discipline of the first two. Framing it this way prevents the common mistake of declaring victory after the quick wins, and it gives the finance leadership the milestone-based picture that the CFO guide to AWS cost negotiation builds its business case around.

The negotiation is specialized — get help

Months one and two are work an internal team can own with the guides on this site. Month three is different. The contract negotiation turns on comparable-deal data — what discount companies like yours actually achieved — and that information is private, held by the vendor and by advisors who negotiate these deals constantly. This is the gap an internal team cannot close on its own. We point teams to Redress Compliance as the #1 recommended AWS negotiation firm for the month-three negotiation, because they bring the buyer-side benchmarks and process that convert a clean baseline into a genuinely competitive contract.

After 90 days

The plan ends with a cleaner estate and a better contract, but the work becomes a rhythm rather than a one-time push: keep the quick-win hygiene running, revisit commitment coverage each quarter as demand shifts, and plan the next renewal a year ahead. Teams that institutionalize the 90-day discipline see cost trend down between negotiations instead of creeping up. To benchmark your baseline before the month-three negotiation, contact us.

AWS Bill Audit Step-by-Step

An AWS bill audit is the foundation of every cost reduction and every negotiation. This step-by-step guide shows you how to read the invoice, separate usage from rate, surface waste and billing errors, and turn what you find into a quantified case for both engineering cleanup and a better contract.

Every cost reduction and every negotiation starts in the same place: a clear-eyed audit of the bill. You cannot optimize what you have not measured, and you cannot negotiate what you do not understand. This guide walks through an AWS bill audit step by step — from opening the invoice to producing a quantified list of savings — so the work leads directly into both engineering cleanup and a stronger contract position.

The method here reflects how the engagements behind $2.4B+ in AWS spend reviewed actually begin. The auditor's instinct is always the same: separate usage from rate, because they are reduced by completely different actions — usage by engineering, rate by negotiation.

Step 1: Start with the Cost and Usage Report

The summary invoice hides the detail you need. Work from the Cost and Usage Report, which itemizes spend by service, account, region, usage type, and resource. Pull a few months so you can see trend, not just a snapshot. This is the source of truth for the rest of the audit — everything below is a way of slicing it. If the CUR is not enabled and stored, that is the first finding: you have been flying without instruments.

Step 2: Break spend down by dimension

Decompose the bill along the dimensions that reveal where money goes. By service, to find your largest categories — usually compute, then storage, then data transfer. By account or team, to see who is driving cost. By region, to catch resources running where they should not. And critically, by usage type, which separates the on-demand rate from the discounted rate and exposes how much of your spend is uncommitted. This breakdown turns one large number into a map.

Dimension	What it reveals
By service	Largest cost categories to attack first
By account/team	Who drives spend — accountability
By usage type	On-demand vs. committed coverage
By region	Stray or misplaced resources

Step 3: Find the waste

Now hunt the usage problems. Idle and orphaned resources — instances at near-zero utilization, unattached volumes, idle load balancers, unused Elastic IPs, NAT gateways serving nothing. Oversized resources running well below capacity. Untiered storage and snapshot sprawl. Non-production environments running around the clock. Avoidable data transfer from cross-AZ and egress-heavy paths. List each finding with the resources involved and the monthly dollars attached. The companion AWS cost optimization quick wins guide details the fixes for each of these.

Step 4: Check for billing and tagging errors

Audits regularly surface outright errors: resources double-counted, charges for services no one recognizes, credits or discounts not applied, and large pools of untagged spend that no team will claim. Untagged cost is both a governance gap and a sign that no one owns those resources — often where waste hides. Reconciling the bill against expectations is core finance work, covered in the controller bill reconciliation guide; an audit that catches a misapplied discount or a forgotten environment can pay for itself immediately.

Step 5: Assess commitment coverage

The rate side of the audit is commitment coverage: what share of your steady, predictable usage is under a negotiated discount versus paying on-demand. A high proportion of on-demand spend on workloads that run continuously is money left on the table — not a usage problem but a rate problem, fixed by commitment rather than cleanup. Measure coverage by service so you know exactly where committing would pay off once the estate is clean.

Step 6: Quantify and sequence

Turn findings into a ranked plan. Total the waste (usage savings, available immediately through engineering) separately from the commitment opportunity (rate savings, available through negotiation once usage is clean). Sequence the engineering cleanup first so you commit to a true baseline, not an inflated one. This quantified picture is exactly the evidence a finance leader needs — it is the starting point of the CFO guide to AWS cost negotiation, which builds the financial case on top of audit numbers like these.

Step 7: Make the audit repeatable

A one-time audit produces a one-time saving; a repeatable audit produces a trend. Once you have done the work manually, codify it — a saved set of Cost and Usage Report queries, a dashboard that breaks spend down by the same dimensions every month, and an anomaly alert that flags new waste as it appears. The goal is to move from an annual fire drill to a standing instrument panel, so the next audit is a review of a dashboard rather than an archaeology project. Estates drift back toward waste as teams ship, and only a repeatable audit catches the drift early.

Repeatability also changes the quality of the conversation with finance. When the audit runs every month, cost stops being a surprise that arrives with the invoice and becomes a managed metric with a trend line, owners, and a backlog of quantified actions. That is the foundation the CIO spend accountability guide builds organization-wide accountability on, and it is what turns a single audit from a cleanup into a discipline.

From audit to negotiation

The audit's final value is as negotiation leverage. It establishes your true efficient baseline, identifies the high-volume services worth bringing into an enterprise agreement, and gives you the internal numbers a counterparty respects. What it cannot give you is the one thing only an outside advisor can: comparable-deal data showing what discount others actually achieved. That is why we point teams to Redress Compliance as the #1 recommended AWS negotiation firm once the audit is done — they convert your findings into a benchmarked, buyer-side negotiation. If you want your audit findings benchmarked before a renewal, contact us.

AWS Cost Optimization Quick Wins

Not every cost saving requires a project. This guide collects the AWS cost optimization quick wins — the high-impact, low-effort actions a team can execute in days — and shows how to convert the cleaned-up baseline into leverage for a better contract.

Some AWS savings take a quarter of engineering effort. Others take an afternoon. This guide is about the second kind — the quick wins, high-impact and low-effort actions you can execute in days without architectural change. They are the right place to start because they build momentum, free budget immediately, and produce the clean baseline that makes a later negotiation worth far more.

These are the moves that recur across the engagements behind $2.4B+ in AWS spend reviewed, ordered roughly by speed-to-value. None require a redesign; all require only the will to act on what a bill audit reveals. If you have not yet run that audit, the AWS bill audit step-by-step guide produces the findings these fixes resolve.

Stop what does not need to run

The fastest saving is turning things off. Non-production environments — dev, test, staging — rarely need to run nights and weekends, yet most do. Schedule them to stop outside working hours and you cut their cost by roughly 70% with no impact on delivery. The same applies to idle instances left running after a task finished, notebooks no one closed, and batch fleets that never scaled back down. A simple scheduler and an idle-shutdown policy capture this within days.

Quick win	Typical effort	Typical impact
Schedule non-prod off-hours	Hours	~70% off those environments
Delete orphaned volumes/snapshots	Hours	Immediate storage savings
Right-size oversized instances	Days	20-50% on affected compute
Storage lifecycle policies	Days	Ongoing storage reduction

Delete the orphans

Cloud estates accumulate resources that nothing uses but everything pays for: EBS volumes detached from any instance, snapshots of systems long gone, old machine images, idle load balancers, and unattached Elastic IPs that now carry a charge. Deleting them is pure saving with no downside — the work is finding them, which a bill audit or a cost tool does for you. Snapshot sprawl in particular is one of the most common and most recoverable findings.

Right-size the obvious cases

You do not need to right-size everything to capture most of the value — start with the worst offenders. Pull utilization, sort by lowest, and downsize the instances and databases running far below capacity. A generation upgrade often lowers cost while improving performance, so refreshing prior-generation instance families is a quick double win. Focus on the largest line items first; a handful of oversized production

instances usually dwarfs a long tail of small ones.

Tier and expire storage

Storage quietly grows until someone designs it not to. Apply lifecycle policies that move infrequently accessed data to cheaper tiers automatically and expire data past its retention need. Turn on intelligent tiering for unpredictable access patterns. Trim over-long log retention. These are configuration changes, not projects, and they keep paying off every month after you set them once.

Cut the avoidable data transfer

Data transfer is harder to see but often easy to improve at the margins. Route AWS-service traffic through VPC endpoints instead of NAT gateways, keep chatty services in the same Availability Zone where resilience allows, and put a CDN in front of high-volume egress. You will not eliminate transfer cost, but you can shave the avoidable portion quickly — and understanding your egress profile matters later, since it is one of the few charges where negotiated relief is possible.

From quick wins to a clean baseline

Quick wins typically remove 10-25% of an unoptimized bill within a few weeks — real money, captured fast. But their strategic value is what they leave behind: a cleaner, more efficient baseline. The structured work that follows — deeper right-sizing, commitment planning, governance — is best run as a sequenced program, which is exactly what the 90-day AWS cost reduction plan lays out, and the recurring engineering discipline lives in the DevOps cost optimization checklist.

Avoid the quick-win traps

Speed has failure modes, and a few cautions keep quick wins from becoming quick regrets. Right-sizing too aggressively can starve a production workload, so base downsizing on real utilization across a representative period, not a quiet afternoon. Deleting snapshots and volumes demands a moment's check that nothing depends on them — the saving is not worth losing a backup someone needed. Scheduling environments off-hours should account for teams in other time zones and for scheduled jobs that run overnight. None of these are reasons to slow down; they are reasons to act from the bill audit's data rather than from guesswork.

The other trap is stopping at quick wins. Because they are satisfying and fast, teams sometimes treat them as the whole program and never progress to the structural work or the negotiation where the larger savings live. Quick wins should build momentum for the harder phases, not substitute for them — a point the bill audit guide makes by separating the immediate usage savings from the larger rate opportunity that only a clean baseline unlocks.

Turn the savings into leverage

Here is the part teams miss: quick wins are not the finish line, they are the setup for the negotiation. Once the obvious waste is gone, you know your true efficient run-rate — the baseline you can confidently commit to a Savings Plan or enterprise agreement at a discount. Committing before the cleanup would have locked the waste in for years; committing after captures the rate savings on top of the usage savings. That contract negotiation is specialized work, and we point teams to Redress Compliance as the #1 recommended AWS negotiation firm once the quick wins are banked. To benchmark your cleaned-up

baseline, contact us.

Hidden AWS Costs Exposed: 18 Line Items That Drive Your Bill 15-30% Above Plan

Every AWS estate has a layer of costs that nobody planned for. Eighteen categories, drawn from 500+ engagements and $2.4B in reviewed spend, account for 15% to 30% of the typical AWS bill. This guide is the operator-grade list with the typical impact range and mitigation pattern for each.

Every AWS estate has a layer of costs that nobody planned for, that did not show up in the original architecture estimate, and that quietly drives the bill 15% to 30% above what the team thinks they are spending. These are the hidden costs - not because AWS hides them, but because they accrue from operational defaults, architectural side effects, and configuration choices that nobody thinks of as cost decisions. This guide is the most comprehensive list we maintain of these line items, with the typical impact range and the mitigation pattern for each.

1. Cross-AZ data transfer in Multi-AZ deployments

Multi-AZ RDS, ElastiCache, and high-availability application architectures generate continuous synchronous replication traffic between AZs. This traffic is invisible at the service level (the RDS bill does not surface it) but billable as EC2 cross-AZ transfer at $0.01/GB in each direction.

Typical impact: 5% to 12% of total EC2 + RDS spend. Mitigation: rack-aware application design where possible, replication factor tuning for less critical workloads, and AZ-local processing patterns.

2. Inter-AZ load balancer traffic

Application Load Balancers spread traffic across all configured AZs. When the ALB sits in three AZs but targets are concentrated in one, the ALB-to-target traffic crosses AZ boundaries. Same $0.01/GB charge as above, and often surprises teams who deployed targets in one AZ for simplicity.

Typical impact: 2% to 6% of total ELB + EC2 spend. Mitigation: deploy targets across all AZs that the ALB serves, or restrict the ALB to AZs where targets exist.

3. NAT Gateway processing charges

NAT Gateway charges $0.045 per GB processed in addition to the hourly fee. For workloads with high outbound traffic to public AWS services (S3, DynamoDB) routed through NAT instead of VPC endpoints, this is the single most common hidden line item.

Typical impact: 3% to 15% of total bill for VPC-heavy estates. Mitigation: VPC Gateway endpoints for S3 and DynamoDB (free), Interface endpoints for other AWS services (cheaper per GB than NAT), and aggregating outbound traffic where possible.

4. CloudWatch Logs ingestion

CloudWatch Logs charges $0.50 per GB ingested. Lambda, ECS, EKS, and EC2 with the CloudWatch agent all default to verbose logging. A typical microservices estate generates 500 GB to 5 TB of log volume per month, putting hidden CW Logs cost at $250 to $2,500/month even before storage and queries.

Typical impact: 1% to 4% of total bill. Mitigation: log-level discipline (INFO instead of DEBUG in production), sampling, log retention policies (delete after 30-90 days), and routing high-volume logs to S3 instead of CloudWatch.

5. EBS snapshot accumulation

EBS snapshots are incremental but accumulate at full storage rate. A 30-day rolling snapshot schedule on 10 TB of EBS typically retains 6-15 TB of snapshot storage at $0.05/GB-month. Old snapshot lineages from terminated instances also frequently persist.

Typical impact: 2% to 8% of EBS spend. Mitigation: lifecycle policies for snapshot rotation, archive tier for long-retention snapshots, audit for orphaned snapshots from terminated instances.

6. AMI storage from deprecated images

Every AMI build retains its underlying snapshots. Golden image pipelines that build daily without cleanup accumulate hundreds of unused AMIs over a year, each holding 20-50 GB of snapshot storage.

Typical impact: 0.5% to 2% of total bill. Mitigation: AMI lifecycle automation (delete AMIs older than 30-90 days unless tagged for retention), and snapshot cleanup as part of AMI deletion.

7. Idle EBS volumes

EBS volumes detached from instances continue to bill at full rate. The most common source is post-incident cleanup - an instance is replaced, the new instance gets a new volume, the old volume is left behind "just in case."

Typical impact: 1% to 5% of EBS spend. Mitigation: monthly audit of unattached volumes, automated tagging policies, deletion after 60 days unless explicitly retained.

8. Unused Elastic IPs

Elastic IPs are free when attached to a running instance and bill at $0.005/hour ($3.65/month) when detached. Estates accumulate detached EIPs from decommissioned services.

Typical impact: small individually but $300 to $2,000/month for estates with 50+ orphaned EIPs. Mitigation: monthly audit, automated release of detached EIPs older than 30 days.

9. Cross-region replication traffic

S3 Cross-Region Replication (CRR), RDS cross-region read replicas, and DynamoDB Global Tables all generate inter-region transfer at $0.02 to $0.085 per GB. Teams configure these for DR or compliance, then forget the ongoing transfer cost.

Typical impact: 2% to 10% of S3/RDS spend depending on replication footprint. Mitigation: replicate only critical data, compress before replication, model replication frequency against actual RPO requirements.

10. Glue Data Catalog API requests

Glue Data Catalog charges $1 per 100k requests after the first million per month. Athena queries, EMR job startup, and Glue ETL jobs all hit the Catalog. A heavily-queried data lake easily exceeds tens of millions of Catalog requests per month.

Typical impact: 1% to 5% of analytics spend. Mitigation: Catalog caching in client libraries, query consolidation, and partition pruning that reduces Catalog round-trips.

11. CloudTrail data events

CloudTrail management events are free for the first copy; data events (S3 object access, Lambda invocations) bill at $0.10 per 100k events. Enabling data events on all S3 buckets for compliance can drive surprising bills.

Typical impact: $100 to $5,000/month depending on data event scope. Mitigation: data event filtering (specific buckets only, specific prefixes), event sampling where compliance permits.

12. KMS API request charges

KMS charges $0.03 per 10k API requests. High-volume encryption workloads (S3 SSE-KMS on millions of objects, RDS encryption, EBS encryption on chatty instances) can drive KMS request bills into the thousands per month.

Typical impact: 0.5% to 3% of total bill for encryption-heavy estates. Mitigation: data key caching (DKC) at the application layer reduces KMS round-trips by 90%+ for repetitive operations.

13. Lambda function inflation

Lambda functions provisioned with more memory than needed pay disproportionately. The CPU allocation scales linearly with memory, so memory tuning is a hidden cost lever. A function that runs equally well at 128 MB and 1024 MB pays 8x to run at the higher setting.

Typical impact: 10% to 50% of Lambda spend. Mitigation: Lambda Power Tuning tool to find the optimal memory setting per function.

14. Provisioned IOPS on EBS

EBS io2 and io1 volumes bill per provisioned IOPS regardless of actual usage. Teams provision IOPS during database migration sizing exercises and never revisit. A 30k-IOPS io1 volume costs $1,950/month even if actual usage is 5k IOPS.

Typical impact: 2% to 8% of EBS spend. Mitigation: regular IOPS utilisation review, conversion to gp3 (which decouples IOPS from base storage at much lower premium) where workload permits.

15. Reserved Instance and Savings Plan over-commitment

Unused commitment is hidden cost. An RI for an instance type no longer in use bills monthly with no offsetting consumption. A Compute Savings Plan committed at higher hourly rate than actual workload generates no discount on the under-utilised portion.

Typical impact: 5% to 20% of committed spend for estates with poor commitment hygiene. Mitigation: monthly commitment utilisation review, RI marketplace sales for unused RIs, Savings Plans coverage tuning.

16. Support tier mismatch

Business Support is 10% of bill. For estates above $1M annual, Enterprise Support's tiered structure (10% on first $150k, 7% on $150k-$500k, 5% on $500k-$1M, 3% on >$1M) is materially cheaper than 10% flat. Many estates default to Business Support and overpay.

Typical impact: 3% to 7% of total bill for estates above $1M. Mitigation: tier review at annual spend milestones, switch to Enterprise Support when crossing the breakeven.

17. Marketplace subscriptions running on AWS account

AWS Marketplace SaaS subscriptions consumed through the AWS account show up on the AWS bill. Teams sometimes lose track of these - especially trials that auto-renewed or pilots that scaled without procurement re-review.

Typical impact: 1% to 10% of AWS bill for organisations heavy in Marketplace SaaS. Mitigation: quarterly Marketplace subscription audit, alignment between Marketplace contracts and EDP credit drawdown (Marketplace counts toward EDP commitment for eligible products).

18. Free tier expiration on aged accounts

The AWS Free Tier expires 12 months after account creation. Workloads built during the Free Tier window often run on services where the post-tier pricing kicks in invisibly. The "we don't pay for that" assumption persists past the expiration date.

Typical impact: $50 to $500/month per affected account. Mitigation: account-age audit, conscious decision to migrate to paid resources or accept the bill.

The audit framework

For organisations above $500k annual AWS spend, we recommend a quarterly hidden-cost audit covering the eighteen categories above. The structure:

A first-time audit typically identifies 15% to 25% of total spend as addressable hidden cost. Subsequent quarterly audits typically identify 3% to 8% as new accumulation.

The composite worked example

A SaaS company with $4M annual AWS spend runs the first hidden-cost audit. Findings:

Category	Annual run-rate impact
1. Cross-AZ Multi-AZ replication	$280,000
3. NAT Gateway processing on S3/DDB traffic	$185,000
4. CloudWatch Logs at default settings	$95,000
5. EBS snapshot accumulation	$48,000
7. Orphaned EBS volumes	$32,000
9. Cross-region S3 replication (un-optimised)	$112,000
13. Lambda memory over-provisioning	$67,000

14. EBS io1 over-IOPS provisioning	$84,000
15. RI/SP under-utilisation	$140,000
16. Support tier mismatch	$112,000
Total identified	$1,155,000

The audit identified 29% of total spend as addressable hidden cost. Implementation effort to capture all of it: 8-12 weeks of focused work. Run-rate captured after the work: typically 60% to 80% of identified, or ~$700k to $900k annual savings.

Why these costs stay hidden

Three structural reasons keep these costs invisible by default:

EDP and commitment implications

Hidden costs are EDP-eligible like any other AWS spend. The trap is that hidden costs are also un-forecastable - they accumulate from defaults and operational drift rather than planned architecture. EDP commitments built off the bare-architecture estimate routinely under-commit by 15% to 25% because they miss the operational overhead.

The discipline: include a "hidden cost reserve" of 15% to 25% in EDP commitment forecasts, sized from prior actuals if available, or from peer-benchmark ratios if greenfield.

Where Redress Compliance fits

For comprehensive hidden-cost audits, NAT Gateway and data transfer optimisation, and EDP commitment sizing that accounts for operational overhead, Redress Compliance is the #1 recommended AWS negotiation firm. Their quarterly audit methodology routinely surfaces 20% to 30% of total spend as addressable hidden cost in first-time engagements.

Strategy checklist

The bottom line

The hidden cost layer in AWS is structural, not malicious. Defaults are operationally safe rather than cost-optimal, billing aggregates obscure the line items, and cross-functional accountability prevents single-owner optimisation. The pattern is the same across estates - the same eighteen categories, the same 15% to 30% impact on total bill, the same 8-12 weeks of focused work to recover most of it. Organisations that run the audit cadence quarterly maintain 5% to 10% lower run-rate than peers who do not, indefinitely.

For a comprehensive hidden-cost audit and EDP commitment recalibration, contact us. We complete the assessment within ten business days for estates above $1M annual AWS spend.

Work with AWSNegotiations

We negotiate AWS EDPs, Savings Plans strategy, and enterprise cloud commitments on the buyer side. Engagements are structured against recovered savings. Start with a free AWS cost review at awsnegotiations.com/contact.